Osborne
McGraw-Hill

6502

Machinetaal Subroutines

Lance A. Leventhal
Winthrop Saville

McGraw Hill

6502

Machinetaal Subroutines

LANCE LEVENTHAL
WINTHROP SAVILLE

OSBORNE/McGRAW-HILL

McGraw-Hill Book Company Gmbh
Hamburg New York St. Louis San Francisco Auckland Bogotá Guatemala
Milaan Johannesburg Lisbon London Madrid Mexico Montreal New Delhi Panama
Paris San Juan São Paulo Singapore Sydney Tokyo Toronto

OSBORNE/McGraw-Hill

Uitgegeven door: McGraw-Hill Book Co. Gmbh
Lademannbogen 136
D-2000 Hamburg 63
West Duitsland

I.S.B.N. 3 89028 036 6

Oorspronkelijke titel 6502 Assembly Language Subroutines
© 1981 Mc Graw-Hill Inc. New York
© Nederlandse vertaling Mc Graw-Hill Book Co. Gmbh Hamburg

Nederlandse vertaling : Paul Molenaar
Programmatuur: Patrick van Kleef
Productiebegeleiding: Uitgeverij Wolfkamp Amsterdam

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt doormidel van druk, fotokopie, microfilm, geluidsband, electronisch of op welke ander wijze dan ook en evenmin in een retrieval system worden opgeslagen zonder voorafgaande toestemming van de uitgever.

Voor informatie over vertalingen en distributie van boeken kunt u schrijven naar OSBORNE -/McGraw-Hill op het hierboven genoemde adres.

Inhoudsopgave

Voorwoord 7
1 Algemene programmeertechnieken 11
2 Toepassing van aanvullende instructies en adresseringsmodi 79
3 Veel voorkomende programmeerfouten 141

Inleiding tot de programmahoofdstukken 161
4 Code conversie 165
5 Array manipulatie en indexering 195
6 Rekenkunde 231
7 Bit manipulatie en schuiven 309
8 String verwerking 353
9 Reeks verwerking 391
10 Invoer/uitvoer 431
11 Interrupts 479
A Overzicht 6502 Instructieset
B Programmeeroverzicht 6522 Versatile Interface Adapter
C ASCII karakterset 517
Woordenlijst 519
Index

Voorwoord

Dit boek is bedoeld als bron van informatie en als naslagwerk voor de machinetaal-programmeur. Het bevat een overzicht voor het programmeren met de 6502 microprocessor en een verzameling handige routines. Bij het schrijven van de routines hebben we een standaard formaat, uitleg en parameter-doorgeef-mechanisme. We hebben de assembleer-regels van de oorspronkelijke fabrikant van de micro-processor gevolgd en doel, procedure, gegevens, resultaat, uitvoeringstijd en geheugengebruik van elke routine beschreven.

Dit overzicht van machinetaal-programmeren geeft, diegenen die geen tijd hebben, of voor wie het niet noodzakelijk is, om een complete handleiding machinetaal-programmeren te lezen zoals in *Assembly Language Programming* wordt gegeven, een korte samenvatting. Hoofdstuk 1 bevat een introductie in machinetaal-programmeren voor de 6502-processor en een kort overzicht van de voornaamste verschillen tussen deze en andere processoren. Hoofdstuk 2 beschrijft hoe u instructies en adresseringen die niet expliciet aanwezig zijn kunt toepassen. Hoofdstuk 3 behandelt de meest voorkomende fouten die de programmeur tegen kan komen.
De verzameling routines bevat in veel programmatuur toepasbare opdrachten zoals conversie, array-manipulatie, wiskundige berekening, bit-manipulatie, schuif-functies, string-manipulatie, optellen, sorteren en zoeken. We geven ook voorbeelden van I/O routines, interrupt-afhandelingen en initialisatie- routines voor veelgebruikte chips, zoals parallelle interfaces, seriele interfaces en timers. U kunt deze routines gebruiken als subroutines in toepassingsprogramma's en als richtlijnen voor complexere programmatuur.
Met dit boek richten we ons op de lezer die niet eerst machinetaal wil leren, maar deze direct wil gaan gebruiken. De lezer zou kunnen zijn:

- Een ingenieur, technicus of programmeur die machinetaal-programma's voor een project moet schrijven.

- Een microcomputer-gebruiker die een I/O-aansturingsroutine, een testprogramma, of een gebruiks- of systeemprogramma in machinetaal wil schrijven.

- Een programmeur of ingenieur met ervaring in machinetaal die een kort overzicht van de programmeertechnieken voor een specifieke processor nodig heeft.

- Een systeemontwerper of systeemprogrammeur die een specifieke routine of techniek snel nodig heeft.

- Een programmeur die in hogere programmeertalen schrijft maar een programma moet optimaliseren of foutvrij maken of een programma, geschreven in een hogere programmeertaal, aan een machinetaalroutine moet koppelen.

- Een systeemontwerper of systeembeheerder die snel de werking van een specifiek machinetaalprogramma moet begrijpen.

- Een gebruiker van een microcomputer die wil begrijpen hoe het operating-systeem van een specifieke computer werkt of die volledige toegang tot de mogelijkheden van de computer wil krijgen.

- Een student, hobbyist of docent die voorbeelden van functionerende machinetaalprogramma's wil zien.

Dit boek kan ook dienen als een aanvulling op *Assembly Language Programming* en kan u tijd en moeite besparen. Het heeft geen zin om standaard routines te schrijven, foutvrij te maken, te testen of te optimaliseren. Evenmin hoeft u door materiaal waarmee u al bekend bent, heen te worstelen. U moet specifieke informatie, een routine of techniek, met een minimum aan moeite kunnen krijgen. Wij hebben dit boek georganiseerd en geindexeerd voor een snel gebruik.
Uiteraard roept een boek met een dergelijke doelstelling om reacties van de lezers. We hebben alle programma's gedegen getest en nauwkeurig van commentaar voorzien. Mocht u nog fouten aantreffen, neem dan contact op met de uitgever. Als u ideen heeft voor aanvullende onderwerpen, routines, programmeertips, index-toevoegingen, enzovoorts, laat het ons dan weten. We hebben uit onze programmeer-ervaring geput om dit boek te schrijven, maar we hebben uw hulp nodig om het te verbeteren. Uw opmerkingen, kritiek en suggesties waarderen wij zeer.

Mededeling van de vertaler

De vertaling van een Engelstalig computerboek is geen sinecure. Veel van de -ook in Nederland-gangbare programmeertermen zijn Engels. Welke termen vertaal je wel? Welke laat je ongewijzigd? Ik heb er voor gekozen om een aantal termen onvertaald te laten en bij twijfel tussen haakjes de Nederlandse uitleg van het woord te geven. Ik heb zelf kunnen constateren hoe onbegrijpelijk een Nederlands woord (stapel) voor een relatief eenvoudige Engelse computerterm (stack) kan zijn. Duidelijkheid prefeleert boven een dwangmatige vertaalzin.

TERMINOLOGIE

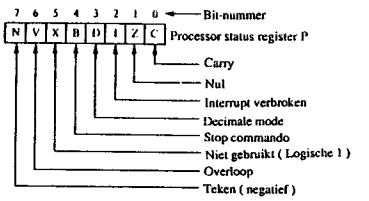
In dit boek hebben wij de volgende terminologie gebruikt om de opbouw van de 6502 processor te beschrijven, de instructies te speciferen en algemene waarden van nummers en adressen voor te stellen.

Opbouw van de 6502

Registers ter grootte van een byte

A (accumulator)
F (vlaggen, zelfde als P)
P (status-register)
S of SP (stack-wijzer)
X (index-register X)
Y (index-register Y)

De registers A, X, en Y zijn de algemeen bruikbare registers. De stack-wijzer bevat altijd het adres van de volgende stack-positie op pagina 1 van het geheugen (adressen 0100_{16} tot en met $01FF_{16}$).
Het P (status) of F (flag) register bestaat uit een serie van 8 bits met onafhankelijke functies en betekenissen. Het volgende diagram toont hoe deze is georganiseerd:



Registers ter grootte van een woord (2 bytes)

PC (programmateller)

Let op: twee opeenvolgende geheugenplaatsen op pagina 0 kunnen ook worden gebruikt als register ter grootte van een woord wanneer zij indirecte adressen bevatten. Het lage adres bevat de laagstwaardige byte (de byte die het adres binnen een geheugenpagina aangeeft), het hoge adres de hoogstwaardige byte (de byte die het nummer van de geheugenpagina aangeeft). Omdat de 6502 automatisch van FF_{16} weer naar 00_{16} springt, is het geheugenpaar $00FF_{16}$ en 0000_{16} een zeer speciaal register.

Tot de vlaggen behoren

Break (B)
Carry (overdracht) (C)
Decimale Modus (D)
Interrupt onmogelijk (I)
Negatief (N)
Overschot (V)
Nul (Z)

Deze vlaggen zijn in het P of F register opgenomen zoals hierboven is te zien.

6502 Assembler

Scheidingstekens zijn

spatie	Na een label of instructie
,	Tussen de operanden in het operand (adres) veld
;	Voor commentaar
:	Na een label (optioneel)
(,)	Rondom een indirect adres

Pseudo instructies zijn

.BLOCK	Reserveer geheugenbytes; reserveer het gespecificeerde aantal geheugenbytes voor tijdelijke opslag
.BYTE	Vorm data ter grootte van een byte; plaats de gespecificeerde 8-bits data in de eerstvolgende geheugenplaats
.DBYTE	Vorm data ter grootte van twee bytes (een woord) met de hoogst- waardige byte als eerste; plaats de gespecificeerde 16-bits data in de eerstvolgende geheugenplaatsen, waarbij de hoogstwaardige byte op de eerste plaats wordt gezet
.END	Einde programma
.EQU	'Is gelijk aan'; definieer de bijgevoegde label
.TEXT	Vorm een rij ASCII karakters; plaats de gespecificeerde ASCII karakters in de eerstvolgende geheugenplaatsen
.WORD	Vorm data ter grootte van twee bytes (een woord) met de laagst waardige byte als eerste; plaats de gespecificeerde 16-bits data in de eerstvolgende geheugenplaats, waarbij de laagst waardige byte op de eerste plaats wordt gezet
*	Bepaal de geheugenplaats waarop de geassembleerde code gezet moet worden
=	'Is gelijk aan'; definieer de bijgevoegde label

Notaties zijn

Numerieke notering:

$ (prefix) of H (suffix)	Hexadecimaal
@ (prefix) of Q (suffix)	Octaal
% (prefix) of B (suffix)	Binair

Is de notering niet gespecificeerd, dan is deze decimaal.

Andere noteringen:

' (voor karakter)	ASCII
*	huidige waarde van programma teller
' ' of " "	(aan weerszijden van karakters) ASCII string
#	Directe adressering
,X	Geindexeerde adressering met het X-register
,Y	Geindexeerde adressering met het Y-register

Is de adressering niet gespecificeerd, dan is deze direct (absoluut)

Algemene terminologie

ADDR	een 16-bit adres in het geheugen
ADDRH	de hoogstwaardige byte van ADR
ADDRL	de laagstwaardige byte van ADR
BASE	een vast 16-bit adres
BASEH	de hoogstwaardige byte van BASIS
BASEL	de laagstwaardige byte van BASIS
DEST	een 16-bit adres in een programma, de bestemming van een sprong of tak instructie
NTIMES	8-bits data
NTIMH	8-bits data
NTIMHC	8-bits data
NTIML	8-bits data
NTIMLC	8-bits data
OPER	een 16-bits adres in het geheugen
OPER1	een 16-bits adres in het geheugen
OPER2	een 16-bits adres in het geheugen
PGZRO	een adres op pagina 0 van het geheugen
PGZRO+1	het adres 1 groter dan PGZRO
POINTER	een 16-bits adres in het geheugen
POINTH	de hoogstwaardige byte van POINTER
POINTL	de laagstwaardige byte van POINTER
RESLT	een 16-bits adres in het geheugen
VAL16	16-bits data
VAL16L	de laagstwaardige byte van VAL16
VAL16M	de hoogstwaardige byte van VAL16
VALUE	8-bits data
ZCOUNT	een 16-bits adres in het geheugen

Hoofdstuk 1 **Algemene programmeertechnieken**

Dit hoofdstuk beschrijft algemene technieken voor het schrijven van machinetaal programma's voor de 6502 en aanverwante processoren. Het toont technieken voor de volgende handelingen:

- Registers laden en bewaren
- Data in het geheugen opbergen
- Rekenkundige en logische functies
- Bit-manipulatie
- Bit-test
- Test op specifieke waarden
- Numerieke vergelijkingen
- Lustechnieken (het herhalen van een serie handelingen)
- Array-verwerking en -manipulatie
- Tabeltechnieken
- Karakter-codemanipulatie
- Code-conversie
- Meervoudige-precisie-rekenkunde
- Delen en vermenigvuldigen
- Lijstbewerking
- Datastructuur-verwerking

Aparte paragrafen behandelen het doorgeven van parameters aan subroutines, het schrijven van I/O-drivers en interruptafhandelingsroutines, en technieken om programma's sneller te laten lopen of efficienter gebruik te laten maken van het geheugen.
De beschreven handelingen zijn nodig bij industriele bewaking, proces bewaking, luchtvaart en militaire systemen, zakelijke en particuliere toepassingen en bij het gebruik van bij voorbeeld testapparatuur, computerapparatuur en communicatieapparatuur. Gebruikers van microcomputers kunnen deze bewerkingen gebruiken bij het schrijven van I/O-drivers, diagnostische- en systeemprogrammatuur en utility-programma's. Bovendien helpt dit boek bij het begrijpen, foutvrij maken en verbeteren van programma's geschreven in hogere programmeertalen. Dit hoofdstuk is een korte gids voor het programmeren in 6502 machinetaal, bedoeld voor degenen die meteen aan de slag willen.

KORT OVERZICHT VOOR ERVAREN PROGRAMMEURS

Degenen die bekend zijn met het programmeren in machinetaal voor andere processoren geven wij hier een kort overzicht van de eigenschappen van de 6502 processor. De programmeur kan zichzelf veel tijd besparen als hij/zij van deze eigenschappen op de hoogte is.

1. De overdrachtvlag (C) gedraagt zich als een omgekeerd 'lenen' bij aftrekken. Een 'Aftrek' (SBC) of 'Vergelijk' (CMP, CPX of CPY) instructie zet de overdrachtvlag uit als de handeling een 'bewaar' nodig heeft en zet de overdrachtvlag als dat niet het geval is. De SBC instructie zorgt voor deze inversie omdat zij 1 - overdracht van het gebruikelijke verschil aftrekt. Zodoende heeft de overdrachtvlag na aftrekking of vergelijking de tegenovergestelde waarde bij de 6502 dan bij de meeste andere computers.

2. De enige optel en aftrek instructies zijn ADC (Tel op met overdrachtvlag) en SBC (Trek af met overdrachtvlag). Als de overdrachtvlag niet gebruikt mag worden, moet deze voor een optelling uitgezet en voor een aftrekinstructie aangezet worden. De functie ziet er voor het optellen dan zo uit:

```
CLC
ADC     geheugen
```

en als volgt voor het aftrekken:

```
SEC
SBC     geheugen
```

3. Er zijn geen 16-bits registers en geen functies die op 16-bits hoeveelheden werken. Het ontbreken van 16-bits registers wordt over het algemeen opgevangen door het gebruik van wijzers op pagina 0 en de indirecte geindexeerde adresseringsmodus. Echter, voor zowel het initialiseren als het wijzigen van die wijzers zijn 8-bits handelingen nodig.

4. Er bestaat, behalve bij JMP, geen echte indirecte adressering. Voor vele andere instructies kan indirecte adressering gesimuleerd worden het indexregister Y te wissen en gebruik te maken van indirecte geindexeerde adressering, of door het indexregister X te wissen en gebruik te maken van geindexeerde indirecte adressering. Beide methoden zijn beperkt tot indirecte adressen op pagina 0.

5. De stack staat altijd op geheugenpagina 1. De stack-wijzer bevat de laagstwaardige byte van het volgende beschikbare adres. Zodoende is de stack beperkt tot 256 geheugenbytes.

6. De JSR (spring naar subroutine) instructie bewaart het adres van haar eigen derde byte op de stack. Ofte wel, JSR bewaart terugkeeradres min 1 op stack. RTS (keer terug van subroutine) laadt de programmateller van de stack-top en telt er dan 1 bij op. Onthoud dit verschil van 1 byte als u JSR of RTS anders dan normaal gebruikt.

7. De Decimale Modus (D) vlag wordt gebruikt voor het verrichten van decimale rekenkundige bewerkingen. Als deze vlag gezet is, geven alle optellingen en aftrekkingen een decimaal resultaat. Ophogingen en verlagingen geven evenwel, ongeacht de stand van dit bit, een binair resultaat. Het probleem bij het gebruik van de Decimale Modus is dat de oorspronkelijke of huidige stand van de D-vlag niet zeker is (de processor initialiseert haar niet bij 'reset'). Er is echter een eenvoudige methode om problemen hiermee te omzeilen in programma's die ADC- of SBC-instructies gebruiken: bewaar de oorspronkelijke D-vlag op de stack, geef D de juiste waarde en herstel de oorspronkelijke waarde voor terugkeer. Interrupt-afhandelingsroutines in het bijzonder moeten altijd D aan- of uitzetten voordat optel- of aftrekinstructies worden uitgevoerd. De PHP (bewaar statusregister op stack) en PLP (haal statusregister van stack) instructies kunnen worden gebruikt om de D-vlag op te bergen en terug te halen. De hoofd-opstartroutine moet D initialiseren (normaal gesproken op 0, hetgeen de binaire modus aangeeft, met CLD). De meeste besturingssystemen die op een 6502 zijn gebaseerd, gaan van de binaire modus als standaard uit en keren altijd zo snel mogelijk naar die modus terug.
Een eigenaardigheidje van de decimale modus van de 6502 is het feit dat de nul- en negatiefvlaggen niet langer algemeen geldend zijn. Deze vlaggen geven alleen het binaire resultaat aan, niet het decimale. Slechts de overdrachtvlag geeft altijd de decimale uitkomst weer. Zo gebeurt het dat als men 80 hex van 50 hex aftrekt in de decimale modus, de nulvlag wordt gezet (omdat het binaire resultaat D0 hex is), ook al heeft de decimale uitkomst (70 hex) een hoogstwaardig bit van 0. Zo heeft het optellen van 50 hex en 50 hex in de decimale modus tot gevolg dat de nulvlag wordt

uitgezet (omdat het binaire resultaat A0 hex is), terwijl de decimale uitkomst 0 is. Het optellen van 50 hex en 50 hex zet wel de overdrachtvlag (C). Daarom moet de programmeur, indien hij in de decimale modus werkt, slechts vertakkingen gebruiken die op de overdrachtvlag zijn gebaseerd, of bewerkingen gebruiken die niet van de modus afhankelijk zijn (zoals aftrekken en vergelijken gevolgd door vertakkingen gebaseerd op de nulvlag).

8. Normale laad (of haal van stack) en verplaatsinstructies (met uitzondering van TXS) beinvloeden de negatief- en nulvlaggen. Dit is niet het geval bij de 8080, 8085 en Z-80 microprocessoren. Opslag van data in het geheugen beinvloedt geen enkele vlag.

9. INC en DEC kunnen niet op de accumulator worden toegepast. Om A op te hogen, gebruikt u:

```
CLC
ADC     #1          ;VERHOOG ACCUMULATOR MET 1
```

Gebruik:

```
SEC
SBC     #1          ;VERLAAG ACCUMULATOR MET 1
```

om A te verlagen.

10. De indexregisters zijn slechts 8 bits lang. Dit creeert voor de hand liggende problemen in arrays of geheugengebieden die groter zijn dan 256 bytes. Om dit te verhelpen kan de indirect geindexeerde (postindexering) adresserings-modus gebruikt worden. Deze modus stelt de programmeur in staat het beginadres van een array in twee geheugenplaatsen op pagina 0 te bewaren. Telkens wanneer het programma voorbij een blok van 256 bytes is, moet het 1 bij het hoogstwaardige bit van het indirecte adres optellen, voordat het naar het volgende blok gaat. De processor weet dat hij een blok heeft afgewerkt als indexregister Y weer 0 wordt. Een dergelijke routine kan er zo uitzien:

```
INY                 ;GA NAAR DE VOLGENDE BYTE
BNE     LUS         ;ZOLANG ER NOG GEEN BLOK (=PAGINA) IS
                    ;AFGEWERKT
INC     INDR+1      ;INDIEN WEL, GA NAAR VOLGENDE PAGINA
```

Geheugenlocatie INDR+1 (op pagina 0) bevat de hoogstwaardige byte van het indirecte adres.

11. 16-bits tellers kunnen in twee geheugenlocaties worden bewaard. Optellen is veel makkelijker dan aftellen, omdat deze routine gebruikt kan worden:

```
INC     TELLERL     ;VERHOOG LAAGSTWAARDIGE BYTE
BNE     LUS
INC     TELLERH     ;EN VERHOOG HOOGSTWAARDIGE BYTE INDIEN
                    ;NODIG
JMP     LUS
```

TELLERL bevat de laagstwaardige byte van een 16-bits teller en TELLERH de hoogstwaardige byte. Merk op dat we de nulvlag controleren in plaats van de overdrachtvlag, zoals op de meeste computers. Verhogen en verlagen heeft geen invloed op de overdrachtvlag (C).

12. De BIT-instructie (logische AND zonder dat het resultaat wordt bewaard) heeft verscheidene ongebruikelijke eigenschappen. Op de eerste plaats staat het alleen directe adressering (absoluut en pagina 0) toe. Als bit 3 van geheugenlocatie ADR getest moet worden, kan deze routine worden gebruikt:

```
LDA     #%00001000
BIT     ADR
```

BIT laadt ook de negatief- en overschotvlaggen met de inhoud van respectievelijk bits 7 en 6 van de geheugenlocatie,ongeacht de waarde van de accumulator. Zodoende kunnen de volgende handelingen uitgevoerd worden zonder de accumulator te laden. Vertak naar BEST indien bit 7 van ADR 1 is

```
BIT     ADR
BMI     BEST
```

Vertak naar BEST indien bit 6 van ADR 0 is

```
BIT     ADR
BVC     DEST
```

Uiteraard dient de programmeur het bijzondere gebruik van de overschotvlag goed te documenteren. Dit is nodig om het programma ook achteraf of voor anderen begrijpelijk te houden.

13. De processor mist een aantal algemene instructies die wel beschikbaar zijn op de 6800, 6809 en soortgelijke processoren. Het merendeel van de ontbrekende instructies is makkelijk te simuleren, hoewel de documentatie ervan lastig kan zijn. In het bijzonder noemen wij de Clear (gebruik laad-direct met 0 hiervoor in de plaats), Complement (gebruik de logische EXCLUSIEVE OR met de een byte met alleen 1-en hiervoor in de plaats), en de eerder genoemde Optel (zonder overdracht) en Aftrek (zonder onthouden). Het is ook niet mogelijk om de stack wijzer direct te laden of op te bergen (dit kan wel via het X-register), het statusregister te laden of op te bergen (dit kan wel via de stack) of bewerkingen tussen registers uit te voeren (een moet er in het geheugen staan). Andere ontbrekende instructies zijn Vertak Altijd (gebruik hiervoor in de plaats JMP of ken een vlag een waarde toe en vertak op die waarde), Verhoog of Verlaag Accumulator (gebruik de optel en aftrekinstructies), Aritmetische Schuiving (kopieer bit 7 naar overdrachtvlag en roteer) en Test nul of min (gebruik een vergelijking met 0 of een verhoog/verlaag routine). Weller beschrijft het definieren van macro's om de ontbrekende instructies te vervangen.

14. De 6502 gebruikt de volgende algemene conventies:

. 16-bits adressen worden met de laagstwaardige byte eerst opgeslagen. De volgorde van de bytes is dezelfde als bij de 8080, Z-80 en 8085 processoren, maar tegenovergesteld aan de volgorde gebruikt door de 6800 en 6809.

. De stack-wijzer bevat het adres (op pagina 1) van de volgende beschikbare locatie. Deze conventie wordt ook in de 6800 gebruikt, maar het voor de hand liggende alternatief (de laatst bezette locatie) wordt gebruikt in de 8080, 8085, Z-80 en 6809 microprocessoren. Instructies bergen data op de stack waarbij zij achteraf verlagen (zij trekken 1 af van de stack-wijzer na het opslaan van elke byte) en laden data van de stack waarbij zij vooraf verhogen (zij tellen 1 op bij de stack-wijzer voordat elke byte wordt geladen).

. De I (Interrupt of onderbreking) vlag werkt als een aan/uit-schakelaar. Het zetten van de vlag (met SEI) maakt de maskeerbare interrupt onmogelijk en het uitzetten van de vlag (met CLI) staat

de maskeerbare interrupt toe. Deze conventie is dezelfde als in de 6800 en 6809 maar tegengesteld aan die van de 8080, 8085 en Z-80.

DE 6502 REGISTERS

De beperkte registerset van de 6502 maakt het programmeren voor deze processor lastig. Er zijn bij voorbeeld geen 16-bits registers. Daarom moeten adressen van variabelen in geheugenparen op pagina 0 gezet worden. Toegang tot deze variabelen is dan te krijgen via preindexering (geindexeerde indirecte adressering) of postindexering (indirecte geindexeerde adressering). Het ontbreken van 16-bits registers bemoeilijkt ook het hanteren van arrays of blokken die meer dan 256 bytes geheugenruimte in beslag nemen.

Als we geheugenlocaties op pagina 0 als uitbreidingen van de registerset beschouwen, dan kunnen we de registers als volgt karakteriseren:

. De accumulator is het hart van databewerking en wordt gebruikt als bron en bestemming door de meeste rekenkundige, logische en andere databewerkings-instructies.

. Indexregister X is de hoofd-index voor niet-indirecte applicaties. het is het enige register dat een indexeermodus op pagina 0 heeft (met uitzondering van de LDX- en STX-instructies) en dat gebruikt kan worden voor indexering bij enkelvoudige instructies als Schuif, Verhoog en Verlaag. Het is ook het enige register dat voor preindexering gebruikt kan worden, hoewel die modus niet vaak wordt toegepast en de mogelijkheid heeft zijn waarde van de stack-wijzer te halen of die er op te zetten.

. Indexregister Y is de hoofd-index voor indirecte toepassingen, omdat deze als enige bij post-indexering gebruikt kan worden.

. Geheugenlocaties op pagina 0 zijn de enige locaties die via pagina 0 (direct), pagina-0-indexering, preindexering en postindexering kunnen worden benaderd.

De tabellen 1-1 tot 1-7 bevatten lijsten van instructies en hun specifieke eigenschappen. Tabel 1-1 toont instructies die op specifieke registers betrekking hebben en tabel 1-2 toont instructies die direct op geheugenlocaties kunnen worden toegepast. Tabel 1-3 tot 1-7 bevatten de instructies die aparte adresseringsmodi toestaan; pagina 0 (tabel 1-3), absoluut (tabel 1-4), pagina 0 geindexeerd (tabel 1-5), absoluut geindexeerd (tabel 1-6) en pre- en postindexering (tabel 1-7).

We kunnen de speciale eigenschappen van de specifieke registers als volgt omschrijven:

. **Accumulator**. Bron en bestemming van alle rekenkundige en logische instructies, behalve CPX, CPY, DEC en INC. Het enige register dat geschoven kan worden met een enkele instructie. Het enige register dat geladen en opgeborgen kan worden met gebruik van preindexering of postindexering.

. **Indexregister X**. Kan verhoogd worden met INX en verlaagd met DEX. Het enige register dat kan worden gebruikt als de index bij preindexering. Het enige register dat gebruikt kan worden om de stack-wijzer te laden of op te bergen.

. **Indexregister Y**. Kan verhoogd worden met INY en verlaagd met DEY. Het enige register dat kan worden gebruikt als de index bij postindexering.

. **Geheugenlocaties op pagina 0**. De enige geheugenlocaties die indirecte adressen kunnen bevatten voor post- of preindexering. De enige geheugenlocaties die benaderd kunnen worden met gebruik van de 0-pagina of indexering via de 0-pagina.

. **Status register**. Kan alleen op de stack worden opgeborgen met behulp van PHP of van de stack worden geladen met behulp van PLP.

Tabel 1-1: Registers en bruikbare instructies

Register	Instructies
A	ADC, AND, ASL, BIT, CMP, EOR, LDA, LSR, ORA, PHA, PLA, ROL, ROR, SBC, STA, TAX, TAY, TXA, TYA
P (processor status)	PHP, PLP, (CLC, CLD, CLV, SEC, en SED veranderen specifieke vlaggen)
S (stack-wijzer)	JSR, PHA, PHP, PLA, PLP, RTS, TSX, TXS
X	CPX, DEX, INX, LDX, STX, TAX, TSX, TXA, TXS
Y	CPY, DEY, INY, LDY, STY, TAY, TYA

Tabel 1-2: Instructies die direct op geheugenlocaties toegepast kunnen worden

Instructie	Functie
ASL	Rekenkundige verschuiving naar links
BIT	Bit test (test bit 6 en 7)
DEC	Verlaag met 1
INC	Verhoog met 1
LSR	Logische verschuiving naar rechts
ROL	Roteer naar links
ROR	Roteer naar rechts

Tabel 1-3: Instructies die met pagina-nul adressering werken

Instructie	Functies
ADC	Optellen met overdracht
AND	Logische AND-instructie
ASL	Rekenkundige verschuiving naar links
BIT	Bit test
CMP	Vergelijk geheugenlocatie met inhoud accumulator
CPX	Vergelijk geheugenlocatie met inhoud indexregister X
CPY	Vergelijk geheugenlocatie met inhoud indexregister Y
DEC	Verlaag met 1
EOR	Logische EXCLUSIVE OR-instructie
INC	Verhoog met 1
LDA	Laad de accumulator
LDX	Laad het indexregister X
LDY	Laad het indexregister Y
LSR	Logische verschuiving naar rechts
ORA	Logische OR-instructie
ROL	Roteer naar links
ROR	Roteer naar rechts
SBC	Verlaag met overdracht
STA	Bewaar de inhoud van de accumulator
STX	Bewaar de inhoud van indexregister X
STY	Bewaar de inhoud van indexregister Y

Tabel 1-4: Instructies die absolute (directe) adressering toestaan

Instructie	Functie
ADC	Optellen met overdracht
AND	Logische AND-instructie
ASL	Rekenkundige verschuiving naar links
BIT	Logische bit test
CMP	Vergelijk geheugenlocatie met inhoud accumulator
CPX	Vergelijk geheugenlocatie met inhoud indexregister X
CPY	Vergelijk geheugenlocatie met inhoud indexregister Y
DEC	Verlaag met 1
EOR	Logische EXCLUSIVE OR-instructie
INC	Verhoog met 1
JMP	Spring altijd
JSR	Spring naar een subroutine
LDA	Laad de accumulator
LDX	Laad het indexregister X
LDY	Laad het indexregister Y
LSR	Logische verschuiving naar rechts
ORA	Logische OR-instructie
ROL	Roteer naar links
ROR	Roteer naar rechts
SBC	Aftrekken met overdracht
STA	Bewaar de inhoud van de accumulator
STX	Bewaar de inhoud van het indexregister X
STY	Bewaar de inhoud van het indexregister Y

Tabel 1-5: Instructies die geindexeerde pagina-nul adressering toelaten

	Instructie	Functie
Indexregister Y	ADC	Optellen met overdracht
	AND	Logische AND-instructie
	ASL	Rekenkundige verschuiving naar links
	CMP	Vergelijk geheugenlocatie met inhoud accumulator
	DEC	Verlaag met 1
	EOR	Logische EXCLUSIVE OR-instructie
	INC	Verhoog met 1
	LDA	Laad de accumulator
	LDY	Laad het indexregister Y
	LSR	Logische verschuiving naar rechts
	ORA	Logische OR-instructies
	ROL	Roteer naar links
	ROR	Roteer naar rechts
	SBC	Aftrekken met overdracht
	STA	Bewaar de inhoud van de accumulator
	STY	Bewaar de inhoud van het indexregister Y
Indexregister X	LDX	Laad het indexregister X
	STX	Bewaar de inhoud van het indexregister X

Tabel 1-6: Instructies die absoluut geindexeerde adressering toestaan

	Instructie	Functie
Indexregister Y	ADC	Optellen met overdracht
	AND	Logische AND-instructie
	ASL	Rekenkundige verschuiving naar links
	CMP	Vergelijk geheugenlocatie met inhoud accumulator
	DEC	Verlaag met 1
	EOR	Logische EXCLUSIVE OR-instructie
	INC	Verhoog met 1
	LDA	Laad de accumulator
	LDY	Laad het indexregister Y
	LSR	Logische verschuiving naar rechts
	ORA	Logische OR-instructie
	ROL	Roteer naar links
	ROR	Roteer naar rechts
	SBC	Aftrekken met overdracht
	STA	Bewaar de inhoud van de accumulator
Indexregister X	ADC	Optellen met overdracht
	AND	Logische AND-instructie
	CMP	Vergelijk geheugenlocatie met inhoud accumulator
	EOR	Logische EXCLUSIVE OR-instructie
	LDA	Laad de accumulator
	LDX	Laad het indexregoister X
	ORA	Logische OR-instructie
	SBC	Aftrekken met overdracht
	STA	Bewaar de inhoud van de accumulator

Tabel 1-7: Instructies die pre- en postindexering toelaten

Instructie	Functie
ADC	Optellen met overdracht
AND	Logische OR-instructie
CMP	Vergelijk geheugenlocatie met inhoud accumulator
EOR	Logische EXCLUSIVE OR-instructie
LDA	Laad de accumulator
ORA	Logische OR-instructie
SBC	Aftrekken met overdracht
STA	Bewaar de inhoud van de accumulator

. **Stack-wijzer**. Refereert altijd naar een adres op pagina 1. Kan alleen vanuit of in het indexregister X worden geladen met respectievelijk TXS en TSX.

Let op het volgende:

. Vrijwel alle databewerkingen maken gebruik van de accumulator, omdat deze over enkelvoudige rekenkundige en logische instructies beschikt. Bovendien wordt het resultaat van die bewerkingen ook in de accumulator weergegeven.

. Slechts een beperkt aantal instructies maakt direct gebruik van de registers en/of geheugenlocaties. Een indexregister kan met 1 verhoogd worden, met 1 verlaagd worden of vergeleken worden met een constante of een absoluut adres. De waarde van een geheugenlocatie kan met 1 verhoogd of verlaagd worden, naar links of rechts geschoven worden en links- of rechtsom geroteerd worden.

. De beschikbare adresseringsmethoden verschillen aanzienlijk van instructie tot instructie. Let vooral op de beperkte mogelijkheden bij de instructies BIT, CPX, CPY, LDX, LDY, STX en STY.

Register verplaatsingen

Er is slechts een beperkt aantal directe verplaatsingen tussen registers. Een enkelvoudige instructie kan een waarde van een indexregister naar de accumulator verplaatsen, van de accumulator naar een indexregister, van de stack-wijzer naar indexregister X of van indexregister X naar de stack-wijzer. De verplaatsinstructies hebben de notatie TBA, waarbij 'B' het bronregister en 'A' het adresregister is, waar de waarde naartoe moet. Het statusregister (P) kan alleen van en naar de stack verplaatst worden door middel van PHP of PLP.

REGISTERS VANUIT HET GEHEUGEN LADEN

De 6502 microprocessor biedt vele mogelijkheden om registers vanuit het geheugen te laden. De volgende adresseringsmodi zijn beschikbaar: pagina 0 (direct), absolute (direct), directe indexering vanuit pagina 0, absolute indexering, post- en preindexering.

Het direct laden van registers

De accumulator, indexregister X en indexregister Y kunnen vanuit het geheugen geladen worden door middel van directe adressering. Een speciale 'pagina 0 mode' laadt registers sneller vanuit adressen op pagina 0 dan van adressen op andere pagina's. De terminologie noemt directe adressering op pagina 0 'pagina 0 adressering' en de meer algemene directe adressering 'absolute adressering'.

Voorbeelden

1. LDA $40

Deze instructie laadt de accumulator vanuit geheugenlocatie 40_{16}. De speciale pagina 0 adressering heeft minder tijd en geheugen nodig dan de meer algemene absolute (directe) adressering.

2. LDX $C000
Deze instructie laadt indexregister X vanuit geheugenlocatie $C000_{16}$. Het maakt gebruik van absolute (directe) adressering.

Het onmiddellijk laden van registers

Deze methode kan gebruikt worden om de accumulator, indexregister X of indexregister Y met een specifieke waarde te laden.

Voorbeelden

1. LDY #6
Deze instructie laadt het indexregister Y met het getal 6. De 6 is een 8-bits waarde, geen 16-bits adres; verwar de waarde 6 niet met het adres 6_{16}.

2. LDA #$E3
Deze instructie laadt de accumulator met het getal $E3_{16}$.

Het geindexeerd laden van registers

De instructies LDA, LDX en LDY kunnen in de geindexeerde modus gebruikt worden. De beperkingen zijn dat het indexregister X niet geladen kan worden met X als index. Hetzelfde geldt voor indexregister Y; dit register kan niet geladen worden met Y als index. Net als met directe adressering is er een speciale pagina 0 modus. Let echter op dat de accumulator niet in de pagina 0 modus met Y als index geladen kan worden.

Voorbeelden

1. LDA $0340,X

Deze instructie laadt de accumulator vanuit het adres dat wordt verkregen wanneer u basisadres 0340_{16} indexeert met het indexregister X; het effectieve adres is dus 0340_{16} + (X). Deze manier van indexeren wordt beschreven in *An Introduction to Microcomputers: Volume 1 - Basic Consepts.* [4]

2. LDX $40,Y

Deze instructie laadt het indexregister X vanuit het adres dat wordt verkregen wanneer u basisadres 0040_{16} indexeert met het indexregister Y. De speciale pagina 0 geindexeerde modus bespaart tijd en geheugenruimte.

Het laden van registers met postindexering.

De instructie LDA kan gebruikt worden in de postindexeringmodus. Hierbij wordt het basisadres vanuit twee geheugenlocaties op pagina 0 verkregen. Voor het overige werkt deze modus hetzelfde als normale indexering.

Voorbeeld

LDA ($40),Y

Deze instructie laadt de accumulator vanuit het adres dat wordt verkregen als u indexeert met indexregister Y vanuit het basisadres in de geheugenlocaties 40_{16} en 41_{16}. Deze modus is beperkt tot pagina 0 en het index-register Y. Hierbij gaat de processor er ook vanuit dat het indirecte adres is opgeborgen met de laagstwaardige byte eerst (op het laagste adres) op de voor de 6502 gebruikelijke manier.

Het laden van registers met preindexering

De instructie LDA kan gebruikt worden in de preindexeringmodus, waarbij het geindexeerde adres zelf indirect wordt gebruikt. Deze modus is beperkt tot pagina 0 en het indexregister X. Let op dat deze methode ook van een tabel met 2-bytes indirecte adressen gebruik maakt, zodat X slechts even waarden mag bevatten.

Voorbeeld

LDA ($40,X)

Deze instructie laadt de accumulator vanuit het indirecte adres dat wordt verkregen wanneer u het basis adres 40_{16} indexeert met het indexregister X. Het indirecte adres staat in de twee geheugenbytes, beginnende bij 40_{16} + (X). Deze modus is vrij ongewoon; hij wordt bij voorbeeld gebruikt om te kiezen uit een tabel van randapparatuuradressen voor in- en uitvoer.

Het laden van registers vanaf de stack.

De instructie PLA laadt de accumulator vanuit de top van de stack en trekt 1 van de stack-wijzer af.

De instructie PLP werkt op dezelfde manier, met dit verschil dat het status register (P) geladen wordt. Dit is de enige methode om het statusregister met een specifieke waarde te laden. De indexregisters kunnen niet direct vanuit de stack maar wel via de accumulator geladen worden. De vereiste handelingen hiervoor zijn:

```
(voor indexregister X)
PLA             ;TOP VAN STACK NAAR A
TAX             ;EN VERDER NAAR X

(voor indexregister Y)
PLA             ;TOP VAN STACK NAAR A
TAY             ;EN VERDER NAAR Y
```

De stack kent de volgende speciale eigenschappen:

. De stack bevindt zich altijd op geheugenpagina 1. De stack-wijzer bevat alleen de laagstwaardige byte van het volgende beschikbare adres.

. Waarden worden via post-verlaging in de stack opgeslagen. De instructies verlagen de stack-wijzer met 1 NADAT elke byte is opgeslagen. Waarden worden met pre-verhoging van de stack geladen; de instructies verlagen de stack-wijzer met 1 VOORDAT elke byte geladen wordt.

. Zoals typisch is voor microprocessors, is er geen overloop- of onderloop-indicatie.

HET OPSLAAN VAN REGISTERS IN HET GEHEUGEN

We kunnen registers op dezelfde manier in het geheugen opslaan als ze vanuit het geheugen geladen werden. De enige verschillen tussen laden en opslaan zijn:

. Opslaginstructies staan geen onmiddellijke adressering toe. Er is geen manier om een getal direct in het geheugen op te slaan. Dit moet via een register lopen.

. STX en STY staan alleen geindexeerde adressering op pagina 0 toe. Geen van beide kent absolute geindexeerde adressering.

. Zoals te verwachten viel, is de volgorde om indexregisters in de stack op te bergen omgekeerd aan die om ze vanuit de stack te laden. De vereiste handelingen zijn:

```
(voor indexregister X)
TXA             ;VERPLAATS X NAAR A
PHA             ;EN DAN NAAR DE TOP VAN DE STACK

(voor indexregister Y)
TYA             ;VERPLAATS Y NAAR A
PHA             ;EN DAN NAAR DE TOP VAN DE STACK
```

Ander opslaghandelingen werken op precies dezelfde manier als is beschreven bij het laden van registers.

Voorbeelden

1. STA $50
Deze instructie bergt de accumulator in geheugenlocatie 0050_{16} op. De speciale pagina 0 modus is zowel korter als sneller dan de absolute modus, omdat ervan uit wordt gegaan dat de hoogstwaardige adresbyte 0 is.

2. STX $17E8
Deze instructie bergt het indexregister X op in geheugenlocatie $17E8_{16}$. Zij maakt gebruik van de absolute adressering modus met een 16-bits adres.

3. STA $A000,Y
Deze instructie bergt de accumulator op in het effectieve adres dat wordt verkregen door het index register Y op te tellen bij basisadres $A000_{16}$. Het effectieve adres is $A000_{16}$ + (Y).

4. STA ($50),Y
Deze instructie bergt de accumulator op in het effectieve adres dat wordt verkregen door het indexregister Y op te tellen bij het basisadres in de geheugenlocaties 0050_{16} en 0051_{16}. De instructie verkrijgt het basisadres indirect.

5. STA ($43,X)
Deze instructie bergt de accumulator op in het effectieve adres dat wordt verkregen door het indexregister X bij het basisadres 0043_{16} op te tellen. Het indirecte adres bevindt zich in de twee geheugenbytes beginnende bij 0043_{16} + (X).

HET OPSLAAN VAN WAARDEN IN RAM

De normale manier om RAM-locaties te initialiseren is door middel van de accumulator, 1 byte per keer. De programmeur kan hiervoor ook de indexregisters X en Y gebruiken.

Voorbeelden

1. Berg een 8-bits voorwerp (WAARDE) op in adres ADR.

```
LDA     #WAARDE     ;LAAD DE WAARDE
STA     ADR         ;INITIALISEER GEHEUGENLOCATIE ADR
```

We zouden ook LDX en STX of LDY en STY in plaats van LDA en STA kunnen gebruiken. Let op dat de 6502 alle waarden op dezelfde manier verwerkt. Er is geen speciale WIS-instructie om 0- en te genereren.

2. Berg een 16-bits voorwerp (WIJZER) op in de adressen ADR en ADR+1 (hoogstwaardige byte in ADR+1). We gaan ervan uit dat WIJZER bestaat uit WIJZH (hoogstwaardige byte) en WIJZL (laagstwaardige byte).

```
LDA     #WIJZL      ;HAAL LWB
STA     ADR         ;INITIALISEER LOCATIE ADR
LDA     #WIJZH      ;HAAL HWB
STA     ADR+1       ;INITIALISEER LOCATIE ADR+1
```

Deze methode maakt het ons mogelijk indirecte adressen op pagina 0 te initialiseren om later met post- en preindexering te gebruiken.

REKENKUNDIGE EN LOGISCHE BEWERKINGEN

De meeste rekenkundige en logische bewerkingen (optellen, aftrekken, AND, OR en EXCLUSIEVE OR) kunnen met behulp van de accumulator en een 8-bits byte in het geheugen uitgevoerd worden. Het resultaat wordt in de accumulator gestopt. Rekenkundige en logische bewerkingen kunnen onmiddellijke, pagina 0 (direct), absolute (directe), geindexeerde, op pagina 0 geindexeerde, geindexeerde indirecte of indirect geindexeerde adresseringsmodi gebruiken.

Voorbeelden

1. Tel geheugenlocatie 0040_{16} op bij de accumulator met overdracht.

```
ADC     $40
```

Deze instructie telt de inhoud van geheugenlocatie 0040_{16} en de inhoud van de overdrachtvlag (C) op bij de accumulator.

2. Voer een logische OR uit tussen de accumulator en een geindexeerd adres met X als index en $17E0_{16}$ als basis.

```
ORA     $17E0,X
```

Het effectieve adres is $17E0_{16}$ + (X)

3. Voer een logische AND uit tussen de accumulator en de inhoud van geheugenlocatie $B470_{16}$.

```
AND     $B470
```

Let op de volgende speciale eigenschappen van de rekenkundige en logische instructies van de 6502:

. De enige optelinstructie is ADC (Add with Carry, Tel op met overdrachtvlag). Om het Overdracht-bit uit te sluiten, moet u hem expliciet uitzetten met

```
CLC                 ;MAAK DE OVERDRACHT 0
ADC     $40         ;TEL OP ZONDER CARRY
```

. De enige aftrekinstructie is SBC (Subtract with Borrow, Trek af met Onthouden). Deze instructie trekt een geheugenlocatie en de gecomplementeerde overdrachtvlag af van de accumulator. SBC geeft

```
(A) = (A) - (G) - (1 - OVERDRACHT)
```

waarbij G de inhoud van het effectieve adres is. Om de overdrachtvlag uit te schakelen moet je hem expliciet aanzetten door

```
SEC             ;MAAK DE GEINVERTEERDE OVERDRACHT 1
SBC     $40     ;TREK AF ZONDER OVERDRACHT
```

Let op dat u de overdrachtvlag moet uitzetten voor optellen en aanzetten voor aftrekken.

. Vergelijkingsinstructies trekken af zonder de registers te veranderen (met uitzondering van de vlaggen in het statusregister). Hiervoor hebben we niet alleen CMP (vergelijk geheugen met de accumulator), maar ook CPX (vergelijk geheugen met het indexregister X) en CPY (vergelijk het geheugen met het indexregister Y). Let op de verschillen tussen CMP en SBC: CMP betrekt de overdrachtvlag niet bij de aftrekking, verandert de accumulator niet en heeft ook geen invloed op de overloopvlag.

. Er is geen expliciete complementeerinstructie. Dit kan echter ook bereikt worden door een Exclusieve OR op de accumulator uit te voeren met een byte die allemaal 1-en bevat (11111111_2 of FF_{16}). Immers, de Exclusieve OR van twee bits heeft een 1 als resultaat wanneer ze ongelijk zijn en 0 als ze gelijk zijn. Zo geeft een Exclusieve OR met 1 een 0 als het andere bit 1 is en geeft een 1 als het andere bit 0 is. Dit is hetzelfde als een logische complementeer (NOT) instructie.

Zodoende hebben we de instructie

```
EOR     #%11111111   ;COMPLEMENTEER DE ACCUMULATOR
```

. De BIT-instructie voert een logische AND uit maar stopt het resultaat daarvan niet in de accumulator. Zij heeft alleen effect op de vlaggen. Let op dat deze instructie alleen directe adressering (pagina 0 of absoluut) toestaat en geen onmiddellijke of geindexeerde adressering. Complexere handelingen vereisen meerdere instructies. Voorbeelden daarvan zijn:

. Tel geheugenlocaties GEGEV1 en GEGEV2 op en stop het resultaat daarvan in RESLT

```
LDA     GEGEV1      ;HAAL EERSTE GEGEVEN
CLC                 ;ZET OVERDRACHT-BIT UIT
ADC     GEGEV2      ;TEL TWEEDE GEGEVEN ERBIJ OP
STA     RESLT       ;BEWAAR DE SOM
```

Let op dat we het eerste gegeven in de accumulator moeten laden en de overdrachtvlag moeten uitzetten voordat we het tweede gegeven gaan optellen.

. Tel een constante (WAARDE) op bij geheugenlocatie WERK

```
LDA     WERK        ;HAAL HUIDIGE WAARDE OP
CLC                 ;ZET OVERDRACHT UIT
ADC     #WAARDE     ;TEL WAARDE ERBIJ OP
STA     WERK        ;EN STOP DE SOM TERUG
```

Indien WAARDE 1 is, kunnen we het inkorten tot

```
INC     WERK        ;TEL 1 BIJ HUIDIGE WAARDE OP
```

Zo ook als de WAARDE -1 is, kunnen we volstaan met

```
DEC     WERK        ;TREK 1 VAN HUIDIGE WAARDE AF
```

BIT-MANIPULATIE

De programmeur kan bits aanzetten, uitzetten, complementeren of testen door middel van logische handelingen met de juiste maskers. Schuifinstructies kunnen de accumulator of een geheugenlocatie roteren of schuiven. Hoofdstuk 7 bevat aanvullende voorbeelden van bit-manipulatie.
U kunt individuele bits in de accumulator als volgt bewerken:

- Aanzetten door een logische OR met 1-en op de goede posities.
- Uitzetten door een logische AND met 0-en op de goede posities.
- Inverteren (complementeren) door een logische EXCLUSIEVE OR met 1-en op de goede posities.
- Testen door een logische AND met 1-en op de juiste posities.

Voorbeelden

1. Zet bit 6 van de accumulator aan.

```
ORA     #%01000000   ;ZET BIT 6 AAN DOOR EEN OR MET 1
```

2. Zet bit 3 van de accumulator uit

```
AND     #%11110111   ;ZET BIT 3 UIT DOOR EEN AND MET 0
```

3. Inverteer (complementeer) bit 2 van de accumulator

```
EOR     #%00000100   ;INVERTEER BIT 2 DOOR EEN XOR MET 1
```

4. Test bit 5 van de accumulator. Zet de nulvlag uit als bit 5 een logische 1 is en zet haar aan als bit 5 een logische 0 is.

```
AND     #%00100000   ;TEST BIT 5 DOOR EEN AND MET 1
```

U kunt meer dan 1 bit tegelijkertijd veranderen door de maskering te veranderen.

5. Zet bits 4 en 5 van de accumulator aan.

```
ORA     #%00110000   ;ZET BITS 4 EN 5 AAN DOOR EEN OR
                     ;MET 1
```

6. Inverteer (complementeer) bits 0 en 7 van de accumulator.

```
EOR     #%10000001   ;INVERTEER BITS 0 EN 7 DOOR EEN XOR
                     ;MET 1
```

De enige algemene manier om bits in andere registers of het geheugen te manipuleren is door hen naar de accumulator te verplaatsen.

- Zet bit 4 van geheugenlocatie 0040_{16} aan.

```
LDA     $40
ORA     #%00010000   ;ZET BIT 4 AAN DOOR EEN OR MET 1
STA     $40
```

. Zet bit 1 van geheugenlocatie $17E0_{16}$ uit

```
LDA     $17E0
AND     #%11111101   ;ZET BIT 1 UIT DOOR EEN AND MET 0
STA     $17E0
```

Soms kunt u van een kortere methode gebruik maken om bit 0 van een register of geheugenlocatie aan- of uit te zetten. Dit kunt u dan doen met een verhoog (INC, INX of INY) instructie om het aan te zetten of een verlaag (DEC, DEX of DEY) instructie om het uit te zetten (als u weet dat het 1 is). De instructie LSR (ASL) schuift de accumulator of een geheugenlocatie 1 positie naar rechts (links), waarbij het meest linkse bit (bit 0) met 0 wordt gevuld. Figuren 1-1 en 1-2 beschrijven het effect van deze twee instructies. De instructies ROL en ROR geven een verschuiving in de accumulator of een geheugenlocatie, waarbij de overdrachtvlag als een soort negende bit fungeert. De figuren 1-3 en 1-4 geven hiervan de beschrijving. Let op het volgende:

. Verschuivingen naar links geven de overdrachtvlag de waarde die bit 7 was en de negatiefvlag krijgt de waarde van bit 6.

. Verschuivingen naar rechts geven de overdrachtvlag de waarde van bit 0.

. Rotaties bewaren alle bits, terwijl LSR en ASL het oude overdrachtvlag -bit vernietigen.

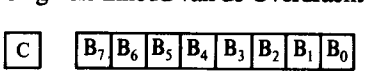
. Rotaties maken het u mogelijk seriele gegevens tussen het geheugen of de accumulator en de overdrachtvlag te verplaatsen. Dit is handig bij seriele in- en uitvoer en bij het verwerken van informatie bestaande uit enkele bits zoals Booleaanse indicators of pariteit.
Meervoudige bit-verschuivingen kunnen uitgevoerd worden door simpelweg het juiste aantal enkelvoudige bit-instructies uit te voeren.

Voorbeelden

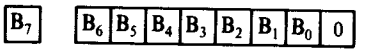
1. Roteer de accumulator drie posities naar rechts.

```
ROR     A
ROR     A
ROR     A
```

Originele inhoud van de Overdrachtvlag en de accumulator of de geheugen locatie



Na ASL (Rekenkundige verschuiving naar links) instructie



Figuur 1-1: De ASL (Rekenkundige verschuiving naar links) instructie

Originele inhoud van de Overdrachtvlag en de accumulator of de geheugen locatie

C	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

Na LSR (Logische verschuiving naar rechts) instructie

B_0	0	B_7	B_6	B_5	B_4	B_3	B_2	B_1

Figuur 1-2: De LSR (Logische verschuiving naar rechts) instructie

Originele inhoud van de Overdrachtvlag en de accumulator of de geheugen locatie

Overdrachtvlag Gegevens

C	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

Na ROL (Rotatie naar links) instructie

Overdrachtvlag Gegevens

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0	C

Figuur 1-3: De ROL (Rotatie naar links) instructie

Originele inhoud van de Overdrachtvlag en de accumulator of de geheugen locatie

Overdrachtvlag Gegevens

C	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

Na ROR (Rotatie naar rechts) instructie

Overdrachtvlag Gegevens

B_0	C	B_7	B_6	B_5	B_4	B_3	B_2	B_1

Figuur 1-4: De ROR (Rotatie naar rechts) instructie

2. Verschuif geheugenlocatie 1700_{16} vier posities logisch naar links.

```
ASL     $1700
ASL     $1700
ASL     $1700
ASL     $1700
```

Een alternatieve methode hiervoor maakt gebruik van de accumulator:

```
LDA     $1700
ASL     A
ASL     A
ASL     A
ASL     A
STA     $1700
```

De tweede methode is korter (10 bytes in plaats van 12) en sneller (16 klokcycli in plaats van 24), maar vernietigt de oude waarde van de accumulator.
U kunt rekenkundige verschuivingen toepassen door de overdrachtvlag te gebruiken om de waarde van bit 7 te bewaren. Rekenkundig naar rechts schuiven heet *teken-uitbreiding*, omdat het teken bit naar rechts gekopieerd wordt. Een verschuiving die op deze manier werkt, bewaart het teken van de complemente waarde van een tweecomplement en kan zo gebruikt worden om getekende getallen te normaliseren of te delen.

Voorbeelden

1. Schuif de accumulator 1 bit rekenkundig naar rechts, waarbij het teken (hoogstwaardige) bit bewaard blijft.

```
TAX                 ;BEWAAR DE ACCUMULATOR
ASL     A           ;SCHUIF BIT 7 NAAR DE OVERDRACHT
TXA                 ;HERSTEL DE ACCUMULATOR
ROR     A           ;SCHUIF DE ACCUMULATOR, WAARBIJ BIT 7
                    ;WORDT GEKOPIEERD
```

Als de processor een ROR A uitvoert, verplaatst deze de inhoud van de overdrachtvlag (de oude bit 7) naar bit 7 en bit 7 naar bit 6. Zodoende blijft het teken van de originele waarde bewaard.

2. Schuif de accumulator 1 bit rekenkundig naar links, waarbij het teken (hoogstwaardige) bit bewaard blijft.

```
ASL     A           ;SCHUIF A, VERPLAATS BIT 7 NAAR
                    ;OVERDRACHT
ROL     A           ;BEWAAR BIT 7 IN POSITIE 0
TAX
LSR     A           ;VERANDER OVERDRACHT IN HET OUDE BIT 7
TXA
ROR     A           ;SCHUIF DE ACCUMULATOR, BIT 7 BLIJFT
                    ;BEWAARD
```

of

```
        ASL     A           ;SCHUIF A, VERPLAATS BIT 7 NAAR
                            ;OVERDRACHT
        BCC     LEEGTEK     ;WAS BIT 7 1?
        ORA     #%10000000  ;  JA, LAAT HET DAN 1 BLIJVEN
        BMI     EINDE
LEEGTEK: AND    #%01111111  ;  NEE, LAAT HET DAN 0 BLIJVEN
EINDE:  RTS
```

BMI EINDE forceert in dit geval altijd een vertakking. *) Het begrip "vertakking" dient te worden gezien vanuit het standpunt van een stroomdiagram en betreft een relatieve sprong, die al of niet wordt genomen op basis van het testen van een vlag

BESLISSINGEN NEMEN

In dit deel bespreken we het nemen van drie soorten beslissingen:

- Vertakken als een bit aan of uit staat (een logische 1 of 0).
- Vertakken als twee waarden al dan niet gelijk zijn.
- Vertakken als de ene waarde groter is of kleiner is dan de andere.

Het eerste soort besluit maakt het de processor mogelijk de waarde van een vlag, schakelaar, statuslijn of andere binaire (AAN/UIT) input af te tasten. Het tweede soort besluit maakt het de processor mogelijk vast te stellen of een input of resultaat een specifieke waarde heeft. Het derde soort stelt de processor in staat te bepalen of een waarde onder of boven een numerieke drempel is. Ervan uitgaande dat de primaire waarde in de accumulator is en de secundaire waarde (indien nodig) zich in adres ADR bevindt, zijn de procedures als volgt:

Vertakken bij een bit aan of uit

. Stel vast of een bit aan of uit staat door een logische AND van de accumulator met een 1 in de juiste bit-positie en 0-en in de andere posities. De nulvlag geeft dan de bit-waarde weer en kan gebruikt worden voor een vertakking (met BEQ of BNE).

Voorbeelden

1. Vertak naar BEST als bit 5 van de accumulator 1 is.

```
        AND     #%00100000  ;TEST BIT 5 VAN A
        BNE     BEST
```

De nulvlag is alleen 1 wanneer bit 5 van de accumulator 0 is. Merk de inversie hiervan op.
Als we ervan uitgaan dat het gegeven in adres ADR zit, kunnen we de BIT-instructie gebruiken om eenzelfde effect te bereiken. Om naar BEST te vertakken als bit 5 van ADR 1 is, kunnen we gebruik maken van

```
        LDA     ADR
        AND     #%00100000
        BNE     BEST
```

of van

```
LDA     #%00100000
BIT     ADR
BNE     BEST
```

We moeten de volgorde van de handelingen omdraaien, omdat BIT geen onmiddellijke adressering toestaat. Het laat de accumulator echter ongewijzigd voor later gebruik.

2. Vertak naar BEST als bit 2 van de accumulator 0 is.

```
AND     #%00000100   ;TEST BIT 2 VAN A
BEQ     BEST
```

Er zijn speciale korte procedures om bit posities 0, 6 of 7 te bekijken. Bit 7 is direct beschikbaar als de negatiefvlag na een laad of verplaatsinstructie; bit 0 kan naar de overdrachtvlag worden verschoven met LSR A of ROR A; bit 6 kan naar de negatiefvlag worden verschoven met ASL A of ROL A.

3. Vertak naar BEST als bit 7 van geheugenlocatie ADR 1 is.

```
LDA     ADR          ;IS BIT 7 1?
BMI     BEST         ;JA, VERTAK
```

Let op dat LDA de nul- en de negatiefvlag beinvloedt. Maak dus gebruik van verplaatsinstructies als TAX, TYA, TSX (maar niet TXS) en PLA. Opberginstructies (inclusief PHA) beinvloeden geen enkele vlag.

4. Vertak naar BEST als bit 6 van de accumulator 0 is.

```
ASL     A            ;SCHUIF BIT 6 NAAR BIT 7
BPL     BEST
```

5. Vertak naar BEST als bit 0 van geheugenlocatie ADR 1 is.

```
ROR     ADR          ;SCHUIF BIT 0 VAN ADR NAAR OVERDRACHT
BCS     BEST         ;EN TEST DAN DE OVERDRACHT-BIT
```

De BIT-instructie heeft een speciale eigenschap die de programmeur in staat stelt om bit 6 en bit 7 kant en klaar te testen. Als de processor BIT uitvoert, krijgt de negatiefvlag de waarde van bit 7 van de aangesproken geheugenlocatie en krijgt de overschotvlag de waarde van bit 6, ongeacht de inhoud van de accumulator.

6. Vertak naar BEST als bit 7 van geheugenlocatie ADR 0 is.

```
BIT     ADR          ;TEST BIT 7 VAN ADR
BPL     BEST
```

Deze handelingen zijn niet afhankelijk van de accumulator en beinvloeden deze evenmin.

7. Vertak naar BEST als bit 6 van geheugenlocatie ADR 1 is.

```
BIT     ADR         ;TEST BIT 6 VAN ADR
BVS     BEST
```

Dit proces vereist nauwgezette documentatie, aangezien de overschotvlag op een speciale manier wordt gebruikt. Net als hiervoor verandert de inhoud van de accumulator niet en evenmin beinvloedt deze de gang van zaken.

Vertakkingen gebaseerd op gelijkheid

. Stel vast of de waarde in de accumulator gelijk is aan een andere waarde door middel van aftrekking. De nulvlag wordt 1 als de waarden gelijk zijn. De vergelijkinstructie (CMP) is handiger dan de aftrekinstructie (SBC) omdat Vergelijk de accumulator niet verandert of het overdracht-bit erbij betrekt.

Voorbeelden

1. Vertak naar BEST als de accumulator het getal WAARDE bevat.

```
CMP     #WAARDE     ;IS INHOUD = WAARDE?
BEQ     BEST        ;JA, VERTAK
```

We zouden ook het indexregister X met CPX of het indexregister Y met CPY kunnen gebruiken.

2. Vertak naar BEST als de inhoud van de accumulator niet gelijk is aan de inhoud van geheugenlocatie ADR.

= inhoud adr

```
CMP     ADR         ;IS INHOUD NUL?
BNE     BEST        ;NEE, VERTAK
```

3. Vertak naar BEST als de geheugenlocatie ADR 0 bevat.

```
LDA     ADR         ;IS INHOUD NUL?
BEQ     DEST        ;JA, VERTAK
```

We kunnen sommige speciale gevallen zonder de accumulator afhandelen.

4. Vertak naar BEST als geheugenlocatie ADR 0 bevat, maar verander de accumulator niet en laat de indexregisters ongewijzigd.

```
INC     ADR         ;TEST GEHEUGEN OP NUL
DEC     ADR
BEQ     BEST        ;VERTAK ALS HET GEVONDEN IS
```

5. Vertak naar BEST als de geheugenlocatie ADR geen 1 bevat.

```
DEC     ADR         ;ZET NULVLAG AAN ALS ADR 1 IS
BNE     BEST
```

Deze procedure wijzigt -uiteraard- de inhoud van de geheugenlocatie.

6. Vertak naar BEST als de geheugenlocatie ADR FF_{16} bevat.

```
INC     ADR         ;ZET NULVLAG AAN ALS ADR $FF IS
BEQ     BEST
```

INC heeft geen invloed op de overdrachtvlag, maar wel op de nulvlag. Let op dat u de accumulator niet met INC of DEC kunt verhogen of verlagen.

Vertakkingen gebaseerd op grootte-vergelijkingen

. Stel vast of de inhoud van de accumulator groter of kleiner is dan een andere waarde door middel van aftrekking. Als, zoals gebruikelijk is, de getallen niet getekend zijn, dan geeft de overdrachtvlag aan welke groter is. Let op dat de overdrachtvlag van de 6502 een negatief 'lenen' is bij vergelijkingen of aftrekkingen. Dit in tegenstelling tot het werkelijk 'lenen' zoals door de 8080, Z-80 en 6800 wordt gedaan. In het algemeen:

. Overdracht-bit = 1 als de inhoud van de accumulator groter dan of gelijk is aan de waarde die ervan is afgetrokken. Overdracht-bit = 1 als de aftrekking niet hoeft te 'lenen'.

. Overdracht-bit = 0 als de afgetrokken waarde groter is dan de inhoud van de accumulator. Oftewel, overdracht-bit = 0 als er bij de aftrekking geleend moeten worden.

Let op dat het overdracht-bit het omgekeerde van een normaal lenen is. Als de twee waarden gelijk zijn, dan is het overdracht-bit gelijk aan 1, net alsof de accumulator groter zou zijn. Als u echter wilt dat gelijke waarden het overdracht-bit beinvloeden alsof de andere waarde groter zou zijn, dan moet u slechts in omgekeerde volgorde aftrekken. Hierbij bergt u de accumulator in het geheugen op en laadt hem in plaats daarvan met de andere waarde.

Voorbeelden

1. Vertak naar BEST als de inhoud van de accumulator groter dan of gelijk is aan het getal WAARDE

```
CMP     #WAARDE     ;IS DATA BOVEN WAARDE
BCS     BEST        ;JA, VERTAK
```

Het overdracht-bit is aan als de niet-signerende aftrekking niet hoeft te lenen.

2. Vertak naar BEST als de inhoud van geheugenlocatie WERK1 kleiner is dan de inhoud van geheugenlocatie WERK2

```
LDA     WERK1       ;HAAL EERSTE INHOUD
CMP     OPER2       ;IS DIE KLEINER DAN DE TWEEDE?
BCC     BEST        ;JA, VERTAK
```

Het overdracht-bit wordt uit gezet als er bij de aftrekking geleend moet worden.

3. Vertak naar BEST als de inhoud van geheugenlocatie WERK1 kleiner dan of gelijk aan de inhoud van geheugenlocatie WERK2 is.

```
LDA     WERK2       ;HAAL TWEEDE INHOUD
CMP     WERK1       ;IS HIJ GROTER DAN OF GELIJK AAN DE
```

```
                        ;EERSTE?
        BCS     BEST    ;JA, VERTAK
```

Als we de accumulator met WERK1 hadden geladen en vergeleken met WERK2, zouden we alleen op de volgende voorwaarden kunnen vertakken:

. WERK1 groter dan of gelijk aan WERK2 (overdracht-bit aan)
. WERK1 kleiner dan WERK2 (overdracht-bit uit)

Aangezien geen van deze twee is wat we willen, moeten we de handelingen in de tegenovergestelde volgorde uitvoeren.
Als de waarden getekend zijn, moeten we rekening houden met de mogelijkheid van een tweecomplements-overschot. Dit is de situatie waarin het verschil tussen de getallen niet in zeven bits past en daarom het teken van het resultaat verandert. Bij voorbeeld als het ene nummer +7 is en het andere -125, dan is het verschil -132. Dit past niet in 8 bits (het is kleiner dan -128, het meest negatieve getal dat in 8 bits past).
Zodoende moeten we bij gesigneerde getallen rekening houden met de volgende mogelijkheden:

. Het resultaat heeft het teken (positief of negatief, aangegeven door de negatiefvlag) dat we willen hebben en de overschotvlag geeft aan dat het teken correct is.

. Het resultaat heeft niet het gewenste teken, maar de Overschot vlag geeft aan dat het tweecomplement-overloop het ware teken heeft veranderd.

We moeten zowel kijken naar een werkelijk positief (het teken dat we willen hebben, niet aangetast door overschot) als naar een vals negatief (het tegenovergestelde van het teken dat we willen zien, maar geinverteerd door een tweecomplement-overloop).

Voorbeelden

1. Vertak naar BEST als de inhoud van de accumulator (een gesigneerd getal) groter is dan of gelijk is aan het getal WAARDE.

```
        SEC             ;WIS DE GEINVERTEERDE LEEN (BORROW)
        SBC     #WAARDE ;VOER DE AFTREKKING UIT
        BVS     VNEG
        BPL     BEST    ;WERKELIJK POSITIEF, GEEN OVERSCHOT
        BMI     KLAAR
VNEG    BMI     BEST    ;VALS NEGATIEF, OVERSCHOT
KLAAR   NOP
```

2. Vertak naar BEST als de inhoud van de accumulator (een gesigneerd getal) kleiner is dan de inhoud van geheugenlocatie ADR.

```
        SEC             ;WIS DE GEINVERTEERDE LEEN (BORROW)
        SBC             ;VOER DE AFTREKKING UIT
        BVS     VNEG
        BMI     BEST    ;WERKELIJK POSITIEF, GEEN OVERSCHOT
        BPL     KLAAR
VNEG    BPL     BEST    ;VALS NEGATIEF, OVERSCHOT
KLAAR   NOP
```

Let op dat we het overdracht-bit aan moeten zetten en gedwongen zijn SBC te gebruiken, omdat CMP de overloopvlag niet beinvloedt.
Tabellen 1-8 en 1-9 vatten de algemene instructiereeksen samen om bij de 6502 beslissingen te nemen. Tabel 1-8 bevat de reeksen die afhankelijk zijn van de waarde van de accumulator; tabel 1-9 bevat de reeksen die afhankelijk zijn van numerieke vergelijkingen tussen de inhoud van de accumulator en een specifieke waarde of de inhoud van een geheugenlocatie. Tabellen 1-10 en 1-11 bevatten instructiereeksen die afhankelijk zijn van een indexregister of van de inhoud van een geheugenlocatie alleen.

Tabel 1-8: Beslissingen die van de accumulator afhangen

Conditie	Vlag zettende Instructie	Conditionele sprongopdracht
Een willekeurigbit van A = 0	AND #MASKER (1 in bit-positie)	BEQ
Een willekeurig bit van A = 1	AND #MASKER (1 in bit-positie)	BNE
Bit 7 van A = 0	ASL A of ROL A	BCC
	CMP #0 (bewaard A)	BPL
Bit 7 van A = 1	ASL A of ROL A	BCS
	CMP #0 (bewaard A)	BMI
Bit 6 van A = 0	ASL A of ROL A	BPL
Bit 6 van A = 1	ASL A of ROL A	BMI
Bit 0 van A = 0	LSR A of ROR A	BCC
Bit 0 van A = 1	LSR A of ROR A	BCS
(A) = 0	LDA, PLA, TAX, TAY, TXA en TYA	BEQ
(A) <> 0	LDA, PLA, TAX, TAY, TXA en TYA	BNE
(A) positief (MSB = 0)	LDA, PLA, TAX, TAY, TXA en TYA	BPL
(A) negatief (MSB = 0)	LDA, PLA, TAX, TAY, TXA en TYA	BMI

Tabel 1-9: Beslissingen gebaseerd op numerieke vergelijkingen

Conditie	Vlag zettende instructie	Conditionele Sprongopdracht
(A) = WAARDE	CMP #WAARDE	BEQ
(A) <> WAARDE	CMP #WAARDE	BNE
(A) >= WAARDE (zonder teken)	CMP #WAARDE	BCS
(A) < WAARDE (zonder teken)	CMP #WAARDE	BCC
(A) = (ADRES)	CMP ADRES	BEQ
(A) <> (ADRES)	CMP ADRES	BNE
(A) >= (ADRES) (zonder teken)	CMP ADRES	BCS
(A) < (ADRES) (zonder teken)	CMP ADRES	BCC

Tabel 1-10: Beslissingen die van een indexregister afhangen

Conditie	Vlag zettende instructie	Conditionele sprongopdracht
(X of Y) = WAARDE	CPX of CPY #WAARDE	BEQ
(X of Y) <> WAARDE	CPX of CPY #WAARDE	BNE
(X of Y) >= WAARDE (zonder teken)	CPX of CPY #WAARDE	BCS
(X of Y) < WAARDE (zonder teken)	CPX of CPY #WAARDE	BCC
(X of Y) = (ADRES)	CPX of CPY ADRES	BEQ
(X of Y) <> (ADRES)	CPX of CPY ADRES	BNE
(X of Y) >= (ADRES) (zonder teken)	CPX of CPY ADRES	BCS
(X of Y) < (ADRES) (zonder teken)	CPX of CPY ADRES	BCC

Tabel 1-11: Beslissingen die alleen van geheugenlocaties afhangen

Conditie	Vlag zettende instructie	Conditionele sprongopdracht
Bit 7 = 0	BIT ADRES	BPL
	ASL ADRES of ROL ADRES	BCC
Bit 7 = 1	BIT ADRES	BMI
	ASL ADRES of ROL ADRES	BCS
Bit 6 = 0	BIT ADRES	BVS
	ASL ADRES of ROL ADRES	BPL
Bit 6 = 1	BIT ADRES	BVS
	ASL ADRES of ROL ADRES	BMI
(ADRES) = 0	INC ADRES of DEC ADRES	BEQ
(ADRES) <> 0	INC ADRES of DEC ADRES	BNE
Bit 0 = 0	LSR ADRES of ROR ADRES	BCC
Bit 0 = 1	LSR ADRES of ROR ADRES	BCS

LUSSEN

De eenvoudigste manier om een programma-lus (herhaling van een reeks instructies) bij de 6502 te gebruiken is als volgt:

1. Laad een indexregister of geheugenlocatie met het aantal keren dat de serie uitgevoerd moet worden.

2. Voer de serie uit.

3. Verlaag het indexregister of de geheugenlocatie met 1.

4. Keer terug naar stap 2 als het resultaat van stap 3 geen 0 is.

Dit zou er in een programma aldus kunnen uitzien:

```
            LDX    #NKEREN    ;AANTAL HERHALINGEN
LUS          .
             .
             .       te herhalen instructies.
             .
            DEX
            BNE  LUS
```

Niets, behalve de duidelijkheid, weerhoudt ons ervan om op te tellen (met gebruik van INX, INY of INC), waarbij de initialisatie uiteraard aangepast dient te worden. Zoals we later zullen zien, is een 16-bits teller makkelijker op dan af te tellen. In ieder geval, de te repeteren instructies mogen het tellen van de repetities niet doorkruisen. De teller kan in een geheugenplaats of in een indexregister opgeborgen worden. De speciale kenmerken van indexregister X zijn het gebruik bij preindexering en de ruime mogelijkheden van pagina 0 indexeringsmodi. Zoals gebruikelijk zijn geheugenlocaties op pagina 0 korter en sneller te gebruiken dan geheugenlocaties op andere pagina's.
Uiteraard kunt u een indexregister of een enkele geheugenlocatie als een teller gebruiken, waarbij het aantal herhalingen beperkt blijft tot 256. Een groter aantal herhalingen is te bereiken door geneste lussen uit te voeren, die een enkele geheugenlocatie of register gebruiken. De lussen kunnen ook twee geheugenlocaties gebruiken, zoals in het volgende voorbeeld te zien is:

. Geneste lussen

```
            LDX    #NMAALM    ;BEGIN UITWENDIGE TELLER
LUSU        LDY    #NMAALL    ;BEGIN INWENDIGE TELLER
LUSI         .
             . te herhalen instructies
             .
            DEY               ;VERLAAG INWENDIGE TELLER
            BNE    LUSI
            DEX               ;VERLAAG UITWENDIGE TELLER
            BNE    LUSU
```

De buitenste lus zet de binnenste teller (indexregister Y) terug naar zijn oorspronkelijke waarde (NMAALL) na elke verlaging van de buitenste teller (indexregister X). Het nesten produceert een vermenigvuldigingsfactor - de instructies die bij LUSI beginnen, worden NMAALM x NMAALL

keer herhaald. Uiteraard zouden bij een meer algemene (en verstandiger) benadering, twee geheugenlocaties op pagina 0 in plaats van twee indexregisters gebruikt worden.

. 16-bit teller in twee geheugenlocaties

```
        LDA     #NMAALLT  ;INITIALISEER LWB VAN TELLER
        STA     TELL
        LDA     #NMAALHT  ;INITIALISEER HWB VAN TELLER
        STA     TELH
LUS      .
         . Te herhalen instructies
         .
        INC     NMAALLT   ;VERHOOG LWB VAN TELLER
        BNE     LUS
        INC     NMAALHT   ;EN OVERDRACHT NAAR HWB VAN TELLER,
                          ;INDIEN NODIG
        BNE     LUS
```

Het idee hierachter is in alleen het laagstwaardige byte te verhogen, tenzij er een overdracht naar de hoogstwaardige byte is. Let op dat we een overdracht alleen kunnen vaststellen door de nulvlag te controleren, omdat INC de overdrachtvlag niet beinvloedt. Optellen is veel makkelijker dan aftellen; de vergelijkbare routine om een 16-bit teller te verlagen is

```
        LDA     NMAALL    ;IS LWB VAN TELLER NUL?
        BNE     TELLWB
        DEC     NMAALH    ;JA, LEEN VAN HWB
TELLWB  DEC     NMAALL    ;VERLAAG LWB VAN TELLER
        BNE     LUS       ;GA VERDER ALS LWB NOG NIET NUL IS
        LDA     NMAALH    ;OF ALS HWB NOG NIET NUL IS
        BNE     LUS
```

Als we echter optellen, moeten we onthouden dat we de teller initialiseren op het complement van de gewenste waarde (aangegeven met de namen NMAALLT en NMAALHT in het programma dat INC gebruikt).

ARRAY-MANIPULATIE

De eenvoudigste methode om toegang tot een afzonderlijk element van een array te krijgen, is door het gebruik van geindexeerde adressering. Men kan dan

1. Het element manipuleren door vanaf het beginadres van het array te indexeren.

2. Het volgende element (op het adres erboven) bereiken door het indexregister te verhogen met INX of INY, of het voorafgaande element bereiken (op het adres er onder) door het indexregister te verlagen met DEX of DEY. Men kan ook de basis veranderen; dit is eenvoudig als de basis een absoluut adres is, maar uiterst lastig als zij een indirect adres is.

3. Een willekeurig element bereiken door een indexregister met haar index te laden. Array-manipulaties zijn makkelijk te programmeren als het array eendimensionaal is, de elementen 1 byte gebruiken en het aantal elementen kleiner dan 256 is. Enkele voorbeelden zijn:

. Tel een element van een array op bij de accumulator. Het basisadres van het array is de constante BASIS. Pas het indexregister X aan, zodat het naar het volgende 8-bits element wijst.

```
        ADC     BASIS,X     ;TEL HUIDIGE ELEMENT OP BIJ DE
                            ;ACCUMULATOR
        INX                 ;ADRES VAN VOLGENDE ELEMENT
```

. Controleer of een element van een array 0 is en tel 1 bij de geheugenlocatie NTEL op, als dit het geval is. Ga ervan uit dat het adres van het array de constante BASIS is en zijn index in indexregister X staat. Pas indexregister X aan, zodat het naar het voorafgaande 8-bits element wijst.

```
        LDA     BASIS,X     ;HAAL HUIDIGE ELEMENT
        BNE     PASAAN      ;IS DE WAARDE NUL?
        INC     NTEL        ;JA, TEL 1 BIJ DE NULTELLER OP
PASAAN  DEX                 ;ADRES VAN VOORAFGAANDE ELEMENT
```

. Laad de accumulator met het 35ste element van een array. Ga er van uit dat het beginadres van het array in BASIS staat.

```
        LDX     #35         ;HAAL INDEX VAN GEWENSTE ELEMENT
        LDA     BASIS,X     ;HAAL HET ELEMENT
```

De meest efficiente methode om een array te bewerken, is door op het hoogste adres te beginnen en terug te werken. Dit is de beste aanpak, omdat u het indexregister naar nul kunt laten aftellen en kunt stoppen als de nulvlag is aan staat. U moet de initialisatie en de geindexeerde bewerkingen een klein beetje aanpassen, omdat de nul-index nooit wordt gebruikt. De wijzigingen zijn:

. Laad het indexregister met het aantal elementen.

. Gebruik het basisadres START-1, waarbij START het laagste adres is dat het array gebruikt.

Als we bij voorbeeld de som willen verkrijgen van LENGTE-elementen, die bij START beginnen, moeten we de volgende routine gebruiken

```
        LDX     #LENGTE     ;BEGIN BIJ HET EINDE VAN HET ARRAY
        LDA     #0          ;WIS OM TE BEGINNEN DE SOM
TELBYTE CLC
        ADC     START-1,X ;TEL HET VOLGENDE ELEMENT ERBIJ OP
        DEX
        BNE     TELBYTE     ;TEL DE ELEMENTEN
```

De behandeling van arrays kan moeilijker worden als u meer dan een element tijdens elke bewerking nodig hebt (bij voorbeeld bij een sortering die de uitwisseling van elementen vereist), als de elementen meer dan een byte lang zijn of als de elementen zelf adressen zijn (bij voorbeeld in een tabel van beginadressen). Oorzaak van dit probleem is het ontbreken van 16-bits registers of 16-bits instructies. De processor kan nooit de opdracht krijgen meer dan 8 bits te bewerken. Voorbeelden van meer algemene array-manipulatie zijn:

. Laad geheugenlocaties WIJSH en WIJSL met een 16-bits element van een array (waarbij de LWB het eerst opgeborgen wordt). Het basisadres van het array is BASIS en de index van het element bevindt zich in indexregister X. Pas X aan, zodat het naar het volgende 16-bits element wijst.

```
        LDA     BASIS,X     ;HAAL LWB VAN ELEMENT
        STA     WIJSL
        INX
        LDA     BASIS,X     ;HAAL HWB VAN ELEMENT
        INX                 ;ADRES VAN VOLGENDE ELEMENT
```

De enkele instructie LDA BASIS+1,X laadt de accumulator vanuit hetzelfde adres als in de reeks

```
        INX
        LDA     BASIS,X
```

veronderstellende dat X in oorsprong geen FF_{16} bevatte. Als we echter een basisadres indirect gebruiken, zijn de andere mogelijkheden

```
        INC     PGNUL       ;VERHOOG BASISADRES
        BNE     INDEX
        INC     PGNUL+1     ;MET OVERDRACHT INDIEN NODIG
INDEX   LDA     (PGNUL),Y
```

of

```
        INY
        LDA     (PGNUL),Y
```

De tweede is veel korter, maar de eerste kan arrays groter dan 256 bytes verwerken.

. Verwissel een element van een array met het volgende element, als de twee nog niet in aflopende volgorde staan. Veronderstel dat de elementen 8-bits grote, niet gesigneerde getallen zijn. Het basisadres van het array is BASIS en de index van het eerste getal staat in indexregister X.

```
        LDA     BASIS,X     ;HAAL ELEMENT
        CMP     BASIS+1,X   ;IS VOLGENDE KLEINER?
        BCS     KLAAR       ;NEE, GEEN UITWISSELING NODIG
        PHA                 ;JA, BEWAAR ELEMENT
        LDA     BASIS+1,X   ;VERWISSEL
        STA     BASIS,X
        PLA
        STA     BASIS+1,X
KLAAR   INX                 ;GA NAAR VOLGENDE ELEMENT
```

. Laad de accumulator vanuit het 12de indirecte adres in een tabel. Veronderstel dat de tabel begint bij adres BASIS

```
        LDX     #24         ;HAAL VERDUBBELDE OFFSET VOOR INDEX
        LDA     BASIS,X     ;HAAL LWB VAN ADRES
        STA     PGNUL       ;BEWAAR OP PAGINA NUL
        INX
        LDA     BASIS,X     ;HAAL MWB VAN ADRES
        STA     PGNUL+1     ;BEWAAR OP PAGINA NUL
        LDY     #0
        LDA     (PGNUL),Y   ;LAAD INDIRECT DOOR INDEXERING MET
                            ;NUL
```

Merk op dat u de index moet verdubbelen om tabellen met adressen te kunnen verwerken, omdat elk 16-bits adres twee geheugenbytes beslaat.
Als de gehele tabel op pagina 0 staat, kunnen we de gepreindexeerde (geindexeerd indirect) adresseringsmodus gebruiken.

```
        LDX     #24         ;HAAL VERDUBBELDE OFFSET VOOR INDEX
        LDA     (BASIS,X)   ;LAAD VANUIT GEINDEXEERD INDIRECT
                            ;ADRES
```

Nog steeds moet de index worden verdubbeld. In dit geval moet ook de tabel met adressen op pagina 0 RAM worden geindexeerd.
We kunnen array-bewerking algemeniseren door het basisadres in twee geheugenlocaties op pagina 0 op te bergen en de gepostindexeerde (indirect geindexeerde) adresseringsmodus te gebruiken. Nu kan het basisadres een variabele zijn. Deze methode gaat van het gebruik van pagina 0 en indexregister Y uit, en is alleen beschikbaar voor een beperkte set instructies.

Voorbeelden

1. Tel een element van een array op bij de accumulator. Het basisadres van het array staat in geheugenlocaties PGNUL en PGNUL+1. De index van het element staat in indexregister Y. Pas het indexregister Y aan, zodat dit naar het volgende 8-bits element wijst.

```
        CLC
        ADC     (PGNUL),Y   ;TEL HUIDIGE ELEMENT OP
        INY                 ;ADRES VAN VOLGENDE ELEMENT
```

2. Controleer of een element van een array 0 is en tel 1 bij de geheugenlocatie ZTEL op als dit het geval is. Veronderstel dat het basisadres van het array in geheugenlocaties PGNUL en PGNUL+1 staat. De index van het element staat in indexregister Y. Pas indexregister Y aan, zodat het naar het voorafgaande 8-bits element wijst.

```
        LDA     (PGNUL),Y   ;HAAL HUIDIGE ELEMENT
        BNE     PASAAN      ;IS DE WAARDE NUL?
        INC     ZTEL        ;JA, TEL 1 BIJ NULTELLER OP
PASAAN  DEY                 ;ADRES VAN VOORAFGAANDE ELEMENT
```

Postindexering stelt ons ook in staat arrays te bewerken die meer dan 256 bytes groot zijn. Zoals eerder beschreven, is het het eenvoudigste om bij grote tellingen een 16-bits complementaire teller in twee geheugenlocaties bij te houden. Als het array aangegeven is met een basisadres op pagina nul, dan kunnen we dat adres aanpassen op het moment dat we de hoogstwaardige byte van de teller moeten verhogen. Als we bij voorbeeld een stuk geheugen willen wissen, beschreven met een complementaire teller in de geheugenlocaties TELH en TELL en met het eerste basisadres in de geheugenlocaties PGNUL en PGNUL+1, dan kunnen we de volgende routine gebruiken:

```
        LDA     #0          ;DATA = NUL
        TAY                 ;INDEX = NUL
WIS     STA     (PGNUL),Y   ;WIS EEN BYTE
        INY                 ;GA NAAR VOLGENDE BYTE
        BNE     CTRLRTEL
        INC     PGNUL+1     ;EN NAAR VOLGENDE PAGINA INDIEN
                             NODIG
CTRLRTEL INC    TELL        ;TEL BYTES
```

```
BNE    WIS
INC    TELH          ;MET OVERDRACHT NAAR HWB
BNE    WIS
```

De methode hier is naar de volgende pagina door te gaan door de hoogstwaardige byte van het indirecte adres te verhogen als we met een deel van 256 bytes klaar zijn.
Men kan array-bewerking ook vereenvoudigen door het aantal vermenigvuldigingen dat voor de indexering bij optellingen nodig is, te verminderen. In het bijzonder kunnen we bij het bewerken van twee-bytes elementen ASL A gebruiken om een index in de accumulator te verdubbelen.

Voorbeeld

Laad de accumulator vanuit een indirect adres, geindexeerd met de inhoud van geheugenlocatie INDEX. Veronderstel dat de tabel begint bij het adres BASIS.

```
LDA    INDEX         ;HAAL INDEX
ASL    A             ;EN VERDUBBEL HEM VOOR 2-BYTES
                     ;ELEMENTEN
TAX
LDA    BASIS,X       ;HAAL LWB VAN INDIRECT ADRES
STA    PGNUL         ;BEWAAR HET OP PAGINA NUL
INX
LDA    BASIS,X       ;HAAL HWB VAN INDIRECT ADRES
STA    PGNUL+1       ;BEWAAR HEM OP PAGINA NUL
LDY    #0            ;PREINDEXERING MET NUL
LDA    (PGNUL),Y
```

Net als voorheen kunnen we, als de hele tabel met indirecte adressen op pagina 0 staat, de gepreindexeerde (geindexeerde indirecte) adresseringsmodus gebruiken.

```
LDA    INDEX      ;HAAL INDEX
ASL    A          ;VERDUBBEL INDEX VOOR 2-BYTES
                  ;ELEMENTEN
TAX
LDA    (BASIS,X)  ;LAAD VANUIT GEINDEXEERD INDIRECT
                  ;ADRES
```

U kunt langere arrays indexeren door gebruik te maken van de postindexering (indirect geindexeerde) modus. Hierbij moeten we een basisadres opbouwen met optelling voor de indexering, omdat de indexregisters van de 6502 slechts 8 bits groot zijn.

Voorbeeld

Laad de accumulator vanuit het element van een array, gedefinieerd door een beginadres BASIS (BASISH hoogstwaardige byte, BASISL laagstwaardige byte) en een 16-bits index in geheugenlocaties INDEX en INDEX+1 (HWB in INDEX+1).

```
LDA    #BASISL       ;VERPLAATS LWB VAN BASIS NAAR PAGINA

                     ;NUL
STA    PGNUL
LDA    #BASISH       ;TEL HWB'S VAN BASIS EN INDEX OP
STA    WIJSL
```

```
CLC
ADC    INDEX+1
STA    PGNUL
LDY    INDEX          ;GEBRUIK NADRUKKELIJK LWB VAN INDEX
LDA    (PGNUL),Y      ;HAAL ELEMENT
```

HET WERKEN MET TABELLEN

Tabellen kunnen met dezelfde procedures worden bewerkt, als bij array-manipulatie. Enkele voorbeelden zijn:

. Laad de accumulator met een element van een tabel. Ga ervan uit dat BASIS (een constante) het basisadres van de tabel is en de 8-bits index zich in geheugenlocatie INDEX bevindt.

```
LDX    INDEX      ;HAAL DE INDEX
LDA    BASIS,X    ;HAAL HET ELEMENT OP
```

Het probleem is gecompliceerder als INDEX een 16-bits getal is.

. Laad de accumulator met een element uit een tabel. Veronderstel dat het basisadres van de tabel de constante BASIS is, die uit de bytes BASISH en BASISL bestaat. De 16-bit index staat in de geheugenlocaties INDEX en INDEX+1 (HWB in INDEX+1).

De procedure is dezelfde als die welke we zojuist voor een array hebben laten zien. U moet de hoogstwaardige byte van de index optelen bij de hoogstwaardige byte van de basis. Daarna kunt u postindexering gebruiken om het gewenste element op te halen.

. Laad de geheugenlocaties WIJSH en WIJSL met een 16-bit element uit een tabel. Veronderstel dat het basisadres van de tabel BASIS (een constante) is en dat de index in geheugenlocatie INDEX staat.

```
LDA    INDEX      ;HAAL DE INDEX
ASL    A          ;VERDUBBEL DEZE VOOR 2-BYTES ELEMENTEN
TAX
LDA    BASIS,X    ;HAAL LWB VAN ELEMENT
STA    WIJSL
INX
LDA    BASIS,X    ;HAAL HWB VAN ELEMENT
STA    WIJSH
```

Als het basisadres bestaat uit een variabele in twee geheugenlocaties op pagina 0 (PGNUL en PGNUL+1), dan kunnen we het volgende programma gebruiken:

```
LDA    INDEX          ;HAAL DE INDEX
ASL    A              ;VERDUBBEL VOOR 2-BYTES ELEMENTEN
TAY
LDA    (PGNUL),Y      ;HAAL LWB VAN ELEMENT
STA    WIJSL
INY
LDA    (PGNUL),Y      ;HAAL HWB VAN ELEMENT
STA    WIJSH
```

Als we het programma aanpassen, dan kunnen we met een array met meer dan 128 elementen werken.

```
        LDA     INDEX       ;HAAL DE INDEX
        ASL     A           ;VERDUBBEL VOOR 2-BYTES ELEMENTEN
        BCC     LDELEM
        INC     PGNUL+1     ;WE HEBBEN OVERDRACHT, VERHOOG
                            ;INDIRECT ADRES
LDELEM  TAY
        LDA     (PGNUL),Y   ;HAAL LWB VAN ELEMENT
        STA     WIJSL
        INY
        LDA     (PGNUL),Y   ;HAAL HWB VAN ELEMENT
        STA     WIJSH
```

Een verdere uitbreiding bewerkt ook een 16-bits index.

```
        LDA     INDEX       ;HAAL LWB VAN INDEX
        ASL     A           ;VERDUBBEL DEZE
        TAY
        LDA     INDEX+1     ;HAAL HWB VAN INDEX
        ROL     A           ;VERDUBBEL DEZE MET OVERDRACHT
        ADC     PGNUL+1     ;EN TEL RESULTAAT OP BIJ INDIRECT
                            ;ADRES
        STA     PGNUL+1
        LDA     (PGNUL),Y   ;HAAL LWB VAN ELEMENT
        STA     WIJSL
        INY
        LDA     (PGNUL),Y   ;HAAL HWB VAN ELEMENT
        STA     WIJSH
```

. Geef de besturing over, (spring) naar een 16-bits adres dat je uit een tabel hebt gekregen. Veronderstel dat het basisadres van de tabel BASIS (een constante) is en dat de index in geheugenlocatie INDEX staat.

Dit zijn de twee mogelijkheden: stop het adres dat u vanuit de tabel haalt in twee geheugenlocaties en gebruik een indirecte sprong of plaats de adressen uit de tabel in de stack en gebruik een RTS (Return from Subroutine, keer terug van subroutine).

OPTIE 1: Indirecte Sprong

```
        LDA     INDEX       ;HAAL INDEX
        ASL     A           ;VERDUBBEL VOOR 2-BYTES ELEMENTEN
        TAX
        LDA     BASIS,X     ;HAAL LWB VAN BESTEMMINGSADRES
        STA     TEMP        ;BERG LWB ERGENS OP
        INX
        LDA     BASIS,X     ;HAAL HWB VAN BESTEMMINGSADRES
        STA     TEMP+1      ;BERG IN VOLGENDE BYTE OP
        JMP     (TEMP)      ;EN INDIRECTE SPRONG NAAR BESTEMMING
```

JMP is de enige 6502 instructie die een echte indirecte adressering toestaat. TEMP en TEMP+1 kunnen een willekeurige plaats in het geheugen zijn; zij hoeven niet op pagina 0 te staan.

OPTIE 2: Sprong Middels de Stack

```
LDA     INDEX      ;HAAL DE INDEX
ASL     A          ;EN VERDUBBEL VOOR 2-BYTES ELEMENTEN
TAX
INX
LDA     BASIS,X    ;HAAL HWB VAN BESTEMMINGSADRES
PHA                ;BEWAAR DEZE OP STACK
DEX
LDA     BASIS,X    ;HAAL LWB VAN BESTEMMINGSADRES
PHA                ;BEWAAR DEZE OP STACK
RTS                ;EN STAP OVER NAAR BESTEMMING
```

Dit alternatief is lastig om de volgende redenen:

. Een RTS verhoogt de programmateller met 1, nadat deze van de stack is geladen. Zodoende moeten de adressen in de tabel 1 lager zijn dan de werkelijke adressen waar u naartoe wilt springen. Deze manier maakt het afhandelen van een JSR (Jump to Subroutine, sprong naar subroutine) instructie, maar het kan de programmeur ook flink verwarren:

. U moet onthouden dat de stack van boven naar beneden 'groeit', naar een lager geheugenadres. Om het bestemmingsadres in de normale volgorde (laagstwaardige byte op het lagere adres) te laten verschijnen, moeten we eerst de hoogstwaardige byte op de stack zetten.

. Het gebruik van een RTS is verwarrend. Hoe kunnen we van een routine terugkeren die nooit aangeroepen is? In feite gebruikt u hier een methode waarbij u met een RTS een subroutine aanroept. U moet onthouden dat een RTS eigenlijk een spronginstructie is die de nieuwe waarde van de programmateller van de stack-top haalt. Hoewel in het algemeen de RTS wordt gebruikt om vanuit een subroutine terug te springen naar het hoofdprogramma (vandaar de mnemonic), is er geen reden om haar tot dit gebruik te beperken. De aanduiding RTS kan de programmeur verwarren, maar de processor doet precies wat hij doen moet. Zorgvuldige documentatie kan een hoop verwarring voorkomen.

Sprongtabellen worden in het algemeen gebruikt om opdrachten vanuit het toetsenbord te decoderen en om op functietoetsen van een terminal te reageren.

KARAKTERBEWERKING

De eenvoudigste manier om karakters te bewerken is om ze als ongesigneerde 8-bits getallen te behandelen. De letters en getallen vormen geordende reeksen van de ASCII-karakters. De ASCII-waarde van de letter A is 1 minder dan de ASCII-waarde van de letter B. Het bewerken van een enkel karakter is gelijk aan het bewerken van een gewoon 8-bits ongesigneerd getal. Voorbeelden zijn

. Vertak naar adres BEST als de accumulator een ASCII E bevat.

```
CMP     #'E        ;IS HET EEN E?
```

```
        BEQ     BEST        ;JA, VERTAK
```

. Zoek in een string (alfanumerieke reeks) die begint in geheugenlocatie STRNG tot het eerste niet-lege karakter is gevonden.

```
        LDX     #0          ;WIJS NAAR BEGINSTRING
        LDA     #'          ;HAAL EEN LEEG KARAKTER (=SPATIE) VOOR
                            ;CONTROLE
CONTRK  CMP     STRNG,X     ;IS HET VOLGENDE KARAKTER LEEG?
        BNE     KLAAR       ;NEE, KLAAR
        INX                 ;JA, GA NAAR VOLGENDE KARAKTER
        JMP     CONTRK
KLAAR   NOP
```

of

```
        LDX     #$FF        ;WIJS NAAR BYTE VOOR EERSTE KARAKTER
        LDA     #'          ;HAAL EEN LEEG KARAKTER TER VERGELIJKING
CONTRK  INX                 ;GA NAAR VOLGENDE KARAKTER
        CMP     STRNG,X     ;IS HET EEN LEGE?
        BEQ     CONTRK      ;JA, BLIJF ZOEKEN
```

Vertak naar adres BEST als de accumulator een letter in de reeks C tot en met F is.

```
        CMP     #'C         ;IS KARAKTER KLEINER DAN C?
        BCC     KLAAR       ;JA, KLAAR
        CMP     #'G         ;IS KARAKTER KLEINER DAN G?
        BCC     BEST        ;JA, DAN MOET HIJ TUSSEN C EN F LIGGEN
KLAAR   NOP
```

Hoofdstuk 8 bevat verdere voorbeelden van string-bewerking.

CODE-CONVERSIE

U kunt gegevens van de ene naar de andere code overzetten door rekenkundige of logische bewerkingen uit te voeren (als de verhouding tussen de codes eenvoudig is) of tabellen gebruiken (als de verhouding complex is).

Voorbeelden

1. Zet een ASCII-waarde om in haar BCD (binary coded decimal) equivalent.

```
        SEC                 ;WIS DE GEINVERTEERDE LEEN
        SBC     #'0         ;ZET ASCII OM NAAR BCD
```

Omdat de ASCII-waarden geordend zijn, hoeft u alleen de offset af te trekken (ASCII 0).

U kunt ook de bit-posities 4 en 5 met een enkele instructie wissen

```
        AND     #%11001111  ;ZET ASCII OM NAAR BCD
```

Zowel de rekenkundige als de logische instructie zet, bij voorbeeld, ASCII 0 (30_{16}) om in decimaal 0 (0_{16}).

2. Zet een BCD-getal om in het ASCII-equivalent.

```
        CLC             ;WIS HET OVERDRACHTBIT
        ADC     #'0     ;ZET BCD OM IN ASCII
```

De geinverteerde conversie is even eenvoudig. U kunt de bit posities 4 en 5 aanzetten met een enkele instructie

```
        ORA     #%00110000  ;ZET BCD OM IN ASCII
```

Zowel de rekenkundige als ook de logische instructie zetten, bij voorbeeld, het decimale getal 6 (6_{16}) om in ASCII 6 (36_{16})

3. Zet een 8-bits code om in een andere, met behulp van een zoektabel. Veronderstel dat de zoektabel begint bij het adres NIEUWCD en is geindexeerd op de waarde in de originele code (het 27ste element is bijvoorbeeld de waarde in de nieuwe code die overeenkomt met 27 in de oude code). Veronderstel dat de gegevens in geheugenlocatie CODE staan.

```
        LDX     CODE        ;HAAL DE OUDE CODE
        LDA     NIEUWCD,X   ;CONVERTEER HEM NAAR DE NIEUWE CODE
```

Hoofdstuk 4 bevat nadere voorbeelden van code-conversie.

MEERVOUDIGE-PRECISIE-REKENKUNDE

Voor meervoudige-precisie-rekenkunde is een serie van 8-bits handelingen nodig. U moet:

. De Overdracht wissen voordat u met optellen begint en de overdracht aan zetten voordat u met aftrekken begint. Dit omdat er nooit een overdracht naar of een lening van de laagstwaardige byte is.

. Gebruik de Add with Carry, optellen met overdracht (ADC) of Subtract with Borrow, aftrekken met onthouden (lenen) (SBC) om een 8-bits handeling uit te voeren en gebruik de overdracht of lening van de vorige handeling.

Een typisch 64-bit optelprogramma is:

```
        LDX     #8          ;AANTAL BYTES = 8
        CLC                 ;WIS DE OVERDRACHT VOOR BEGIN
TELOPB  LDA     NUM1-1,X    ;HAAL EEN BYTE VAN EEN WERKGETAL
        ADC     NUM2-1,X    ;TEL EEN BYTE VAN HET ANDERE WERKGETAL
                            ;OP
        STA     NUM1-1,X    ;BERG DE 8-BITS SOM OP
        DEX
        BNE     TELOPB      ;TEL AANTAL BYTE-HANDELINGEN
```

Hoofdstuk 6 bevat nadere voorbeelden.

VERMENIGVULDIGING EN DELING

Vermenigvuldiging kan op een aantal manieren worden uitgevoerd. Een techniek is om eenvoudige vermenigvuldigingen om te zetten in optellingen of schuivingen naar links.

Voorbeelden

1. Vermenigvuldig de inhoud van de accumulator met 2.

```
ASL     A           ;VERDUBBEL A
```

2. Vermenigvuldig de inhoud van de accumulator met 5

```
STA     TEMP
ASL     A           ;A MAAL 2
ASL     A           ;A MAAL 4
ADC     TEMP        ;A MAAL 5
```

Deze benadering gaat ervan uit dat een schuiving van de accumulator naar links een overdracht teweegbrengt. Deze methode is vaak handig om de locatie van elementen van een tweedimensionaal array te bepalen. Laten we bij voorbeeld eens aannemen dat we een serie temperatuurmetingen op vier verschillende locaties uit elk van drie verschillende tanks hebben. We organiseren de metingen in een tweedimensionaal array T(I,J), waarbij I het nummer van de tank is (1, 2 of 3) en J het nummer van de positie in de tank (1, 2, 3 of 4). We bergen de metingen als volgt achter elkaar op in het lineaire geheugen van de computer, beginnende bij tank 1:

BASIS	T(1,1)	meting in tank 1, locatie 1
BASIS + 1	T(1,2)	meting in tank 1, locatie 2
BASIS + 2	T(1,3)	meting in tank 1, locatie 3
BASIS + 3	T(1,4)	meting in tank 1, locatie 4
BASIS + 4	T(2,1)	meting in tank 2, locatie 1
BASIS + 5	T(2,2)	meting in tank 2, locatie 2
BASIS + 6	T(2,3)	meting in tank 2, locatie 3
BASIS + 7	T(2,4)	meting in tank 2, locatie 4
BASIS + 8	T(3,1)	meting in tank 3, locatie 1
BASIS + 9	T(3,2)	meting in tank 3, locatie 2
BASIS + 10	T(3,3)	meting in tank 3, locatie 3
BASIS + 11	T(3,4)	meting in tank 3, locatie 4

Zodoende bevindt de meting T(I,J) zich op adres BASIS + 4 x (I-1) + (J-1). Als I in geheugenlocatie IND1 en J in geheugenlocatie IND2 staat, kunnen we de accumulator als volgt met T(I,J) laden:

```
LDA     IND1        ;HAAL I
SEC
SBC     #1          ;BEREKEN I-1
ASL     A           ;2 X (I-1)
ASL     A           ;4 X (I-1)
SEC
SBC     #1          ;4 X (I-1) + J-1
TAX
```

```
LDA     BASIS,X    ;HAAL T(I,J)
```

We kunnen deze methode uitbreiden, zodat we arrays met meerdere dimensies kunnen bewerken. Uiteraard is het programma eenvoudiger als we I-1 in geheugenlocatie IND1 en J-1 in geheugenlocatie IND2 stoppen. We kunnen de accumulator dan met T(I,J) laden door

```
LDA     IND1          ;HAAL I-1
ASL     A             ;2 X (I-1)
ASL     A             ;4 X (I-1)
CLC
ADC     IND2          ;4 X (I-1) + (J-1)
TAX
LDA     BASIS,X       ;HAAL T(I,J)
```

. Eenvoudige delingen kunnen ook als logische schuivingen naar rechts uitgevoerd worden.

Voorbeeld

Deel de inhoud van de accumulator door 4

```
LSR     A           ;DEEL DOOR 2
LSR     A           ;EN WEER DOOR 2
```

Als u gesigneerde getallen vermenigvuldigt of deelt, moet u in de gaten houden dat u het teken van de waarde scheidt. U moet de logische schuivingen vervangen door rekenkundige verschuivingen, die de waarde van het tekenbit behouden.

. Rekenkundige bewerkingen met schuivingen en optellingen (vermenigvuldiging) of schuivingen en aftrekkingen (deling), kunnen worden gebruikt zoals is beschreven in hoofdstuk 6.

. Zoektabellen kunnen worden gebruikt zoals eerder in dit hoofdstuk werd beschreven.

Hoofdstuk 6 bevat aanvullende voorbeelden van rekenkundige programma's.

LIJSTVERWIJZING[5]

Lijsten kunnen net als arrays worden bewerkt, als de elementen in opeenvolgende adressen zijn opgeborgen. Als de elementen echter in een queue staan of gekoppeld zijn, blijken de beperkingen van de instructieset uit het feit dat

. er geen 16-bits registers zijn;

. indirecte adressering alleen via wijzers (pointers) op pagina 0 is toegestaan;

. er geen werkelijk indirecte adressering anders dan via JMP mogelijk is.

Voorbeelden

1. Haal een adres op dat is opgeborgen in de geheugenlocaties PGNUL en PGNUL+1. Plaats het adres in de geheugenlocaties WIJSL en WIJSH.

```
        LDY     #0          ;INDEX = 0
        LDA     (PGNUL),Y   ;HAAL LWB VAN ADRES
        STA     WIJSL
        INY
        LDA     (PGNUL),Y   ;HAAL HWB VAN ADRES
        STA     WIJSH
```

Hiermee kunt u van het ene naar het andere element in een gelinkte lijst gaan.

2. Haal gegevens uit het adres dat nu in de geheugenlocaties PGNUL en PGNUL+1 staat en verhoog dat adres met 1.

```
        LDY     #0          ;INDEX = 0
        LDA     (PGNUL),Y   ;HAAL GEGEVENS MET BEHULP VAN
                            ;WIJZER
                            ;(POINTER)
        INC     PGNUL       ;VERHOOG WIJZER
        BNE     KLAAR
        INX     PGNUL+1
KLAAR   NOP
```

Deze techniek maakt het mogelijk om het adres in het geheugen te gebruiken als een wijzer naar de volgende beschikbare locatie in een buffer. Uiteraard kunt u de pointer zelf ongewijzigd laten en een buffer-index verhogen. Als die index in geheugenlocatie BUFIND staat, hebben we de routine:

```
        LDY     BUFIND      ;HAAL BUFFER-INDEX
        LDA     (PGNUL),Y   ;HAAL GEGEVEN UIT BUFFER
        INC     BUFIND      ;PAS BUFFER MET 1 AAN
```

3. Berg een adres op dat begint op het adres dat nu in de geheugenlocaties PGNUL en PGNUL+1 staat. Verhoog het adres in de geheugenlocaties PGNUL en PGNUL+1 met 2

```
        LDY     #0          ;INDEX = 0
        LDA     #ADRL       ;BEWAAR LWB VAN ADRES
        STA     (PGNUL),Y
        LDA     #ADRH       ;BEWAAR HWB VAN ADRES
        INY
        STA     (PGNUL),Y
        CLC                 ;VERHOOG WIJZER MET 2
        LDA     PGNUL
        ADC     #2
        STA     PGNUL
        BCC     KLAAR       ;MET OVERDRACHT INDIEN NODIG
        INC     PGNUL+1
KLAAR   NOP
```

Hiermee kunt u een lijst van adressen opbouwen. Een dergelijke lijst zou kunnen worden gebruikt om lint-code te schrijven, waarbij elke routine afsluit met het overstappen naar haar opvolger. De lijst zou ook beginadressen kunnen bevatten van een serie testprocedures, opdrachten, geheugenlocaties of I/O- apparaten, die van de programmeur specifieke functies hebben gekregen. Sommige lijsten zullen uiteraard op pagina 0 moeten worden gezet, om de pre- of postindexeringsmodi van de 6502 te kunnen gebruiken.

ALGEMENE DATA STRUCTUREN

Meerdere algemene datastructuren kunnen behandeld worden met de methoden die we voor array-manipulatie, zoektabellen en lijstbewerking hebben beschreven. De belangrijkste beperkingen in de instructieset zijn dezelfde als degenen die we bij de lijstbewerkingen hebben genoemd.

Voorbeelden.

1. Kettinglijsten. Veronderstel dat we een eerste element (kop) in de wachtlijst hebben, aangewezen door de inhoud van de geheugenlocaties HOOFD en HOOFD+1 op pagina 0. Als er geen elementen in de lijst staan, bevatten HOOFD en HOOFD+1 allebei 0. De eerste twee locaties in elk element bevatten het adres van het volgende element of een 0 als er geen volgend element is.

. Tel het element in de geheugenlocaties PGNUL en PGNUL+1 op bij het hoofd van de lijst.

```
        LDX     PGNUL           ;VERVANG HOOFD, BEWAAR OUDE WAARDE
        LDA     HOOFD
        STX     HOOFD
        PHA
        LDA     PGNUL+1
        LDX     HOOFD+1
        STA     HOOFD+1
        LDY     #0              ;INDEX = 0
        PLA                     ;KOPPEL HET OUDE HOOFD AAN HET NIEUWE
                                ;HOOFD
        STA     (HOOFD),Y
        TXA
        INY
        STA     (HOOFD),Y
```

. Vervang een element uit het hoofd van de rij en zet de nulvlag aan als er geen element beschikbaar is.

```
        LDY     #0              ;HAAL ADRES VAN EERSTE ELEMENT
        LDA     (HOOFD),Y       ;HAAL LWB
        STA     PGNUL
        INY
        LDA     (HEAD),Y
        STA     PGNUL+1
        ORA     PGNUL           ;ZIJN ER ELEMENTEN IN DE RIJ?
        BEQ     DONE            ;NEE, KLAAR (SCHAKEL = 0000)
        LDA     (PGNUL),Y       ;JA, MAAK VOLGEND ELEMENT HET NIEUWE
                                ;HOOFD
        STA     (HOOFD),Y
        DEY
        LDA     (PGNUL),Y
        STA     (HOOFD),Y
        INY                     ;WIS DE NULVLAG DOOR Y 1 TE MAKEN
DONE    NOP
```

Merk op dat we de reeks

```
        LDA     ADR
        ORA     ADR+1
```

kunnen gebruiken om het 16-bits getal in de geheugenlocaties ADR en ADR+1 te controleren. De nulvlag wordt alleen dan aangezet als beide bytes 0 zijn.

2. **Stacks.** Veronderstel dat we een stack-structuur hebben die bestaat uit 8-bits elementen. Het adres van de volgende lege locatie staat in de adressen SWZR en SWZR+1 op pagina 0. Het laagste adres dat de stack kan gebruiken is LAAG en het hoogste adres is HOOG.

. Wis de overdrachtvlag en stop als de stack overloopt. Berg anders de accumulator op de stack op en verhoog de stack-wijzer met 1. De stack loopt over als deze buiten zijn toegestane gebied komt.

```
        LDA     #HOOGL      ;STACK-WIJZER GROTER DAN HOOG?
        CMP     SWZR
        LDA     #HOOGH
        SBC     SWZR+1
        BCC     KLAAR       ;JA, WIS DE OVERDRACHTVLAG EN STOP
                            ; (OVERGELOPEN)
        LDY     #0          ;NEE, BERG ACCUMULATOR OP STACK OP
        STA     (SWZR),Y
        INC     SWZR        ;VERHOOG STACK-WIJZER
        BNE     KLAAR
        INC     SWZR+1
KLAAR   NOP
```

. Zet de overdrachtvlag aan en stop als de stack 'onderloopt'. Verlaag anders de stack wijzer met 1 en haal de accumulator van de stack. Onderlopen betekent dat er niets meer op de stack staat.

```
        LDA     #LAAGL      ;STACK-WIJZER OP OF ONDER LAAG?
        CMP     SWZR
        LDA     #LAAGH
        SBC     SWZR+1
        BCS     KLAAR       ;JA, ZET DE OVERDRACHTVLAG AAN EN STOP
                            ; (ONDERLOOP)
        LDA     SWZR        ;NEE, VERLAAG DE STACK-WIJZER
        BNE     GEENLEEN
        DEC     SWZR+1
GEENLEEN DEC    SWZR
        LDY     #0          ;HAAL ACCUMULATOR VAN STACK
        LDA     (SWZR),Y
KLAAR   NOP
```

TECHNIEKEN OM PARAMETERS DOOR TE GEVEN

De meest gebruikelijke methoden om met de 6502 microprocessor parameters door te geven, zijn

1. **In registers.** Drie 8-bits registers zijn beschikbaar (A, X en Y). Deze methode kan in eenvoudige gevallen worden gebruikt, maar mist algemeenheid en kan slechts een beperkt aantal

parameters doorgeven. De programmeur moet het normale gebruik van de registers weten als hij parameters gaat toekennen. Anders gezegd,

. De accumulator is de voor de hand liggende plaats om een 8-bits parameter te stoppen.

. Indexregister X is de voor hand liggende plaats om een index te stoppen, omdat het het meest toegankelijke register is en ook de meeste instructies voor adressering heeft. Indexregister X wordt ook gebruikt bij preindexering (indirect geindexeerde adressering).

. Indexregister Y wordt gebruikt bij postindexering (indirect geindexeerde adressering).

Deze wijze van benaderen is meer dan een keer te gebruiken, mits de routines die interrupts afhandelen de registers bewaren en herstellen.

2. In een toegekend deel van het geheugen. De makkelijkste manier om deze techniek toe te passen is door het beginadres van het toegekende deel van het geheugen in twee geheugenlocaties op pagina 0 te zetten. De aanroepende routine moet de parameters in het geheugen opbergen en het beginadres in twee geheugenlocaties laden voordat de controle aan de subroutine wordt gegeven. Deze techniek is algemeen en kan elke hoeveelheid parameters verwerken, maar vereist een grote mate van controle en beheer. Als u verschillende geheugenlocaties voor iedere routine reserveert, creeert u eigenlijk uw eigen stack. Als je een algemeen deel van het geheugen gebruikt, verspeelt u de mogelijkheid van hergebruik. Bij deze methode is de programmeur verantwoordelijk voor: het toekennen van geheugengebieden; het voorkomen dat verschillende routines door elkaar gaan werken; en het bewaren en herstellen van pointers die nodig zijn om weer naar routines terug te springen na het aanroepen van een subroutine of een interrupt. De extra geheugenlocaties op pagina 0 dienen als registers te worden gebruikt.

3. In het programmageheugen, meteen na het aanroepen van de subroutine. Als u deze methode gebruikt, moet u op het volgende letten:

. Het beginadres van het geheugengebied min 1 staat op de top van de stack. Dit houdt in dat het beginadres het normale terugkeeradres is. Het terugkeeradres is 1 groter dan het adres dat de JSR-instructie van de 6502 op de stack bewaart. U kunt het beginadres naar de geheugenlocaties TERADR en TERADR+1 op pagina 0 verplaatsen met de volgende routine:

```
        PLA                 ;HAAL LWB VAN TERUGKEERADRES
        STA     TERADR
        PLA                 ;HAAL HWB VAN TERUGKEERADRES
        STA     TERADR+1
        INC     TERADR      ;TEL 1 BIJ TERUGKEERADRES OP
        BNE     KLAAR
        INC     TERADR+1
KLAAR   NOP
```

Nu kunnen we de parameters via het indirecte adres bereiken. Dat wil zeggen, u kunt de accumulator met de eerste parameter laden door de volgende routine te gebruiken:

```
        LDY     #0              ;INDEX = 0
        LDA     (TERADR),Y      ;LAAD EERSTE PARAMETER
```

Een voor de hand liggend alternatief is dat we het terugkeeradres ongewijzigd laten en de index met 1 beginnen. Dat zou er dan zo uitzien:

```
        PLA                     ;HAAL LWB VAN TERUGKEERADRES
        STA     TERADR
        PLA                     ;HAAL HWB VAN TERUGKEERADRES
        STA     TERADR+1
```

Nu zouden we de accumulator met de eerste parameter kunnen laden door

```
        LDY     #1              ;INDEX = 1
        LDA     (TERADR),Y      ;LAAD EERSTE PARAMETER
```

. Alle parameters moeten voor het aanroepen van een subroutine vaststaan, omdat het programmageheugen typisch "Read Only" is.

De subroutine moet het feitelijke terugkeeradres (het adres van de laatste byte in het parametergebied) berekenen en dit op de stack zetten, voordat een RTS-instructie wordt gegeven.

Voorbeeld

Veronderstel dat subroutine SUBR een 8-bits en een 16-bits parameter nodig heeft. Maak een hoofdprogramma dat SUBR aanroept en de vereiste parameters bevat. Maak ook het inleidende deel van de subroutine dat de parameters ophaalt, het 8-bits voorwerp in de accumulator zet, het 16-bits voorwerp in de geheugenlocaties PGNUL en PGNUL+1 opbergt en het juiste terugkeeradres boven in de stack plaatst.

Subroutine-aanroep

```
        JSR     SUBR            ;VOER SUBROUTINE UIT
        .BYTE   PAR8            ;8-BITS PARAMETER
        .WOORD  PAR16           ;16-BITS PARAMETER
         ... volgende instructie ...
```

```
Subroutine
SUBR    PLA                     ;HAAL LWB VAN PARAMETERADRES
        STA     TERADR
        PLA                     ;HAAL HWB VAN PARAMETERADRES
        STA     TERADR+1
        LDY     #1              ;GA NAAR EERSTE PARAMETER
        LDA     (TERADR),Y      ;HAAL EERSTE PARAMETER
        TAX
        INY
        LDA     (TERADR),Y      ;HAAL LWB VAN 16-BITS PARAMETER
        STA     PGNUL
        INY
        LDA     (TERADR),Y      ;HAAL HWB VAN 16-BITS PARAMETER
        STA     PGNUL+1
        LDA     TERADR          ;BEREKEN FEITELIJK TERUGKEERADRES
        CLC
        ADC     #3
        TAY
        BCC     BRGHWB
        INC     TERADR+1
BRGHWB  LDA     TERADR+1        ;PLAATS TERUGKEERADRES BOVEN OP STACK
        PHA
```

```
        TYA
        PHA
```

De inleidende instructies van de subroutine halen het terugkeeradres van de top van de stack (waar de JSR het had geplaatst) en bergt het op in de geheugenlocaties TERADR en TERADR+1. In feite bevat het terugkeeradres geen instructie, maar de eerste parameter. Onthoud dat JSR het terugkeeradres minus 1 bewaart. Dat is waarom de index op 1 wordt geinitialiseerd, in plaats van op 0. Door ten slotte 3 bij het terugkeeradres op te tellen en het resultaat op de stack te zetten, keren we, na een RTS, terug op de plek na de parameters.
Deze techniek staat parameterlijsten van onbeperkte lengte toe. Het ophalen van de parameters en het aanpassen van het terugkeeradres is op z'n zachtst gezegd lastig; het wordt een langer en trager proces naarmate het aantal parameters stijgt.

4. **Op de stack.** Als u deze techniek gebruikt, moet u op het volgende letten:

. JSR bewaart het terugkeeradres boven op de stack. De parameters die de aanroepende routine op de stack heeft gezet, beginnen op adres 01ss+3, waarbij ss de inhoud van de stack-wijzer is. Het 16-bits terugkeeradres bezet de bovenste twee locaties en de stack-wijzer zelf wijst altijd naar het eerstvolgende lege adres, niet het laatst bezette. Voordat de subroutine de parameters kan ophalen, moet deze eerst het terugkeeradres van de stack verwijderen en ergens opbergen.

. De subroutine kan alleen door middel van de instructie TSX de waarde van de stack-wijzer bepalen. Nadat TSX is uitgevoerd, kunt u naar de top van de stack gaan door met X te indexeren vanaf het basisadres 0101_{16}. De extra offset (afstand) van 1 is nodig omdat de top van de stack leeg is.

. Het aanroepende programma moet de parameters eerst op de stack zetten, voordat het de subroutine aanroept.

. Ruimte vrijmaken op de stack is in het gunstigste geval moeilijk. Als u de stack-wijzer met NRESLT wilt verlagen, zijn er twee methoden:

```
        TSX                 ;VERPLAATS STACK-WIJZER NAAR A VIA X
        TXA
        SEC                 ;TREK NRESLT VAN WIJZER AF
        SBC     #NRESLT
        TAX                 ;ZET VERSCHIL IN STACK-WIJZER
        TXS
of
        LDX     #NRESLT     ;TELLER = NRESLT
PLAATSB PHA                 ;VERLAAG STACK-WIJZER MET 1
        DEX
        BNE     PLAATSB
```

Beide technieken zetten NRESLT lege locaties boven op de stack, zoals in figuur 1-5 is te zien. Het is uiteraard makkelijker en sneller om, als NRESLT 1 of 2 is, het gewenste aantal keren een PHA uit te voeren. Deze technieken kunnen ook worden gebruikt om ruimte op de stack te creeren voor tijdelijke opslag.

Voorbeeld

Veronderstel dat subroutine SUBR een 8-bits en een 16-bits parameter nodig heeft en dat ze twee 8-bits resultaten produceert. Toon de aanroep van SUBR, het verwijderen van het terugkeeradres

van de stack en het wissen van de stack na de terugkeer. Figuur 1-6 toont de stack zoals hij er eerst uitziet, na het aanroepen van de subroutine en bij het einde. Als u de stack altijd voor het doorgeven van parameters gebruikt, houdt u de parameters in de top van de stack over het algemeen op volgorde. Dan hoeft u de parameters niet op te bergen of ruimte te creeren voor de resultaten, want die vervangen dan een deel van de oorspronkelijke parameters.U zult echter wel ruimte op stack moeten vrijmaken om de algemeenheid te bewaren en hergebruik toe te staan.

Aanroepende programma

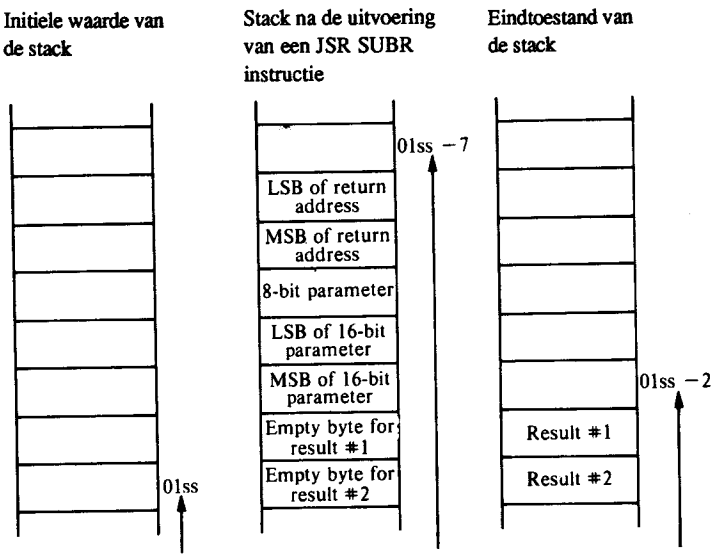
```
        TSX                     ;LAAT PLAATS OP STACK VRIJ VOOR
                                ;RESULTATEN
        TXA                     ;EEN ALGEMENE MANIER OM DE SW AAN TE
                                ;PASSEN
        CLC
        ADC     #2
        TAX
        TXS
        LDA     #PAR16H         ;VERPLAATS 16-BITS PARAMETER NAAR
                                ;STACK
        PHA
        LDA     #PAR16L
        PHA
        LDA     #PAR8           ;VERPLAATS 8-BITS PARAMETER NAAR
                                ;STACK
        JSR     SUBR            ;VOER SUBROUTINE UIT
        TSX                     ;WIS PARAMETERS VAN STACK
        TXA
        CLC
        ADC     #3
        TAX
        TXS                     ;RESULTAAT STAAT NU BOVEN IN STACK
```

Subroutine

```
SUBR    PLA                     ;VERWIJDER TERUGKEERADRES VAN STACK
        STA     TERADR
        PLA
        STA     TERADR+1
```

Geen waarde wordt in de locaties gezet.
De initiele waarde van de stack-wijzer is ss.

Figuur 1-5: De stack voor en na het toewijzen van NRRESULT lege locaties voor het resultaat



De initiele waarde van de stack-wijzer is ss

Figuur 1-6: Het effect van een subroutine op de stack

EENVOUDIGE INPUT/OUTPUT

Door het gebruik van elke geheugenadres en elke instructie die aan het geheugen refereert, kan eenvoudige input/output worden uitgevoerd. De meest gebruikte instructies zijn:

. LDA (laad accumulator) verplaatst 8 bits data van een input-poort naar de accumulator.

. STA (store accumulator, berg accumulator op) verplaatst 8 bits data van de accumulator naar een output-poort.

Andere instructies kunnen de input-handeling met rekenkundige of logische bewerking combineren. Typische voorbeelden daarvan zijn:

. AND voert een logische 'AND' uit van de inhoud van de accumulator met de data van de input-poort.

. BIT voert een logische 'AND' uit van de inhoud van de accumulator met de data van de input-poort, maar bergt het resultaat daarvan nergens op. Echter, het laadt wel de negatiefvlag met bit 7 en de overschotvlag met bit 6 van de inputpoort, ongeacht de inhoud van de accumulator.

. CMP trekt de data van een input-poort af van de inhoud van de accumulator, waarbij de vlaggen gezet worden, maar de inhoud van de accumulator ongewijzigd blijft.

Instructies die met data in het geheugen werken, kunnen ook voor input en output worden gebruikt. Aangezien deze instructies geheugen zowel lezen als schrijven, kan het effect op input- en output-poorten moeilijk te bepalen zijn. Vergeet niet dat input-poorten in het algemeen niet geschreven en output-poorten in het algemeen niet gelezen kunnen worden. De meest gebruikte instructies zijn:

. ASL verschuift de data naar links, en schuift zodoende bit 7 in de Overdracht voor mogelijke seriele input.

. DEC verlaagt de data. Naast andere effecten, inverteert dit bit 0.

. INC verhoogt de data. Naast andere effecten, inverteert dit bit 0.

. LSR schuift de data naar rechts, waarbij bit 0 in de overdrachtvlag wordt geschoven voor mogelijke seriele input.

. ROR roteert de data naar rechts, waarbij de oude overdrachtvlag naar bit 7 en bit 0 naar de Carry geschoven wordt.

. ROL roteert de data naar links, waarbij de oude overdrachtvlag naar bit 0 en bit 7 naar de Carry geschoven wordt.

De effecten van deze instructies op een input-poort, zijn gelijksoortig aan hun effecten op een ROM-locatie. De microprocessor kan de data lezen, er naar handelen en de vlaggen zetten, maar het resultaat niet terug in het geheugen plaatsen. De effecten op een output-poort zijn nog vreemder, tenzij de poort is geschakeld en gebufferd. Als dit niet het geval is, zijn de data die de processor leest systeem-afhankelijk en houden ze geen verband met wat er het laatst was neergezet.

Voorbeelden.

1. Zorg voor een 8-bits input vanuit de input-poort die aan geheugenadres $B000_{16}$ is toegekend.

```
LDA     $B000       ;INPUT-DATA
```

2. Zorg voor een 8-bits output naar de output-poort van geheugenadres $3A5E_{16}$.

```
STA     $3A5E       ;OUTPUT-DATA
```

3. Zet de nulvlag aan als bit 5 van de input-poort van geheugenadres $75D0_{16}$ de waarde 0 heeft.

```
LDA     #%00100000  ;HAAL MASKER
AND     $75D0       ;ZET VLAG AAN INDIEN BIT 5 NUL IS
```

We kunnen hiervoor ook gebruiken

```
LDA     #%00100000  ;HAAL MASKER
BIT     $75D0       ;ZET VLAG AAN INDIEN BIT 5 NUL IS
```

Als de bit positie waarover we informatie willen hebben, bit 6 is, dan kunnen we volstaan met

```
BIT     $75D0
```

om de overschotvlag naar de bit-waarde te zetten.

4. Zet de nulvlag aan als de inhoud van de input-poort van geheugenadres 1700_{16} gelijk is aan $1B_{16}$.

```
LDA     #$1B
CMP     $1700
```

5. Laad de overdrachtvlag met de waarde van bit 7 van de input-poort die aan geheugenadres $33A5_{16}$ is toegekend.

```
ASL     $33A5
```

Merk op dat deze instructie de inhoud van geheugenlocatie $33A5_{16}$ niet verandert, tenzij deze locatie gebufferd en geschakeld is. Als er bij voorbeeld 8 eenvoudige schakelaars direct aan de poort zijn bevestigd, dan zal de instructie er zeker geen effect op hebben of de schakelaars geopend, dan wel gesloten zijn.

6. Plaats een logische 1 in bit 0 van de output-poort van geheugenadres $B070_{16}$.

```
LDA     $B070
ORA     #%00000001
STA     $B070
```

Als geen van de andere bits van adres $B070_{16}$ aangesloten zijn, dan kunnen we volstaan met

```
SEC
ROL     $B070
```

Als we weten dat bit 0 van adres $B070_{16}$ op het ogenblik een logische 0 is, dan kunnen we de handeling af met een enkele instructie

```
INC     $B070
```

Al deze alternatieven zullen vreemde effecten hebben als geheugenadres $B070_{16}$ niet gelezen kan worden. De eerste twee zullen bit 0 zeker een logische 1 maken, maar hun effect op de andere bits is onzeker. Het resultaat van het derde alternatief zou een compleet raadsel zijn, omdat we geen idee hebben wat er verhoogd wordt. We kunnen de onzekerheid voorkomen door een kopie van de inhoud in RAM-locatie TEMP te bewaren. Nu kunnen we met de kopie werken:

```
LDA     TEMP          ;HAAL DE KOPIE VAN DE OUTPUT INHOUD
ORA     #%00000001    ;ZET BIT 0 AAN
STA     $B070         ;ZET DE NIEUWE INHOUD TERUG
STA     TEMP          ;EN BEWAAR EEN KOPIE
```

LOGISCHE EN FYSIEKE APPARATEN

Een manier om input/output (I/O) apparaten op nummer te selecteren, is met behulp van een I/O-apparaattabel. Een I/O-apparaattabel geeft het echte I/O-adres (het fysieke apparaat) voor het apparaatnummer (het logische apparaat) waaraan een programma refereert. Met behulp van deze methode kan een programma dat in een hogere taal is geschreven, de input van apparaat nummer 2 krijgen en de output naar apparaat nummer 5 sturen. Om te kunnen testen, kan de systeembeheerder de apparaten nummer 2 en 5, de in- en output-poorten van zijn terminal toekennen. Voor normale bediening kan apparaat nummer 2 een analoge input en apparaat nummer 5 de printer zijn. Als het systeem op afstand bediend moet worden, kunnen apparaten 2 en 5 communicatie-onderdelen zijn.
Een manier om in het onderscheid tussen logische en fysieke apparaten te voorzien, is het gebruik van de geindexeerde indirecte adressering of de preindexeringsmodus van de 6502. Deze modus gaat ervan uit dat de apparatentabel op pagina 0 staat en dat deze wordt benaderd met een index in register X. Als we een apparaatnummer in geheugenlocatie APPNO hebben staan, kunnen we de volgende programma's gebruiken:

. Laad de accumulator met het apparaatnummer, dat in de geheugenlocatie APPNO staat.

```
LDA     APPNO         ;HAAL APPARAATNUMMER
ASL     A             ;VERDUBBEL HET OM APPARAATADRESSEN TE
                      ;VERWERKEN
TAX
LDA     (APPTAB,X)    ;HAAL DATA UIT APPARAAT
```

. Stop de waarde van de accumulator in het apparaatnummer, gegeven door de inhoud van geheugenlocatie APPNO.

```
PHA                   ;BEWAAR DE DATA
LDA     APPNO         ;HAAL APPARAATNUMMER
ASL     A             ;VERDUBBEL HET OM APPARAATADRESSEN TE
                      ;VERWERKEN
TAX
PLA
STA     (APPTAB,X)    ;STUUR DATA NAAR APPARAAT
```

In beide gevallen gaan we ervan uit dat de I/O-apparaattabel begint bij adres APPTAB (op pagina 0) en bestaat uit 2-bytes grote adressen. Let op dat de 6502 wel de goede adreseringsmethode kent, maar geen foutmeldingen geeft als de programmeur de modus foutief gebruikt door vreemde adressen aan te spreken of voorbij het einde van pagina 0 indexeert. Bij echte toepassingen (zie hoofdstuk 10) zal de apparaattabel waarschijnlijk beginadressen van sturings-programma's bevatten en niet de feitelijke adressen van de apparaten.

STATUS EN CONTROLE

U kunt status- en controle signalen net zo behandelen als andere gegevens. Het enige bijzondere probleem is dat de processor mogelijk niet in staat is output-poorten te lezen. In dat geval moet u kopieen van de gegevens die naar die poorten worden gestuurd (in RAM) bewaren.

Voorbeelden

1. Vertak naar adres BEST als bit 3 van de input-poort van geheugenadres A100 op 1 staat.

```
LDA     $A100         ;HAAL INPUT-DATA
AND     #%00001000    ;MASKEER BIT 3
BNE     BEST
```

2. Vertak naar adres BEST als de bits 4, 5 en 6 van de input-poort van adres STAT gelijk zijn aan 5 (101 binair).

```
LDA     STAT          ;HAAL STATUS
AND     #%01110000    ;MASKEER BITS 4, 5 EN 6
CMP     #%01010000    ;IS STATUS VELD 5?
BEQ     BEST          ;JA, VERTAK
```

3. Zet bit 5 van adres CNTL aan.

```
LDA     CNTL          ;HAAL HUIDIGE DATA UIT POORT
ORA     #%00100000    ;ZET BIT 5 AAN
STA     CNTL          ;EN ZET NIEUWE DATA TERUG
```

Als het adres CNTL niet goed gelezen kan worden, kunnen we een kopie van de inhoud in geheugenadres TEMP gebruiken.

```
LDA     TEMP          ;HAAL HUIDIGE DATA UIT POORT
ORA     #%00100000    ;ZET BIT 5 AAN
STA     CNTL          ;ZET NIEUWE WAARDE TERUG
STA     TEMP          ;PAS KOPIE VAN DATA AAN
```

U moet de kopie aanpassen iedere keer nadat u de data heeft veranderd.

4. Zet bits 2, 3 en 4 van adres CNTL op 6 (110 binair)

```
LDA     CNTL          ;HAAL HUIDIGE DATA UIT POORT
AND     #%1110011     ;WIS BITS 2, 3 EN 4
ORA     #%00011000    ;ZET CONTROLEVELD OP 6
```

```
STA     CNTL          ;EN ZET NIEUWE WAARDE TERUG
```

We kunnen een kopie in het geheugenadres TEMP gebruiken als, net als bij voorbeeld 3, adres CNTL niet goed gelezen kan worden.

```
LDA     TEMP          ;HAAL HUIDIGE WAARDE UIT POORT
AND     #%11100011    ;WIS BITS 2, 3 EN 4
ORA     #%00011000    ;ZET CONTROLEVELD OP 6
STA     CNTL          ;PAS POORT AAN
STA     TEMP          ;PAS KOPIE AAN
```

Door kopieen van data in het geheugen te bewaren (of waarden te gebruiken die in een geschakelde en gebufferde output-poort staan) kunt u delen van de gegevens veranderen zonder daardoor ook andere delen te beinvloeden die mogelijk een totaal verschillende functie hebben. U zoudt zo de stand kunnen veranderen van een indicatielamp (een die bij voorbeeld aangeeft of de computer direct of van afstand wordt gebruikt) zonder de lampjes die aan dezelfde poort zitten van stand te veranderen. Op een soortgelijke manier zoudt u een enkele controle-lijn kunnen veranderen (bij voorbeeld een lijn die aangeeft of de stroom in een positieve of negatieve X-baan loopt) zonder de andere controle-lijnen aan dezelfde poort te veranderen.

RANDAPPARATUUR-CHIPS

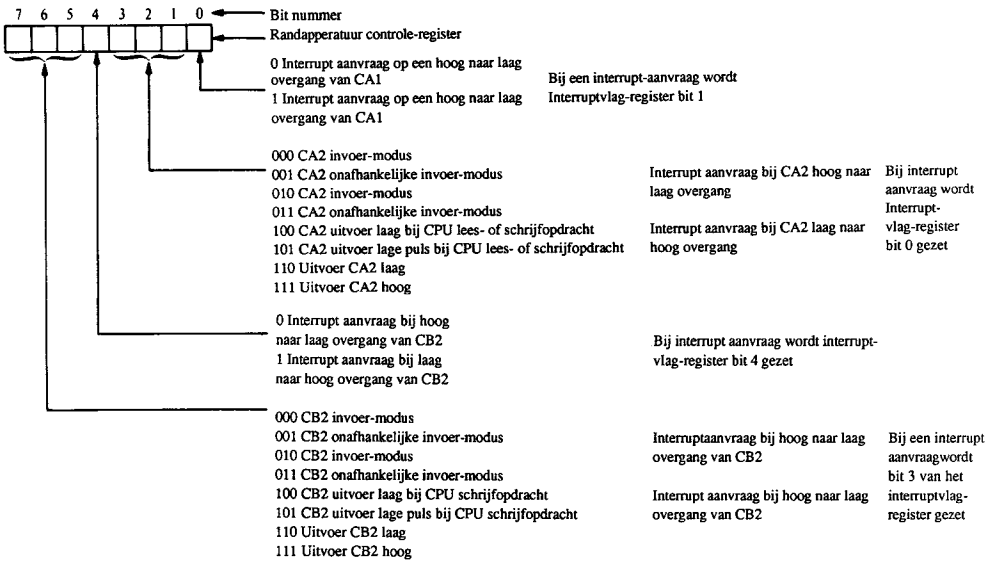
De belangrijkste randapparatuur-chips in 6502 computers zijn de 6520 en de 6522 parallelle interfaces (bekend als de Peripheral Interface Adapter of PIA, respectievelijk de Versatile Interface Adapter of VIA), de 6551 en 6850 seriele interfaces (bekend als de Asynchrone Communicatie Interface Adapters of ACIA's) en de 6530 en 6532 multifunctie-apparaten (bekend als ROM-I/O-timers, RAM-I/O-timers of ROM-RAM-I/O-timers, afgekort als RIOT of RRIOT en soms 'combo' chips genoemd). Al deze chips kunnen een reeks functies verrichten, net als de microprocessor zelf. Uiteraard kunnen deze randapparatuur-chips minder veelsoortige functies verrichten dan de processoren en is het aantal functies beperkt. De filosofie achter deze programmeerbare chips is dat elk vele bruikbare circuits bevat; de ontwerper kiest die welke hij of zij wil gebruiken door een of meerdere selectie-codes naar de controle-registers te sturen. De voordelen van programmeerbare chips bestaan hierin dat een enkele kaart die dergelijke chips bevat, vele handelingen kan verrichten. Bovendien kunnen correcties worden aangebracht door selectie-codes te veranderen, zodat niet de hele kaart opnieuw ontworpen hoeft te worden.
Het nadeel van programmeerbare chips is het gebrek aan standaarden en de moeilijkheid om te bepalen hoe specifieke chips werken.
Hoofdstuk 10 bevat typische initialisatieroutines voor de 6520, 6522, 6551, 6850, 6530 en 6532 chips. Deze apparaten worden ook in detail besproken in het Osborne 4- en 8-Bits Microprocessor Handboek. Wij zullen hier een beknopt overzicht geven van de 6522, omdat dit het meeste gebruikte apparaat is. 6522 apparaten worden onder meer gebruikt in de Rockwell AIM-65, Synertek SYM-1, Apple, en andere populaire microcomputers alsook in uitbreidingskaarten.

6522 Parallelle Interface (Versatile Interface Adapter)

Een VIA bevat twee 8-bits parallelle I/O-poorten (A en B), vier status- en controle-lijnen (CA1, CACA2, CB1 EN CB2 - twee voor elk van de twee poorten), twee 16-bits tellers/timers (timer 1 en timer 2), een 8-bits schuifregister en interrupt-logica. Elke VIA neemt 16 geheugenadressen in beslag. De RS (Register Selectie) lijnen kiezen de verschillende interne registers, zoals beschreven

in tabel 1-12. De manier waaarop een VIA werkt, wordt bepaald door de waarden die het programma in vier registers stopt.

. Data-richtingregister A (Data Direction Register A - DDRA) bepaalt of de aansluitingen op poort A inputs (0-en) of outputs (1-en) zijn. Een Data-richtingregister bepaalt slechts de richting waarin de gegevens stromen; het is te vergelijken met en pijl die aangeeft in welke richting het verkeer in een straat mag rijden. Het data-richtingregister heeft geen invloed op de stromende data, noch op het aantal keren dat deze veranderen; het kan alleen de richting veranderen. Elke aansluiting in de I/O-poort heeft een corresponderend bit in het data-richtingregister. Daarom kan iedere aansluiting afzonderlijk als een in- of output geselecteerd worden. De meeste gangbare keuze is om een complete 8-bits I/O-poort input of output te maken door 0-en of 1-en in alle 8 bits van hjet data richting register te plaatsen.



Figuur 1-7: Bit toewijzingen voor het controleregister van de 6522 VIA

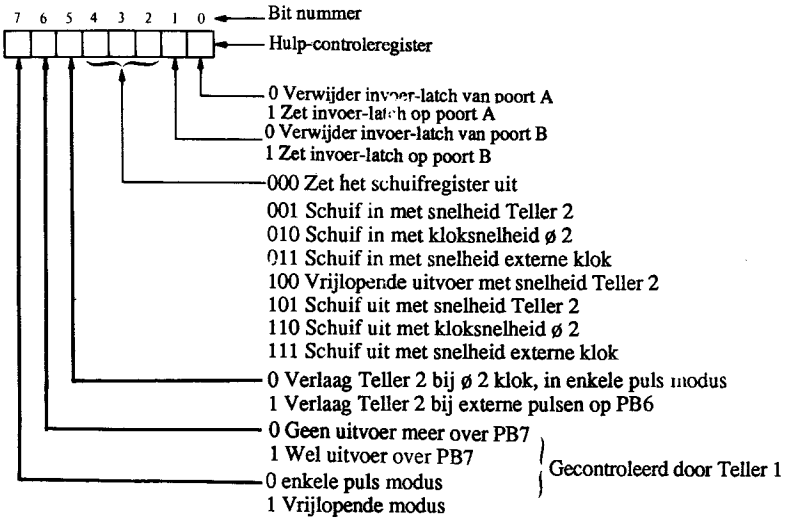
. Data-richtingregister B (Data Direction Register B - DDRB) bepaalt op dezelfde wijze als boven of de aansluitingen van poort B in- dan wel output zijn.

. Het randapparatuur-controleregister (Peripheral Control Register - PCR) bepaalt hoe de handshaking (communicatie tussen zendend en ontvangend apparaat) controle-lijnen (CA1, CB1, CA2 en CB2) werken. Figuur 1-7 toont de bit-functies van dit register. We zullen de werkingen van deze bits en het gebruik in algemene toepassingen in het kort beschrijven.

. Het hulp-controleregister (Auxiliary Control Register - ACR) bepaalt hoe de timers en schuifregisters werken en of de input-datapoorten gelatcht zijn (een latch is voor te stellen als een luik, dat een signaal kan vasthouden). Figuur 1-8 toont de bit-functies van dit register. We zullen ook de werking van deze bits en de algemene toepassingen in het kort beschrijven.

Tabel 1-12: Adressering van de Interne registers van de 6522 VIA

	geselecteerde lijnen				
Label	**RS3**	**RS2**	**RS1**	**RS0**	**Geadresseerde locatie**
DEV	0	0	0	0	Uitvoer-register voor I/O poort B
DEV+1	0	0	0	1	Uitvoer-register voor I/O poort A met aansluitings-bevestiging
DEV+2	0	0	1	0	I/O poort B datarichtings-register
DEV+3	0	0	1	1	I/O poort A datarichtings-register
DEV+4	0	1	0	0	Lees Tijdklok 1 Teller laagstwaardige byte Schrijf naar Tijdklok 1 Latch hoogstwaardige byte
DEV+5	0	1	0	1	Lees Tijdklok 1 Teller laagstwaardige byte Schrijf naar Tijdklok 1 Latch hoogstwaardige byte en initiele teller
DEV 6	0	1	1	0	Toegang tot Tijdklok 1 Latch laagstwaardige byte
DEV+7	0	1	1	0	Toegang tot Tijdklok 1 Latch hoogstwaardige byte
DEV+8	1	0	0	0	Lees laagstwaardige byte van Tijdklok 2 en zet de teller-interrupt terug. Schrijf naar het laagstwaardige byte van Tijdklok 2 maar zet de teller-interrupt niet terug
DEV+9	1	0	0	1	Toegang tot Tijdklok 2. Zet de teller-interrupt terug bij een schrijfopdracht
DEV+A	1	0	1	0	Seriele I/O schuif-register
DEV+B	1	0	1	1	Hulp controle-register
DEV+C	1	1	0	0	Randapperatuur controle-register
DEV+D	1	1	0	1	Interrupt vlag-register
DEV+E	1	1	1	0	Interrupt aanzet-register
DEV+F	1	1	1	1	Uitvoer register voor I/O poort A, zonder aansluitings-bevestiging



Figuur 1-8: Bit toewijzingen voor het hulp controleregister van de 6522 VIA

Om een VIA goed te initialiseren, moeten we weten wat de beginstand is. Een reset wist alle VIA-registers en maakt zodoende van alle data- en controle- lijnen inputs, schakelt alle latches, interrupts en andere functies uit en wist alle statusbits.
De data-richtingregisters zijn makkelijk te initialiseren. Typische routines zijn:

- Maak van poort A een input-poort.

```
LDA     #0
STA     DDRA
```

- Maak van poort B een output-poort.

```
LDA     #%11111111
STA     DDRB
```

- Maak van de bits 0 to 3 van poort A input, aansluitingen 4 tot 7 output.

```
LDA     #%11110000
STA     DDRA
```

- Maak van bit 0 van poort B input, bits 1 tot 7 output.

```
LDA     #%11111110
STA     DDRB
```

Bit 0 zou zo bij voorbeeld een seriele input-lijn kunnen zijn.
Het randapperatuur-controleregister (PCR) is lastiger te initialiseren. We zullen de functies van de controle-lijnen kort beschrijven, voor we enkele voorbeelden laten zien.

Controle-lijnen CA1, CA2, CB1 en CB2 zijn in principe bedoeld voor handshaking-signalen. In een handshake geeft de zender aan dat er nieuwe data klaarstaan om overgestuurd te worden door middel van een signaal op een seriele lijn; dit signaal vertelt de ontvanger dat er nieuwe data beschikbaar is op de parallelle lijnen. VALID DATA, DATA READY en DATA AVAILABLE (Data Klaar) zijn de meest gangbare namen voor deze seriele lijn. De ontvanger moet na ontvangst van dit signaal de data ontvangen en aangeven dat dit is gebeurd met een signaal op een andere seriele lijn. Dit signaal vertelt de zender dat de meest recente parallelle data zijn ontvangen en dat een volgende zending kan beginnen. Gangbare namen voor die andere seriele lijn zijn DATA ACCEPTED (Data Ontvangen), PERIPHERAL READY (randapparatuur klaar), BUFFER EMPTY (Buffer Leeg) en DATA ACKNOWLEDGE (Data Bevestigd).
Typische voorbeelden van zo'n procedure zijn:

. **Invoer vanaf het toetsenbord.** Als de bediener een toets indrukt, geeft het toetsenbord een parallelle code die correspondeert met de ingedrukte toets en stuurt een signaal over de DATA READY of VALID DATA (Data Klaar) lijn. De computer moet dan vaststellen dat het signaal is gestuurd, de data lezen en een signaal over de DATA ACCEPTED (Data Ontvangen) lijn sturen.

. **Uitvoer naar een printer.** De printer vertelt de computer dat hij klaar is om te ontvangen door een signaal over de BUFFER EMPTY of PERIPHERAL READY (Buffer Leeg of Randapparatuur Klaar) lijn te sturen. Peripheral Ready is gewoon een geinverteerde DATA ACCEPTED, omdat de randapparatuur natuurlijk nooit klaar kan zijn, zolang zij de laatste data nog niet heeft geaccepteerd. De computer moet vaststellen dat de printer klaar is, de data sturen en een signaal op de DATA READY lijn zetten om aan te geven dat nieuwe data beschikbaar zijn. Input en output zijn soms min of meer een spiegelbeeld van elkaar. In het geval van de invoer is de randapparatuur de zender en is de computer de ontvanger. In het geval van de uitvoer is de computer de zender en de randapparatuur de ontvanger.
Daarom moet een chip die handshaking verwerkt, de volgende functies hebben:

. Hij moet de juiste signalen op de DATA READY, of PERIPHERAL READY lijnen herkennen.

. Als er een signaal is verstuurd, moet hij dit in een voor de computer verwerkbare vorm doorgeven.

. Hij moet antwoord toestaan. De computer moet DATA ACCEPTED kunnen doorgeven aan input-randapparatuur en DATA READY aan output-randapparatuur.

Er is een aantal voor de hand liggende variaties die de chip ook moet toestaan. Zoals deze situaties:

. Het actieve signaal kan zowel een hoog-laag als een laag-hoog richting hebben. Als de chip geen van beide toestaat, hebben we extra inverterende sluizen nodig, omdat beide richtingen algemeen gebruikt worden.

. De respons kan als hoog-laag of laag-hoog signaal worden verwacht. Het is zelfs mogelijk dat er een korte puls of een lang signaal wordt verwacht, welke aanhoudt tot de randapparatuur met de volgende handeling begint.

Het is gebleken dat de handshake-chip tegen nauwelijks extra kosten voor nog meer gemak kan zorgen:

. Hij kan de signalen op de DATA READY of PERIPHERAL READY lijnen latchen, zodat deze worden vastgehouden tot de computer er klaar voor is. De computer hoeft zo niet constant de statuslijnen in de gaten te houden om te voorkomen dat hij een signaal mist.

. Hij kan de status-latches automatisch wissen als een input-poort gelezen of een output-poort beschreven is. Daarmee zet hij ze klaar voor de volgende handeling.

. Hij kan volledig automatisch de respons produceren als een input-poort gelezen of een output-poort beschreven is. Hierdoor hoeven geen aparte instructies gegeven te worden. Het nadeel van deze automatische modus is dat, hoe flexibel de ontwerpers hem ook maken, hij nooit aan alle applicaties kan voldoen. Daarom zijn de meeste chips ook voorzien van een handbediende modus, waarbij het programma de controle over de respons behoudt.

. In die gevallen waar de randapparatuur slechts uit schakelaars of lampjes bestaat en dus geen controle of statuslijnen nodig heeft, zou de chip in een onafhankelijk gebruik van de statuslijnen moeten voorzien. De ontwerper kan deze lijnen (die anders nutteloos zouden zijn) gebruiken voor drempel-detectie of klok-inputs. De ontwerper wil dat de status- en controle-lijnen in die gevallen volledig onafhankelijk zijn van de gebeurtenissen op de parallelle poort. Het wissen van de latches en het sturen van responses mag niet automatisch gebeuren. Deze modus is bekend als de onafhankelijke modus (independent mode).
Het randapperatuur-controleregister (PCR) van de 6522 heeft al deze functies. De bits 0 tot 3 regelen de werking van poort B en haar controle- signalen; bits 4 tot 7 regelen de werking van poort A en haar controle- signalen. De status-indicatoren staan in het interrupt-vlagregister (zie figuur 1-9). We kunnen de bits in het controleregister als volgt beschrijven:
. Bit 0 (voor poort A) en bit 4 (voor poort B) bepalen of het actieve signaal op controle-lijn 1 hoog-laag (0) of laag-hoog (1) is. Als controle-lijn 2 een extra input is, dan hebben bit 2 (voor poort A) en bit 6 (voor poort B) een soortgelijke functie.

. Als controle-lijn 2 een extra input is, dan bepalen bit 1 (voor poort A) en bit 5 (voor poort B) of deze onafhankelijk van de parallelle datapoort werkt. Dit bit is 0 voor normale handshaking en 1 voor de onafhankelijke modus.

. Bit 3 (voor poort A) en bit 7 (voor poort B) bepalen of controle-lijn 2 een extra input-lijn (0) of een output-respons-lijn (1) is.

. Als controle-lijn 2 een output-respons-lijn is, dan bepalen bit 2 (voor poort A) en bit 6 (voor poort B) of zij in de automatische (0) of handbediende (1) modus werkt.

. Als controle-lijn 2 in de automatische modus werkt, bepalen bit 1 (voor poort A) en bit 5 (voor poort B) of de respons een klokcyclus (1) duurt of aanhoudt tot de randapparatuur een nieuw signaal over controle-lijn 1 stuurt (0).

. Als controle-lijn 2 in de handbediende modus werkt, bepalen bit 1 (voor poort A) en bit 5 (voor poort B) het niveau hiervan.

Enkele typerende voorbeelden zijn:

. De randapparatuur geeft DATA READY of PERIPHERAL READY aan met een hoog-laag signaal op controle-lijn 1. Er is geen respons nodig.
Van de 4 bits die een specifieke poort besturen wordt alleen vereist dat bit 0 0 moet zijn om een hoog-laag signaal op controle-lijn 1 te kunnen herkennen. De stand van de andere bits doet er niet toe, hoewel er een zekere algemene voorkeur bestaat om niet-gebruikte bits te wissen. De bits zouden dus 0000 moeten zijn.

. De randapparatuur geeft DATA READY of PERIPHERAL READY aan met een hoog-laag signaal op controle-lijn 1. Er is geen respons nodig. Bit 0 moet 1 zijn; de stand van de andere bits doet er niet toe. Bit 0 bepaalt welk signaal (hoog-laag of laag-hoog) de VIA herkent.

7	6	5	4	3	2	1	0	← Bit Nummer
IRQ	T1	T2	CB1	CB2	SR	CA1	CA2	← Interruptvlag-register

Bit nummer	Gezet door	Gewist door
0	Actieve overgang van het signaal op de CA2-lijn.	Lees- of schrijfopdracht op het uitvoer-register van de A-poort (ORA) met behulp van adres 0001.
1	Actieve overgang van het signaal op de CA1-lijn.	Lees- of schrijfopdracht op het uitvoer-register van de A-poort (ORA) met behulp van adres 0001.
2	Voltooing van 8 verschuivingen.	Lees- of schrijfopdracht op het schuif-register
3	Actieve overgang van het signaal op de CB2-lijn.	Lees- of schrijfopdracht op het uitvoer-register van de B-poort.
4	Actieve overgang van het signaal op de CB1-lijn.	Lees- of schrijfopdracht op het uitvoer-register van de B-poort.
5	"Time-out"-signaal van Teller 2.	Leesopdracht op laagstwaardige teller van T2 of schrijfopdracht op hoogstwaardige teller van T2.
6	"Time-out"-signaal van Teller 1.	Leesopdracht op laagstwaardige teller van T1 of schrijfopdracht op hoogstwaardige latch van T1.
7	Actieve en aangezette interrupt-conditie.	Opdracht die de interrupt-conditie wist.
Bits 0, 1, 3 en 4 zijn de I/O aansluitingsbevestigings-signalen. Bit 7 (IRQ) is 1 als een van de interrupts actief en aangezet is.		

Figuur 1-9: Het interruptvlag-register van de 6522 VIA

. De randapparatuur geeft DATA READY of PERIPHERAL READY aan met een hoog-laag signaal op controle-lijn 1. De poort moet een puls op controle-lijn 2 als respons sturen. De puls moet 1 klokcyclus duren, nadat de processor de input leest of de output schrijft.
De 4 bits moeten er als volgt uitzien:

Bit 3 = 1 om van controle-lijn 2 een uitvoer-lijn te maken
Bit 2 = 0 om controle-lijn 2 in de automatische modus te laten werken

De poort geeft zo de respons zonder tussenkomst van de processor.

Bit 1 = 1 om de respons 1 klokcyclus te laten duren
Bit 0 = 0 om een hoog-laag signaal op controle-lijn 1 te herkennen

. De randapparatuur geeft DATA READY of PERIPHERAL READY met een laag-hoog signaal aan op controle-lijn 1. De poort moet antwoorden door controle-lijn 2 laag te zetten tot de randapparatuur weer klaar is.

De 4 bits zien er dan zo uit:

Bit 3 = 1 omvan controle-lijn 2 uitvoer-lijn te maken
Bit 2 = 0 om controle-lijn 2 in de automatische modus te laten werken
Bit 1 = 0 om de respons te laten aanhouden tot de randapparatuur weer klaar is
Bit 0 = 1 om een laag-hoog signaal op controle-lijn 1 als een gereed-signaal te herkennen

. De randapparatuur geeft een DATA READY of PERIPHERAL READY aan met een laag-hoog signaal op controle-lijn 1. De processor moet op commando van een programma reageren. De bits moeten er zo uit zien

Bit 3 = 1 om van controle-lijn 2 een uitvoer-lijn te maken
Bit 2 = 1 om controle-lijn handbediend te laten werken
Bit 1 is de beginstand voor de controle-lijn
Bit 0 = 1 om een laag-hoog signaal op controle-lijn 1 als gereed-signaal te herkennen

De volgende innstructies kunnen worden gebruikt om de respons te produceren:

Maak CA2 een logische 1

```
LDA    VIAPCR       ;LEES HET RANDAPPARATUUR REGISTER
ORA    #%00000010   ;ZET CONTROLE-LIJN 2 OP 1
STA    VIAPCR
```

Maak CA2 een logische 0

```
LDA    VIAPCR       ;LEES HET RANDAPPARATUUR REGISTER
ORA    #%11111101   ;ZET CONTROLE-LIJN 2 OP 0
STA    VIAPCR
```

Maak CB2 een logische 1

```
LDA    VIAPCR       ;LEES HET RANDAPPARATUUR REGISTER
ORA    #%00100000   ;ZET CONTROLE-LIJN 2 OP 1
STA    VIAPCR
```

Maak CB2 een logische 0

```
LDA    VIAPCR       ;LEES HET RANDAPPARATUUR REGISTER
AND    #%11011111   ;ZET CONTROLE LIJN 2 OP 0
```

```
STA    VIAPCR
```

Deze instructies zijn niet afhankelijk van de inhoud van het controle-register, aangezien zij geen andere bits veranderen dan degenen die de respons bepalen.
Tabellen 1-13 en 1-14 vatten de werkingsmodi van de controle-lijnen CA1 en CA2 samen. Let op dat de automatische outputmodi niet helemaal gelijk zijn, omdat poort A een respons produceert na zowel een lees- als een schrijfactie, terwijl poort B alleen respons produceert na een schrijfactie.

Tabel 1-13: Operatie-modes van de controle-lijn CA2 van de 6522 VIA

PCR3	PCR2	PCR1	Modus
0	0	0	Invoer-modus - Zet CA2 interrupt-register bij een negatieve overgang van het invoer-signaal. Wis IFR0 bij een lees- of schrijfopdracht op het Randapperatuur-uitvoerregister A
0	0	1	Onafhankelijke interrupt gestuurde invoer-modus - Zet IFR0 bij een negatieve overgang van het invoer-signaal van CA2. Een lees- of schrijfopdracht op ORA wist de interruptvlag van CA2 niet.
0	1	0	Invoer-modus - Zet de CA2 interruptvlag bij een positieve overgang van het CA2 invoer-signaal. Wis IFR0 bij een lees- of schrijfopdracht op het Randapparatuur-uitvoer-register A
0	1	1	Onafhankelijke interrupt gestuurde invoer-modus - Zet IFR0 bij een positieve overgang van het CA2 invoer-signaal. Een lees- of schrijfopdracht op ORA wist de CA2 interruptvlag niet.
1	0	0	Uitvoer-modus met aansluitingsbevestiging - Zet CA2 uitvoer laag bij een lees- of schrijfopdracht op het Randapparatuur-uitvoerregister A. Zet de CA2 terug hoog bij een actieve overgang op CA1
1	0	1	Pulsuitvoer-modus - CA2 wordt 1 cyclus lang laag na een lees- of schrijfopdracht op het Randapparatuur-uitvoer-register A
1	1	0	Handmatige uitvoer-modus - De CA2 wordt laag gehouden in deze modus
1	1	1	Handmatige uitvoer-modus - De CA2 wordt hoog gehouden in deze modus

Het hulp-controleregister (auxiliary control register) is minder belangrijk dan het randapperatuur-controleregister. De bits van het hulp-controle- register hebben de volgende functies (zie figuur 1-8):

. Bits 0 en 1 zorgen er voor dat, als zij aan staan, de VIA de input data van poort A (bit 0) of poort B (bit 1) van een latch voorziet, als er een signaal op controle lijn 1 binnenkomt. Deze optie is er voor, dat als de randapparatuur slechts heel erg kort data stuurt, de data vastgehouden wordt totdat de processor klaar is om de data te verwerken.

. Bits 2, 3 en 4 bepalen de werking van het schuifregister, dat overigens zelden wordt gebruikt. Dit register geeft een eenvoudige seriele I/O-optie, maar de meeste ontwerpers gebruiken seriele I/O-chips als de 6551 en de 6850, die een compleet seriele-interface ingebouwd hebben.

Tabel 1-14: Operatie-modes van de controle-lijn CB2 van de 6522 VIA

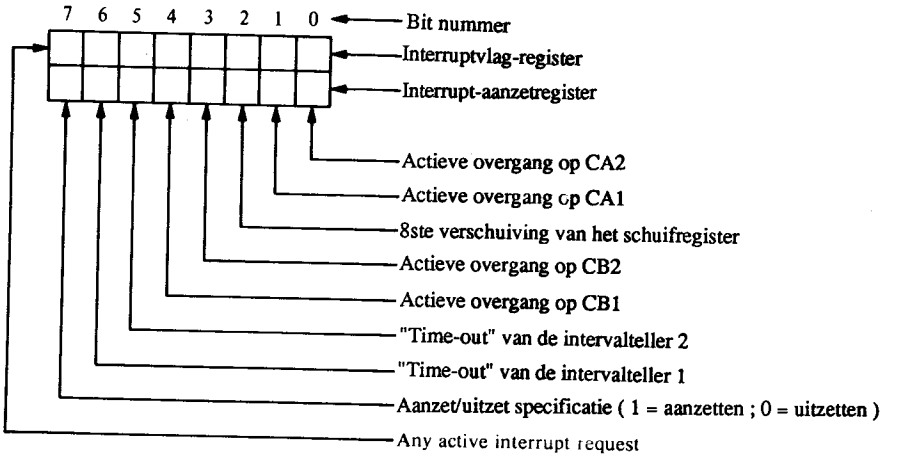
PCR7	PCR6	PCR5	Modus
0	0	0	Invoer-modus - Zet CB2 interrupt-register bij een negatieve overgang van het invoer-signaal. Wis IFR3 bij een lees- of schrijfopdracht op het Randapperatuur-uitvoerregister B
0	0	1	Onafhankelijke interrupt gestuurde invoer-modus - Zet IFR3 bij een negatieve overgang van het invoer-signaal van CB2. Een lees- of schrijfopdracht op ORB wist de interruptvlag van CB2 niet.
0	1	0	Invoer-modus - Zet de CB2 interruptvlag bij een positieve overgang van het CB2 invoer-signaal. Wis de CB2 interruptvlag bij een lees- of schrijfopdracht op ORB.
0	1	1	Onafhankelijke interrupt gestuurde invoer-modus - Zet IFR3 bij een positieve overgang van het CB2 invoer-signaal. Een lees- of schrijfopdracht op ORB wist de CB2 interruptvlag niet.
1	0	0	Uitvoer-modus met aansluitingsbevestiging - Zet CB2 uitvoer laag bij een schrijfopdracht op ORB. Zet de CB2 terug hoog bij een actieve overgang op CA1
1	0	1	Pulsuitvoer-modus - CB2 wordt 1 cyclus lang laag na een schrijfopdracht op ORB
1	1	0	Handmatige uitvoer-modus - De CB2 wordt laag gehouden in deze modus
1	1	1	Handmatige uitvoer-modus - De CB2 wordt hoog gehouden in deze modus

. Bit 5 bepaalt of timer 2 een enkel tijd-interval genereert (de zogenoemde 'one-shot' modus) (0) of pulsen op lijn PB6 telt (puls-tel modus) (0).

. Bit 6 bepaalt of timer 1 een enkel tijd-interval genereert (0) of continu werkt (1), waarbij zijn tellers vanuit de latches herladen worden na de afloop van elk interval.

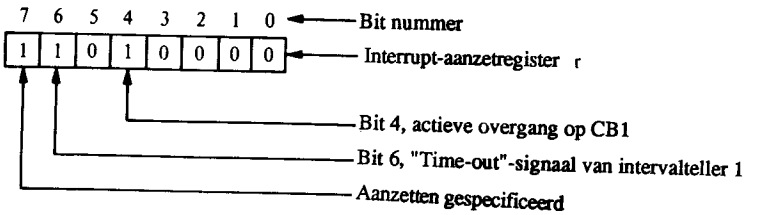
. Bit 7 bepaalt of timer 1 output pulsen op PB7 genereert (een logische 1 genereert pulsen).

Het gebruik van de meeste van deze functies is rechttoe-rechtaan. Ze zijn niet zo algemeen als de handshake-functies die door het randapperatuur- controle- register geregeld worden.
Je kan een 6522 VIA ook in een, door interrupts aangestuurde, modus laten werken. Interrupts worden in- of uitgeschakeld door bits in het 'interrupt enableregister' (interrupts aan/uitschakel register) (zie figuren 1-10 en 1-11) te zetten. Bit 7 (aan/uitschakel vlag) bepaalt of interrupts aan (1) of uit (0) staan. Interrupts kunnen worden herkend door het interrupt vlag register te bekijken (zie figuur 1-9). Tabel 1-16 vat het aan- en uitzetten van interrupt vlaggen op de 6522 VIA samen.



Het interruptvlag-register indentificeerd de actieve interrupts.
Een 1 in een willekeurige bit-positie geeft een actieve interrupt aan,
terwijl een 0 een nonactieve interrupt aangeeft.

Figuur 1-10: Het Interruptvlag- en het Interrupt-aanzetregister van de 6522 VIA



Figuur 1-11: Een typische aanzetoperatie op het interrupt-aanzetregister van de 6522 VIA

Tabel 1-15: Een samenvatting van de condities voor het aan- en uitzetten van de interrupt-vlaggen van de 6522 VIA.

	Gezet door	Gewist door
6	"Time-out"-signaal van Teller 1	Leesopdracht op laagstwaardige teller van Teller 1. Schrijfopdracht op hoogstwaardige latch van Teller 1.
5	"Time-out"-signaal van Teller 2	Leesopdracht op laagstwaardige teller van Teller 2. Schrijfopdracht op hoogstwaardige teller van Teller 2.
4	Actieve overgang van het signaal op CB1	Lees- of schrijfopdracht naar I/O poort |B.
3	Actieve overgang van het signaal op CB2 (invoer-modus)	Lees- of schrijfopdracht naar I/O poort B, alleen in invoer-modus.
2	Beeindiging van 8 verschuivingen	Lees- of schrijfopdracht op schuifregister
1	Actieve overgang van het signaal op CA1	Lees- of schrijfopdracht naar I/O poort A uitvoer-register (ORA) met behulp van adres 0001_2, alleen in invoer-modus.
0	Actieve overgang van het signaal op CA2 (invoer modus)	Lees- of schrijfopdracht naarI/O poort A uitvoerregister (ORA) met behulp van adres 0001_2, alleen invoer modus

HET SCHRIJVEN VAN INTERRUPT-GESTUURDE CODE

De 6502 reageert als volgt op een interrupt (een nonmaskable (niet uitschakelbare) interrupt, een ingeschakelde maskable interrupt of een BRK-instructie):

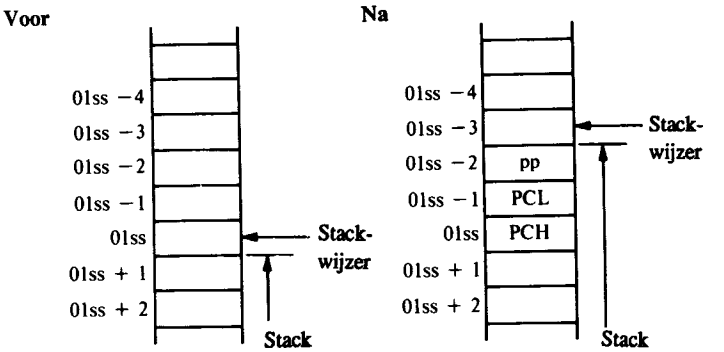
. Door de programmateller (hoogstwaardige byte eerst) en het statusregister op de stack te zetten, op de in figuur 1-12 aangegeven volgorde. Merk op dat het status register boven de programmateller terechtkomt; de reeks PHP, JSR zou de omgekeerde volgorde produceren. De waarde van de programmateller is in dit geval het adres van de volgende instructie; er is geen afstand van 1, zoals dit bij een JSR het geval is.

. Door de maskable interrupt uit te schakelen door de I-vlag in het statusregister aan te zetten.

. Door een bestemmingsadres van twee specifieke, opeenvolgende geheugenadressen af te halen en die bestemming in de programmateller te zetten.

Daarom moet de programmeur de volgende richtlijnen in acht nemen als hij interruptgestuurde code voor de 6502 schrijft:

. De accumulator en de indexregisters moeten nadrukkelijk bewaard en hersteld worden, als de afhandelings-routine deze wijzigt. Alleen het statusregister wordt automatische bewaard.



ss = Originele inhoud van de Stack-wijzer
pp = Originele inhoud van het Status (P) register
PCH = Originele inhoud van de hoogstwaardige byte van de Programma-teller
PCL = Originele inhoud van de laagstwaardige byte cab de Programma-teller

Figuur 1-12: Het antwoord van de 6502 Microprocessor op een interrupt

De afhandelingsroutine moet de accumulator voor de indexregisters wegschrijven. Dit omdat deze een indexregister alleen via de accumulator naar de stack kan wegschrijven. Typerende voorbeelden voor het bewaren en herstellen, zijn

```
PHA             ;BEWAAR ACCUMMULATOR
TXA             ;BEWAAR INDEXREGISTER X
PHA
TYA             ;BEWAAR INDEXREGISTER Y

PLA             ;HERSTEL INDEXREGISTER Y
TAY
PLA             ;HERSTEL INDEXREGISTER X
TAX
PLA             ;HERSTEL ACCUMULATOR VANUIT STACK
```

De volgorde waarin de indexregisters afgehandeld worden doet er niet toe, zolang het bewaren en herstellen in omgekeerde volgorde gaat.

. De interrupt hoeft niet expliciet weer ingeschakeld te worden, omdat de RTI (Return from Interrupt, keer terug van interrupt) instructie het oude statusregister herstelt als een onderdeel van haar werk. Dit herstelt de oorspronkelijke staat van de Interrupt Disable (interrupt uitschakel) vlag. Als u met de interrupts uitgeschakeld wenst terug te keren, kunt u de Interrupt Disable vlag met de volgende routine in de stack wijzigen

```
PLA                 ;HAAL STATUS REGISTER
ORA    #%00000100   ;SCHAKEL INTERRUPT IN STACK UIT
PHA                 ;EN ZET HEM TERUG IN STACK
```

Tabel 1-16: Interrupt vectoren van de 6502 Microprocessor

Bron	Gebruikte adres
Interrupt aanvraag (IRQ) en BRK-instructie Reset (RESET) Niet maskeerbare interrupt (NMI)	FFFE en FFFF FFFC en FFFD FFFA en FFFB
De adressen worden opgeslagen op de voor de 6502 gebruikelijke manier, met het laagstwaardige byte op het laagste adres.	

Merk hierbij op dat het toch wel gemakkelijk is dat het statusregister boven in de stack zit, in plaats van onder het terugkeeradres.

. Als u code hebt die de processor met uitgeschakelde interrupts moet uitvoeren, kunt u de SEI (Set Interrupt Disable) instructie gebruiken om de maskable interrupts uit te schakelen. Met een CLI (Clear Interrupt Disable) staat u de maskable interrupts weer toe. Als de routine mogelijk met zowel een in- als een uitgeschakelde Interrupt Disable vlag bereikt wordt, is het zaak de oorspronkelijke stand van de Interrupt Disable Vlag te herstellen. Dit houdt in dat u het statusregister, op bij voorbeeld de volgende wijze, moet bewaren en herstellen:

```
PHP                 ;BEWAAR OUDE INTERRUPT DISABLE
SEI                 ;SCHAKEL INTERRUPTS UIT
 .
 .  Code die met uitgeschakelde interrupts moet worden
 .  uitgevoerd
 .
PLP                 ;HERSTEL OUDE INTERRUPT DISABLE
```

De andere mogelijkheid (automatisch aan het einde de interrupts weer inschakelen) levert problemen op als de interrupts bij het binnenkomen van de routine al uitgeschakeld waren.

. Als u wilt dat de gebruiker het feitelijke beginadres van de afhandelingsroutine bepaalt, moet u een indirecte sprong naar een gevectord adres doen. Het adres van het begin van de routine wordt in de vectoren GEBRINT en GEBRINT+1 geplaatst. De routine ziet er dan zo uit:

```
JMP     (GEBRINT)     ;GA NAAR GESPECIFICEERD ADRES
```

Deze procedure verhoogt de respons-tijd van de interrupt met de uitvoeringstijd van een indirecte sprong (vijf klokcycli)

. U moet onthouden dat u eventuele informatie die van belang is voor het afhandelingsprogramma bewaart en herstelt. Dit soort informatie kan bestaan uit geheugenlocaties op pagina 0, belangrijke registers (zeker als ze alleen beschreven kunnen worden) en andere statusindicatoren.

. Om het programma weer algemeen toegankelijk te maken, moet u de stack gebruiken voor alle tijdelijke opslag die buiten het bereik van de registers valt. Zoals we in het deel over parameterdoorgifte hebben besproken, kunt u ruimte op de stack reserveren (NPARAM) met de routine

```
TSX                     ;VERPLAATS S NAAR A
TXA
SEC                     ;KEN NPARAM LEGE BYTES TOE
SBC     #NPARAM         ;EEN ALGEMENE METHODE OM DE SW AAN TE
                        ;PASSEN
TAX
TXS
```

Later kunt u de tijdelijke opslagruimte verwijderen met

```
TSX                     ;VERPLAATS S NAAR A
TXA
CLC
ADC     #NPARAM         ;VERWIJDER NPARAM BYTES
TAX
TXS
```

Als NPARAM slechts 1 of 2 is, kunt je deze instructies vervangen door het juiste aantal push (naar stack) of pull (van stack) instructies, waarbij niet naar de data gekeken wordt.

. De afhandelingsroutine moet de Decimale Modus vlag initialiseren met CLD of SED als deze ADC- of SBC-instructies gebruikt. De oude waarde van die vlag wordt automatisch bewaard en hersteld omdat deze onderdeel van het statusregister is, maar de afhandelingsroutine mag niet van een bepaalde beginstand uitgaan.

LAAT PROGRAMMA'S SNELLER LOPEN

. Over het algemeen kunt u programma's aanzienlijk sneller laten lopen, door eerst vast te stellen waar de tijd besteed wordt. Hierbij is vereist dat u te weten komt welke (andere dan vertragings-) lussen het meest uitgevoerd worden. Het terugbrengen van de uitvoeringstijd van een veelgebruikte lus, zal een groot effect hebben als gevolg van de vermenigvuldigingsfactor (het aantal instructies binnen de lus maal het aantal keren dat deze wordt doorlopen).
Als u eenmaal hebt vastgesteld welke lussen het meest worden uitgevoerd, kunt u de volgende technieken gebruiken om uitvoeringstijd te verkorten:

. Verwijder overbodige handelingen. Dit kan bij voorbeeld een constante zijn die na elke herhaling wordt verhoogd of het controleren op een speciale situatie. Het kan ook een constante waarde of een geheugenadres zijn die elke keer weer wordt opgehaald, in plaats van in een register te worden ondergebracht of indirect te worden gebruikt.

. Gebruik pagina 0 zoveel mogelijk voor tijdelijk data-opslag.

. Reorganiseer de lus om het aantal sprong (JMP of JSR) instructies te verminderen. U kunt vertakkingen vaak voorkomen door de begincondities te veranderen, de volgorde van de handelingen om te draaien of handelingen te combineren. Het omkeren van de handelingen kan vooral dan veel helpen als er numerieke vergelijkingen plaatsvinden. Dit omdat de vertakkingen op basis van gelijkheid niet afzonderlijk afgehandeld hoeven te worden. Het reorganiseren kan het ook mogelijk maken om voorwaarde-controle te combineren binnen de lus.

. Ga van achteren naar voren door arrays, in plaats van andersom. Dit maakt het mogelijk het index-register naar 0 af te tellen en de stand van de nulvlag te gebruiken als een eind-voorwaarde.

Er is dan geen expliciete vergelijking nodig. Let wel op dat u 1 van het basisadres moet aftrekken, omdat 1 de kleinste afstand is die feitelijk wordt gebruikt.

. Verhoog 16-bits tellers en indirecte adressen in plaats van ze te verlagen. 16-bits getallen zijn gemakkelijk te verhogen, omdat u kunt vaststellen of er een overdracht heeft plaatsgevonden door achteraf de laagstwaardige byte op 0 te controleren. Als u verlaagt moet u eerst op 0 controleren.

. Plaats de handelingen zoveel mogelijk in een lijn. Dit spaart minstens een JSR- en een RTS-instructie uit.

. Let op het speciale gebruik van de index-registers zodat u geen data tussen index-registers onderling hoeft te verplaatsen. Het enige register dat voor indirecte geindexeerde adressering kan worden gebruikt, is register Y; het enige register dat voor geindexeerde indirecte adressering of laden van c.q. schrijven naar de stapelwijzer dienst kan doen, is register X.

. Gebruik de instructies ASL, DEC, INC, LSR, ROL en ROR om data direct in het geheugen te bewerken zonder dat een register verplaatst hoeft te worden.

. Gebruik de BIT-instructie om bits 6 en 7 van een geheugenlocatie te testen, zonder de accumulator te laden.

. Gebruik CPX- en CPY-instructies om vergelijkingen uit te voeren zonder de accumulator te gebruiken.

Een algemene methode om de uitvoeringstijd te verkorten is om lange reeksen van instructies te vervangen door tabellen. Een enkele naslagtabel kan dezelfde handelingen verrichten als een reeks van instructies, mits er geen bijzondere vertakkingen of programmalogica in verwerkt moet zijn. Het kost weliswaar meer geheugen, maar dat maakt niet uit als er voldoende beschikbaar is. Als er voldoende geheugen is, kan een naslagtabel een verstandige aanpak zijn zelfs al komen vele van de bewerkingen meermaals voor, ook al geven vele invoers dezelfde uitvoer. Een naslagtabel is niet alleen snel, maar is ook gemakkelijk te schrijven, gemakkelijk te wijzigen en uiterst flexibel.

BESPAAR GEHEUGEN[8]

U kunt een programma aanzienlijk minder geheugen laten gebruiken door veelgebruikte reeksen van instructies te herkennen en te vervangen door sprongen naar subroutines. Hoe meer instructies u binnen een subroutine plaatst, hoe meer ruimte u bespaart. De nadelen hiervan zijn dat JSR en RTS zelf geheugen nodig hebben en tijd nodig hebben om te worden uitgevoerd. Bovendien zijn juist subroutines niet erg algemeen en kunnen daarom moeilijk te begrijpen en te gebruiken zijn. Sommige reeksen van instructies kunnen zelfs als subroutines in een monitor of systeemprogramma (dat altijd aanwezig is) staan. Dan kunt u die instructiereeksen vervangen door JSR naar de systeemroutine, zolang de terugkeer goed geregeld is.
Enkele van de methoden die rekentijd besparen, verminderen ook het geheugengebruik. Met name het gebruik van pagina 0, het reorganiseren van lussen, van achteren naar voren door een array werken, het verhogen van 16-bits hoeveelheden, direct met het geheugen werken en het gebruik van speciale instructies als CPX, CPY en BIT verminderen zowel de uitvoeringstijd als het geheugengebruik. Uiteraard vermindert u de rekentijd door het gebruik van lijncode (waarbij alle handelingen achter elkaar worden uitgevoerd), maar wordt het geheugengebruik verhoogd.
Naslagtabellen nemen over het algemeen meer geheugenruimte in, maar besparen uitvoeringstijd. U kunt hierbij geheugenruimte besparen door tussentijdse waarden weg te laten en de resultaten te

interpoleren, overbodige waarden te verwijderen met behulp van speciale tests en het bereik van de invoerwaarden te verkleinen. U zult vaak bemerken dat enkele controles of beperkingen vooraf de grootte van de benodigde tabel aanmerkelijk kunnen verkleinen.

LITERATUUR

1. Weller, W.J., *Practical Microcomputers Programming: The 6502*, Northern Technology Books, Evanston, Ill., 1980

2. Fischer, W.P., "Microprocessor Assembly Language Draft Standard", *IEEE Computer*, december 1979, pag. 96-109. Verdere discussies over de standaard zijn verschenen op pag. 79-80 van *IEEE Computer*, april 1980, en op pag. 8-9 van *IEEE Computer*, mei 1981. Zie ook Duncan, F.G., "Level-Independent Notation for Microcomputer Programs", *IEEE Micro*, mei 1981, pag. 47-56.

3. Osborne, A. *An Introduction to Microcomputers: Volume 1 - Basic Concepts*, 2de ed., Berkley: Osborne/McGraw-Hill, 1980.

4. Idem

5. Shankar, K.S., "Data Structures, Types, and Abstractions", *IEEE Computer*, april 1980, pag. 67-77.

6. Idem

5. Osborne, A. en G. Kane, *4 and 8 Bit Microprocessor Handbook*, Berkeley: Osborne/McGraw-Hill, 1981, pag. 9-55 tot 9-61 (6850 ACIA), Hoofdstuk 10 (6500 processoren en bijbehorende chips).

8. Schember, K.A. en J.R Rumsey, "Minimal Storage Sortimg and Searching Techniques for RAM Applications", *Computer*, juni 1977, pag. 92-100.

9. Seim, T.A., "Numerical Interpolation for Microprocessor-Based Systems", *Computer Designs*, februari 1978, pag. 111-16.

10. Abramovich, A. en T.R. Crawford, "An Interpolation Algorithm for Control Applications on Microprocessors", Verslag van de conferentie over industriele toepassingen voor microprocessors 1978, Philadelphia, Penn., pag.195-201 (verslag is verkrijgbaar bij *IEEE* of *IEEE Computer Society*).

Twee hobbybladen die vaak artikelen over 6502 assembleertaal programmeren plaatsen, zijn:

Compute	(P.O. Box 5406, Greensboro, NC 27403) en
Micro	(P.O. Box 6502, Chelmsford, MA 01824).

Hoofdstuk 2 Toepassing van aanvullende instructies en adresseringsmodi

Dit hoofdstuk geeft aan hoe u instructies en adresseringsmodi die de 6502 niet kent kunt toepassen. Uiteraard kan geen enkele instructieset alle mogelijke combinaties bevatten. Ontwerpers moeten hun keuze maken op basis van hoeveel opdrachtcodes er zijn, hoe gemakkelijk nog een andere combinatie geimplementeerd kan worden en hoe vaak deze dan zou worden gebruikt. Een beschrijving van aanvullende instructies en adresseringsmodi betekent niet dat de basisinstructieset onvolledig of slecht ontworpen is.
We richten onze aandacht op aanvullende instructies en adresseringsmodi die

- Duidelijk parallel zijn aan die welke in de instructieset zijn opgenomen.
- Beschreven zijn in de ontwerp-standaard Microprocessor Assembly Language Standard (IEEE Task P694)
- Besproken zijn in deel 1 van *An introduction to Microcomputers1*
- Opgenomen zijn in andere microprocessoren, in het bijzonder die welke nauw verwant of deels gelijk werkend zijn.[2, 3]

Dit hoofdstuk zal vooral interressant zijn voor diegenen met kennis van de machinetaal van andere processoren.

UITBREIDINGEN VAN DE INSTRUCTIESET

Bij de beschrijving van de uitbreidingen van de instructieset, volgen we de organisatiemethode zoals deze is voorgesteld in de ontwerp-standaard voor IEEE Task P694.[4] We brengen de instructies onder in de volgende groepen: rekenkundig, logisch, dataverplaatsing, vertak, sprong, subroutine-sprong, subroutine-terugkeer en diversen. In deze volgorde zullen zij ook worden behandeld. Binnen elk type instructie, bespreken we de opdrachtsoorten in deze volgorde: byte (8-bits), woord (16-bits), decimaal, bit, nibble of digit en meervoudig. Bij de beschrijving van de adresseringsmethoden, hanteren we deze volgorde: direct, indirect, onmiddellijk, geindexeerd, register, zelfvoorverhogend, zelfnaverhogend, zelfvoorverlagend, zelfnaverlagend, indirect gepreindexeerd (ook gepreindexeerd of geindexeerd indirect genoemd) en indirect gepostindexeerd (ook gepostindexeerd of indirect geindexeerd genoemd).

REKENKUNDIGE INSTRUCTIES

Tot deze groep rekenen wij optellen, optellen met overdracht, omgekeerd aftrekken, aftrekken met overdracht (lenen), verhogen, verlagen, vermenigvuldigen, delen, vergelijken, tweecomplement en uitbreiding. Instructies die niet vanzelfsprekend in een bepaalde categorie ondergebracht kunnen worden, worden voor het gemak herhaald.

Optelinstructies (zonder overdracht)

1. Tel geheugenlocatie ADR op bij de accumulator.

```
CLC                 ;WIS OVERDRACHT
ADC     ADR         ;(A) = (A) + (ADR)
```

Dezelfde methode werkt voor alle adresseringsmodi

2. Tel WAARDE op bij de accumulator.

```
CLC                 ;WIS OVERDRACHT
ADC     #WAARDE     ;(A) = (A) + WAARDE
```

3. Tel de Overdracht op bij de accumulator.

```
ADC     #0          ;(A) = (A) + 0 + OVERDRACHT
```

4. Tel geheugenlocatie ADR decimaal op bij de accumulator.

```
SED                 ;SCHAKEL DECIMALE MODUS IN
CLC                 ;WIS OVERDRACHT
ADC     ADR         ;(A) = (A) + (ADR) IN DECIMAAL
CLD                 ;SCHAKEL DECIMALE MODUS UIT
```

Een meer algemene methode herstelt de oorspronkelijk waarde van de D-vlag:

```
PHP                 ;BEWAAR OUDE D-VLAG
SED                 ;SCHAKEL DECIMALE MODUS IN
CLC                 ;WIS OVERDRACHT
ADC     ADR         ;(A) = (A) + (ADR) IN DECIMAAL
PLP                 ;HERSTEL OUDE D-VLAG
```

Let op dat het herstellen van het statusregister de overdracht van de optelling vernietigt.

5. Tel WAARDE decimaal op bij de accumulator.

```
SED                 ;SCHAKEL DECIMALE MODUS IN
CLC                 ;WIS OVERDRACHT
ADC     #WAARDE     ;(A) + (A) + WAARDE IN DECIMAAL
CLD                 ;SCHAKEL DECIMALE MODUS UIT
```

6. Tel Overdracht decimaal bij de accumulator op.

```
SED                 ;SCHAKEL DECIMALE MODUS IN
ADC     #0          ;(A) = (A) + OVERDRACHT IN DECIMAAL
CLD                 ;SCHAKEL DECIMALE MODUS UIT
```

7. Tel indexregister op bij de accumulator (met gebruik van de geheugenlocatie NPAG).

```
STX     NPAG        ;BEWAAR INDEXREGISTER OP PAGINA NUL
CLC                 ;WIS OVERDRACHT
ADC     NPAG        ;(A) = (A) + (X)
```

Deze methode geldt ook voor indexregister Y.

8. Tel de inhoud van de geheugenlocaties ADR en ADR+1 (HWB in ADR+1) op bij de geheugenlocaties SOM en SOM+1 (HWB in SOM+1).

```
CLC                     ;WIS OVERDRACHT
LDA     SOM
ADC     ADR             ;TEL LWB'S OP
STA     SOM
LDA     SOM+1           ;TEL HWB'S OP MET OVERDRACHT
ADC     ADR+1
STA     SOM+1
```

9. Tel het 16-bits getal WAARDE16 (WAARDE16H hoogstwaardige byte, WAARDE16L laagstwaardige byte) op bij de geheugenlocaties SOM en SOM+1 (HWB in SOM+1).

```
CLC                     ;WIS OVERDRACHT
LDA     SOM             ;TEL LWB'S OP ZONDER OVERDRACHT
ADC     #WAARDE16L
STA     SOM
LDA     SOM+1           ;TEL HWB'S OP MET OVERDRACHT
ADC     #WAARDE16H
STA     SOM+1
```

Optelinstructies (met overdracht)

1. Tel de Overdracht op bij de accumulator.

```
ADC     #0          ;(A) = (A) + OVERDRACHT
```

2. Tel WAARDE decimaal op bij de accumulator met Overdracht.

```
SED                 ;SCHAKEL DECIMALE MODUS IN
ADC     #WAARDE     ;(A) = (A) + WAARDE + OVERDRACHT
                    ; IN DECIMAAL
CLD
```

3. Tel de geheugenlocatie ADR decimaal op bij de accumulator met Overdracht.

```
SED                 ;SCHAKEL DECIMALE MODUS IN
ADC     ADR         ;(A) = (A) + (ADR) + OVERDRACHT
                    ; IN DECIMAAL
CLD                 ;SCHAKEL DECIMALE MODUS UIT
```

4. Tel de inhoud van geheugenlocaties ADR en ADR+1 (HWB in ADR+1) met Overdracht op bij de geheugenlocaties SOM en SOM+1 (HWB in SOM+1).

```
LDA     SOM         ;TEL LWB'S OP MET OVERDRACHT
ADC     ADR
STA     SOM
LDA     SOM+1       ;TEL HWB'S OP MET OVERDRACHT
ADC     ADR+1
```

```
STA     SOM+1
```

5. Tel 16-bits getal WAARDE16 (WAARDE16H hoogstwaardige byte, WAARDE16L laagstwaardige byte) met Overdracht op bij geheugenlocaties SOM en SOM+1 (HWB in SOM+1).

```
LDA     SOM             ;TEL LWB'S OP MET OVERDRACHT
ADC     WAARDE16L
STA     SOM
LDA     SOM+1           ;TEL HWB'S OP MET OVERDRACHT
ADC     WAARDE16H
STA     SOM+1
```

Aftrekinstructies (zonder lenen)

1. Trek geheugenlocatie ADR af van de accumulator.

```
SEC                     ;ZET HET GEINVERTEERDE LEENBIT AAN
SBC     ADR             ;(A) = (A) - (ADR)
```

De overdrachtvlag gedraagt zich als een geinverteerde Leen, dus deze moet 1 worden als haar waarde geen invloed op de aftrekking mag hebben.

2. Trek WAARDE af van de accumulator.

```
SEC                     ;ZET HET GEINVERTEERDE LEENBIT AAN
SBC     #WAARDE         ;(A) = (A) - WAARDE
```

3. Trek het geinverteerde leenbit af van de accumulator.

```
SBC     #0;(A) = (A) - (1-OVERDRACHT)
```

Het resultaat is (A)-1 als de Overdracht 0 en (A) als de Overdracht 1 is.

4. Trek geheugenlocatie ADR decimaal van de accumulator af.

```
SED                     ;GA NAAR DECIMALE MODUS
SEC                     ;ZET HET GEINVERTEERDE LEENBIT AAN
SBC     ADR             ;(A) = (A) - (ADR) IN DECIMAAL
CLD                     ;VERLAAT DECIMALE MODUS
```

De overdrachtvlag heeft in decimale modus dezelfde betekenis als in de binaire modus.

5. Trek WAARDE decimaal af van de accumulator.

```
SED                     ;SCHAKEL DECIMALE MODUS IN
SEC                     ;ZET HET GEINVERTEERDE LEENBIT AAN
SBC     #WAARDE         ;(A) = (A) - WAARDE IN DECIMAAL
CLD                     ;VERLAAT DECIMALE MODUS
```

6. Trek de inhoud van de geheugenlocaties ADR en ADR+1 (HWB in ADR+1) af van de geheugenlocaties VERSCHIL en VERSCHIL+1 (HWB in VERSCHIL+1).

```
LDA     VERSCHIL      ;TREK LWB'S AF ZONDER LEENBIT
SEC
SBC     ADR
STA     VERSCHIL
LDA     VERSCHIL+1    ;TREK HWB'S AF MET LEENBIT
SBC     ADR+1
STA     VERSCHIL+1
```

7. Trek het 16-bits getal WAARDE16 (WAARDE16H hoogstwaardige byte, WAARDE16L, laagstwaardige byte) af van de geheugenlocaties VERSCHIL en VERSCHIL+1 (HWB in VERSCHIL+1).

```
LDA     VERSCHIL      ;TREK LWB'S AF ZONDER LEENBIT
SEC
SBC     #WAARDE16L
STA     VERSCHIL
LDA     VERSCHIL+1    ;TREK HWB'S AF ZONDER LEENBIT
SBC     #WAARDE16H
STA     VERSCHIL+1
```

8. Trek het geinverteerde leenbit decimaal van de accumulator af.

```
SED                   ;SCHAKEL DECIMALE MODUS IN
SBC     #0            ;(A) = (A) - (1-OVERDRACHT) IN DECIMAAL
CLD                   ;SCHAKEL DECIMALE MODUS UIT
```

Aftrekken in omgekeerde instructies

1. Trek de accumulator af van WAARDE en stop het verschil in de accumulator.

```
EOR     #$FF          ;EENCOMPLEMENT OP A
CLC
ADC     #1            ;TWEECOMPLEMENT OP A
CLC
ADC     #WAARDE       ;MAAKT -A + WAARDE
```

of

```
STA     TEMP          ;BEWAAR A TIJDELIJK
LDA     #WAARDE       ;ZORGT VOOR WAARDE - A
SEC
SBC     TEMP
```

De Overdracht gedraagt zich in beide methoden als een geinverteerd leenbit. Dit houdt in dat de Overdracht op 1 wordt gezet als er geen leenbit nodig is.

2. Trek de accumulator af van de inhoud van geheugenlocatie ADR en plaats het verschil in de accumulator.

```
EOR     #$FF          ;EENCOMPLEMENT OP A
```

```
CLC
ADC    #1          ;TWEECOMPLEMENT OP A
CLC
ADC    ADR         ;ZORG VOOR -A + (ADR)
```

of

```
STA    TEMP        ;BEWAAR A TIJDELIJK
LDA    ADR         ;ZORG VOOR (ADR) - A
SEC
SBC    TEMP
```

3. Trek de accumulator decimaal af van WAARDE en stop het verschil weer in de accumulator.

```
SED                ;SCHAKEL DECIMALE MODUS IN
STA    TEMP        ;ZORG VOOR WAARDE - A
LDA    #WAARDE
SEC
SBC    TEMP
CLD                ;SCHAKEL DECIMALE MODUS UIT
```

4. Trek de accumulator decimaal af van de inhoud van geheugenlocatie ADR en plaats het verschil in de accumulator.

```
SED                ;GA NAAR DECIMALE MODUS
STA    TEMP        ;ZORG VOOR (ADR) - A
LDA    ADR
SEC
SBC    TEMP
CLD                ;VERLAAT DECIMALE MODUS
```

Aftrekken met Leenbit (Overdracht) instructies

1. Trek het geinverteerde leenbit van de accumulator af.

```
SBC    #0          ;(A) = (A) - (1-OVERDRACHT)
```

2. Trek WAARDE decimaal van de accumulator af, met leenbit.

```
SED                ;GA NAAR DECIMALE MODUS
SBC    #WAARDE     ;(A) = (A) - WAARDE - LEENBIT
                   ;IN DECIMAAL
CLD                ;VERLAAT DECIMALE MODUS
```

3. Trek de geheugenlocatie ADR met leenbit van de accumulator af.

```
SED                ;SCHAKEL DECIMALE MODUS IN
SBC    ADR         ;(A) = (A) - WAARDE - LEENBIT
                   ;IN DECIMAAL
CLD                ;VERLAAT DECIMALE MODUS
```

4. Trek de inhoud van geheugenlocaties ADR en ADR+1 (HWB in ADR+1) met leenbit af van de geheugenlocaties VERSCHIl en VERSCHIL+1 (HWB in VERSCHIL+1).

```
LDA     VERSCHIL     ;TREK LWB'S AF MET LEENBIT
SBC     ADR
STA     VERSCHIL
LDA     VERSCHIL+1   ;TREK HWB'S AF MET LEENBIT
SBC     ADR+1
STA     VERSCHIL+1
```

5. Trek 16-bits getal WAARDE16 (WAARDE16H hoogstwaardige byte, WAARDE16L laagstwaardige byte) met leenbit af van geheugenlocaties VERSCHIL en VERSCHIL+1 (HWB in VERSCHIL+1).

```
LDA     VERSCHIL     ;TREK LWB'S AF MET LEENBIT
SBC     WAARDE16L
STA     VERSCHIL
LDA     VERSCHIL+1   ;TREK HWB'S AF MET LEENBIT
SBC     WAARDE16H
STA     VERSCHIL
```

Verhooginstructies

1. Verhoog de accumulator, waarbij de overdrachtvlag aan wordt gezet als het resultaat 0 is.

```
CLC                  ;WIS OVERDRACHT
ADC     #1           ;VERHOOG DOOR 1 OP TE TELLEN
```

of

```
SEC                  ;ZET OVERDRACHT AAN
ADC     #0           ;VERHOOG DOOR 1 OP TE TELLEN
```

2. Verhoog de accumulator zonder de overdrachtvlag te beinvloeden.

```
TAX                  ;VERPLAATS A NAAR X
INX                  ;VERHOOG X
TXA                  ;EN VERPLAATS X NAAR A
```

INX heeft geen invloed op de overdrachtvlag, maar wel op de nulvlag.

3. Verhoog de stack-wijzer.

```
TSX                  ;VERPLAATS S NAAR X
INX                  ;VERHOOG X
TXS                  ;EN ZET NIEUWE WAARDE TERUG
```

of

```
TAX                  ;BEWAAR A
PLA                  ;VERHOOG STACK-POINTER
```

```
        TXA             ;HERSTEL A
```

Onthoud dat PLA de nul- en negatiefvlaggen beinvloedt.

4. Verhoog de accumulator decimaal met 1 (tel 1 decimaal bij A op).

```
        SED             ;GA NAAR DECIMALE MODUS
        CLC
        ADC   #1        ;(A) = (A) + 1 DECIMAAL
        CLD             ;VERLAAT DECIMALE MODUS
```

Onthoud dat INC en DEC binaire resultaten geven, zelfs wanneer de D-vlag aan staat.

5. Verhoog de inhoud van geheugenlocaties ADR en ADR+1 (HWB in ADR+1).

```
        INC   ADR       ;VERHOOG LWB
        BNE   KLAAR
        INC   ADR+1     ;OVERDRACHT NAAR HWB ALS LWB OP 0 KOMT
KLAAR   NOP
```

of

```
        LDA   ADR       ;VERHOOG LWB
        CLC
        ADC   #1
        STA   ADR
        LDA   ADR+1     ;MET OVERDRACHT NAAR HWB
        ADC   #0
        STA   ADR+1
```

De eerste mogelijkheid is duidelijk korter.

6. Verhoog decimaal de inhoud van geheugenlocaties ADR en ADR+1 (HWB in ADR+1).

```
        SED             ;GA NAAR DECIMALE MODUS
        LDA   ADR       ;TEL 1 OP BIJ LWB
        CLC
        ADC   #1
        STA   ADR
        BCC   KLAAR
        LDA   ADR+1     ;OVERDRACHT NAAR HWB INDIEN NODIG
        ADC   #0
        STA   ADR+1
KLAAR   CLD             ;VERLAAT DECIMALE MODUS
```

INC geeft een binair resultaat, ongeacht de stand van de decimaal-vlag. Merk op dat we de BCC-instructie kunnen verwijderen, zonder dat dit gevolgen heeft voor het resultaat. Maar deze verandering zou de gemiddelde uitvoertijd van het programma verhogen.

Verlaaginstructies

1. Verlaag de accumulator, waarbij de overdrachtvlag wordt gewist als het resultaat FF_{16} is.

```
        SEC             ;ZET HET GEINVERTEERDE LEENBIT AAN
        SBC     #1      ;VERLAAG DOOR 1 AF TE TREKKEN
```

of

```
        CLC             ;WIS HET GEINVERTEERDE LEENBIT
        SBC     #0      ;VERLAAG DOOR 1 AF TE TREKKEN
```

of

```
        CLC             ;WIS OVERDRACHT
        ADC     #$FF    ;VERLAAG DOOR -1 OP TE TELLEN
```

2. Verlaag de accumulator zonder de overdrachtvlag te beinvloeden.

```
        TAX             ;VERPLAATS A NAAR X
        DEX             ;VERLAAG X
        TXA
```

DEX heeft geen invloed op de stand van de overdrachtvlag; het heeft echter wel gevolgen voor de nulvlag.

3. Verlaag de stackwijzer.

```
        TSX             ;VERPLAATS S NAAR X
        DEX             ;VERLAAG X EN ZET NIEUWE WAARDE TERUG
        TXS
```

De stackwijzer kan ook verlaagd worden met PHA en PHP. Geen enkele vlag wordt door deze twee beinvloed.

4. Verlaag de accumulator decimaal (trek 1 decimaal van A af)

```
        SED             ;GA NAAR DECIMALE MODUS
        SEC
        SBC     #1      ;(A) = (A) - 1 DECIMAAL
        CLD             ;VERLAAT DECIMALE MODUS
```

5. Verlaag de inhoud van geheugenlocaties ADR en ADR+1 (HWB in ADR+1).

```
        LDA     ADR     ;IS LWB NUL?
        BNE     VERLLWB
        DEC     ADR+1   ;JA, LEEN VAN HWB
VERLLWB DEC     ADR     ;VOOR HET VERLAGEN VAN LWB
```

Het verlagen van een 16-bits getal is beduidend lastiger dan het verhogen. Bovendien is verhogen niet alleen sneller, maar het laat de accumulator ook ongewijzigd. Uiteraard is het mogelijk LDA te vervangen door LDX, LDY of de reeks INC, DEC. Een alternatief, dat geen registers gebruikt, is:

```
        INC     ADR     ;IS LWB NUL?
        DEC     ADR
        BNE     VERLLWB
```

```
          DEC    ADR+1      ;JA, LEEN VAN HWB
VERLLWB   DEC    ADR        ;VOORALEER DE LWB TE VERLAGEN
```

6. Verlaag de inhoud van geheugenlocaties ADR en ADR+1 decimaal (HWB in ADR+1).

```
          SED               ;SCHAKEL DECIMALE MODUS IN
          LDA    ADR        ;TREK 1 VAN LWB AF
          SEC
          SBC    #1
          STA    ADR
          BCS    KLAAR
          LDA    ADR+1      ;LEEN VAN HWB INDIEN NODIG
          SBC    #0
          STA    ADR+1
KLAAR     CLD               ;VERLAAT DECIMALE MODUS
```

DEC geeft een binair resultaat, zelfs als de decimaal-vlag aan staat. Merk op dat we de BCSinstructie zouden kunnen verwijderen zonder het resultaat te beinvloeden, ware het niet dat dit de gemiddelde looptijd van het programma zou verhogen.

Vermenigvuldigingsinstructies

1. Vermenigvuldig de accumulator met 2.

```
          ASL    A          ;VERMENIGVULDIG DOOR NAAR LINKS
                            ;TE SCHUIVEN
```

De volgende versie plaatst de Overdracht (indien aanwezig) in Y.

```
          LDY    #0         ;GA VAN HWB=0 UIT
          ASL    A          ;VERMENIGVULDIG DOOR NAAR LINKS
                            ;TE SCHUIVEN
          BCC    KLAAR
          INY               ;EN ZET OVERDRACHT IN Y
KLAAR     NOP
```

2. Vermenigvuldig de accumulator met 3 (waarbij ADR voor tijdelijke opslag wordt gebruikt).

```
          STA    ADR        ;BEWAAR A
          ASL    A          ;2 X A
          ADC    ADR        ;3 X A
```

3. Vermenigvuldig accumulator met 4.

```
          ASL    A          ;2 X A
          ASL    A          ;4 X A
```

We kunnen gevallen 1, 2 en 3 gemakkelijk uitbreiden door vermenigvuldiging met andere kleinere integers.

4. Vermenigvuldig een index register met 2

```
TAX                 ;VERPLAATS NAAR A
ASL     A           ;VERMENIGVULDIG DOOR NAAR LINKS
                    ;TE SCHUIVEN
TXA                 ;ZET RESULTAAT TERUG
```

5 Vermenigvuldig de inhoud van geheugenlocaties ADR en ADR + 1 (HWB in ADR +1 met 2

```
ASL     ADR         ;VERMENIGVULDIG DOOR NAAR LINKS
                    ;TE SCHUIVEN,
ROL     ADR+1       ;VERPLAATS OVERDRACHT NAAR HWB,
```

6. Vermenigvuldig de inhoud van geheugenlocaties ADR en ADR+1 (HWB in ADR+1) met 4.

```
ASL     ADR         ;VERMENIGVULDIG DOOR NAAR LINKS
                    ;TE SCHUIVEN,
ROL     ADR+1       ;VERPLAATS OVERDRACHT NAAR HWB,
ASL     ADR         ;EN VERMENIGVULDIG WEER
ROL     ADR+1
```

Uiteraard is het sneller door een byte in de accumulator te zetten, de accumulator te schuiven en het resultaat terug in het geheugen te zetten, dan beide bytes in het geheugen te laten staan.

Deelinstructies

1. Deel de accumulator ongetekend door 2

```
LSR     A           ;DEEL DOOR NAAR RECHTS TE SCHUIVEN
```

2. Deel de accumulator ongetekend door 4

```
LSR     A           ;DEEL DOOR NAAR RECHTS TE SCHUIVEN
LSR     A
```

3. Deel de accumulator door 2 getekend.

```
TAX                 ;BEWAAR DE ACCUMULATOR
ASL     A           ;SCHUIF TEKENBIT NAAR CARRY
TXA                 ;HERSTEL DE ACCUMULATOR
ROR     A           ;SCHUIF NAAR RECHTS MAAR BEWAAR TEKEN
```

De tweede instructie verplaatst het originele tekenbit (bit 7) naar de overdrachtvlag, zodat de uiteindelijke rotatie het kan bewaren. Dit is bekend als een rekenkundige schuiving, aangezien deze het teken van het getal bewaart, terwijl de grootte verkleind wordt. Het naar rechts copieren van het tekenbit heet tekenverlenging (sign extension).

4. Deel de inhoud van de geheugenlocaties ADR en ADR+1 (HWB in ADR+1) ongetekend door 2.

```
LSR     ADR+1       ;DEEL DOOR NAAR RECHTS TE SCHUIVEN
ROR     ADR         ;EN DE OVERDRACHT NAAR DE LWB
                    ;TE VERPLAATSEN
```

5. Deel de inhoud van de geheugenlocaties ADR en ADR+1 (HWB in ADR+1) getekend door 2.

```
LDA     ADR         ;VERPLAATS TEKEN NAAR OVERDRACHT
```

```
        ASL     A
        ROR     ADR+1       ;DEEL DOOR HET TEKEN NAAR RECHTS
                            ;TE SCHUIVEN
        ROR     ADR         ;EN DE OVERDRACHT NAAR DE LWB
                            ;TE VERPLAATSEN
```

Vergelijkingsinstructies

1. Vergelijk WAARDE bit-voor-bit met de accumulator, waarbij elke bit-positie die verschillend is wordt aangezet.

```
        EOR     #WAARDE
```

Onthoud dat een EXCLUSIVE OR van twee bits alleen 1 oplevert als de twee bits ongelijk zijn.

2. Vergelijk geheugenlocaties ADR1 en ADR1+1 (HWB in ADR1+1) met de geheugenlocaties ADR2 en ADR2+1 (HWB in ADR2+1). Zet de overdrachtvlag als ADR1 groter of gelijk aan ADR2 is (dat is, als ADR1 en ADR1+1 een 16-bits, ongetekend getal bevatten dat groter of gelijk is aan de inhoud van ADR2 en ADR2+1). Wis de overdrachtvlag als dit niet het geval is. Zet de nulvlag als beide getallen gelijk zijn en wis haar indien dit niet zo is.

```
        LDA     ADR1+1      ;VERGELIJK DE HWB'S
        CMP     ADR2+1
        BCC     KLAAR       ;WIS OVERDRACHT, WIS NUL ALS TWEEDE
                            ;GROTER IS
        BNE     KLAAR       ;ZET OVERDRACHT, WIS NUL ALS EERSTE
                            ;GROTER IS
        LDA     ADR1        ;HWB'S GELIJK? DAN VERGELIJK LWB'S
        CMP     ADR2        ;WIS OVERDRACHT ALS TWEEDE GROTER IS
KLAAR   NOP
```

3. Vergelijk geheugenlocaties ADR1 en ADR1+1 (HWB in ADR1+1) met het 16-bits getal WAARDE16 (WAARDE16H hoogstwaardige byte, WAARDE16L laagstwaardige byte). Zet de overdrachtvlag als de inhoud van ADR1 en ADR1+1 in de ongetekende zin groter of gelijk is aan WAARDE16. Wis anders de Overdracht. Zet de nulvlag als de inhoud van ADR1 en ADR+1 gelijk is aan WAARDE16 en wis deze als dit niet zo is.

```
        LDA     ADR1+1      ;VERGELIJK HWB'S
        CMP     WAARDE16H
        BCC     KLAAR       ;WIS OVERDRACHT, NUL ALS WAARDE16
                            ;GROTER IS
        BNE     KLAAR       ;ZET OVERDRACHT, WIS NUL ALS GEHEUGEN
                            ;GROTER IS
        LDA     ADR1        ;HWB'S GELIJK? VERGELIJK LWB'S
        CMP     #WAARDE16L  ;WIS OVERDRACHT ALS WAARDE16
                            ;GROTER IS
KLAAR   NOP
```

4. Vergelijk geheugenlocaties ADR1 en ADR+1 (HWB in ADR1+1) met de geheugenlocaties ADR2 en ADR2+1 (HWB in ADR2+1). Zet de overdrachtvlag als de eerste in ongetekende zin groter of gelijk is aan de tweede.

```
LDA     ADR1        ;VERGELIJK LWB'S
CMP     ADR2
LDA     ADR1+1      ;TREK HWB'S AF MET LEENBIT
SBC     ADR2+1
```

We gebruiken de SBC-instructie bij de hoogstwaardige bytes om het leenbit van de laagstwaardige bytes erbij te betrekken. Deze reeks vernietigt de oorspronkelijke waarde van A en zet de nulvlag alleen vanuit de laatste aftrekking.

5. Vergelijk de geheugenlocaties ADR1 en ADR+1 (HWB in ADR1+1) met het 16-bits getal WAARDE16 (WAARDE16H hoogstwaardige byte, WAARDE16L laagstwaardige byte). Zet de overdracht-vlag als de inhoud van ADR1 en ADR+1 in ongetekende zin groter of gelijk is aan WAARDE16.

```
LDA     ADR1        ;VERGELIJK LWB'S
CMP     WAARDE16L
LDA     ADR1+1      ;TREK HWB'S AF MET LEENBIT
SBC     WAARDE16H
```

Als de overdrachtvlag aangezet moet worden wanneer de inhoud van ADR1 en ADR1+1 groter is dan WAARDE16, dan moet de vergelijking met WAARDE16H uitgevoerd worden.

6. Vergelijk de stack-wijzer met de inhoud van geheugenlocatie ADR. Zet de overdrachtvlag als de stack wijzer in ongetekende zin groter of gelijk is aan de inhoud van de geheugenlocatie. Wis anders de overdrachtvlag. Zet de nulvlag als de twee waarden gelijk zijn en wis de vlag als dit niet zo is.

```
TSX                 ;VERPLAATS STACK WIJZER NAAR X
CPX     #ADR        ;EN VERGELIJK
```

7. Vergelijk de stack-wijzer met het 8-bits getal WAARDE. Zet de overdrachtvlag als de stack-wijzer in ongetekende zin groter dan of gelijk aan WAARDE is. Zet de nulvlag als de twee waarden gelijk zijn en wis deze als dit niet het geval is.

```
TSX                 ;VERPLAATS STACK-WIJZER NAAR X
CPX     #WAARDE     ;EN VERGELIJK HAAR
```

8. Blokvergelijking. Vergelijk de accumulator met de geheugenbytes die beginnen bij adres BASIS en ga door tot er of een gelijke is gevonden (aangegeven door Overdracht = 1) of tot een byte teller in geheugenlocatie TELLER de nul bereikt (aangegeven door Overdracht = 0).

```
            LDY     TELLER      ;HAAL TELLER OP
            BEQ     NIETVIND    ;EINDE ALS TELLER NUL IS
            LDX     #0          ;BEGIN INDEX MET 0
VRGLBYT     CMP     BASIS,X     ;BEKIJK HUIDIGE BYTE
            BEQ     KLAAR       ;KLAAR INDIEN GELIJK (OVERDRACHT = 1)
            INX                 ;ANDERS, GA NAAR VOLGENDE BYTE
            DEY
            BNE     VRGLBYT     ;ALS ER NOG OVER ZIJN
NIETVIND    CLC                 ;ANDERS ERUIT, OVERDRACHT GEWIST
KLAAR       NOP
```

Onthoud dat twee gelijke getallen de overdrachtvlag zetten.

Tweecomplement-instructies

1. Inverteer accumulator.

```
EOR     #$FF        ;EENCOMPLEMENT
CLC
ADC     #1          ;TWEECOMPLEMENT
```

Het tweecomplement is een eencomplement plus 1.

```
STA     TEMP        ;ALTERNATIEF IS 0 - (A)
LDA     #A
SEC
SBC     TEMP
```

2. Inverteer geheugenlocatie ADR,

```
LDA     #0          ;VORM 0 - (ADR)
SEC
SBC     ADR
STA     ADR
```

3. Inverteer geheugenlocaties ADR en ADR+1 (HWB in ADR+1).

```
LDA     ADR         ;EENCOMPLEMENT VAN LWB
EOR     #$FF
CLC                 ;TEL 1 ERBIJ OP VOOR TWEECOMPLEMENT
ADC     #1
STA     ADR
LDA     ADR+1       ;EENCOMPLEMENT VAN HWB
EOR     #$FF
ADC     #0          ;PLUS OVERDRACHT VOOR TWEECOMPLEMENT
STA     ADR+1
```

of

```
LDA     #0          ;VORM 0 - (ADR+1) (ADR)
SEC
SBC     ADR         ;TREK LWB'S AF ZONDER LEENBIT
STA     ADR
LDA     #0          ;TREK HWB'S AF MET LEENBIT
SBC     ADR+1
STA     ADR+1
```

4. Negencomplement van de accumulator (dat is, vervang A door 99-A).

```
STA     TEMP        ;VORM 99-A
LDA     #$99
SEC
```

```
        SBC     TEMP
```

We hoeven niet in de decimale modus te werken, omdat 99-A altijd een geldig BCD-getal is indien A oorspronkelijk een geldig BCD-getal bevatte.

5. Tien's complement van de accumulator (dat is, vervang A door 100-A)

```
        SED                     ;SCHAKEL NAAR DECIMALE MODUS
        STA     TEMP            ;VORM 100-A
        LDA     #0
        SEC
        SBC     TEMP
        CLD     ;VERLAAT DECIMALE MODUS
```

Verlenginstructies

1. Verleng de accumulator tot een 16-bits ongetekend getal in geheugenlocaties ADR en ADR+1 (HWB in ADR+1).

```
        STA     ADR             ;8-BIT VERPLAATSING
        LDA     #0              ;VERLENGEN TOT 16 BITS MET 0-EN
        STA     ADR+1
```

2. Verleng de accumulator tot een 16-bits getekend getal in de geheugenlocaties ADR en ADR+1 (HWB in ADR+1).

```
        STA     ADR             ;8-BITS VERPLAATSING
        ASL     A               ;VERPLAATS TEKENBIT NAAR OVERDRACHT
        LDA     #$FF            ;(A) = -1 + TEKENBIT
        ADC     #0
        EOR     #$FF            ;(A) = -TEKENBIT
        STA     ADR+1           ;ZET HWB NAAR -TEKENBIT
```

Het resultaat van de berekening is -(-1 + TEKENBIT) - 1 = -TEKENBIT. Ofte wel, (ADDR+1) = 00 als A positief was en FF_{16} als A negatief was. Een alternatief hiervoor is:

```
        STA     ADR             ;8-BITS VERPLAATSING
        LDX     #$FF            ;(X) = -1
        ASL     A
        BCS     BRGTKN
        INX                     ;(X) = -1 + (1 - TEKENBIT) = -TEKENBIT
BRGTKN  STX     ADR+1           ;ZET HWB NAAR -TEKENBIT
```

3. Verleng bit 0 van de accumulator over de hele accumulator. Ofte wel (A) = 00 als bit 0 = 0 en FF_{16} als bit 0 = 1.

```
        LSR     A               ;OVERDRACHT = BIT 0
        LDA     #$FF            ;(A) = -1 + BIT 0
        ADC     #0
        EOR     #$FF            ;(A) = -BIT 0
```

Net als in geval 2, willen we -1 als uitkomst als het gespecificeerde bit 1 is en 0 als het gespecificeerde bit 0 is. Ofte wel, we willen het negatief van de originele bit-waarde. De reeks LDA #$FF, ADC #0 geeft uiteraard het resultaat -1 + Overdracht. Het eencomplement geeft dan het negatief van datgene wat we hadden, min 1 (of 1 - Overdracht - 1 = -Overdracht).

4. Tekenfunctie. Vervang de waarde in de accumulator door 0 als deze positief is en door FF_{16} als deze negatief is.

```
        ASL     A           ;VERPLAATS TEKENBIT NAAR OVERDRACHT
        LDA     #$FF        ;(A) = -1 + TEKENBIT
        ADC     #0
        EOR     #$FF        ;(A) = -TEKENBIT
```

5. Teken functie van een geheugenlocatie. Maak accumulator 00 als geheugenlocatie ADR positief is en FF_{16} als deze negatief is.

```
        LDX     #$FF        ;GA VAN NEGATIEF UIT
        LDA     ADR         ;IS (ADR) POSITIEF?
        BMI     KLAAR
        INX                 ;JA, ZET TEKEN OP NUL
KLAAR   TXA
```

De methode zoals deze in geval 4 werd getoond, kan ook worden gebruikt.

LOGISCHE INSTRUCTIES

In deze categorie bekijken we de logische AND, logische OR, logische EXCLUSIVE OR, logische NOT (complement), schuif (shift), roteer- en testinstructies.

Logische AND-instructies

1. Wis accumulator -bit

```
        AND     #MASKER     ;WIS BIT DOOR MASKERING
```

MASKER heeft een 0 op de bit-posities staan die gewist moeten worden en een 1 op de bit posities die ongewijzigd moeten blijven. Bij voorbeeld:

```
        AND     #%11011011  ;WIST DE BITS 2 EN 5
```

Onthoud dat door een logische AND met 1, het bit ongewijzigd blijft.

2. Bit-test. Zet de vlaggen volgens de waarde van een bit van geheugenlocatie ADR.

Bits 0 tot en met 5

```
        LDA     #MASKER
        BIT     ADR         ;TEST BIT VAN ADR
```

MASKER moet een 1 op de te testen positie hebben staan en een 0 op alle andere plaatsen. De nulvlag wordt op 1 gezet als het geteste bit 0 is en op 0 als het geteste bit 1 is.

Bits 6 of 7

```
BIT    ADR        ;TEST BITS 6 EN 7 VAN ADR
```

Deze enkele instructie zet de negatiefvlag naar bit 7 van ADR en de overschotvlag naar bit 6 van ADR, ongeacht de waarde van de accumulator. Let op dat de vlaggen niet als bij de nulvlag worden geinverteerd bij normaal maskeren.

3. Doe een logische AND direct met de status-indicatoren (vlaggen). Voer een logische AND uit op een byte met daarin de inhoud van het statusregister, waarbij die vlaggen gewist worden die logisch met 0 ge-AND zijn. Deze instructie is geimplementeerd op de 6809.

```
PHP                ;PLAATS STATUS IN A
PLA
AND    #MASKER     ;WIS DE VLAGGEN
PHA                ;EN RESULTAAT TERUG IN STATUS
PLP
```

Logische OR instructies

1. Zet een bit van de accumulator.

```
ORA    #MASKER     ;ZET EEN BIT DOOR MASKERING
```

MASKER heeft 1-en in de bit-posities staan die gezet moeten worden en 0-en op die plaatsen die ongewijzigd dienen te blijven. Bij voorbeeld:

```
ORA    #%00010010  ;ZET BITS 1 en 4
```

Onthoud dat een logische OR met 0 het bit ongewijzigd laat.

2. Test de geheugenlocaties ADR en ADR+1 op 0. Zet de nulvlag aan als beide bytes 0 zijn.

```
LDA    ADR         ;TEST 16-BITS GETAL OP NUL
ORA    ADR+1
```

De nulvlag wordt alleen dan gezet als beide bytes van het 16-bit getal 0 zijn. De andere vlaggen zijn ook gewijzigd.

3. Een directe logische OR met de statusindicatoren (vlaggen). Doe een logische OR op een byte directe data (MASKER) met de inhoud van het status-register. Hierbij worden de vlaggen gezet die met een 1 ge-ORed worden. Deze instructie is geimplementeerd op de 6809 microprocessor.

```
PHP                ;VERPLAATS STATUS NAAR A
PLA
ORA    #MASKER     ;ZET VLAGGEN
PHA                ;EN ZET RESULTAAT TERUG
```

```
PLP
```

Logische EXCLUSIVE OR-instructies

1. Complementeer bit van de accumulator.

```
EOR    #MASKER    ;COMPLEMENTEER BIT DOOR MASKERING
```

MASKER heeft 1-en op de bit-posities die gecomplementeerd dienen te worden en 0-en op de bitposities die ongewijzigd moeten blijven. Bijvoorbeeld,

```
EOR    #%11000000    ;COMPLEMENTEER BITS 6 EN 7
```

Onthoud dat een als een bit met 0 wordt geEXCLUSIVE ORed, dat dit dan onveranderd blijft.

2. Complementeer de accumulator, waarbij de vlaggen worden gezet.

```
EOR    #%11111111    ;COMPLEMENTEER DE ACCUMULATOR
```

Een logische EXCLUSIVE OR op de accumulator met alleen 1-en, inverteert alle bits.

3. Vergelijk bit voor bit de geheugenlocatie ADR met de accumulator, waarbij elke afwijkende bitpositie wordt gezet.

```
EOR    ADR        ;BIT VOOR BIT VERGELIJKING
```

De EXCLUSIVE OR functie is dezelfde als een 'niet gelijk' (not equal) functie. Let op dat de Negatief (Teken) -vlag 1 is als de twee gegevens verschillende waarden hebben in bitpositie 7.

4. Tel de geheugenlocatie ADR logisch op bij de accumulator (dit betekent zonder overdrachten tussen bit-posities).

```
EOR    ADR        ;LOGISCHE OPTELLING
```

De EXCLUSIVE OR functie is gelijk aan een bit bij bit som zonder overdrachten. Logische sommen worden vaak gebruikt bij het uitvoeren van checksums (controle-sommen) en het zoeken en verbeteren van fouten.

Logische NOT instructies

1. Complementeer de accumulator, waarbij de vlaggen worden gezet.

```
EOR    #$FF       ;COMPLEMENTEER DE ACCUMULATOR
```

Een logische EXCLUSIVE OR met alleen 1-en inverteert alle bits.

2. Complementeer een bit van de accumulator.

```
EOR    #MASKER    ;COMPLEMENTEER BIT DOOR MASKERING
```

MASKER heeft 1-en op de bit-posities die gecomplementeerd moeten worden en 0-en op de plaatsen die ongewijzigd dienen te blijven. Bij voorbeeld:

```
EOR    #$01010001   ;COMPLEMENTEER BITS 0, 4, EN 6
```

Onthoud dat een EXCLUSIVE OR van een bit met 0, dit ongewijzigd laat.

3. Complementeer een geheugenlocatie.

```
LDA    ADR
EOR    #$FF        ;COMPLEMENTEER
STA    ADR
```

4. Complementeer bit 0 van een geheugenlocatie.

```
INC    ADR         ;COMPLEMENTEER DOOR VERHOGING
```

of

```
DEC    ADR         ;COMPLEMENTEER DOOR VERLAGING
```

Elk van deze twee instructies kan uiteraard de andere bits in de geheugenlocatie beinvloeden. Echter, de uiteindelijke waarde van bit 0 zal zeker 0 zijn als het oorspronkelijk 1 was en 1 als het oorspronkelijk 0 was.

5. Complementeer digit (4 bits) van de accumulator.

. Laagstwaardige digit

```
EOR    #%00001111   ;COMPLEMENTEER 4 LAAGSTWAARDIGE BITS
```

. Hoogstwaardige digit

```
EOR    #%11110000   ;COMPLEMENTEER 4 HOOGSTWAARDIGE BITS
```

Deze handelingen zijn nuttig als de accumulator een decimale digit met negatieve logica bevat (als, bij voorbeeld, input van een draaischakelaar met 10 posities).

6. Complementeer de overdrachtvlag

```
ROR    A           ;VERPLAATS OVERDRACHT NAAR BIT 7 VAN A
EOR    #$FF        ;COMPLEMENTEER ALLES VAN A
ROL    A           ;EN ZET GECOMPLEMENTEERDE
                   ;OVERDRACHTVLAG TERUG.
```

Andere combinaties zoals ROL, EOR, ROR, of ROR, EOR, ASL werken net zo goed. We kunnen de accumulator in tact laten door deze eerst op de stack op te bergen en achteraf terug te halen.

Een alternatief dat geen invloed op de accumulator heeft, is

```
BCC    ZETOVER
CLC                ;WIS OVERDRACHT ALS DEZE GEZET WAS
```

```
        BCC     KLAAR
ZETOVER SEC                     ;ZET OVERDRACHT ALS DEZE GEWIST WAS
KLAAR   NOP
```

Schuifinstructies

1. Schuif de accumulator rekenkundig naar rechts, waarbij het tekenbit bewaard moet blijven.

```
        TAX                     ;BEWAAR ACCUMULATOR
        ASL     A               ;ZET TEKENBIT IN OVERDRACHT
        TXA
        ROR     A               ;SCHUIF NAAR RECHTS, BEWAAR TEKEN
```

We hebben een kopie van het tekenbit nodig voor een rekenkundige schuiving. Uiteraard kunnen we ook een geheugenlocatie in plaats van een indexregister gebruiken voor tijdelijke opslag.

2. Schuif geheugenlocaties ADR en ADR+1 (HWB in ADR+1) logisch naar links.

```
        ASL     ADR             ;SCHUIF LWB LOGISCH NAAR LINKS
        ROL     ADR+1           ;EN PLAATS OVERDRACHT IN HWB
```

De crux zit hem hierin dat we de hoogstwaardige byte moeten roteren. De eerste 8-bits schuiving verplaatst 1 bit (het laagstwaardige bit voor een schuiving naar rechts en het hoogstwaardige bit voor een schuiving naar links) naar de Overdracht. De 8-bits rotatie zet dat bit vanuit de Overdracht in de tweede helft van het woord.

3. Schuif geheugenlocaties ADR en ADR+1 (HWB in ADR+1) logisch naar rechts.

```
        LSR     ADR+1           ;SCHUIF HWB LOGISCH NAAR RECHTS
        ROR     ADR             ;EN ZET OVERDRACHT IN LWB
```

4. Schuif geheugenlocaties ADR en ADR+1 (HWB in ADR+1) rekenkundig naar rechts.

```
        LDA     ADR+1           ;VERPLAATS TEKENBIT NAAR OVERDRACHT
        ASL     A
        ROR     ADR+1           ;SCHUIF HWB REKENKUNDIG NAAR RECHTS
        ROR     ADR             ;EN VERPLAATS OVERDRACHT NAAR LWB
```

5. Schuif geheugenlocaties ADR en ADR+1 (HWB in ADR+1) 1 digit naar links; ofte wel schuif het 16-bits getal 4 bits logisch naar links.

```
        LDX     #4              ;AANTAL SCHUIVINGEN = 4
        LDA     ADR             ;VERPLAATS LWB NAAR A
SCHF1   ASL     A               ;SCHUIF LWB LOGISCH NAAR LINKS
        ROL     ADR+1           ;EN ZET OVERDRACHT IN HWB
        DEX
        BNE     SCHF1           ;TEL DE BITS
        STA     ADR             ;ZET LWB TERUG IN ADR
```

Een kortere maar langzamere versie die de accumulator niet gebruikt is:

```
        LDX     #4          ;AANTAL SCHUIVINGEN = 4
SCHF1   ASL     ADR         ;SCHUIF LWB LOGISCH NAAR LINKS
        ROL     ADR+1       ;EN VERPLAATS OVERDRACHT NAAR HWB
        DEX
        BNE     SCHF1       ;TEL AANTAL SCHUIVINGEN
```

6. Schuif geheugenlocaties ADR en ADR+1 (HWB in ADR+1) 1 digit naar rechts; ofte wel, schuif het 16-bits getal 4 bits logisch naar rechts.

```
        LDX     #4          ;AANTAL SCHUIVINGEN = 4
        LDA     ADR         ;ZET LWB IN A
SCHF1   LSR     ADR+1       ;SCHUIF HWB LOGISCH NAAR RECHTS
        ROR     A           ;EN ZET OVERDRACHT IN LWB
        DEX
        BNE     SCHF1       ;TEL SCHUIVINGEN
        STA     ADR         ;ZET LWB TERUG IN ADR
```

Een kortere, maar langzamere versie die de accumulator niet gebruikt is:

```
        LDX     #4          ;AANTAL SCHUIVINGEN = 4
SCHF1   LSR     ADR+1       ;SCHUIF HWB LOGISCH NAAR RECHTS
        ROR     ADR         ;EN ZET OVERDRACHT IN LWB
        DEX
        BNE     SCHF1       ;TEL SCHUIVINGEN
```

7. Normaliseer de geheugenlocaties ADR en ADR+1 (HWB in ADR+1); ofte wel, schuif het 16-bit getal naar links, totdat het hoogstwaardige bit 1 is. Schuif niet als het hele getal 0 is.

```
        LDA     ADR+1       ;EINDE INDIEN REEDS GENORMALISEERD
        BMI     KLAAR
        ORA     ADR         ;OF ALS HELE GETAL = 0
        BEQ     KLAAR
        LDA     ADR         ;ZET LWB IN A
SCHUIF  ASL     A           ;SCHUIF LWB 1 BIT LOGISCH NAAR LINKS
        ROL     ADR+1       ;EN ZET OVERDRACHT IN HWB
        BPL     SCHUIF      ;GA DOOR TOT HWB 1 IS
        STA     ADR         ;ZET LWB TERUG IN ADR
KLAAR   NOP
```

Roteerinstructies

Een rotatie door of met de Overdracht gedraagt zich alsof de gegevens in een cirkel opgeslagen zijn, waarbij het laagstwaardige bit met het hoogstwaardige bit verbonden is door de overdrachtvlag. Een rotatie zonder de Overdracht verschilt hierin, dat ze zich gedraagt alsof het laagstwaardige bit direct verbonden is met het hoogstwaardige bit.

1. Roteer geheugenlocaties ADR en ADR+1 (HWB in ADR+1) 1 bit-positie naar rechts door de Overdracht.

```
        ROR     ADR+1       ;ZET BIT 8 IN OVERDRACHT
        ROR     ADR         ;EN VERDER IN BIT 7
```

2. Roteer de geheugenlocaties ADR en ADR+1 (HWB in ADR+1) 1 bit-positie naar rechts zonder Overdracht.

```
LDA     ADR         ;ZET BIT 0 IN OVERDRACHT
ROR     A
ROR     ADR+1       ;ROTEER HWB WAARBIJ BIT 0 LINKS
                    ;BINNENKOMT
ROR     ADR         ;ROTEER LWB
```

3. Roteer geheugenlocaties ADR en ADR+1 (HWB in ADR+1) 1 bit-positie naar links, door de Overdracht.

```
ROL     ADR         ;ROTEER BIT 7 NAAR OVERDRACHT
ROL     ADR+1       ;EN VERDER IN BIT 8
```

4. Roteer geheugenlocaties ADR en ADR+1 (HWB in ADR+1) 1 bit-positie naar links, zonder Overdracht

```
LDA     ADR+1       ;ZET BIT 15 IN OVERDRACHT
ROL     A
ROL     ADR         ;ROTEER LWB WAARBIJ BIT 15 VAN RECHTS
                    ;BINNENKOMT
ROL     ADR+1
```

Testinstructies

1. Test accumulator. Zet vlaggen naar de waarde van de accumulator zonder deze waarde te veranderen.

```
TAX                 ;VERPLAATS EN ZET VLAGGEN
```

of

```
TAY                 ;VERPLAATS EN ZET VLAGGEN
```

Het volgende alternatief beinvloedt geen van beide indexregisters.

```
CMP     #0          ;TEST ACCUMULATOR
```

De instructies AND #$FF of ORA #0 hebben hetzelfde effect, zonder dat de overdrachtvlag wordt gezet (hetgeen bij een CMP #0 wel gebeurt).

2. Test indexregister. Zet de vlaggen naar de waarde in een indexregister zonder dat de waarde ervan wordt veranderd.

```
CPX     #0          ;CONTROLEER WAARDE IN INDEXREGISTER
```

3. Test geheugenlocatie. Zet de vlaggen naar de waarde in geheugenlocatie ADR zonder dat deze verandert.

```
INC    ADR        ;CONTROLEER WAARDE IN GEHEUGENLOCATIE
DEC    ADR
```

4. Test een paar van geheugenlocaties. Zet de nulvlag naar de waarde in de geheugenlocaties ADR en ADR+1

```
LDA    ADR        ;TEST 16-BITS GETAL OP 0
ORA    ADR+1
```

Deze reeks zet de nulvlag op 1 als beide bytes van het 16-bits getal 0 zijn. Deze procedure kan gemakkelijk worden uitgebreid om getallen van elke willekeurige lengte te testen.

5. Test bit van accumulator.

```
AND    #MASKER    ;TEST BIT DOOR MASKERING
```

MASKER heeft een 1 op de bit-positie staan die getest moet worden en een 0 op alle andere bit-posities. De instructie zet de nulvlag op 1 als de geteste bit-positie een 0 bevat en 0 als de geteste bitpositie een 1 bevat. Bij voorbeeld:

```
AND    #%00001000   ;TEST BIT 3 DOOR MASKERING
```

Het resultaat is 0 als bit 3 van A = 0 en 00001000 (binair) als bit 3 van A = 1. Dus de nulvlag bevat uiteindelijk het logische complement van bit 3.

6. Vergelijk geheugenlocatie ADR bit-voor-bit met de accumulator. Zet iedere verschillende bit-positie.

```
EOR    ADR        ;BIT-VOOR-BIT VERGELIJKING
```

De EXCLUSIVE OR functie is dezelfde als een niet gelijk (not equal) functie.

DATA-OVERDRACHTINSTRUCTIES

In deze categorie worden de laad-, opslag-, verplaatsings-, uitwisselings-, wis- en zetinstructies bekeken.

Laadinstructies

1. Laad de accumulator indirect vanuit het adres dat in de geheugenlocaties PGNUL en PGNUL+1 is opgeslagen.

```
LDY    #0           ;GEEN INDEXERING
LDA    (PGNUL),Y    ;LAAD INDIRECT GEINDEXEERD
```

De enige instructie die echte indirecte adressering kent, is JMP. U kunt echter ook normale indirecte adressering gebruiken door de postindexering (indirect geindexeerd) adresseringsmodus te benutten. Hiervoor moet indexregister Y op 0 gezet zijn.
Een alternatief hiervoor is indexregister X te wissen en preindexering te gebruiken.

```
LDX    #0            ;GEEN INDEXERING
LDA    (PGNUL,X)     ;LAAD GEINDEXEERD INDIRECT
```

Het voordeel van de eerste benadering is dat vanaf het indirecte adres met Y geindexeerd kan worden. We zouden bij voorbeeld de adressen WIJSL en WIJSH als volgt indirect vanaf het adres in de geheugenlocaties PGNUL en PGNUL+1 kunnen halen:

```
LDY    #0            ;GEEN INDEXERING
LDA    (PGNUL),Y     ;HAAL INDIRECT LWB VAN ADRES
STA    WIJSL
INY                  ;HAAL INDIRECT HWB VAN ADRES
LDA    (PGNUL),Y
STA    WIJSH
```

2. Laad het indexregister X indirect vanuit het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
LDY    #0            ;GEEN INDEXERING
LDA    (PGNUL),Y     ;LAAD DE ACCUMULATOR
                     ;INDIRECT GEINDEXEERD
TAX
```

Alleen de accumulator kan door middel van de indirecte modi geladen worden, maar de inhoud ervan is gemakkelijk aan een indexregister over te dragen.

3. Laad indexregister Y indirect vanuit het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
LDX    #0            ;GEEN INDEXERING
LDA    (PGNUL,X)     ;LAAD DE ACCUMULATOR
                     ;GEINDEXEERD INDIRECT
       TAY
```

4. Laad de stack-wijzer onmiddellijk met het 8-bits getal WAARDE.

```
LDX    #WAARDE   ;INITIALISEER STACK-WIJZER
TXS
```

Slechts indexregister X kan van en naar de stack-wijzer overgedragen worden.

5. Laad de stack-wijzer direct vanuit de geheugenlocatie ADR.

```
LDX    ADR       ;INITIALISEER STACK-WIJZER
TXS
```

6. Laad het statusregister onmiddellijk met het 8-bits getal WAARDE.

```
LDA    #WAARDE   ;HAAL DE WAARDE
PHA              ;ZET DEZE OP DE STACK
PLP              ;EN IN HET STATUSREGISTER
```

7. Laad het statusregister direct vanuit de geheugenlocatie ADR.

```
LDA    ADR        ;HAAL DE WAARDE
PHA               ;ZET DEZE OP DE STACK
PLP               ;EN IN HET STATUSREGISTER
```

8. Laad indexregister vanuit de stack.

```
PLA               ;OVERDRACHT VAN STACK NAAR X, VIA A
TAX
```

Als er waarden vanuit de stack hersteld worden, moeten X en Y voor A herladen worden. Dit omdat er geen directe verbinding van de stack naar X of Y is. Het moet altijd via A.

9. Laad de geheugenlocaties PGNUL en PGNUL+1 (een pointer op pagina 0) met ADR (ADRH hoogstwaardige byte, ADRL laagstwaardige byte).

```
LDA    #ADRL      ;INITIALISEER LWB
STA    PGNUL
LDA    #ADRH      ;INITIALISEER HWB
STA    PGNUL+1
```

Er is geen eenvoudige manier om het indirecte adres te initialiseren dat op pagina 0 opgeslagen moet worden.

Opslag Instructies

1. Sla de accumulator indirect op in het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
LDY    #0           ;GEEN INDEXERING
STA    (PGNUL),Y    ;SLA INDIRECT GEINDEXEERD OP

of

LDX    #0           ;GEEN INDEXERING
STA    (PGNUL,X)    ;SLA GEINDEXEERD INDIRECT OP
```

2. Sla indexregister X indirect op in het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
LDY    #0           ;GEEN INDEXERING
TXA                 ;SLA X VIA A INDIRECT GEINDEXEERD OP
STA    (PGNUL),Y
```

3. Sla indexregister X indirect op in het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
LDX    #0           ;GEEN INDEXERING
TYA                 ;SLA X GEINDEXEERD INDIRECT VIA A OP
STA    (PGNUL,X)
```

4. Sla de stack-wijzer op in de geheugenlocatie ADR.

```
TSX                 ;SLA S VIA X OP
STX    ADR
```

5. Sla het statusregister op in geheugenlocatie ADR.

```
PHP                 ;SLA P VIA STACK EN A OP
PLA
STA    ADR
```

6. Sla het indexregister op in de stack.

```
TXA                 ;SLA X (OF Y) VIA A IN STACK OP
PHA
```

Bij het opslaan van waarden in de stack, moet A voor X en Y opgeborgen worden, omdat er geen directe verbinding tussen X of Y en de stack is.

Verplaatsinginstructies

1. Verplaats de accumulator naar het statusregister.

```
PHA                 ;OVERDRACHT VIA STACK
PLP
```

2. Verplaats het statusregister naar de accumulator.

```
PHP                 ;OVERDRACHT VIA STACK
PLA
```

3. Verplaats indexregister X naar indexregister Y.

```
TXA                 ;OVERDRACHT VIA ACCUMULATOR
TAY
```

of, zonder de accumulator te veranderen:

```
STX    TEMP         ;OVERDRACHT VIA GEHEUGEN
LDY    TEMP
```

4. Verplaats de accumulator naar de stack-wijzer.

```
TAX                 ;OVERDRACHT VIA X REGISTER
TXS
```

5. Verplaats de stack-wijzer naar de accumulator.

```
TSX                 ;OVERDRACHT VIA X REGISTER
TXA
```

6. Verplaats de inhoud van de geheugenlocaties ADR en ADR+1 (HWB in ADR+1) naar de programmateller (program counter).

```
        JMP     (ADR)       ;SPRING INDIRECT
```

Een JMP via indirecte adressering laadt de programmateller met de inhoud van de geheugenlocaties ADR en ADR+1; hij gedraagt zich eerder als een LDA met directe adressering dan een LDA met indirecte (geindexeerde) adressering.

7. Blokverplaatsing. Verplaats data van adressen beginnende bij het adres dat in de geheugenlocatie BRON en BRON+1 (op pagina 0) staat, tot het adres dat in de geheugenlocaties BEST en BEST+1 (op pagina 0) staat. Register Y bevat het aantal bytes dat verplaatst dient te worden.

```
VERPLBYT  DEY                 ;TEST AANTAL BYTES
          LDA    (BRON),Y     ;HAAL EEN BYTE VAN BRON
          STA    (DEST),Y     ;VERPLAATS HEM NAAR DE BESTEMMING
          TYA
          BNE    VERPLBYT
```

We gaan er hierbij vanuit dat de adressen elkaar niet overlappen en dat de beginwaarde van Y 1 of groter is. Hoofdstuk 5 bevat een meer algemene blokverplaatsing.
Het programma wordt eenvoudiger als we de basisadressen met 1 verlagen. Ofte wel, laat de geheugenlocaties BRON en BRON+1 een adres bevatten dat 1 kleiner is dan het laagste adres in het brongebied. Evenzo moeten de geheugenlocaties BEST en BEST+1 een adres bevatten dat 1 kleiner is dan het laagste adres in het bestemmingsgebied. Dan kunnen we stoppen als Y tot 0 is verlaagd.

```
VERPLBYT  LDA    (BRON),Y     ;HAAL EEN BYTE VAN BRON
          STA    (BEST),Y     ;EN ZET DEZE IN BESTEMMING
          DEY
          BNE    VERPLBYT     ;TEL BYTES
```

De indexwaarde 0 wordt nooit gebruikt.

8. Meervoudige verplaatsing (fill, of vollopen). Zet de inhoud van de accumulator in de geheugenlocaties beginnende bij die welke in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
VULBYT    DEY
          STA    (PGNUL),Y    ;VUL EEN BYTE
          INY
          DEY
          BNE    VULBYT       ;TEL BYTES
```

Hoofdstuk 5 bevat een algemenere versie hiervan.
Ook in dit geval kunnen we het programma vereenvoudigen door de geheugenlocaties PGNUL en PGNUL+1 een adres te laten bevatten dat 1 kleiner is dan het laagste adres in het te vullen gebied. Het gewijzigde programma ziet er dan zo uit:

```
VULBYT    STA    (PGNUL),Y    ;VUL EEN BYTE
          DEY
          BNE    VULBYT       ;TEL BYTES
```

Uitwisselingsinstructies

1. Verwissel de index registers X en Y.

```
STX    TEMP        ;BEWAAR X
TYA                ;Y NAAR X
TAX
LDY    TEMP        ;X NAAR Y
```

of

```
TXA                ;BEWAAR X
PHA
TYA                ;Y NAAR X
TAX
PLA                ;X NAAR Y
TAY
```

Beide versies gebruiken hetzelfde aantal bytes (ervan uitgaande dat TEMP een adres op pagina 0 is). De tweede is langzamer maar meer dan 1 keer toegankelijk.

2. Verwissel de geheugenlocaties ADR1 en ADR2.

```
LDA    ADR1
LDX    ADR2
STX    ADR1
STA    ADR2
```

3. Verwissel de accumulator en de stack-top.

```
TAY                ;BEWAAR A
PLA                ;HAAL STACK-TOP
TAX                ;BEWAAR STACK-TOP
TYA                ;A NAAR STACK-TOP
PHA
TXA                ;STACK-TOP NAAR A
```

Wisinstructies

1. Wis de accumulator.

```
LDA    #0
```

De 6502 behandelt 0 als elk ander getal. Er is geen speciale wisinstructie.

2. Wis een indexregister.

```
LDX    #0
```

of

```
LDY    #0
```

3. Wis de geheugenlocatie ADR.

```
LDA     #0
STA     ADR
```

Uiteraard kunnen we net zo goed X of Y in plaats van A gebruiken.

4. Wis de geheugenlocaties ADR en ADR+1.

```
LDA     #0
STA     ADR
STA     ADR+1
```

5. Wis een bit van de accumulator.

```
AND     #MASKER       ;WIS BIT DOOR MASKERING
```

MASKER heeft 0-en op de bit-posities staan die gewist moeten worden en 1-en op de bit-posities die ongewijzigd moeten blijven. Bij voorbeeld:

```
AND     #%10111110   ;WIS BITS 0 EN 6 VAN A
```

Een logische AND van een bit met 1, laat het onveranderd.

Zetinstructies

1. Zet de accumulator op FF_{16} (alleen 1-en in binair).

```
LDA     #$FF
```

2. Zet een indexregister op FF_{16}.

```
LDX     #$FF
```

of

```
LDY     #$FF
```

3. Zet de stack-wijzer op FF_{16}.

```
LDX     #$FF
TXS
```

De volgende beschikbare plaats van de stack is nu op adres $01FF_{16}$.

4. Zet een geheugenlocatie op FF_{16}.

```
LDA     #$FF
STA     ADR
```

5. Zet een bit van de accumulator.

```
ORA     #MASKER       ;ZET BIT DOOR MASKERING
```

MASKER heeft 1-en op de bit-posities die gezet moeten worden en 0-en op de andere plaatsen. Bij voorbeeld:

```
ORA     #%10000000    ;ZET BIT 7 (TEKENBIT)
```

Een logische OR van een bit met 0 laat het bit onveranderd.

VERTAK (SPRING) INSTRUCTIES

Onvoorwaardelijke vertakinstructies

1. Onvoorwaardelijke relatieve vertakking naar BEST.

```
CLC             ;OPZETTELIJK WISSEN OVERDRACHT
BCC     BEST    ;FORCEER EEN ONVOORWAARDELIJKE VERTAKKING
```

Het is altijd mogelijk een onvoorwaardelijk vertakking uit te voeren door een voorwaardelijke vertakking te doen op een voorwaarde waarvan zeker is dat deze waar is. Voor de hand liggende alternatieven zijn:

```
SEC
BCS     BEST
```

of

```
LDA     #0
BEQ     BEST
```

of

```
LDA     #1
BNE     DEST
```

2. Spring indirect naar het adres boven in de stack.

```
RTS
```

RTS is een gewone indirecte sprong waarbij de processor het bestemmingsadres van boven uit de stack haalt. Houd er echter rekening mee dat de processor 1 bij het adres optelt vooraleer hij verder gaat.

3. Spring geindexeerd, ervan uit gaande dat de basis van de adrestabel BASIS is en de index in geheugenlocatie INDEX staat. De adressen staan in de voor de 6502 gebruikelijke manier, gerangschikt met de laagstwaardige byte eerst.

. Met indirecte adressering:

```
LDA     INDEX
ASL     A           ;VERDUBBEL INDEX VOOR 2 BYTES GROTE
                    ;GEGEVENS
TAX
LDA     BASIS,X     ;HAAL LWB VAN BESTEMMING
STA     INDIR
INX
LDA     BASIS,X     ;HAAL HWB VAN BESTEMMING
STA     INDIR+1
JMP     (INDIR)     ;SPRING INDIRECT NAAR BESTEMMING
```

. Met gebruik van de stack:

```
LDA     INDEX
ASL     A           ;VERDUBBEL INDEX VOOR 2 BYTES GROTE
                    ;GEGEVENS
TAX
LDA     BASIS+1,X   ;HAAL HWB VAN BESTEMMING
PHA
LDA     BASIS,X     ;HAAL LWB VAN BESTEMMING
PHA
RTS                 ;SPRING INDIRECT NAAR BESTEMMING, OFFSET
                    ;1
```

De tweede methode is sneller maar minder duidelijk. Let op het volgende:

1. De laagstwaardige byte moet het eerst op de stack gezet worden, aangezien deze naar een lager adres groeit. Dan komen de bytes in de gebruikelijke volgorde terecht.

2. Omdat RTS 1 bij de programmateller optelt nadat deze vanuit de stack is geladen, moeten de gegevens in de tabel, om deze methode goed te laten werken, allemaal 1 lager zijn dan het werkelijke bestemmingsadres.

3. Documentatie is van levensbelang, omdat bij deze techniek de RTS gebruikt om de controle aan een subroutine over te dragen, terwijl zij in het algemeen juist wordt gebruikt om een subroutine te stoppen. De mnemonic kan de lezer verwarren, maar uiteraard niet de microprocessor.

Voorwaardelijke vertakkingsinstructies.

1. Vertak indien nul.

. Vertak als de accumulator nul bevat.

```
TAX                 ;TEST DE ACCUMULATOR
BEQ     BEST
```

of

```
CMP     #0          ;TEST DE ACCUMULATOR
BEQ     BEST
```

Zowel AND #$FF als ORA #0 zullen de nulvlag zetten als (A) = 0, zonder de overdrachtvlag te beinvloeden. Een CMP #0 zet daarentegen wel de overdrachtvlag.

. Vertak als een indexregister 0 bevat.

```
CPX    #0      ;TEST INDEXREGISTER
BEQ    BEST
```

De instructie TXA of de reeks INX, DEX kunnen gebruikt worden om de inhoud van indexregister X te testen, zonder de overdrachtvlag te zetten (hetgeen CPX #0 wel doet). TXA verandert uiteraard wel de inhoud van de accumulator.

. Vertak als een geheugenlocatie 0 bevat.

```
INC    ADR          ;TEST GEHEUGENLOCATIE
DEC    ADR
BEQ    DEST
```

of

```
LDA    ADR          ;TEST GEHEUGENLOCATIE
BEQ    DEST
```

. Vertak als een paar geheugenlocaties (ADR en ADR+1) beide 0 bevatten.

```
LDA    ADR          ;TEST 16-BITS GETAL OP NUL
ORA    ADR+1
BEQ    DEST
```

. Vertak als een bit van de accumulator nul is.

```
AND    #MASKER    ;TEST BIT VAN DE ACCUMULATOR
BEQ    BEST
```

MASKER heeft een 1 op de bit-positie die getest moet worden en 0-en op de andere plaatsen. Merk de inversie hier op; als het bit van de accumulator 0, is het resultaat 0 en wordt de nulvlag op 1 gezet. Speciale gevallen zijn

Bit-positie 7

```
ASL    A          ;ZET BIT 7 IN OVERDRACHT
BCC    BEST
```

Bit-positie 6

```
ASL    A          ;ZET BIT 6 IN NEGATIEFVLAG
BPL    BEST
```

Bit-positie 0

```
LSR    A          ;ZET BIT 0 IN OVERDRACHT
BCC    BEST
```

. Vertak als een bit van een geheugenlocatie 0 is.

```
LDA     #MASKER
BIT     ADR         ;TEST BIT VAN GEHEUGEN
BEQ     DEST
```

MASKER heeft een 1 op de te testen bit-positie en 0-en op de andere plaatsen. Speciale gevallen zijn:

Bit-positie 7

```
BIT     ADR         ;TEST GEHEUGEN
BPL     BEST        ;VERTAK OP BIT 7
```

Bit-positie 6

```
BIT     ADR         ;TEST GEHEUGEN
BVC     BEST        ;VERTAK OP BIT 6
```

De BIT-instructie zet de negatiefvlag naar bit 7 van de geheugenlocatie en de overschotvlag volgens bit 6, ongeacht de inhoud van de accumulator.
We kunnen ook de schuifinstructies gebruiken om de bits aan de uiteinden van de byte te testen. Dit uiteraard op de voorwaarde dat de geheugenlocaties van inhoud mogen veranderen.

Bit-positie 7

```
ASL     ADR         ;TEST BIT 7
BCC     BEST
```

Bit-positie 6

```
ASL     ADR         ;TEST BIT 6
BPL     BEST
```

Bit-positie 0

```
LSR     ADR         ;TEST BIT 0
BCC     BEST
```

. Vertak als de Interrupt Disable (interrupt uitschakel) vlag (bit 2 van het statusregister) 0 is.

```
PHP                 ;STATUS NAAR A
PLA
AND     #%00000100  ;TEST INTERRUPT DISABLE
BEQ     BEST        ;VERTAK ALS INTERRUPTS AAN STAAN
```

. Vertak als de Decimale Modus vlag (bit 3 van het statusregister) 0 is.

```
PHP                 ;STATUS NAAR A
PLA
AND     #%00001000  ;TEST DECIMALE MODUS VLAG
BEQ     BEST        ;VERTAK ALS MODUS BINAIR IS
```

2. Vertak indien ongelijk 0.

. Vertak als de accumulator geen 0 bevat

```
TAX                 ;TEST ACCUMULATOR
BNE     BEST
```

of

```
CMP     #0          ;TEST ACCUMULATOR
BNE     BEST
```

. Vertak als een indexregister geen 0 bevat.

```
CPX     #0          ;TEST INDEXREGISTER
BNE     BEST
```

. Vertak als een geheugenlocatie geen 0 bevat.

```
INC     ADR         ;TEST GEHEUGENLOCATIE
DEC     ADR
BNE     BEST
```

of

```
LDA     ADR         ;TEST GEHEUGENLOCATIE
BNE     BEST
```

. Vertak als een paar geheugenlocaties (ADR en ADR+1) niet beide 0 zijn.

```
LDA     ADR         ;TEST 16-BITS GETAL OP NUL
ORA     ADR+1
BNE     BEST
```

. Vertak als een bit van de accumulator 1 is.

```
AND     #MASKER     ;TEST BIT VAN ACCUMULATOR
BNE     BEST
```

MASKER heeft een 1 op de te testen bit-positie en 0 op alle andere plaatsen. Merk de inversie hier op; als het bit van de accumulator 1 is, is het resultaat geen 0 en wordt de nulvlag op 0 gezet. Speciale gevallen zijn:

Bit-positie 7

```
ASL     A           ;VERPLAATS BIT 7 NAAR OVERDRACHT
BCS     BEST        ;EN TEST OVERDRACHT
```

Bit-positie 6

```
ASL     A           ;ZET BIT 6 IN TEKEN
BMI     BEST        ;EN TEST TEKEN
```

Bit-positie 0

```
LSR    A          ;ZET BIT 0 IN OVERDRACHT
BCS    BEST       ;EN TEST OVERDRACHT
```

. Vertak als een bit van een geheugenlocatie 1 is.

```
LDA    #MASKER
BIT    ADR        ;TEST BIT IN GEHEUGEN
BNE    BEST
```

MASKER heeft een 1 op de bit-positie die getest moet worden en 0-en op alle andere plaatsen. Speciale gevallen zijn:

Bit-positie 7

```
BIT    ADR        ;TEST BIT 7 VAN GEHEUGEN
BMI    BEST
```

Bit-positie 6

```
BIT    ADR        ;TEST BIT 6 VAN GEHEUGEN
BVS    BEST
```

De BIT-instructie zet de negatiefvlag volgens bit 7 van de geheugenlocatie en de overschotvlag volgens bit 6, ongeacht de inhoud van de accumulator.
We kunnen ook de schuifinstructies gebruiken om de bits aan beide einden van de byte te testen, zolang wijzigingen van de inhoud van de geheugenlocaties zijn toegestaan.

Bit-positie 7

```
ASL    ADR        ;TEST BIT 7 VAN GEHEUGEN
BCS    BEST
```

Dit alternatief is 2 klokcycli langzamer dan BIT, omdat het resultaat terug in het geheugen gezet moet worden.

Bit-positie 6

```
ASL    ADR        ;TEST BIT 6 VAN GEHEUGEN
BMI    BEST
```

Bit-positie 0

```
LSR    ADR        ;TEST BIT 0 VAN GEHEUGEN
BCS    BEST
```

. Vertak als de Interrupt Disable vlag (bit 2 van het statusregister) 1 is.

```
PHP                  ;BEWAAR STATUS, VIA STACK, IN A
PLA
AND    #%00000100    ;TEST INTERRUPT DISABLE
```

```
        BNE     BEST            ;VERTAK ALS DE INTERRUPTS
                                ;UITGESCHAKELD ZIJN
```

. Vertak als de Decimale Modus vlag (bit 3 van het statusregister) 1 is.

```
        PHP                     ;BEWAAR STATUS, VIA STACK, IN A
        AND     #%00001000      ;TEST DECIMALE MODUS VLAG
        BNE     BEST            ;VERTAK ALS MODUS DECIMAAL IS
```

3. Vertak indien gelijk.

. Vertak als (A) = WAARDE.

```
        CMP     #WAARDE     ;VERGELIJK DOOR AFTREKKING
        BEQ     BEST
```

. Vertak als (X) = WAARDE.

```
        CPX     #WAARDE     ;VERGELIJK DOOR AFTREKKING
        BEQ     BEST
```

Twee speciale gevallen zijn:

. Vertak als (X) = 1

```
        DEX
        BEQ     BEST
```

. Vertak als (X) = FF_{16}

```
        INX
        BEQ     BEST
```

. Vertak als (A) = (ADR).

```
        CMP     ADR         ;VERGELIJK DOOR AFTREKKING
        BEQ     BEST
```

. Vertak als (X) = (ADR).

```
        CPX     ADR         ;VERGELIJK DOOR AFTREKKING
        BEQ     BEST
```

. Vertak als de inhoud van de geheugenlocaties PGNUL en PGNUL+1 gelijk is aan WAARDE16 (WAARDE16L laagstwaardige byte, WAARDE16H hoogstwaardige byte).

```
        LDA     PGNUL+1         ;VERGELIJK HWB'S
        CMP     #WAARDE16H
        BNE     KLAAR
        LDA     PGNUL           ;EN LWB'S ALLEEN INDIEN NODIG
        CMP     #WAARDE16L
        BEQ     BEST
KLAAR   NOP
```

. Vertak als de inhoud van geheugenlocaties PGNUL en PGNUL+1 gelijk is aan die van de inhoud van de geheugenlocatie LIML en LIMH.

```
        LDA     PGNUL+1     ;VERGELIJK HWB'S
        CMP     LIMH
        BNE     KLAAR
        LDA     PGNUL       ;EN LWB'S ALLEEN INDIEN NODIG
        CMP     LIML
        BEQ     BEST
KLAAR   NOP
```

Let op: Geen van beide programmadelen kan worden gebruikt om te testen of de stack over- of onderloopt. Dit omdat tussenkomende instructies (een enkele JSR of RTS, bij voorbeeld) de stack-wijzer met meer dan 1 kunnen wijzigen.

. Vertak als (S) = WAARDE.

```
        TSX                 ;CONTROLEER OF STACK AAN ZIJN LIMIET IS
        CPW     #WAARDE
        BEQ     BEST
```

. Vertak als (S) = (ADR).

```
        TSX                 ;CONTROLEER OF STACK AAN ZIJN LIMIET IS
        CPX     ADR
        BEQ     BEST
```

4. Vertak indien ongelijk.

. Vertak als (A) <> WAARDE.

```
        CMP     #WAARDE     ;VERGELIJK DOOR AFTREKKING
        BNE     BEST
```

. Vertak als (X) <> WAARDE.

```
        CPX     #WAARDE     ;VERGELIJK DOOR AFTREKKING
        BNE     BEST
```

Twee speciale gevallen zijn:

. Vertak als (X) <> 1.

```
        DEX
        BNE     BEST
```

. Vertak als (X) <> FF_{16}.

```
        INX
        BNE     BEST
```

. Vertak als (A) <> (ADR).

```
CMP     ADR         ;VERGELIJK DOOR AFTREKKING
BNE     BEST
```

. Vertak als (X) <> (ADR).

```
CPX     ADR         ;VERGELIJK DOOR AFTREKKING
BNE     BEST
```

. Vertak als de inhoud van geheugenlocaties PGNUL en PGNUL+1 ongelijk is aan WAARDE16 (WAARDE16L laagstwaarddige byte, WAARDE16H hoogstwaardige byte).

```
LDA     PGNUL+1       ;VERGELIJK HWB'S
CMP     #WAARDE16H
BNE     BEST
LDA     PGNUL         ;EN LWB'S ALLEEN INDIEN NODIG
CMP     #WAARDE16L
BNE     BEST
```

. Vertak als de inhoud van de geheugenlocaties PGNUL en PGNUL+1 ongelijk is aan die van de geheugenlocaties LIML en LIMH.

```
LDA     PGNUL+1     ;VERGELIJK HWB'S
CMP     LIMH
BNE     BEST
LDA     PGNUL       ;EN LWB'S ALLEEN INDIEN NODIG
CMP     LIML
BNE     BEST
```

Let op: Geen van de twee routines hierboven mag gebruikt worden voor het testen van het over- of onderlopen van de stack. Dit omdat tussenkomende instructies (een enkele JSR of RTS, bij voorbeeld) de stack-wijzer met meer dan 1 kunnen wijzigen.

. Vertak als (S) <> WAARDE.

```
TSX                 ;CONTROLEER OF STACK AAN LIMIET IS
CPX     #WAARDE
BNE     BEST
```

. Vertak als (S) <> (ADR).

```
TSX                 ;CONTROLEER OF STACK AAN LIMIET IS
CPX     ADR
BNE     BEST
```

5. Vertak indien positief.

. Vertak als de inhoud van de accumulator positief is.

```
TAX                 ;TEST ACCUMULATOR
BPL     BEST
```

of

```
CMP     #0          ;TEST ACCUMULATOR
BPL     BEST
```

. Vertak als de inhoud van indexregister X positief is.

```
TXA                 ;TEST INDEXREGISTER X
BPL     BEST
```

of

```
CPX     #0          ;TEST INDEXREGISTER X
BPL     BEST
```

. Vertak als de inhoud van een geheugenlocatie positief is.

```
LDA     ADR         ;TEST GEHEUGENLOCATIE
BPL     BEST
```

of

```
BIT     ADR
BPL     BEST
```

. Vertak als het 16-bits getal in geheugenlocaties ADR en ADR+1 (HWB in ADR+1) positief is.

```
BIT     ADR+1       ;TEST HWB
BPL     BEST
```

Onthoud dat BIT de negatiefvlag zet naar bit 7 van de geheugenlocatie, ongeacht de inhoud van de accumulator.

6. Vertak indien negatief.

. Vertak als de inhoud van de accumulator negatief is.

```
TAX                 ;TEST ACCUMULATOR
BMI     BEST
```

of

```
CMP     #0          ;TEST ACCUMULATOR
BMI     BEST
```

. Vertak als de inhoud van indexregister X negatief is.

```
TXA                 ;TEST INDEX REGISTER X
BMI     BEST
```

of

```
CPX     #0          ;TEST INDEX REGISTER X
```

```
        BMI     BEST
```

. Vertak als de inhoud van een geheugenlocatie negatief is.

```
        BIT     ADR         ;TEST EEN GEHEUGENLOCATIE
        BMI     BEST
```

of

```
        LDA     ADR         ;TEST EEN GEHEUGENLOCATIE
        BMI     BEST
```

. Vertak als het 16-bits getal in de geheugenlocaties ADR en ADR+1 (HWB in ADR+1) negatief is.

```
        BIT     ADR+1       ;TEST HWB
        BMI     BEST
```

Onthoud dat een BIT de negatiefvlag naar bit 7 van de geheugenlocatie zet, ongeacht de inhoud van de accumulator.

7. Vertak indien groter dan (getekend).

. Vertak als (A) > WAARDE.

```
        CMP     #WAARDE     ;VERGELIJK DOOR AFTREKKING
        BEQ     KLAAR       ;GEEN VERTAKKING INDIEN GELIJK
        BVS     TSTOVER     ;WAS ER EEN OVERSCHOT?
        BPL     BEST        ;NEE, VERTAK DAN OP POSITIEF
        BMI     KLAAR
TSTOVER BMI     BEST        ;JA, VERTAK DAN OP NEGATIEF
KLAAR   NOP
```

Het idee hier is om te vertakken als het resultaat groter is dan nul er geen overschot optrad, of als het resultaat kleiner is dan nul en er wel een overschot optrad. Een overschot zorgt er voor dat het schijnbare teken het tegenovergestelde van het echte teken wordt.

. Vertak als (A) > (ADR).

```
        CMP     ADR         ;VERGELIJK DOOR AFTREKKING
        BEQ     KLAAR       ;GEEN VERTAKKING ALS GELIJK
        BVS     TSTOVER     ;WAS ER EEN OVERSCHOT?
        BPL     BEST        ;NEE, VERTAK DAN OP POSITIEF
        BMI     KLAAR
TSTOVER BMI     BEST        ;JA, VERTAK DAN OP NEGATIEF
KLAAR   NOP
```

8. Vertak indien groter dan of gelijk aan (getekend)

. Vertak als (A) >= WAARDE.

```
        CMP     #WAARDE     ;VERGELIJK DOOR AFTREKKING
        BVS     TSTOVER     ;WAS ER EEN OVERSCHOT?
```

```
          BPL    BEST       ;NEE, VERTAK DAN OP POSITIEF
          BMI    KLAAR
TSTOVER   BMI    BEST       ;JA, VERTAK DAN OP NEGATIEF
KLAAR     NOP
```

Hier is het idee om te vertakken als het resultaat groter dan of gelijk aan 0 is en er geen sprake was van overschot, of als het resultaat kleiner is dan 0 en er wel sprake was van overschot.

. Vertak als (A) >= (ADR).

```
          CMP    ADR        ;VERGELIJK DOOR AFTREKKING
          BVS    TSTOVER    ;WAS ER EEN OVERSCHOT?
          BPL    BEST       ;NEE, VERTAK DAN OP POSITIEF
          BMI    KLAAR
TSTOVER   BMI    BEST       ;JA, VERTAK DAN OP NEGATIEF
KLAAR     NOP
```

9. Vertak indien kleinder dan (getekend)

. Vertak als (A) < WAARDE (getekend).

```
          CMP    #WAARDE    ;VERGELIJK DOOR AFTREKKING
          BVS    TSTOVER    ;WAS ER EEN OVERSCHOT?
          BMI    BEST       ;JA, VERTAK DAN OP NEGATIEF
          BPL    KLAAR
TSTOVER   BPL    BEST       ;JA, VERTAK DAN OP POSITIEF
KLAAR     NOP
```

Het idee is hier om te vertakken als het resultaat negatief is en er geen sprake was van overschot, of als het resultaat positief is en er wel sprake was van overschot.

. Vertak als (A) < (ADR) (getekend).

```
          CMP    ADR        ;VERGELIJK DOOR AFTREKKING
          BVS    TSTOVER    ;WAS ER EEN OVERSCHOT?
          BMI    BEST       ;NEE, VERTAK DAN OP NEGATIEF
          BPL    KLAAR
TSTOVER   BPL    BEST       ;JA, VERTAK DAN OP POSITIEF
KLAAR     NOP
```

10. Vertak indien kleiner dan of gelijk (getekend).

. Vertak als (A) <= WAARDE (getekend).

```
          CMP    #WAARDE    ;VERGELIJK DOOR AFTREKKING
          BEQ    BEST       ;VERTAK ALS GELIJK
          BVS    TSTOVER    ;WAS ER EEN OVERSCHOT?
          BMI    BEST       ;NEE, VERTAK DAN OP NEGATIEF
          BPL    KLAAR
TSTOVER   BPL    BEST       ;JA, VERTAK DAN OP POSITIEF
KLAAR     NOP
```

Het idee is hier te vertakken als het resultaat 0 is, negatief zonder overschot of positief met overschot.

. Vertak als (A) <= (ADR) (getekend).

```
          CMP    ADR        ;VERGELIJK DOOR AFTREKKING
          BEQ    BEST       ;VERTAK INDIEN GELIJK
          BVS    TSTOVER    ;WAS ER EEN OVERSCHOT?
          BMI    BEST       ;NEE, VERTAK DAN OP NEGATIEF
          BPL    KLAAR
TSTOVER   BPL    BEST       ;JA, VERTAK DAN OP POSITIEF
KLAAR     NOP
```

11. Vertak indien groter (ongetekend). Ofte wel, vertak als de ongetekende vergelijking geen nul is en geen lening nodig heeft.

. Vertak als (A) > WAARDE (ongetekend).

```
          CMP    #WAARDE    ;VERGELIJK DOOR AFTREKKING
          BEQ    KLAAR      ;GEEN VERTAKKING ALS GELIJK
          BCS    BEST       ;VERTAK ALS ER GEEN LENING NODIG WAS
KLAAR     NOP
```

of

```
          CMP    #WAARDE+1    ;VERGELIJK DOOR AFTREKKING WAARDE + 1
          BCS    BEST         ;VERTAK ALS ER GEEN LENING NODIG WAS
```

Het is korter en iets efficienter om eenvoudigweg met een getal te vergelijken dat 1 groter is dan de werkelijke drempelwaarde. Dan kunnen we BCS gebruiken, hetgeen een vertakking veroorzaakt als de inhoud van de accumulator groter dan of gelijk aan WAARDE + 1 (ongetekend) is.

. Vertak als (A) > (ADR) (ongetekend).

```
          CMP    ADR        ;VERGELIJK DOOR AFTREKKING
          BEQ    KLAAR      ;GEEN VERTAKKING ALS GELIJK
          BCS    BEST       ;VERTAK ALS GEEN LENING NODIG WAS
KLAAR     NOP
```

. Vertak als (X) > WAARDE (ongetekend).

```
          CPX    #WAARDE+1    ;VERGELIJK DOOR AFTREKKING WAARDE + 1
          BCS    BEST
```

. Vertak als (X) > (ADR) (ongetekend).

```
          CPX    ADR        ;VERGELIJK DOOR AFTREKKING
          BEQ    KLAAR      ;GEEN VERTAKKING ALS GELIJK
          BCS    BEST       ;VERTAK ALS ER GEEN LENING NODIG WAS
KLAAR     NOP
```

. Vertak als de inhoud van geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1) groter is (ongetekend) dan WAARDE16 (WAARDE16L laagstwaardige byte, WAARDE16H hoogstwaardige byte).

```
        LDA     #WAARDE16L   ;WEK LENING OP DOOR LWB'S TE
                             ;VERGELIJKEN
        CMP     PGNUL
        LDA     #WAARDE16H   ;VERGELIJK HWB'S MET LEENBIT
        SBC     PGNUL+1
        BCC     BEST         ;VERTAKKING ALS ER GELEEND IS
```

. Vertak als de inhoud van de geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1) groter is (ongetekend) dan de inhoud van geheugenlocaties LIML en LIMH (HWB in LIMH).

```
        LDA     LIML       ;WEK LENING OP DOOR LWB'S TE VERGELIJKEN
        CMP     PGNUL
        LDA     LIMH       ;VERGELIJK HWB'S MET LEENBIT
        SBC     PGNUL+1
        BCC     BEST       ;VERTAK ALS ER EEN LENING NODIG WAS
```

. Vertak als (S) > WAARDE (ongetekend).

```
        TSX                ;CONTROLEER OF STACK BUITEN LIMIET IS
        CPX     #WAARDE
        BEQ     KLAAR      ;NEE, VERTAK ALS GELIJK
        BCS     BEST       ;VERTAK ALS ER GEEN LENING NODIG WAS
KLAAR   NOP
```

of

```
        TSX                   ;CONTROLEER OF STACK BUITEN LIMIET IS
        CPX     #WAARDE+1     ;VERGELIJK DOOR AFTREKKING WAARDE + 1
        BCS     BEST          ;VERTAK ALS ER GEEN LENING NODIG WAS
```

. Vertak als (S) > (ADR) (ongetekend).

```
        TSX                ;CONTROLEER OF STACK BUITEN LIMIET IS
        BEQ     KLAAR      ;GEEN VERTAKKING ALS GELIJK
        BCS     BEST       ;VERTAK ALS ER GEEN LENING NODIG WAS
KLAAR   NOP
```

12. Vertak indien niet groter (ongetekend). Vertak als de ongetekende vergelijking 0 is of een lening nodig heeft.

. Vertak als (A) <= WAARDE (ongetekend).

```
        CMP     #WAARDE    ;VERGELIJK DOOR AFTREKKING
        BCC     BEST       ;VERTAK ALS ER EEN LENING NODIG WAS
        BEQ     BEST       ;VERTAK ALS GELIJK
```

Als de twee waarden gelijk zijn, zorgt CMP ervoor dat de Overdracht aangeeft dat er geen lening nodig was.

of

```
CMP     #WAARDE+1     ;VERGELIJK DOOR AFTREKKING WAARDE + 1
BCC     BEST          ;VERTAK ALS ER EEN LENING NODIG WAS
```

. Vertak als (A) <= (ADR) (ongetekend).

```
CMP     ADR         ;VERGELIJK DOOR AFTREKKING
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
BEQ     BEST        ;VERTAK ALS GELIJK
```

. Vertak als (A) <= WAARDE (ongetekend).

```
CPX     #WAARDE     ;VERGELIJK DOOR AFTREKKING
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
BEQ     BEST        ;VERTAK ALS GELIJK
```

of

```
CPX     #WAARDE+1     ;VERGELIJK DOOR AFTREKKING WAARDE + 1
BCC     BEST          ;VERTAK ALS ER EEN LENING NODIG WAS
```

. Vertak als (X) <= (ADR) (ongetekend).

```
CPX     ADR         ;VERGELIJK DOOR AFTREKKING
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
BEQ     BEST        ;VERTAK ALS GELIJK
```

. Vertak als de inhoud van de geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1) kleiner of gelijk is aan (ongetekend) de WAARDE16 (WAARDE16H hoogstwaardigde byte, WAARDE16L laagstwaardige byte).

```
LDA     #WAARDE16L    ;WEK LENING OP DOOR LWB'S TE
                      ;VERGELIJKEN
CMP     PGNUL
LDA     #WAARDE16H    ;VERGELIJK HWB'S MET LENING
SBC     PGNUL+1
BCS     BEST          ;VERTAK ALS ER GEEN LENING IS
                      ;GEGENEREERD
```

. Vertak als de inhoud van de geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1) kleiner of gelijk is aan (ongetekend) de inhoud van de geheugenlocaties LIML en LIMH (HWB in LIMH).

```
LDA     LIML        ;WEK LENING OP DOOR LWB'S TE VERGELIJKEN
CMP     PGNUL
LDA     LIMH        ;VERGELIJK HWB'S MET LENING
SBC     PGNUL+1
BCS     BEST        ;VERTAK ALS ER GEEN LENING IS
                    ;GEGENEREERD
```

. Vertak als (S) <= WAARDE (ongetekend).

```
TSX                 ;TEST OF STACK OP OF ONDER LIMIET IS
CPX     #WAARDE
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
BEQ     BEST        ;VERTAK ALS GELIJK
```

of

```
TSX                     ;TEST OF STACK OP OF ONDER LIMIET IS
CPX     #WAARDE+1       ;VERGELIJK DOOR AFTREKKING WAARDE + 1
BCC     BEST
```

. Vertak als (S) <= (ADR) (ongetekend).

```
TSX                 ;TEST OF STACK OP OF ONDER LIMIET IS
CPX     ADR
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
BEQ     BEST        ;VERTAK ALS GELIJK
```

13. Vertak indien kleiner (ongetekend). Ofte wel, vertak als de ongetekende vergelijking een lening nodig heeft.

. Vertak als (A) < WAARDE (ongetekend).

```
CMP     #WAARDE     ;VERGELIJK DOOR AFTREKKING
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
```

De overdrachtvlag wordt op 0 gezet als de aftrekking een lening nodig heeft.

. Vertak als (A) < (ADR) (ongetekend).

```
CMP     ADR          ;VERGELIJK DOOR AFTREKKING
BCC     BEST         ;VERTAK ALS ER EEN LENING NODIG WAS
```

. Vertak als (X) < WAARDE (ongetekend).

```
CPX     WAARDE      ;VERGELIJK DOOR AFTREKKING
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
```

. Vertak als (X) < (ADR) (ongetekend).

```
CPX     ADR         ;VERGELIJK DOOR AFTREKKING
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
```

. Vertak als de inhoud van geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1) kleiner is dan (ongetekend) WAARDE16 (WAARDE16L laagstwaardige byte, WAARDE16H hoogstwaardige byte).

```
LDA     PGNUL           ;WEK LENING OP DOOR LWB'S TE
                        ;VERGELIJKEN
CMP     #WAARDE16L
LDA     PGNUL+1         ;VERGELIJK HWB'S MET LENING
SBC     #WAARDE16H
BCC     BEST            ;VERTAK ALS ER EEN LENING NODIG WAS
```

. Vertak als de inhoud van de geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1) kleiner is dan (ongetekend) de inhoud van geheugenlocaties LIML en LIMH (HWB in LIMH).

```
LDA     PGNUL       ;WEK LENING OP DOOR LWB'S TE VERGELIJKEN
CMP     LIML
LDA     PGNUL+1     ;VERGELIJK HWB'S MET LENING
SBC     LIMH
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
```

. Vertak als (S) < WAARDE (ongetekend).

```
TSX                 ;TEST OF STACK ONDER LIMIET IS
CPX     #WAARDE
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
```

. Vertak als (S) < (ADR) (ongetekend).

```
TSX                 ;TEST OF STACK ONDER LIMIET IS
CPX     ADR
BCC     BEST        ;VERTAK ALS ER EEN LENING NODIG WAS
```

14. Vertak indien niet kleiner (ongetekend). Ofte wel, vertak als de ongetekende vergelijking geen lening nodig heeft.

. Vertak als (A) >= WAARDE (ongetekend).

```
CMP     #WAARDE     ;VERGELIJK DOOR AFTREKKING
BCS     BEST        ;VERTAK ALS ER GEEN LENING NODIG WAS
```

De overdrachtvlag wordt op 1 gezet als de aftrekking geen lening genereert.

. Vertak als (A) >= (ADR) (ongetekend).

```
CMP     ADR         ;VERGELIJK DOOR AFTREKKING
BCS     BEST
```

. Vertak als (X) >= WAARDE (ongetekend).

```
CPX     #WAARDE     ;VERGELIJK DOOR AFTREKKING
BCS     BEST        ;VERTAK ALS ER GEEN LENING NODIG WAS
```

. Vertak als (X) >= (ADR) (ongetekend).

```
CPX     ADR         ;VERGELIJK DOOR AFTREKKING
BCS     BEST
```

. Vertak als de inhoud van geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1) groter dan of gelijk aan de (ongetekende) WAARDE16 (WAARDE16L laagstwaardige byte, WAARDE16H hoogstwaardige byte) is.

```
LDA     PGNUL           ;WEK LENING OP DOOR LWB'S TE
                        ;VERGELIJKEN
```

```
CMP     #WAARDE16L
LDA     PGNUL+1       ;VERGELIJK HWB'S MET LENING
SBC     #WAARDE16H
BCS     BEST          ;VERTAK ALS ER GEEN LENING NODIG WAS
```

. Vertak als de inhoud van de geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1) groter of gelijk is aan (ongetekend) de inhoud van de geheugenlocaties LIML en LIMH (HWB in LIMH).

```
LDA     PGNUL      ;WEK LENING OP DOOR LWB'S TE VERGELIJKEN
CMP     LIML
LDA     PGNUL+1    ;VERGELIJK HWB'S MET LENING
SBC     LIMH
BCS     BEST       ;VERTAK ALS ER GEEN LENING NODIG WAS
```

. Vertak als (S) >= WAARDE (ongetekend).

```
TSX                ;TEST STACK OP OF BOVEN LIMIET IS
CPX     #WAARDE
BCS     BEST       ;VERTAK ALS ER GEEN LENING NODIG WAS
```

. Vertak als (S) >= (ADR) (ongetekend).

```
TSX                 ;TEST OF DE STACK OP OF BOVEN LIMIET IS
CPX     ADR
BCS                 ;VERTAK ALS ER GEEN LENING NODIG WAS
```

SKIP-INSTRUCTIES

Het is mogelijk op de 6502 microprocessor instructies te implementeren die delen overslaan, door gebruik te maken van vertak of spring instructies met de juiste bestemming. Die bestemming moet een instructie voorbij diegene zijn die de processor meteen na de vertakking zou uitvoeren. Skip-instructies zijn op de meeste microprocessoren lastig te gebruiken, omdat de instructies daarvan verschillen in lengte, en het zodoende moeilijk is om te bepalen hoe lang een sprongopdrachtmoet worden om een instuctie over te slaan.

INSTRUCTIES OM SUBROUTINES AAN TE ROEPEN

Onvoorwaardelijke aanroep instructies

Het is mogelijk op de 6502 microprocessor een routine indirect aan te roepen door het aanroepen van een subroutine die een gewone indirecte sprong uitvoert. Een RETURN FROM SUBROUTINE (keert terug van subroutine) (RTS) instructie aan het einde van de subroutine zal dan de controle teruggeven aan het oorspronkelijke aanroeppunt. Het hoofdprogramma voert een

```
JSR     OVRDRCHT
```

uit waarbij OVRDRCHT de subroutine is die de controle daadwerkelijk overdraagt door middel van een sprongopdracht. Zorg ervoor dat OVRDRCHT met een sprong en niet met een RTS eindigt. Typische voorbeelden van een OVRDRCHT routine zijn:

. Naar het adres in de geheugenlocaties INDIR en INDIR+1 (HWB in INDIR+1).

```
        JMP     (INDIR)
```

. Naar het adres in de tabel die begint bij geheugenlocatie BASIS, gebruik makend van de index in geheugenlocatie INDEX.

```
        LDA     INDEX
        ASL     A           ;VERDUBBEL INDEX VOOR 2-BYTES GROTE
                            ;GEGEVENS
        TAX
        LDA     BASIS,X     ;HAAL LWB VAN BESTEMMING
        STA     INDIR
        INX
        LDA     BASIS,X     ;HAAL HWB VAN BESTEMMING
        STA     INDIR+1
        JMP     (INDIR)
```

of

```
        LDA     INDEX
        ASL     A               ;VERDUBBEL INDEX VOOR 2-BYTES GROTE
                                ;GEGEVENS
        TAX
        LDA     BASIS+1,X       ;HAAL HWB VAN BESTEMMING
        PHA
        LDA     BASIS,X         ;HAAL LWB VAN BESTEMMING
        PHA
        RTS                     ;SPRING NAAR BESTEMMING PLUS 1
```

In het tweede geval moet de tabel de feitelijke adressen minus 1 bevatten, aangezien een RTS een 1 bij de programmateller optelt nadat deze van de stack is geladen.

Voorwaardelijke aanroep instructies

Hoewel deze niet direct aanwezig is, kunt u een voorwaardelijk aanroep instructie op de 6502 processor implementeren door op de tegenovergestelde voorwaarde te vertakken rond de aanroep. De functie Aanroep Bij Lege Overdracht (Call On Carry Clear) zou door de volgende reeks verkregen kunnen worden:

```
        BCS     VERDER      ;VERTAK VOORBIJ AANROEP ALS OVERDRACHT
                            ;IS GEZET
        JSR     SUBR        ;AANROEP ALS OVERDRACHT LEEG IS
VERDER  NOP
```

TERUGKEERINSTRUCTIES

Onvoorwaardelijk terugkeerinstructies

De RTS-instructie geeft automatisch de controle terug aan het adres (+1) dat boven in de stack is opgeslagen. Als het terugkeeradres ergens anders is opgeslagen (in twee geheugenlocaties, bij voorbeeld) kan hier naartoe teruggekeerd worden door een indirecte sprong. Let op dat er 1 bij het terugkeeradres moet worden opgeteld om een RTS te simuleren.
De volgende routine popt het terugkeeradres uit de top van de stack, telt er 1 bij op en bergt de aangepaste waarde op in de geheugenlocaties TERADR en TERADR+1.

```
        PLA                 ;POP LWB VAN TERUGKEERADRES
        CLC                 ;TEL 1 BIJ LWB
        ADC     #1
        STA     TERADR
        PLA                 ;POP HWB VAN TERUGKEERADRES
        ADC     #0          ;EN OVERDRACHT NAAR HWB
        STA     TERADR+1
```

Een besluitende JMP (TERADR) draagt de controle nu over aan de goede locatie.

Voorwaardelijke terugkeerinstructies

Het is mogelijk een voorwaardelijke terugkeer te implementeren door voorwaardelijke vertakkingen (op de tegenovergestelde voorwaarde) om een RTS heen te laten wijzen. Een RETURN ON NOT ZERO (terugkeer bij ongelijk nul) is op de volgende manier te bewerkstelligen:

```
        BEQ     VERDER      ;EROMHEEN INDIEN NUL
        RTS                 ;TERUGKEER BIJ ONGELIJK NUL
VERDER  NOP
```

Keer terug en sla over instructies

. Geef de controle terug aan het adres boven in de stack nadat dit is verhoogd met een offset AANT. Deze routine stelt de gebruiker in staat de controle terug te geven aan een adres voorbij parameters, data of andere niet-uitvoerbare gegevens.

```
        PLA                 ;POP TERUGKEERADRES
        CLC
        ADC     #AANT+1     ;VERHOOG MET AANT
        STA     TERADR
        PLA
        ADC     #0          ;MET OVERDRACHT INDIEN NODIG
        STA     TERADR+1
        JMP     (TERADR)
```

of

```
        TSX                 ;ZET STACK-WIJZER IN INDEXREGISTER
        LDA     $101,X      ;VERHOOG TERUGKEERADRES MET AANT
```

```
        CLC
        ADC    #NUM
        STA    $101,X
        BCC    KLAAR
        INC    $102,X     ;MET OVERDRACHT, INDIEN NODIG
KLAAR   RTS
```

. Verander het terugkeeradres TERPT. Veronderstel dat het terugkeeradres op het ogenblik boven in de stack is opgeborgen. TERPT bestaat uit TERPTH (HWB) en TERPTL (LWB).

```
        TSX
        LDA    #TERPTL
        STA    $101,X
        LDA    #TERPTH
        STA    $102,X
        RTS
```

Het feitelijke terugkeeradres is TERPT+1.

Keer terug van interruptinstructies

Als het begin van een interrupt-afhandelingsroutine alle registers met de volgende reeks opbergt:

```
        PHA                ;BEWAAR ACCUMULATOR
        TXA                ;BEWAAR INDEXREGISTER X
        PHA
        TYA                ;BEWAAR INDEXREGISTER Y
        PHA
```

is een standaard terugkeer routine

```
        PLA                ;HERSTEL INDEXREGISTER Y
        TAY
        PLA                ;HERSTEL INDEXREGISTER X
        TAX
        PLA                ;HERSTEL ACCUMULATOR
```

DIVERSE INSTRUCTIES

Tot deze categorie behoren de push en pop (het opslaan in en halen uit de stack) instructies, stop, wacht, onderbreking, decimale aanpassing, in- en uitschakelen van interrupts, vertaling (tabelnaslag) en andere instructies die niet tot de eerder genoemde categorieen behoren.

1. Push-instructies.

. Push indexregister X.

```
        TXA    ;BEWAAR X VIA A IN STACK
        PHA
```

. Push indexregister Y.

```
        TYA                ;BEWAAR Y VIA A IN STACK
```

```
PHA
```

. Push geheugenlocatie ADR.

```
LDA    ADR        ;SLA GEHEUGENLOCATIE IN STACK OP
PHA
```

ADR zou ook een extern voorrangsregister of een kopie daarvan kunnen zijn.

. Push geheugenlocaties ADR en ADR+1 (HWB in ADR+1).

```
LDA    ADR+1      ;BEWAAR 16-BITS GETAL IN STACK
PHA
LDA  ADR
PHA
```

Aangezien de stack naar een lager adres groeit, wordt het 16-bits adres feitelijk op de gebruikelijke 6502 manier opgeslagen.

2. Pop-instructies
. Pop indexregister X.

```
PLA               ;HERSTEL X VIA A VANUIT STACK
TAX
```

. Pop indexregister Y.

```
PLA               ;HERSTEL Y VIA A VANUIT STACK
TAY
```

. Pop geheugenlocatie ADR.

```
PLA               ;HERSTEL GEHEUGENLOCATIE VANUIT STACK
STA    ADR
```

ADR zou ook een extern voorrangsregister of een kopie daarvan kunnen zijn.

. Pop de geheugenlocaties ADR en ADR+1 (HWB in ADR+1).

```
PLA               ;HERSTEL 16-BITS GETAL VANUIT STACK
STA    ADR
PLA
STA    ADR+1
```

We gaan ervan uit dat het 16-bits getal in de gebruikelijke 6502 vorm is opgeslagen, met de laagstwaardige byte op het lagere adres.

Wachtinstructies

De eenvoudigste manier om op de 6502 een wachtinstructie te implementeren, is door het gebruik van een oneindige lus. Zoals deze:

```
HIER    JMP    HIER
```

De processor blijft deze instructie uitvoeren tot er een interrupt plaatsvindt en keert hier weer terug nadat de interrupt-afhandelingsroutine de controle teruggeeft. Uiteraard moeten de maskeerbare interrupts wel ingeschakeld zijn, want anders wordt de lus eindeloos uitgevoerd. De niet-maskeerbare interrupt kan de processor op elk willekeurig moment onderbreken.

Een ander alternatief is de routine die wacht op een hoog-laag signaal op de Zet Overschot inputlijn. Een dergelijk signaal zet de Overschot (V) vlag. Dus is de gewenste reeks instructies:

```
        CLV                 ;WIS DE OVERSCHOTVLAG
WACHT   BVC    WACHT        ;EN WACHT OP EEN SIGNAAL DAT DE VLAG ZET
```

Deze routine is feitelijk een Wacht op Input van Signaal instructie.

Aanpassinginstructies

1. Vertak als de accumulator geen geldig decimaal (BCD) getal bevat.

```
STA    TEMP        ;BEWAAR DE ACCUMULATOR
SED                ;SCHAKEL DECIMALE MODUS IN
CLC                ;TEL 0 OP IN DECIMALE MODUS
ADC    #0
CLD                ;EN VERLAAT DECIMALE MODUS
```

2. Verhoog de accumulator decimaal (tel decimaal 1 bij A op).

```
SED                ;SCHAKEL DECIMALE MODUS IN
CLC
ADC    #1          ;TEL 1 DECIMAAL OP
CLD                ;VERLAAT DECIMALE MODUS
```

3. Verlaag de accumulator decimaal (trek decimaal 1 van A af).

```
SED                ;SCHAKEL DECIMALE MODUS IN
SEC
SBC    #1          ;TREK DECIMAAL 1 AF
CLD                ;VERLAAT DECIMALE MODUS
```

4. Schakel decimale modus in, maar bewaar de oude Decimale Modus vlag.

```
PHP                ;BEWAAR DE OUDE DECIMALE MODUS VLAG
SED                ;SCHAKEL DECIMALE MODUS IN
```

Een besluitende PLP-instructie zal de oude waarde van de Decimale Modus vlag (en de rest van het statusregister) herstellen.

5. Schakel de binaire modus in, maar bewaar de oude Decimale Modusvlag.

```
PHP                ;BEWAAR OUDE DECIMALE MODUS VLAG
CLD                ;SCHAKEL BINAIRE MODUS IN
```

Een besluitende PLP-instructie zal de oude waarde van de Decimale Modusvlag (en de rest van het statusregister) herstellen.

Interrupt In- en uitschakelinstructies

1. Schakel interrupts in, maar bewaar de oude waarde van de I-vlag.

```
PHP                 ;BEWAAR OUDE I-VLAG
CLI                 ;SCHAKEL INTERRUPTS IN
```

Na een routine die ingeschakelde interrupts vereiste, zal de PLP-instructie de oorspronkelijke staat van de I-vlag (en de andere vlaggen van het statusregister) herstellen.

```
PHP                 ;BEWAAR OUDE I-VLAG
SEI                 ;SCHAKEL INTERRUPTS UIT
```

Na een routine die uitgeschakelde interrupts vereiste, zal de PLP-instructie de oorspronkelijke staat van de I-vlag (en de andere vlaggen van het statusregister) herstellen.

Vertaalinstructies

1. Vertaal het gegeven in A met de corresponderende invoer in een tabel die begint bij het adres dat in de geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1) staat.

```
TAY
LDA    (PGNUL),Y ;VERVANG GEGEVEN DOOR INVOER UIT TABEL
```

Deze procedure kan gebruikt worden om data van de ene code naar de andere om te zetten.

2. Vertaal het gegeven in A naar de 16-bits waarde die op de corresponderende plaats in een tabel staat. Het beginadres van de tabel staat in de geheugenlocaties PGNUL en PGNUL+1 (HWB in PGNUL+1). Berg de nieuwe waarde op in de geheugenlocaties TEMPL en TEMPH (HWB in TEMPH).

```
ASL    A            ;VERDUBBEL INDEX VOOR 2-BYTES GROTE
                    ;WAARDEN
TAY
LDA    (PGNUL),Y    ;HAAL LWB VAN TABEL-INVOER
STA    TEMPL
INY
LDA    (PGNUL),Y    ;HAAL HWB VAN TABEL-INVOER
STA    TEMPH
```

AANVULLENDE ADRESSERINGSWIJZEN

. **Indirecte adressering.** Het is mogelijk op de 6502 processor indirecte adressering (voor adressen op pagina 0) te gebruiken. Dit kan door middel van postindexering (indirecte indexering), waarbij het indexregister Y op 0 is gezet. Een ietwat minder krachtig alternatief (omdat niet vanaf het indirecte adres geindexeerd kan worden) is preindexering (geindexeerde indirecte adressering) waarbij het indexregister X op 0 is gezet. Voor het overige is indirecte adressering alleen mogelijk

voor de JMP-instructie. Onthoud dat het indirecte adres voor de JMP overal in het geheugen mag staan; het is niet gelimiteerd tot pagina 0.

Voorbeelden

1. Laad de accumulator indirect vanuit het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
LDY     #0          ;ZET INDEX OP 0
LDA     (PGNUL),Y   ;LAAD INDIRECT GEINDEXEERD
```

2. Berg de accumulator indirect op in het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
LDY     #0          ;ZET INDEX OP 0
STA     (PGNUL),Y   ;BERG INDIRECT GEINDEXEERD OP
```

Voor de instructies die geen indirecte modus hebben (zoals ASL, DEC, INC, LSR, ROL, ROR), moeten de data eerst in de accumulator gezet worden, daarna bewerkt en vervolgens weer terug in het geheugen gezet worden.

3. Verhoog de data op het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staan.

```
LDY     #0          ;ZET INDEX OP 0
LDA     (PGNUL),Y   ;HAAL DE DATA
CLC
ADC     #1          ;VERHOOG DE DATA
STA     (PGNUL),Y
```

4. Voer een logische schuiving naar rechts uit op de data die op het adres in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
LDY     #0          ;ZET INDEX OP 0
LDA     (PGNUL),Y   ;HAAL DE DATA
LSR     A           ;SCHUIF ZE NAAR RECHTS
STA     (PGNUL),Y   ;EN BERG HET RESULTAAT WEER OP
```

5. Wis het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat.

```
LDY     #0          ;ZET INDEX OP 0
TYA                 ;DATA = 0
STA     (PGNUL),Y   ;WIS HET INDIRECTE ADRES
```

De enige manier om indirecte adressering op andere pagina's mogelijk te maken, is het indirecte adres eerst op pagina 0 te zetten.

6. Wis het adres dat in de geheugenlocaties INDIR en INDIR+1 (niet op pagina 0) staat.

```
LDA     INDIR       ;VERPLAATS INDIRECT ADRES NAAR PAGINA
                    ;NUL
STA     PGNUL
LDA     INDIR+1
STA     PGNUL+1
```

```
LDY     #0          ;ZET INDEX OP 0
TYA                 ;DATA = NUL
STA     (PGNUL),Y   ;WIS HET INDIRECTE ADRES
```

. Geindexeerde adressering. Geindexeerde adressering is beschikbaar voor de meeste instructies in de 6502 set. In het kort behandelen we de enkele waarvoor deze adressering niet toegestaan is, om vervolgens het verwerken van indexen, groter dan 256, te behandelen.
Indexering is niet mogelijk voor de instructies BIT, CPX, CPY, JMP en JSR. Indexering is alleen op pagina 0 mogelijk voor STX en STY. Deze beperkingen kunnen we als volgt opheffen:

1. BIT

Een geindexeerde BIT kan gesimuleerd worden door de accumulator op te slaan, een AND uit te voeren en de accumulator weer te herstellen. Let op dat het herstellen van de accumulator met LDA, PHA, TXA of TYA de nul- en de negatiefvlaggen beinvloedt. Een typerend voorbeeld voor het herstellen van de accumulator is:

```
PHA                 ;BEWAAR A
AND     BASIS,X     ;GEINDEXEERDE LOGISCHE AND
```

De nulvlag wordt gezet alsof er een geindexeerde BIT heeft plaatsgevonden en de inhoud van A staat boven in de stack.

2. CPX of CPY

Een geindexeerde CPX of CPY kan gesimuleerd worden door het indexregister naar A te verplaatsen en een CMP te gebruiken. Een CPX met Y geindexeerd kan op de volgende manier gesimuleerd worden:

```
TXA                 ;VERPLAATS X NAAR A
CMP     BASIS,X     ;VERGELIJK GEINDEXEERD
```

3. JMP

Een geindexeerde JMP kan gesimuleerd worden door het gewenste geindexeerde adres te berekenen, dit in het geheugen op te slaan en er met behulp van een indirecte JMP of een RTS naar toe te springen.
De routine hiervoor is:

```
LDA     INDEX
ASL     A           ;VERDUBBEL INDEX VOOR 2-BYTES GROTE
                    ;WAARDEN
TAX
LDA     BASIS,X     ;HAAL LWB VAN BESTEMMING
STA     INDIR
INX
LDA     BASIS,X     ;HAAL HWB VAN BESTEMMING
STA     INDIR+1
JMP     (INDIR)     ;SPRING INDIRECT NAAR BESTEMMING
```

of

```
LDA     INDEX
ASL     A               ;VERDUBBEL INDEX VOOR 2-BYTES GROTE
                        ;WAARDEN
TAX
LDA     BASIS+1,X       ;HAAL HWB VAN BESTEMMING
PHA
LDA     BASIS,X         ;HAAL LWB VAN BESTEMMING
PHA
RTS                     ;SPRING NAAR BESTEMMING MET EEN OFFSET
                        ;VAN 1
```

De tweede methode vereist een tabel die adressen bevat die allemaal 1 kleiner zijn dan de feitelijke adressen. Dit omdat een RTS de programmateller 1 verhoogt nadat deze vanuit de stack is geladen.

4. JSR

Een geindexeerde JSR kan worden gesimuleerd door een programma aan te roepen dat een als boven geindexeerde JMP uitvoert. Het uiteindelijke terugkeeradres blijft boven in de stack staan en een besluitende RTS geeft de controle terug aan het programma dat de routine oorspronkelijk aanriep. Het hoofdprogramma bevat dus:

```
JSR     OVRDRCHT
```

OVRDRCHT voert een geindexeerde sprongopdracht uit en draagt zo de controle over aan de feitelijke subroutine.

5. STX of STY.

Een geindexeerde STX of STY kan gesimuleerd worden door het indexregister naar A te verplaatsen en STA te gebruiken. Zo kunnen we een op Y geindexeerde STX simuleren door de volgende routine te gebruiken:

```
TXA             ;VERPLAATS X NAAR A
STA     BASIS,Y ;BERG GEINDEXEERD OP
```

BASIS kan op elke plek in het geheugen staan, maar niet op pagina 0.
Het is mogelijk te indexeren met indexen die groter dan 256 bytes zijn. Hiervoor moeten we een optelling verrichten bij de hoogstwaardige bytes en de indirect geindexeerde adresseringsmodus gebruiken. Als het basisadres zich bij voorbeeld in de geheugenlocaties PGNUL en PGNUL+1 bevindt en de index in de geheugenlocaties INDEX en INDEX+1 staat, zal de volgende routine het gecorrigeerde basisadres in de geheugenlocaties TEMP en TEMP+1 (op pagina 0) stoppen.

```
LDA     PGNUL       ;VERPLAATS LWB NORMAAL
STA     TEMP
LDA     PGNUL+1     ;TEL HWB'S OP
CLC
ADC     INDEX+1
STA     TEMP+1
```

TEMP en TEMP+1 bevatten nu het basis adres dat (samen met INDEX) gebruikt kan worden in de indirect geindexeerde modus.

Voorbeelden

1. Laad de accumulator geindexeerd.

```
LDY    INDEX      ;HAAL LWB VAN INDEX
LDA    (TEMP),Y   ;LAAD DE ACCUMULATOR INDIRECT
                  ;GEINDEXEERD
```

2. Sla de accumulator geindexeerd op, ervan uitgaande dat we A bovenop de stack bewaard hebben.

```
LDY    INDEX      ;HAAL LWB VAN INDEX
PLA               ;HERSTEL A
STA    (TEMP),Y   ;SLA A INDIRECT GEINDEXEERD OP
```

. **Automatische verhoging vooraf.** Automatische verhoging vooraf wil zeggen dat de inhoud van het indexregister automatisch wordt verhoogd voordat deze wordt gebruikt. Op de 6502 processor is dit te bewerkstelligen door middel van een INX of INY van het indexregister of door het gebruik van de 16-bits methoden om een basisadres in het geheugen te verhogen.

Voorbeelden

. Laad de accumulator vanuit het basisadres BASIS waarbij het indexregister X automatisch vooraf wordt verhoogd.

```
INX                ;VERHOOG X
LDA    BASIS,X
```

We gaan er hierbij vanuit dat het array minder dan 256 elementen bevat.

. Laad de accumulator vanuit het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat, waarbij de inhoud van de geheugenlocaties INDEX en INDEX+1 automatisch vooraf wordt verhoogd.

```
       INC    INDEX      ;VERHOOG INDEX
       BNE    KLAAR
       INC    INDEX+1    ;MET OVERDRACHT INDIEN NODIG
KLAAR  LDA    PGNUL      ;VERPLAATS LWB
       STA    TEMP
       LDA    PGNUL+1    ;TEL HWB'S OP
       CLC
       ADC    INDEX+1
       STA    TEMP+1
       LDY    INDEX      ;HAAL LWB VAN INDEX
       LDA    (TEMP),Y   ;LAAD ACCUMULATOR
```

Als het noodzakelijk is om vooraf automatisch met 2 te verhogen (zoals bij het verwerken van een array met adressen), gebruik dan de reeks

```
LDA    INDEX      ;VERHOOG MET 2
CLC
ADC    #2
STA    INDEX
```

```
        BCC     KLAAR
        INC     INDEX+1     ;OVERDRACHT NAAR HWB INDIEN NODIG
KLAAR   NOP
```

. **Automatische verhoging achteraf.** Automatische verhoging achteraf houdt in dat de inhoud van het indexregister, nadat deze gebruikt, automatisch wordt verhoogd. Op de 6502 is het mogelijk dit te bereiken met een INX of INY op het indexregister of door het gebruik van 16-bits methoden om een index in het geheugen te verhogen.

Voorbeelden

. Laad de accumulator vanuit het adres BASIS, waarbij het indexregister Y achteraf automatische wordt verhoogd.

```
        LDA     BASIS,Y     ;VERHOOG Y ACHTERAF
        INY
```

. Laad de accumulator vanuit het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat, waarbij de inhoud van de geheugenlocaties INDEX en INDEX+1 achteraf automatisch wordt verhoogd.

```
        LDA     PGNUL       ;VERPLAATS LWB VAN BASIS
        STA     TEMP
        LDA     PGNUL+1     ;TEL HWB'S VAN BASIS EN INDEX BIJ ELKAAR
                            ;OP
        CLC
        ADC     INDEX+1
        STA     TEMP+1
        LDY     INDEX       ;HAAL LWB VAN INDEX
        LDA     (TEMP),Y    ;LAAD ACCUMULATOR
        CPY     #0          ;VERLAAG INDEX AUTOMATISCH ACHTERAF
        BNE     VERLLWB
        DEC     INDEX+1     ;LEEN VAN HWB INDIEN NODIG
VERLLWB DEC     INDEX
```

. **Automatische verlaging vooraf.** Automatische verlaging vooraf betekend dat de inhoud van het indexregister, voordat deze wordt gebruikt, automatisch verlaagd wordt. Op de 6502 is het mogelijk dit te bereiken met een DEX of DEY op het indexregister of door het gebruik van 16-bits methoden om een index in het geheugen te verlagen.

Voorbeelden

. Laad de accumulator vanuit het adres dat in BASIS staat, waarbij de index X vooraf automatisch wordt verlaagd.

```
        DEX                 ; VERLAAG DE INDEX VOORAF
        LDA     BASIS,X
```

We gaan er hierbij vanuit dat een array minder dan 256 elementen bevat.

. Laad de accumulator vanuit het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat, waarbij de inhoud van de geheugenlocaties INDEX en INDEX+1 vooraf automatisch wordt verlaagd.

```
          LDA    INDEX      ;VERLAAG DE INDEX VOORAF
          BNE    VERLLWB
          DEC    INDEX+1    ;LEEN VAN HET HWB
VERLLLWB  DEC    INDEX
          LDA    PGNUL      ;HAAL HET LWB VAN DE BASIS OP
          STA    TEMP
          LDA    PGNUL+1    ;TEL INDEX BIJ HWB VAN BASIS OP
          CLC
          ADC    INDEX+1
          STA    TEMP+1
          LDY    INDEX      ;HAAL HET LSB VAN DE INDEX OP
          LDA    (TEMP),Y   ;LAAD DE ACCUMULATOR
```

Als het noodzakelijk is om vooraf automatisch met 2 te verlagen (zoals bij het verwerken van een array met adressen), gebruik dan de reeks:

```
          LDA    INDEX      ;VERLAAG DE INDEX AUTOMATISCH MET 2
          SEC
          SBC    #2
          STA    INDEX
          BCS    DONE
          DEC    INDEX+1    ;LEEN VAN HET HWB ALS DAT NODIG IS
DONE      NOP
```

. **Automatische verlaging achteraf.** Automatische verlaging achteraf betekend dat de inhoud van het indexregister, nadat deze is gebruikt, automatisch verlaagd wordt. Op de 6502 is het mogelijk dit te bereiken met een DEX of DEY op het indexregister of door het gebruik van 16-bits methoden om een index in het geheugen te verlagen.

Voorbeelden

. Laad de accumulator vanuit het adres dat in BASIS staat, waarbij de index X achteraf automatisch wordt verlaagd.

```
          LDA    BASIS,Y    ;VERLAAG Y ACHTERAF AUTOMATISCH
          DEY
```

. Laad de accumulator vanuit het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat, waarbij de inhoud van de geheugenlocaties INDEX en INDEX+1 achterafautomatish wordt verlaagd.

```
          LDA    PGNUL      ;HAAL DE LWB VAN BASIS OP
          STA    TEMP
          LDA    PGNUL+1    ;TEL DE HWB VAN BASIS
          CLC
          ADC    INDEX+1
          STA    TEMP+1
          LDY    INDEX      ;HAAL DE LWB VAN INDEX OP
          LDA    (TEMP),Y   ;LAAD DE ACCUMULATOR
```

```
        CPY    #0          ;VERLAAG AUTOMATISCH DE INDEX
        BNE    VERLLWB
        DEC    INDEX+1     ;LEEN VAN DE HWB ALS DAT NOGIG IS
VERLLWB DEC    INDEX
```

. **Geindexeerde indirecte adressering (preindexering).** De 6502 staat preindexering voor vele instructies toe. Voor die instructies waar deze mogelijkheid ontbreekt, kunnen we preindexering simuleren door de data met preindexering naar de accumulator te verplaatsen, deze te bewerken en (indien nodig) het resultaat met preindexering weer terug in het geheugen te zetten.

Voorbeelden

1. Voer een rotatie naar rechts uit op de data van het gepreindexeerde adres dat wordt verkregen door met X vanaf het basisadres PGNUL te indexeren.

```
LDA    (PGNUL,X)    ;HAAL DE DATA
ROR    A            ;ROTEER DATA NAAR RECHTS
STA    (PGNUL),X    ;EN BERG RESULTAAT WEER IN HET
                    ;GEHEUGEN OP
```

2. Wis het gepreindexeerde adres, dat wordt verkregen door met X vanaf het basisadres PGNUL te indexeren.

```
LDA    #0           ;DATA = NUL
STA    (PGNUL,X)    ;WIS GEPREINDEXEERD ADRES
```

Let op dat als de berekening van het effectieve adres bij preindexering een resultaat oplevert dat te groot is voor 8 bits, het overschot er wordt afgehaald zonder dat er een foutmelding komt. Ofte wel, de processor voorziet in een automatische omwikkeling op pagina 0.

. **Indirecte geindexeerde adressering (postindexering).** De 6502 staat postindexering voor vele instructies toe. Voor die instructies waar postindexering niet is toegestaan, kunnen we deze adresseringsmodus simuleren door de data met postindexering in de accumulator te zetten, deze te bewerken en (indien nodig) het resultaat met postindexering weer terug in het geheugen te zetten.

Voorbeelden

1. Verlaag de data op het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat, waarbij Y als index wordt gebruikt.

```
LDA    (PGNUL),Y    ;HAAL DE DATA
SEC
SBC    #1           ;VERLAAG DATA MET 1
STA    (PGNUL),Y    ;BERG HET RESULTAAT WEER IN HET
                    ;GEHEUGEN OP
```

2. Voor een rotatie naar links uit op de data op het adres dat in de geheugenlocaties PGNUL en PGNUL+1 staat, waarbij Y als een index wordt gebruikt.

```
LDA    (PGNUL),Y    ;HAAL DE DATA
ROL    A            ;ROTEER DE DATA NAAR LINKS
```

```
STA     (PGNUL),Y     ;BERG HET RESULTAAT WEER IN HET
                      ;GEHEUGEN OP
```

LITERATUUR

1. Osborne, A. *An Introduction to Microcomputers, Volume 1: Basic Concepts*, 2de editie Berkeley: Osborne/McGraw-Hill, 1980.

2. Leventhal, L.A. *6800 Assembly Language Programming*, Berkeley: Osborne/McGraw-Hill, 1978.

3. Leventhal, L.A. *6809 Assembly Language Programming*, Berkeley: Osborne/McGraw-Hill, 1981.

4. Fischer, W.P. *Microproccessor Assembly Language Draft Standard, IEEE Computer* , december 1979, pg 96-109.

5. Scanlon, L.J. *6502 Software Design*, Howard W. Sams, Indianapolis, Ind., 1980, pg 111-113

Hoofdstuk 3 Veel voorkomende programmeerfouten

Dit hoofdstuk beschrijft veel voorkomende programmeerfouten in 6502 machinetaalprogramma's. Het laatste deel beschrijft veel voorkomende fouten bij het schrijven van input/output drivers en interruptafhandelingsroutines. Onze doelen hierbij zijn de volgende:

. Programmeurs waarschuwen voor potentiele probleemgevallen en verwarring scheppende situaties.

. De waarschijnlijke oorzaken van programmeerfouten aangeven.

. Enkele technieken en waarschuwingen benadrukken zoals deze in de hoofdstukken 1 en 2 werden aangegeven.

. Onderhoudsprogrammeurs informeren over waar ze moeten kijken voor fouten en verkeerde gegevensverwerking.

. De beginner op weg helpen bij het moeizame proces van het opsporen en corrigeren van fouten.

Het spreekt voor zich dat geen enkele foutenlijst compleet kan zijn. In dit hoofdstuk beschrijven we de meest voorkomende fouten; we hebben geen pogingen ondernomen om de zeldzame, kleine of toevallige fouten die zelfs de meeste ervaren programmeur frustreren te beschijven. Hoe dan ook, de meeste fouten zijn opvallend eenvoudig als ze eenmaal ontmaskerd zijn. Deze lijst kan de programmeur helpen de meeste programma's foutvrij te maken.

INDELING VAN PROGRAMMEERFOUTEN

Over het algemeen kunnen we de meest voorkomende 6502 programmeerfouten in de volgende categorieen onderbrengen.

. Onjuist gebruik van de overdracht. Typerende fouten zijn: vergeten de overdrachtvlag te wissen voor optellen of te zetten voor aftrekken, en de stand van de vlag verkeerd interpreteren na vergelijkingen (ze gedraagt zich als een geinverteerde lening).

. Onjuist gebruik van de andere vlaggen. Tot de typerende voorbeelden behoren: de verkeerde vlag gebruiken (bij voorbeeld de negatief- in plaats van de overdrachtvlag), vertakken na instructies die een bepaalde vlag niet beinvloeden, het omkeren van de vertak-voorwaarden (in het bijzonder waar het om de nulvlag gaat), en een vlag per ongeluk voor de vertakking veranderen.

. Adressen en data verwarren. Tot de typerende voorbeelden behoren: onmiddellijke in plaats van directe adressering of andersom, en adressen op pagina 0 verwarren met de adressen die indirect door de locaties benaderd kunnen worden.

. Het verkeerde formaat gebruiken. Tot de typerende voorbeelden behoren: BCD (decimaal) formaat gebruiken in plaats van binair formaat (of andersom), en binair of hexadecimaal gebruiken in plaats van ASCII.

. Arrays verkeerd verwerken. Tot de typerende voorbeelden behoren: per ongeluk aan de ene of de andere zijde voorbij het einde gaan, en paginagrenzen veronachtzamen als het array meer dan 256 bytes groot is.

. Impliciete effecten negeren. Tot de typerende voorbeelden behoren: het gebruik van accumulator, indexregister, stack-wijzer, vlaggen of pagina 0 locaties zonder het effect van tussenliggende instructies op deze inhouden te bedenken. De meeste fouten komen van instructies die onverwachte, impliciete of indirecte effecten hebben.

. Niet de juiste beginvoorwaarden scheppen voor routines of zelfs de computer als geheel. De meeste routines vereisen de initialisatie van tellers, indirecte adressen, indexen, registers, vlaggen en tijdelijke opslaglocaties. De computer in zijn geheel vereist de initialisatie van de Interrupt Disable en de Decimale Modus vlaggen en alle algemene RAM adressen (let hierbij vooral op indirecte adressen en andere tijdelijke opslag in pagina 0).

. Het programma verkeerd organiseren. Tot de typerende voorbeelden behoren: initialisatie-routines overslaan of herhalen, het niet aanpassen van indexen, tellers of indirecte adressen, en vergeten tussentijdse of uiteindelijke uitkomsten op te slaan.

Een veel voorkomende bron van fouten, die buiten het bereik van dit hoofdstuk ligt, wordt gevormd door conflicten tussen gebruikerprogrammatuur en systeemprogrammatuur. Een eenvoudig voorbeeld is een programma van een gebruiker dat resultaten in tijdelijke opslaglocaties zet, die de systeemprogrammatuur of hulpprogrammatuur voor eigen doeleinden nodig heeft. De resultaten verdwijnen op mysterieuze wijzen, ofschoon ook het gedetailleerd volgen van het programma geen enkele zichtbare fout oplevert.
Complexere bronnen van conflicten kunnen door het interrupt-systeem, input/output-poorten, de stack of de vlaggen gevormd worden. De systeemprogrammatuur moet namelijk van dezelfde middelen gebruik maken als de programmatuur van de gebruiker (over het algemeen probeert de systeemprogrammatuur de omgeving van de gebruiker te herstellen, maar er treden vaak subtiele of onverwachte effecten op). Een operating-systeem (systeem-programma) transparant voor de gebruiker te maken is ongeveer hetzelfde als het opstellen van een stelsel van richtlijnen, wetten of belastingregels dat geen mazen of neveneffecten heeft.

ONJUIST GEBRUIK VAN DE OVERDRACHT

De volgende instructies en afspraken zijn de meest voorkomende oorzaken van fouten:

. CMP, CPX en CPY beinvloeden de overdracht alsof deze een geinverteerde lening (borrow) is. Ofte wel, zij zetten de overdracht als het aftrekken van de geheugenlocatie van het register geen lening nodig had, en wissen de overdracht als dit wel het geval was. Zodoende overdracht = 1 als er geen lening nodig was en overdracht = 0 als er wel een lening nodig was. Dit is tegenovergesteld aan de betekenis van de overdracht in de meeste andere microprocessoren (de 6800, 6809, 8080, 8085 of Z-80).

. SBC trekt de geinverteerde overdrachtvlag af van het resultaat van de normale aftrekking van een geheugenlocatie van de accumulator. Ofte wel, de instructie produceert het resultaat (A) - (G) - (1 - overdracht). Als de overdracht het resultaat niet mag beinvloeden, moet deze met SEC gezet worden. Net als vergelijkingen, beinvloedt SBC de overdracht als ware deze een geinverteerde lening; overdracht = 0 als de aftrekking een lening nodig had en 1 als dit niet het geval was.

. ADC betrekt altijd de overdracht bij de optelling. Dit geeft het resultaat (A) = (A) + (G) + overdracht. Als de overdracht het resultaat niet mag beinvloeden, moet deze eerst met CLC gewist worden. De overdracht heeft haar normale betekenis na ADC.

Voorbeelden

1. CMP ADR

Deze instructie zet de vlaggen alsof de inhoud van geheugenlocatie ADR gewoon van de accumulator was afgetrokken. De overdrachtvlag wordt gezet als de aftrekking geen lening nodig heeft en gewist als dit wel het geval is. Dus:

overdracht = 1 als (A) >= (ADR)
overdracht = 0 als (A) < (ADR)

We gaan ervan uit dat beide getallen ongetekend zijn. Onthoud dat de overdracht (op 1) wordt gezet als de nummers gelijk zijn.

2. SBC #WAARDE

Deze instructie trekt WAARDE en 1-overdracht van de accumulator af. Ze zet de vlaggen net als een vergelijking. Om WAARDE alleen van de accumulator af te trekken moet de volgende reeks gebruikt worden:

```
SEC                 ;ZET GEINVERTEERDE LENING
SBC     #WAARDE     ;TREK WAARDE AF
```

Deze reeks produceert het resultaat (A) = (A) - WAARDE. Als WAARDE = 1, is de reeks gelijk aan een Verlaag Accumulator instructie (onthoud dat DEC niet voor A gebruikt kan worden).

3. ADC #WAARDE

Deze instructie telt WAARDE en overdracht bij de accumulator op. Om alleen WAARDE bij de accumulator op te tellen, moet de volgende reeks worden gebruikt:

```
CLC                 ;WIS OVERDRACHT
ADC     #WAARDE     ;TEL WAARDE OP
```

Deze reeks geeft het resultaat (A) = (A) + WAARDE. Als WAARDE = 1, dan is deze reeks gelijk aan de Verhoog Accumulator instructie (onthoud dat INC niet op A toepasbaar is).

ONJUIST GEBRUIK VAN DE ANDERE VLAGGEN

6502 instructies hebben bijna altijd de verwachte gevolgen op de vlaggen. Het enige bijzondere geval is de BIT-instructie. Omstandigheden die iets meer aandacht nodig hebben, zijn de volgende:

. Opslaginstructies (STA, STX en STY) hebben geen invloed op de vlaggen, dus hoeft het niet zo te zijn dat zij de de waarde weergeven van datgene wat net is opgeslagen. Het kan zijn dat het register getest moet worden door het van het ene register naar het andere te verplaatsen of het met 0 te vergelijken. Let op dat de laadinstructies (PHA incluis) en overdrachtinstructies (TXS uitgezonderd) de nul- en negatiefvlaggen beinvloeden.

. Na een vergelijking (CMP, CPX of CPY), geeft de nulvlag aan of de gegevens gelijk waren. De nulvlag is gezet als de gegevens gelijk waren en gewist als dit niet het geval was. Dit kan aanleiding zijn voor enige verwarring - BEQ betekent vertak als het resultaat gelijk is aan 0, ofte wel vertak als de nulvlag 1 is. Let op dit verschil tussen resultaat en de stand van de nulvlag. Deze twee zijn tegenovergestelden. De nulvlag is 0 als het resultaat niet 0 is.

. Bij het vergelijken van ongetekende getallen, geeft de overdrachtvlag aan welk getal groter is. CMP, CPX of CPY wissen de overdracht als de inhoud van het register groter of gelijk is aan het andere gegeven en zetten de overdracht als de inhoud van het register kleiner is. Let op dat het vergelijken van twee gelijke grootheden de overdracht zet. Als deze alternatieven (groter dan of gelijk en kleiner dan) niet datgene zijn wat nodig is (bij voorbeeld groter dan en kleiner dan of gelijk), kan de aftrekking worden omgedraaid; trek 1 van de accumulator af of tel 1 bij het andere gegeven op.

. Bij het vergelijken van getekende getallen, geeft de negatiefvlag aan welk gegeven groter is, tenzij er een tweecomplement-overloop plaatsvond. We moeten dus eerst naar de overloopvlag kijken. Is deze 0, dan geeft de negatiefvlag aan welk gegeven groter is. Is de overloopvlag 1, dan is de betekenis van de negatiefvlag geinverteerd.
Na een vergelijking (als er geen overschot optreedt), wordt de negatiefvlag gezet als de inhoud van het register kleiner is dan het andere gegeven en gewist als de inhoud van het register groter is dan of gelijk is aan het andere gegeven. Let op dat het vergelijken van twee gelijke grootheden de negatiefvlag wist. Net als bij de overdracht kan het geval van gelijkheid omgekeerd worden afgehandeld door een van de twee gegevens aan te passen of de aftrekking om te draaien.

. Als aan een voorwaarde is voldaan en de computer iets moet doen, dan is een veelgebruikte methode om rond een deel van het programma te vertakken op de tegenovergestelde voorwaarde. Om bij voorbeeld de geheugenlocatie OVRSCHT te verhogen als de overdracht 1 is, is de reeks

```
        BCC     VOLG
        INC     OVRSCHT
VOLG    NOP
```

te gebruiken. De voorwaarde om te vertakken is de tegenovergestelde voorwaarde waaronder het deel uitgevoerd zou moeten worden.

. Verhoog- en verlaaginstructies beinvloeden de overdrachtvlag niet. Hierdoor kunnen deze instructies worden gebruikt voor het tellen in lussen die rekenkundige handelingen verrichten met meer dan 1 byte (het overdracht-bit is dan nodig om overdrachten en leningen tussen bytes te verrichten). Verhoog- en verlaaginstructies hebben echter wel invloed op de Nul- en negatiefvlaggen; het effect op de negatiefvlag is te gebruiken om te zien of een verhoging een overdracht heeft veroorzaakt. Let op de volgende typerende routines:

1. Een 16-bits verhoging van de geheugenlocaties INDEX en INDEX+1 (HWB in INDEX+1)

```
        LDA     INDEX       ;VERHOOG LWB
        BNE     KLAAR
        INC     INDEX+1     ;EN OVERDRACHT NAAR HWB INDIEN NODIG
KLAAR   NOP
```

We kijken of er een overdracht is gegenereerd door de nulvlag te onderzoeken nadat de laagstwaardige byte is verhoogd.

2. Een 16-bits verlaging van de geheugenlocaties INDEX en INDEX+1 (HWB in INDEX+1)

```
          LDA    INDEX      ;CONTROLEER LWB
          BNE    VERLLWB
          DEC    INDEX+1    ;LEEN VAN HWB INDIEN NODIG
VERLLWB   DEC    INDEX       ;VERLAAG HWB
```

Door de laagstwaardige byte te bekijken voordat deze wordt verlaagd, kunnen we bepalen of er al dan niet een lening nodig zal zijn.

. De BIT-instructie heeft nogal ongebruikelijke effecten op de vlaggen. De instructie plaatst bit 6 van de geheugenlocatie in de overloopvlag en bit 7 in de negatiefvlag, ongeacht de inhoud van de accumulator. Zodoende geeft alleen de nulvlag het resultaat weer van een logische AND tussen accumulator en geheugenlocatie.

. Slechts een paar instructies beinvloedt de overdracht- of overloopvlaggen. De instructies die de overdracht beinvloeden, naast de voor de hand liggende CLC en SEC, zijn de rekenkundige (ADC, SBC), vergelijkende (CMP, CPX en CPY) en schuivende (ASL, LSR, ROL en ROR). De enige instructies die de overloopvlag beinvloeden zijn ADC, BIT, CLV en SBC. Vergelijkende en schuivende instructies hebben geen invloed op de overloopvlag. Dit in tegenstelling tot de situatie bij de nauw verwante 6800 en 6809 processoren.

Voorbeelden

1. De reeks

```
          STA    $1700
          BEQ    KLAAR
```

heeft onvoorspelbare resultaten, omdat STA geen enkele vlag beinvloedt. Om een sprong te genereren nadat een 0 is opgeslagen, is de volgende routine nodig:

```
          STA    $1700
          CMP    #0         ;TEST ACCUMULATOR
          BEQ    KLAAR
```

of

```
          STA    $1700
          TAX               ;TEST ACCUMULATOR
          BEQ    KLAAR
```

2. De instructie CMP #$25 zet de nulvlag als volgt:

Nul = 1 als de inhoud van A = 25_{16}
Nul = 0 als de inhoud van A <> 25_{16}

Als de geheugenlocatie TELLER verhoogd moet worden als (A) = 25_{16}, dan is de volgende reeks nodig:

```
          CMP    #$25       ;IS A 25?
          BNE    KLAAR
          INC    TELLER     ;JA, VERHOOG TELLER
```

```
KLAAR       NOP
```

Merk op dat we de instructie BNE gebruiken om langs de verhoging van de teller te gaan als niet aan de voorwaarde ((A) = 25_{16}) is voldaan. Vergissingen kunnen makkelijk ontstaan als de voorwaarde waarop het programma om de teller heengaat, geinverteerd wordt.

3. De instructie CPX #$25 zet de overdrachtvlag als volgt:

overdracht = 0 als de inhoud van X in de reeks 00_{16} tot en met 24_{16} ligt.
overdracht = 1 als de inhoud van X in de reeks 25_{16} tot en met FF_{16} ligt.

Zodoende wordt de overdrachtvlag gewist als een X een ongetekend getal bevat dat kleiner is dan het andere gegeven en gezet als X een ongetekend getal bevat dat groter is dan of gelijk is aan het andere gegeven.

Als de overdracht gewist moet worden als het X-register 25_{16} bevat, moet de opdracht CPX #$26 in plaats van CPX #$25 gebruikt worden:

```
CPX     #$25
BCC     MIND        ;VERTAK ALS (X) KLEINER DAN 25 IS
```

of

```
CPX     #$26
BCC     MINDGELK    ;VERTAK ALS (X) KLEINER DAN OF GELIJK
                    ;AAN 25 IS
```

4. De reeks SEC, SBC #$40 zet de negatiefvlag (Teken) als volgt:

Negatief = 0 als A tussen de 40_{16} en $7F_{16}$ (normaal getekend rekenkundig) of tussen de 80_{16} en $C0_{16}$ ligt (als gevolg van het tweecomplement-overschot).

Negatief = 1 als A tussen de 00_{16} en $3F_{16}$ of tussen de $C1_{16}$ en FF_{16} (normaal getekend rekenkundig) ligt.

Een tweecomplement-overschot ontstaat als A een getal tussen 80_{16} (-128_{10} in tweecomplement) en $C0_{16}$ (-64_{10} in tweecomplement) bevat. Als dan 40_{16} (64_{10}) wordt afgetrokken, ontstaat er een resultaat dat kleiner is dan -128_{16}, hetgeen buiten het bereik van een 8-bits getekend getal komt. Het zetten van de overloopvlag geeft deze 'buiten bereik' situatie aan.
De volgende reeks zal dus een vertakking veroorzaken als A een getekend getal kleiner dan 40_{16} bevat.

```
SEC                 ;ZET GEINVERTEERDE LENING
SBC     #$40        ;TREK 40 HEX AF
BVS     BEST        ;VERTAK ALS OVERSCHOT GEZET IS
BMI     BEST        ;OF ALS VERSCHIL NEGATIEF IS
```

Let op dat we hier geen CMP kunnen gebruiken, omdat deze instructie de overloopvlag niet beinvloedt. We kunnen echter wel de volgende reeks gebruiken:

```
CMP     #0          ;VERTAK ALS A NEGATIEF IS
BMI     BEST
CMP     #$40        ;OF ALS A POSITIEF MAAR KLEINER DAN 40
```

```
                    ;HEX IS
BCC     BEST
```

We schakelen de mogelijkheid van een overschot uit door de negatieve getallen apart te verwerken.

5. De reeks

```
INC     ADR
BCS     VLGPG
```

zal onvoorspelbare resultaten hebben, aangezien INC de overdrachtvlag niet beinvloedt. Een reeks die een sprong produceert als het resultaat van de verhoging 0 is (waarmee het ontstaan van een overdracht wordt aangegeven), luidt:

```
INC     ADR
BEQ     VLGPG
```

We kunnen zeggen een verhoging een overdracht heeft geproduceerd, maar niet of een verlaging een lening nodig had, omdat het resultaat FF_{16} en niet 0 is. Daarom is het veel eenvoudiger een getal van meerdere bytes te verhogen dan te verlagen.

6. De reeks

```
BIT     ADR
BVS     BEST
```

geeft een vertakking als bit 6 van ADR 1 is. De inhoud van de accumulator heeft er geen invloed op. Evenzo geeft de reeks

```
BIT     ADR
BPL     BEST
```

een vertakking als bit 7 van ADR 0 is. De inhoud van de accumulator heeft er geen invloed op. De enige vaker gebruikte reeks met een BIT-instructie waarbij de accumulator een rol speelt, is

```
LDA     #MASKER
BIT     ADR
```

Deze reeks zet de nulvlag indien een logische AND tussen MASKER en de inhoud van geheugenlocatie ADR het resultaat 0 geeft. Een typerend voorbeeld voor het gebruik van de nulvlag is:

```
LDA     #%00010000
BIT     ADR
BNE     BEST            ;VERTAK ALS BIT 4 VAN ADR 1 IS
```

Deze reeks geeft een vertakking als het resultaat van de logische AND ongelijk aan nul is, ofte wel als bit 4 van ADR 1 is.
De effecten van BIT op de overloop- en negatiefvlaggen leiden over het algemeen niet tot fouten, omdat er geen standaard, veel gebruikte effecten zijn die verwarring kunnen wekken. Wel leveren deze effecten problemen op bij de documentatie van de programmatuur, omdat de benadering uniek is en diegenen die niet bekend zijn met de 6502 niet kunnen worden geacht te weten wat er gebeurt.

7. De reeks

```
CMP     #WAARDE
BVS     BEST
```

geeft onberekenbare resultaten, omdat CMP de overloopvlag niet beinvloedt. Om een vertakking te genereren als de aftrekking tot een tweecomplement leidt, moet de reeks

```
SEC                 ;ZET GEINVERTEERDE LENING
SBC     #WAARDE     ;TREK WAARDE AF
BVS     BEST
```

worden gebruikt.

HET VERWARREN VAN ADRESSEN EN DATA

De te onthouden regels zijn

. De onmiddellijke adresseringsmodus heeft de feitelijke data nodig als gegeven. Ofte wel, LDA #$40 laadt de accumulator met het getal 40_{16}.

. De absolute en pagina nul (directe) adresseringsmodi hebben het adres van de data als gegeven nodig. Ofte wel, LDA $40 laadt de accumulator met de inhoud van geheugenlocatie 40_{16}.

. De indirect geindexeerde en geindexeerde indirecte adresseringsmodi halen hun indirecte adres uit twee geheugenlocaties op pagina 0. Het indirecte adres staat in twee geheugenlocaties, beginnende bij het gespecificeerde adres en is ondersteboven opgeslagen met de laagstwaardigste byte op het laagste adres. Gelukkig wordt de geindexeerde indirecte (gepreindexeerde) adresseringsmodus zelden gebruikt en is daarom haast nooit een bron van problemen. De betekenis van adresseringsmodi met JMP en JSR kan verwarrend zijn, omdat deze instructies de adressen gebruiken als zijnde data. De veronderstelling is dat de controle nooit aan een getal overgegeven kan worden en directe adressering daarom zonder betekenis is. Echter, de instructie JMP $1C80 laadt, net als een laadinstructie bij onmiddellijk adressering, $1C80_{16}$ in de programmateller, ondanks dat we gewend zijn te zeggen dat de instructie absolute adressering gebruikt. Evenzo laadt de instructie JMP (ADR) de programmateller met het adres dat in de geheugenlocaties ADR en ADR+1 staat; zodoende gedraagt deze instructie zich als een laadinstructie met absolute (directe) adressering.

Voorbeelden

1. LDX #$20 laadt het getal 20_{16} in indexregister X. LDX $20 laadt de inhoud van geheugenlocatie 0020_{16} in indexregister X.

2. LDA ($40),Y laadt de accumulator vanuit het adres dat wordt verkregen door met Y te indexeren vanaf het basisadres in geheugenlocaties 0040_{16} en 0041_{16} (HWB in 0041_{16}). Let op dat als LDA ($40),Y zinnig is, LDA ($41),Y over het algemeen onzin is, omdat het het basis-adres in de geheugenlocaties 0041_{16} en 0042_{16} gebruikt. Zodoende zijn de indirecte adresseringsmodi alleen zinnig als de indirecte adressen in een juiste verhouding ten opzichte van elkaar staan op woordgrenzen. De 6502 controleert deze verhouding echter niet op de manier waarop veel (vooral IBM) computers dat doen. De programmeur dient er zorg voor te dragen dat alle indirect gebruikte geheugenlocaties adressen bevatten waarvan de bytes goed gerangschikt zijn.

Het verwarren van adressen en hun inhoud vormt een regelmatig terugkerend probleem bij het verwerken van datastructuren. De rij door testinstrumenten uit te voeren handelingen, bij voorbeeld, kan uit een blok informatie voor elke handeling bestaan. Dat blok kan het volgende bevatten

. Het beginadres van de testroutine.

. Het aantal seconden dat de test uitgevoerd moet worden.

. Het adres waarin het resultaat opgeslagen moet worden.

. De onder- en bovengrenswaarden waarmee het resultaat vergeleken moet worden.

. Het adres van het volgende blok in de rij.

Zodoende bevat het blok data en directe en indirecte adressen. Typerende fouten die de programmeur kan maken, zijn:

. De controle overdragen naar de geheugenlocaties die het beginadres van de testroutine bevatten, in plaats van het feitelijke beginadres.

. Het resultaat opslaan in het blok, in plaats van in het adres dat in het blok gespecificeerd is.

. Het gebruik van een drempelwaarde als adres, in plaats van als data.

. De veronderstelling dat het volgende blok binnen het huidige blok begint, in plaats van op het adres dat in het blok wordt gegeven.

Sprongtabellen zijn een andere, veel voorkomende bron van fouten. Houd bij het programmeren de volgende regels in acht:

. Maak een tabel van sprongopdrachten en draag de controle over aan de inhoud van het juiste element (bij voorbeeld aan het derde element).

. Maak een tabel van bestemmingsadressen een draag de controle over aan het juiste element (bij voorbeeld aan het adres in het derde element).

Problemen zullen zich zeker voordoen bij het gebruikt van sprongopdrachten als indirecte adressen of bij het uitvoeren van het de indirecte adressen.

FORMATEERFOUTEN

De te onthouden regels zijn:

. Een $ voor een getal (of een H erachter) geeft de assembler aan dat het getal hexadecimaal is. Een % voor, of een B achter een getal geeft aan dat dit binair is. Let erop dat enkele assemblers andere symbolen gebruiken.

. Als een getal niet nader gespecificeerd is, gaan de meeste assemblers ervan uit dat het getal decimaal is. Slechts enkele assemblers (zoals Apple's mini-assembler en de mnemonic-invoer modus van Rockwell's AIM-65) veronderstellen hexadecimale waarden als standaard.

. ADC- en SBC-instructies geven decimale resultaten als de Decimale Modus vlag op 1 staat en binaire resultaten als de Decimale Modus vlag op 0 staat. Alle andere instructies, inclusief DEC, DEX, DEY, INC, INX en INY geven altijd een binair resultaat.

De programmeur moet de moeite nemen om de volgende fouten te voorkomen:

. Het weglaten van het hexadecimale-notatieteken ($ of H) bij hexadecimale data of adressen. De assembler gaat ervan uit dat het gegeven decimaal is als het geen letters bevat. Hij zal het gegeven als een naam behandelen als deze geldig is (in de meeste assemblers moet deze met een letter beginnen). De assembler zal alleen een foutmelding geven als het gegeven niet als een decimaal getal of als een naam geinterpreteerd kan worden.

. Het weglaten van het binaire-notatieteken (% of B) bij een binair gegeven. De assembler zal het als een decimaal getal behandelen.

. Het verwarren van decimale (BCD) weergaven met binaire weergaven. Onthoud dat 10 niet een gehele macht van twee is, zodat binaire en BCD-weergaven boven 9 verschillen. Standaard BCD-constanten moeten genoteerd worden als hexadecimale in plaats van decimale getallen.

. Het verwarren van binaire of decimale weergaven met ASCII-weergaven. Een ASCII-input-apparaat produceert ASCII-karakters en een ASCII-output-apparaat reageert op ASCII-karakters.

Voorbeelden

1. LDA 2000

Deze instructie laadt de accumulator vanuit het geheugenadres 2000_{10} ($7D0_{16}$), niet het adres 2000_{16}. De assembler zal geen foutmelding geven, omdat 2000 een geldig decimaal getal is.

2. AND #00000011

Deze instructie voert een logische AND uit tussen de accumulator en het decimale getal 11 (1011_2), niet met het binaire getal 11 (3_{10}). De assembler zal geen foutmelding geven omdat 00000011, ondanks de ongebruikelijk vorm, een geldig decimaal getal is.

3. ADC #40

Deze instructie telt 40_{10} (niet $40_{16} = 64_{10}$) en de overdracht op bij de accumulator. Let op dat 40_{10} niet hetzelfde is als 40 in BCD-notatie, hetgeen 40_{16} is; $40_{10} = 28_{16}$. De assembler zal geen foutmelding geven omdat 40 een geldig decimaal getal is.

4. LDA #3

Deze instructie laadt de accumulator met het getal 3. Als deze waarde nu naar een ASCII-output-apparaat wordt gestuurd, zal het reageren alsof het karakter ETX (03_{16}) en niet het karakter 3 (33_{16}) is ontvangen. De juiste versie is

```
LDA     #'3         ;HAAL EEN ASCII 3
```

5. Als de geheugenlocatie 0040_{16} een enkel cijfer bevat, zal de reeks

```
LDA     $40
STA     POORT
```

dat cijfer niet op het ASCII-output-apparaat zetten. De juiste reeks is

```
LDA     $40         ;HAAL DECIMAAL CIJFER
CLC
ADC     #'0         ;PAS AAN VOOR ASCII
STA     POORT
```

of

```
LDA     $40           ;HAAL DECIMAAL CIJFER
ORA     #%00110000    ;PAS AAN VOOR ASCII
STA     POORT
```

6. Als input-poort IPOORT een enkel ASCII decimaal cijfer bevat, zal de reeks

```
LDA     IPORT
STA     $40
```

het feitelijke cijfer niet in de geheugenlocatie 0040_{16} stoppen. In plaats daarvan wordt de ASCII-versie van dat cijfer (het feitelijke cijfer + 30_{16}) in de geheugenlocatie opgeborgen. De juiste reeks is

```
LDA     IPORT       ;HAAL ASCII-CIJFER
SEC
SBC     #'0         ;PAS AAN VOOR DECIMAAL
STA     $40
```

of

```
LDA     IPORT        ;HAAL ASCII-CIJFER
AND     #%11001111   ;PAS AAN VOOR DECIMAAL
STA     $40
```

Het afhandelen van decimale rekenkunde op de 6502 microprocessor is makkelijk omdat deze processor een Decimale Modus (D) vlag kent. Als de vlag gezet is (door SED), produceren alle optellingen en aftrekkingen decimale resultaten. Dus implementeren de volgende reeksen decimale optelling en aftrekking:

. Tel de inhoud van geheugenlocatie ADR decimaal bij de accumulator op

```
SED                 ;SCHAKEL DECIMALE MODUS IN
CLC
ADC     ADR         ;TEL DECIMAAL OP
CLD                 ;SCHAKEL DECIMALE MODUS UIT
```

. Trek de inhoud van geheugenlocatie ADR decimaal af van de accumulator

```
SED                 ;SCHAKEL DECIMALE MODUS IN
```

```
SEC
SBC    ADR        ;TREK DECIMAAL AF
CLD               ;SCHAKEL DECIMALE MODUS UIT
```

Aangezien verhoog- en verlaaginstructies altijd binaire resultaten geven, moeten we de volgende reeksen gebruiken (ervan uitgaande dat de D-vlag is gezet):

Verhoog geheugenlocatie 0040_{16} in de decimale modus

```
LDA    $40
CLC
ADC    #1
STA    $40
```

Verlaag geheugenlocatie 0040_{16} in de decimale modus

```
LDA    $40
SEC
SBC    #1
STA    $40
```

Het nadeel van de decimale modus is, dat deze neveneffecten heeft. Dezelfde ADC- en SBC-instructies op dezelfde data zullen verschillende resultaten hebben, afhankelijk van de status van de Decimale Modusvlag. De volgende procedures zullen de kans op deze effecten die onvoorzienbare fouten geven:

. Initialiseer de Decimale Modus vlag (met CLD) als een deel van de normale systeem-initialisatie. Onthoud dat RESET geen invloed op de Decimale Modusvlag heeft.

. Wis de Dvlag zodra de decimale rekenkundige bewerkingen zijn uitgevoerd.

. Initialiseer de Dvlag in interrupt-afhandelingsroutines die ADC- of SBC-instructies gebruiken. Ofte wel, dergelijke routines moeten een CLD-instructie uitvoeren voor er een binaire aftrekking of optelling plaatsvindt.

ARRAYS ONJUIST VERWERKEN

De volgende situaties zijn de meest voorkomende bron van fouten:

. Bij het tot 0 aftellen van een indexregister, mag de index waarde nul nooit gebruikt worden. De oplossing hiervoor is het basisadres of de basisadressen met 1 te verlagen. Als bij voorbeeld de afsluitende reeks in een lus

```
DEX
BNE    LUS
```

is, gaat de processor verder zodra X tot 0 is verlaagd. Een aangepaste lus (die NMAAL geheugenbytes wist) is:

```
LDX    #NMAAL
LDA    #0
```

```
WIS         STA    BASIS-1,X
            DEX
            BNE    WIS
```

Merk het gebruik van BASIS-1 op bij de geindexeerde opslaginstructie. Het programma wist de adressen BASIS tot en met BASIS + NMAAL-1

. Hoewel van achteren naar voren door een array gaan vaak efficienter is dan andersom, vinden programmeurs dit over het algemeen verwarrend. Onthoud dat het adres BASIS+(X) de voorafgaande data bevat in een lus zoals die van het voorbeeld hierboven. Ofschoon processoren net zo makkelijk terug als vooruit werken, hebben programmeurs vaak moeite om achteruit te denken.

. Pas op dat er niet een lus te veel of een lus te weinig wordt uitgevoerd. Onthoud dat de geheugenlocaties BASIS tot en met BASIS+N, N+1 en niet N gegevens bevatten. De laatste invoer van de tabel is snel vergeten en de eerste valt makkelijk weg. Anderzijds, als er N gegevens zijn, zullen zij de geheugenlocaties BASIS tot en met BASIS+N-1 bestrijken; het is nu gemakelijk om vanaf het einde van het array te werken.

. Absolute geindexeerde adressering of op pagina nul geindexeerde adressering kan niet verder reiken dan 256 bytes. Als een indexregister FF_{16} bevat, geeft verhoging 00_{16} als resultaat. Evenzo, als een indexregister 00 bevat, zal verlaging FF_{16} als resultaat hebben. Daarom is voorzichtigheid geboden bij het verhogen of verlagen van indexregisters, als de kans aanwezig is dat de 8-bits capaciteit wordt overschreden. Gebruik de indirect geindexeerde adresseringsmodus om voorbij de 256 bytes te laten komen. De volgende reeks telt 1 op bij de hoogstwaardige byte van het indirecte adres als het indexregister Y tot 0 is verhoogd.

```
            INY                ;VERHOOG INDEXREGISTER
            BNE    KLAAR
            INC    INDIR+1
KLAAR       NOP
```

In dit geval zijn INDIR en INDIR+1 de geheugenlocaties op pagina 0 die het indirecte adres bevatten.

Voorbeeld

Laten we aannemen dat (INDIR) = 80_{16} en (INDIR+1) = $4C_{16}$. Het initiele basisadres is dus $4C80_{16}$. Als de lus het adres (INDIR),Y aanroept, is het effectieve adres (INDIR+1) (INDIR) + (Y) of $4C80_{16}$ + (Y). Als Y = FF_{16}, is het effectieve adres

$$4C80_{16} + (Y) = 4C80_{16} + FF_{16} = 4D7F_{16}$$

De hierboven getoonde reeks om de index en het indirecte adres te verhogen, geeft het resultaat

$$(Y) = (Y) + 1 = 00$$
$$(INDIR+1) = (INDIR+1) + 1 = 4D_{16}$$

Het effectieve adres voor de volgende herhaling zal

$$4D80_{16} + (Y) = 4D80_{16} + 00_{16} = 4D80_{16}$$

zijn, hetgeen het volgende hogere adres in de normale opeenvolgende reeks is.

NEVENEFFECTEN

Enkele van de neveneffecten die de programmeur moet onthouden, zijn

. Het wijzigen van de negatief- en nulvlag door laad- en overdrachtinstructies als LDA, LDX, LDY, PLA, TAX, TAY, TSX, TXA en TYA.

. Dat de uitkomst van de instructies ADC en SBC afhankelijk is van de stand van de overdracht- en Decimale Modus vlaggen.

. Het speciale gebruik van de negatief- en overloopvlaggen door de BIT-instructie.

. Het gebruik van het geheugenadres dat 1 groter is dan die welke is aangegeven in de indirecte, de indirect geindexeerde en de geindexeerde indirecte adresseringsmodi.

. Het veranderen van de stack-wijzer door PHA, PHP, PLA, PLP, JSR, RTS, RTI en BRK. Onthoud dat JSR en RTS de stack-wijzer 2 hoger of lager zetten, BRK en RTI deze met 3 veranderen.

. Dat een JSR-instructie het terugkeeradres minus 1 opslaat en de RTS-instructie 1 bij het herstelde terugkeeradres optelt.

. Dat ROL en ROR de overdracht gebruiken. De rotatie wordt met 9, in plaats van 8 bits uitgevoerd.

Voorbeelden

1. LDX $40

Deze instructie beinvloedt de negatief- en de nulvlaggen, zodat deze vlaggen niet langer de waarde in de accumulator of het resultaat van de laatste bewerking weergeven.

2. ADC #$20

Deze instructie telt zowel in de overdrachtvlag als de onmiddellijke data (20_{16}) op. Het resultaat zal binair zijn als de Decimale Modus vlag gewist is, maar BCD als de Decimale Modus vlag gezet is.

3. BIT $1700

Deze instructie zet de overloopvlag naar de waarde van bit 6 van geheugenlocatie 1700_{16}. Dit is de enige instructie die een volkomen onvoorspelbaar effect op die vlag heeft.

4. JMP ($1C00)

Deze instructie draagt de controle over aan het adres dat in de geheugenlocaties $1C00_{16}$ en $1C01_{16}$ (HWB in $1C01_{16}$) staat. Onthoud dat $1C01_{16}$, hoewel niet aangegeven, wel bij deze routine is betrokken. Dit omdat indirecte adressen altijd twee geheugenbytes beslaan.

5. PHA

Deze instructie slaat niet alleen de accumulator in het geheugen op, maar verlaagt de stack-wijzer ook met 1.

6. RTS

Deze instructie laadt niet alleen de programmateller met de twee bovenste waarden in de stack, maar verhoogt ook de stack-wijzer met 2 en de programmateller met 1.

7. ROR A

Deze instructie roteert de accumulator 1 bit naar rechts, waarbij de voormalige inhoud van bit 0 in de overdracht wordt gezet en de voormalige inhoud van de overdracht in bit-positie 7 terechtkomt.

INITIALISATIEFOUTEN

De initialisatieroutines moeten, voor zowel de computer in zijn geheel als specifieke routines, de volgende taken uitvoeren:

. Alle RAM-locaties met beginwaarden laden. Hierbij horen indirecte adressen en andere tijdelijke opslag op pagina 0. De programmeur mag niet veronderstellen dat een geheugenlocatie 0 bevat, alleen omdat hij deze nog niet heeft gebruikt.

. Laad alle registers en vlaggen met initiele waarden. Reset initialiseert alleen de Interrupt Disable vlag (naar 1). Onthoud vooral de noodzaak de Decimale Modus vlag (in de meeste gevallen met CLD) en de stack-wijzer (d.m.v. de LDX, TXS reeks) te initialiseren.

. Laad alle tellers en indirecte adressen met initiele waarden. Let vooral op adressen op pagina 0, die worden gebruikt in of de indirect geindexeerde (gepostindexeerde) adresseringsmodus of de geindexeerde indirecte (gepreindexeerde) adresseringsmodus.

HET PROGRAMMA VERKEERD ORGANISEREN

De volgende problemen komen het meeste voor:

. Een register, vlag of geheugenlocatie niet initialiseren. De programmeur mag er niet van uitgaan dat een register, vlag of geheugenlocatie 0 bevatten, alleen omdat ze nog niet zijn gebruikt.

. Per ongeluk een register, vlag, geheugenlocatie, index, teller of indirect adres opnieuw initialiseren. Zorg ervoor dat de vertakkingen van het programma geen volledige of gedeeltelijke herinitialisatie teweegbrengen.

. Vergeten indexen, tellers of indirecte adressen aan te passen. De veroorzaker van een fout zou in dit geval een programmapad kunnen zijn dat langs de instructies voert die de aanpassing(en) verzorgen of enkele voorwaarden verandert alvorens de instructies uit te voeren.

. Vergeten tussentijdse of uiteindelijke resultaten op te slaan. Het is verrassend makkelijk om een resultaat te berekenen en vervolgens iets anders in de accumulator te laden. Fouten als deze zijn erg moeilijk te achterhalen, omdat alle instructies die het resultaat berekenen keurig werken, terwijl

het resultaat zelf verdwijnt. Oorzaak van een dergelijk probleem kan ook een vertakking zijn die de controle aan een routine, waarin het zojuist berekende resultaat wordt overschreven, geeft.

. Vergeten rond instructies te vertakken die in een bepaald programmapad niet mogen worden uitgevoerd. Onthoud dat de computer de instructies na elkaar uitvoert, tenzij hij de opdracht krijgt dat anders te doen. Zodoende kan het makkelijk gebeuren dat het programma naar een deel doorloopt, dat eigenlijk alleen via een vertakinstructie bereikt had mogen worden. Een belangrijk nadeel van de 6502 is dat deze geen onvoorwaardelijke relatieve sprongen kent; de programmeur is verplicht om of JMP met absolute adressering te gebruiken, of een voorwaarde te scheppen en te vertakken op het voldoen aan deze voorwaarde (SEC, BCS BEST en CLV, BVC BEST).

FOUTENHERKENNING DOOR ASSEMBLERS

De meeste assemblers zullen de volgende veelvoorkomende fouten onmiddellijk herkennen:

. Niet gedefinieerde instructie (meestal door foutieve spelling)
. Niet gedefinieerde naam (meestal weggelaten of fout gespeld)
. Illegaal karakter (bij voorbeeld een 2 in een binair getal of een B in een decimaal getal)
. Illegaal formaat (bij voorbeeld een onjuist afgrenskarakter of een verkeerd register of verkeerde operand)
. Illegale waarde (meestal een getal dat te groot is voor 8 of 16 bits)
. Ontbrekende operand
. Dubbele definiering (twee verschillende waarden aan dezelfde naam toegekend)
. Illegaal label (bij voorbeeld een label toegekend aan een pseudo-bewerking die geen label toestaat)
. Ontbrekend label (bij voorbeeld bij een pseudo-bewerking die er wel een nodig heeft)

Dit soort fouten is meestal makkelijk te corrigeren. Vaak ontstaat een serie verwarrende fouten alleen omdat er een puntkomma of een afgrenzend karakter ontbreekt voor een uitleg. De assembler heeft een dergelijk karakter nodig om commentaar en programma te kunnen scheiden.
Er is echter ook een groot aantal fouten dat een assembler niet herkent. De programmeur moet er op attent zijn dat zijn of haar programma zulke fouten kan bevatten, ook al meldt de assembler ze niet. Typische voorbeelden zijn:

. Ontbrekende regels. Een assembler weet uiteraard niet dat er regels ontbreken, tenzij een van die regels een label of een definitie bevat die later in het programma wordt gebruikt. De regels die het makkelijkst vergeten worden zijn herhalingen (een of meerdere regels die hetzelfde zijn of routines die op dezelfde manier beginnen) of instructies die overbodig lijken te zijn. Tot de herhalingen behoren reeksen van schuivingen, vertakkingen, verhogingen en verlagingen. Tot de instructies die wel eens overbodig lijken, behoren CLC, SEC, enz.

. Ontbrekende notatietekens. De assembler kan niet weten dat er een #, H, $, B of % voor dan wel achter een naam ontbreekt, tenzij het weglaten tot een illegaal karakter leidt (zoals bij voorbeeld C in een decimaal getal). De assembler gaat er standaard van uit dat alle adressen direct en alle getallen decimaal zijn. Problemen ontstaan natuurlijk vooral bij getallen die zowel in decimale als in hexadecimale vorm geldig zijn (44 of 2050, bij voorbeeld) en met binaire getallen (als 00000110).

. Verkeerde spelling die toch geldig is. Als de programmeur bij voorbeeld BCC in plaats van BCS, LDX in plaats van LDY of SEC in plaats van SED typt. Tenzij de verkeerde spelling ongeldig is, weet de assembler niet wat de bedoeling feitelijk was. Geldige misspellingen komen

vaak voor als de programmeur veel op elkaar lijkende labels (bij voorbeeld XXX en XXX, L121 en L112 of VAR1I en VARII) gebruikt.

. Instructies als commentaar noteren. Als de programmeur een puntkomma aan het begin van een instructieregel plaatst, behandelt de assembler deze regel als commentaar. Dit kan een frustrerende fout zijn, omdat de regel in de listing verschijnt maar niet tot objectcode wordt geassembleerd.

Soms is het mogelijk de assembler te verwarren door ongeldige instructies te geven. Het is mogelijk dat een assembler een volkomen onlogische instructie accepteert, alleen omdat de schrijver van de assembler niet met deze mogelijkheid rekening gehouden heeft. Het resultaat (te vergelijken met wat er gebeurt als iemand het nummer van een rijbewijs in plaats van een girorekening krijgt) is onvoorspelbaar. Gevallen waarbij een 6502 in de fout zou kunnen gaan zijn:

. Als de programmeur een onmogelijk register of een pnmogelijke adresseringsmodus aangeeft. Enkele assemblers accepteren instructies als INC A, LDA ($40),X of LDY BASIS,Y. Zij geven zonder enige melding foutieve objectcode.

. Als de programmeur een ongeldig cijfer intikt, zoals een Q in decimale of hexadecimale getallen, of een 7 in een binair getal. Enkele assemblers zullen willekeurige waarden aan deze foute getallen toekennen.

. Als de programmeur een ongeldige operand (bij voorbeeld LDA #$HX) aangeeft. Enkele assemblers accepteren dit en produceren onjuiste code.

De assembler herkent alleen fouten die de auteur ervan heeft voorzien. Programmeurs zijn echter vaak in staat fouten te maken die de auteur nooit bedacht had, net als dat automobilisten vaak in staat zijn handelingen te verrichten die de ontwerper van de snelweg of de verkeersonderzoeker van zijn leven niet had kunnen bedenken. Onthoud dat dit soort fouten alleen uit een programma te halen is, door het programma stap-voor-stap, regel-voor-regel na te lopen.

FOUTEN IN DE UITVOERING

Zo nu en dan werken de instructies van een microprocessor niet op de manier die de ontwerpers of de gebruikers ervan hadden verwacht. De 6502 heeft een bekende uitvoeringsfout die, gelukkig, vrij zeldzaam is. De instructie JMP ($XXFF), waarbij XX voor een paginanummer staat, wordt niet goed uitgevoerd. Men zou verwachten dat de instructie het bestemmingsadres uit de geheugenlocaties XXFF en (XX+1)00 zou halen. Echter, de processor verhoogt kennelijk niet de hoogstwaardige byte van het adres. Het bestemmingsadres wordt daarom uit de geheugenlocatie XXFF en XX00 gehaald. Een JMP ($1CFF), bij voorbeeld, zal naar het adres dat in de geheugenlocaties $1CFF_{16}$ (LWB) en $1C00_{16}$ (HWB) is opgeslagen springen. De meeste assemblers gaan ervan uit dat de programmeur er zelf op let dat geen enkele indirecte sprong het bestemmingsadres over een paginagrens heen haalt.

VEEL VOORKOMENDE FOUTEN IN I/O-AANSTURINGSROUTINES

Tot de meeste fouten in I/O-aansturingsroutines behoren zowel hardware- als software-fouten. Zij zijn dus moeilijk in categorieen onder te brengen. Enkele fouten waarvoor de programmeur moet waken zijn:

. In- en output-poorten verwarren. Veel I/O-interfaces gebruiken de LEES/SCHRIJF-lijn voor adressering, dus leidt het lezen en beschrijven van hetzelfde geheugenadres tot handelingen op verschillende fysieke registers. Zelfs als dit niet is gebeurd, kan het nog onmogelijk zijn om outputdata terug te lezen, tenzij deze gelatcht en gebufferd zijn.

. Handelingen proberen te verrichten die fysiek onmogelijk zijn. Het lezen van data uit een output-apparaat (zoals een beeldscherm) of het sturen van data naar een input-apparaat (zoals een toetsenbord) is onzin. Per ongeluk het verkeerde adres gebruiken veroorzaakt geen assembleer-fouten; het adres is per slot van rekening geldig en de assembler kan niet weten dat bepaalde handelingen niet op dat adres verricht kunnen worden. Evenzo zou een programma pogingen kunnen doen om data in een adres dat niet bestaat of in ROM op te slaan.

. Geen rekening houden met zijdelingse hardware-effecten. Soms kan het oversturen van data van of naar een poort, de statuslijnen vanzelf veranderen. In het bijzonder als er gebruik wordt gemaakt van een 6520 of 6522 parallelle interface. Zelfs het lezen of schrijven van een poort terwijl een programma foutvrij wordt gemaakt, zal de statuslijnen veranderen. Wees vooral voorzichtig met instructies als vergelijkingen en BIT. Deze instructies lezen een geheugenadres zelfs als zij geen enkel register veranderen. Let ook op verlaag-, verhoog- en schuifinstructies, die een geheugenadres zowel lezen als schrijven (de feitelijke bewerking vindt uiteraard binnen de processor plaats).

. Lezen en schrijven zonder de status te controleren. Veel apparaten kunnen alleen data ontvangen of sturen als een statuslijn aangeeft dat zij klaar staan. Onvoorspelbare resultaten ontstaan als er op andere momenten data gestuurd of gelezen worden.

. De verschillen tussen input en output negeren. Onthoud dat een input-apparaat normaal gesproken in een niet klaar stand begint - het heeft geen data beschikbaar, hoewel de computer klaar staat om deze te ontvangen. Anderzijds begint een output-apparaat meestal in een klaar-stand. Met andere woorden, het kan data ontvangen, maar de computer heeft meestal geen data om te versturen. In veel situaties, in het bijzonder bij het gebruik van de 6520, 6522, 6551 en 6850 interfaces, zal de programmeur de output-poorten in eerste instantie moeten uitschakelen of een null karakter (een dat geen effect heeft) naar elke output poort moeten sturen, alleen om hun status van klaar in niet klaar te veranderen.

. Vergeten kopieen van output-data te bewaren. Onthoud dat het niet altijd mogelijk is de verstuurde data ook weer van de output-poort terug te lezen. Als het later nodig is (als de data niet correct zijn ontvangen) de data opnieuw te sturen, een deel ervan te veranderen (om een of meerdere waarschuwingslampjes op dezelfde poort aan of uit te zetten) of om ze te bewaren als een onderdeel van de geinterrumpeerde status (de data zijn het huidige prioriteitsniveau), moet de programmeur een kopie van de data in het geheugen bewaren. De kopie moet elke keer aangepast worden als de feitelijke data zijn veranderd.

. Data lezen voordat deze stabiel zijn of terwijl deze nog gewijzigd worden. De programmeur moet ervoor zorgen dat hij of zij zeker weet wanneer het input apparaat stabiele data levert. Als er mogelijk sprake is van denderende schakelaars, kan het nuttig zijn om ze twee keer te lezen (met een pauze groter dan de tijd die nodig is om terug te denderen), voordat er enige actie op de stand van de schakelaars wordt ondernemen. Als er toetsen denderen kan het verstandig zijn om te wachten tot een toets wordt losgelaten, in plaats van te reageren als deze wordt ingedrukt. Handelen op het moment dat de toets wordt losgelaten dwingt de bediener ook de toets los te laten, in plaats van deze ingedrukt te houden. Als er sprake is van een onophoudelijke datastroom (zoals bij seriele I/O), is het het beste om de data vlak bij het middelpunt van de pulsen te lezen, in plaats van bij de randen. Daar kunnen de waarden veranderen.

. Nalaten de polariteit om te draaien van data die van of naar apparaten met een negatieve logica worden gestuurd. Vele eenvoudige I/O-apparaten, zoals schakelaars en displays, gebruiken negatieve logica. Een logische 0 betekent dat een schakelaar gesloten is of een display aan staat. Veelgebruikte schakelaars met tien posities of draaischijven produceren vaak data in negatieve logica. De oplossing is eenvoudig: complementeer de data (met EOR #$FF) na het lezen of voor het sturen.

. Feitelijke I/O-poorten verwarren met registers binnen de I/O-apparaten. Programmeerbare I/O-apparaten als de 6520, 6522, 6551 en de 6850 hebben controle- of commandoregisters die bepalen hoe het apparaat werkt en statusregisters die de huidige status van het apparaat of de verzending aangeven. Deze registers bevinden zich binnen in de I/O-apparaten; zij zijn niet met de randapparatuur verbonden. Data naar of van het status- of controle register versturen is niet hetzelfde als data naar of van de feitelijke I/O-poort sturen.

. Onjuist gebruik van bidirectionele poorten. Veel apparaten als de 6520, 6522, 6530 en de 6532 hebben bidirectionele I/O-poorten, ofte wel poorten met tweerichtingsverkeer. De poorten (en mogelijk zelfs de afzonderlijke lijnen) kunnen voor zowel in- als output worden gebruikt. Om toevallige uitsturing van data te voorkomen, worden deze poorten bij een reset van de computer meestal als input-poorten geschakeld. Als het nodig is, moeten deze poorten dus expliciet als output-poort ingesteld worden. Wees voorzichtig met het lezen van bits of poorten die als outputs zijn ingesteld of bij het schrijven naar bits of poorten die als inputs zijn ingesteld. De enige manier om te weten wat er zal gebeuren is de handleiding van het specifieke apparaat te lezen.

. Nalaten de status te wissen na een I/O-handeling. Als de processor eenmaal data van een input-poort heeft gelezen, mag deze poort nooit terugkeren naar de niet klaar stand. Evenzo mag een output-poort, nadat de processor er informatie naartoe heeft gestuurd, nooit terugkeren naar de niet klaar stand. Sommige I/O-apparaten veranderen de status van hun poorten automatisch na input- of outputhandelingen. Andere (zoals de 6520) doen dit niet of wijzigen de status alleen automatisch na een input-handeling. Als de status ongewijzigd blijft, kan dit leiden tot een eindeloze lus of andere, zeer vreemde, resultaten.

VEEL VOORKOMENDE FOUTEN IN INTERRUPT-AFHANDELINGSROUTINES

Veel fouten op het gebied van interrupts, hebben te maken met zowel hard- als software. Fouten kunnen echter ook zijn:

. Nalaten de interrupts weer in te schakelen tijdens de afhandelingsroutine. De 6502 processor schakelt de interrupts automatisch uit, nadat hij er een heeft ontvangen. Hij schakelt de interrupts weer in nadat de RTI instructie is uitgevoerd, omdat RTI het statusregister vanuit de stack herstelt.

. Nalaten de registers op te slaan en te herstellen. De 6502 bewaart, met uitzondering van de programmateller en het statusregister, geen enkel register automatisch. De accumulator, indexregisters en tijdelijke opslaglocaties moeten expliciet op de stack bewaard worden.

. De registers in de verkeerde volgorde opslaan of herstellen. De registers moeten in de omgekeerde volgorde worden hersteld als waarin zij zijn opgeslagen.

. Interrupts inschakelen zonder de prioriteiten en andere parameters van het interrupt systeem in te stellen.

. Vergeten dat het statusregister en de programmateller boven op de stack gezet worden als een van de reacties van de processor op een interrupt. Het statusregister staat bovenin en de waarde van de programmateller is het feitelijke terugkeeradres. Deze situatie verschilt hierin met subroutines, waarbij het feitelijke terugkeeradres minus 1 boven op de stack staat.

. Nalaten de interrupts uit te schakelen tijdens de overdracht van meerdere bytes of het uitvoeren van (reeksen van) instructies die niet onderbroken kunnen of mogen worden. In het bijzonder moet de programmeur voorkomen dat data (als bij voorbeeld tijd) die een interrupt-afhandelingsroutine mogelijk gebruikt, slechts gedeeltelijk worden aangepast. Over het algemeen moeten de interrupts uitgeschakeld zijn als het hoofdprogramma geheugenlocaties verandert, die het samen met de afhandelingsroutine gebruikt.

. Nalaten de interrupts weer in te schakelen na een routine die alleen met uitgeschakelde interrupts mag werken. Een probleem dat hier automatisch op volgt, ontstaat als de programmeur niet wil dat de interrupts worden ingeschakeld indien deze niet waren ingeschakeld op het moment dat de routine werd aangeroepen. Om dit op te lossen moet de status van de Interrupt Disable vlag (met PHP) bewaard worden, voordat de routine wordt uitgevoerd. Een PLP instructie na de routine herstelt de oorspronkelijke status van de I-vlag. Onthoud echter dat PLP het gehele statusregister terughaalt.

. Nalaten de waarde van de Decimale Modus-vlag te initialiseren of te bepalen. Een interrupt-afhandelingsroutine mag niet van een bepaalde stand (0) van de D-vlag uitgaan. Deze moet echter de vlag met CLD of SED initialiseren als er ADC of SBC instructies uitgevoerd worden. Het is niet nodig de stand van de D-vlag op te slaan en te herstellen, omdat dit vanzelf gebeurt als een onderdeel van het opslaan en herstellen van het statusregister. Het initialiseren van de D-vlag voorkomt problemen als de interrupt- afhandelingsroutine wordt uitgevoerd als onderdeel van een programma dat de vlag had gezet.

. Nalaten het signaal te wissen dat de interrupt genereerde. De afhandelingsroutine moet de interrupt wissen, ook al is er geen onmiddellijke respons of een input- dan wel output-handeling nodig. Ook al heeft de processor bij voorbeeld geen data om naar een interrumperend output apparaat te sturen, dan nog moet de interrupt gewist of uitgeschakeld worden. Anders komt de processor in een oneindige lus terecht. Evenzo heeft een real-time klok geen andere afhandeling nodig dan het aanpassen van de tijd, maar toch moet de afhandelingsroutine de klok-interrupt wissen. Dit kan gedaan worden door bij voorbeeld een 6520/6522 I/O-poort of timer uit te lezen.

. Nalaten met het hoofdprogramma te communiceren. Het hoofdprogramma weet niet dat de interrupt is afgehandeld, tenzij dit expliciet wordt doorgegeven. Dit geschiedt over het algemeen door de afhandelingsroutine een vlag te laten zetten die door het hoofdprogramma kan worden bekeken. Dan weet het hoofdprogramma dat de afhandelingsroutine uitgevoerd is. Deze eenvoudige procedure houdt overigens wel in dat het hoofdprogramma de vlag vaak genoeg moet bekijken om te voorkomen dat het data mist. Uiteraard kan de programmeur altijd een tijdelijke opslagruimte (of buffer) creeren die veel data kan bevatten.

. Nalaten de prioriteit op te slaan en te herstellen. De prioriteit van een interrupt wordt vaak vastgehouden in een register dat alleen beschreven kan worden. Die prioriteit moet, net als de registers, opgeslagen en aan het einde van de afhandelingsroutine hersteld worden. Als het prioriteitsregister alleen beschreven kan worden, moet een kopie van de inhoud in het geheugen worden bewaard.

Inleiding tot de programmahoofdstukken

De programmahoofdstukken bevatten machinetaalsubroutines voor de 6502 processor. Elke subroutine is gedocumenteerd met een inleiding en commentaar; bij elke routine staat op z'n minst een voorbeeld van haar gebruik. De inleidende beschrijving bevat de volgende informatie:

1. Het doel van de routine
2. De gevolgde procedure
3. De gebruikte registers
4. De uitvoeringstijd
5. De programma grootte
6. Het benodigde data geheugen
7. Speciale gevallen
8. Begincondities
9. Eindcondities
10. Voorbeelden.

We hebben elke routine zo algemeen mogelijk gehouden. Dit is het moeilijkst in geval van I/O- en interrupt-afhandelingsroutines zoals die in de hoofdstukken 10 en 11 staan, omdat deze routines in de praktijk computer-gebonden zijn. In dergelijke gevallen hebben we de computer-gebondenheid beperkt tot algemeen geldende input- en output-routines en interrupt regelaars. We hebben specifieke voorbeelden van de populaire Apple II computer genomen, maar de algemene principes gelden ook voor andere, op de 6502 gebaseerde computers.
In alle routines hebben we de volgende technieken gebruikt om parameters door te geven:

1. Een enkele 8-bits parameter wordt via de accumulator overgegeven. Een tweede 8-bits parameter via indexregister Y.

2. Een enkele 16-bits parameter wordt via de accumulator en indexregister Y doorgegeven, met de hoogstwaardige byte via de accumulator. Een begeleidende 8-bits parameter wordt via indexregister X doorgegeven.

3. Grotere aantallen parameters worden direct dan wel indirect via de stack doorgegeven. We gaan ervan uit dat de routines via een JSR aangeroepen worden. De JSR plaatst het terugkeeradres boven in de stack en dus boven op de parameters.

Waar er gekozen moest worden tussen looptijd en geheugengebruik, hebben we gekozen voor de methode met de kortste looptijd. Arrays die groter dan 256 bytes zijn, bij voorbeeld, kunnen sneller als hele pagina's worden verwerkt, waarbij het resterende deel van de pagina apart wordt behandeld, dan wanneer de array in een enkele reeks wordt verwerkt. De reden hiervoor is dat bij de eerste methode een 8-bits teller in een indexregister kan worden gebruikt, terwijl de tweede methode een 16-bits teller in het geheugen nodig heeft.
We hebben ook gekozen voor de methode met een gering aantal herhalende berekeningen. In het geval van array-indexering, bij voorbeeld, waarbij het aantal bytes tussen de beginadressen van de elementen die in een bepaald subscript slechts 1 verschillen (bekend als de grootte van het subscript) alleen afhangt van het aantal bytes per element en de grenzen van het array. Zodoende kunnen de grootten van de verschillende subscripts berekend worden zodra de grenzen van het

array bekend zijn. De grootten worden daarom gebruikt als parameters voor de indexeringsroutines, zodat zij niet telkens berekend hoeven te worden voor de indexering van een bepaald array.
De looptijd (uitvoeringstijd) van de meeste kortere routines hebben we gespecificeerd. Voor langere routines hebben we een geschatte looptijd aangegeven. De looptijd van programma's met veel vertakkingen hangt uiteraard af van welk pad in een bepaald geval wordt gekozen. De 6502 zorgt hierbij voor verdere complicaties, omdat vertakkingsinstructies zelf verschillende klokcycli nodig hebben. Dit hangt ervan af of de vertakking al dan niet wordt genomen en of deze binnen of buiten de huidige pagina wordt genomen. Zodoende is een precieze looptijd vaak moeilijk aan te geven. De documentatie bevat ten minste een typerend voorbeeld waarbij een geschatte of maximum looptijd wordt aangegeven.
Onze filosofie met betrekking tot foutindicaties en bijzondere gevallen is:

1. Routines moeten een makkelijk te testen indicator (als de overdrachtvlag) gebruiken om aan te geven of er wel of geen fouten zijn opgetreden.

2. Nulwaarde-situaties - als er geen elementen in een array staan of als er strings met nul lengte zijn - dienen tot een onmiddellijk einde te leiden, met minimaal effect op de onderliggende data.

3. Verkeerd gespecificeerde data - als een maximale stringlengte van nul of een index voorbij het einde van het array - dienen tot een onmiddellijk einde te leiden, met een minimaal effect op de onderliggende data.

4. De documentatie moet een samenvatting van fouten en uitzonderingen bevatten (onder het kopje Speciale gevallen).

5. Uitzonderingen die feitelijk handig zijn voor de gebruiker (zoals het wissen van meer karakters dan een string nog kon bevatten, in plaats van het precieze aantal tellen) moeten op een redelijke manier afgehandeld worden. Desalniettemin moeten ze nog steeds als fout aangegeven worden.

Uiteraard kan geen enkele manier om fouten of uitzonderingen te verwerken volkomen consequent of voor elke toepassing geschikt zijn. Wij zijn ervan uit-gegaan dat een redelijk aantal subroutines deze situaties moet afhandelen, in plaats van ze te negeren of aan te nemen dat de gebruiker steeds de juiste gegevens aanbiedt.

De subroutines zijn als volgt ondergebracht:

Codeconversie

4A Binair naar BCD conversie 165
4B BCD naar binair conversie 168
4C Binair naar hexadecimaal ASCII conversie 170
4D Hexadecimaal ASCII naar binair conversie 173
4E Conversie van een binair getal naar decimaal ASCII 176
4F Conversie van decimaal ASCII naar binair 182
4G Kleine letterASCII naar Hoofdletter ASCII conversie 187
4H ASCII naar EBCDIC conversie 189
4I EBCDIC naar ASCII conversie 192

Array Manipulatie en Indexering

5A Geheugenvulling 195
5B Blok verplaatsing 199
5C Eendimensionale byte array indexering 205
5D Eendimensionale woord array indexering 208
5E Tweedimensionale byte array indexering 211
5F Tweedimensionale woord array indexering 216
5G N-dimensionale array indexering 221

Rekenkunde

6A 16-Bits optelling 231
6B 16-Bits aftrekking 234
6C 16-Bits vermenigvuldiging 237
6D 16-Bits deling 241
6E 16-Bits vergelijking 250
6F Meervoudige precisie binaire optelling 255
6G Meervoudige precisie binaire aftrekking 259
6H Meervoudige precisie binaire vermenigvuldiging 263
6I Meervoudige precisie binaire deling 270
6J Meervoudige precisie binaire vergelijking 278
6K Meervoudige precisie decimale optelling 283
6L Meervoudige precisie decimale aftrekking 287
6M Meervoudige precisie decimale vermenigvuldiging 291
6N Meervoudige precisie decimale deling 298
6O Meervoudige precisie decimale vergelijking 307

Bit Manipulatie en Schuivingen

7A Zet bit 309
7B Wis bit 312
7C Bit test 315
7D Bitveld uittreksel 318
7E Bitveld tussenvoeging 323
7F Meervoudige precisie rekenkundige schuiving rechts 328
7G Meervoudige precisie logische schuiving links 333
7H Meervoudige precisie logische schuiving rechts 338
7I Meervoudige precisie rotatie rechts 343
7J Meervoudige precisie rotatie links 348

String Manipulatie

8A String vergelijking 353
8B String rijging 358
8C Zoekt de positie van een substring 364

8D Kopieer een substring van een string 370
8E Haal een substring uit een string 377
8F Voeg een substring in een string 383

Array Bewerkingen

9A 8-Bits array optelling 391
9B 16-Bits array optelling 395
9C Zoek element met maximale byte lengte 399
9D Zoek element met minimale byte lengte 403
9E Binair zoeken 407
9F 'Bubble sort' 414
9G RAM test 419
9H Sprongtabel 427

Input/Output

10A Lees een regel karakters uit een terminal 431
10B Schrijf een regel karakters naar een uitvoer apparaat 438
10C Genereer even pariteit 441
10D Controleer pariteit 444
10E CRC-16 controle en uitvoering 447
10F I/O apparaatttabel verwerker 453
10G Initialiseer I/O poorten 469
10H Vertraag milliseconden 475

Interrupts

11A Interrupt-gestuurde invoer/uitvoer zonder buffer voor een 6850 ACIA 479
11B Interrupt-gestuurde invoer/uitvoer zonder buffer voor een 6522 VIA 488
11C Interrupt-gestuurde invoer/uitvoer met buffer voor een 6850 ACIA 497
11D Realtime klok en kalender 508

Binair naar BCD conversie - 4A

Converteert een byte binaire data naar twee bytes BCD-data.
Procedure: Het programma trekt herhaald 100 van de originele data af om het aantal honderdtallen te bepalen, trekt dan herhaald 10 van de rest af om het aantal tientallen te bepalen en schuift ten slotte het cijfer dat de tientallen aangeeft vier posities naar links en combineert dit met het cijfer dat de eenheden aangeeft.

Gebruikte registers:	Alle
Looptijd:	Maximaal 133 klokcycli, afhankelijk van het aantal aftrekkingen dat nodig is om de tien- en hondertallen te bepalen.
Programmalengte:	38 bytes
Benodigd datageheugen:	Een byte ergens in RAM (adres TEMP)

Begin conditie

Binaire data in de accumulator.

Eindconditie

Het cijfer dat de honderdtallen aangeeft in de accumulator.
Het cijfer dat de tientallen en eenheden aangeeft in het indexregister Y.

Voorbeelden

1. Data: (A) = $6E_{16}$ (110 decimaal)

   Resultaat: (A) = 01_{16} (het cijfer dat de honderdtallen aangeeft)
   (Y) = 10_{16} (het cijfer dat de tientallen en eenheden aangeeft)

2. Data: (A) = $B7_{16}$ (183 decimaal)

   Resultaat: (A) = 01_{16} (het cijfer dat de honderdtallen aangeeft)
   (Y) = 83_{16} (het cijfer dat de tientallen en eenheden aangeeft)

```
;
;     Titel     : Binair naar BCD conversie
;     Naam      : BN2BCD
;
;
;     Doel      : Vertaal een byte binaire data naar twee bytes in BCD-
;                 formaat
;
;
;     Begin     : Register A = Binaire data
;
;     Eind      : Register A = hoogstwaardige byte van de BCD-data
```

```
;                 Register y = laagstwaardige byte van de BCD-data
;
;     Registers : Alle
;
;     Tijd      : 133 cycli
;
;     Grootte   : programma 38  bytes
;                 data       1  bytes

BN2BCD:
          ; Bereken de honderdtallen
          ;   Deel door honderd
          ; Y = Quotient
          ; A = Rest

                    ldy     #0FFH         ; Begin te delen met -1
                    sec                   ; Zet de overdrachtvlag voor de
                                          ; eerste aftrekking

D100LP:
                    iny                   ; Verhoog het quotient met 1
                    sbc     #100          ; Trek er honderd vanaf
                    bcs     D100LP        ; Spring zolang A groter dan 100 is
                    adc     #100          ; Tel de laatste 100 er weer bij op
                    tax                   ; Bewaar de Rest
                    tay
                    pha                   ; Bewaar de honderdtallen op de
                                          ; stack
                    txa                   ; Haal de Rest op

          ; Bereken de tientallen en de eenheden
          ;   Deel de rest verkregen bij de honderdtallen door 10
          ; Y = Quotient
          ; A = Eenheden

                    ldy     #0FFH         ; Begin te delen met -1
                    sec                   ; Set de overdrachtvlag voor de
                                          ; eerste aftrekking

D10LP:
                    iny                   ; Verhoog het quotient met 1
                    sbc     #10
                    bcs     D10LP         ; Spring zolang A groter dan 10 is
                    adc     #10           ; Tel de laatste 10 er weer bij op

          ; Combineer de eenheden en de tientallen

                    sta     TEMP          ; Bewaar de eenheden
                    tya                   ; Haal de tientallen op
                    asl     A
                    asl     A
                    asl     A
                    asl     A             ; Schuif de tientallen in de
                                          ; bovenste 4 bits van A
                    ora     TEMP          ; Zet de eenheden op hun plaats

          ; Keer terug met in Y de laagstwaardige byte en in A de meest
          ; significante byte

                    tay                   ; Stop de laagstwaardige byte in
                                          ; het Y-register
                    pla                   ; Haal de honderdtallen op
                    rts
```

```
; Data
TEMP:           .BLOCK  1           ; Tijdelijk gebruikt om de
                                    ; tientallen en de eenheden te
                                    ; combineren


; Voorbeeld uitvoering


SC0401:
        ; Vertaal 0A hexadecimaal naar 10 BCD

                lda     #0AH
                jsr     BN2BCD
                brk                 ; A = 0 ; Y = 10H

        ; Vertaal FF hexadecimaal naar 255 BCD

                lda     #0FFH
                jsr     BN2BCD
                brk                 ; A = 02H ; Y = 55H

        ; Vertaal 0 Hexadecimaal naar 0 BCD

                lda     #0
                jsr     BN2BCD
                brk                 ; A = 0 ; Y = 0

                .END
```

BCD naar binair conversie - 4B

Converteert een byte met BCD-data naar een byte met binaire data.
Procedure: Het programma maskeert de hoogstwaardige digit, vermenigvuldigt het met 10 door te schuiven (10 = 8 + 2, en vermenigvuldigen met twee of met acht is hetzelfde als drie plaatsen, respectievelijk een plaats naar links schuiven) en telt het produkt op bij het laagstwaardige digit.

Gebruikte registers:	A, P, Y
Looptijd:	38 cycli
Programmalengte:	24 bytes
Benodigd datageheugen:	Een byte ergens in RAM (adres TEMP)

Begincondities

BCD-data in de accumulator.

Eindcondities

Binaire data in de accumulator.

Voorbeelden

1. Data: (A) = 99_{16}
   Resultaat: (A) = $63_{16} = 99_{10}$

2. Data: (A) = 23_{16}
   Resultaat: (A) = $17_{16} = 23_{10}$

```
;
;
;       Titel     : BCD naar binair conversie
;       Naam      : BCD2BN
;
;
;       Doel      : Het vertalen van een byte in BCD-formaat naar binair
;
;
;       Begin     : Register A = BCD-data
;
;       Eind      : Register A = binaire data
;
;       Registers : A, P, Y
;
;       Tijd      : 38 cycli
;
```

```
;       Grootte    : programma 24  bytes
;                    data        1  byte

BCD2BN:
          ; Vermenigvuldig de bovenste 4 bits met 10 en bewaar deze
          ;  In TEMP staat dus 10 maal de bovenste 4 bits, ofwel bovenste 4
          ;  bits maal ( 8 plus 2 )

                    tay                   ; Bewaar het origineel
                    and     #0F0H         ; Haal de bovenste 4 bits op
                    lsr     A             ; Deel door 2 hetgeen de bovenste 4
                                          ; bits maal 8 is
                    sta     TEMP          ; Bewaar dit getal
                    lsr     A             ; Deel door 4
                    lsr     A             ; Deel door 8: A bevat nu bovenste
                                          ; 4 bits maal 2
                    clc
                    adc     TEMP
                    sta     TEMP          ; Reg A bevat nu de bovenste 4 bits
                                          ; maal 10

                    tya                   ; Haal de originele waarde weer op
                    and     #0FH          ; Haal de onderste 4 bits op
                    clc
                    adc     TEMP          ; Tel deze bij de bovenste 4 bits
                                          ; op
                    rts

; Data
TEMP:               .BLOCK  1

;
;
; Voorbeeld uitvoering
;
;


SC0402:
          ; Vertaal 0 BCD naar 0 hexadecimaal

                    lda     #0
                    jsr     BCD2BN
                    brk                   ; A = 0

          ; Vertaal 99 BCD naar 63 hexadecimaal

                    lda     #099H
                    jsr     BCD2BN
                    brk                   ; A = 63H

          ; Vertaal 23 BCD naar 17 hexadecimaal

                    lda     #23H
                    jsr     BCD2BN        ; A = 17H

                  .END
```

Binair naar hexadecimaal ASCII conversie - 4C

Converteert een byte met binaire data naar de twee ASCII-karakters die met de twee hexadecimale digits corresponderen.
Procedure: Het programma maskeert elk hexadecimale digit afzonderlijk en converteert het naar het ASCII-equivalent. Hierbij hoeft slechts 30_{16} opgeteld te worden, als het digit decimaal is. Als het digit niet decimaal is, moet er een extra factor zeven bij opgeteld worden om het verschil tussen ASCII 9 (39_{16}) en ASCII A (41_{16}) te overbruggen.

Gebruikte registers:	Alle
Looptijd:	77 cycli plus drie extra cycli voor elk niet-decimaal cijfer
Programmalengte:	31 bytes
Benodigd Data Geheugen:	Geen

Begincondities

Binaire data in de accumulator.

Eindcondities

ASCII-equivalent van het hoogstwaardige hexadecimale cijfer in de accumulator.
ASCII-equivalent van het laagstwaardige hexadecimale cijfer in het indexregister Y.

Voorbeelden

1. Data: (A) = FB_{16}
   Resultaat: (A) = 46_{16} (ASCII F)
   (Y) = 42_{16} (ASCII B)

2. Data: (A) = 59_{16}
   Resultaat: (A) = 35_{16} (ASCII 5)
   (Y) = 39_{16} (ASCII 9)

```
;
;
;     Titel     : Binair naar hexadecimaal ASCII conversie
;     Naam      : BN2HEX
;
;
;     Doel      : Vertaal een byte binaire data in twee ASCII-karakters
;
;
;     Begin     : Register A = binair data
;
;     Eind      : Register A = eerste ASCII-karakter, hoogstwaardige waarde
;                 Register Y = tweede ASCII-karakter, laagstwaardige waarde
```

```
;
;     Registers : Alle
;
;     Tijd      : Ongeveer 77 cycli
;
;     Grootte   : programma 31  bytes
;                 data       0  bytes

BN2HEX:
          ;
          ; Vertaal bovenste 4 bits

                    tax                 ; Bewaar het origineel
                    and     #0F0H       ; Haal de bovenste 4 bits op
                    lsr     A
                    lsr     A
                    lsr     A
                    lsr     A           ; Schuif ze naar de onderkant van
                                        ; de byte
                    jsr     NASCII      ; Vertaal dit naar een ASCII-waarde
                    pha                 ; Bewaar dit op de stack

          ; Vertaal de onderste 4 bits

                    txa
                    and     #0FH        ; Haal de onderste 4 bits op
                    jsr     NASCII      ; Vertaal dit naar een ASCII-waarde

                    tay                 ; Stop de onderste 4 bits in Y
                    pla                 ; Stop de bovenste 4 bits in A
                    rts

;
; Subroutine : NASCII
; Doel       : Vertaal een hexadecimaal cijfer naar ASCII
; Begin      : A = Binaire data in de onderste 4 bits
; Einde      : A = ASCII-karakter
; Registers  : A, P

NASCII:
                    cmp     #10
                    bcc     NAS1        ; Spring als het > 10 is
                    clc
                    adc     #7          ; Tel er anders 7 bij op zodat na
                                        ; het optellen van '0' het karakter
                                        ; in de reeks 'A' ... 'F' zit

NAS1:
                    adc     #'0'        ; Maak er een karakter van
                    rts
;
;
; Voorbeeld uitvoering
;
;
SC0403:
          ; Vertaal 0 naar '00'

                    lda     #0
                    jsr     BN2HEX
                    brk                 ; A = '0' = 30H
                                        ; Y = '0' = 30H
```

```
; Vertaal FF hexadecimaal naar 'FF'

        lda     #0FFH
        jsr     BN2HEX
        brk                     ; A = 'F' = 46H
                                ; Y = 'F' = 46H

; Vertaal 23 hexadecimaal naar '23'

        lda     #23H
        jsr     BN2HEX
        brk                     ; A = '2' = 32H
                                ; Y = '3' = 33H

      .END
```

Hexadecimaal ASCII naar binair conversie - 4D

Converteert twee ASCII-karakters (die twee hexadecimale cijfers voorstellen) tot een byte binaire data.
Procedure: Het programma converteert elk ASCII-karakter afzonderlijk tot een hexadecimaal cijfer. Hiervoor is slechts een eenvoudige aftrekking van 30_{16} (ASCII nul) nodig als het cijfer decimaal is. Als het cijfer niet-decimaal is, moet er een extra factor van 7 afgetrokken worden om het verschil tussen ASCII 9 (39_{16}) en ASCII A (41_{16}) te overbruggen. Het programma schuift vervolgens het hoogstwaardige cijfer vier bits naar links en combineert het met het laagstwaardige cijfer. Het programma controleert niet op de geldigheid van de ASCII-karakters (of de ASCII-karakters bij voorbeeld ook werkelijk hexadecimale cijfers weergeven).

Gebruikte registers:	A, P, Y
Looptijd:	74 cycli plus 3 extra cycli voor elk niet-decimale cijfer
Programmalengte:	30 bytes
Benodigd datageheugen:	Een byte ergens in RAM (adres TEMP)

Begincondities

Hoogstwaardige ASCII-cijfer in de accumulator.
Laagstwaardige ASCII-cijfer in het indexregister Y.

Eindcondities

Binaire data in de accumulator.

Voorbeelden

1.	Data:	(A) = 44_{16} (ASCII D)
		(Y) = 37_{16} (ASCII 7)
	Resultaat:	(A) = $D7_{16}$
2.	Data:	(A) = 31_{16} (ASCII 1)
		(Y) = 42_{16} (ASCII B)
	Resultaat:	(A) = $1B_{16}$

```
;
;
;       Titel       : Hexadecimaal ASCII naar binair conversie
;       Naam        : HEX2BN
;
;
;       Doel        : Vertaal twee ASCII-karakters naar een byte binaire data
;
;
```

```
;       Begin      : Register A = eerste ASCII-karakter
;                    Register Y = tweede ASCII-karakter
;
;       Eind       : Register A = binaire data
;
;       Registers  : A, P, Y
;
;       Tijd       : Ongeveer 74 cycli
;
;       Grootte    : programma 30  bytes
;                    data       1  bytes

HEX2BN:
                     pha                    ; Bewaar het eerste karakter
                     tya                    ; Haal het tweede karakter op
                     jsr     A2HEX          ; Vertaal het
                     sta     TEMP           ; Bewaar de onderste 4 bits
                     pla                    ; Haal het eerste karakter op
                     jsr     A2HEX          ; Vertaal het
                     asl     A
                     asl     A
                     asl     A
                     asl     A              ; Schuif het resultaat in de
                                            ; bovenste 4 bits van A
                     ora     TEMP           ; Zet de onderste 4 bits goed
                     rts


;
; Subroutine : A2HEX
; Doel       : Vertaal een ASCII- naar een hexadecimaal getal
; Begin      : A = ASCII-karakter
; Einde      : A = binaire waarde van het ASCII-karakter
; Registers  : A, P
;

A2HEX:
                     sec                    ; Trek de ASCII-index eraf
                     sbc     #'0'
                     cmp     #10
                     bcc     A2HEX1         ; Spring als A een decimaal cijfer
                                            ; is
                     sbc     #7             ; Trek de index voor de letters
                                            ; eraf
A2HEX1:
                     rts

; Data
TEMP:                .BLOCK  1

;
;
; Voorbeeld uitvoering
;
;

SC0404:
          ; Vertaal 'C7' naar C7 hexadecimaal

                     lda     #'C'
                     ldy     #'7'
                     jsr     HEX2BN
                     brk                    ; A = C7H
```

```
; Vertaal '2F' naar 2F hexadecimaal

        lda     #'2'
        ldy     #'F'
        jsr     HEX2BN
        brk                     ; A = 2FH

; Vertaal '23' naar 23 hexadecimaal

        lda     #'2'
        ldx     #'3'
        jsr     HEX2BN
        brk                     ; A = 23H

     .END
```

Conversie van een binair getal naar decimaal ASCII - 4E

Converteert een 16-bits getekend binair getal naar een ASCII-string, bestaande uit de lengte van het getal (in bytes), een ASCII-minteken (indien nodig) en de ASCII-cijfers.
Procedure: Het programma pakt de absolute waarde van het getal als dit negatief is en blijft dan door tien delen tot er een quotient van nul uitkomt. Het converteert elk cijfer van het quotient naar ASCII (door ASCII 0 erbij op te tellen) en rijgt de cijfers aan elkaar, voorafgegaan door een ASCII-minteken als het oorspronkelijke getal negatief was.

Gebruikte registers:	Alle
Looptijd:	Ongeveer 7000 cycli
Programmalengte:	174 bytes
Benodigd datageheugen:	Zeven bytes ergens in RAM voor het terugkeeradres (twee bytes beginnende op adres TERADR), het teken van de oorspronkelijke waarde (adres NGVLAG), tijdelijke opslag voor de oorspronkelijke waarde (twee bytes beginnende op adres WAARDE) en tijdelijke opslag voor de waarde mod 10 (twee bytes beginnende op adres MOD10). Eveneens, twee bytes op pagina 0 voor de buffer-pointer (adres BUFPTR, in de listing genomen als $00D0 en $00D1). Dit datageheugen omvat niet de uitvoer-buffer, die zeven bytes lang moet zijn.

Begincondities

Volgorde op stack (van boven af)

Laagstwaardige byte van het terugkeeradres.
Hoogstwaardige byte van het terugkeeradres.
Laagstwaardige byte van het adres van de output-buffer.
Hoogstwaardige byte van het adres van de output-buffer.
Laagstwaardige byte van de te converteren waarde.
Hoogstwaardige byte van de te converteren waarde.

Eindcondities

Volgorde in buffer

Lengte van de string in bytes.
ASCII - (als het oorspronkelijke getal negatief was).
ASCII cijfers (hoogstwaardige cijfer eerst).

Voorbeelden

1. Data: Te converteren waarde = $3EB7_{16}$

   Resultaat (in uitvoer-buffer):

   05 (aantal bytes in buffer)
   31 (ASCII 1)
   36 (ASCII 6)
   30 (ASCII 0)
   35 (ASCII 5)
   35 (ASCII 5)

Ofwel: $3EB7_{16} = 16055_{10}$

2. Data: Te converteren waarde = $FFC8_{16}$

   Resultaat (in uitvoer-buffer):

   03 (aantal bytes in buffer)
   2D (ASCII -)
   35 (ASCII 5)
   36 (ASCII 6)

Ofwel: $FFC8_{16} = -56_{10}$ als het wordt beschouwd als een getekend tweecomplement getal.

```
;
;
;       Titel     : Conversie van een binair getal naar decimaal ASCII
;       Naam      : BN2DEC
;
;
;       Doel      : Vertaal een 16-bits binair nummer met teken naar ASCII-
;                   data
;
;
;       Begin     : Top van de stack
;                     Laagstwaardige byte van het terugkeeradres
;                     Hoogstwaardige byte van het terugkeeradres
;                     Laagstwaardige byte van het adres van de uitvoer-buffer
;                     Hoogstwaardige byte van het adres van de uitvoer-buffer
;                     Laagstwaardige byte van de te vertalen waarde
;                     Hoogstwaardige byte van de te vertalen waarde
;
;       Eind      : De eerste byte in de buffer geeft de lengte aan gevolgd
;                   door de afzonderlijke karakters
;
;       Registers : Alle
;
;       Tijd      : Ongeveer 7.000 cycli
;
;       Grootte   : programma 170  bytes
;                   data        7  bytes plus
```

```
;                                   2  bytes op pagina nul

; Wijzer naar pagina nul

BUFPTR:             .EQU    0D0H            ; Wijzer naar de buffer op pagina 0

; Programma

BN2DEC:
          ; Bewaar de parameters

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1        ; Bewaar het terugkeeradres
                    pla
                    sta     VALUE
                    pla
                    sta     VALUE+1         ; Bewaar de waarde
                    sta     NGFLAG          ; Bewaar de hoogstwaardige byte ook
                                            ; nog voor het teken
                    bpl     GETBP           ; Spring als de waarde positief is
                    lda     #0              ; Maak het positief door ( 0 -
                                            ; VALUE) te berekenen
                    sec
                    sbc     VALUE
                    sta     VALUE
                    lda     #0
                    sbc     VALUE+1
                    sta     VALUE+1

GETBP:
                    pla                     ; Bewaar beginadres van de buffer
                    sta     BUFPTR
                    pla
                    sta     BUFPTR+1

          ; Maak de buffer leeg

                    lda     #0
                    ldy     #0
                    sta     (BUFPTR),Y


          ; Vertaal waarde naar een string


CNVERT:
          ; VALUE := VALUE div 10
          ; MOD10 := VALUE mod 10

                    lda     #0
                    sta     MOD10
                    sta     MOD10+1

                    ldx     #16
                    clc                     ; Reset de overdrachtvlag

DVLOOP:
                    rol     VALUE           ; Schuif de overdrachtvlag in bit 0
                                            ; van de teller
                    rol     VALUE+1         ; hetgeen het quotient wordt
                    rol     MOD10           ; en schuif de noemer mee
```

```
                rol     MOD10+1

        ; A, Y = Teller - Noemer

                sec
                lda     MOD10
                sbc     #10
                tay                     ; Bewaar laagstwaardige byte in Y
                lda     MOD10+1
                sbc     #0              ; Trek de overdrachtvlag eraf
                bcc     DECCNT          ; Spring als teller < noemer
                sty     MOD10           ; anders
                sta     MOD10+1         ;  volgende bit van het quotient is
                                        ;   1, en teller := teller - noemer

DECCNT:
                dex
                bne     DVLOOP
                rol     VALUE           ; Haal de overdrachtvlag van de
                                        ; laatste deling binnen
                rol     VALUE+1

        ; Combineer het volgende karakter

CONCH:
                lda     MOD10
                clc
                adc     #'0'            ; Vertaal 0...9 naar ASCII
                                        ; '0'...'9'
                jsr     CONCAT

        ; Ga door als VALUE <> 0 is

                lda     VALUE
                ora     VALUE+1
                bne     CNVERT          ; Spring als VALUE ongelijk nul is

EXIT:
                lda     NGFLAG
                bpl     POS             ; Spring als de oorspronkelijke
                                        ; waarde positief was
                lda     #'-'            ; anders
                jsr     CONCAT          ; Zet er een min voor

POS:
                lda     RETADR+1
                pha
                lda     RETADR
                pha
                rts                     ; Keer terug


;
; Subroutine : CONCAT
; Doel       : Plak het karakter dat in A staat voor de string die aan-
;              gewezen wordt door BUFPTR
; Begin      : BUFPTR[0] = lengte
; Einde      : Register A ingevoegd vlak na het lengteveld
; Registers  : A, P, Y
;
```

```
CONCAT:
                pha                     ; Bewaar het karakter op de stack

        ; Schuif de karakters in de buffer een plaats op

                ldy     #0
                lda     (BUFPTR),Y
                tay
                beq     EXITMR          ; Spring als de lengte nul is
MVELP:
                lda     (BUFPTR),Y      ; Haal het volgende karakter op
                iny
                sta     (BUFPTR),Y      ; Zet het weer weg
                dey
                dey
                bne     MVELP           ; Totdat alle karakters verschoven
                                        ; zijn

EXITMR:
                pla                     ; Haal het karakter van de stack af
                ldy     #1
                sta     (BUFPTR),Y      ; Zet het op zijn plaats
                ldy     #0
                lda     (BUFPTR),Y      ; Haal de lengte op
                clc
                adc     #1              ; Verhoog de lengte met 1
                sta     (BUFPTR),Y      ; Zet de nieuwe lengte weg
                rts

; Data
RETADR:         .BLOCK  2               ; De bewaarde waarde
NGFLAG:         .BLOCK  1               ; Teken van de beginwaarde
VALUE:          .BLOCK  2               ; Waarde die vertaald wordt
MOD10:          .BLOCK  2               ; Tijdelijke modulo 10 waarde


;
;
; Voorbeeld uitvoering
;
;

SC0405:
        ; Vertaal 0 naar '0'

                lda     BUFADR+1        ; Hoogstwaardige byte van buffer
                pha
                lda     BUFADR          ; Laagstwaardige byte van buffer
                pha
                lda     VALUE1+1        ; Hoogstwaardige byte van de waarde
                pha
                lda     VALUE1          ; Laagstwaardige byte van de waarde
                pha
                jsr     BN2DEC          ; Vertaal
                brk                     ; Buffer zou '0' moeten bevatten

        ; Vertaal 32767  naar '32767'

                lda     BUFADR+1        ; Hoogstwaardige byte van buffer
                pha
                lda     BUFADR          ; Laagstwaardige byte van buffer
                pha
                lda     VALUE2+1        ; Hoogstwaardige byte van de waarde
```

```
                pha
                lda     VALUE2          ; Laagstwaardige byte van de waarde
                pha
                jsr     BN2DEC          ; Vertaal
                brk                     ; Buffer zou '32767' moeten
                                        ; bevatten

        ; Vertaal -32768 naar '-32768'

                lda     BUFADR+1        ; Hoogstwaardige byte van buffer
                pha
                lda     BUFADR          ; Laagstwaardige byte van buffer
                pha
                lda     VALUE3+1        ; Hoogstwaardige byte van de waarde
                pha
                lda     VALUE3          ; Laagstwaardige byte van de waarde
                pha
                jsr     BN2DEC          ; Vertaal
                brk                     ; Buffer zou '-32768' moeten
                                        ; bevatten

VALUE1:         .WORD   0               ; Test waarde 1
VALUE2:         .WORD   32767           ; Test waarde 2
VALUE3:         .WORD   -32768          ; Test waarde 3
BUFADR:         .WORD   BUFFER          ; Adres van BUFFER
BUFFER:         .BLOCK  7               ; 7-bytes buffer

        .END
```

Conversie van decimaal ASCII naar binair - 4F

Cónverteert een ASCII-string, bestaande uit de lengte van het getal (in bytes), een mogelijk ASCII - of + teken en een reeks ASCII-cijfers, naar twee bytes binaire data. Onthoud dat de lengte een gewoon binair en geen ASCII-getal is.
Procedure: Het programma zet een vlag als het eerste ASCII-karakter een minteken was en slaat een plusteken over. Dan converteert het elk volgend cijfer naar decimaal (door ASCII nul af te trekken), vermenigvuldigt de voorgaande cijfers met 10 (gebruik makend van het feit dat 10 = 8 + 2, zodat een vermenigvuldiging met tien gereduceerd kan worden tot schuivingen naar links en optellingen), en telt het nieuwe cijfer bij het produkt op. Uiteindelijk trekt het programma het resultaat van nul af als het oorspronkelijke getal negatief was. Het programma stopt onmiddellijk en zet de overdrachtvlag als het iets anders dan een notatieteken of een decimaal cijfer in de string vindt.

Gebruikte registers: Alle
Looptijd: Ongeveer 670 cycli
Programmalengte: 171 bytes
Benodigd datageheugen: Vier bytes ergens in RAM voor een index, een twee-bytes grote accumulator (beginadres ACCUM) en een vlag die het teken van het getal aangeeft (adres NGVLAG), twee bytes op pagina nul voor een pointer naar de string (adres BUFPTR, in de listing genomen als $00F0_{16}$ en $00F1_{16}$).

Speciale gevallen:

1. Als de string iets anders dan een notatieteken voorin of een decimaal cijfer bevat, stopt de routine met de overdrachtvlag op 1 gezet. Het resultaat in de A- en Y-registers is ongeldig.
2. Als de string slechts een notatieteken (ASCII + of ASCII -) voorin bevat, stopt de routine met de overdrachtvlag op 1 gezet en nul als resultaat.

Begincondities

(A) = Hoogstwaardige byte van het stringadres.
(Y) = Laagstwaardige byte van het stringadres.

Eindcondities

(A) = Hoogstwaardige byte van de binaire waarde.
(Y) = Laagstwaardige byte van de binaire waarde.

Overdrachtvlag = 0 als de string geldig was; De overdrachtvlag is 1 als de string een illegaal karakter bevatte. Onthoud dat de string een getekend tweecomplements 16-bits getal is.

Voorbeelden

1. Data: String bestaat uit

   04 (aantal bytes in string)
   31 (ASCII 1)
   32 (ASCII 2)
   33 (ASCII 3)
   34 (ASCII 4)

   Het getal is dus $+1.234_{10}$

   Resultaat: $(A) = 04_{16}$ (hoogstwaardige byte van binaire data)
   $(Y) = C2_{16}$ (laagstwaardige byte van binaire data)

   Dus, het getal $+1.234_{10} = 04C2_{16}$

2. Data: String bestaat uit

   06 (aantal bytes in string)
   2D (ASCII -)
   33 (ASCII 3)
   32 (ASCII 2)
   37 (ASCII 7)
   35 (ASCII 5)
   30 (ASCII 0)

   Het getal is dus -32.750_{10}

   Resultaat: $(A) = 80_{16}$ (hoogstwaardige byte van binaire data)
   $(Y) = 12_{16}$ (laagstwaardige byte van binaire data)

   Dus, het getal $-32.750_{10} = 8012_{16}$

```
;
;
;
;     Titel     : Decimaal ASCII naar binair vertaler
;     Naam      : DEC2BN
;
;
;     Doel      : Vertaalt ASCII-karakters naar twee bytes binaire data
;
;
;     Begin     : Register A = hoogstwaardige byte van het stringadres
;                 Register Y = laagstwaardige byte van het stringadres
;                 De eerste byte in de string bevat de lengte van de string
;
;     Eind      : Register A = hoogstwaardige byte van de waarde
;                 Register Y = laagstwaardige byte van de waarde
;                 Als er geen fouten zijn is de overdrachtvlag 0
;                 anders is de overdrachtvlag 1
;
;     Registers : Alle
;
;     Tijd      : Ongeveer 670 cycli
```

```
;
;     Grootte    : programma 171  bytes
;                  data        4  bytes plus
;                              2  bytes op pagina 0

; Locatie van pagina 0

BUFPTR:            .EQU    0F0H          ; Wijzer naar string op pagina  0
;
; Programma

DEC2BN:
                   sta     BUFPTR+1
                   sty     BUFPTR        ; Bewaar het adres van de string

          ; Initialiseer

                   ldy     #0
                   lda     (BUFPTR),Y    ; Haal de lengte op
                   tax                   ; Zet deze in Register X
                   lda     #1
                   sta     INDEX         ; INDEX := 1
                   lda     #0
                   sta     ACCUM         ; ACCUM := 0
                   sta     ACCUM+1
                   sta     NFLAG         ; Teken van het nummer is positief

          ; Kijk of de buffer niet leeg is

                   txa
                   bne     INIT1         ; Keer terug met ACCUM = 0 als de
                                         ; buffer leeg is
                   jmp     EREXIT        ; Keer terug met overdrachtvlag
                                         ; gezet
                                         ; als de buffer leeg is

INIT1:
                   ldy     INDEX
                   lda     (BUFPTR),Y    ; Haal het eerste karakter op
                   cmp     #'-'          ; Is het een min-karakter?
                   bne     PLUS          ; Spring als dat niet zo was
                   lda     #0FFH
                   sta     NGFLAG        ; Maak het teken van het nummer
                                         ; negatief
                   inc     INDEX         ; Stap over het minteken heen
                   dex                   ; Maak de lengte een kleiner
                   beq     EREXIT        ; Spring als er alleen een '-' in de
                                         ; buffer zat
PLUS
                   cmp     #'+'
                   bne     CHKDIG        ; begin met omzetting als eerste
                                         ; karakter ongelijk aan +
                   inc     INDEX
                   dex                   ; teller start, +teken niet tellen
                   beq     EREXIT        ; verlaat bij alleen het + teken
CNVERT:
                   ldy     INDEX
                   lda     (BUFPTR),Y    ; Haal een volgend karakter op
CHKDIG:
                   cmp     #'0'
                   bmi     EREXIT        ; Spring als het karakter < '0' is
                                         ;   (geen cijfer)
```

```
                cmp     #'9'+1
                bpl     EREXIT          ; Spring als het karakter > '9' is
                                        ;   (geen cijfer)
                pha                     ; Bewaar het cijfer op de stack

        ; We hebben een legaal cijfer gevonden dus
        ;   ACCUM := ACCUM*10
        ;     = ACCUM * ( 8 + 2 )
        ;     = ( ACCUM * 8 ) + ( ACCUM * 2 )

                asl     ACCUM
                rol     ACCUM+1         ; Maal 2
                lda     ACCUM
                ldy     ACCUM+1         ; Bewaar ACCUM * 2
                asl     ACCUM
                rol     ACCUM+1
                asl     ACCUM
                rol     ACCUM+1         ; Maal 8
                clc
                adc     ACCUM           ; Tel ACCUM * 2 erbij op
                sta     ACCUM
                tya
                adc     ACCUM+1
                sta     ACCUM+1         ; ACCUM := ACCUM * 10

        ; Tel het gevonden cijfer erbij
        ;  ACCUM := ACCUM + cijfer

                pla                     ; Haal het cijfer van de stack
                sec
                sbc     #'0'            ; Vertaal '0'...'9' -> 0...9
                clc
                adc     ACCUM
                sta     ACCUM
                bcc     D2B1            ; Spring als er geen overloop is
                inc     ACCUM+1         ; Neem anders de overloop mee
D2B1:
                inc     INDEX           ; Ga naar het volgende karakter
                                        ; wijzen
                dex
                bne     CNVERT          ; Ga door met het omzetten
                lda     NGFLAG
                bpl     OKEXIT          ; Spring als de waarde positief was
                lda     #0
                sec
                sbc     ACCUM
                sta     ACCUM
                lda     #0
                sbc     ACCUM+1
                sta     ACCUM+1

        ; Haal de binaire waarde op en keer terug

OKEXIT:
                clc
                bcc     EXIT

EREXIT:
                sec

EXIT:
```

```
                lda     ACCUM+1         ; Haal de hoogstwaardige byte van de
                                        ; waarde op
                ldy     ACCUM
                rts


; DATA
INDEX:          .BLOCK  1               ; Index in de string
ACCUM:          .BLOCK  2               ; De vergaarbak voor de waarde
NGFLAG:         .BLOCK  1               ; Teken van het nummer

;
;
; Voorbeeld uitvoering
;
;

SC0406:
        ; Vertaal '1234' naar 04D2 hexadecimaal

                lda     ADRS1+1
                ldy     ADRS1           ; AY = adres van S1
                jsr     DEC2BN
                brk                     ; A = 04 hexadecimaal
                                        ; Y = D2 hexadecimaal

        ; Vertaal '+32767' naar 7FFF hexadecimaal

                lda     ADRS2+1
                ldy     ADRS2           ; AY = adres van S2
                jsr     DEC2BN
                brk                     ; A = 7F hexadecimaal
                                        ; Y = FF hexadecimaal

        ; Vertaal '-32768' naar 8000 hexadecimaal

                lda     ADRS3+1
                ldy     ADRS3           ; AY = adres van S3
                jsr     DEC2BN
                brk                     ; A = 80 hexadecimaal
                                        ; Y = 00 hexadecimaal


S1:             .BYTE   4,'1234'
S2:             .BYTE   6,'+32767'
S3:             .BYTE   6,'-32768'

ADRS1:          .WORD   S1              ; Adres van S1
ADRS2:          .WORD   S2              ; Adres van S2
ADRS3:          .WORD   S3              ; Adres van S3


                .END
```

KLEINE letter omzetten naar een HOOFDLETTER- 4G

Vertaalt een ASCII-kleine letter naar zijn Hoofdletter equivalent.
Procedure: Het programma bepaalt m.b.v. vergelijkingen of de data een ASCII-kleine letter is. Als dit het geval is, trekt het programma 20_{16} van de waarde af, waarmee het equivalent is onstaan. Is het geen kleine letter, dan blijft de data ongewijzigd.

Gebruikte registers:	A, P
Looptijd:	18 cycli als het oorspronkelijke karakter geldig is, anders minder cycli
Programmalengte:	12 bytes
Benodigd datageheugen:	Geen.

Begincondities

Karakter in de accumulator.

Eindcondities

Als het karakter een kleine ASCII-letter is, bevindt de hoofdletter zich in de accumulator. Is het karakter geen kleine ASCII-letter, dan verandert de inhoud van de accumulator niet.

Voorbeelden

1. Data: (A) = 62_{16} (ASCII b)
   Resultaat: (A) = 42_{16} (ASCII B)

2. Data: (A) = 74_{16} (ASCII t)
   Resultaat: (A) = 54_{16} (ASCII T)

```
;
;
;
;     Titel     : Kleine Letter omzetten naar hoofdletters
;     Naam      : LC2UC
;
;
;     Doel      : Het vertalen van een ASCII-karakter naar een hoofd-
;                 letter, als dat nodig is
;
;
;     Begin     : Register A = kleine ASCII-karakter
;
;     Eind      : Register A = hoofd ASCII-letter als A een kleine-
;                 letter was, anders blijft A onveranderd
;
;     Registers : A, P
;
```

```
;
;     Tijd       : 18 cycli als A klein is, anders minder
;
;     Grootte    : programma 12  bytes
;                  data       0  bytes

LC2UC:
                    cmp     #'a'
                    bcc     EXIT          ; Spring als het karakter < 'a' is
                    cmp     #'z'+1
                    bcs     EXIT          ; Spring als het karakter > 'z' is
                    sec
                    sbc     #20H          ; Verander 'a'..'z' in 'A'..'Z'
EXIT:
                    rts

;
;
; Voorbeeld uitvoering
;
;

SC0407:
          ; Vertaal kleine 'e' naar hoofdletter

                    lda     #'e'
                    jsr     LC2UC
                    brk                   ; A = 'E' = 45H

          ;Vertaal kleine 'z' naar hoofdletter

                    lda     #'z'
                    jsr     LC2UC
                    brk                   ; A = 'Z' = 5AH

          ; Vertaal hoofdletter 'A' naar kleine 'A'

                    lda     #'A'
                    jsr     LC2UC
                    brk                   ; A = 'A' = 41H


                    .END
```

ASCII naar EBCDIC conversie - 4H

Converteert een ASCII-karakter naar het EBCDIC-equivalent.
Procedure: Het programma gebruikt een eenvoudige naslagtabel met de data als index en adres EBCDIC als basis. ASCII-karakters die wel geprint kunnen worden, maar geen EBCDIC-equivalent hebben, worden vervangen door een EBCDIC-spatie (40_{16}); ASCII-karakters die niet geprint kunnen worden en geen EBCDIC-equivalent hebben, worden vertaald door een EBCDIC-NUL (00_{16}).

Gebruikte registers:	A, P, Y
Looptijd:	14 cycli
Programmalengte:	Zeven bytes, plus 128 bytes voor de conversie tabel
Benodigd datageheugen:	Geen

Begincondities

ASCII-karakter in de accumulator.

Eindcondities

EBCDIC-equivalent in de accumulator.

Voorbeelden

1. Data: (A) = 35_{16} (ASCII 5)
   Resultaat: (A) = $F5_{16}$ (EBCDIC 5)

2. Data: (A) = 77_{16} (ASCII w)
   Resultaat: (A) = $A6_{16}$ (EBCDIC w)

3. Data: (A) = $2A_{16}$ (ASCII *)
   Resultaat: (A) = $5C_{16}$ (EBCDIC *)

```
;
;
;
;     Titel     : ASCII naar EBCDIC conversie
;     Naam      : ASC2EB
;
;
;     Doel      : Vertaal een ASCII-karakter naar zijn corresponderende
;                 EBCDIC-karakter
;
;
;     Begin     : Register A = ASCII-karakter
;
;     Eind      : Register A = EBCDIC-karakter
;
;     Registers : A, P, Y
```

```
;
;       Tijd        : 14 cycli
;
;       Grootte     : programma 7  bytes
;                     data      128  bytes voor de tabel

ASC2EB:
                      and     #7FH          ; Nu is bit 7 zeker nul
                      tay                   ; Gebruik de ASCII-waarde als index
                                            ; in de tabel
                      lda     EBCDIC,Y      ; Haal EBCDIC-waarde op
                      rts


; ASCII naar EBCDIC tabel
;
; Afdrukbare ASCII-karakters die geen EBCDIC-equivalent hebben worden
; vertaald naar een EBCDIC-spatie (040H), niet afdrukbare ASCII karakters
; zonder equivalent worden vertaald naar een EBCDIC-nul (000H)

EBCDIC:

                      ;       nul  soh  stx  etx  eot  eng  ack  bel      ;  ASCII
                      .BYTE 000H,000H,000H,022H,037H,000H,000H,000H      ; EBCDIC
                      ;       bs   ht   lf   vt   ff   cr   so   si       ;  ASCII
                      .BYTE 000H,02BH,025H,000H,000H,02DH,000H,000H      ; EBCDIC
                      ;       dle  dc1  dc2  dc3  dc4  nak  syn  etb      ;  ASCII
                      .BYTE 000H,000H,000H,000H,037H,000H,000H,000H      ; EBCDIC
                      ;       can  em   sub  esc  fs   gs   rs   vs       ;  ASCII
                      .BYTE 000H,000H,000H,000H,000H,000H,000H,000H      ; EBCDIC
                      ;       space !    "    #    $    %    &    '       ;  ASCII
                      .BYTE 040H,05AH,07EH,040H,05BH,06CH,050H,07CH      ; EBCDIC
                      ;        (    )    *    +    ,    -    .    /       ;  ASCII
                      .BYTE 04DH,05DH,05CH,04EH,06BH,060H,04BH,061H      ; EBCDIC
                      ;        0    1    2    3    4    5    6    7       ;  ASCII
                      .BYTE 0F0H,0F1H,0F2H,0F3H,0F4H,0F5H,0F6H,0F7H      ; EBCDIC
                      ;        8    9    :    ;    <    =    >    ?       ;  ASCII
                      .BYTE 0F8H,0F9H,07AH,05EH,04CH,07DH,06EH,06FH      ; EBCDIC
                      ;        @    A    B    C    D    E    F    G       ;  ASCII
                      .BYTE 07BH,0C1H,0C2H,0C3H,0C4H,0C5H,0C6H,0C7H      ; EBCDIC
                      ;        H    I    J    K    L    M    N    O       ;  ASCII
                      .BYTE 0C8H,0C9H,0D1H,0D2H,0D3H,0D4H,0D5H,0D6H      ; EBCDIC
                      ;        P    Q    R    S    T    U    V    W       ;  ASCII
                      .BYTE 0D7H,0D8H,0D9H,0E2H,0E3H,0E4H,0E5H,0E6H      ; EBCDIC
                      ;        X    Y    Z    [    \    ]    ^    <-      ;  ASCII
                      .BYTE 0E7H,0E8H,0E9H,040H,040H,040H,06AH,040H      ; EBCDIC
                      ;        `    a    b    c    d    e    f    g       ;  ASCII
                      .BYTE 07CH,081H,082H,083H,084H,085H,086H,087H      ; EBCDIC
                      ;        h    i    j    k    l    m    n    o       ;  ASCII
                      .BYTE 088H,089H,091H,092H,093H,094H,095H,096H      ; EBCDIC
                      ;        p    q    r    s    t    u    v    w       ;  ASCII
                      .BYTE 097H,098H,099H,0A2H,0A3H,0A4H,0A5H,0A6H      ; EBCDIC
                      ;        x    y    z    {    |    }    ~   del      ;  ASCII
                      .BYTE 0A7H,0A8H,0A9H,040H,40FH,040H,05FH,007H      ; EBCDIC


;
;
; Voorbeeld uitvoering
;
;
```

```
SC0408:
        ; Vertaal ASCII 'A'

                lda     #'A'
                jsr     ASC2EB
                brk                     ; EBCDIC 'A' = 0C1H

        ; Vertaal ASCII '1'

                lda     #'1'
                jsr     ASC2EB
                brk                     ; EBCDIC '1' = 0F1H

        ; Vertaal ASCII 'a'

                lda     #'a'
                jsr     ASC2EB
                brk                     ; EBCDIC 'a' = 081H

                .END
```

EBCDIC naar ASCII conversie - 4I

Converteert een EBCDIC-karakter naar het ASCII-equivalent.
Procedure: Het programma gebruikt een eenvoudige naslagtabel met de data als index en adres ASCII als basis. EBCDIC-karakters die wel geprint kunnen worden, maar geen ASCII-equivalent hebben, worden vervangen door een ASCII-spatie (20_{16}); EBCDIC-karakters die niet geprint kunnen worden en geen ASCII-equivalent hebben, worden vertaald door een ASCII-NUL (00_{16}).

Gebruikte registers:	A, P, Y
Looptijd:	12 cycli
Programmalengte:	Vijf bytes, plus 256 bytes voor de conversietabel
Benodigd datageheugen:	Geen

Begincondities

EBCDIC-karakter in de accumulator.

Eindcondities

ASCII-equivalent in de accumulator.

Voorbeelden

1. Data: (A) = 85_{16} (EBCDIC e)
   Resultaat: (A) = 65_{16} (ASCII e)

2. Data: (A) = $4E_{16}$ (EBCDIC +)
   Resultaat: (A) = $2B_{16}$ (ASCII +)

```
;
;
;
;     Titel     : EBCDIC naar ASCII conversie
;     Naam      : EB2ASC
;
;
;     Doel      : Het vertalen van een EBCDIC-karakter naar zijn
                  bijbehorende
;                 ASCII-karakter
;
;
;     Begin     : Register A = EBCDIC-karakter
;
;     Eind      : Register A = ASCII-karakter
;
;     Registers : A, P, Y
;
;     Tijd      : 12 cycli
;
;     Grootte   : programma   5  bytes
```

```
;                   data      256  bytes voor de tabel

EB2ASC:
                    tay
                    lda     ASCII,Y       ; Vertaal
                    rts


; EBCDIC naar ASCII tabel
;
; Afdrukbare EBCDIC-karakters waar geen ASCII-equivalenten van zijn worden
; vertaald naar ASCII-spatie (020H), niet afdrukbare ECBDIC-karakters die
; geen equivalent hebben worden vertaald naar een ASCII-nul (000H)

ASCII:
                    ;     nul                         tab       del      ; EBCDIC
                    .BYTE 000H,000H,000H,000H,000H,009H,000H,07FH       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;                             newline               ; EBCDIC
                    .BYTE 000H,000H,000H,000H,000H,00DH,000H,000H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;           etx                 lf                  ; EBCDIC
                    .BYTE 000H,003H,000H,000H,000H,00AH,000H,000H       ;  ASCII
                    ;                     tab       cr                  ; EBCDIC
                    .BYTE 000H,000H,000H,009H,000H,00DH,000H,000H       ;  ASCII
                    ;                                           eot     ; EBCDIC
                    .BYTE 000H,000H,000H,000H,000H,000H,000H,004H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;    space                                          ; EBCDIC
                    .BYTE ' ' ,000H,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;           british $                               ; EBCDIC
                    .BYTE 000H,000H,' ' ,'.' ,'<' ,'(' ,'+' ,'|'        ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE '&' ,000H,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,'!' ,'$' ,'*' ,')' ,';' ,'~'        ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,'^' ,',' ,'%' ,040H,'>' ,'?'        ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,':' ,'@' ,''' ,'=' ,'"' ,000H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,'a' ,'b' ,'c' ,'d' ,'e' ,'f' ,'g'        ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 'h' ,'i' ,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,'j' ,'k' ,'l' ,'m' ,'n' ,'o' ,'p'        ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 'q' ,'r' ,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,'s' ,'t' ,'u' ,'v' ,'w' ,'x'        ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 'y' ,'z' ,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,000H,000H,000H,000H,000H,000H,000H       ;  ASCII
                    ;                                                   ; EBCDIC
                    .BYTE 000H,'A' ,'B' ,'C' ,'D' ,'E' ,'F' ,'G'        ;  ASCII
```

```
                ;                                                   ; EBCDIC
                .BYTE 'H' ,'I' ,000H,000H,000H,000H,000H,000H       ;  ASCII
                ;                                                   ; EBCDIC
                .BYTE 000H,'J' ,'K' ,'L' ,'M' ,'N' ,'O' ,'P'        ;  ASCII
                ;                                                   ; EBCDIC
                .BYTE 'Q' ,'R' ,000H,000H,000H,000H,000H,000H       ;  ASCII
                ;                                                   ; EBCDIC
                .BYTE 000H,000H,'S' ,'T' ,'U' ,'V' ,'W' ,'X'        ;  ASCII
                ;                                                   ; EBCDIC
                .BYTE 'Y' ,'Z' ,000H,000H,000H,000H,000H,000H       ;  ASCII
                ;                                                   ; EBCDIC
                .BYTE '0' ,'1' ,'2' ,'3' ,'4' ,'5' ,'6' ,'7'        ;  ASCII
                ;                                                   ; EBCDIC
                .BYTE '8' ,'9' ,000H,000H,000H,000H,000H,000H       ;  ASCII

;
;
; Voorbeeld uitvoering
;
;

SC0409:
        ; Vertaal EBCDIC 'A'

                lda     #'A'
                jsr     EB2ASC
                brk                     ; ASCII 'A' = 041H

        ; Vertaal EBCDIC '1'

                lda     #'1'
                jsr     EB2ASC
                brk                     ; ASCII '1' = 031H

        ; Vertaal EBCDIC 'a'

                lda     #'a'
                jsr     EB2ASC
                brk                     ; ASCII 'a' = 061H


                .END
```

Geheugenvulling - 5A

Plaatst een gespecificeerde waarde in iedere byte van een geheugengebied met een bekende grootte, beginnende op een gegeven adres.
Procedure: Het programma vult eerst alle hele pagina's met de gespecificeerde waarde om daarna het resterende deel van de overblijvende pagina te vullen. Deze benadering is sneller dan het hele gebied in een lus bewerken, omdat hierbij 8-bits in plaats van 16-bits tellers gebruikt kunnen worden. Het kost wel iets meer geheugen dan een enkele lus met een 16-bits teller. Het programma stopt zonder het geheugen te veranderen, als er een gebiedsgrootte van 0000_{16} is opgegeven.

Gebruikte registers: Alle
Looptijd: Ongeveer 11 cycli per byte plus 93 cycli basistijd
Programmalengte: 68 bytes
Benodigddatageheugen: Vijf bytes ergens in RAM voor de array-grootte (twee bytes beginnende bij adres ARYGR), de waarde (een byte op adres WAARDE) en het terugkeeradres (twee bytes beginnende bij het adres TERADR). Verder twee bytes op pagina 0 voor een array-pointer (in de listing genomen als $00D0_{16}$ en $00D1_{16}$).
Speciale gevallen:
1. Bij een grootte van nul stopt het programma meteen zonder het geheugen te wijzigen.
2. Gebieden vullen waar het programma zelf in staat of die het zelf gebruikt, heeft onvoorspelbare gevolgen. Uiteraard is er voorzichtigheid geboden bij het vullen van gebieden op pagina 0, omdat zowel deze routine als de meeste systeem-programmatuur die pagina gebruiken.

Begincondities

Volgorde op stack (van bovenaf)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Waarde die in het geheugen geplaatst moet worden
Laagstwaardige byte van de grootte van het gebied (in bytes)
Hoogstwaardige byte van de grootte van het gebied (in bytes)
Laagstwaardige byte van het beginadres
Hoogstwaardige byte van het beginadres

Eindcondities

Het gebied vanaf het beginadres ter grootte van het opgegeven aantal bytes wordt met de gespecificeerde waarde gevuld. Het gevulde gebied begint dus bij BASIS en loopt door tot en met BASIS + GROOTTE -1 (BASIS is het beginadres en GROOTTE de grootte van het gebied).

Voorbeelden

1. Data: Waarde = FF_{16}
Geheugengrootte (in byte) = 0380_{16}
Beginadres = $1AE0_{16}$

Resultaat: FF_{16} wordt geplaatst in de geheugenadressen $1AE0_{16}$ tot en met $1E5F_{16}$

2. Data: Waarde = EA_{16} (6502 instructiecode voor NOP)
Geheugengrootte (in bytes) = $1C65_{16}$
Beginadres = $E34C_{16}$

Resultaat: EA_{16} wordt geplaatst in de geheugenadressen $E34C_{16}$ tot en met $FFB0_{16}$

```
;
;
;       Titel      : Geheugenvuller
;       Naam       : MFILL
;
;       Doel       : Vul een geheugenblok met een bepaalde waarde
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Waarde die in het geheugenblok moet komen te staan
;                      Laagstwaardige byte van de lengte van het blok
;                      Hoogstwaardige byte van de lengte van het blok
;                      Laagstwaardige byte van het beginadres
;                      Hoogstwaardige byte van het beginadres
;
;       Eind       : Het geheugenblok gevuld met de waarde
;
;       Registers  : Alle
;
;       Tijd       : Ongeveer 11 cycli per byte plus
;                    93 cycli algemeen
;
;       Grootte    : programma 68  bytes
;                    data        5  bytes plus
;                                2  bytes op pagina 0

; Pointer naar pagina 0

ARYPTR:             .EQU    0D0H          ; Pointer naar het blok op pagina 0

MFILL:
          ; Haal de parameters van de stack

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1      ; Haal het terugkeeradres van de
                                          ; stack
```

```
                pla
                sta     VALUE           ; Haal de waarde van de stack

                pla
                sta     ARYSZ
                pla
                sta     ARYSZ+1         ; Haal de lengte van het blok van
                                        ;de stack
                pla
                sta     ARYPTR
                pla
                sta     ARYPTR+1        ; Haal het beginadres van het blok
                                        ; van de stack
                lda     RETADR+1
                pha
                lda     RETADR
                pha                     ; Zet het terugkeeradres weer terug
                                        ; op de stack

        ; Doe eerst de hele pagina's

                lda     VALUE           ; Haal de vulwaarde op
                ldx     ARYSZ+1         ; X = aantal pagina's die gevuld
                                        ; moeten worden
                beq     PARTPG          ; Spring als de hoogstwaardige byte
                                        ; van de lengte 0 is
                ldy     #0

FULLPG:
                sta     (ARYPTR),Y      ; Vul de huidige geheugen cel
                iny                     ; Ga naar de volgende byte
                bne     FULLPG          ; Ga door tot deze pagina vol is
                inc     ARYPTR+1        ; Ga naar de volgende pagina
                dex
                bne     FULLPG          ; Ga door zolang er nog hele
                                        ; pagina's gedaan moeten worden

        ; Doe nu nog een stukje op de huidige pagina als dat nodig is
        ; In A staat nog steeds de invulwaarde

PARTPG:
                ldx     ARYSZ           ; Haal het aantal bytes op dat nog
                                        ; gedaan moet worden
                beq     EXIT            ; Spring als er geen bytes over
                                        ; zijn
                ldy     #0
PARTLP:
                sta     (ARYPTR),Y      ; Zet de waarde op zijn plaats
                iny                     ; Verhoog de index
                dex                     ; Verlaag de lengte
                bne     PARTLP          ; Spring zolang er nog adressen
                                        ; gedaan moeten worden
EXIT:
                rts

; DATA
ARYSZ:          .BLOCK  2               ; De lengte van het blok
VALUE:          .BLOCK  1               ; Waarde die in het blok moet komen
                                        ; te staan
RETADR:         .BLOCK  2               ; Tijdelijke opslag van het
                                        ; terugkeeradres
```

```
; Voorbeeld uitvoering

SC0501:
        ; Vul een kleine buffer met 00 hexadecimaal

                lda     BF1ADR+1
                pha
                lda     BF1ADR
                pha                     ; Beginadres van de buffer
                lda     BF1SZ+1
                pha
                lda     BF1SZ
                pha                     ; Lengte van de buffer
                lda     #0
                pha                     ; De invulwaarde
                jsr     MFILL           ; Vul de buffer
                brk

        ; Vul een grote buffer met EA hexadecimaal

                lda     BF2ADR+1
                pha
                lda     BF2ADR
                pha                     ; Beginadres van de buffer
                lda     BF2SZ+1
                pha
                lda     BF2SZ
                pha                     ; Lengte van de buffer
                lda     #0EAH
                pha                     ; De invulwaarde
                jsr     MFILL           ; Vul de buffer
                brk
                jmp SC0501

SIZE1:          .EQU    47h
SIZE2:          .EQU    6000H

BF1ADR:         .WORD   BF1
BF2ADR:         .WORD   BF2

BF1SZ:          .WORD   SIZE1
BF2SZ:          .WORD   SIZE2

BF1:            .BLOCK  SIZE1
BF2:            .BLOCK  SIZE2

                .END
```

Blokverplaatsing - 5B

Verplaatst een blok data van een bron- naar een bestemmingsgebied.
Procedure: Het programma bepaalt of het beginadres van het bestemmingsgebied binnen het brongebied ligt. Als dit het geval is, zou een gedeelte van het brongebied overschreven kunnen worden als het programma vanaf het startadres naar boven gaat. Om dit probleem te voorkomen, werkt het programma vanaf het hoogste adres naar beneden (ook wel *verplaatsing naar rechts* genoemd). Als het beginadres van het bestemmingsgebied niet binnen het brongebied valt, verplaatst het programma de data eenvoudigweg vanaf het laagste adres (ook wel *verplaatsing naar links* genoemd). In beide gevallen verplaatst het programma de data door hele pagina's afzonderlijk van een gedeeltelijke pagina te bewerken. Deze methode maakt het mogelijk 8-bits in plaats van 16-bits tellers te gebruiken, waardoor de looptijd van het programma wordt verkort (hoewel het geheugengebruik wordt vergroot). Bij een gebiedsgrootte van 0000_{16} (aantal te verplaatsen bytes) stopt het programma zonder het geheugen te veranderen.
Belangrijke opmerking: De gebruiker moet erop letten of de tijdelijke opslaglokaties soms binnen het bron- of bestemmingsgebied vallen. Het programma is voorzien van automatische omwikkeling (wrap-around), maar het resultaat van een verplaatsing van de tijdelijke opslaglokaties is onvoorspelbaar.

Gebruikte registers: Alle

Looptijd: 128 cycli vast plus:

1. Als de data vanaf het laagste adres verplaatst kunnen worden (d.i. naar links): 20 + 4110 * (hoogstwaardige byte van het aantal te verplaatsen bytes) + 18 * (laagstwaardige byte van het aantal te verplaatsen bytes).
2. Als de data vanaf het hoogste adres (naar rechts) verplaatst moetten worden omdat er van overlapping sprake is: 42 + 4622 * (hoogstwaardige byte van het aantal te verplaatsen bytes) + 18 * (laagstwaardige byte van het aantal te verplaatsen bytes).

Programmalengte: 157 bytes

Benodigd datageheugen: Twee bytes ergens in RAM voor de lengte van de verplaatsing (beginnende bij VPLLEN), vier bytes op pagina 0 voor bron- en bestemmingspointers (beginnende bij de adressen VPLBRON en VPLBEST, in de listing, de adressen $00D0_{16}$ en $00D1_{16}$ voor - bron pointer - en de adressen $00D2_{16}$ en $00D3_{16}$ voor - bestemmingspointer).

Speciale gevallen:

1. Het programma stopt zonder het geheugen te veranderen als de grootte (aantal te verplaatsen bytes) gelijk is aan nul.
2. Het verplaatsen van data uit geheugengebieden die door het programma zelf in beslag worden genowen of worden gebruikt, geeft onvoorspelbare resultaten. Uiteraard is voorzichtigheid geboden bij het verplaatsen van data vanaf of naar pagina 0, omdat zowel deze routine als de meeste systeemprogrammatuur deze pagina gebruiken. Deze routine is voorzien van automatische omwikkeling (wrap-around), maar de gebruiker moet desalniettemin oppassen bij verplaatsingen waar pagina 0 toe behoort.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van het aantal te verplaatsen bytes
Hoogstwaardige byte van het aantal te verplaatsen bytes
Laagstwaardige byte van het laagste adres van bestemmingsgebied
Hoogstwaardige byte van het laagste adres van bestemmingsgebied
Laagstwaardige byte van het laagste adres van brongebied
Hoogstwaardige byte van het laagste adres van brongebied

Eindcondities

Het geheugenblok wordt verplaatst van het brongebied naar het bestemmingsgebied. Als het aantal te verplaatsen bytes gelijk is aan NBYTES, het laagste adres in het bestemmingsgebied BEST is, en het laagste adres in het brongebied BRON is, dan wordt het gebied vanaf BRON tot en met BRON + NBYTES - 1 verplaatst naar de adressen BEST tot en met BEST + NBYTES - 1

Voorbeelden

1. Data: Aantal te verplaatsen bytes = 0200_{16}
Laagste adres in bestemmingsgebied = $05D1_{16}$
Laagste adres in brongebied = $035E_{16}$

Resultaat: De inhoud van geheugenlocaties $035E_{16}$ tot en met $055D_{16}$ wordt verplaatst naar $05D1_{16}$ tot en met $07D0_{16}$.

2. Data: Aantal te verplaatsen bytes = $1B7A_{16}$
Laagste adres in geheugengebied = $C946_{16}$
Laagste adres in brongebied = $C300_{16}$

Resultaat: De inhoud van geheugenlocaties $C300_{16}$ tot en met $DE79_{16}$ wordt verplaatst naar $C946_{16}$ tot en met $E4BF_{16}$.

Let op dat het tweede voorbeeld complexer is dan het eerste, omdat het bron- en bestemmingsgebied elkaar overlappen. Als het programma bij voorbeeld de data van beneden naar boven zou verplaatsen, wordt de inhoud van $C300_{16}$ in $C946_{16}$ gezet. Dit zou de oude inhoud van $C946_{16}$ vernietigen, terwijl deze later in de verplaatsing nog nodig is. De oplossing hiervoor is de data vanaf het hoogste adres te verplaatsen indien het bestemmingsgebied boven het brongebied ligt maar niet overlapt.

```
;
;       Titel       : Blokken verplaatsen
;       Naam        : BLKMOV
;
```

```
;
;     Doel      : kopieeren van data van het ene deel van het geheugen naar
;                 een ander deel
;
;     Begin     : Top van de stack
;                   Laagstwaardige byte van het terugkeeradres
;                   Hoogstwaardige byte van het terugkeeradres
;                   Laagstwaardige byte van de lengte van het blok
;                   Hoogstwaardige byte van de lengte van het blok
;                   Laagstwaardige byte van hetadres van het origineel
;                   Hoogstwaardige byte van het adres van het origineel
;                   Laagstwaardige byte van het adres van het kopieblok
;                   Hoogstwaardige byte van het adres van het kopieblok
;
;     Eind      : De data zijn verplaatst
;
;     Registers : Alle
;
;     Tijd      : 102 cycli zonder het verplaatsen plus
;
;                 verplaatsing naar links kost
;                     20 +
;                     (hoogstwaardige byte van de lengte * 4110) +
;                     (laagstwaardige byte van de lengte *18)
;
;                 verplaatsing naar rechts kost
;                     42 +
;                     (hoogstwaardige byte van de lengte * 4622) +
;                     (laagstwaardige byte van de lengte *18)
;
;     Grootte   : programma 146  bytes
;                 data        2  bytes plus
;                             4  bytes op pagina 0

; Wijzers op pagina 0

MVSRCE:           .EQU     0D0H        ; Adres van het origineel
MVDEST:           .EQU     0D2H        ; Adres van de kopie

BLKMOV:
          ; Haal het terugkeeradres van de stack

                  pla
                  tay                  ; De laagstwaardige byte
                  pla
                  tax                  ; De hoogstwaardige byte

          ; Haal het aantal bytes van de stack af

                  pla
                  sta     MVELEN       ; De laagstwaardige byte
                  pla
                  sta     MVELEN+1     ; De hoogstwaardige byte

          ; Haal het adres van het kopieblok van de stack af

                  pla
                  sta     MVDEST       ; De laagstwaardige byte
                  pla
                  sta     MVDEST+1     ; De hoogstwaardige byte

          ; Haal het beginpunt van de stack af
```

```
                pla
                sta     MVSRCE          ; De laagstwaardige byte
                pla
                sta     MVSCRE+1        ; De hoogstwaardige byte

        ; Zet het terugkeeradres weer terug op de stack

                txa
                pha                     ; De hoogstwaardige byte
                tya
                pha                     ; De laagstwaardige byte
;
; Bepaal of het eindpunt rechts van het beginpunt, maar toch binnen het te
; kopieeren gebied ligt. N.B.: het overlappen kan ook binnen mod 64K zijn
; Het overlappen kan worden berekend door het eindpunt minus het beginpunt
; (modulus 64K) te vergelijken met het aantal bytes dat gekopieerd moet
; worden. Is de berekende waarde kleiner dan dat aantal bytes, dan wordt
; een deel van het beginblok overschreven voor het gekopieerd kan worden

                lda     MVDEST          ; Bereken eindpunt - beginpunt
                sec
                sbc     MVSRCE
                tax
                lda     MVDEST+1
                sbc     MVSRCE+1        ; Modulus 64K gebeurt automatisch
                tay
                txa
                cmp     MVELEN
                tya
                sbc     MVELEN+1
                bcs     DOLEFT          ; Spring als er geen problemen zijn
                                        ; met het overlappen

        ; De twee gebieden in het geheugen overlappen elkaar
        ; Kopieer dus van achter naar voren

                jsr     MVRHT
                jmp     EXIT

        ; Geen problemen, kopieer dus het blok te beginnen met het
        ; kleinste adres in het te kopieeren gebied

DOLEFT:
                jsr     MVELFT

EXIT:
                rts

; Subroutine : MVELFT
; Doel       : Het verplaatsen van de data van het beginpunt naar het
;              eindpunt, te beginnen bij het kleinste adres
; Begin      : MVSRCE = kleinste adres in het origineel (2 bytes)
;              MVDEST = kleinste adres in de kopie (2 bytes)
;              MVELEN = het aantal bytes dat verplaatst moet worden
;              (2 bytes)
; Einde      : De data zijn gekopieerd naar het kopieblok

MVELFT:
                ldy     #0              ; Zet de index op 0
                ldx     MVELEN+1        ; In X staat nu het aantal pagina's
                beq     MLPART          ; Als X = 0 dan kan er nog een
```

```
                                        ; deelpagina moeten worden gedaan
MLPAGE:
                lda     (MVSRCE),Y
                sta     (MVDEST),Y      ; Verplaats een byte
                iny                     ; Ga naar de volgende byte
                bne     MLPAGE          ; Ga door tot er 256 bytes
                                        ;verplaatst
                                        ; zijn
                inc     MVSRCE+1        ; Ga door naar de volgende pagina
                inc     MVDEST+1        ; Ook in het kopieblok
                dex                     ; Verlaag de paginateller
                bne     MLPAGE          ; Ga door tot alle hele pagina's
                                        ; verplaatst zijn

MLPART:
                ldx     MVELEN          ; Haal de lengte van de laatste
                                        ; pagina op
                beq     MLEXIT          ; Spring als deze 0 is

MLLAST:
                lda     (MVSRCE),Y
                sta     (MVDEST),Y      ; Verplaats een byte
                iny                     ; Ga naar de volgende byte
                dex                     ; Verlaag de teller
                bne     MLLAST          ; Ga door tot de laatste byte
                                        ; verplaatst is
MLEXIT:
                rts

; Subroutine : MVERHT
; Doel       : Verplaats het blok te beginnen met het grootste adres in het
;              blok
; Begin      : MVSRCE = kleinste adres in het origineel (2 bytes)
;              MVDEST = kleinste adres in de kopie (2 bytes)
;              MVELEN = het aantal bytes dat verplaatst moet worden
; Einde      : De data zijn gekopieerd naar het kopieblok

MVERHT:
        ; Bereken eerst de nieuwe beginpunten

                lda     MVELEN+1
                clc
                adc     MVSRCE+1
                sta     MVSRCE+1        ; Wijs nu naar de laatste byte van
                                        ; het origineel
                lda     MVELEN+1
                clc
                adc     MVDEST+1
                sta     MVDEST+1        ; Wijs nu naar de laatste byte van
                                        ; de kopie

        ; Verplaats nu eerst de deelpagina

                ldy     MVELEN          ; Haal de lengte van de deel-pagina
                                        ; op
                beq     MRPAGE          ; Spring als de lengte 0 is

MR0:
                dey                     ; Ga naar de volgende byte
                lda     (MVSRCE),Y
                sta     (MVDEST),Y      ; Verplaats een byte
```

```
                bne     MR0             ; Ga door zolang er nog bytes op
                                        ; deze pagina verplaatst moeten
                                        ; worden
MRPAGE:
                ldx     MVELEN+1        ; Het aantal hele pagina's
                beq     MREXIT          ; Spring als er niets meer gedaan
                                        ; behoeft te worden
MR1:
                dec     MVSRCE+1        ; Ga door met de vorige pagina
                dec     MVDEST+1        ;  idem
MR2:
                dey                     ; Ga door met de volgende byte
                lda     (MVSRCE),Y
                sta     (MVDEST),Y      ; Verplaats een byte
                cpy     #0
                bne     MR2             ; Ga door zolang er nog bytes op
                                        ; deze pagina moeten komen te staan
                dex                     ; Verlaag de paginateller
                bne     MR1             ; Ga door zolang er nog hele
                                        ; pagina's zijn
MREXIT:
                rts

; Data
MVELEN:         .BLOCK  2               ; De lengte van het blok

; Voorbeeld uitvoering

SC0502:
                lda     SRCE+1
                pha                     ; Hoogstwaardige byte van het
                                        ; origineel
                lda     SRCE
                pha                     ; Laagstwaardige byte van het
                                        ; origineel
                lda     DEST+1
                pha                     ; Hoogstwaardige byte van de kopie
                lda     DEST
                pha                     ; Laagstwaardige byte van de kopie
                lda     LEN+1
                pha                     ; Hoogstwaardige byte van de lengte
                lda     LEN
                pha                     ; Laagstwaardige byte van de lengte
                jsr     BLKMOV          ; Verplaats de data van 800
                                        ; hexadecimaal t/m 97F hexadecimaal
                                        ; naar 900 hexadecimaal t/m A7F
                                        ; hexadecimaal
                jmp     SC0502

; Test-data. Deze moet u zelf veranderen om andere waarden te testen
SRCE:           .WORD 0800H             ; Beginadres van het origineel
DEST:           .WORD 0900H             ; Beginadres van de kopie
LEN:            .WORD 0180H             ; Het aantal bytes dat verplaatst
                                        ; moet worden

                .END
```

Eendimensionale byte-array-indexering - 5C

Berekent het adres van een element van een array dat 1 byte lang is, waarbij het beginadres en het subscript (index) van het element is gegeven.
Procedure: Het programma telt eenvoudigweg het beginadres bij het subscript op. De som is het adres van het element.

Gebruikte registers:	Alle
Looptijd:	74 cycli
Programmalengte:	37 bytes
Benodigd datageheugen:	Vier bytes ergens in RAM om het terugkeeradres (twee bytes beginnende bij TERADR) en het subscript (twee bytes beginnende bij SUBSCR) te bewaren.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van het subscript
Hoogstwaardige byte van het subscript
Laagstwaardige byte van het beginadres van het array
Hoogstwaardige byte van het beginadres van het array

Eindcondities

(A) = Hoogstwaardige byte van het adres van het element
(Y) = Laagstwaardige byte van het adres van het element

Voorbeelden

1. Data: Beginadres = $0E00_{16}$
Subscript = $012C_{16}$

Resultaat: Adres van element = $0E00_{16} + 012C_{16} = 0F2C_{16}$

2. Data: Beginadres = $C4E1_{16}$
Subscript = $02E4_{16}$

Resultaat: Adres van element = $C4E1_{16} + 02E4_{16} = C7C5_{16}$

```
;
;       Titel      : Eendimensionale byte-array indexering
;       Naam       : D1BYTE

;       Doel       : Gegeven het beginadres van het byte-array 'A' en en de
;                    index 'I', bereken het adres van A[I]

;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Laagstwaardige byte van de index
;                      Hoogstwaardige byte van de index
;                      Laagstwaardige byte van het beginadres van 'A'
;                      Hoogstwaardige byte van het beginadres van 'A'
;
;       Eind       : Register A = hoogstwaardige byte van het adres
;                    Register y = laagstwaardige byte van het adres
;
;       Registers  : Alle
;
;       Tijd       : 74 cycli
;
;       Grootte    : programma 37  bytes
;                    data        4  bytes

D1BYTE:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal de index van de stack af

                    pla
                    sta     SS
                    pla
                    sta     SS+1

          ; Tel het beginadres van het array bij de index op

                    pla
                    clc
                    adc     SS
                    tay                     ; Register Y = laagstwaardige byte
                    pla
                    adc     SS+1
                    tax                     ; Register X = hoogstwaardige byte

          ; Zet het terugkeeradres weer op de stack

                    lda     RETADR+1
                    pha
                    lda     RETADR
                    pha                     ; Het terugkeeradres staat nu weer
                                            ; op de stack
                    txa                     ; Zet waarde van X weer in A
                    rst                     ; Keer terug

; Data
RETADR:             .BLOCK  2               ; Tijdelijke opslag van het
```

```
                                        ; terugkeeradres
SS:             .BLOCK  2               ; Index in het array

; Voorbeeld aanroep

SC0503:
        ; Zet het adres van het array op de stack

                lda     ARYADR+1        ; Hoogstwaardige byte
                pha
                lda     ARYADR          ; Laagstwaardige byte
                pha

        ; Zet de index op de stack

                lda     SUBSCR+1        ; Hoogstwaardige byte
                pha
                lda     SUBSCR          ; Laagstwaardige byte
                pha
                jsr    D1BYTE           ; Bereken het adres
                brk                     ; AY = ARY + 2
                                        ;    = adres van A ( 2 )
                                        ;      waarin 3 staat
                jmp     SC0503

; Test-data. Verander de index zelf naar andere waarden
SUBSCR:         .WORD   2               ; Index in het array
ARYADR:         .WORD   ARY             ; Adres van het array

; Het array zelf (8 waarden)
ARY:            .BYTE   1,2,3,4,5,6,7,8

                .END
```

Eendimensionale woordarray-indexering - 5D

Berekent het beginadres van een element ter grootte van een woord (16-bits), gegeven het beginadres van het array en de index (subscript) van het element. Het element bevindt zich op het beginadres en het adres 1 hoger daarvan; de elementen kunnen of met de laagstwaardige of met de hoogstwaardige byte op het beginadres georganiseerd zijn.
Procedure: Het programma vermenigvuldigt het subscript met twee (d.m.v. een logische schuiving naar links) voordat dit bij het beginadres wordt opgeteld. De som (BASIS + 2 * SUBSCRIPT) is dan het beginadres van het element.

Gebruikte registers:	Alle
Looptijd:	78 cycli
Programmalengte:	39 bytes
Benodigd datageheugen:	Vier bytes ergens in RAM voor het terugkeeradres (twee bytes beginnende bij TERADR) en het subscript (twee bytes beginnende bij adres SUBSCR).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van het subscript
Hoogstwaardige byte van het subscript
Laagstwaardige byte van het beginadres van het array
Hoogstwaardige byte van het beginadres van het array

Eindcondities

(A) = Hoogstwaardige byte van het beginadres van het element
(Y) = Laagstwaardige byte van het beginadres van het element

Voorbeelden

1. Data: Beginadres = $A148_{16}$
Subscript = $01A9_{16}$

Resultaat: Adres van de eerste byte van het element = $A148_{16} + 2 \times 01A9_{16} = A148_{16} + 0342_{16} = A49A_{16}$. Dus het woord-grote element staat in de adressen $A49A_{16}$ en $A49B_{16}$

2. Data: Beginadres = $C4E0_{16}$
Subscript = $015B_{16}$

Resultaat: Adres van de eerste byte van het element = $C4E0_{16} + 2 \times 015B_{16} = C4E0_{16} + 02B6_{16} = C796_{16}$. Dus het woord-grote element staat in de adressen $C796_{16}$ en $C797_{16}$.

```
;
;
;
;     Titel     : Eendimensionale woord-array indexering
;     Naam      : D1WORD
;
;
;     Doel      : Gegeven het beginadres van het woord-array 'A'
;                 en de index 'I', bereken het adres van A[I]
;
;
;     Begin     : Top van de stack
;                   Laagstwaardige byte van het terugkeeradres
;                   Hoogstwaardige byte van het terugkeeradres
;                   Laagstwaardige byte van de index
;                   Hoogstwaardige byte van de index
;                   Laagstwaardige byte van het beginadres van 'A'
;                   Hoogstwaardige byte van het beginadres van 'A'
;
;
;     Eind      : Register A = Hoogstwaardige byte van het adres
;                 Register y = Laagstwaardige byte van het adres
;
;     Registers : Alle
;
;     Tijd      : 78 cycli
;
;     Grootte   : programma 39  bytes
;                 data       4  bytes

D1WORD:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal de index van de stack af en vermenigvuldig met 2

                    pla
                    asl     A
                    sta     SS
                    pla
                    rol     A
                    sta     SS+1

          ; Tel het beginadres van het array bij de index op

                    pla
                    clc
```

```
                adc     SS
                tay                     ; Register Y = laagstwaardige byte
                pla
                adc     SS+1
                tax                     ; Register X = hoogstwaardige byte

        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha                     ; Het terugkeeradres staat nu
                                        ;weer op de stack
                txa                     ; Zet waarde van X weer in A
                rst                     ; Keer terug


; Data
RETADR:         .BLOCK  2               ; Tijdelijke opslag van het
                                        ; terugkeeradres
SS:             .BLOCK  2               ; Index in het array

;
;
; Voorbeeld uitvoering
;
;

SC0504:
        ; Zet het adres van het array op de stack

                lda     ARYADR+1        ; Hoogstwaardige byte
                pha
                lda     ARYADR          ; Laagstwaardige byte
                pha

        ; Zet de index op de stack

                lda     SUBSCR+1        ; Hoogstwaardige byte
                pha
                lda     SUBSCR          ; Laagstwaardige byte
                pha
                jsr    D1WORD           ; Bereken het adres
                brk                     ; AY = beginadres van ARY(3)
                                        ;    = ARY +(3 * 2)
                                        ;    = ARY + (6)
                                        ;    = ARY (3) waarin het getal 240
                                        ;  hexadecimaal staat
                jmp     SC0504


;
; Test-data. Verander de index zelf naar andere waarden
SUBSCR:         .WORD   3               ; Index in het array
ARYADR:         .WORD   ARY             ; Adres van het array

; Het array zelf (8 waarden)
ARY:            .WORD 0180H,01C0H,0200H,0240H,0280H,02C0H,03E7H,0A34H


                .END
```

Tweedimensionale byte-array-indexering - 5E

Berekent het adres van een element van een tweedimensionaal, byte-groot array, gegeven het beginadres van het array, de twee subscripts van het element en de lengte van een rij (ofte wel het aantal kolommen). Het array wordt verondersteld per rij te zijn opgeslagen en beide subscripten worden verondersteld bij nul te beginnen.
Procedure: Het programma vermenigvuldigt de rij-lengte (aantal kolommen in een rij) met het subscript van de rij (aangezien de elementen per rij opgeslagen zijn) en telt het produkt op bij het subscript van de kolom. Vervolgens telt het de som hiervan op bij het beginadres. De vermenigvuldiging in dit programma gebeurt met behulp van een standaard schuif-en-tel-op algoritme.

Gebruikte registers:	Alle
Looptijd:	Ongeveer 1500 cycli, grotendeels afhankelijk van de tijd die voor het vermenigvuldigen nodig is
Programmalengte:	119 bytes
Benodigd datageheugen:	Tien bytes ergens in het geheugen voor het terugkeeradres (twee bytes beginnende bij adres TERADR), het rij-subscript (twee bytes beginnende bij adres SS1), de lengte van de rijen (twee bytes beginnende bij adres SS1SZ), het kolom-subscript (twee bytes beginnende bij adres SS2) en het produk t van de rij-lengte maal het rij-subscript (twee bytes beginnende bij adres PROD).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van het kolom-subscript
Hoogstwaardige byte van het kolom-subscript
Laagstwaardige byte van de lengte van een rij
Hoogstwaardige byte van de lengte van een rij
Laagstwaardige byte van het rij-subscript
Hoogstwaardige byte van het rij-subscript
Laagstwaardige byte van het beginadres van het array
Hoogstwaardige byte van het beginadres van het array

Eindcondities

(A) = Hoogstwaardige byte van het adres van het element
(Y) = Laagstwaardige byte van het adres van het element

Voorbeelden

1. Data: Beginadres = $3C00_{16}$
Kolom subscript = 0004_{16}
Rij-lengte (aantal kolommen) = 0018_{16}
Rij-subscript = 0003_{16}

Resultaat: Adres van het element = $3C00_{16} + 0003_{16} \times 0018_{16} + 0004_{16} = 3C00_{16} + 0048_{16} + 0004_{16} = 3C4C_{16}$.

Zodoende is $3C4C_{16}$ het adres van ARRAY (3,4).

2. Data: Beginadres = $6A4A_{16}$
Kolom-subscript = 0035_{16}.
Rij-lengte (aantal kolommen) = 0050_{16}
Rij-subscript = 0002_{16}

Resultaat: Adres van het element = $6A4A_{16} + 0002_{16} \times 0050_{16} + 0035_{16} = 6A4A_{16} + 00A0_{16} + 0035_{16} = 6B1F_{16}$.

Zodoende is $6B1F_{16}$ het adres van ARRAY (2,35).

De algemeen geldende formule is:

ADRES VAN ELEMENT = BEGINADRES VAN ARRAY + RIJ-SUBSCRIPT x RIJ-LENGTE + KOLOM-SUBSCRIPT

Merk op dat we het steeds hebben over de *lengte* van het rij-subscript; met *lengte* bedoelen wij het aantal opeenvolgende geheugenadressen waarvoor het subscript dezelfde waarde heeft. Dit is ook het aantal bytes vanaf het beginadres van een element tot het beginadres van een element met hetzelfde kolom-subscript, maar met een rij-subscript dat 1 groter is.

```
;
;
;
;     Titel      : Twee dimensionale byte-array indexering
;     Naam       : D2BYTE
;
;
;     Doel       : Gegeven het beginadres van het byte-array 'A', de indices
;                  'I' 'J' en de lengte van de eerste index in bytes,
;                  bereken het adres van A[I,J]. Het array wordt
;                  verondersteld te zijn opgeslagen in volgorde van rijen
;                  (A[0,0], A[0,1],....,A[K,L]), en beide dimensies beginnen
;                  bij 0 zoals in de volgende Pascal-declaratie:
;
;                    A : ARRAY [ 0..2, 0..7 ] of BYTE;
;
;     Begin      : Top van de stack
;                    Laagstwaardige byte van het terugkeeradres
;                    Hoogstwaardige byte van het terugkeeradres
;                    Laagstwaardige byte van de tweede index
```

```
;                   Hoogstwaardige byte van de tweede index
;                   Laagstwaardige byte van de lengte van de eerste index
;                   Hoogstwaardige byte van de lengte van de eerste index
;                   Laagstwaardige byte van de eerste index
;                   Hoogstwaardige byte van de eerste index
;                   Laagstwaardige byte van het beginadres van 'A'
;                   Hoogstwaardige byte van het beginadres van 'A'
;
;
;     Eind      : Register A = hoogstwaardige byte van het adres
;                 Register y = laagstwaardige byte van het adres
;
;     Registers : Alle
;
;     Tijd      : Ongeveer 1500 cycli
;
;     Grootte   : programma 119  bytes
;                 data       10  bytes

D2BYTE:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal de tweede index van de stack af

                    pla
                    sta     SS2
                    pla
                    sta     SS+1

          ; Haal de lengte van de eerste index van de stack

                    pla
                    sta     SS1SZ
                    pla
                    sta     SS1SZ+1

          ; Haal de eerste index van de stack

                    pla
                    sta     SS1
                    pla
                    sta     SS1+1

          ; Vermenigvuldig de eerste index met de lengte van de eerste
          ; index met behulp van het optel en schuif algoritme
          ; Plaats het resultaat in SS1

                    lda     #0          ; Begin met 0 te vermenigvuldigen
                    sta     PROD
                    sta     PROD+1
                    ldx     #17         ; Er moet 17 keer geshift worden
                    clc
MULLP:
                    ror     PROD+1      ; Shift het deelresultaat
                    ror     PROD
                    ror     SS1+1       ; Shift de multiplicator
                    ror     SS1
                    bcc     DECCNT
```

```
                clc                     ; Tel de multiplicator by het
                                        ; deelresultaat op als het volgende
                                        ; bit van de multiplicator 1 is
                lda     SS1SZ
                adc     PROD
                sta     PROD
                lda     SS1SZ+1
                adc     PROD+1
                sta     PROD+1
DECCNT:
                dex
                bne     MULLLP

        ; Tel de tweede index erbij op

                lda     SS1
                clc
                adc     SS2
                sta     SS1
                lda     SS1+1
                adc     SS2+1
                sta     SS1+1

        ; Tel het beginadres van het array bij de index op

                pla
                clc
                adc     SS1
                tay                     ; Register Y = laagstwaardige byte
                pla
                adc     SS1+1
                tax                     ; Register X = hoogstwaardige byte

        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha                     ; Het terugkeeradres staat nu weer
                                        ; op de stack
                txa                     ; Zet waarde van X weer in A
                rst                     ; Keer terug


; Data
RETADR:         .BLOCK  2               ; Tijdelijke opslag van het
                                        ; terugkeeradres
SS1:            .BLOCK  2               ; Eerste index in het array
SS1SZ:          .BLOCK  2               ; Lengte van de eerste index
SS2:            .BLOCK  2               ; Tweede index in het array
PROD:           .BLOCK  2               ; Tijdelijke opslag voor het
                                        ; vermenigvuldigen

;
;
; Voorbeeld uitvoering
;
;

SC0505:
        ; Zet het adres van het array op de stack

                lda     ARYADR+1        ; Hoogstwaardige byte
```

```
                pha
                lda     ARYADR          ; Laagstwaardige byte
                pha

        ; Zet de eerste index op de stack

                lda     SUBS1+1         ; Hoogstwaardige byte
                pha
                lda     SUBS1           ; Laagstwaardige byte
                pha

        ; Zet de lengte van de eerste index op de stack

                lda     SSUBS1+1
                pha
                lda     SSUBS1
                pha

        ; Zet de tweede index op de stack

                lda     SUBS2+1
                pha
                lda     SUBS2
                pha

                jsr     D2BYTE          ; Bereken het adres
                brk                     ; AY = beginadres van A[2,4]
                                        ;    = A + (2 * 8) + 4
                                        ;    = A + 20, waarin het getal 21
                                        ;      staat
                jmp     SC0505


;
; Data
SUBS1:          .WORD   2               ; Eerste index in het array
SSUBS1:         .WORD   8               ; Lengte van de eerste index
SUBS2:          .WORD   4               ; Tweede index in het array
ARYADR:         .WORD   ARY             ; Adres van het array

; Het array zelf (3 rijen, 8 kolommen)
ARY:            .WORD 1 ,2 ,3 ,4 ,5 ,6 ,7 ,8
                .WORD 9 ,10,11,12,13,14,15,16
                .WORD 17,18,19,20,21,22,23,24


                .END
```

Tweedimensionale woord-array-indexering - 5F

Berekent het beginadres van een element van een tweedimensionaal array ter grootte van een woord (16 bits), gegeven het beginadres van het array, de twee subscripts van het element en de lengte van een rij in bytes. Verondersteld wordt dat het array in rijen is opgeslagen en beide subscripts bij nul beginnen.
Procedure: Het programma vermenigvuldigt de rij-lengte (in bytes) met het rij-subscript (aangezien de elementen per rij zijn opgeslagen), telt het produkt hiervan op bij het verdubbelde kolom-subscript (verdubbeld omdat elk element twee bytes beslaat) en telt de som hiervan op bij het beginadres. Het programma maakt voor het vermenigvuldigen gebruik van een standaard schuif-en-tel-op algoritme (zie subroutine 6H).

Gebruikte registers:	Alle
Looptijd:	Ongeveer 1500 cycli, voornamelijk afhankelijk van de tijd die nodig is om de vermenigvuldiging van de rij-lengte in bytes met het rij-subscript uit te voeren.
Programmalengte:	121 bytes
Benodigd datageheugen:	Tien bytes ergens in het geheugen voor het terugkeeradres (twee bytes beginnende bij adres TERADR), het rij-subscript (twee bytes beginnende bij adres SS1), de rij-lengte in bytes (twee bytes beginnende bij adres SS1SSZ), het kolom-subscript (twee bytes beginnende bij adres SS2) en het produkt van de rij-lengte vermenigvuldigd met het rij-subscript (twee bytes beginnende bij adres PROD).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van het kolom-subscript
Hoogstwaardige byte van het kolom-subscript
Laagstwaardige byte van de rij-lengte (in bytes)
Hoogstwaardige byte van de rij-lengte (in bytes)
Laagstwaardige byte van het rij-subscript
Hoogstwaardige byte van het rij-subscript
Laagstwaardige byte van het rij-subscript
Hoogstwaardige byte van het rij-subscript
Laagstwaardige byte van het beginadres van het array
Hoogstwaardige byte van het beginadres van het array

Eindcondities

(A) = Hoogstwaardige byte van het beginadres van het element
(Y) = Laagstwaardige byte van het beginadres van het element
Het element beslaat het adres in AY en het daarop volgende adres

Voorbeelden

1. Data:

   Beginadres = $5E14_{16}$
   Kolom-subscript = 0008_{16}
   Lengte van een rij (in bytes) = $001C_{16}$ (d.w.z. elke rij bevat 0014_{10} of $000E_{16}$ elementen ter grootte van een woord)
   Rij-subscript = 0005_{16}

   Resultaat:

   Het beginadres van het element = $5E14_{16} + 0005_{16} \times 001C_{16} + 0008_{16} \times 2 = 5E14_{16} + 008C_{16} + 0010_{16} = 5EB0_{16}$.
   Zodoende is $5EB0_{16}$ het beginadres van ARRAY (5,8) en beslaat het element de adressen $5EB0_{16}$ en $5EB1_{16}$.

2. Data:

   Beginadres = $B100_{16}$
   Kolom-subscript = 0002_{16}
   Lengte van een rij (in bytes) = 0008_{16} (d.w.z. elke rij heeft 4 elementen ter grootte van een woord)
   Rij-subscript = 0006_{16}

   Resultaat:

   Het beginadres van het element = $B100_{16} + 0006_{16} \times 0008_{16} + 0002_{16} \times 2 = B100_{16} + 0030_{16} + 0004_{16} = B134_{16}$.
   Zodoende is $B134_{16}$ het beginadres van ARRAY (6,2) en beslaat het element de adressen $B134_{16}$ en $B135_{16}$

De algemeen geldende formule is:

BEGINADRES VAN ELEMENT = BEGINADRES VAN ARRAY + RIJ-SUBSCRIPT x LENGTE VAN RIJ + KOLOM-SUBSCRIPT x 2

Merk op dat een van de parameters van deze routine de lengte van een rij in bytes is. De lengte bij elementen ter grootte van een woord is het aantal kolommen (per rij) maal twee (de grootte van een element). Wij hebben voor deze parameter gekozen in plaats van het aantal kolommen of de maximale kolomindex omdat deze parameter eenmalig berekend kan worden (als de array-grenzen zijn bepaald) en elke keer bruikbaar is als het array geraadpleegd wordt. De alternatieve parameters (aantal kolommen of maximale kolomindex) zou extra berekeningen nodig maken als onderdeel van elke indexering.

```
;
;
;
;       Titel       : Tweedimensionale woord-array-indexering
;       Naam        : D2WORD
;
;
;       Doel        : Gegeven het beginadres van het woord-array 'A', de
;                     indices 'I', 'J' en de lengte van de eerste index in
;                     woorden, bereken het adres van A[I,J]. Het array wordt
;                     verondersteld te zijn opgeslagen in volgorde van rijen
;                     (A[0,0], A[0,1], ..., A[K,L]), en beide dimensies
```

```
;                 beginnen bij 0 zoals in de volgende Pascal-declaratie:
;
;                   A : ARRAY [ 0..2, 0..7 ] of WORD;
;
;     Begin      : Top van de stack
;                    Laagstwaardige byte van het terugkeeradres
;                    Hoogstwaardige byte van het terugkeeradres
;                    Laagstwaardige byte van de tweede index
;                    Hoogstwaardige byte van de tweede index
;                    Laagstwaardige byte van de lengte van de eerste index
;                    Hoogstwaardige byte van de lengte van de eerste index
;                    Laagstwaardige byte van de eerste index
;                    Hoogstwaardige byte van de eerste index
;                    Laagstwaardige byte van het beginadres van 'A'
;                    Hoogstwaardige byte van het beginadres van 'A'
;
;
;     Eind       : Register A = hoogstwaardige byte van het adres
;                  Register y = laagstwaardige byte van het adres
;
;     Registers  : Alle
;
;     Tijd       : Ongeveer 1500 cycli
;
;     Grootte    : programma 121  bytes
;                  data       10  bytes

D2WORD:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal de tweede index van de stack af en vermenigvuldig hem met
          ; twee (woorden zijn twee bytes lang)

                    pla
                    asl     A
                    sta     SS2
                    pla
                    asl     A
                    sta     SS+1

          ; Haal de lengte van de eerste index van de stack

                    pla
                    sta     SS1SZ
                    pla
                    sta     SS1SZ+1

          ; Haal de eerste index van de stack

                    pla
                    sta     SS1
                    pla
                    sta     SS1+1

          ; Vermenigvuldig de eerste index met de lengte van de eerste
          ; index met behulp van het schuif en tel op algoritme
          ; Plaats het resultaat in SS1
```

```
                lda     #0              ; Begin met 0 te vermenigvuldigen
                sta     PROD
                sta     PROD+1
                ldx     #17             ; Er moet 17 keer geshift worden
                clc
MULLP:
                ror     PROD+1          ; Shift het deelresultaat
                ror     PROD
                ror     SS1+1           ; Shift de multiplicator
                ror     SS1
                bcc     DECCNT
                clc                     ; Tel de multiplicator bij het
                                        ; deelresultaat op als het volgende
                                        ; bit van de multiplicator 1 is
                lda     SS1SZ
                adc     PROD
                sta     PROD
                lda     SS1SZ+1
                adc     PROD+1
                sta     PROD+1
DECCNT:
                dex
                bne     MULLLP

        ; Tel de tweede index erbij op (deze is al verdubbeld)

                lda     SS1
                clc
                adc     SS2
                sta     SS1
                lda     SS1+1
                adc     SS2+1
                sta     SS1+1

        ; Tel het beginadres van het array bij de index op

                pla
                clc
                adc     SS1
                tay                     ; Register Y = laagstwaardige byte
                pla
                adc     SS1+1
                tax                     ; Register X = hoogstwaardige byte
        ;
        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha                     ; Het terugkeeradres staat nu weer
                                        ; op de stack
                txa                     ; Zet waarde van X weer in A
                rst                     ; Keer terug


; Data
RETADR:         .BLOCK  2               ; Tijdelijke opslag van het
                                        ; terugkeeradres
SS1:            .BLOCK  2               ; Eerste index in het array
SS1SZ:          .BLOCK  2               ; Lengte van de eerste index
SS2:            .BLOCK  2               ; Tweede index in het array
PROD:           .BLOCK  2               ; Tijdelijke opslag voor het
                                        ; vermenigvuldigen
```

```
;
;
; Voorbeeld uitvoering
;
;

SC0506:
          ; Zet het adres van het array op de stack

                    lda     ARYADR+1      ; Hoogstwaardige byte
                    pha
                    lda     ARYADR        ; Laagstwaardige byte
                    pha

          ; Zet de eerste index op de stack

                    lda     SUBS1+1       ; Hoogstwaardige byte
                    pha
                    lda     SUBS1         ; Laagstwaardige byte
                    pha

          ; Zet de lengte van de eerste index op de stack

                    lda     SSUBS1+1
                    pha
                    lda     SSUBS1
                    pha

          ; Zet de tweede index op de stack

                    lda     SUBS2+1
                    pha
                    lda     SUBS2
                    pha
                    jsr    D2WORD         ; Bereken het adres
                    brk                   ; AY = beginadres van A[2,4]
                                          ;    = A + (2 * 8) + (4 * 2)
                                          ;    = A + 40, waarin het getal
                                          ;      2100 hexadecimaal staat

                    jmp     SC0506


;
; Data
SUBS1:              .WORD   2             ; Eerste index in het array
SSUBS1:             .WORD   16            ; Lengte van de eerste index
SUBS2:              .WORD   4             ; Tweede index in het array
ARYADR:             .WORD   ARY           ; Adres van het array

; Het array zelf (3 rijen, 8 kolommen)
ARY:                .WORD 0100H,0200H,0300H,0400H,0500H,0600H,0700H,0800H
                    .WORD 0900H,1000H,1100H,1200H,1300H,1400H,1500H,1600H
                    .WORD 1700H,1800H,1900H,2000H,2100H,2200H,2300H,2400H


                    .END
```

N-dimensionale array-indexering - 5G

Berekent het beginadres van een element van een N-dimensionaal array, gegeven het beginadres en het N-tal paren van grootten en subscripts. De grootte van een dimensie is het aantal bytes vanaf het beginadres van het element tot het beginadres van het element met een index die een groter is in de dimensie, maar dezelfde in alle andere dimensies. Verondersteld wordt dat het array is georganiseerd in rijen van oplopende volgorde (waardoor rechter subscripts veranderen voor de linker subscripts).
Merk op dat de grootte van het meest rechts subscript eenvoudigweg het aantal elementen (in bytes) is; de grootte van het volgende subscript is de grootte van de elementen vermenigvuldigd met de maximale waarde van het meest rechtse subscript plus 1, enz. Verondersteld wordt dat alle subscripts bij nul beginnen; anders dient de gebruiker de subscripts te normaliseren (zie het tweede voorbeeld aan het einde van de listing).
Procedure: Het programma lust op elke dimensie, waarbij de offset in die dimensie wordt berekend door het subscript met de grootte te vermenigvuldigen. Als de grootte makkelijk te verwerken is (een gehele macht van 2), vereenvoudigt het programma de vermenigvuldiging tot schuivingen naar links. Anders vermenigvuldigt het d.m.v. het schuif-en-tel-op algoritme van subroutine 6H. Als het programma eenmaal de algemene offset heeft berekend, telt het deze offset bij het beginadres op om het beginadres van het element te verkrijgen.

Gebruikte registers:	Alle
Looptijd:	Ongeveer 1100 cycli per dimensie plus 90 cycli vaste looptijd. Hangt voornamelijk af van de tijd die nodig is om de vermenigvuldigingen uit te voeren.
Programmalengte:	192 bytes
Benodigd datageheugen:	Elf bytes ergens in het geheugen voor het terugkeeradres (twee bytes beginnende bij adres TERADR), het huidige subscript (twee bytes beginnende bij adres SS), de huidige grootte (twee bytes beginnende bij adres SIZE), de berekende offset (twee bytes beginnende bij adres OFFSET), het aantal dimensies (een byte op adres NUMDIM) en het produkt van de grootte vermenigvuldigd met het subscript (twee bytes beginnende bij adres PROD).
Speciaal geval:	Als het aantal dimensies nul is, keert het programma terug met het beginadres in de registers A (hoogstwaardige byte) en Y (laagstwaardige byte).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Aantal dimensies
Laagstwaardige byte van de grootte van de meest rechtse dimensie
Hoogstwaardige byte van de grootte van de meest rechtse dimensie
Laagstwaardige byte van het meest rechtse subscript
Hoogstwaardige byte van het meest rechtse subscript

.
.
.

Laagstwaardige byte van de meest linkse dimensie
Hoogstwaardige byte van de meest linkse dimensie
Laagstwaardige byte van het meest linkse subscript
Hoogstwaardige byte van het meest linkse subscript
Laagstwaardige byte van het beginadres van het array
Hoogstwaardige byte van het beginadres van het array

Eindcondities

(A) = Hoogstwaardige byte van het adres van het element
(Y) = Laagstwaardige byte van het adres van het element

Het element beslaat de geheugenadressen vanaf het berekende beginadres tot en met dat adres plus het meest rechtse subscript minus 1. Ofte wel, het element beslaat de geheugenadressen START tot en met START + SIZE - 1, waarbij START het berekende adres en SIZE de grootte van het element in bytes is.

Voorbeelden

Data:

Beginadres = $3C00_{16}$
Aantal dimensies = 03_{16}
Meest rechtse subscript = 0005_{16}
Meest rechtse grootte = 0003_{16} (3-bytes waarden)
Midden-subscript = 0003_{16}
Midden-grootte = 0012_{16} (zes 3-bytes waarden)
Meest linkse subscript = 0004_{16}
Meest linkse grootte = $007E_{16}$ (zeven sets van zes 3-bytes waarden)

Resultaat:

Adres van element = $3C00_{16} + 0005_{16} \times 0003_{16} + 0003_{16} \times 0012_{16} + 0004_{16} \times 007E_{16} = 3C00_{16} + 000F_{16} + 0036_{16} + 01F8_{16} = 3E3D_{16}$.

Dus het element is ARRAY (4,3,5); het beslaat de adressen $3E3D_{16}$ tot en met $3E3F_{16}$. De maximum waarden van de diverse subscripts zijn 6 (meest linkse) en 5 (midden). Elke element bestaat uit 3 bytes.
De algemene formule is:

$$\text{BEGINADRES} = \text{BEGINADRES VAN HET ARRAY} + \sum_{i=0}^{N-1} \text{SUBSCRIPT}_i \times \text{GROOTTE}_i$$

waarbij

N het aantal dimensies is
SUBSCRIPT_i het *i* -de subscript is
GROOTTE_i de grootte van de *i* -de dimensie is

Merk op dat we de grootten van elke dimensie als parameters gebruiken om het aantal zichzelf herhalende vermenigvuldigingen te beperken en de procedure algemeen te houden. De grootten kunnen worden berekend (en opgeslagen) zodra de grenzen van het array bekend zijn. Die grootten kunnen elke keer weer worden gebruikt als het array geindexeerd wordt. Uiteraard veranderen de grootten niet als de grenzen vastliggen en dienen dan niet te worden herberekend als een onderdeel van het indexeren. De grootten zijn eveneens algemeen, aangezien de elementen zelf uit elke willekeurig aantal bytes kunnen bestaan.

```
;
;
;
;       Titel      : N-dimensionale array-indexering
;       Naam       : NDIM
;
;
;       Doel       : Berekenen van het adres van een element in een N-
;                    dimensionaal array, als er N indices en hun lengte, het
;                    aantal indices en het beginadres van het array gegeven
;                    zijn. Het array wordt verondersteld te zijn opgebouwd in
;                    oplopende rijvolgorde (A[0,0,0], A[0,0,1], .. A[0,1,0] ..
;                    A[I,J,K]). De indices worden verondersteld te beginnen
;                    bij 0 zoals in dePascal declaratie:
;
;                      A : ARRAY [ 0..10, 0..3, 0..5 ] of SOMETHING
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Het aantal dimensies
;                      Laagstwaardige byte van de lengte van de index (dim N-1)
;                      Hoogstwaardige byte van de lengte van de index (dim N-1)
;                      Laagstwaardige byte van de index  (dim N-1)
;                      Hoogstwaardige byte van de index  (dim N-1)
;                                          .
;                                          .
;                                          .
;                                          .
;                      Laagstwaardige byte van de lengte van de index (dim 0)
;                      Hoogstwaardige byte van de lengte van de index (dim 0)
;                      Laagstwaardige byte van de index  (dim 0)
;                      Hoogstwaardige byte can de index  (dim 0)
;                      Laagstwaardige byte van het beginadres van A
;                      Hoogstwaardige byte van het beginadres van A
;
;       Eind       : Register A = hoogstwaardige byte van adres
;                    Register Y = laagstwaardige byte van adres
;
;       Registers  : Alle
;
;       Tijd       : Ongeveer 1100 cycli per dimensie plus
;                    90 cycli basistijd
;
;       Grootte    : programma 192  bytes
;                    data       11  bytes
```

```
NDIM:
        ; Haal het terugkeeradres van de stack

                pla
                sta     RETADR
                pla
                sta     RETADR+1        ; Bewaar het adres tijdelijk

                pla
                sta     NUMDIM          ; Het aantal dimensies

        ; Index := 0

                lda     #0
                sta     OFFSET
                sta     OFFSET+1

        ; Kijk of het aantal dimensies 0 is

                lda     NUMDIM
                beq     ADBASE          ; Ga ervan uit dat er wel een
                                        ; beginadres is van het array

        ; Doe nu voor het aantal dimensies dat er is
        ;   Offset = offset + (index * lengte index)

LOOP:
                pla                     ; Haal de lengte van de stack
                sta     SIZE
                pla
                sta     SIZE+1

                pla                     ; Haal de index van de stack
                sta     SS
                pla
                sta     SS+1
                jsr     NXTOFF          ; Bereken de nieuwe offset
                dec     NUMDIM          ; Verlaag het aantal dimensies
                bne     LOOP            ; Ga alle dimensies af


ADBASE:
        ; Bereken het beginadres van het element
        ;   Offset := beginadres van het array + offset

                pla                     ; Laagstwaardige byte van het
                                        ; beginadres van A
                clc
                adc     OFFSET          ; Tel de laagstwaardige byte van
                                        ; het adres van A erbij op
                sta     OFFSET          ; Zet hem in OFFSET
                pla
                adc     OFFSET+1        ; Tel de hoogstwaardige byte van
                                        ; het adres van A erbij op
                sta     OFFSET+1        ; Zet hem in OFFSET+1

        ; Zet het terugkeeradres weer op de stack

                lda    RETADR+1
                pha
                lda    RETADR
                pha
                lda     OFFSET+1        ; Zet de OFFSET in de A en Y
```

```
                ldy     OFFSET          ; registers
                rts


;
; Subroutine : NXTOFF
; Doel       : Bereken OFFSET := OFFSET + (index * lengte index)
; Begin      : OFFSET is de huidige offset
;              Index is de huidige index
;              Lengte index is de lengte van de huidige index
; Einde      : OFFSET := OFFSET + (index * lengte index)
; Registers  : Allemaal
;


NXTOFF:
        ; Test of de lengte een macht van twee of acht is (Dan is het
        ; vermenigvuldigen gemakkelijk)

                lda     SIZE+1          ; Hoogstwaardige byte = 0?
                bne     BIGSZ           ; Spring als de lengte groot is
                lda     SIZE
                ldy     #0              ; Y is de index in het makkelijke
                                        ; array
                ldx     #SZEASY         ; Lengte van dat array in X
EASYLP:
                cmp     EASY,Y
                beq     ISEASY          ; Spring als het een makkelijk
                                        ; nummer is
                iny                     ; Verhoog de index
                dex                     ; Verlaag de lengte
                bne     EASYLP          ; Spring zolang er nog makkelijke
                                        ; elementen zijn
                beq     BIGSZ           ; Spring als het geen makkelijke
                                        ; vermenigvuldiging gaat worden

ISEASY:
                cpy     #0
                beq     ADDOFF          ; Spring als de shift-factor 0 is

        ; Elementlengte * index kan worden berekend
        ; door naar links te schuiven

SHL:
                asl     SS              ; Shift naar links
                rol     SS+1
                dey
                bne ADDOFF              ; Ga door tot u genoeg geshift hebt

BIGSZ:
        ; Dit is geen gemakkelijke vermenigvuldiging, dus moeten we het
        ; maar op de moeilijke manier doen

                lda     #0              ; Begin het deelprodukt met 0
                sta     PROD
                sta     PROD+1
                ldx     #17             ; We moeten 17 maal schuiven
                clc
MULLP:
                ror     PROD+1          ; Shift het deelprodukt
                ror     PROD
                ror     SS+1            ; Shift de multiplicator
                ror     SS
```

```
                bcc     DECCNT
                clc                     ; Tel de multiplicant bij het
                                        ; deelprodukt op
                lda     SIZE
                adc     PROD
                sta     PROD
                lda     SIZE+1
                adc     PROD+1
                sta     PROD+1
DECCNT:
                dex
                bne     MULLP

ADDOF:
                lda     SS
                clc
                adc     OFFSET          ; Tel de laagstwaardige byte van de
                                        ; offset erbij
                sta     OFFSET
                lda     SS+1
                adc     OFFSET+1        ; Tel de hoogstwaardige byte van de
                                        ; offset erbij
                rts


EASYAY:   ; De shift-factoren voor het makkelijk vermenigvuldigen

                .BYTE   1               ; 0
                .BYTE   2               ; 1
                .BYTE   4               ; 2
                .BYTE   8               ; 3
                .BYTE   16.             ; 4
                .BYTE   32.             ; 5
                .BYTE   64.             ; 6
                .BYTE   128.            ; 7
SZEASY:         .EQU    $-EASYAY


;
; Data
RETADR:         .BLOCK  2               ; Tijdelijke bewaarplaats van het
                                        ; terugkeeradres
SS:             .BLOCK  2               ; Index in het array
SIZE:           .BLOCK  2               ; Lengte van een array-element
OFFSET:         .BLOCK  2               ; Tijdelijke bewaarplaats voor het
                                        ; berekenen van de index
NUMDIM:         .BLOCK  1               ; Het aantal dimensies
PROD:           .BLOCK  2               ; Tijdelijke opslag van het
                                        ; deelprodukt in NXTOFF


;
;
; Voorbeeld uitvoering
;

SC0507:
          ;
          ; Zoek het adres van AY1[1,3,0]
          ; Omdat de ondergrenzen van het array 1 gelijk aan nul zijn, is
;           het niet nodig af te ronden
```

```
; Zet het beginadres van ARRAY 1 op de stack

        lda     AY1ADR+1
        pha
        lda     AY1ADR
        pha

; Zet index 1 plus zijn lengte op de stack

        lda     #0
        pha
        lda     #1
        pha
        lda     #0
        pha
        lda     #A1SZ1
        pha

; Zet index 2 plus zijn lengte op de stack

        lda     #0
        pha
        lda     #3
        pha
        lda     #0
        pha
        lda     #A1SZ2
        pha

; Zet index 3 plus zijn lengte op de stack

        lda     #0
        pha
        lda     #0
        pha
        lda     #0
        pha
        lda     #A1SZ3
        pha

; Zet het aantal dimensies op de stack

        lda     #A1DIM
        pha

; Bereken het adres van het element AY1[1,3,0]

        jsr     NDIM            ; Bereken het adres
        brk                     ; AY = beginadres van het element
                                ;    = ARY + 1 * 126 + 3 * 21 +
                                ;      0 * 3
                                ;    = ARY + 189


; Bereken het adres van het element AY2[-1,6]
; Omdat de ondergrenzen van ARRAY 2 niet allemaal 0 zijn moeten
; we eerst het array normaliseren

; Zet het beginadres van ARRAY 2 op de stack

        lda     AY2ADR+1
```

```
                pha
                lda     AY2ADR
                pha

        ; Zet index 1 (= index - ondergrens) plus zijn lengte op de stack

                lda     #-1
                sec
                sbc     #A2D1L
                tax                     ; Bewaar de laagstwaardige byte
                lda     #0FFH           ; Hoogstwaardige byte van index - 1
                sbc     #0FFH           ; Hoogstwaardige byte van A2D1L
                pha                     ; Zet het op de stack
                txa
                pha                     ; Zet het op de stack
                lda     #0
                pha
                lda     #A2SZ1
                pha

        ; Zet index 2 (= index - ondergrens) plus zijn lengte
        ; op de stack

                lda     #6
                sec
                sbc     #A2D2L
                tax                     ; Bewaar de laagstwaardige byte
                lda     #0
                sbc     #0
                pha                     ; Zet de hoogstwaardige byte op de
                                        ; stack
                txa
                pha                     ; Zet de laagstwaardige byte op de
                                        ; stack
                lda     #0
                pha
                lda     #A2SZ2
                pha

        ; Zet het aantal dimensies op de stack

                lda     #A2DIM
                pha

        ; Bereken het adres van het element AY2[-1,6]

                jsr     NDIM            ; Bereken het adres
                brk                     ; AY = beginadres van het element
                                        ;    = ARY + (((-1) - (-5)) * 18)
                                        ;          + ((6 - 2) * 2)
                                        ;    = ARY + 80

                jmp     SC0507


; Data
AY1ADR:         .WORD   AY1             ; Beginadres van ARRAY 1
AY2ADR:         .WORD   AY2             ; Beginadres van ARRAY 2

;
; AY1 : ARRAY [A1D1L..A1D1H, A1D2L..A1D2H, A1D3L..A1D3H] of THREE BYTE
;       ELEMENTS
;             [    0 .. 3  ,    0 .. 5   ,    0 .. 6   ]
```

```
;
A1DIM:          .EQU    3           ; Aantal dimensies
A1D1L:          .EQU    0           ; Ondergrens index 1
A1D1H:          .EQU    3           ; Bovengrens index 1
A1D2L:          .EQU    0           ; Ondergrens index 2
A1D2H:          .EQU    5           ; Bovengrens index 2
A1D3L:          .EQU    0           ; Ondergrens index 3
A1D3H:          .EQU    6           ; Bovengrens index 3
A1SZ3:          .EQU    3           ; Lengte van een element
                                    ; in dimensie 3
                                    ; Lengte van een element
                                    ; in dimensie 2
A1SZ2:          .EQU    ((A1D3H - A1D3L) + 1) * A1SZ3
                                    ; Lengte van een element
                                    ; in dimensie 1
A1SZ1:          .EQU    ((A1D2H - A1D2L) + 1) * A1SZ2
                                    ; Het array zelf
AY1:            .BLOCK  ((A1D1H - A1D1L) + 1) * A1SZ1

;
; AY2 : ARRAY [A1D1L..A1D1H, A1D2L..A1D2H] OF WORD
;             [  -5 .. -1  ,    2 .. 10  ]
;
A2DIM:          .EQU    2           ; Aantal dimensies
A2D1L:          .EQU    -5          ; Ondergrens index 1
A2D1H:          .EQU    -1          ; Bovengrens index 1
A2D2L:          .EQU    2           ; Ondergrens index 2
A2D2H:          .EQU    10          ; Bovengrens index 2
A2SZ2:          .EQU    2           ; Lengte van een element
                                    ; in dimensie 2
                                    ; Lengte van een element
                                    ; in dimensie 1
A2SZ1:          .EQU    ((A2D2H - A2D2L) + 1) * A2SZ2
                                    ; Het array zelf
AY2:            .BLOCK  ((A2D1H - A2D1L) + 1) * A2SZ1


                .END
```

16-bits optelling - 6A

Telt twee 16-bits getallen die van de stack zijn verkregen, bij elkaar op en zet de som op de stack. Alle 16-bit getallen worden op de voor de 6502 gebruikelijke wijze opgeslagen: laagstwaardige byte boven hoogstwaardige byte.
Procedure: Het programma wist eerst de overdrachtvlag telt de getallen byte voor byte bij elkaar op, beginnende bij de laagstwaardige bytes. De overdrachtvlag wordt gezet naar de optelling van de hoogstwaardige bytes.

Gebruikte registers:	A, P, Y
Looptijd:	80 cycli
Programmalengte:	38 bytes
Benodigd datageheugen:	Vier bytes ergens in het geheugen voor het tweede getal (twee bytes beginnende bij adres ADEND2) en het terugkeeradres (twee bytes beginnende bij adres RETADR).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van het eerste getal
Hoogstwaardige byte van het eerste getal
Laagstwaardige byte van het tweede getal
Hoogstwaardige byte van het tweede getal

Eindcondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van de som
Hoogstwaardige byte van de som

Voorbeelden

1.	Data:	Eerste getal = $03E1_{16}$ Tweede getal = $07E4_{16}$
	Resultaat:	Som = $0BC5_{16}$ Overdrachtvlag = 0
2.	Data:	Eerste getal = $A45D_{16}$ Tweede getal = $97E1_{16}$
	Resultaat:	Som = $3C3E_{16}$ Overdrachtvlag = 1

```
;
;
;       Titel      : 16-bits optelling
;       Naam       : ADD16
;
;
;       Doel       : Tel twee 16-bits woorden met of zonder teken bij elkaar
;                    op en geef een 16-bits resultaat terug met of zonder
;                    teken
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Laagstwaardige byte van operand 2
;                      Hoogstwaardige byte van operand 2
;                      Laagstwaardige byte van operand 1
;                      Hoogstwaardige byte van operand 1
;
;       Eind       : Som = operand 1 + operand 2
;                    Top van de stack
;                      Laagstwaardige byte van de som
;                      Hoogstwaardige byte van de som
;
;       Registers : A, P, Y
;
;       Tijd       : 80 cycli
;
;       Grootte    : programma 38  bytes
;                    data        4  bytes

ADD16:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal operand 2 van de stack

                    pla
                    sta     ADEND2
                    pla
                    sta     ADEND2+1

          ; Som = operand 1 + operand 2

                    pla
                    clc
                    adc     ADEND2
                    tay                     ; Bewaar deze even
                    pla
                    adc     ADEND2+1

          ; Zet het resultaat weer op de stack

                    pha                     ; Hoogstwaardige byte
                    tya
                    pha                     ; Laagstwaardige byte
```

```
        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha
                rts


; Data
ADEND2:         .BLOCK  2               ; Tijdelijke opslag operand 2
RETADR:         .BLOCK  2               ; Opslag terugkeeradres


;
;
; Voorbeeld uitvoering
;
;

SC0601:
        ; Som van OPRND1 en OPRND2

                lda     OPRND1+1
                pha
                lda     OPRND1
                pha
                lda     OPRND2+1
                pha
                lda     OPRND2
                pha
                jsr     ADD16
                pla
                tay
                pla
                brk                     ; A = hoogstwaardige byte
                                        ; Y = laagstwaardige byte
                jmp SC0601


; Test-data, verander deze zelf naar andere waarden

OPRND1:         .WORD   1023
OPRND2:         .WORD   123             ; 1023 + 123 = 1146 = 047AH


                .END
```

16-Bit aftrekking - 6B

Trekt twee 16-bits getallen die van de stack zijn gehaald, van elkaar af en zet het verschil op de stack. Alle 16-bits getallen worden op de voor de 6502 gebruikelijke manier opgeslagen met de laagstwaardige byte boven de hoogstwaardige byte. De aftrekker (het getal dat afgetrokken moet worden) wordt boven het aftrektal (getal waarvan de aftrekker wordt afgetrokken) opgeslagen. De overdrachtvlag fungeert in haar gebruikelijke rol bij de 6502, als een geinverteerd leenbit.
Procedure: Het programma zet eerst de overdrachtvlag (de geinverteerde leen) en trekt byte voor byte de aftrekker van het aftrektal af, beginnende bij de laagstwaardige bytes. De overdracht wordt gezet naar de aftrekking van de hoogstwaardige bytes.

Gebruikte registers:	A, P, Y
Looptijd:	80 cycli
Programmalengte:	38 bytes
Benodigd datageheugen:	Vier bytes ergens in het geheugen voor de aftrekker (twee bytes beginnende bij adres SUBTRA) en het terugkeeradres (twee bytes beginnende bij adres RETADR).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van de aftrekker
Hoogstwaardige byte van de aftrekker
Laagstwaardige byte van het aftrektal
Hoogstwaardige byte van het aftrektal

Eindcondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het verschil (aftrektal - aftrekker)
Hoogstwaardige byte van het verschil (aftrektal - aftrekker)

Voorbeelden

1. Data: Aftrektal = $A45D_{16}$
Aftrekker = $97E1_{16}$

Resultaat: Verschil = Aftrektal - Aftrekker = $0C7C_{16}$
Overdrachtvlag = 1 (geen leen)

2. Data: Aftrektal = $03E1_{16}$
Aftrekker = $07E4_{16}$

Resultaat: Verschil = Aftrektal - Aftrekker = $FBFD_{16}$
Overdrachtvlag = 0 (leen opgetreden)

```
;
;
;
;       Titel     : 16-bits aftrekking
;       Naam      : SUB16
;
;
;       Doel      : Trek twee 16-bits woorden met of zonder teken van elkaar
;                   af en geef een 16-bits resultaat terug met of zonder
;                   teken
;
;
;       Begin     : Top van de stack
;                     Laagstwaardige byte van het terugkeeradres
;                     Hoogstwaardige byte van het terugkeeradres
;                     Laagstwaardige byte van operand 2
;                     Hoogstwaardige byte van operand 2
;                     Laagstwaardige byte van operand 1
;                     Hoogstwaardige byte van operand 1
;
;       Eind      : Verschil = operand 1 - operand 2
;                   Top van de stack
;                     Laagstwaardige byte van de som
;                     Hoogstwaardige byte van de som
;
;       Registers : A, P, Y
;
;       Tijd      : 80 cycli
;
;       Grootte   : programma 38  bytes
;                   data       4  bytes

SUB16:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal operand 2 van de stack

                    pla
                    sta     SUBSTRA
                    pla
                    sta     SUBSTRA+1

          ; Verschil = operand 1 + operand 2

                    pla
                    sec
                    sbc     SUBSTRA
                    tay                     ; Bewaar deze even
                    pla
                    sbc     SUBSTRA+1
```

```
        ; Zet het resultaat weer op de stack

                pha                     ; Hoogstwaardige byte
                tya
                pha                     ; Laagstwaardige byte

        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha
                rts


; Data
SUBSTRA:        .BLOCK  2               ; Tijdelijke opslag operand 2
RETADR:         .BLOCK  2               ; Opslag terugkeeradres


;
;
; Voorbeeld uitvoering
;
;

SC0602:
        ; Verschil van OPRND1 en OPRND2

                lda     OPRND1+1
                pha
                lda     OPRND1
                pha
                lda     OPRND2+1
                pha
                lda     OPRND2
                pha
                jsr     SUB16
                pla
                tay
                pla
                brk                     ; A = hoogstwaardige byte
                                        ; Y = laagstwaardige byte
                jmp SC0602


; Test-data, verander deze zelf naar andere waarden
OPRND1:         .WORD   123
OPRND2:         .WORD   1023            ;123 - 1023 = -900 = 0FC7H


                .END
```

16-Bits vermenigvuldiging - 6C

Vermenigvuldigt twee 16-bits getallen die van de stack zijn gehaald, met elkaar en zet het laagstwaardige woord van het produkt boven op de stack. Alle 16-bits getallen worden op de voor de 6502 gebruikelijke wijze met de laagstwaardige boven de hoogstwaardige byte opgeslagen.
Procedure: Het programma gebruikt een gewoon tel-op-en-schuif algoritme, waarbij het vermenigvuldigtal elke keer bij het deelprodukt wordt opgeteld als hij een 1-bit in de vermenigvuldiger vindt. Het deelprodukt en de vermenigvuldiger worden 17 keer geschoven (het aantal bits plus 1 in de vermenigvuldiger), waarbij de extra lus nodig is om de uiteindelijke overdrachtvlag in het produkt te schuiven. Het programma geeft (beginnende met de hoogstwaardige byte) een volledig 32-bits ongetekend deelprodukt in de geheugenlocaties HIPROD+1, HIPROD, MLIER+1 en MLIER terug. Het laagstwaardige woord van het produkt vervangt de vermenigvuldiger omdat deze geschoven en bekeken is op 1-bits.

Gebruikte registers:	Alle
Looptijd:	Ongeveer 650 tot 1100 cycli, voornamelijk afhankelijk van het aantal 1-bits in de vermenigvuldiger
Programmalengte:	238 bytes
Benodigd datageheugen :	Acht bytes ergens in het geheugen voor het vermenigvuldigtal (twee bytes beginnende bij adres MCAND), de vermenigvuldiger en laagstwaardige woord van het produkt (twee bytes beginnende bij adres MLIER), het hoogstwaardige woord van het deelprodukt (twee bytes beginnende bij adres HIPROD) en het terugkeeradres (twee bytes beginnende bij adres RETADR).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van de vermenigvuldiger
Hoogstwaardige byte van de vermenigvuldiger
Laagstwaardige byte van het vermenigvuldigtal
Hoogstwaardige byte van het vermenigvuldigtal

Eindcondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het laagstwaardige woord van het produkt
Hoogstwaardige byte van het laagstwaardige woord van het produkt

Voorbeelden

1. Data: Vermenigvuldiger = 0012_{16} (18_{10})
   Vermenigvuldigtal = $03D1_{16}$ (977_{10})

   Resultaat: Produkt = $44B2_{16}$ (17.586_{10})

2. Data: Vermenigvuldiger = $37D1_{16}$ (14.289_{10})
   Vermenigvuldigtal = $A045_{16}$ (41.029_{10})

   Resultaat: Produkt = $AB55_{16}$ (43.861_{10}). Dit is in feite het laagstwaardige 16-bits woord van het 32-bits produkt $22F1AB55_{16}$ ($586.264.381_{10}$)

Merk op dat MUL16 alleen het laagstwaardige woord van het produkt geeft. Dit is om verenigbaar te blijven met andere 16-bits rekenkundige handelingen. Het hoogstwaardige woord van het produkt is beschikbaar in de geheugenlocaties HIPROD (laagstwaardige byte) en HIPROD+1 (hoogstwaardige byte), maar de gebruiker dient te onthouden dat dit alleen klopt als de twee getallen ongetekend zijn. Als vermenigvuldiger en vermenigvuldigtal getekend zijn en een van beide negatief is, moet de gebruiker het teken van het produkt bepalen en de negatieve getallen vervangen door hun absolute waarden (tweecomplementen) voordat MUL16 aangeroepen wordt.

```
;       Titel     : 16-bits vermenigvuldigen
;       Naam      : MUL16
;
;
;       Doel      : Multipliceer 2 16-bits woorden met of zonder teken en
;                   keer terug met een 16-bits antwoord met of zonder teken
;
;
;       Begin     : Top van de stack
;                     Laagstwaardige byte van het terugkeeradres
;                     Hoogstwaardige byte van het terugkeeradres
;                     Laagstwaardige byte van de multiplicator
;                     Hoogstwaardige byte van de multiplicator
;                     Laagstwaardige byte van de multiplicant
;                     Hoogstwaardige byte van de multiplicant
;
;       Eind      : Produkt = multiplicant * multiplicator
;                   Top van de stack
;                     Laagstwaardige byte van het produkt
;                     Hoogstwaardige byte van het produkt
;
;       Registers : Alle
;
;       Tijd      : Ongeveer 650 tot 1100 cycli
;
;       Grootte   : programma 238  bytes
;                   data        8  bytes

MUL16:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
```

```
        pla
        sta     RETADR+1

; Haal de multiplicator van de stack

        pla
        sta     MLIER
        pla
        sta     MLIER+1

; Haal de multiplicant van de stack

        pla
        sta     MCAND
        pla
        sta     MCAND+1

; Voer de vermenigvuldiging uit met behulp van het schuif en
; optel algoritme. Dit algoritme produceert een 32-bits
; antwoord in HIPROD en MLIER, met HIPROD als hoogstwaardige
; woord

        lda     #0
        sta     HIPROD
        sta     HIPROD+1        ; Maak het hoogstwaardige woord
                                ; leeg
        ldx     #17             ; Het aantal bits in de
                                ; multiplicator + 1
                                ; Deze laatste is om de laatste
                                ; overdrachtvlag mee in het
                                ; produkt te nemen
        clc                     ; Maak de overdrachtvlag leeg

MULLP:
; Als het volgende bit 1 is dan
;  HIPROD := HIPROD + MULTIPLICANT

        ror     HIPROD+1
        ror     HIPROD
        ror     MLIER+1
        ror     MLIER
        bcc     DECCNT          ; Spring als het volgende bit 0 is
        clc                     ; Het volgende bit was 1, tel dus
                                ; nu de multiplicant bij het
                                ; produkt op
        lda     MCAND
        adc     HIPROD
        sta     HIPROD
        lda     MCAND+1
        adc     HIPROD+1
        sta     HIPROD+1        ; Overdrachtvlag = overloop bij de
                                ; som

DECCNT:
        dex
        bne     MULLP           ; Spring totdat er 17 maal geshift
                                ; is

; Zet het laagstwaardige woord op de stack

        lda     MLIER+1
        pha
```

```
                lda     MLIER
                pha

        ; Zet het terugkeeradres op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha
                rts


; Data
MCAND:          .BLOCK 2            ; Multiplicant
MLIER:          .BLOCK 2            ; Multiplicator plus laagstwaardige
                                    ; woord van het produkt

HIPROD:         .BLOCK 2            ; Hoogstwaardige woord van het
                                    ; produkt
RETADR:         .BLOCK 2            ; Het terugkeeradres


;
; Voorbeeld uitvoering
;
;

SC0603:
        ; Vermenigvuldig OPRND1 met OPRND2 en zet het antwoord in RESULT

                lda     OPRND1+1
                pha
                lda     OPRND1
                pha
                lda     OPRND2+1
                pha
                lda     OPRND2
                pha
                jsr     MUL16
                pla
                sta     RESULT
                pla
                sta     RESULT+1
                brk                 ; Het resultaat van
                                    ; 1023 * -2 = -2046 = 0F802H
                                    ;   RESULT    = 02H
                                    ;   RESULT+1  = F8H
                jmp     SC0603


OPRND1:         .WORD   -2
OPRND2:         .WORD   1023
RESULT:         .BLOCK  2           ; 2-bytes resultaat


                .END
```

16-Bits Deling - 6D

Deelt twee 16-bits getallen die van de stack zijn gehaald, door elkaar en plaatst het quotient of het restgetal boven op de stack. Er zijn vier ingangspunten in het programma: SDIV16 geeft een 16-bits getekend quotient terug na twee getekende 16-bits getallen gedeeld te hebben, UDIV16 geeft een 16-bits ongetekend quotient na twee ongetekende 16-bits getallen te hebben gedeeld, SREM16 geeft een getekend 16-bits restgetal terug na twee getekende 16-bits getallen te hebben gedeeld en UREM16 geeft een ongetekend 16-bits restgetal terug na twee ongetekende 16-bits getallen te hebben gedeeld. Alle 16-bits getallen worden op de voor de 6502 gebruikelijke wijze opgeslagen met de laagstwaardige byte boven de hoogstwaardige byte. De deler wordt boven het deeltal opgeslagen. Als de deler nul is, wordt de overdrachtvlag gezet en wordt een nul als resultaat teruggegeven; in alle andere gevallen wordt de overdracht gewist.
Procedure: Als de getallen getekend zijn, bepaalt het programma het teken van het quotient en neemt het de absolute waarde van elk negatief getal. Het moet ook het teken van het deeltal bewaren, aangezien dit het teken van het restgetal bepaalt. Het programma voert dan de feitelijke ongetekende deling uit met behulp van het gebruikelijke schuif-en-trek-af algoritme. Hierbij worden quotient en deeltal geschoven en een 1-bit in het quotient geplaatst elke keer als een proefaftrekking succesvol is. Als de getallen getekend zijn, moet het programma elk negatief resultaat (quotient of restgetal) aftrekken van nul. De overdrachtvlag wordt gewist als de deling goed is en gezet als de deler nul blijkt te zijn. Een nuldeler veroorzaakt ook een terugkeer met nul als resultaat (quotient of restgetal).

Gebruikte registers:	Alle
Looptijd:	Ongeveer 1000 tot 1160 cycli, voornamelijk afhankelijk van het aantal proefaftrekkingen die succesvol zijn en dus een vervanging van het vorige deeltal door het restgetal noodzakelijk maken.
Programmalengte:	293 bytes
Benodigd datageheugen:	Elf bytes ergens in het geheugen. Deze worden als volgt benut: twee bytes voor de deler (beginnende bij adres DVSOR); vier bytes voor het vergrote deeltal (beginnende bij adres DVEND), ook gebruikt voor het quotient en het restgetal; twee bytes voor het terugkeeradres (beginnende bij het adres RETADR); een byte voor het teken van het quotient (adres SQUOT); een byte voor het teken van het restgetal (adres SREM); en een byte voor een index naar het resultaat (adres RSLTIX).
Speciaal geval:	Als de deler nul is, is bij terugkeer de overdrachtvlag gezet en het resultaat nul. Zowel het quotient als het restgetal zijn nul.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van de deler
Hoogstwaardige byte van de deler
Laagstwaardige byte van het deeltal
Hoogstwaardige byte van het deeltal

Eindcondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het resultaat
Hoogstwaardige byte van het resultaat

Als de deler geen nul is, dan is de overdracht 0 en het resultaat normaal. Als de deler nul is, is de overdracht 1 en het resultaat 0000_{16}.

Voorbeelden

1. Data: Deeltal = $03E0_{16} = 992_{10}$
   Deler = $00B6_{16} = 182_{10}$

   Resultaat: Quotient (van UDIV16) = 0005_{16}
   Restgetal (van UREM16) = $0052_{16} = 0082_{10}$
   Overdrachtvlag = 0 (geen deling-door-nul fout)

2. Data: Deeltal = $D73A_{16} = -10.438_{10}$
   Deler = $2F1_{16} = 753_{10}$

   Resultaat: Quotient (van SDIV16) = $FFF3_{16} = -13_{10}$
   Restgetal (van SREM16) = $FD77_{16} = -649_{10}$
   Overdrachtvlag = 0 (geen deling-door-nul fout)

Merk op dat wij ervan uitgaan dat het restgetal van een getekende deling zowel positief als negatief kan zijn. In onze methode neemt het restgetal altijd het teken van deeltal over. De gebruiker kan op een eenvoudige wijze het quotient controleren en de vorm veranderen zodat hij of zij een restgetal krijgt dat altijd positief is. In dat geval zou het resultaat van Voorbeeld 2 zijn:

Quotient = $FFF2_{16} = -14_{10}$
Restgetal (altijd positief) = $0068_{16} = 104_{10}$

Ongeacht welk ingangspunt is gebruikt, berekent het programma altijd zowel het quotient als het restgetal. Bij terugkeer is het quotient beschikbaar in de adressen DVEND en DVEND+1 (hoogstwaardige byte in DVEND+1) en het restgetal in de adressen DVEND+2 en DVEND+3 (hoogstwaardige byte in DVEND+3). Zodoende kan de gebruiker altijd het resultaat verkrijgen, ook als het niet op de stack opgeslagen is.

```
;
;
;     Titel      : 16-bits deling
;     Naam       : SDIV16, UDIV16, SREM16, UREM16
;
;
;     Doel       : SDIV16
;                    Deel twee 16-bits getallen met teken en retourneer een
;                    16-bits quotient met teken
```

```
;                   UDIV16
;                     Deel twee 16-bits getallen zonder teken en retourneer
;                     een 16-bits quotient zonder teken
;
;                   SREM16
;                     Deel twee 16-bits getallen met teken en retourneer een
;                     16-bits rest met teken
;
;                   UREM16
;                     Deel twee 16-bits getallen zonder teken en retourneer
;                     een 16-bits rest zonder teken
;
;
;     Begin      : Top van de stack
;                     Laagstwaardige byte van het terugkeeradres
;                     Hoogstwaardige byte van het terugkeeradres
;                     Laagstwaardige byte van de noemer
;                     Hoogstwaardige byte van de noemer
;                     Laagstwaardige byte van de teller
;                     Hoogstwaardige byte van de teller
;
;     Eind       : Top van de stack
;                     Laagstwaardige byte van het resultaat
;                     Hoogstwaardige byte van het resultaat
;
;     Registers : Alle
;
;     Tijd       : Ongeveer 1000 tot 1160 cycli
;
;     Grootte    : programma 293  bytes
;                  data       13  bytes

; Deling zonder teken
UDIV16:
                    lda     #0            ; Resultaat is quotient (index = 0)
                    beq     UDIVMD

; Rest zonder teken
UREM16:
                    lda     #2            ; Rest = index 2

UDIVMD:
                    sta     RSLTIX        ; Bewaar de index

          ; Haal het terugkeeradres van de stack

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal de teller van de stack

                    pla
                    sta     DVSOR
                    pla
                    sta     DVSOR+1

          ; Haal de noemer van de stack

                    pla
```

```
                sta     DVEND
                pla
                sta     DVEND+1

        ; Voer de deling uit

                jsr     UDIV
                bcc     DIVOK
DIVER:
                jmp     EREXIT
DIVOK:
                jmp     OKEXIT


; Delen met teken

SDIV16:
                lda     #0              ; Quotient = index 0
                beq     SDIVMD

SDREM16:
                lda     #2              ; Rest = index 2

SDIVMD:
                sta     RSLTIX          ; Bewaar de index

        ; Haal het terugkeeradres van de stack

                pla
                sta     RETADR
                pla
                sta     RETADR+1

        ; Haal de teller van de stack

                pla
                sta     DVSOR
                pla
                sta     DVSOR+1

        ; Haal de noemer van de stack

                pla
                sta     DVEND
                pla
                sta     DVEND+1

        ; Bepaal het teken van het quotient door een 'exlusive or' van de
        ; hoogstwaardige bytes te nemen. Als de tekens gelijk zijn dan is
        ; bit 7 gelijk aan 0 en is het quotient positief, anders is het

        ; quotient negatief

                lda     DVEND+1
                eor     DVSOR+1
                sta     SQUOT

        ; Het teken van de rest is het teken van de teller

                lda     DVEND+1
                sta     SREM
```

```
        ; Neem de absolute waarde van de noemer

                lda     DVSOR+1
                bpl     CHKDE           ; Spring als deze al positief is
                lda     #0              ; Trek het getal van 0 af om het
                sec                     ; positief te maken
                sbc     DVSOR
                sta     DVSOR
                lda     #0
                sbc     DVSOR+1
                sta     DVSOR+1

        ; Neem de absolute waarde van de teller
CHKDE:
                lda     DVEND+1
                bpl     DODIV           ; Spring als deze al positief is
                lda     #0              ; Trek het getal van 0 af om het
                sec                     ; positief te maken
                sbc     DVEND
                sta     DVEND
                lda     #0
                sbc     DVEND+1
                sta     DVEND+1

        ; Deel de absolute waarden

DODIV:
                jsr     UDIV
                bcs     EREXIT          ; U mag niet door 0 delen

        ; Maak het quotient positief als het negatief is

                lda     SQUOT
                bpl     DOREM           ; Spring als deze al positief is
                lda     #0              ; Trek het getal van 0 af om het
                sec                     ; positief te maken
                sbc     DVEND
                sta     DVEND
                lda     #0
                sbc     DVEND+1
                sta     DVEND+1

        ; Keer het teken van de rest om als het negatief is
DOREM:
                lda     SREM
                bpl     OKEXIT          ; Spring als deze al positief is
                lda     #0              ; Trek het getal van 0 af om het
                sec                     ; positief te maken
                sbc     DVEND+2
                sta     DVEND+2
                lda     #0
                sbc     DVEND+3
                sta     DVEND+3
                jmp     OKEXIT

        ; Foute boel (overdrachtvlag = 1, resultaat = 0)

EREXIT:
                lda     #0
                sta     DVEND
                sta     DVEND+1
```

```
                sta     DVEND+2
                sta     DVEND+3
                sec                     ; Overdrachtvlag = 1 duidt op een
                                        ; fout
                bcc     DVEXIT

        ; Alles ging goed (overdrachtvlag = 0)
OKEXIT:
                clc                     ; Overdrachtvlag = 0 betekent:
                                        ; alles ok

DVEXIT:
        ; Zet het antwoord op de stack

                ldx     RSLTIX          ; Haal de index op (0 = quotient,
                                        ; 2 = rest)
                lda     DVEND+1,X
                pha
                lda     DVEND,X
                pha

        ; Zet het terugkeeradres op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha
                rts


;
; Subroutine : UDIV
; Doel       : Deel twee 16-bits getallen op elkaar
; Begin      : DVEND = teller
;              DVSOR = noëmer
; Einde      : DVEND = quotient
;              DVEND+2 = rest
; Registers  : Alle
;

UDIV:
        ; Maak hoogstwaardige woord van de teller 0
        ; We zullen dit woord verder woord[1] noemen

                lda     #0
                sta     DVEND+2
                sta     DVEND+3

        ; Kijk eerst of het delen door 0 is

                lda     DVSOR
                ora     DVSOR+1
                bne     OKUDIV          ; Spring als dit niet het geval is,
                                        ; anders
                sec                     ; geef een foutmelding
                rts                     ; en keer terug

        ; Voer de deling uit door herhaalt af te trekken

OKUDIV:
                ldx     #16             ; Er moeten 16-bits afgewerkt
                                        ; worden
```

```
DIVLP:
                rol     DVEND
                rol     DVEND+1
                rol     DVEND+2
                rol     DVEND+3

        ; Test of woord[1] kleiner dan de noemer is

CHKLT:
                sec
                lda     DVEND+2
                sbc     DVSOR
                tay                     ; Bewaar het in Y
                lda     DVEND+3
                sbc     DVSOR+1         ; Trek de hoogstwaardige bytes
                                        ; van elkaar af
                bcc     DECCNT          ; Spring als teller < noemer
                                        ; en overdrachtvlag
                sty     DVEND+2         ; anders teller[1]
                sta     DVEND+3         ; = teller[1] - noemer


DECCNT:
                dex
                bne     DIVLP
                rol     DVEND
                rol     DVEND+1         ; Stop de laatste overdrachtvlag
                                        ; in het resultaat
                clc                     ; Maak de overdrachtvlag leeg:
                                        ; geen fouten

                rts


; Data

DVSOR:          .BLOCK  2               ; De noemer
DVEND:          .BLOCK  2               ; De teller[1] en het quotient
                .BLOCK  2               ; De teller[0] en de rest
RETADR:         .BLOCK  2               ; Het terugkeeradres
SQUOT:          .BLOCK  1               ; Het teken van de deling
SREM:           .BLOCK  1               ; het teken van de rest
RSLTIX:         .BLOCK  1               ; Index in het resultaat:
                                        ;    0 = quotient
                                        ;    2 = rest



;
;
; Voorbeeld uitvoering
;
;

SC0604:
        ; Deling met teken, OPRND1 / OPRND2, bewaar het quotient in QUOT

                lda     OPRND1+1
                pha
                lda     OPRND1
                pha
                lda     OPRND2+1
```

```
        pha
        lda     OPRND2
        pha
        jsr     SDIV16          ; Deling met teken
        pla
        sta     QUOT
        pla
        sta     QUOT+1
        brk                     ; Resultaat van -1023 / 123 = -8
                                ;  QOUT     = F8 hex
                                ;  QUOT + 1 = FF hex

; Deling zonder teken, OPRND1 / OPRND2, bewaar het quotient in
; QUOT

        lda     OPRND1+1
        pha
        lda     OPRND1
        pha
        lda     OPRND2+1
        pha
        lda     OPRND2
        pha
        jsr     UDIV16                  ; Deling zonder teken
        pla
        sta     QUOT
        pla
        sta     QUOT+1
        brk                     ; Resultaat van 64513 / 123 = 524
                                ;    QOUT     = 0C hexadecimaal
                                ;    QUOT + 1 = 02 hexadecimaal

; Rest van deling met teken, OPRND1 / OPRND2, bewaar de rest in
; REM

        lda     OPRND1+1
        pha
        lda     OPRND1
        pha
        lda     OPRND2+1
        pha
        lda     OPRND2
        pha
        jsr     SREM16                  ; Rest
        pla
        sta     REM
        pla
        sta     REM+1
        brk                     ; Rest van -1023 / 123 = -39
                                ;    REM     = D9 hexadecimaal
                                ;    REM + 1 = FF hexadecimaal

; Rest van deling zonder teken, OPRND1 / OPRND2, bewaar de rest
; in REM

        lda     OPRND1+1
        pha
        lda     OPRND1
        pha
        lda     OPRND2+1
```

```
                pha
                lda     OPRND2
                pha
                jsr     UREM16              ; Rest
                pla
                sta     REM
                pla
                sta     REM+1
                brk                         ; Rest van 64513 / 123 = 61
                                            ;    REM   = 3D hexadecimaal
                                            ;    REM+1 = 00 hexadecimaal

                jmp SC0604

; Data
OPRND1:         .WORD   -1023               ; Teller (64513 zonder teken)
OPRND2:         .WORD   123                 ; Noemer
QUOT:           .BLOCK  2                   ; Quotient
REM:            .BLOCK  2                   ; Rest

                .END
```

16-Bits Vergelijking - 6E

Vergelijkt twee van de stack verkregen 16-bits getallen en zet de vlaggen hiernaar. Alle 16-bits getallen worden op de voor de 6502 gebruikelijke wijze opgeslagen met de laagstwaardige byte boven de hoogstwaardige byte. De vergelijking wordt verricht door het bovenste getal (of aftrekker) van het onderste getal (of aftrektal) af te trekken. De nulvlag geeft altijd aan of de getallen gelijk zijn. Als de getallen ongetekend zijn, geeft de overdrachtvlag aan welk getal groter is (overdrachtvlag = 0 als bovenste getal of aftrekker groter is. Anders is de overdrachtvlag 1). Als de getallen getekend zijn, geeft de negatiefvlag aan welke groter is (negatiefvlag = 1 als het bovenste getal of de aftrekker groter is en 0 in het andere geval); met een tweecomplement-overloop is rekening gehouden. De negatiefvlag wordt geinverteerd als dit zich voordoet.
Procedure: Het programma vergelijkt eerst de laagstwaardige bytes van de aftrekker en het aftrektal. Vervolgens trekt het de hoogstwaardige byte van de aftrekker af van de hoogstwaardige byte van het aftrektal. Hiermee worden de vlaggen gezet. Als de laagstwaardige bytes van de getallen niet gelijk zijn, wist het programma de nulvlag door een logische OR van de accumulator met 01_{16}. Als de aftrekking tot een tweecomplement-overloop leidt, complementeert het programma de negatiefvlag door een logische Exclusieve OR van de accumulator met 80_{16} (10000000_2); eveneens wist het de nulvlag op de eerder beschreven methode.

Gebruikte registers:	A, P
Looptijd:	Ongeveer 90 cycli
Programmalengte:	65 bytes
Benodigd datageheugen:	Zes bytes ergens in het geheugen voor het aftrektal of WORD1 (twee bytes beginnende bij adres MINEND), de aftrekker of WORD2 (twee bytes beginnende bij adres SUBTRA) en het terugkeeradres (twee bytes beginnende bij adres RETADR).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van de aftrekker (bovenste getal of WORD2)
Hoogstwaardige byte van de aftrekker (bovenste getal of WORD2)
Laagstwaardige byte van het aftrektal (onderste getal of WORD1)
Hoogstwaardige byte van het aftrektal (onderste getal of WORD1)

Eindcondities

Vlaggen gezet alsof de aftrekker van het aftrektal gesubtraheerd is, met een aanpassing als er een tweecomplement-overloop opgetreden is.

Nulvlag = 1 als aftrekker en aftrektal gelijk zijn, 0 indien dit niet zo is.

Overdrachtvlag = 0 als de aftrekker in de ongetekende zin groter is dan het aftrektal, 1 als hij kleiner dan of gelijk is aan het aftrektal.

Negatiefvlag = 1 als de aftrekker in de getekende zin groter is dan het aftrektal, 0 als hij kleiner dan of gelijk is aan het aftrektal. De stand van de vlag wordt gecorrigeerd als er een tweecomplement-overloop optreedt.

Voorbeelden

1. Data: Aftrektal (onderste getal) = $03E1_{16}$
   Aftrekker (bovenste getal) = $07E4_{16}$

   Resultaat: Overdrachtvlag = 0, aangevende dat de aftrekker in ongetekende zin groter is.
   Nulvlag = 0, aangevende dat de getallen niet gelijk zijn
   Negatiefvlag = 1, aangevende dat de aftrekker in de getekende zin groter is.

2. Data: Aftrektal (onderste getal) = $C51A_{16}$
   Aftrekker (bovenste getal) = $C51A_{16}$

   Resultaat: Overdrachtvlag = 1, aangevende dat de aftrekker in ongetekende zin niet groter is
   Nulvlag = 1, aangevende dat de beide getallen gelijk zijn
   Negatiefvlag = 0, aangevende dat de aftrekker in getekende zin niet groter is

3. Data: Aftrektal (onderste getal) = $A45D_{16}$
   Aftrekker (bovenste getal) = $77E1_{16}$

   Resultaat: Overdracht = 1, aangevende dat de aftrekker in ongetekende zin niet groter is
   Nulvlag = 0, aangevende dat beide getallen niet gelijk zijn
   Negatiefvlag = 1, aangevende dat de aftrekker in getekende zin groter is.

In voorbeeld 3 is het onderste getal een negatief tweecomplement, en het onderste een positief tweecomplement-getal is.

```
;       Titel      : 16-bits vergelijken
;       Naam       : CMP16
;
;
;       Doel       : Vergelijk twee 16-bits woorden met of zonder teken en
;                    geef in de vlaggen C, Z, N aan wat het resultaat is
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Laagstwaardige byte van woord 2
;                      Hoogstwaardige byte van woord 2
;                      Laagstwaardige byte van woord 1
```

```
;                    Hoogstwaardige byte van woord 1
;
;     Eind       : De vlaggen zijn gezet naar de volgende regels:
;                   IF woord1 en woord2 2-complement getallen zijn
;                     THEN
;                        IF woord1 = woord2 THEN
;                            Z=1, N=0
;                        IF woord1 > woord2 THEN
;                            Z=0, N=0
;                        IF woord1 < woord2 THEN
;                            Z=0, N=1
;                     ELSE
;                        IF woord1 = woord2 THEN
;                            Z=1, C=1
;                        IF woord1 > woord2 THEN
;                            Z=0, C=1
;                        IF woord1 < woord2 THEN
;                            Z=0, C=0
;
;     Registers : A, P
;
;     Tijd      : Ongeveer 90 cycli
;
;     Grootte   : programma 65  bytes
;                 data       6  bytes

CMP16:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal woord 2 van de stack

                    pla
                    sta     SUBTRA
                    pla
                    sta     SUBTRA+1

          ; Haal woord 1 van de stack

                    pla
                    sta     MINEND
                    pla
                    sta     MINEND+1

          ; Zet het terugkeeradres weer op de stack

                    lda     RETADR+1
                    pha
                    lda     RETADR
                    pha

                    lda     MINEND
                    cmp     SUBTRA        ; Vergelijk de laagstwaardige bytes
                    beq     EQUAL         ; Spring als ze gelijk zijn

          ; De laagstwaardige bytes zijn niet gelijk - vergelijk dus nu de
          ; hoogstwaardige bytes

                    lda     MINEND+1
```

```
                sbc     SUBTRA+1        ; Vergelijk de hoogstwaardige bytes
                ora     #0              ; Maak Z = 0 want de laagstwaardige
                                        ; bytes zijn niet gelijk
                bvs     OVFLOW          ; We moeten testen op overloop
                                        ; omdat we te maken heben met
                                        ; bewerkingen van getallen met
                                        ; teken
                rts                     ; Keer terug

        ; De laagstwaardige bytes zijn gelijk - Vergelijk de
        ; hoogstwaardige bytes

EQUAL:
                lda     MINEND+1
                sbc     SUBTRA+1        ; De hoogstwaardige bytes
                bvs     OVFLOW          ; We werken nog steeds met getallen
                                        ; met tekens
                rts                     ; Het is gebeurd. Keer terug met
                                        ; het resultaat in P

        ; Overloop door bewerkingen met getallen met een teken
        ; We klappen de negatiefvlag om zonder de overschotvlag te
        ; veranderen door een ex-or van A met 80 hexadecimaal en maken
        ; de nulvlag gelijk aan 0
OVFLOW:
                eor     #80H
                ora     #1              ; Als er overloop is dan zijn de
                                        ; woorden niet gelijk
                                        ; Maak dus Z = 0 en laat de
                                        ; overschotvlag zoals die was
                rts

; Data
MINEND:         .BLOCK  2               ; Woord 1
SUBTRA:         .BLOCK  2               ; Woord 2
RETADR:         .BLOCK  2               ; Het terugkeeradres

;
; Voorbeeld uitvoering
;
;

SC0605:
        ; Vergelijk OPRND1 met OPRND2

                lda     OPRND1+1
                pha
                lda     OPRND1
                pha
                lda     OPRND2+1
                pha
                lda     OPRND2
                pha
                jsr     CMP16
                brk                     ; Bekijk de vlaggen eens voor
                                        ;     123 en 1023:
                                        ;   C = 0, Z = 0, N = 1
                jmp     SC0605
```

```
; Data
OPRND1:         .WORD   123         ; Test woord 1
OPRND2:         .WORD   1023        ; Test woord 2


                .END
```

Meervoudige precisie binaire optelling - 6F

Telt twee, uit meerdere bytes bestaande ongetekende getallen bij elkaar op. Beide getallen worden met hun laagstwaardige bytes eerst opgeslagen (op het laagste adres). De som vervangt een van de getallen (degene met de lagere plaats op de stack als beginadres). De lengte van de getallen in bytes is 255 of minder.
Procedure: Het programma wist de overdrachtvlag eerst en telt dan de getallen byte voor byte bij elkaar op, beginnende bij de laagstwaardige byte. De uiteindelijke overdrachtvlag geeft de optelling van de hoogstwaardige bytes weer. De som vervangt het getal met de lagere plaats in de stack (array 1 in de listing). Een lengte van 0 brengt een onmiddellijke terugkeer zonder optellingen teweeg.

Gebruikte registers:	Alle
Looptijd:	23 cycli per byte plus 82 cycli vast. Het optellen van twee 6-bytes getallen, bij voorbeeld, duurt 23 x 6 + 82 = 220 cycli.
Programmalengte:	48 bytes
Benodigd datageheugen:	Twee bytes ergens in RAM plus vier bytes op pagina 0. De twee bytes ergens in RAM dienen als tijdelijke opslag voor het terugkeeradres (beginnende bij adres RETADR). De vier bytes op pagina 0 bevatten pointers naar de getallen (beginnende op de adressen AY1PTR, respectievelijk AY2PTR). In de listing worden AY1PTR als adres $00D0_{16}$ en AY2PTR als adres $00D2_{16}$ genomen.
Speciaal geval:	Het programma keert terug als de getallen een lengte van nul hebben. De som is dan gelijk aan het onderste getal (m.a.w. array 1 is ongewijzigd). De overdrachtvlag wordt op 1 gezet.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Lengte van de getallen in bytes
Laagstwaardige byte van het beginadres van het tweede getal (het adres dat de laagstwaardige byte van array 2 bevat)
Hoogstwaardige byte van het beginadres van het tweede getal (het adres dat de laagstwaardige byte van array 2 bevat)
Laagstwaardige byte van het beginadres van het eerste getal en het resultaat (het adres dat de laagstwaardige byte van array 1 bevat)
Hoogstwaardige byte van het beginadres van het eerste getal en het resultaat (het adres dat de laagstwaardige byte van array 1 bevat)

Eindcondities

Het eerste getal (array 1) vervangen door de som van het eerste getal (array 1) en het tweede getal (array 2).

Voorbeelden

Data: Lengte van de getallen (in bytes) = 6
Bovenste getal (array 2) = $19D028A193EA_{16}$
Onderste getal (array 1) = $293EABF059C7_{16}$

Resultaat: Onderste getal (array 1) = Onderste getal (array 1) + bovenste getal (array 2) = $430ED491EDB1_{16}$.

Overdrachtvlag = 0

```
;       Titel      : Multi-precies binair optellen
;       Naam       : MPBADD
;
;
;       Doel       : Tel twee arrays binaire data bij elkaar op
;                       Array1 := Array1 + Array2
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Lengte van het array in bytes
;                      Laagstwaardige byte van het beginadres van array 2
;                      Hoogstwaardige byte van het beginadres van array 2
;                      Laagstwaardige byte van het beginadres van array 1
;                      Hoogstwaardige byte van het beginadres van array 1
;
;                      De arrays bestaan uit binaire nummers zonder teken met
;                      een maximale lengte van 255 bytes, ARRAY[0] is de
;                      laagstwaardige byte van en ARRAY[LENGTE-1] is de
;                      hoogstwaardige byte
;
;       Eind       : Array1 := Array1 + Array2
;
;       Registers  : Alle
;
;       Tijd       : 23 cycli per byte plus 82 basiscycli
;
;       Grootte    : programma  48  bytes
;                    data        2  bytes plus
;                                4  bytes op pagina 0

; Wijzers naar adressen op pagina 0

AY1PTR:             .EQU    0D0H          ; Wijzer naar array 1
AY2PTR:             .EQU    0D2H          ; Wijzer naar array 2


MPBADD:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal de lengte van de array's van de stack
```

```
                pla
                tax

        ; Haal het beginadres van array 2 van de stack

                pla
                sta     AY2PTR
                pla
                sta     AY2PTR+1

        ; Haal het beginadres van array 1 van de stack

                pla
                sta     AY1PTR
                pla
                sta     AY1PTR+1

        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha

        ; Initialiseer

                ldy     #0
                cpx     #0              ; Is de lengte 0 ?
                beq     EXIT            ; Zo ja, spring naar EXIT
                clc                     ; Maak de overdrachtvlag schoon

LOOP:
                lda     (AY1PTR),Y      ; Haal de volgende byte op
                adc     (AY2PTR),Y      ; Tel ze bij elkaar op
                sta     (AY1PTR),Y      ; Zet het resultaat in array 1
                iny                     ; Verhoog de array-index
                dex                     ; Verlaag de lengteteller
                bne     LOOP            ; Spring totdat alle bytes opgeteld
                                        ; zijn
EXIT:
                rts


; Data
RETADR:         .BLOCK  2               ; Het terugkeeradres


;
;
; Voorbeeld uitvoering
;
;

SC0606:
                lda     AY1ADR+1
                pha
                lda     AY1ADR
                pha                     ; Beginadres van array 1

                lda     AY2ADR+1
                pha
                lda     AY2ADR
```

```
                pha                     ; Beginadres van array 2

                lda     #SZAYS
                pha                     ; De lengte van de arrays
                jsr     MPBADD          ; Tel de arrays bij elkaar op
                brk                     ; Resultaat van
                                        ;   1234567H + 1234567H =
                                        ;      2468ACEH
                                        ;    AY1        = CEH
                                        ;    AY1+1      = 8AH
                                        ;    AY1+2      = 46H
                                        ;    AY1+3      = 02H
                                        ;    AY1+4      = 00H
                                        ;    AY1+5      = 00H
                                        ;    AY1+6      = 00H

                jmp     SC0606


; Data

SZAYS:          .EQU    7               ; Lengte van de arrays

AY1ADR:         .WORD   AY1             ; Adres van array 1
AY2ADR:         .WORD   AY2             ; Adres van array 2

AY1:
                .BYTE   067H
                .BYTE   045H
                .BYTE   023H
                .BYTE   001H
                .BYTE   0
                .BYTE   0
                .BYTE   0


AY2:
                .BYTE   067H
                .BYTE   045H
                .BYTE   023H
                .BYTE   001H
                .BYTE   0
                .BYTE   0
                .BYTE   0


                .END
```

Meervoudige precisie binaire aftrekking - 6G

Trekt twee, uit meerdere bytes bestaande ongetekende binaire getallen van elkaar af. Beide getallen worden opgeslagen met hun laagstwaardige byte op het laagste adres. Het beginadres van de aftrekker (het getal dat afgetrokken wordt) is opgeslagen boven het beginadres van het aftrektal (het getal waar de aftrekker vanaf getrokken wordt). Het verschil vervangt het aftrektal in het geheugen. De lengte van de getallen is 255 of kleiner.
Procedure: Het programma zet de overdrachtvlag (de geinverteerde leen) eerst en trekt dan byte-voor-byte de aftrekker van het aftrektal af, beginnende bij de laagstwaardige bytes. De uiteindelijke stand van de overdrachtvlag reflecteert de aftrekking van de hoogstwaardige bytes. Het verschil vervangt het aftrektal (het getal met het beginadres lager in de stack, array 1 in de listing). Een lengte 0 veroorzaakt een onmiddellijke terugkeer vanuit het programma, zonder dat er afgetrokken is.

Gebruikte registers:	Alle
Looptijd:	23 cycli per byte plus 82 cycli vast. Het aftrekken van twee 6 bytes grote getallen duurt 23 x 6 + 82 = 220 cycli.
Programmalengte:	48 bytes
Benodigd datageheugen:	Twee bytes ergens in RAM plus vier bytes op pagina 0. De twee bytes ergens in RAM zijn tijdelijke opslag voor het terugkeeradres (beginnende bij adres RETADR). De vier bytes op pagina 0 bevatten pointers naar de twee getallen (beginnende bij de adressen MINPTR, respectievelijk SUBPTR). In de listing zijn MINPTR als $00D0_{16}$ en SUBPTR als $00D2_{16}$ genomen.
Speciaal geval:	Het programma keert onmiddellijk terug zonder enige aftrekking te hebben verricht als de lengte van de getallen 0 is. Het aftrektal blijft ongewijzigd, ofte wel het verschil is gelijk aan het onderste getal. De overdrachtvlag wordt op 1 gezet.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Lengte van de getallen in bytes
Laagstwaardige byte van het beginadres van de aftrekker (adres dat de laagstwaardige byte van array 2 bevat)
Hoogstwaardige byte van het beginadres van de aftrekker (adres dat de laagstwaardige byte van array 2 bevat)
Laagstwaardige byte van het beginadres van het aftrektal (adres dat de laagstwaardige byte van array 1 bevat)
Hoogstwaardige byte van het beginadres van het aftrektal (adres dat de laagstwaardige byte van array 1 bevat)

Eindcondities

Aftrektal vervangen door aftrektal minus aftrekker.

Voorbeelden

Data: Lengte van de getallen (in bytes) = 4
Aftrektal = $2F5BA7C3_{16}$
Aftrekker = $14DF35B8_{16}$

Resultaat: Verschil = $1A7C720B_{16}$

Dit getal vervangt het oorspronkelijke aftrektal in het geheugen. De overdrachtvlag wordt op 1 gezet, in haar voor de 6502 gebruikelijke rol als geinverteerde leen.

```
;       Titel     : Multi-precies binair aftrekken
;       Naam      : MPBSUB
;
;
;       Doel      : Trek twee arrays binaire data van elkaar af
;                       Array1 := Array1 - Array2
;
;
;       Begin     : Top van de stack
;                     Laagstwaardige byte van het terugkeeradres
;                     Hoogstwaardige byte van het terugkeeradres
;                     Lengte van het array in bytes
;                     Laagstwaardige byte van het beginadres van array 2
;                     Hoogstwaardige byte van het beginadres van array 2
;                     Laagstwaardige byte van het beginadres van array 1
;                     Hoogstwaardige byte van het beginadres van array 1
;
;                     De arrays bestaan uit binaire nummers zonder teken met
;                     een maximale lengte van 255 bytes, ARRAY[0] is de
;                     laagstwaardige byte van en ARRAY[LENGTE-1] is de
;                     hoogstwaardige byte
;
;       Eind      : Array1 := Array1 - Array2
;
;       Registers : Alle
;
;       Tijd      : 23 cycli per byte plus 82 basiscycli
;
;       Grootte   : programma  48  bytes
;                   data        2  bytes plus
;                               4  bytes op pagina 0

; Wijzers op pagina 0

MINPTR:             .EQU    0D0H          ; Wijzer naar array 1
SUBPTR:             .EQU    0D2H          ; Wijzer naar array 2


MPBSUB:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal de lengte van de arrays van de stack
```

```
        pla
        tax

; Haal het beginadres van array 2 van de stack

        pla
        sta     SUBPTR
        pla
        sta     SUBPTR+1

; Haal het beginadres van array 1 van de stack

        pla
        sta     MINPTR
        pla
        sta     MINPTR+1

; Zet het terugkeeradres weer op de stack

        lda     RETADR+1
        pha
        lda     RETADR
        pha

; Initialiseer

        ldy     #0
        cpx     #0              ; Is de lengte 0?
        beq     EXIT            ; Zo ja, spring naar EXIT
        clc                     ; Maak de overdrachtvlag schoon

LOOP:
        lda     (MINPTR),Y      ; Haal de volgende byte op
        sbc     (SUBPTR),Y      ; Trek ze van elkaar af
        sta     (MINPTR),Y      ; Zet het resultaat in array 1
        iny                     ; Verhoog de array-index
        dex                     ; Verlaag de lengteteller
        bne     LOOP            ; Spring totdat alle bytes
                                ; afgetrokken zijn
EXIT:
        rts

; Data
RETADR:         .BLOCK  2       ; Het terugkeeradres

;
;
; Voorbeeld uitvoering
;
;

SC0607:
        lda     AY1ADR+1
        pha
        lda     AY1ADR
        pha                     ; Beginadres van array 1

        lda     AY2ADR+1
        pha
        lda     AY2ADR
```

```
                pha                     ; Beginadres van array 2

                lda     #SZAYS
                pha                     ; De lengte van de arrays
                jsr     MPBSUB          ; Trek de twee arrays van elkaar af
                brk                     ; Resultaat van
                                        ;   7654321H - 7654321H =
                                        ;       641FDBAH
                                        ;    AY1          = 0BAH
                                        ;    AY1+1        = 0FDH
                                        ;    AY1+2        = 41H
                                        ;    AY1+3        = 06H
                                        ;    AY1+4        = 00H
                                        ;    AY1+5        = 00H
                                        ;    AY1+6        = 00H

                jmp     SC0607


; Data

SZAYS:          .EQU    7               ; Lengte van de arrays

AY1ADR:         .WORD   AY1             ; Adres van array 1
AY2ADR:         .WORD   AY2             ; Adres van array 2

AY1:
                .BYTE   021H
                .BYTE   043H
                .BYTE   065H
                .BYTE   007H
                .BYTE   0
                .BYTE   0
                .BYTE   0

AY2:
                .BYTE   021H
                .BYTE   043H
                .BYTE   065H
                .BYTE   007H
                .BYTE   0
                .BYTE   0
                .BYTE   0

                .END
```

Meervoudige precisie binaire vermenigvuldiging - 6H

Vermenigvuldigt twee, uit meerdere bytes bestaande ongetekende binaire getallen. Beide getallen worden met hun laagstwaardige bytes op het laagste adres opgeslagen. Het produkt vervangt een van de getallen (degene met het beginadres lager in de stack). De lengte van de getallen (in bytes) is 255 of minder. Alleen de laagstwaardige bytes van het produkt worden teruggegeven om compatibiliteit met andere meervoudige precisie binaire handelingen te houden.
Procedure: Het programma gebruikt een gewoon tel-op-en-schuif algoritme, waarbij het vermenigvuldigtal (array 2) telkens bij het deelprodukt wordt opgeteld als het een 1 in de vermenigvuldiger (array 1) vindt. Het deelprodukt en de vermenigvuldiger worden door de bit-lengte plus 1 geschoven. De extra lus is nodig om de uiteindelijke overdrachtvlag in het produkt te brengen. Het programma bewaart een volledig ongetekend deelprodukt in de geheugenlocaties beginnende bij HIPROD (hoogstwaardige bytes) en in array 1 (laagstwaardige bytes). De laagstwaardige bytes van het produkt vervangen de vermenigvuldiger wanneer de vermenigvuldiger is geschoven en bekeken op 1-bits. Een lengte van 00 zorgt voor een onmiddellijke terugkeer zonder vermenigvuldiging.

Gebruikte registers: Alle

Looptijd: Hangt af van de lengte van de handelingen en het aantal 1-bits in de vermenigvuldiger (waarbij optellingen nodig zijn). Als het gemiddelde aantal 1-bits per byte vier is, is de looptijd ongeveer

$$316 \times \text{LENGTE}^2 + 223 \times \text{LENGTE} + 150$$

cycli, waarbij LENGTE het aantal bytes van de getallen voorstelt. Als, bij voorbeeld, LENGTE = 4, is de looptijd ongeveer

$$316 \times 4^2 + 223 \times 4 + 150 = 316 \times 16 + 892 + 150 = 5056 + 1042 = 6.098 \text{ cycli.}$$

Programmalengte: 145 bytes

Benodigd datageheugen: 260 bytes ergens in RAM plus vier bytes op pagina 0. De 260 bytes ergens in RAM dienen als tijdelijke opslag voor de hoogstwaardige bytes van het produkt (255 bytes, beginnende bij adres HIPROD), het terugkeeradres (twee bytes beginnende bij adres RETADR), de lusteller (twee bytes beginnende bij adres COUNT), en de lengte van de getallen in bytes (een byte beginnende bij adres LENGTH). De vier bytes op pagina 0 bevatten pointers naar de twee getallen (de pointers beginnen bij de adressen AY1PTR, respectievelijk AY2PTR). In de listing worden AY1PTR en AY2PTR genomen als de adressen $00D0_{16}$, respectievelijk $00D2_{16}$.

Speciaal geval: Een lengte van nul veroorzaakt een onmiddellijke terugkeer, waarbij het produkt gelijk is aan de oorspronkelijke vermenigvuldiger (ofte wel, array 1 is ongewijzigd). De hoogstwaardige bytes van het produkt (beginnende bij adres HIPROD) zijn niet gedefinieerd.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Lengte van de getallen in bytes
Laagstwaardige byte van het beginadres van het vermenigvuldigtal (het adres dat de laagstwaardige byte van array 2 bevat)
Hoogstwaardige byte van het beginadres van het vermenigvuldigtal (het adres dat de laagstwaardige byte van array 2 bevat)
Laagstwaardige byte van het beginadres van de vermenigvuldiger (het adres dat de laagstwaardige byte van array 1 bevat)
Hoogstwaardige byte van het beginadres van de vermenigvuldiger (het adres dat de laagstwaardige byte van array 1 bevat)

Eindcondities

Vermenigvuldiger (array 1) vervangen door het produkt van vermenigvuldiger (array 1) maal vermenigvuldigtal (array 2).

Voorbeelden

Data:	Lengte van de getallen (in bytes) = 4 Bovenste getal (array 2 of vermenigvuldigtal) = $0005D1F7_{16}$ = 381.431_{10} Onderste getal (array 1 of vermenigvuldiger) = $00000AB1_{16} = 2.737_{10}$
Resultaat:	Onderste getal (array 1) = Onderste getal (array 1) * Bovenste getal (array 2) = $3E39D1C7_{16} = 1.043.976.647_{10}$.

Let op dat dit programma slechts de laagstwaardige bytes (m.a.w. het aantal bytes in het vermenigvuldigtal en de vermenigvuldiger) van het produkt retourneert. Dit om compatibiliteit met andere meervoudige precisie binaire rekenkundige handelingen te houden. De hoogstwaardige bytes van het produkt zijn beschikbaar met hun laagstwaardige bytes op adres HIPROD. De gebruiker zal deze bytes moeten controleren op een mogelijk overschot of de getallen met extra nullen moeten verlengen.

```
;       Titel      : Multi-precies binair vermenigvuldigen
;       Naam       ; MPBMUL
;
;
;       Doel       : Vermenigvuldig 2 arrays binaire bytes met elkaar
;                        Array1 := Array1 * Array2
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
```

```
;                   Lengte van het array in bytes
;                   Laagstwaardige byte van het beginadres van array 2
;                   Hoogstwaardige byte van het beginadres van array 2
;                   Laagstwaardige byte van het beginadres van array 1
;                   Hoogstwaardige byte van het beginadres van array 1
;
;                   De arrays bestaan uit binaire nummers zonder teken met
;                   een maximale lengte van 255 bytes, ARRAY[0] is de
;                   laagstwaardige byte, en ARRAY[LENGTE-1] is de
;                   hoogstwaardige byte
;
;     Eind      : Array1 := Array1 * Array2
;
;     Registers : Alle
;
;     Tijd      : Aangenomen dat de gemiddelde aantal 1-bits in array 1 is
;                 4 * de lengte, dan is de benodigde tijd ongeveer
;                 (316 * de lengte tot de tweede macht) + (223 * de lengte)
;                 + 150 basiscycli
;
;     Grootte   : programma  145  bytes
;                 data       260  bytes plus
;                              4  bytes op pagina 0

; Wijzers naar adressen op pagina 0

AY1PTR:             .EQU    0D0H        ; Wijzer naar array 1
AY2PTR:             .EQU    0D2H        ; Wijzer naar array 2


MPBMUL:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal de lengte van de arrays van de stack

                    pla
                    sta     LENGTH

          ; Haal het beginadres van array 2 van de stack en trek er 1 van
          ; af zodat de index bij 1 kan beginnen in plaats van 0


                    pla
                    sec
                    sbc     #1          ; Trek er een vanaf
                    sta     AY2PTR
                    pla
                    sbc     #0          ; Neem de overdrachtvlag mee
                    sta     AY2PTR+1

          ; Haal het beginadres van array 1 van de stack en trek er 1 van
          ; af zodat de index bij 1 kan beginnen in plaats van 0

                    pla
                    sec
                    sbc     #1          ; Trek er een van af
                    sta     AY1PTR
```

```
        pla
        sbc     #0              ; Neem de overdrachtvlag mee
        sta     AY1PTR+1

; Zet het terugkeeradres weer op de stack

        lda     RETADR+1
        pha
        lda     RETADR
        pha

; Er hoeft niets gedaan te worden als de lengte 0 is

        lda     LENGTH          ; Is de lengte 0 ?
        beq     EXIT            ; Zo ja, spring naar EXIT

; Maak COUNT gelijk aan het aantal bits in het array + 1
;   COUNT := (Lengte * 8) + 1

        sta     COUNT           ; Initialiseer de lengte op 0
        lda     #0
        asl     COUNT           ; COUNT * 2
        rol     A               ; A bevat de hoogstwaardige byte
        asl     COUNT           ; COUNT * 4
        rol     A
        asl     COUNT           ; COUNT * 8
        rol     A
        sta     COUNT+1         ; Bewaar de hoogstwaardige byte
        inc     COUNT           ; Verhoog de laagstwaardige byte
        bne     ZEROPD
        inc     COUNT+1         ; Verhoog de hoogstwaardige byte

; Maak de hoogstwaardige byte van het array gelijk aan nul

ZEROPD:
        ldx     LENGTH
        lda     #0

ZEROLP:
        sta     HIPROD-1,X      ; De min 1 om de index van 1 tot
                                ; en met LENGTH te laten lopen
        dex
        bne     ZEROLP

; Voer de vermenigvuldiging uit met behulp van het optel en
; schuif algoritme
; ARRAY 1 is de multiplicator en ARRAY 2 de multiplicant

        clc                     ; Maak de overdrachtvlag schoon
                                ; voordat de lus wordt
                                ; binnengestapt
LOOP:
; Neem de overdrachtvlag mee in het bovenste array produkt en zet
; de laagstwaardige byte van het bovenste array produkt in de
; overdrachtvlag

        ldx     LENGTH

SRPLP:
        ror     HIPROD-1,X      ; Min 1 om de index van 1 tot en
                                ; met LENGTH te laten lopen
```

```
            dex
            bne     SRPLP           ; Ga door tot de index 0 is

        ; Neem de overdrachtvlag, het volgende bit van het kleinste
        ; product, op in de hoogstwaardige bit van ARRAY 1.
        ; Dit is het volgende bit van de vermenigvuldiging.
        ; Dit zorgt er meteen voor dat het volgende bit van de
        ; multiplicator de nieuwe overdrachtvlag wordt

            ldy     LENGTH

SRA1LP:
            lda     (AY1PTR),Y
            ror     A               ; Roteer de volgende byte
            sta     (AY1PTR),Y
            dey
            bne     SRA1LP          ; Ga door tot de index 0 is

        ; Tel het bovenste array produkt bij ARRAY 2 op als het volgende
        ; bit van de multiplicator 1 is

            bcc     DECCNT          ; Spring als het volgende bit 0 is

        ; Tel ARRAY 2 en HIPROD bij elkaar op

            ldy     #1
            ldx     LENGTH
            clc


ADDLP:
            lda     (AY2PTR),Y
            adc     HIPROD-1,Y      ; Tel de bytes op
            sta     HIPROD-1,Y
            iny                     ; Verhoog de index
            dex                     ; Verlaag de teller
            bne     ADDLP           ; Ga door tot de teller 0 is


        ; Verlaag de bit-teller en verlaat de routine als we alle bits
        ; hebben gehad
        ; De overdrachtvlag moet gelijk blijven!
DECCNT:
            dec     COUNT           ; Verlaag laagstwaardige byte van
                                    ; de teller
            bne     LOOP            ; Ga door zolang er nog bytes
                                    ; verwerkt moeten worden
            ldx     COUNT+1         ; Haal de hoogstwaardige byte op
            beq     EXIT            ; Verlaat de routine als COUNT 0 is

            dex                     ; Verlaag anders hoogstwaardige
                                    ; byte van de teller

            stx     COUNT+1
            jmp     LOOP


EXIT:
            rts


; Data
```

```
RETADR:         .BLOCK  2       ; Het terugkeeradres
COUNT:          .BLOCK  2       ; De cyclusteller
LENGTH:         .BLOCK  1       ; De lengte van de arrays
HIPROD:         .BLOCK  255     ; Buffer voor hoogstwaardige
                                ; produkt


;
;
; Voorbeeld uitvoering
;
;

SC0608:
                lda     AY1ADR+1
                pha
                lda     AY1ADR
                pha             ; Zet het adres van ARRAY 1 op de
                                ; stack

                lda     AY2ADR+1
                pha
                lda     AY2ADR
                pha             ; Zet het adres van ARRAY 2 op de
                                ; stack

                lda     #SZAYS
                pha             ; Zet de lengte op de stack
                jsr     MPBMUL  ; Voer de vermenigvuldiging uit
                brk             ; Het resultaat van
                                ;   12345 hexadecimaal
                                ;         maal
                                ;   1234 hexadecimaal =
                                ;       14B604040 hexadecimaal
                                ;  In het geheugen:
                                ;    AY1     = 04H
                                ;    AY1+1   = 04H
                                ;    AY1+2   = B6H
                                ;    AY1+3   = 14H
                                ;    AY1+4   = 00H
                                ;    AY1+5   = 00H
                                ;    AY1+6   = 00H

                jmp     SC0608

SZAYS:          .EQU    7       ; De lengte van de arrays
AY1ADR:         .WORD   AY1     ; Het adres van ARRAY 1
AY2ADR:         .WORD   AY2     ; Het adres van ARRAY 2


AY1:
                .BYTE   045H
                .BYTE   023H
                .BYTE   001H
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0
```

```
AY2:
                .BYTE   034H
                .BYTE   012H
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0

                .END
```

Meervoudige precisie binaire deling - 6I

Deelt twee, uit meerdere bytes bestaande, ongetekende binaire getallen. Beide getallen worden met hun laagstwaardige bytes op het laagste adres opgeslagen. Het quotient vervangt het deeltal (het getal met het beginadres lager in de stack). De lengte van de getallen (in bytes) is 255 of minder. Het restgetal wordt niet teruggegeven, maar het beginadres ervan (laagstwaardige byte) is beschikbaar in de geheugenlocaties HDEPTR en HDEPTR+1. De overschotvlag wordt gewist als er geen fouten optraden; als er een poging wordt gedaan door 0 te delen, wordt de overschotvlag op 1 gezet, blijft het deeltal ongewijzigd en wordt het restgetal op nul gezet.
Procedure: Het programma voert de deling met behulp van het gebruikelijke schuif-en-trek-af algoritme uit. Het quotient en deeltal worden geschoven, waarbij telkens een 1-bit in het quotient wordt gezet als een proefaftrekking succesvol is. Een extra buffer wordt gebruikt om het resultaat van de proefaftrekking in op te slaan. Deze buffer wordt eenvoudigweg omgewisseld met de buffer die het deeltal bevat, als de proefaftrekking succesvol is. Het programma stopt onmiddellijk, waarbij de overschotvlag wordt gezet, als de deler nul blijkt te zijn. De overschotvlag wordt anders gewist.

Gebruikte registers: Alle

Looptijd: Hangt af van de lengte van de getallen en van het aantal 1-bits in het quotient (waarbij een bufferwisseling nodig is). Als het aantal 1-bits in het quotient gemiddeld vier per byte is, is de looptijd ongeveer

$$480 \times LENGTE^2 + 438 \times LENGTE + 208$$

cycli, waarbij LENGTE het aantal bytes in de getallen is. Als, bij voorbeeld, LENGTE = 4 (32-bits deling), is de looptijd ongeveer

$$480 \times 4^2 + 438 \times 4 + 208 = 480 \times 16 + 1752 + 208 = 7680 + 1960 = 9.640 \text{ cycli}$$

Programmalengte: 206 bytes

Benodigd datageheugen: 519 bytes ergens in RAM plus acht bytes op pagina 0. De 519 ergens in RAM dienen als tijdelijke opslag voor het hoge deeltal (255 bytes beginnende bij adres HIDE1), het resultaat van de proefaftrekking (255 bytes beginnende bij adres HIDE2), het terugkeeradres (twee bytes beginnende bij RETADR), de lusteller (twee bytes beginnende bij adres COUNT), de lengte van de getallen (een byte op adres LENGTH) en de adressen van de hoog-deeltal-buffers (twee bytes beginnende bij adres AHIDE1 en twee bytes beginnende bij adres AHIDE2). De acht bytes op pagina 0 bevatten pointers naar de twee getallen en naar de twee tijdelijke buffers voor het hoge deeltal. De pointers beginnen op de adressen AY1PTR ($00D0_{16}$ in de listing), HDEPTR ($00D4_{16}$ in de listing) en ODEPTR ($00D6_{16}$ in de listing). HDEPTR bevat na het uitvoeren van de routine het adres van de laagstwaardige byte van het restgetal.

Speciale gevallen:

1. Een lengte van nul veroorzaakt een onmiddellijke terugkeer vanuit het programma, waarbij de overschotvlag is gewist, het quotient gelijk is aan het oorspronkelijke deeltal en het restgetal niet is gevormd.
2. Een deler van 0 veroorzaakt een onmiddellijke terugkeer vanuit het programma, waarbij de overschotvlag op 1 is gezet, het quotient gelijk is aan het oorspronkelijke deeltal en het restgetal gelijk is aan nul.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Lengte van de getallen in bytes
Laagstwaardige byte van het beginadres van de deler (adres bevattende de laagstwaardige byte van array 2)
Hoogstwaardige byte van het beginadres van de deler (adres bevattende de laagstwaardige byte van array 2)
Laagstwaardige byte van het beginadres van het deeltal (adres bevattende de laagstwaardige byte van array 1)
Hoogstwaardige byte van het beginadres van het deeltal (adres bevattende de laagstwaardige byte van array 1)

Eindcondities

Deeltal (array 1) vervangen door het quotient van deeltal (array 1) gedeeld door deler (array 2).
Als de deler ongelijk aan nul is, is de overschotvlag 0 en het resultaat normaal.
Als de deler nul is, is de overschotvlag 1, het deeltal ongewijzigd en het restgetal nul.
Het restgetal is beschikbaar met de laagstwaardige byte opgeslagen in het adres dat wordt aangegeven door HDEPTR en HDEPTR+1.

Voorbeeld

Data:
Lengte van de getallen (in bytes) = 03
Bovenste getal (array 2 of deler) = $000F45_{16} = 3.909_{10}$
Onderste getal (array 1 of deeltal) = $35A2F7_{16} = 3.515.127_{10}$

Resultaat:
Onderste getal (array 1) = Onderste getal (array 1) / Bovenste getal (array 2) = $000383_{16} = 899_{10}$
Restgetal (beginnende op het adres dat in de locaties HDEPTR en HDEPTR+1 staat) = $0003A8_{16} = 936_{10}$.
Overschotvlag = 0 om aan te geven dat er geen deling door nul fout optrad.

```
;       Titel      : Multi-precies binair delen
;       Naam       : MPBDIV
;
;
;       Doel       : Deel 2 arrays binaire bytes op elkaar
;                       Array1 := Array1 / Array2
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Lengte van het array in bytes
;                      Laagstwaardige byte van het beginadres van array 2
;                      Hoogstwaardige byte van het beginadres van array 2
;                      Laagstwaardige byte van het beginadres van array 1
;                      Hoogstwaardige byte van het beginadres van array 1
;
;                      De arrays bestaan uit binaire getallen zonder teken met
;                      een maximale lengte van 255 bytes, ARRAY[0] is de
;                      laagstwaardige byte van en ARRAY[LENGTE-1] is de
;                      hoogstwaardige byte
;
;       Eind       : Array1 := Array1 / Array2
;
;       Registers  : Alle
;
;       Tijd       : Aangenomen dat er lengte / 2 bits op 1 staan in het
;                    quotient dan is de benodigde tijd ongeveer
;                    (480 * de lengte tot de tweede macht) + (438 * de lengte)
;                    + 209 basiscycli
;
;       Grootte    : programma  206  bytes
;                    data       519  bytes plus
;                                 8  bytes op pagina 0

; Wijzers naar adressen op pagina 0

AY1PTR:             .EQU    0D0H        ; Wijzer naar array 1
AY2PTR:             .EQU    0D2H        ; Wijzer naar array 2
HDEPTR:             .EQU    0D4H        ; Wijzer naar hoogstwaardige
                                        ; deeltal
ODEPTR:             .EQU    0D6H        ; Wijzer naar het andere
                                        ; hoogstwaardige deeltal



MPBDIV:
          ; Bewaar het terugkeeradres

                    pla
                    sta     RETADR
                    pla
                    sta     RETADR+1

          ; Haal de lengte van de arrays van de stack


                    pla
                    sta     LENGTH

          ; Haal het beginadres van array 2 van de stack
```

```
        pla
        sta     AY2PTR
        pla
        sta     AY2PTR+1

; Haal het beginadres van array 1 van de stack

        pla
        sta     AY1PTR
        pla
        sta     AY1PTR+1

; Zet het terugkeeradres weer op de stack

        lda     RETADR+1
        pha
        lda     RETADR
        pha

; Er hoeft niets gedaan te worden als de lengte 0 is

        lda     LENGTH          ; Is de lengte 0 ?
        bne     INIT
        beq     OKEXIT          ; Zo ja, spring naar OKEXIT

; Maak COUNT gelijk aan het aantal bits in het array + 1
;   COUNT := (Lengte * 8) + 1
INIT:
        sta     COUNT           ; Initialiseer de lengte op 0
        lda     #0
        asl     COUNT           ; COUNT * 2
        rol     A               ; A bevat hoogstwaardige byte
        asl     COUNT           ; COUNT * 4
        rol     A
        asl     COUNT           ; COUNT * 8
        rol     A
        sta     COUNT+1         ; Bewaar hoogstwaardige byte
        inc     COUNT           ; Verhoog laagstwaardige byte
        bne     ZEROPD
        inc     COUNT+1         ; Verhoog hoogstwaardige byte

; Maak de hoogstwaardige byte van het array gelijk aan nul

ZEROPD:
        ldx     LENGTH
        lda     #0

ZEROLP:
        sta     HIDE1-1,X       ; De min 1 om de index van 1 tot
                                ; en met LENGTH te laten lopen
        sta     HIDE2-1,X
        dex
        bne     ZEROLP

; Maak het hoogstwaardige deeltal gelijk aan HIDE1

        lda     AHIDE1
        sta     HDEPTR
        lda     AHIDE1+1
        sta     HDEPTR+1

; Maak het andere hoogstwaardige deeltal gelijk aan HIDE2
```

```
                lda     AHIDE2
                sta     ODEPTR
                lda     AHIDE2+1
                sta     ODEPTR+1

        ; Test of de deler gelijk aan 0 is

                ldx     LENGTH          ; Logische or van alle bits van de
                                        ; deler
                ldy     #0
                tya
CHK0LP:
                ora     (AY2PTR),Y
                iny                     ; Verhoog de index
                dex
                bne     CHK0LP          ; Ga door tot de lengte 0 is
                cmp     #0
                bne     DIV             ; Spring als de deler ongelijk aan
                                        ; 0 is
                jmp     EREXIT          ; Keer terug met foutindicatie

        ; Deel met behulp van het aftrek-algoritme

DIV:
                clc                     ; Maak de overdrachtvlag schoon
                                        ; voor u de lus ingaat
LOOP:
        ; Schuif de overdrachtvlag in het laagstwaardige bit van het
        ; onderste deeltal, als het volgende bit van het quotient en
        ; maak de overdrachtvlag daarna gelijk aan het hoogstwaardige
        ; bit van het onderste deeltal

                ldx     LENGTH
                ldy     #0

SLLP1:
                lda     (AY1PTR),Y
                rol     A               ; Roteer de volgende byte
                sta     (AY1PTR),Y
                iny                     ; Verhoog de index
                dex
                bne     SLLP1           ; Ga door tot de lengte 0 is

        ; Verlaag de bit-teller en spring naar OKEXIT als alles gedaan is
        ; De overdrachtvlag wordt niet veranderd!

DECCNT:
                dec     COUNT           ; Verlaag laagstwaardige byte van
                                        ; de teller
                bne     SLUPR           ; Spring als hij nog niet gelijk
                                        ; aan 0 is
                ldx     COUNT+1         ; Zet hoogstwaardige byte van de
                                        ; teller in X
                beq     OKEXIT          ; Spring als de teller 0 is
                dex                     ; Verlaag hoogstwaardige byte van
                                        ; de teller

                stx     COUNT+1
```

```
        ; Schuif de overdrachtvlag in het laagstwaardige bit van het
        ; bovenste deeltal

SLUPR:
                ldx     LENGTH
                ldy     #0


SLLP2:
                lda     (HDEPTR),Y
                rol     A
                sta     (HDEPTR),Y
                iny                     ; Verhoog de index
                dex
                bne     SLLP2           ; Ga door tot de lengte 0 is

        ; Trek ARRAY 2 van het hoogstwaardige deeltal af, en zet het
        ; resultaat in het andere hoogstwaardige deeltal

                ldx     LENGTH
                ldy     #0
                sec

SUBLP:
                lda     (HDEPTR),Y
                sbc     (AY2PTR),Y      ; Trek de bytes van elkaar af
                sta     (ODEPTR),Y      ; Sla het verschil op
                iny                     ; Verhoog de index
                dex
                bne     SUBLP           ; Ga door tot de lengte 0 is

        ; Als de overdrachtvlag niet gezet is bij het aftrekken dan is
        ; het hoogstwaardige deeltal kleiner dan ARRAY 2, dus moet het
        ; volgende bit van het quotient 0 zijn.
        ; Als de overdrachtvlag wel gezet is dan moet het quotient 1
        ; zijn en wisselen we het deeltal en de rest om door de wijzers
        ; te verwisselen

                bcc     LOOP            ; Was de aftrekking een succes
                ldy     HDEPTR          ; Ja, verwissel dus de wijzers
                                        ; om zo het deeltal en de rest
                                        ; te verwisselen
                ldx     HDEPTR+1
                lda     ODEPTR
                sta     HDEPTR
                lda     ODEPTR+1
                sta     HDEPTR+1
                sty     ODEPTR
                stx     ODEPTR+1

        ; Ga verder met het volgende bit van A = 1 (overdrachtvlag = 1)

                jmp     LOOP


        ; Maak de overdrachtvlag schoon als indicatie dat alles goed ging

OKEXIT:
                clc
                bcc     EXIT
```

```
        ; Zet de overdrachtvlag  aan als indicatie dat er iets mis is
        ; gegaan

EREXIT:
                sec


EXIT:
        ; ARRAY 1 bevat het quotient
        ; HDEPTR bevat het adres van de rest

                rts


; Data
RETADR:         .BLOCK  2           ; Het terugkeeradres
COUNT:          .BLOCK  2           ; De lus-teller
LENGTH:         .BLOCK  1           ; De lengte van de arrays
AHIDE1:         .WORD   HIDE1       ; Adres van buffer 1 voor het
                                    ; hoogstwaardige deeltal
AHIDE2:         .WORD   HIDE2       ; Adres van buffer 2 voor het
                                    ; hoogstwaardige deeltal
HIDE1:          .BLOCK  255         ; Buffer 1
HIDE2:          .BLOCK  255         ; Buffer 2


;
;
; Voorbeeld uitvoering
;
;

SC0609:
                lda     AY1ADR+1
                pha
                lda     AYADR
                pha                 ; Zet het adres van ARRAY 1 op de
                                    ; stack

                lda     AY2ADR+1
                pha
                lda     AY2ADR
                pha                 ; Zet het adres van ARRAY 2 op de
                                    ; stack

                lda     #SZAYS
                pha                 ; Zet de lengte van de arrays op de
                                    ; stack
                jsr     MPBDIV      ; Voor de deling uit
                brk                 ; Het resultaat van
                                    ;  14B060404H / 1234H = 12345H
                                    ;  staat als volgt in het geheugen
                                    ;
                                    ;    AY1     = 45H
                                    ;    AY1+1   = 23H
                                    ;    AY1+2   = 01H
                                    ;    AY1+3   = 00H
                                    ;    AY1+4   = 00H
                                    ;    AY1+5   = 00H
                                    ;    AY1+6   = 00H
                jmp     SC0609
```

```
SZAYS:          .EQU    7           ; De lengte van de arrays
AY1ADR:         .WORD   AY1         ; Het adres van het deeltal
AY2ADR:         .WORD   AY2         ; Het adres van de deler

AY1:
                .BYTE   004H
                .BYTE   004H
                .BYTE   0B6H
                .BYTE   014H
                .BYTE   0
                .BYTE   0
                .BYTE   0


AY2:
                .BYTE   034H
                .BYTE   012H
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0

                .END
```

Meervoudige Precisie Binaire Vergelijking - 6J

Vergelijkt twee, uit meerdere bytes bestaande, ongetekende binaire getallen en zet de overschot- en nulvlaggen overeenkomstig. De nulvlag wordt op 1 gezet als de getallen gelijk zijn en op 0 als dit niet het geval is. De overschotvlag wordt op 0 gezet als het getal met het adres hoger in de stack (de aftrekker) groter is dan het andere getal (het aftrektal); zo niet dan wordt de overschotvlag op 1 gezet. Zodoende worden de vlaggen gezet alsof de aftrekker van het aftrektal is afgetrokken.
Procedure: Het programma vergelijkt de getallen byte-voor-byte, beginnend bij de hoogstwaardige bytes, en gaat door tot het bij elkaar horende bytes vindt die niet gelijk zijn. Als alle bytes gelijk zijn, keert het programma terug met de nulvlag op 1 gezet. Merk op dat het programma vanaf de hoogstwaardige bytes door de getallen heen gaat, terwijl de aftrekkingsroutine (subroutine 6G) begint bij de laagstwaardige bytes.

Gebruikte registers: Alle

Looptijd: 17 cycli per te vergelijken byte, plus 90 cycli vast. Ofte wel, het programma gaat door tot het bij elkaar horende bytes vindt die niet gelijk zijn; elk paar bytes dat vergeleken moet worden, kost 17 cycli.

Voorbeelden:

1. Het vergelijken van twee gelijke, 6 bytes grote getallen, duurt 17 x 6 + 90 = 192 cycli.

2. Het vergelijken van twee 8-bytes getallen die verschillen in de op een na hoogstwaardige byte duurt 17 x 2 + 90 = 124 cycli.

Programmalengte: 54 bytes

Benodigd datageheugen: Twee bytes ergens in RAM en vier bytes op pagina 0. De twee bytes ergens in RAM dienen als tijdelijke opslag voor het terugkeeradres (beginnende bij adres RETADR). De vier bytes op pagina 0 bevatten pointers naar de twee getallen; de pointers beginnen bij de adressen MINPTR ($00D0_{16}$ in de listing) en SUBPTR ($00D2_{16}$ in de listing).

Speciaal geval: Het programma stopt onmiddelijk met zowel de overschot- als de nulvlag op 1 gezet als de lengte van de getallen nul is.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Lengte van de getallen in bytes
Laagstwaardige byte van het beginadres van de aftrekker (het adres dat de laagstwaardige byte bevat)
Hoogstwaardige byte van het beginadres van de aftrekker (het adres dat de laagstwaardige byte bevat)

Laagstwaardige byte van het beginadres van het aftrektal (het adres dat de laagstwaardige byte bevat)
Hoogstwaardige byte van het beginadres van het aftrektal (het adres dat de laagstwaardige byte bevat)

Eindcondities

Vlaggen gezet alsof de aftrekker van het aftrektal is afgetrokken

Nulvlag = 1 als de aftrekker en aftrektal gelijk zijn, 0 indien zij niet gelijk zijn

Overschotvlag = 0 als de aftrekker in ongetekende zin groter is dan het aftrektal, 1 als hij kleiner is dan of gelijk is aan het aftrektal.

Voorbeelden

1. Data: Lengte van de getallen (in bytes) = 6
Bovenste getal (aftrekker) = $19D028A193EA_{16}$
Onderste getal (aftrektal) = $4E67BC15A266_{16}$

   Resultaat: Nulvlag = 0 (getallen zijn niet gelijk)
Overschotvlag = 1 (aftrekker is niet groter dan aftrektal)

2. Data: Lengte van de getallen (in bytes) = 6
Bovenste getal (aftrekker) = $19D028A193EA_{16}$
Onderste getal (aftrektal) = $19D028A193EA_{16}$

   Resultaat: Nulvlag = 1 (getallen zijn gelijk)
Overschotvlag = 1 (aftrekker is niet groter dan aftrektal)

3. Data: Lengte van de getallen (in bytes) = 6
Bovenste getal (aftrekker) = $19D028A193EA_{16}$
Onderste getal (aftrektal) = $F37E5991D7C_{16}$

   Resultaat: Nulvlag = 0 (getallen zijn niet gelijk)
Overschotvlag = 0 (aftrekker is groter dan aftrektal)

```
;       Titel     : Multi-precies binair vergelijken
;       Naam      : MPBCMP
;
;
;       Doel      : Vergelijk 2 arrays binaire bytes en keer terug met de
;                   overschot- en de nulvlaggen gezet of niet
;
;
;       Begin     : Top van de stack
;                     Laagstwaardige byte van het terugkeeradres
;                     Hoogstwaardige byte van het terugkeeradres
```

```
;                   Lengte van de arrays in bytes
;                   Laagstwaardige byte van array 2
;                   Hoogstwaardige byte van array 2
;                   Laagstwaardige byte van array 1
;                   Hoogstwaardige byte van array 1
;
;                 De arrays bestaan uit binaire getallen zonder teken met
;                 een lengte van maximaal 255 bytes
;                 ARRAY[0] is de laagstwaardige byte en ARRAY[LENGTH-1]
;                 de hoogstwaardige byte van het array
;
;     Eind      : Als geldt
;                     ARRAY 1 = ARRAY 2
;                        dan C = 1, Z = 1
;                     ARRAY 1 > ARRAY 2
;                        dan C = 1, Z = 0
;                     ARRAY 1 < ARRAY 2
;                        dan C = 0, Z = 0
;
;     Registers : Alle
;
;     Tijd      : 17 cycli per byte die bekeken moet worden plus 90
;                 basiscycli
;
;     Grootte   : programma 54  bytes
;                 data       2  bytes plus
;                            4  bytes op pagina 0

; Wijzers naar adressen van arrays op pagina 0

MINPTR:                .EQU   0D0H        ; Wijzer naar array 1
SUBPTR:                .EQU   0D2H        ; Wijzer naar array 2

MPBCMP:
          ; Bewaar het terugkeeradres

                   pla
                   sta     RETADR
                   pla
                   sta     RETADR+1

          ; Haal de lengte van de arrays van de stack

                   pla
                   tay


          ; Haal het adres van het array 2 van de stack en trek er 1 vanaf
          ; om het indexeren te vergemakkelijken

                   pla
                   sec
                   sbc     #1             ; Verlaag de laagstwaardige byte
                                          ; van het adres
                   sta     SUBPTR
                   pla
                   sbc     #0             ; Correctie voor het eventuele
                                          ; lenen bij de vorige aftrekking
                   sta     SUBPTR+1

          ; Haal het adres van array 1 van de stack en trek er ook 1 vanaf
```

```
                pla
                sec
                sbc     #1
                sta     MINPTR
                pla
                sbc     #0
                sta     MINPTR+1

        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha

        ; Initialiseer

                cpy     #0              ; Is de lengte van de arrays 0?
                beq     EXIT            ; Zo ja, dan zijn we al klaar

LOOP:
                lda     (MINPTR),Y      ; Haal de volgende byte op
                cmp     (SUBPTR),Y      ; Vergelijk de bytes
                bne     EXIT            ; Spring als ze niet gelijk zijn
                dey                     ; Verlaag de teller
                bne     LOOP            ; Ga door tot de teller 0 is
                                        ;  Als we door deze test heen
                                        ;  vallen zijn de arrays gelijk
                                        ;  en staan de vlaggen goed


EXIT:
                rts


; Data
RETADR:         .BLOCK  2               ; Het terugkeeradres


;
;
; Voorbeeld uitvoering
;
;

SC0610:
                lda     AY1ADR+1

                pha
                lda     AY1ADR
                pha                     ; Zet het adres van array 1 op de
                                        ; stack

                lda     AY2ADR+1
                pha
                lda     AY2ADR
                pha                     ; Zet het adres van array 2 op de
                                        ; stack

                lda     #SZAYS
                pha                     ; Zet de lengte op de stack
                jsr     MPBCMP          ; Vergelijk de arrays
```

```
                brk                     ; Het resultaat van het vergelijken
                                        ; van 7654321H met 1234567H levert
                                        ; op :
                                        ;   C = 1, Z = 0

                jmp     SC0610


SZAYS:          .EQU    7               ; Lengte van de arrays
AY1ADR:         .WORD   AY1             ; Adres van array 1
AY2ADR:         .WORD   AY2             ; Adres van array 2

AY1:
                .BYTE   021H
                .BYTE   043H
                .BYTE   065H
                .BYTE   007H
                .BYTE   0
                .BYTE   0
                .BYTE   0


AY2:
                .BYTE   067H
                .BYTE   045H
                .BYTE   023H
                .BYTE   001H
                .BYTE   0
                .BYTE   0
                .BYTE   0


                .END
```

Meervoudige precisie decimale optelling - 6K

Telt twee, uit meerdere bytes bestaande ongetekende decimale getallen bij elkaar op. Beide getallen worden met hun laagstwaardige digits op het laagste adres opgeslagen. De som vervangt een van de getallen (degene met het beginadres lager in de stack). De lengte van de getallen (in bytes) is 255 of minder. Het programma keert terug met de Decimale Modus (D) vlag gewist (binaire modus).
Procedure: Het programma schakelt eerst de decimale modus in door de D-vlag te zetten. Vervolgens wist het de overdrachtvlag en telt dan de getallen byte (d.i. twee digits) voor byte bij elkaar op, beginnende bij de laagstwaardige digits. De som vervangt het bovenste getal op de stack. Een lengte van 00 veroorzaakt een onmiddellijke terugkeer zonder optellingen. Het programma wist de D-vlag (en plaatst de processor zodoende in de binaire modus) voordat het stopt. De uiteindelijke overdrachtvlag geeft de optelling van de hoogstwaardige digits weer.

Gebruikte registers:	Alle
Looptijd:	23 cycli per byte, plus 82 cycli vast. De optelling, bij voorbeeld van twee 8-bytes (16 digits) getallen duurt 23 x 8 + 86 = 270 cycli.
Programmalengte:	50 bytes
Benodigd datageheugen:	Twee bytes ergens in RAM en vier bytes op pagina 0. De twee bytes ergens in RAM dienen als tijdelijke opslag voor het terugkeeradres (beginnende bij adres RETADR). De vier bytes op pagina 0 bevatten pointers naar de twee getallen; de pointers beginnen op de adressen AY1PTR ($00D0_{16}$ in de listing) en AY2PTR ($00D2_{16}$ in de listing).
Speciaal geval	Een lengte van nul veroorzaakt een onmiddellijke terugkeer, waarbij array 1 ongewijzigd is (ofte wel, de som is gelijk aan het onderste getal). De Decimale Modus vlag is gewist (binaire modus) en de overdrachtvlag is op 1 gezet.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Lengte van de getallen in bytes
Laagstwaardige byte van het beginadres van het tweede getal (adres dat de laagstwaardige byte van array 2 bevat)
Hoogstwaardige byte van het beginadres van het tweede getal (adres dat de laagstwaardige byte van array 2 bevat)
Laagstwaardige byte van het beginadres van het eerste getal (adres dat de laagstwaardige byte van array 1 bevat)
Hoogstwaardige byte van het beginadres van het eerste getal (adres dat de laagstwaardige byte van array 1 bevat)

Eindcondities

Eerste getal (array 1) vervangen door het eerste getal (array 1) plus het tweede getal (array 2)

D-vlag op nul gezet (binaire modus)

Voorbeeld

Data: Lengte van de getallen (in bytes) = 6
Bovenste getal (array 2) = 196028819315_{16}
Onderste getal (array 1) = 293471605987_{16}

Resultaat: Onderste getal (array 1) = Onderste getal (array 1) + Bovenste getal (array 2) = 489500425302_{16}
Overdrachtvlag = 0, Decimale Modusvlag = 0 (binaire modus)

```
;       Titel      : Multi-precies decimaal optellen
;       Naam       : MPDADD
;
;
;       Doel       : Tel twee arrays data in BCD-formaat bij elkaar op
;                       Array1 := Array1 + Array2
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Lengte van het array in bytes
;                      Laagstwaardige byte van het beginadres van array 2
;                      Hoogstwaardige byte van het beginadres van array 2
;                      Laagstwaardige byte van het beginadres van array 1
;                      Hoogstwaardige byte van het beginadres van array 1
;
;                      De arrays bestaan uit BCD getallen zonder teken met een
;                      maximale lengte van 255 bytes, ARRAY[0] is de
;                      laagstwaardige byte, en ARRAY[LENGTE-1] is de
;                      hoogstwaardige byte
;
;       Eind       : Array1 := Array1 + Array2
;
;       Registers  : Alle
;
;       Tijd       : 23 cycli per byte plus 86 basiscycli
;
;       Grootte    : programma  50  bytes
;                    data        2  bytes plus
;                                4  bytes op pagina 0

; Wijzers naar adressen op pagina 0

AY1PTR:             .EQU    0D0H          ; Wijzer naar array 1
AY2PTR:             .EQU    0D2H          ; Wijzer naar array 2


MPDADD:
```

```
        ; Bewaar het terugkeeradres

                pla
                sta     RETADR
                pla
                sta     RETADR+1

        ; Haal de lengte van de arrays van de stack

                pla
                tax

        ; Haal het beginadres van array 2 van de stack

                pla
                sta     AY2PTR
                pla
                sta     AY2PTR+1

        ; Haal het beginadres van array 1 van de stack

                pla
                sta     AY1PTR
                pla
                sta     AY1PTR+1

        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha

        ; Initialiseer de som en de decimale modus
        ; Spring naar EXIT als de lengte 0 is

                ldy     #0
                cpx     #0              ; Is de lengte 0?
                beq     EXIT            ; Zo ja, spring naar EXIT
                sed                     ; Zet de decimale modus aan
                clc                     ; Maak de overdrachtvlag schoon

LOOP:
                lda     (AY1PTR),Y      ; Haal volgende byte op
                adc     (AY2PTR),Y      ; Tel ze bij elkaar op
                sta     (AY1PTR),Y      ; Zet het resultaat in array 1
                iny                     ; Verhoog de array-index
                dex                     ; Verlaag de lengteteller
                bne     LOOP            ; Spring totdat alle bytes opgeteld
                                        ; zijn
EXIT:
                cld                     ; Ga terug naar binaire modus
                rts


; Data
RETADR:         .BLOCK  2               ; Het terugkeeradres


;
;
; Voorbeeld uitvoering
```

```
SC0611:
                lda     AY1ADR+1
                pha
                lda     AY1ADR
                pha                     ; Beginadres van array 1

                lda     AY2ADR+1
                pha
                lda     AY2ADR
                pha                     ; Beginadres van array 2

                lda     #SZAYS
                pha                     ; De lengte van de arrays
                jsr     MPDADD          ; Tel de arrays bij elkaar op
                brk                     ; Resultaat van

                                        ;   1234567 + 1234567 =
                                        ;      2469134
                                        ; In het geheugen opgeslagen als:
                                        ;
                                        ;    AY1          = 34H
                                        ;    AY1 + 1      = 91H
                                        ;    AY1 + 2      = 46H
                                        ;    AY1 + 3      = 02H
                                        ;    AY1 + 4      = 00H
                                        ;    AY1 + 5      = 00H
                                        ;    AY1 + 6      = 00H

                jmp     SC0611


; Data

SZAYS:          .EQU    7               ; Lengte van de arrays

AY1ADR:         .WORD   AY1             ; Adres van array 1
AY2ADR:         .WORD   AY2             ; Adres van array 2

AY1:
                .BYTE   067H
                .BYTE   045H
                .BYTE   023H
                .BYTE   001H
                .BYTE   0
                .BYTE   0
                .BYTE   0


AY2:
                .BYTE   067H
                .BYTE   045H
                .BYTE   023H
                .BYTE   001H
                .BYTE   0
                .BYTE   0
                .BYTE   0


                .END
```

Meervoudige precisie decimale aftrekking - 6L

Trekt twee, uit meerdere bytes bestaande ongetekende getallen van elkaar af. Beide getallen worden met hun laagstwaardige digits op het laagste adres opgeslagen. Het beginadres van de aftrekker (het getal dat afgetrokken moet worden) is opgeslagen boven het beginadres van het aftrektal (het getal waarvan afgetrokken moet worden). Het verschil vervangt het aftrektal in het geheugen. De lengte van de getallen is 255 of minder. Het programma keert terug met de Decimale Modus (D) vlag gewist (binaire modus).
Procedure: Het programma schakelt eerste de decimale modus in door de D-vlag te zetten. Vervolgens wordt de overdrachtvlag (geinverteerde leen) gezet en de aftrekker byte (twee digits) voor byte van het aftrektal afgetrokken, beginnende bij de laagstwaardige digits. De uiteindelijke overdrachtvlag geeft de aftrekking van de hoogstwaardige bytes weer. Het verschil vervangt het aftrektal (het getal met beginadres lager in de stack, array 1 in de listing). Een lengte van 00 veroorzaakt een onmiddellijke terugkeer zonder aftrekkingen. Het programma wist de D-vlag (en plaatst zodoende de processor in de binaire modus) voordat het terugkeert.

Gebruikte registers:	Alle
Looptijd:	23 cycli per byte plus 86 cycli vast. De aftrekking, bij voorbeeld, van twee 8-bytes (16-digits) getallen duurt 23 x 8 + 86 = 270 cycli.
Programmalengte:	50 bytes
Benodigd datageheugen:	Twee bytes ergens in RAM en vier bytes op pagina 0. De twee bytes ergens in RAM dienen als tijdelijke opslag voor het terugkeeradres (beginnende bij adres RETADR). De vier bytes op pagina 0 bevatten pointers naar de twee getallen; de pointers beginnen op de adressen AY1PTR ($00D0_{16}$ in de listing) en AY2PTR ($00D2_{16}$ in de listing).
Speciaal geval	Een lengte van nul veroorzaakt een onmiddellijke terugkeer, waarbij het verschil gelijk is aan het oorspronkelijke aftrektal, de Decimale Modus vlag gewist (binaire modus) en de overdrachtvlag op 1 gezet is.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Lengte van de getallen in bytes
Laagstwaardige byte van het beginadres van de aftrekker (het adres dat de laagstwaardige byte van array 2 bevat)
Hoogstwaardige byte van het beginadres van de aftrekker (het adres dat de laagstwaardige byte van array 2 bevat)
Laagstwaardige byte van het beginadres van het aftrektal (het adres dat de laagstwaardige byte van array 1 bevat)
Laagstwaardige byte van het beginadres van het aftrektal (het adres dat de laagstwaardige byte van array 1 bevat)

Eindcondities

Aftrektal (array 1) vervangen door aftrektal (array 1) minus aftrekker (array 2)

D-vlag op nul gezet (binaire modus)

Voorbeeld

Data: Lengte van de getallen (in bytes) = 6
Aftrektal (array 1) = 293471605987_{16}
Aftrekker (array 2) = 196038819315_{16}

Resultaat: Verschil (array 1) = 097442786672_{16}
Dit getal vervangt het oorspronkelijke aftrektal in het geheugen. De overdrachtvlag wordt, in overeenstemming met haar gebruikelijke rol (bij het programmeren van de 6502) als een geinverteerde leen, op 1 gezet.
Decimale Modus vlag = 0 (binaire modus)

```
;       Titel      : Multi-precies decimaal aftrekken
;       Naam       : MPDSUB
;
;
;       Doel       : Trek twee arrays data in BCD-formaat van elkaar af
;                       Array1 := Array1 - Array2
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Lengte van het array in bytes
;                      Laagstwaardige byte van het beginadres van array 2
;                      Hoogstwaardige byte van het beginadres van array 2
;                      Laagstwaardige byte van het beginadres van array 1
;                      Hoogstwaardige byte van het beginadres van array 1
;
;                     De arrays bestaan uit decimale getallen zonder teken met
;                     een maximale lengte van 255 bytes, ARRAY[0] is de
;                     laagstwaardige byte, en ARRAY[LENGTE-1] is de
;                     hoogstwaardige byte
;
;       Eind       : Array1 := Array1 - Array2
;
;       Registers  : Alle
;
;       Tijd       : 23 cycli per byte plus 86 basiscycli
;
;       Grootte    : programma  50  bytes
;                    data        2  bytes plus
;                                4  bytes op pagina 0

; Wijzers naar adressen op pagina 0

MINPTR:             .EQU    0D0H          ; Wijzer naar array 1
SUBPTR:             .EQU    0D2H          ; Wijzer naar array 2
```

```
MPDSUB:
        ; Bewaar het terugkeeradres

                pla
                sta     RETADR
                pla
                sta     RETADR+1

        ; Haal de lengte van de arrays van de stack

                pla
                tax

        ; Haal het beginadres van array 2 van de stack

                pla
                sta     SUBPTR
                pla
                sta     SUBPTR+1

        ; Haal het beginadres van array 1 van de stack

                pla
                sta     MINPTR
                pla
                sta     MINPTR+1

        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha

        ; Initialiseer

                ldy     #0
                cpx     #0              ; Is de lengte 0?
                beq     EXIT            ; Zo ja, spring naar EXIT
                sed                     ; Zet de decimale modus aan
                clc                     ; Maak de overdrachtvlag schoon

LOOP:
                lda     (MINPTR),Y      ; Haal de volgende byte op
                sbc     (SUBPTR),Y      ; Trek ze van elkaar af
                sta     (MINPTR),Y      ; Zet het resultaat in array 1
                iny                     ; Verhoog de array-index
                dex                     ; Verlaag de lengteteller
                bne     LOOP            ; Spring totdat alle bytes opgeteld
                                        ; zijn
EXIT:
                cld                     ; Ga terug naar binaire modus
                rts


; Data
RETADR:         .BLOCK  2               ; Het terugkeeradres


;
;
; Voorbeeld uitvoering
```

```
SC0612:
                lda     AY1ADR+1
                pha
                lda     AY1ADR
                pha                     ; Beginadres van array 1

                lda     AY2ADR+1
                pha
                lda     AY2ADR
                pha                     ; Beginadres van array 2

                lda     #SZAYS
                pha                     ; De lengte van de arrays
                jsr     MPDSUB          ; Trek de twee arrays van elkaar af
                                        ;
                brk                     ; Resultaat van

                                        ;   2469134 - 1234567 =
                                        ;      1234567
                                        ;    AY1          = 67H
                                        ;    AY1 + 1      = 45H
                                        ;    AY1 + 2      = 23H
                                        ;    AY1 + 3      = 01H
                                        ;    AY1 + 4      = 00H
                                        ;    AY1 + 5      = 00H
                                        ;    AY1 + 6      = 00H

                jmp     SC0612


; Data

SZAYS:          .EQU    7               ; Lengte van de arrays

AY1ADR:         .WORD   AY1             ; Adres van array 1
AY2ADR:         .WORD   AY2             ; Adres van array 2

AY1:
                .BYTE   034H
                .BYTE   091H
                .BYTE   046H
                .BYTE   002H
                .BYTE   0
                .BYTE   0
                .BYTE   0

AY2:
                .BYTE   076H
                .BYTE   045H
                .BYTE   023H
                .BYTE   001H
                .BYTE   0
                .BYTE   0
                .BYTE   0


                .END
```

Meervoudige precisie decimale vermenigvuldiging - 6M

Vermenigvuldigt twee uit meerdere bytes bestaande, ongetekende decimale getallen. Beide getallen worden met hun laagstwaardige digits op het laagste adres opgeslagen. Het produkt vervangt een van de getallen (degene met het beginadres lager in de stack). De lengte van de getallen is 255 of minder. Alleen de laagstwaardige bytes van het produkt worden teruggeven om compatibiliteit te houden met andere meervoudige precisie decimale handelingen. Het programma keert terug met de Decimale Modus (D) vlag gewist (binaire modus).
Procedure: Het programma bewerkt elk digit van de vermenigvuldiger (array 1) apart. Het maskeert het digit, schuift het (als het zich in de bovenste nibble van een byte bevindt) en gebruikt het vervolgens als teller om te bepalen hoe vaak de vermenigvuldiger bij het deelprodukt opgeteld moet worden. Het laagstwaardige digit van het deelprodukt wordt bewaard als het volgende digit van het volledige produkt en het deelprodukt wordt vier bits naar rechts geschoven. Het programma gebruikt een vlag om aan te geven of het op dat moment met het bovenste of onderste digit van een byte werkt. Een lengte van 00 veroorzaakt een terugkeer zonder vermenigvuldiging.

Gebruikte registers: Alle

Looptijd: Hangt af van de lengte van de getallen en van de grootte van de digits in het vermenigvuldigtal (aangezien deze bepalen hoe vaak de vermenigvuldiger bij het deelprodukt wordt opgeteld).
Als de gemiddelde waarde van een digit in het vermenigvuldigtal 5 is, is de looptijd ongeveer

$$322 \times \text{LENGTE}^2 + 390 \times \text{LENGTE} + 100$$

cycli, waarbij LENGTE het aantal bytes in het getal is. Als, bij voorbeeld, LENGTE = 6 (12 digits), is de looptijd ongeveer

$$322 \times 6^2 + 390 \times 6 + 100 = 322 \times 36 + 2340 + 100 = 11.592 + 2440 = 14.032 \text{ cycli.}$$

Programmalengte: 203 bytes

Benodigd datageheugen: 517 bytes ergens in RAM, plus vier bytes op pagina 0. De 517 bytes ergens in RAM dienen voor tijdelijke opslag van het deelprodukt (255 bytes beginnende bij adres PROD), het vermenigvuldigtal (255 bytes beginnende bij adres MCAND), het terugkeeradres (twee bytes beginnende bij adres RETADR), de lengte van de getallen in bytes (een byte op adres LENGTH), het volgende digit in het getal (een byte op adres NDIGIT), de digitteller (een byte op adres DCNT), de index, ter grootte van een byte, in de getallen (een byte op adres IDX) en de overschot byte (1 byte op adres OVERFLW). De vier bytes op pagina 0 bevatten pointers naar de twee getallen; de pointers beginnen op de adresen AY1PTR ($00D0_{16}$ in de listing) en AY2PTR ($00D2_{16}$ in de listing).

Speciaal geval: Een lengte van nul veroorzaakt een onmiddellijke terugkeer, waarbij het produkt gelijk is aan het oorspronkelijke vermenigvuldigtal (array 1 is ongewijzigd), de Decimale Modus vlag gewist is (binaire modus) en de hoogstwaardige bytes van het produkt (beginnende bij adres PROD) niet gevormd zijn.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Lengte van de getallen in bytes
Laagstwaardige byte van het beginadres van de vermenigvuldiger (het adres dat de laagstwaardige byte van array 2 bevat)
Hoogstwaardige byte van het beginadres van de vermenigvuldiger (het adres dat de laagstwaardige byte van array 2 bevat)
Laagstwaardige byte van het beginadres van het vermenigvuldigtal (het adres dat de laagstwaardige byte van array 1 bevat)
Hoogstwaardige byte van het beginadres van het vermenigvuldigtal (het adres dat de laagstwaardige byte van array 1 bevat)

Eindcondities

Vermenigvuldigtal (array 1) vervangen door vermenigvuldigtal (array 1) maal vermenigvuldiger (array 2)

D-vlag op nul gezet (binaire modus)

Voorbeeld

Data:	Lengte van de getallen (in bytes) = 04 Bovenste getal (array 2 of vermenigvuldiger) = 00003518_{16} Onderste getal (array 1 of vermenigvuldigtal) = 00006294_{16}
Resultaat:	Onderste getal (array 1) = Onderste getal (array 1) * Bovenste getal (array 2) = 22142292_{16}
	Decimale Modus vlag = 0 (binaire modus)

Merk op dat dit programma slechts de laagstwaardige bytes van het produkt teruggeeft (ofte wel het aantal bytes in het vermenigvuldigtal en vermenigvuldiger) om compatibiliteit te houden met andere meervoudige precisie decimale rekenkundige handelingen. De hoogstwaardige bytes van het produkt zijn beschikbaar, beginnende met hun laagstwaardige digits op adres PROD. De gebruiker zal, indien nodig, deze bytes moeten controleren op een mogelijk overschot, of de getallen met extra nullen moeten verlengen.

```
;
;     Titel      : Multi-precies decimaal vermenigvuldigen
;     Naam       : MPDMUL
;
;
;     Doel       : Vermenigvuldig 2 arrays bytes in BCD-formaat met elkaar
;                      Array1 := Array1 * Array2
;
;
;     Begin      : Top van de stack
;                    Laagstwaardige byte van het terugkeeradres
;                    Hoogstwaardige byte van het terugkeeradres
;                    Lengte van het array in bytes
;                    Laagstwaardige byte van het beginadres van array 2
;                    Hoogstwaardige byte van het beginadres van array 2
;                    Laagstwaardige byte van het beginadres van array 1
;                    Hoogstwaardige byte van het beginadres van array 1
;
;                   De arrays bestaan uit decimale getallen zonder teken met
;                   een maximale lengte van 255 bytes, ARRAY[0] is de
;                   laagstwaardige byte, en ARRAY[LENGTE-1] is de
;                   hoogstwaardige byte
;
;     Eind       : Array1 := Array1 * Array2
;
;     Registers : Alle
;
;     Tijd       : Aangenomen dat de gemiddelde cijferwaarde van array 1
;                  5 is dan is de benodigde tijdsduur
;                  (322 * de lengte tot de tweede macht) + (390 * de lengte)
;                  + 100 basiscycli
;
;     Grootte    : programma  203  bytes
;                  data       517  bytes plus
;                               4  bytes op pagina 0

; Pointers naar adressen op pagina 0
AY1PTR:            .EQU    0D0H          ; Pointer naar array 1
AY2PTR:            .EQU    0D2H          ; Pointer naar array 2


MPDMUL:
          ; Bewaar het terugkeeradres

                   pla
                   sta     RETADR
                   pla
                   sta     RETADR+1

          ; Haal de lengte van de arrays van de stack

                   pla
                   sta     LENGTH

          ; Haal het beginadres van array 2 van de stack

                   pla
                   sta     AY2PTR
                   pla
```

```
        sta     AY2PTR+1

; Haal het beginadres van array 1 van de stack

        pla
        sta     AY1PTR
        pla
        sta     AY1PTR+1

; Zet het terugkeeradres weer op de stack

        lda     RETADR+1
        pha
        lda     RETADR
        pha

; Initialiseer

        sed                     ; Ga naar decimale modus
        ldy     #0
        ldx     LENGTH          ; Is de lengte 0?
        bne     INITLP
        jmp     EXIT            ; Zo ja, spring naar EXIT

; Kopieer de inhoud van array 1 in het multiplicant-array
; Maak array 1 leeg
; Maak het produkt-array leeg
INITLP:
        lda     (AY1PTR),Y
        sta     MCAND,Y         ; Kopieer ARY[Y] naar MCAND[Y]
        lda     #0
        sta     (AY1PTR),Y      ; Maak ARY1[Y] leeg
        sta     PROD,Y          ; Maak PROD[Y] leeg
        iny
        dex                     ; Verlaag de lusteller
        bne     INITLP          ; Ga door tot we klaar zijn

; Maak de huidige index gelijk aan 0

        lda     #0
        sta     IDX

; Loop alle bytes van de multiplicant af

LOOP:
        lda     #0
        sta     DCNT

; Verwerk 2 cijfers per byte
;  Tijdens het eerste cijfer is DCNT gelijk aan 0
;  Tijdens het tweede cijfer is DCNT gelijk aan FFH (-1)

DLOOP:
        lda     #0
        sta     OVRFLW          ; Zet de overloop op 0
        ldy     IDX
        lda     MCAND,Y         ; Haal de volgende byte op
        ldx     DCNT
        bpl     DLOOP1          ; Spring als hij het eerste cijfer
                                ; is
        lsr     A               ; Schuif A, 4 bits naar rechts
        lsr     A
        lsr     A
```

```
                lsr     A
DLOOP1:
                and     #0FH        ; Haal het hoogste cijfer eraf
                beq     SDIGIT      ; Spring als het volgende cijfer
                                    ; gelijk aan 0 is
                sta     NDIGIT      ; Bewaar het


ADDLP:
                ldy     #0          ; Y = index in de arrays
                ldx     LENGTH      ; X = lengte in bytes
                clc                 ; Zet de overdrachtvlag op 0

INNER:
                lda     (AY2PTR),Y  ; Haal de volgende byte op
                adc     PROD,Y      ; Tel dit bij het produkt op
                sta     PROD,Y      ; Bewaar het produkt
                iny                 ; Verhoog de array-index
                dex                 ; Verlaag de byte-teller
                bne     INNER       ; Ga door tot de byte-teller gelijk
                                    ; aan 0 is
                bcc     DECND       ; Spring als er bij het optellen
                                    ; geen overloop is opgetreden
                inc     OVRFLW      ; Verhoog anders de overloopbyte


DECND:
                dec     NDIGIT
                bne     ADDLP       ; Ga door tot NDIGIT = 0

        ; Bewaar het laagstwaardige cijfer van het produkt
        ;  als het volgende cijfer van array 1

SDIGIT:
                lda     PROD
                and     #0FH        ; Haal het bovenste cijfer eraf
                ldx     DCNT
                bpl     SD1         ; Spring als dit het eerste cijfer
                                    ; is
                asl     A           ; Anders schuif A 4 bits naar
                asl     A           ; links om het cijfer in de
                asl     A           ; bovenste positie te brengen
                asl     A


SD1:
                ldy     IDX         ; Haal de huidige index op
                ora     (AY1PTR),Y  ; Stop het volgende cijfer
                sta     (AY1PTR),Y  ; bij wat er al in array 1 stond

        ; Shift het produkt 1 cijfer (4 bits) naar rechts

                ldy     LENGTH      ; Begin met het laatste cijfer
                                    ; van het array


SHFTLP:
                dey                 ; Verlaag Y zodat naar de
                                    ; volgende byte gewezen wordt
                lda     PROD,Y
                pha                 ; Bewaar het cijfer wat nu
                                    ; wordt aangewezen op de stack
                and     #0F0H       ; Haal het laagste cijfer
```

```
                                        ; eraf

        ; Maak het laagstwaardige cijfer van OVRFLW gelijk
        ;  aan het hoogstwaardige cijfer van PROD,Y
        ; Maak het hoogstwaardige cijfer van PROD,Y gelijk aan
        ;  het laagstwaardige cijfer van PROD,Y

                lsr     OVRFLW          ; Schuif OVRFLW naar rechts
                ora     OVRFLW          ; Bits 0..2 en de overdrachtvlag
                                        ; vormen de overloop
                                        ; Bits 4..7 = PROD

                ror     A
                ror     A
                ror     A
                ror     A               ; Nu zit PROD in bits 0..3 en
                                        ; de overloop in 4..7
                sta     PROD,Y          ; Bewaar het nieuwe produkt
                pla                     ; Haal het oude produkt van de
                                        ; stack af
                and     #0FH            ; Haal het bovenste cijfer eraf
                sta     OVRFLW          ; Bewaar het in OVRFLW
                tya                     ; Kijk of Y = 0
                bne     SHFTLP          ; Spring als we nog niet klaar zijn

        ; Kijk of we beide cijfers van deze byte afgewerkt hebben

                dec     DCNT            ; Verschuif 0 naar FF hex om het
                                        ; tweede cijfer aan te geven
                lda     DCNT
                cmp     #0FFH           ; Hebben we beide cijfers gehad?
                beq     DLOOP           ; Zo nee, spring terug

        ; Verhoog de indices zodat ze naar de volgende byte wijzen
        ; en kijk of we klaar zijn

                inc     IDX
                lda     IDX
                cmp     LENGTH
                bcs     EXIT            ; Spring als de byte-index
                                        ; >= LENGTH is
                jmp     LOOP            ; Ga door


EXIT:
                cld                     ; Ga terug naar binaire modus
                rts


; Data
RETADR:         .BLOCK  2               ; Tijdelijke opslag voor
                                        ; terugkeeradres
LENGTH:         .BLOCK  1               ; De lengte van de arrays
NDIGIT:         .BLOCK  1               ; Volgende cijfer in het array
DCNT:           .BLOCK  1               ; Cijferteller voor bytes in
                                        ; het array
IDX:            .BLOCK  1               ; Byte-index in het array
OVRFLW:         .BLOCK  1               ; Overloop-byte
PROD:           .BLOCK  255             ; Buffer voor het product
MCAND:          .BLOCK  255             ; Buffer voor de multiplicant
```

```
; Voorbeeld uitvoering

SC0613:
                lda     AY1ADR+1
                pha
                lda     AY1ADR
                pha                     ; Zet adres van array 1 op de stack

                lda     AY2ADR+1
                pha
                lda     AY2ADR
                pha                     ; Zet adres van array 2 op de stack

                lda     #SZAYS
                pha                     ; Zet de lengte op de stack
                jsr     MPDMUL          ; Vermenigvuldig de arrays
                brk                     ; Het resultaat van
                                        ;   1234 * 1234 = 1522756
                                        ;
                                        ; Opgeslagen als
                                        ;
                                        ;    AY1      = 56H
                                        ;    AY1+1    = 26H
                                        ;    AY1+2    = 52H
                                        ;    AY1+3    = 01H
                                        ;    AY1+4    = 00H
                                        ;    AY1+5    = 00H
                                        ;    AY1+6    = 00H

                jmp     SC0613


SZAYS:          .EQU    7               ; Lengte van de arrays
AY1ADR:         .WORD   AY1             ; Adres van array 1
AY2ADR:         .WORD   AY2             ; Adres van array 2

AY1:
                .BYTE   034H
                .BYTE   012H
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0


AY2:
                .BYTE   034H
                .BYTE   012H
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0


                .END
```

Meervoudige Precisie Decimale Deling - 6N

Deelt twee, uit meerdere bytes bestaande, ongetekende decimale getallen, op elkaar. Beide getallen zijn opgeslagen met hun laagstwaardige byte op het laagste adres. Het quotient vervangt het deeltal (het getal met het beginadres lager in de stack). De lengte van de getallen (in bytes) is 255 of minder. Het restgetal wordt niet teruggegeven, maar het adres van de laagstwaardige byte ervan is beschikbaar in de geheugenlocatie HDEPTR. De overdrachtvlag is gewist als er geen fouten optraden; als er poging wordt gedaan door nul te delen, wordt de overdrachtvlag op 1 gezet, blijft het deeltal ongewijzigd wordt het restgetal op nul gezet. Het programma keert terug met de Decimale Modus (D) vlag gewist (binaire modus).
Procedure: Het programma deelt middels proef-aftrekkingen, digit voor digit. Het bepaalt hoe vaak de deler van het deeltal afgetrokken kan worden, bewaart dat getal in het quotient en maakt van het restgetal het nieuwe deeltal. Vervolgens roteert het het restgetal en het quotient een digit naar links. Het programma stopt onmiddelijk, waarbij de overdrachtvlag wordt gezet, als de deler nul blijkt te zijn. Anders wordt de overdrachtvlag gewist.

Gebruikte Registers: Alle

Looptijd: Hangt af van de lengte van de getallen en de grootte van de digits in het quotient (waardoor het aantal proef-aftrekkingen wordt bepaald). Als de gemiddelde digit in het quotient een waarde van 5 heeft, is de looptijd ongeveer

$$440 \times LENGTE^2 + 765 \times LENGTE + 228$$

cycli, waarbij LENGTE het aantal bytes in de getallen is. Als, bijvoorbeeld, LENGTE = 6 (12 digits), is de looptijd ongeveer

$$440 \times 6^2 + 765 \times 6 + 228 = 440 \times 36 + 4590 + 228 = 15.840 + 4818 = 20.658$$ cycli.

Benodigd datageheugen: 522 bytes ergens in RAM plus acht bytes op pagina 0. De 522 bytes ergens in RAM dienen voor tijdelijke opslag van het hoge deeltal (255 bytes beginnende bij adres HIDE1), het resultaat van de proef-aftrekking (255 bytes beginnende bij adres HIDE2), het terugkeer-adres (twee bytes beginnende bij adres RETADR), een pointer naar het deeltal (twee bytes beginnende bij adres AY1PTR), de lengte van de getallen (een byte op adres LENGTH), het volgende digit in de array (een byte op adres NDIGIT), de deel lus teller (een byte op adres COUNT) en de adressen van de hoge deeltal-buffers (twee bytes elk, beginnende bij de adressen AHIDE1 en AHIDE2). De acht bytes op pagina 0 bevatten pointers naar de deler (adres AYPTR2, $00D0_{16}$ in de listing), het andere hoge deeltal (adres ODEPTR, $00D4_{16}$ in de listing) en het tijdelijke dat wordt gebruikt bij de rotatie naar links (adres RLPTR, $00D6_{16}$ in de listing).

Speciale Gevallen:

1. Een lengte van nul veroorzaakt een onmiddelijke terugkeer, waarbij de overdrachtvlag gewist is, het quotient gelijk is aan het oorspronkelijke deeltal (array 1 ongewijzigd), het restgetal niet gevormd en de Decimale Modus vlag gewist is.

2. Een deler die nul is veroorzaakt een terugkeer waarbij de overdrachtvlag op 1 is gezet, het quotient gelijk is aan het oorspronkelijke deeltal (array 1 ongewijzigd), het restgetal nul is en de Decimale Modus-vlag gewist is (binaire modus).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeer-adres
Hoogstwaardige byte van het terugkeer-adres
Lengte van de getallen in bytes
Laagstwaardige byte van het beginadres van de deler (het adres dat de laagstwaardige byte van array 2 bevat)
Hoogstwaardige byte van het beginadres van de deler (het adres dat de laagstwaardige byte van array 2 bevat)
Laagstwaardige byte van het beginadres van het deeltal (het adres dat de laagstwaardige byte van array 1 bevat)
Hoogstwaardige byte van het beginadres van het deeltal (het adres dat de laagstwaardige byte van array 1 bevat)

Eindcondities

Deeltal (array 1) vervangen door deeltal (array 1) gedeeld door deler (array 2)
Als de deler ongelijk is aan nul, is de overdrachtvlag gewist en is het resultaat normaal.
Als de deler nul is, is de overdrachtvlag gezet, het deeltal ongewijzigd en het restgetal nul.
Het restgetal is beschikbaar met de laagstwaardige digits opgeslagen vanaf de adressen in de locaties HDEPTR en HDEPTR+1.
De D-vlag is op nul gezet (binaire modus).

Voorbeeld

Data: Lengte van de getallen (in bytes) = 04
Bovenste getal (array 2 of deler) = 00006294_{16}
Onderste getal (array 1 of deeltal) = 22142298_{16}

Resultaat: Onderste getal (array 1) = Onderste getal (array 1) / Bovenste getal (array 2) = 00003518_{16}
Restgetal (beginnende bij het adres dat in de locaties HDEPTR en HDEPTR+1 staat) = 00000006_{16} = 6
Decimale Modus-vlag = 0 (binaire modus)
Overdrachtvlag is 0, om aan te geven dat er geen 'deling door nul' fout optrad.

```
;       Titel       : Multi-precies decimaal delen
;       Naam        : MPDDIV
;
```

```
;
;     Doel       : Deel 2 arrays bytes in BCD-formaat op elkaar
;                     Array1 := Array1 / Array2
;
;
;     Begin      : Top van de stack
;                    Laagstwaardige byte van het terugkeeradres
;                    Hoogstwaardige byte van het terugkeeradres
;                    Lengte van het array in bytes
;                    Laagstwaardige byte van het beginadres van array 2
;                    Hoogstwaardige byte van het beginadres van array 2
;                    Laagstwaardige byte van het beginadres van array 1
;                    Hoogstwaardige byte van het beginadres van array 1
;
;                   De arrays bestaan uit decimale getallen zonder teken met
;                   een maximale lengte van 255 bytes, ARRAY[0] is de
;                   laagstwaardige byte, en ARRAY[LENGTE-1] is de
;                   hoogstwaardige byte
;
;     Eind       : Array1 := Array1 / Array2
;
;     Registers  : Alle
;
;     Tijd       : Aangenomen dat de gemiddeldecijfer waarde van het
;                  quotient 5 is dan is de benodigde tijdsduur
;                  (400 * de lengte tot de tweede macht) + (765 * de lengte)
;                  + 228 basiscycli
;
;     Grootte    : programma  246  bytes
;                  data       522  bytes plus
;                               8  bytes op pagina 0


AY2PTR:            .EQU    0D0H        ; Wijzer naar array 2
HDEPTR:            .EQU    0D2H        ; Wijzer naar de huidige
                                       ; hoogstwaardige deeltal
ODEPRT:            .EQU    0D4H        ; Wijzer naar het andere deeltal
RLPTR:             .EQU    0D6H        ; Wijzer naar het array waarin naar
                                       ; links geroteerd wordt


MPDDIV:
          ; Haal het terugkeeradres van de stack

                   pla
                   sta     RETADR
                   pla
                   sta     RETADR+1

          ; Haal de lengte van de arrays van de stack

                   pla
                   sta     LENGTH


          ; Haal het beginadres van array 2 van de stack

                   pla
                   sta     AY2PTR
                   pla
                   sta     AY2PTR+1
```

```
        ; Haal het beginadres van array 1 van de stack

                pla
                sta     AY1PTR
                pla
                sta     AY1PTR+1

        ; Zet het terugkeeradres weer op de stack

                lda     RETADR+1
                pha
                lda     RETADR
                pha

        ; Initialiseer

                cld                     ; Zet de processor in binaire modus

                lda     LENGTH
                bne     INIT            ; Spring als de lengte ongelijk aan
                                        ; 0 is
                jmp     OKEXIT          ; Anders zijn we klaar


        ; Vul beide deeltal-buffers met 0

INIT:
                lda     #0              ; A := 0
                ldy     LENGTH          ; Y := LENGTH

INITLP:
                sta     HIDE1-1,Y
                sta     HIDE2-1,Y
                dey
                bne     INITLP

        ; Zet de deeltal-buffer-wijzers goed

                lda     AHIDE1
                sta     HDEPTR
                lda     AHIDE1+1
                sta     HDEPTR+1
                lda     AHIDE2
                sta     ODEPTR
                lda     AHIDE2+1
                sta     ODEPTR+1

        ; NDIGIT := 0

                lda     #0
                sta     NDIGIT

        ; Maak COUNT gelijk aan het aantal cijfers plus 1
        ;   COUNT := (LENGTH * 2) + 1

                lda     LENGTH
                asl     A               ; LENGTH * 2
                sta     COUNT
                lda     #0
                rol     A               ; Stop de overloop van de
                                        ; vermenigvuldiging in A
```

```
                inc     COUNT+1         ; Bewaar de hoogstwaardige byte van
                                        ; COUNT
                inc     COUNT
                bne     CHKDV0          ; Spring als er geen overloop was
                inc     COUNT+1

        ; Test op delen door 0

CHKDV0:
                ldx     LENGTH
                ldy     #0
                tya

DV01:
                ora     (AY2PTR),Y
                iny
                dex
                bne     DV01            ; Ga door zolang er nog bytes zijn

                cmp     #0
                bne     DVLOOP          ; Spring als de deler ongelijk aan
                                        ; 0 is
                jmp     EREXIT          ; Geef een foutindicatie en keer
                                        ; terug

        ; Voer de deling uit door middel van herhaald aftrekken

DVLOOP:
        ; Roteer het laagstwaardige deeltal en het quotient naar links
        ; Het hoogstwaardige cijfer van NDIGIT wordt het laagstwaardige
        ; cijfer van het quotient en het hoogstwaardige cijfer van het
        ; deeltal gaat naar het laagstwaardige cijfer van NDIGIT

                lda     AY1PTR+1
                ldy     AY1PTR
                jsr     RLARY           ; Roteer array 1

        ; Als COUNT = 0 dan zijn we klaar

                dec     COUNT
                bne     ROLDVB          ; Spring als het laagstwaardige
                                        ; byte niet 0 is
                lda     COUNT+1         ; Zet de hoogstwaardige byte in A
                beq     OKEXIT          ; Ga door totdat COUNT = 0
                dec     COUNT+1         ; Verlaag de hoogstwaardige byte
                                        ; van COUNT

        ; Roteer het deeltal op de plaats waarbij het laagstwaardige
        ; cijfer van het hoogstwaardige deeltal, het hoogstwaardige
        ; cijfer van NDIGIT wordt
ROLDVB:
                lda     HDEPTR+1
                ldy     HDEPTR
                jsr     RLARY

        ; Kijk hoeveel maal de deler in het hoogstwaardige deeltal kan.
        ; Als we uit deze lus komen is het hoogstwaardige cijfer van

        ; NDIGIT, het volgende cijfer van de deling, en het
        ; hoogstwaardige deeltal, de rest.

                lda     #0
```

```
                sta     NDIGIT          ; NDIGIT = 0
                sed                     ; Zet de processor in decimale
                                        ; modus

SUBLP:
                ldy     #0              ; Y is de index in de arrays
                ldx     LENGTH          ; X is de lengte van de arrays
                sec                     ; Zet het geinverteerde leenbit

INNER:
                lda     (HDEPTR),Y      ; Haal volgende byte van deeltal op
                sbc     (AY2PTR),Y      ; Trek volgende byte van deler eraf
                sta     (ODEPTR),Y      ; Bewaar het verschil voor volgende
                                        ; aftrekking
                iny                     ; Verhoog de index
                dex                     ; Verlaag de teller
                bne     INNER           ; Ga door totdat we alle bytes
                                        ; gehad hebben
                bcc     DVLOOP          ; Spring als er geleend moest
                                        ; worden, NDIGIT bevat het aantal
                                        ; keer dat de deler in het
                                        ; oorspronkelijke hoogstwaardige
                                        ; deeltal past
                                        ; Het hoogstwaardige deeltal bevat
                                        ; nu de rest

        ; Verhoog NDIGIT

                lda     NDIGIT
                clc
                adc     #10H
                sta     NDIGIT

        ; Verwissel de wijzers, zodat de rest, het nieuwe deeltal wordt

                ldx     HDEPTR
                ldy     HDEPTR+1
                lda     ODEPTR
                sta     HDEPTR
                lda     ODEPTR+1
                sta     HDEPTR+1
                stx     ODEPTR
                sty     ODEPTR+1

                jmp     SUBLP           ; Ga door tot het verschil negatief
                                        ; wordt

        ; Geen fouten, wis de overdrachtvlag

OKEXIT:
                clc
                bcc     EXIT

        ; Delen door 0, zet de overdrachtvlag aan

EREXIT:
                sec

EXIT:
```

```
        ; HDEPTR bevat het adres van de rest

                cld                     ; Keer terug in binaire modus
                rts


;
; Subroutine : RLARY
; Doel       : Roteer de cijfers in een array 1 plaats opzij (4 bits)
; Begin      : A = hoogstwaardige byte van het arrayadres
;              Y = laagstwaardige byte van het arrayadres
;              Het hoogstwaardige cijfer van NDIGIT is cijfer dat aangeeft
;              hoeveel maal geroteerd moet worden
; Einde      : Het array in NDIGIT maal geroteerd
; Registers  : Allemaal

RLARY:
        ; Bewaar het adres van het array

                sta     RLPTR+1
                sty     RLPTR

        ; Schuif NDIGIT in het laagstwaardige cijfer van het array
        ; Schuif het array naar links

                ldx     LENGTH
                ldy     #0              ; Begin bij ARY[0]

SHIFT:
                lda     (RLPTR),Y       ; Haal volgende byte op
                pha                     ; Bewaar die op stack
                and     #0FH            ; Nu eerst alleen het
                                        ; laagstwaardige cijfer

                asl     NDIGIT
                ora     NDIGIT          ; Bits 0-3 = laagstwaardige cijfer
                                        ; van het array
                                        ; Bits 5-7 en overdrachtvlag
                                        ; = volgende cijfer
                rol     A
                rol     A
                rol     A
                rol     A               ; Nu zit NDIGIT in bit 0-3 en
                                        ; laagstwaardige cijfer op de
                                        ; plaats van het hoogstwaardige
                                        ; cijfer

                sta     (RLPTR),Y       ; Bewaar het
                pla                     ; Haal het oude hoogstwaardige
                                        ; cijfer van de stack
                and     #0F0H           ; Maak het laagstwaardige cijfer 0
                sta     NDIGIT          ; Bewaar het resultaat
                iny                     ; Verhoog de index
                dex                     ; Verlaag de teller
                bne     SHIFT           ; Ga door als we nog niet genoeg
                                        ; geroteerd hebben
                rts


; Data
RETADR:         .BLOCK  2               ; Het terugkeeradres
```

```
AY1PTR:         .BLOCK  2           ; Beginadres van array 1
LENGTH:         .BLOCK  1           ; Lengte van de arrays
NDIGIT:         .BLOCK  1           ; Volgende cijfer in het array
COUNT:          .BLOCK  2           ; Deel-lus-teller
AHIDE1:         .WORD   HIDE1       ; Beginadres hoogstwaardige
                                    ; deeltal-buffer 1
AHIDE2:         .WORD   HIDE2       ; Beginadres hoogstwaardige
                                    ; deeltal-buffer 2
HIDE1:          .BLOCK  255.        ; Deze twee buffers zelf
HIDE2:          .BLOCK  255.


;
;
; Voorbeeld uitvoering
;
;

SC0614:
                lda     AY1ADR+1
                pha
                lda     AY1ADR
                pha                 ; Zet beginadres array 1 op de
                                    ; stack

                lda     AY2ADR+1
                pha
                lda     AY2ADR
                pha                 ; Zet beginadres array 2 op de
                                    ; stack

                lda     #SZAYS
                pha                 ; Zet lengte van de arrays op de
                                    ; stack
                jsr     MPDDIV      ; Deel de arrays op elkaar
                brk                 ; Resultaat van
                                    ;  1522756 / 1234 = 1234H
                                    ;
                                    ; In het geheugen opgeslagen als:
                                    ;
                                    ;    AY1     = 34H
                                    ;    AY1 + 1 = 12H
                                    ;    AY1 + 2 = 00H
                                    ;    AY1 + 3 = 00H
                                    ;    AY1 + 4 = 00H
                                    ;    AY1 + 5 = 00H
                                    ;    AY1 + 6 = 00H


                jmp     SC0614


SZAYS:          .EQU    7           ; Lengte van de arrays

AY1ADR:         .WORD   AY1         ; Beginadres array 1 (deeltal)
AY2ADR:         .WORD   AY2         ; Beginadres array 2 (deler)

AY1:
                .BYTE   056H
                .BYTE   027H
                .BYTE   052H
                .BYTE   01H
                .BYTE   0
```

```
                .BYTE   0
                .BYTE   0

AY2:
                .BYTE   034H
                .BYTE   012H
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0
                .BYTE   0

                .END
```

Meervoudige precisie decimale vergelijking - 6O

Vergelijkt twee, uit meerdere bytes bestaande, ongetekende decimale (BCD) getallen en zet de overschot- en nulvlaggen overeenkomstig. De nulvlag wordt op 1 gezet als de getallen gelijk zijn en op 0 als dit niet het geval is. De overschotvlag wordt op 0 gezet als het getal met het adres hoger in de stack (de aftrekker) groter is dan het andere getal (het aftrektal); de overschotvlag wordt anders op 1 gezet. Zodoende worden de vlaggen gezet alsof de aftrekker van het aftrektal is afgetrokken.
Noot: Dit programma is precies hetzelfde als subroutine 6J, de meervoudige binaire vergelijking, aangezien de CMP-instructie in de decimale modus hetzelfde doet als in de binaire modus. Zie daarom subroutine 6J voor een listing en de details.

Voorbeelden

1. Data:
Lengte van de getallen (in bytes) = 6
Bovenste getal (aftrekker) = 196528719340_{16}
Onderste getal (aftrektal) = 456780153266_{16}

Resultaat:
Nulvlag = 0 (getallen zijn niet gelijk)
Overschotvlag = 1 (aftrekker is niet groter dan aftrektal)

2. Data;
Lengte van de getallen (in bytes) = 6
Bovenste getal (aftrekker) = 196528719340_{16}
Onderste getal (aftrektal) = 196528719340_{16}

Resultaat:
Nulvlag = 1 (getallen zijn gelijk)
Overschotvlag = 1 (aftrekker is niet groter dan aftrektal)

3. Data:
Lengte van de getallen (in bytes) = 6
Bovenste getal (aftrekker) = 196528719340_{16}
Onderste getal (aftrektal) = 073785991074_{16}

Resultaat:
Nulvlag = 0 (getallen zijn niet gelijk)
Overschotvlag = 0 (aftrekker is groter dan aftrektal)

Zet bit - 7A

Zet een opgegeven bit op 1 in een 16-bits woord.
Procedure: Het programma gebruikt bits 0 tot en met 2 van register X om te bepalen welk bit gezet moet worden. Bit 3 wordt gebruikt om een bepaalde byte van de oorspronkelijke, woord-grote data te kiezen. Vervolgens voert het een logische OR uit op de gekozen byte, met een masker dat een 1 op de gewenste bit-positie heeft en een 0 op alle andere posities. De maskers met een 1-bit zijn beschikbaar in een tabel.

Gebruikte registers:	Alle
Looptijd:	57 cycli
Programmalengte:	42 bytes
Benodigd datageheugen:	Twee bytes ergens in RAM (beginnende bij adres VALUE)
Speciaal geval:	Bit-posities boven 15 worden mod 16 beschouwd. Bit-positie 16, bij voorbeeld, is voor dit programma hetzelfde als bit-positie 0.

Begincondities

Hoogstwaardige databyte in de accumulator
Laagstwaardige databyte in het register Y
Te zetten bit-nummer in het register X

Eindcondities

Hoogstwaardige byte van het resultaat in de accumulator
Laagstwaardige byte van het resultaat in het register Y

Voorbeelden

1.	Data:	(A) = $6E_{16}$ = 01101110_2	(hoogstwaardige byte)
		(Y) = 39_{16} = 00111001_2	(laagstwaardige byte)
		(X) = $0C_{16}$ = 12_{10}	(te zetten bit-positie)
	Resultaat:	(A) = $7E_{16}$ = 01111110_2	(hoogstwaardige byte, bit 12 op 1 gezet)
		(Y) = 39_{16} = 00111001_2	(laagstwaardige byte)
2.	Data:	(A) = $6E_{16}$ = 01101110_2	(hoogstwaardige byte)
		(Y) = 39_{16} = 00111001_2	(laagstwaardige byte)
		(X) = 02_{16} = 2_{10}	(te zetten bit-positie)
	Resultaat:	(A) = $6E_{16}$ = 01101110_2	(hoogstwaardige byte)
		(Y) = $3D_{16}$ = 00111101_2	(laagstwaardige byte, bit 2 op 1 gezet)

```
;
;
;       Titel     : Bit zet
;       Naam      : BITSET
;
;
;       Doel      : Zet een bit aan in een 16-bits woord
;
;
;       Begin     : Register A = hoogstwaardige byte van het woord
;                   Register Y = laagstwaardige byte van het woord
;                   Register X = het bit dat op 1 gezet moet worden
;
;       Eind      : Register A = hoogstwaardige byte van het veranderde woord
;                   Register Y = laagstwaardige byte van het veranderde woord
;
;       Registers : Alle
;
;       Tijd      : 57 cycli
;
;       Grootte   : programma 42  bytes
;                   data       2  bytes

BITSET:
          ; Bewaar het woord

                    sta     VALUE+1
                    sty     VALUE

          ; Zorg ervoor dat het bit-nummer tussen de 0 en de 15 zit

                    txa
                    and     #0FH

          ; Bepaal welk byte en welk bit in die byte gezet moeten worden

                    tax                     ; Bewaar het bit-nummer in X
                    and     #07H            ; De laagste drie bits van het bit-
                                            ; nummer, is het bit in de byte,
                    tay                     ; bewaar dit in Y
                    txa                     ; Herstel het bit-nummer
                    lsr     A               ; Deel door 8 om de byte te bepalen
                    lsr     A
                    lsr     A
                    tax                     ; Bewaar het byte-nummer in X
                                            ; (0 of 1)

          ; Zet het bit op 1

                    lda     VALUE,X         ; Haal de byte op
                    ora     BITMSK,Y        ; Zet het bit op 1
                    sta     VALUE,Y

          ; Geef het resultaat in de registers A en Y terug

                    lda     VALUE+1
                    ldy     VALUE
                    rts


BITMSK:
                    .BYTE   00000001B       ; Bit 0 = 1
```

```
                .BYTE   00000010B    ; Bit 1 = 1
                .BYTE   00000100B    ; Bit 2 = 1
                .BYTE   00001000B    ; Bit 3 = 1
                .BYTE   00010000B    ; Bit 4 = 1
                .BYTE   00100000B    ; Bit 5 = 1
                .BYTE   01000000B    ; Bit 6 = 1
                .BYTE   10000000B    ; Bit 7 = 1


; Data
VALUE:          .BLOCK 2             ; Tijdelijke bewaarplaats voor het
                                     ; datawoord

;
;
; Voorbeeld uitvoering
;
;

SC0701:
                lda     VAL+1        ; Stop het datawoord in A,Y
                ldy     VAL
                ldx     BITN         ; Stop het bit-nummer in X
                jsr     BITSET       ; Zet het bit op 1
                brk                  ; Resultaat van VAL = 5555H
                                     ; BITN = 0F is :
                                     ;  Register A = D5H
                                     ;  Register Y = 55H

                jmp     SC0701


; Testwaarden, verander deze zelf om andere waarden te testen
VAL:            .WORD   5555H
BITN:           .BYTE   0FH


                .END
```

Wis bit - 7B

Wist een opgegeven bit in een 16-bits woord.
Procedure: Het programma gebruikt bits 0 tot en met 2 van register X om te bepalen welk bit gewist moet worden. Bit 3 wordt gebruikt om een bepaalde byte van de oorspronkelijke, woordgrote data te kiezen. Vervolgens voert het een logische AND uit op de gekozen byte, met een masker dat een 0 op de gewenste bit-positie heeft en een 1 op alle andere posities. De maskers met een 0-bit zijn beschikbaar in een tabel.

Gebruikte registers:	Alle
Looptijd:	57 cycli
Programmalengte:	42 bytes
Benodigd datageheugen:	Twee bytes ergens in RAM (beginnende bij adres VALUE)
Speciaal geval:	Bit-posities boven 15 worden mod 16 beschouwd. Bit-positie 16, bij voorbeeld, is voor dit programma hetzelfde als bit-positie 0.

Begincondities

Hoogstwaardige databyte in de accumulator
Laagstwaardige databyte in het register Y
Te wissen bit-nummer in het register X

Eindcondities

Hoogstwaardige byte van het resultaat in de accumulator
Laagstwaardige byte van het resultaat in het register Y

Voorbeelden

1.	Data:	(A) = $6E_{16}$ = 01101110_2	(hoogstwaardige byte)
		(Y) = 39_{16} = 00111001_2	(laagstwaardige byte)
		(X) = $0E_{16}$ = 14_{10}	(te wissen bit-positie)
	Resultaat:	(A) = $2E_{16}$ = 01101110_2	(hoogstwaardige byte, bit 14 gewist)
		(Y) = 39_{16} = 00111001_2	(laagstwaardige byte)
2.	Data:	(A) = $6E_{16}$ = 01101110_2	(hoogstwaardige byte)
		(Y) = 39_{16} = 00111001_2	(laagstwaardige byte)
		(X) = 04_{16} = 4_{10}	(te wissen bit-positie)
	Resultaat:	(A) = $6E_{16}$ = 01101110_2	(hoogstwaardige byte)
		(Y) = 29_{16} = 00101001_2	(laagstwaardige byte, bit 4 gewist)

```
;
;
;
;       Titel      : Bit reset
;       Naam       : BITCLR
;
;
;       Doel       : Zet een bit op 0 in een 16-bits woord
;
;
;       Begin      : Register A = hoogstwaardige byte van het woord
;                    Register Y = laagstwaardige byte van het woord
;                    Register X = het bit dat op 0 gezet moet worden
;
;       Eind       : Register A = hoogstwaardige byte van het veranderde woord
;                    Register Y = laagstwaardige byte van het veranderde woord
;
;       Registers  : Alle
;
;       Tijd       : 57 cycli
;
;       Grootte    : programma 42  bytes
;                    data        2  bytes

BITCLR:
          ; Bewaar het woord

                    sta     VALUE+1
                    sty     VALUE

          ; Zorg ervoor dat het bit-nummer tussen de 0 en de 15 zit

                    txa
                    and     #0FH

          ; Bepaal welke byte en welk bit in die byte op 0 moet worden
          ; gezet

                    tax                     ; Bewaar het bit-nummer in X
                    and     #07H            ; De laagstwaardige 3 bits van het
                                            ; bit-nummer is het gewenste bit
                    tay                     ; Bewaar dit in Y
                    txa                     ; Haal de oorspronkelijke waarde op
                    lsr     A               ; Deel door 8 om het byte te
                    lsr     A               ; bepalen
                    lsr     A
                    tax                     ; Zet dit in X (0 of 1)

          ; Zet het bit op 0

                    lda     VALUE,X         ; Haal het byte op
                    and     BITMSK,Y        ; Zet het bit op 0
                    sta     VALUE,Y

          ; Keer terug met het resultaat in registers A en Y

                    lda     VALUE+1
                    ldy     VALUE
                    rts

; De maskeer waarden
```

```
BITMSK:
                    .BYTE   11111110B
                    .BYTE   11111101B
                    .BYTE   11111011B
                    .BYTE   11110111B
                    .BYTE   11101111B
                    .BYTE   11011111B
                    .BYTE   10111111B
                    .BYTE   01111111B


; Data
VALUE:              .BLOCK 2            ; Bewaarplaats datawoord

;
;
; Voorbeeld uitvoering
;
;

SC0702:
                    lda     VAL+1       ; Laad het woord in A,Y
                    ldy     VAL
                    ldx     BITN        ; Zet het bit-nummer in X
                    jsr     BITCLR      ; Zet dit bit op 0
                    brk                 ; Resultaat van
                                        ;    VAL = 5555H en BITN = 00H  is
                                        ;
                                        ;   Register A = 55H,
                                        ;   Register Y = 54H

                    jmp     SC0702


; Testwaarden, verander deze zelf om andere waarden te testen
VAL:                .WORD   5555H
BITN:               .BYTE   0


                    .END
```

Bit-test - 7C

Zet de overschotvlag naar de waarde van een opgegeven bit in een 16-bits woord.
Procedure: Het programma gebruikt bits 0 tot en met 2 van register X om te bepalen welk bit getest moet worden. Bit 3 wordt gebruikt om een bepaalde byte van de oorspronkelijke, woord-grote data te kiezen. Vervolgens voert het een logische AND uit op de gekozen byte, met een masker dat een 1 op de gewenste bit-positie heeft en een 0 op alle andere. Aangezien het resultaat 0 is als het geteste bit 0 is, wordt de nulvlag naar het complement van het geteste bit gezet. Uiteindelijk zet het programma de overschotvlag naar het complement van de nulvlag. Door deze dubbele inversie krijgt de overschotvlag dezelfde waarde als het geteste bit.

Gebruikte registers:	Alle
Looptijd:	Ongeveer 50 cycli
Programmalengte:	37 bytes
Benodigd datageheugen:	Twee bytes ergens in RAM (beginnende bij adres VALUE)
Speciaal geval:	Bit-posities boven 15 worden mod 16 beschouwd. Bit-positie 16, bij voorbeeld, is voor dit programma hetzelfde als bit-positie 0.

Begincondities

Hoogstwaardige databyte in de accumulator
Laagstwaardige databyte in het register Y
Te zetten bit-nummer in het register X

Eindcondities

Overschotvlag gezet naar waarde van gespecificeerde bit-positie in data.

Voorbeelden

1. Data: $(A) = 6E_{16} = 01101110_2$ (hoogstwaardige byte)
$(Y) = 39_{16} = 00111001_2$ (laagstwaardige byte)
$(X) = 0B_{16} = 11_{10}$ (te testen bit-positie)

Resultaat: Overschotvlag = 1 (waarde van bit 11)

2. Data: $(A) = 6E_{16} = 01101110_2$ (hoogstwaardige byte)
$(Y) = 39_{16} = 00111001_2$ (laagstwaardige byte)
$(X) = 06_{16} = 6_{10}$ (te zetten bit-positie)

Resultaat: Overschotvlag = 0 (waarde van bit 6)

```
;
;
;
;       Titel      : Bit test
;       Naam       : BITTST
;
;
;       Doel       : Test een bit in een 16-bits woord
;
;
;       Begin      : Register A = hoogstwaardige byte van het woord
;                    Register Y = laagstwaardige byte van het woord
;                    Register X = het bit-nummer dat getest moet worden
;
;       Eind       : De overdrachtvlag geeft de waarde van het bit aan
;
;       Registers  : Alle
;
;       Tijd       : Ongeveer 50 cycli
;
;       Grootte    : programma 37  bytes
;                    data       2  bytes

BITTST:
          ; Bewaar het datawoord

                    sta     VALUE+1
                    sty     VALUE

          ; Zorg ervoor dat het bit-nummer tussen de 0 en de 15 zit

                    txa
                    and     #0FH

          ; Bepaal welke byte en welk bit in die byte

                    tax                     ; bewaar het bit-nummer in X
                    and     #07H            ; De laagstwaardige 3 bits van het
                                            ; bit-nummer is het gewenste bit
                    tay                     ; Bewaar dit in Y
                    txa                     ; Haal de oorspronkelijke waarde op
                    lsr     A               ; Deel door 8 om de byte te
                    lsr     A               ; bepalen
                    lsr     A
                    tax                     ; Bewaar het byte-nummer
                                            ; (0 of 1) in X

          ; Maak de nulvlag gelijk aan het complement van het bit

                    lda     VALUE,X         ; Haal de byte op
                    and     BITMSK,Y        ; Haal het bit op
                                            ; Als het bit 0 was dan is
                                            ;   A = 0 en Z = 1
                                            ; anders
                                            ;   A = ongelijk 0 en Z = 0

          ; Maak de overdrachtvlag gelijk aan het complement van de nulvlag

                    clc                     ; Veronderstel dat het bit 0 is
                    bne     EXIT            ; Spring als dat zo was
```

```
                    sec                     ; Anders was het bit 1
EXIT:
                    rts

; Array met masker-waarden

BITMSK:
                    .BYTE   00000001B
                    .BYTE   00000010B
                    .BYTE   00000100B
                    .BYTE   00001000B
                    .BYTE   00010000B
                    .BYTE   00100000B
                    .BYTE   01000000B
                    .BYTE   10000000B


; Data
VALUE:              .BLOCK 2            ; Bewaarplaats datawoord

;
;
; Voorbeeld uitvoering
;
;

SC0703:
                    lda     VAL+1       ; Zet het woord in A,Y
                    ldy     VAL
                    ldx     BITN        ; Zet het bit-nummer in X
                    jsr     BITTST      ; Test het bit
                    brk                 ; Resultaat van
                                        ;   VAL = 5555H en BITN = 01
                                        ;           is
                                        ;   overdrachtvlag = 0

                    jmp     SC0703


; Testwaarden, verander deze zelf om andere waarden te testen
VAL:                .WORD   5555H
BITN:               .BYTE   01H

                    .END
```

Bitveld-uittreksel - 7D

Trekt een veld van bits uit een woord en geeft het terug in de laagstwaardige bit-posities. De lengte van het veld en de begin-positie worden opgegeven.
Procedure: Het programma haalt een masker met het opgegeven aantal 1-bits uit een tabel, schuift het masker naar links om het in lijn te brengen met de opgegeven begin-positie en verkrijgt het veld door het masker logisch te AND-en met de data. Vervolgens wordt het veld genormaliseerd door het zover naar rechts te schuiven dat het op bit-positie 0 begint.

Gebruikte registers: Alle

Looptijd: 34 * BEGINBIT-POSITIE plus 138 cycli vast. De beginbit-positie bepaalt het aantal keren dat het masker naar links en het bit-veld naar rechts geschoven moet worden. Als, bij voorbeeld, het veld in bit 6 begint, is de looptijd

34 * 6 + 138 = 204 + 138 = 342 cycli.

Programmalengte: 134 bytes

Benodigd datageheugen: Zes bytes ergens in RAM voor de index (een byte op adres INDEX), de lengte van het veld (een byte op adres WIDTH), de datawaarde (twee bytes beginnende op adres VALUE) en het masker (twee bytes beginnende op adres MASK).

Speciale gevallen:

1. Een verzoek om een veld dat voorbij het einde van het woord zou komen, zorgt ervoor dat het programma terugkeert met de bits tot en met bit 15. Er is geen omwikkeling (wrap-around). Als de gebruiker bij voorbeeld vraagt om een 10-bits veld, beginnende bij bit 8, geeft het programma slechts 8 bits (bits 8 tot en met 15) terug.
2. Zowel de beginbit-positie als het aantal bits in het veld worden mod 16 behandeld. Dus bit-positie 17 komt overeen met bit-positie 1 en een veld van 20 bits komt overeen met een veld van 4 bits.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Begin (laagste) bit-positie van het veld
Aantal bits in het veld
Laagstwaardige byte van datawaarde
Hoogstwaardige byte van datawaarde

Eindcondities

Hoogstwaardige byte van bit-veld in de accumulator
Laagstwaardige byte van bit-veld in het register Y

Voorbeelden

1. Data: Waarde = $F67C_{16}$ = 1111011001111100_2
Begin bit-positie = 4
Aantal bits in het veld = 8

Resultaat: Bit-veld = 006_{16} = 0000000001100111_2
Er zijn 8 bits uit de oorspronkelijk data gehaald, beginnend bij bit 4 (dus bits 4 tot en met 11).

2. Data: Waarde = $A2D4_{16}$ = 1010001011010100_2
Beginbit-positie = 6
Aantal bits in het veld = 5

Resultaat: Bit-veld = 000B16 = 0000000000001011_2
Er zijn 5 bits uit de oorspronkelijke data gehaald, beginnend

```
;
;
;
;     Titel     : Bit-veld extraktor
;     Naam      : BFE
;
;
;     Doel      : Sloop een aantal bits uit een 16-bits woord en geef deze
;                 bits terug genormaliseerd naar bit 0
;
;
;     Begin     : Top van de stack
;                   Laagstwaardige byte van het terugkeeradres
;                   Hoogstwaardige byte van het terugkeeradres
;                   Nummer van het eerste bit van het veld (0..15)
;                   Het aantal bits in het veld
;                   Laagstwaardige byte van het datawoord
;                   Hoogstwaardige byte van het datawoord
;
;     Eind      : Register A = hoogstwaardige byte van het veld
;                 Register Y = laagstwaardige byte van het veld
;
;     Registers : Alle
;
;     Tijd      : 138 basis- plus (34 * beginpositie van het bit-veld)
;                 cycli
;
;     Grootte   : programma 134  bytes
;                 data        6  bytes
```

```
BFE:
        ; Bewaar het terugkeeradres in Y,X

                pla
                tay
                pla
                tax

        ; Haal het index van het eerste bit van het veld op

                pla
                and     #0FH            ; Zorg ervoor dat het tussen de 0
                                        ; en de 1 zit
                sta     INDEX           ; Bewaar deze index

        ; Haal het aantal bits van het veld op (transponeer 1..WIDTH
        ; naar 0..WITH-1)

                pla
                sec
                sbc     #1              ; Trek er 1 vanaf
                and     #0FH            ; Zorg ervoor dat het tussen de 0
                                        ; en de 15 zit
                sta     WIDTH           ; Bewaar de lengte

        ; Haal het datawoord op
                pla
                sta     VALUE
                pla
                sta     VALUE+1

        ; Zet het terugkeeradres weer op de stack

                txa
                pha
                tya
                pha

        ; Construeer het bit-masker
        ;   Gebruik de WIDTH parameter als index in de tabel

                lda     WIDTH
                asl     A               ; Maal 2 want de maskers hebben
                                        ; de lengte van een woord
                tay
                lda     MSKARY,Y
                sta     MASK
                iny
                lda     MSKARY,Y
                sta     MASK+1

        ; Schuif het masker INDEX maal naar links om het precies op de
        ; goede plaats te zetten

                ldy     INDEX
                beq     GETFIELD        ; Spring als INDEX = 0
SHFTLP:
                asl     MASK            ; Schuif een woord naar, 1
                rol     MASK+1          ; positie naar links
                dey
                bne     SHFTLP          ; Ga door tot INDEX = 0
```

```
GETFLD:
        ; Haal het veld op door logische AND-operatie op VALUE en MASK

                lda     VALUE
                and     MASK            ; AND met de laagstwaardige byte
                sta     VALUE           ; Bewaar dat
                lda     VALUE+1
                and     MASK+1          ; AND met de hoogstwaardige byte
                sta     VALUE+1         ; Bewaar dat

        ; Schuif het veld INDEX maal naar rechts om het bij 0 te laten
        ; beginnen

                ldy     INDEX
                beq     EXIT            ; Spring als INDEX = 0

NORMLP:
                lsr     VALUE+1         ; Schuif woord 1 bit naar rechts
                ror     VALUE
                dey
                bne     NORMLP          ; Ga door tot we klaar zijn


EXIT:
                ldy     VALUE
                lda     VALUE+1
                rts

        ; Array met masker-waarden, om het juiste masker te creeren

MSKARY:
                .WORD   0000000000000001B
                .WORD   0000000000000011B
                .WORD   0000000000000111B
                .WORD   0000000000001111B
                .WORD   0000000000011111B
                .WORD   0000000000111111B
                .WORD   0000000001111111B
                .WORD   0000000011111111B
                .WORD   0000001111111111B
                .WORD   0000011111111111B
                .WORD   0000111111111111B
                .WORD   0001111111111111B
                .WORD   0011111111111111B
                .WORD   0111111111111111B
                .WORD   1111111111111111B


; Data
INDEX:          .BLOCK  1               ; Index in het woord
WIDTH:          .BLOCK  1               ; Lengte van het veld
VALUE:          .BLOCK  2               ; Data woord
MASK:           .BLOCK  2               ; Masker

;
;
; Voorbeeld uitvoering
;
;
```

```
SC0704:
        ; Zet het datawoord op de stack

                lda     VAL+1
                pha
                lda     VAL
                pha

        ; Zet de lengte van het veld op de stack

                lda     NBITS
                pha

        ; Zet de index naar het eerste bit op de stack

                lda     POS
                pha
                jsr     BFE             ; Haal het veld eruit
                brk                     ; Resultaat voor
                                        ;
                                        ;  VAL = 1234H, NBITS = 4, POS = 4
                                        ; is
                                        ;  Register A = 0
                                        ;  Register Y = 3
                jmp     SC0704


; Test-data

VAL:            .WORD   01234H
NBITS:          .BYTE   4
POS:            .BYTE   4


                .END
```

Bit-veld-tussenvoeging - 7E

Voegt een veld van bits in een woord. De lengte van het veld en de begin (laagste) positie ervan worden opgegeven.
Procedure: Het programma haalt een masker met het opgegeven aantal 0-bits uit een tabel en schuift het masker en het bit-veld naar links om ze in lijn te brengen met de opgegeven beginpositie. Het voert dan een logische AND uit van het masker op het oorspronkelijke datawoord, waardoor de gewenste bit-posities gewist worden. Vervolgens zorgt een logische OR van het resultaat met het geschoven bit-veld ervoor dat het veld wordt tussengevoegd.

Gebruikte registers: Alle

Looptijd: 31 * BEGIN BIT-POSITIE plus 142 cycli vast. De beginbit-positie bepaalt het aantal keren dat het masker en het bit-veld naar links geschoven moeten worden. Als, bij voorbeeld, het veld vanaf bit 10 tussengevoegd moet worden, is de looptijd

31 * 10 + 142 = 310 + 142 = 452 cycli

Programmalengte: 130 bytes

Benodigd datageheugen: Acht bytes ergens in RAM voor de index (een byte op adres INDEX), de lengte van het veld (een byte op adres WIDTH), de tussen te voegen waarde (twee bytes beginnende op INSVAL), de datawaarde (twee bytes beginnende op adres VALUE) en het masker (twee bytes beginnende op adres MASK).

Speciale gevallen:

1. Een poging een veld tussen te voegen dat voorbij het einde van het woord zou komen, zorgt ervoor dat het programma slechts de bits tot en met bit 15 tussenvoegt. Er is dus geen omwikkeling (wrap-around). Als de gebruiker bij voorbeeld een 6-bits veld dat bij bit 14 begint wil tussenvoegen, worden slechts twee bits (bits 14 en 15) vervangen.
2. Zowel de beginbit-positie als het aantal bits in het veld worden mod 16 behandeld. Dus bit-positie 17 is gelijk aan bit-positie 1 en een veld van 20 bits is gelijk aan een veld van 4 bits.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Begin (laagste) bit-positie van het veld
Aantal bits in het veld
Laagstwaardige byte van het bit-veld (in te voegen waarde)
Hoogstwaardige byte van het bit-veld (in te voegen waarde)
Laagstwaardige byte van datawaarde
Hoogstwaardige byte van datawaarde

Eindcondities

Hoogstwaardige byte van bit-veld in accumulator
Laagstwaardige byte van bit-veld in register Y

Het resultaat is de oorspronkelijke datawaarde, waarbij het bit-veld is tussengevoegd, beginnende bij de opgegeven bit-positie.

Voorbeelden

1. Data:
   Waarde = $F67C_{16}$ = 1111011001111100_2
   Beginbit-positie = 4
   Aantal bits in het veld = 8
   Bit-veld = $008B_{16}$ = 0000000010001011_2

   Resultaat:
   Waarde met tussengevoegd bit-veld = $F8BC_{16}$ = 1111000101111100_2
   Het 8-bits veld is vanaf bit-positie 4 in de oorspronkelijke waarde tussengevoegd (dus bits 4 tot en met 11)

2. Data:
   Waarde = $A2D4_{16}$ = 1010001011010100_2
   Beginbit-positie = 6
   Aantal bits in het veld = 5
   Bit-veld = 0015_{16} = 0000000000010101_2

   Resultaat:
   Waarde met tussengevoegd bit-veld = $A554_{16}$ = 1010010101010100_2
   Het 5-bits veld is vanaf bit 6 in de oorspronkelijke waarde tussengevoegd (dus bits 6 tot en met 10). Die vijf bits waren 01011_2 ($0B_{16}$) en zijn nu 10101_2 (15_{16}).

```
;
;
;
;       Titel       : Bit-veld-invoeger
;       Naam        : BFI
;
;
;       Doel        : Voeg een aantal bits genormaliseerd naar bit 0 toe aan
;                     een 16-bits woord
;
;
;       Begin       : Top van de stack
;                       Laagstwaardige byte van het terugkeeradres
;                       Hoogstwaardige byte van het terugkeeradres
;                       Nummer van het eerste bit van het veld (0..15)
;                       Het aantal bits in het veld
;                       Laagstwaardige byte van het invoegwoord
;                       Hoogstwaardige byte van het invoegwoord
;                       Laagstwaardige byte van het datawoord
;                       Hoogstwaardige byte van het datawoord
;
;       Eind        : Register A = hoogstwaardige byte van het veranderde woord
;                     Register Y = laagstwaardige byte van het veranderde woord
```

```
;
;       Registers : Alle
;
;       Tijd      : 142 basis- plus (31 * begin positie van het bit- veld)
;                   cycli
;
;       Grootte   : programma 130  bytes
;                   data        8  bytes

BFI:
            ; Bewaar het terugkeeradres in Y,X

                    pla
                    tay
                    pla
                    tax

            ; Haal het laagste bit-nummer van het veld op

                    pla
                    and     #0FH            ; Modulo 15
                    sta     INDEX           ; Bewaar het in INDEX

            ; Haal het aantal bits in het veld op
            ;  (1..WIDTH --> 0..WIDTH-1)

                    pla
                    sec
                    sbc     #1              ; Trek er 1 vanaf
                    and     #0FH            ; Modulo 15
                    sta     WIDTH           ; Bewaar deze waarde in WIDTH

            ; Haal de in te vullen waarde op

                    pla
                    sta     INSVAL
                    pla
                    sta     INSVAL+1

            ; Haal het datawoord op

                    pla
                    sta     VALUE
                    pla
                    sta     VALUE+1

            ; Zet het terugkeeradres terug op de stack

                    txa
                    pha
                    tya
                    pha

            ; Constueer een bit-masker
            ;  Gebruik de WIDTH parameter als index in het array

                    lda     WIDTH           ; Haal de 8-bits waarde WIDTH op
                    asl     A               ; Maal 2 omdat de maskers 16-bits
                                            ; woorden zijn
                    tay
                    lda     MSKARY,Y
```

```
        sta     MASK
        iny
        lda     MSKARY,Y
        sta     MASK+1

; Schuif MASK en het in te vullen woord INDEX maal naar links om
; ze op gelijke hoogte te brengen met het begin van het veld

        ldy     INDEX
        beq     INSERT          ; Spring als INDEX = 0
SHFTLP:
        sec                     ; Vul het masker aan met 1
        rol     MASK            ; Roteer MASK 1 byte naar links
        rol     MASK+1
        asl     INSVAL          ; Schuif de in te vullen
        rol     INSVAL+1        ; waarde naar links, vul aan met 0
        dey
        bne     SHFTLP          ; Ga door tot INDEX = 0


INSERT:
; Gebruik het masker om de bits in het veld op 0 te zetten
; Vul daarna met een logische OR-opdracht de in te vullen waarde
; in

        lda     VALUE
        and     MASK            ; AND van laagstwaardige byte van
                                ; VALUE met MASK
        ora     INSVAL
        tay                     ; Register Y = laagstwaardige byte
                                ; van de nieuwe waarde
        lda     VALUE+1
        and     MASK+1          ; AND van hoogstwaardige byte van
                                ; VALUE met MASK
        ora     INSVAL+1        ; Register A = hoogstwaardige byte
                                ; van de nieuwe waarde

; Keer terug

        rts


MSKARY:
; Array van maskerwaarden om het masker te maken

        .WORD   1111111111111110B
        .WORD   1111111111111100B
        .WORD   1111111111111000B
        .WORD   1111111111110000B
        .WORD   1111111111100000B
        .WORD   1111111111000000B
        .WORD   1111111110000000B
        .WORD   1111111100000000B
        .WORD   1111111000000000B
        .WORD   1111110000000000B
        .WORD   1111100000000000B
        .WORD   1111000000000000B
        .WORD   1110000000000000B
        .WORD   1100000000000000B
        .WORD   1000000000000000B
        .WORD   0000000000000000B
```

```
INDEX:          .BLOCK  1           ; Index in het woord
WIDTH:          .BLOCK  1           ; Lengte van het veld
INSVAL:         .BLOCK  2           ; De in te vullen waarde
VALUE:          .BLOCK  2           ; Het datawoord
MASK:           .BLOCK  2           ; Het masker


;
;
; Voorbeeld uitvoering
;
;

SC0705:
        ; Zet het datawoord op de stack

                lda     VAL+1
                pha
                lda     VAL
                pha

        ; Zet de in te vullen waarde op de stack
                lda     VALINS+1
                pha
                la      VALINS
                pha

        ; Zet het aantal bits op de stack

                lda     NBITS
                pha

        ; Zet de beginpositie van het veld op de stack

                lda     POS
                pha
                jsr     BFI             ; Vul de waarde in
                brk                     ; Resultaat voor
                                        ;
                                        ;  VAL    = 1234H
                                        ;  VALINS = 0EH
                                        ;  NBITS  = 4
                                        ;  POS    = 0CH
                                        ;
                                        ;     is
                                        ;
                                        ;  Register A = E2H
                                        ;  Register Y = 34H

                jmp     SC0705


; Test-data

VAL:            .WORD   01234H
VALINS:         .WORD   0EH
NBITS:          .BYTE   04H
POS:            .BYTE   0CH
                .END
```

Meervoudige precisie rekenkundige verschuiving naar rechts - 7F

Verschuift een uit meerdere bytes bestaand getal een opgegeven aantal bit-posities rekenkundig naar rechts. De lengte van het getal (in bytes) is 255 of minder. De overdrachtvlag wordt gezet naar de waarde van het bit dat het laatst uit de meest rechtse bit-positie is geschoven. Het getal is met de laagstwaardige byte op het laagste adres opgeslagen.
Procedure: Het programma pakt het tekenbit van de hoogstwaardige byte, schuift dat bit in de overdrachtvlag en roteert dan het hele getal een bit naar rechts, beginnend met de hoostwaardige byte. Het herhaalt deze handeling gedurende een opgegeven aantal verschuivingen.

Gebruikte registers: Alle

Looptijd: AANTAL SCHUIVINGEN * (18 + 18 * LENGTE VAN GETAL IN BYTES) + 85 cycli.
Als, bij voorbeeld, AANTAL SCHUIVINGEN = 6 en LENGTE VAN GETAL IN BYTES = 8, dan is de looptijd

6 * (18 + 18 * 8) + 85 = 6 * 162 + 85 = 1057 cycli.

Programmalengte: 69 bytes

Benodigd datageheugen: Drie bytes ergens in RAM plus twee bytes op pagina 0. De drie bytes ergens in RAM dienen voor de tijdelijke opslag van het aantal schuivingen (een byte op adres NBITS), de lengte van het getal (een byte op adres LENGTH) en de hoogstwaardige byte van het getal (een byte op adres MSB). De twee bytes op pagina 0 bevatten een pointer naar het getal (beginnend op adres PTR, $00D0_{16}$ in de listing).

Speciale gevallen:

1. Als de lengte van het getal nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.
2. Als het aantal schuivingen nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Aantal schuivingen (bit-posities)
Lengte van het getal in bytes
Laagstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)
Hoogstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)

Eindcondities

Getal het opgegeven aantal bit-posities rekenkundig naar rechts geschoven. Het oorspronkelijke tekenbit is naar rechts verlengd. De overdrachtvlag is gezet overeenkomstig het laatste bit dat van de meest rechtse positie is geschoven (of gewist als of het aantal schuivingen of de lengte van het getal nul is).

Voorbeelden

1. Data: Lengte van het getal (in bytes) = 8
   Getal = $85A4C719FE06741E_{16}$
   Aantal schuivingen = 04

   Resultaat: Geschoven getal = $F85A4C719FE06741_{16}$
   Dit is het oorspronkelijke getal vier bits rekenkundig naar rechts geschoven (de vier hoogstwaardige bits nemen zodoende alle de waarde van het oorspronkelijke tekenbit over, dat 1 was).
   Overdrachtvlag = 1, aangezien het laatste bit dat vanuit de meest rechtse positie werd geschoven, 1 was.

2. Data: Lengte van getal (in bytes) = 04
   Getal = $3F6A42D3_{16}$
   Aantal schuivingen = 03

   Resultaat: Geschoven getal = $07ED485A_{16}$
   Dit is het oorspronkelijke getal drie bits rekenkundig naar rechts geschoven (de drie hoogstwaardige bits nemen zodoende alle de waarde van het oorspronkelijke tekenbit over, dat 0 was).
   Overdrachtvlag = 0, aangezien het laatste bit dat vanuit de meest rechtse positie werd geschoven, 0 was.

```
;
;
;
;     Titel     : Multi-precieze rekenkundige verschuiving naar rechts
;     Naam      : MPASR
;
;
;     Doel      : Schuif een multi-byte operand N bits naar rechts
;
;
;     Begin     : Top van de stack
;                   Laagstwaardige byte van terugkeeradres
;                   Hoogstwaardige byte van terugkeeradres
;                   Aantal bits dat geshift moet worden
;                   Lengte van de operand in bytes
;                   Laagstwaardige byte van het adres van de operand
;                   Hoogstwaardige byte van het adres van de operand
;
;                 De operand is opgeslagen met ARRAY[0] als zijn
;                 laagstwaardige byte en ARRAY[LENGTH-1] als zijn
;                 hoogstwaardige byte
```

```
;
;     Eind       : Operand is naar rechts geshift met het hoogstwaardige
;                  bit gepropageerd
;                  Overdrachtvlag = Laatste bit dat is geshift vanaf de
;                  laagstwaardige positie
;
;     Registers  : Allemaal
;
;     Tijd       : 85 basis plus
;                  ((18 * lengte) + 18) cycli per verschuiving
;
;     Grootte    : programma 69  bytes
;                  data        3  bytes plus
;                              2  bytes op pagina 0

PTR:                .EQU   0D0H          ; Pointer naar operand op pagina 0

MPASR:
          ; Bewaar het terugkeeradres

                    pla
                    tay
                    pla
                    tax

          ; Haal het aantal bits van de stack af

                    pla
                    sta     NBITS

          ; Haal de lengte van de operand van de stack af

                    pla
                    sta     LENGTH

          ; Haal het beginadres van de operand van de stack af

                    pla
                    sta     PTR
                    pla
                    sta     PTR+1

          ; Zet het terugkeeradres terug op de stack

                    txa
                    pha
                    tya
                    pha

          ; Initialisatie

                    clc                  ; Maak de overdrachtvlag schoon
                    lda     LENGTH
                    beq     EXIT         ; Keer terug als lengte = 0
                    lda     NBITS
                    beq     EXIT         ; Keer terug als aantal bits
                                         ; gelijk aan 0 is  (met
                                         ;   overdrachtvlag = 0)

          ; Verlaag de pointer zodat de lengte-byte zowel gebruikt kan
          ; worden als teller en als index
```

```
                lda     PTR
                bne     MPASR1
                dec     PTR+1           ; Verlaag de hoogstwaardige byte
                                        ; als er geleend moet worden
MPASR1:
                dec     PTR             ; Verlaag altijd de laagstwaardige
                                        ; byte

        ; Doe dit net zolang tot alle verschuivingen zijn uitgevoerd

                ldy     LENGTH
                lda     (PTR),Y         ; Haal de hoogstwaardige byte op
                sta     MSB             ; Bewaar hem voor het teken


ASRLP:
                lda     MSB             ; Haal de hoogstwaardige byte weer
                                        ; op
                asl     A               ; Schuif bit 7 in het resultaat als
                                        ; als teken-bit
                ldy     LENGTH          ; Y = index naar de laatste byte en
                                        ; de teller


LOOP:
        ; Schuif 1 bit naar rechts

                lda     (PTR),Y         ; Haal de volgende byte op
                ror     A               ; Roteer   BIT 7 = overdracht
                                        ;          overdracht = BIT 0
                sta     (PTR),Y         ; Zet de nieuwe waarde terug
                dey                     ; Verlaag de teller
                bne     LOOP            ; Ga door tot we alle byte langs
                                        ; zijn geweest

        ; Verlaag het aantal shifts met 1

                dec     NBITS           ; Verlaag de teller
                bne     ASRLP           ; Ga door zolang er nog shifts
                                        ; uitgevoerd moeten worden

EXIT:
                rts


; Data
NBITS:          .BLOCK  1               ; Aantal bits
LENGTH:         .BLOCK  1               ; Lengte van de operand
MSB:            .BLOCK  1               ; hoogstwaardige byte

;
;
; Voorbeeld uitvoering
;
;

SC0706:
        ; Zet het beginadres van de operand op de stack

                lda     AYADR+1
                pha
                lda     AYADR
```

```
        pha

; Zet de lengte van de operand op de stack

        lda     #SZAY
        pha

; Zet het aantal shifts op de stack

        lda     SHIFTS
        pha
        jsr     MPASR           ; shift
        brk                     ; Resultaat van
                                ;
                                ;  AY = EDCBA987654321H  4 bits
                                ;
                                ;         is
                                ;
                                ;  AY = FEDCBA987654321H
                                ;  overdrachtvlag = 0
                                ;
                                ;   AY     = 032H
                                ;   AY + 1 = 054H
                                ;   AY + 2 = 076H
                                ;   AY + 3 = 098H
                                ;   AY + 4 = 0BAH
                                ;   AY + 5 = 0DCH
                                ;   AY + 6 = 0FEH


        jmp     SC0706

; Data
SZAY:           .EQU    7               ; Lengte van de operand
SHIFTS:         .BYTE   4               ; Aantal shifts
AYADR:          .WORD   AY              ; Beginadres van de operand
AY:             .BYTE 21H, 43H, 65H, 87H, 0A9H, 0CBH, 0EDH

                .END
```

Meervoudige precisie rekenkundige verschuiving naar links - 7G

Verschuift een uit meerdere bytes bestaand getal een opgegeven aantal bit-posities logisch naar links. De lengte van het getal (in bytes) is 255 of minder. De overdrachtvlag wordt gezet naar de waarde van het bit dat het laatst uit de meest linkse bit-positie is geschoven. Het getal is met de laagstwaardige byte op het laagste adres opgeslagen.
Procedure: Het programma wist de overdrachtvlag eerst (om er een 0-bit in te plaatsen) en roteert dan het hele getal een bit naar links, beginnend met de laagstwaardige byte. Het herhaalt deze handeling gedurende het opgegeven aantal schuivingen.

Gebruikte registers: Alle

Looptijd: AANTAL SCHUIVINGEN * (16 + 20 * LENGTE VAN GETAL IN BYTES) + 73 cycli
Als, bij voorbeeld, AANTAL SCHUIVINGEN = 4 en LENGTE VAN GETAL IN BYTES = 6, dan is de looptijd

4 * (6 + 20 * 6) + 73 = 4 * 136 + 73 = 617 cycli.

Benodigd datageheugen: Twee bytes ergens in RAM, plus twee bytes op pagina 0. De twee bytes ergens in RAM dienen voor de tijdelijke opslag van het aantal schuivingen (een byte op adres NBITS) en de lengte van het getal in bytes (een byte op adres LENGTH). De twee bytes op pagina 0 bevatten een pointer naar het getal (beginnend op adres PTR, $00D0_{16}$ in de listing).

Speciale gevallen:

1. Als de lengte van het getal nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.
2. Als het aantal schuivingen nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Aantal schuivingen (bit-posities)
Lengte van het getal in bytes
Laagstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)
Hoogstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)

Eindcondities

Getal het opgegeven aantal bit-posities rekenkundig naar links geschoven (de laagstwaardige bit-posities zijn met nullen gevuld). De overdrachtvlag is gezet overeenkomstig het laatste bit dat van de meest linkse positie is geschoven (of gewist als of het aantal schuivingen of de lengte van het getal nul is).

Voorbeelden

1. Data: Lengte van het getal (in bytes) = 8
   Getal = $85A4C719FE06741E_{16}$
   Aantal schuivingen = 04

   Resultaat: Geschoven getal = $5A4C719FE06741E0_{16}$
   Dit is het oorspronkelijke getal vier bits rekenkundig naar links geschoven; de vier laagstwaardige bits zijn alle gewist.
   Overdrachtvlag = 0, aangezien het laatste bit dat van de meest linkse positie werd geschoven, 0 was.

2. Data: Lengte van getal (in bytes) = 04
   Getal = $3F6A42D3_{16}$
   Aantal schuivingen = 03

   Resultaat: Geschoven getal = $FB521698_{16}$
   Dit is het oorspronkelijke getal drie bits rekenkundig naar links geschoven; de drie laaggstwaardige bits zijn alle gewist.
   Overdrachtvlag = 1, aangezien het bit dat het laatst vanuit de meest linkse positie werd geschoven, 1 was.

```
;
;
;
;     Titel      : Multi-precieze logische shift naar links
;     Naam       : MPLSL
;
;
;     Doel       : Verschuif een multi-byte operand N bits naar links
;
;
;     Begin      : Top van de stack
;                    Laagstwaardige byte van terugkeeradres
;                    Hoogstwaardige byte van terugkeeradres
;                    Aantal bits dat verschoven moet worden
;                    Lengte van de operand in bytes
;                    Laagstwaardige byte van het adres van de operand
;                    Hoogstwaardige byte van het adres van de operand
;
;                  De operand is opgeslagen met ARRAY[0] als zijn
;                  laagstwaardige byte en ARRAY[LENGTH-1] als zijn
;                  hoogstwaardige byte
```

```
;       Eind       : Operand is naar links geshift met het laagstwaardige bit
;                    gepropageerd
;                    Overdrachtvlag = Laatste bit dat is geshift vanaf de
;                    hoogstwaardige positie
;
;       Registers  : Alle
;
;       Tijd       : 73 basis plus ((20 * lengte) + 16) cycli per verschuiving
;
;       Grootte    : programma 54  bytes
;                    data        2  bytes plus
;                                2  bytes op pagina 0

PTR:                .EQU   0D0H          ; Pointer naar operand op pagina 0

MPLSL:
          ; Bewaar het terugkeeradres

                    pla
                    tay
                    pla
                    tax

          ; Haal het aantal bits van de stack af

                    pla
                    sta     NBITS

          ; Haal de lengte van de operand van de stack af

                    pla
                    sta     LENGTH

          ; Haal het beginadres van de operand van de stack af

                    pla
                    sta     PTR
                    pla
                    sta     PTR+1

          ; Zet het terugkeeradres terug op de stack

                    txa
                    pha
                    tya
                    pha

          ; Initialisatie

                    clc                  ; Maak de overdrachtvlag schoon
                    lda     LENGTH
                    beq     EXIT         ; Keer terug als lengte = 0
                    lda     NBITS
                    beq     EXIT         ; Keer terug als aantal bits gelijk
                                         ; aan 0 is  ( overdrachtvlag = 0)

LSLLP:
          ; Voer dit uit zolang er nog shifts uitgevoerd moeten worden

                    ldy     #0           ; Y = Index naar het laagstwaardige
                                         ; byte van de operand
```

```
                ldx     LENGTH          ; X = Aantal bytes
                clc                     ; Maak de overdracht schoon

LOOP:
        ; Schuif 1 bit naar links

                lda     (PTR),Y         ; Haal de volgende byte op
                rol     A               ; Roteer    BIT 0 = overdracht
                                        ;           overdracht = BIT 7
                sta     (PTR),Y         ; Zet de nieuwe waarde terug
                iny                     ; Verhoog de index
                dex                     ; Verlaag de teller
                bne     LOOP            ; Ga door zolang we nog niet alle
                                        ; bytes gehad hebben

        ; Verlaag het aantal shifts met 1

                dec     NBITS
                bne     LSLLP           ; Ga door zolang er nog shifts
                                        ; gedaan moeten worden

EXIT:
                rts


; Data
NBITS:          .BLOCK  1               ; Aantal bits
LENGTH:         .BLOCK  1               ; Lengte van de operand

;
;
; Voorbeeld uitvoering
;
;

SC0707:
        ; Zet het beginadres van de operand op de stack

                lda     AYADR+1
                pha
                lda     AYADR
                pha

        ; Zet de lengte van de operand op de stack

                lda     #SZAY
                pha

        ; Zet het aantal shifts op de stack

                lda     SHIFTS
                pha
                jsr     MPLSL           ; shift
                brk                     ; Resultaat van
                                        ;
                                        ;  AY = EDCBA987654321H  4 BITS
                                        ;
                                        ;     is
                                        ;
                                        ;  AY = DCBA9876543210H
                                        ;  overdrachtvlag = 0
                                        ;
```

```
                                        ; in het geheugen opgeslagen als:
                                        ;
                                        ;    AY     = 010H
                                        ;    AY + 1 = 032H
                                        ;    AY + 2 = 054H
                                        ;    AY + 3 = 076H
                                        ;    AY + 4 = 098H
                                        ;    AY + 5 = 0BAH
                                        ;    AY + 6 = 0DCH

                jmp     SC0707

; Data
SZAY:           .EQU    7               ; Lengte van de operand
SHIFTS:         .BYTE   4               ; Aantal shifts
AYADR:          .WORD   AY              ; Beginadres van de operand
AY:             .BYTE 21H, 43H, 65H, 87H, 0A9H, 0CBH, 0EDH

                .END
```

Meervoudige precisie logische verschuiving naar rechts - 7H

Verschuift een uit meerdere bytes bestaand getal een opgegeven aantal bit-posities logisch naar rechts. De lengte van het getal (in bytes) is 255 of minder. De overdrachtvlag wordt gezet naar de waarde van het laatste bit dat vanuit de meest rechtse bit-positie is geschoven. Het getal is met de laagstwaardige byte op het laagste adres opgeslagen.
Procedure: Het programma wist eerst de overdrachtvlag (door met een 0-bit te vullen) en roteert dan het hele getal, beginnend met de hoogstwaardige byte, een bit naar rechts. Het herhaalt deze handeling gedurende een opgegeven aantal verschuivingen.

Gebruikte registers: Alle

Looptijd: AANTAL SCHUIVINGEN * (14 + 18 * LENGTE VAN GETAL IN BYTES) + 80 cycli.
Als, bij voorbeeld, AANTAL SCHUIVINGEN = 4 en LENGTE VAN GETAL IN BYTES = 8 (d.w.z. een 4-bits verschuiving van een 8-bytes getal), dan is de looptijd

4 * (14 + 18 * 8) + 80 = 4 * 158 + 80 = 712 cycli.

Programmalengte: 59 bytes

Benodigd datageheugen: Twee bytes ergens in RAM, plus twee bytes op pagina 0. De drie bytes ergens in RAM dienen voor de tijdelijke opslag van het aantal verschuivingen (een byte op adres NBITS) en de lengte van het getal in bytes (een byte op adres LENGTH). De twee bytes op pagina 0 bevatten een pointer naar het getal (beginnende op adres PTR, $00D0_{16}$ in de listing)

Speciale gevallen:

1. Als de lengte van het getal nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.
2. Als het aantal verschuivingen nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Aantal verschuivingen (bitposities)
Lengte van het getal in bytes
Laagstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)
Hoogstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)

Eindcondities

Getal het aantal opgegeven bit-posities logisch naar rechts verschoven (de hoogstwaardige bit-posities zijn met nullen gevuld). De overdrachtvlag is gezet overeenkomstig het laatste bit dat vanuit de meest rechtse positie is verschoven (of gewist als of het aantal verschuivingen of de lengte van het getal nul is).

Voorbeelden

1. Data: Lengte van het getal (in bytes) = 8
Getal = $85A4C719FE06741E_{16}$
Aantal verschuivingen = 04

Resultaat: Verschoven getal = $085A4C719FE06741_{16}$
Dit is het oorspronkelijke getal vier bits logisch naar rechts verschoven; de vier hoogstwaardige bits zijn gewist.
Overdrachtvlag = 1, aangezien het laatste bit dat vanuit de meest rechtse positie werd verschoven, 1 was.

2. Data: Lengte van getal (in bytes) = 04
Getal = $3F6A42D3_{16}$
Aantal verschuivingen = 03

Resultaat: Verschoven getal = $07ED485A_{16}$
Dit is het oorspronkelijke getal drie bits logisch naar rechts verschoven; de drie hoogstwaardige bits zijn gewist.
Overdrachtvlag = 0, aangezien het laatste bit dat vanuit de meest rechtse positie werd geschoven, 0 was.

```
;
;
;
;     Titel     : Multi-precieze logische verschuiving naar rechts
;     Naam      : MPLSR
;
;
;     Doel      : Verschuif een multi-byte operand N bits naar rechts
;
;
;     Begin     : Top van de stack
;                   Laagstwaardige byte van terugkeeradres
;                   Hoogstwaardige byte van terugkeeradres
;                   Aantal bits dat geshift moet worden
;                   Lengte van de operand in bytes
;                   Laagstwaardige byte van het adres van de operand
;                   Hoogstwaardige byte van het adres van de operand
;
;                 De operand is opgeslagen met ARRAY[0] als zijn
;                 laagstwaardige byte en ARRAY[LENGTH-1] als zijn
;                 hoogstwaardige byte

;     Eind      : Operand is naar rechts geshift met de hoogstwaardige
;                 bits op 0 gezet
```

```
;                 Overdrachtvlag = Laatste bit dat is verschoven vanaf de
;                 laagstwaardige positie
;
;     Registers : Alle
;
;     Tijd      : 85 basis plus ((18 * lengte) + 14) cycli per shift
;
;     Grootte   : programma 59  bytes
;                 data       2  bytes plus
;                            2  bytes op pagina 0

PTR:                .EQU   0D0H          ; Wijzer naar operand op pagina 0

MPLSR:
          ; Bewaar het terugkeeradres

                    pla
                    tay
                    pla
                    tax

          ; Haal het aantal bits van het stack

                    pla
                    sta     NBITS

          ; Haal de lengte van de operand van de stack

                    pla
                    sta     LENGTH

          ; Haal het beginadres van de operand van de stack

                    pla
                    sta     PTR
                    pla
                    sta     PTR+1

          ; Zet het terugkeeradres terug op de stack

                    txa
                    pha
                    tya
                    pha

          ; Initialiseer

                    clc                  ; Maak de overdracht schoon
                    lda     LENGTH
                    beq     EXIT         ; Keer terug als LENGTH = 0
                    lda     NBITS
                    beq     EXIT         ; Keer terug als NBITS = 0
                                         ; met overdrachtvlag = 0

          ; Verlaag de pointer met 1 zodat de lengte-byte zowel gebruikt
          ; kan worden als index als voor een teller

                    lda     PTR
                    bne     MPLSR1
                    dec     PTR+1        ; Verlaag als er geleend moet
                                         ; worden
```

```
MPLSR1:
                dec     PTR             ; Verlaag deze altijd
LSRLP:
                ldy     LENGTH          ; Y = index naar hoogstwaardige
                                        ; byte en teller
                clc                     ; Maak de overdrachtvlag schoon

LOOP:
        ; Schuif 1 bit naar rechts

                lda     (PTR),Y         ; Haal de volgende byte op
                ror     A               ; Roteer    BIT 7 = overdracht
                                        ;           overdracht = BIT 0
                sta     (PTR),Y         ; Zet de nieuwe waarde terug
                dey                     ; Verlaag de teller
                bne     LOOP            ; Ga door tot alle bytes gedaan
                                        ; zijn

                dec     NBITS           ; Verlaag de schuif-teller
                bne     LSRLP           ; Ga door tot we klaar zijn

EXIT:
                rts


; Data
NBITS:          .BLOCK  1               ; Aantal bits
LENGTH:         .BLOCK  1               ; Lengte van de operand

;
;
; Voorbeeld uitvoering
;
;

SC0708:
        ; Zet het beginadres van de operand op de stack

                lda     AYADR+1
                pha
                lda     AYADR
                pha

        ; Zet de lengte van de operand op de stack

                lda     #SZAY
                pha

        ; Zet het aantal verschuivingen op de stack

                lda     SHIFTS
                pha
                jsr     MPLSR           ; Schuif
                brk                     ; Resultaat van
                                        ;
                                        ;  AY = EDCBA987654321H   4 BITS
                                        ;
                                        ;         is
                                        ;
                                        ;  AY = 0EDCBA98765432H
                                        ;  overdracht = 0
                                        ;
```

```
                                        ; opgeslagen als
                                        ;
                                        ;   AY   = 032H
                                        ;   AY+1 = 054H
                                        ;   AY+2 = 076H
                                        ;   AY+3 = 098H
                                        ;   AY+4 = 0BAH
                                        ;   AY+5 = 0DCH
                                        ;   AY+6 = 00EH

                    jmp     SC0708


; Data
SZAY:               .EQU    7           ; Lengte van de operand
SHIFTS:             .BYTE   4           ; Aantal verschuivingen
AYADR:              .WORD   AY          ; Beginadres van de operand
AY:                 .BYTE 21H, 43H, 65H, 87H, 0A9H, 0CBH, 0EDH


                    .END
```

Meervoudige precisie logische rotatie rechts - 7I

Roteert een uit meerdere bytes bestaand getal een opgegeven aantal bit-posities naar rechts (alsof de hoogst- en laagstwaardige bits direct gekoppeld zijn). De lengte van het getal (in bytes) is 255 of minder. De overdrachtvlag wordt gezet naar de waarde van het laatste bit dat vanuit de meest rechtse bit-positie is geschoven. Het getal is met de laagstwaardige byte op het laagste adres opgeslagen.
Procedure: Het programma schuift bit 0 van de laagstwaardige byte van het getal naar de overdrachtvlag en roteert dan het hele getal een bit, beginnende bij de hoogstwaardige byte, naar rechts. Het herhaalt deze handeling gedurende het opgegeven aantal schuivingen.

Gebruikte registers: Alle

Looptijd: AANTAL SCHUIVINGEN * (21 + 18 * LENGTE VAN GETAL IN BYTES) + 85 cycli.
Als, bij voorbeeld, AANTAL SCHUIVINGEN = 6 en LENGTE VAN GETAL IN BYTES = 4 (d.w.z. een 6-bits schuiving van een 4-bytes getal), dan is de looptijd

6 * (21 + 18 * 4) + 85 = 6 * 93 + 85 = 643 cycli.

Programmalengte: 63 bytes

Benodigd datageheugen: Twee bytes ergens in RAM plus twee bytes op pagina 0. De twee bytes ergens in RAM dienen voor de tijdelijke opslag van het aantal schuivingen (een byte op adres NBITS) en de lengte van het getal in bytes (een byte op adres LENGTH). De twee bytes op pagina 0 bevatten een pointer naar het getal (beginnend op adres PTR, $00D0_{16}$ in de listing).

Speciale gevallen:

1. Als de lengte van het getal nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.
2. Als het aantal schuivingen nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Aantal schuivingen (bit-posities)
Lengte van het getal in bytes
Laagstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)
Hoogstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)

Eindcondities

Getal het opgegeven aantal bit-posities naar rechts geroteerd (de hoogstwaardige bit-posities zijn met de inhoud van de laagstwaardige bit-posities gevuld). De overdrachtvlag is gezet overeenkomstig het laatste bit dat vanuit de meest rechtse positie is geschoven (of gewist als of het aantal schuivingen of de lengte van het getal nul is).

Voorbeelden

1. Data: Lengte van het getal (in bytes) = 08
   Getal = $85A4C719FE06741E_{16}$
   Aantal schuivingen = 04

   Resultaat: Geschoven getal = $E85A4C719FE06741_{16}$
   Dit is het oorspronkelijke getal vier bits naar rechts geroteerd; de vier hoogstwaardige bits zijn gelijk aan de oorspronkelijke vier laagstwaardige bits.
   Overdrachtvlag = 1, aangezien het laatste bit datuit van de meest rechtse positie werd geschoven, 1 was.

2. Data: Lengte van getal (in bytes) = 04
   Getal = $3F6A42D3_{16}$
   Aantal schuivingen = 03

   Resultaat: Geschoven getal = $67ED485A_{16}$
   Dit is het oorspronkelijke getal drie bits naar rechts geroteerd; de drie hoogstwaardige bits (011) zijn gelijk aan de oorsponkelijke drie laagstwaardige bits.
   Overdrachtvlag = 0, aangezien het bit dat het laatst vanuit de meest rechtse positie werd geschoven, 0 was.

```
;
;
;
;     Titel      : Multi-precieze rotatie naar rechts
;     Naam       : MPRR
;
;
;     Doel       : Roteer een multi-byte operand N bits naar rechts
;
;
;     Begin      : Top van de stack
;                    Laagstwaardige byte van het terugkeeradres
;                    Hoogstwaardige byte van het terugkeeradres
;                    Aantal bits dat geshift moet worden
;                    Lengte van de operand in bytes
;                    Laagstwaardige byte van het beginadres van de operand
;                    Hoogstwaardige byte van het beginadres van de operand
;
;     Eind       : Operand naar rechts geroteerd
;                  Overdrachtvlag = Laatste bit dat verschoven werd vanaf de
;                  laagstwaardige positie
```

```
;
;       Registers : Alle
;
;       Tijd      : 85 basis cycli plus ((18 * lengte) + 21) cycli per
;                   verschuiving
;
;       Grootte   : programma 63  bytes
;                   data       2  bytes plus
;                              2  bytes op pagina 0

PTR:                .EQU    0D0H        ; Pointer naar operand op pagina 0

MPRR:
          ; Bewaar het terugkeeradres

                    pla
                    tay
                    pla
                    tax

          ; Haal het aantal bits van de stack

                    pla
                    sta     NBITS

          ; Haal de lengte van de operand van de stack

                    pla
                    sta     LENGTH

          ; Haal het beginadres van de operand van de stack

                    pla
                    sta     PTR
                    pla
                    sta     PTR+1

          ; Zet het terugkeeradres terug op de stack

                    txa
                    pha
                    tya
                    pha

          ; Initialiseer

                    clc                 ; Maak de overdracht schoon
                    lda     LENGTH
                    beq     EXIT        ; Keer terug als LENGTH = 0
                    lda     NBITS
                    beq     EXIT        ; Keer terug als NBITS = 0
                                        ;  overdracht = 0

          ; Verlaag de pointer zodat de lengte-byte zowel als teller
          ; als ook als index gebruikt kan worden

                    lda     PTR
                    bne     MPRR1
                    dec     PTR+1       ; Verlaag hoogstwaardige byte als
                                        ; er geleend moet worden
```

```
MPRR1:
                dec     PTR             ; Verlaag laagstwaardige byte
                                        ; altijd

        ; Doe dit zolang er nog verschoven moeten worden

RRLP:
                ldy     #1
                lda     (PTR),Y         ; Haal de laagstwaardige byte van
                                        ; de operand op
                lsr     A               ; Overdrachtvlag = BIT 0
                ldy     LENGTH          ; Y = index naar het hoogstwaardige
                                        ; byte en de teller

LOOP:
        ; Roteer een bit naar rechts

                lda     (PTR),Y         ; Haal de volgende byte op
                ror     A               ; Roteer  BIT 7 = overdracht
                                        ;         overdracht = BIT 0
                sta     (PTR),Y         ; Zet de nieuwe waarde terug
                dey                     ; Verlaag de teller
                bne     LOOP            ; Ga door zolang er nog bytes
                                        ; gedaan moeten worden

                dec     NBITS           ; Verlaag het aantal shifts
                bne     RRLP            ; Ga door zolang er nog shifts
                                        ; gedaan moeten worden

EXIT:
                rts


; Data
NBITS:          .BLOCK  1               ; Aantal bits
LENGTH:         .BLOCK  1               ; Lengte van de operand

;
;
; Voorbeeld uitvoering
;
;

SC0709:
        ; Zet het beginadres van de operand op de stack

                lda     AYADR+1
                pha
                lda     AYADR
                pha

        ; Zet de lengte van de operand op de stack

                lda     #SZAY
                pha

        ; Zet het aantal bits op de stack

                lda     SHIFTS
                pha
                jsr     MPRR            ; Roteer
```

```
                brk                     ; Resultaat van
                                        ;
                                        ;   AY = EDCBA987654321H   4 BITS
                                        ;
                                        ;         is
                                        ;
                                        ;   AY = 1EDCBA98765432H
                                        ;   Overdrachtvlag = 0
                                        ;
                                        ; opgeslagen als
                                        ;
                                        ;    AY   = 032H
                                        ;    AY+1 = 054H
                                        ;    AY+2 = 076H
                                        ;    AY+3 = 098H
                                        ;    AY+4 = 0BAH
                                        ;    AY+5 = 0DCH
                                        ;    AY+6 = 01EH
                jmp     SC0709


; Data
SZAY:           .EQU    7               ; Lengte van de operand
SHIFTS:         .EQU    4               ; Aantal shifts
AYADR:          .WORD   AY              ; Beginadres van de operand

AY:             .BYTE   21H,43H,65H,87H,0A9H,0CBH,0EDH


                .END
```

Meervoudige precisie rotatie links - 7J

Roteert een, uit meerdere bytes bestaand getal een opgegeven aantal bit-posities naar links (alsof het hoogstwaardige bit en het laagstwaardige bit direct aan elkaar vastzitten). De lengte van het getal (in bytes) is 255 of minder. De overdrachtvlag wordt gezet naar de waarde van het laatste bit dat vanuit de meest linkse bit-positie is geschoven. Het getal is met zijn laagstwaardige byte op het laagste adres opgeslagen.
Procedure: Het programma schuift bit 7 van de hoogstwaardige byte van het getal naar de overdrachtvlag en roteert dan het gehele getal een bit naar links. Het herhaalt deze handeling gedurende het opgegeven aantal schuivingen.

Gebruikte registers: Alle

Looptijd: AANTAL SCHUIVINGEN * (27 + 20 * LENGTE VAN GETAL IN BYTES) + 73 cycli.

Programmalengte: 60 bytes

Uitvoeringstijd: Als, bijvoorbeeld, AANTAL SCHUIVINGEN = 4 en LENGTE VAN GETAL IN BYTES = 8 (ofwel een 4-bit schuiving van een 8-byte getal), dan is de looptijd

4 * (27 + 20 * 8) + 73 = 4 * 187 + 73 = 821 cycli.

Benodigd datageheugen: Twee bytes ergens in RAM, plus twee bytes op pagina 0. De twee bytes ergens in RAM dienen voor de tijdelijke opslag van het aantal schuivingen (een byte op adres NBITS) en de lengte van het getal in bytes (een byte op adres LENGTH). De twee bytes op pagina 0 bevatten een pointer naar het getal (beginnend op adres PTR, $00D0_{16}$ in de listing).

Speciale gevallen:

1. Als de lengte van het getal nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.
2. Als het aantal schuivingen nul is, stopt het programma onmiddellijk, verandert het getal niet en wist de overdrachtvlag.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Aantal schuivingen (bit-posities)
Lengte van het getal in bytes
Laagstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)
Hoogstwaardige byte van het beginadres van het getal (adres van de laagstwaardige byte)

Eindcondities

Getal het opgegeven aantal bit-posities naar links geroteerd (de laagstwaardige bit-posities zijn gevuld met de inhoud van de hoogstwaardige bit-posities). De overdrachtvlag is gezet overeenkomstig het laatste bit dat vanuit de meest linkse positie is geschoven (of gewist als of het aantal schuivingen of de lengte van het getal nul is).

Voorbeelden

1. Data: Lengte van het getal (in bytes) = 8
   Getal = $85A4C719FE06741E_{16}$
   Aantal schuivingen = 04

   Resultaat: Geschoven getal = $5A4C719FE06741E8_{16}$
   Dit is het oorspronkelijke getal vier bits naar links geroteerd; de vier laagstwaardige bits zijn gelijk aan de vier oorspronkelijke hoogstwaardige bits.
   Overdrachtvlag= 0, aangezien het laatste bit dat van de meest linkse positie werd geschoven, 0 was.

2. Data: Lengte van getal (in bytes) = 04
   Getal = $3F6A42D3_{16}$
   Aantal schuivingen = 03

   Resultaat: Geschoven getal = $FB521699_{16}$
   Dit is het oorspronkelijke getal drie bits naar links geroteerd; de drie laagstwaardige bits (001_2) zijn gelijk aan de drie oorspronkelijke hoogstwaardige bits.
   Overdrachtvlag= 1, aangezien het laatste bit dat vanuit de meest linkse positie werd geschoven, 1 was.

```
;
;
;
;       Titel     : Multi-precieze rotatie naar links
;       Naam      : MPRL
;
;
;       Doel      : Roteer een multi-byte operand N bits naar links
;
;
;       Begin     : Top van de stack
;                     Laagstwaardige byte van het terugkeeradres
;                     Hoogstwaardige byte van het terugkeeradres
;                     Aantal bits dat geshift moet worden
;                     Lengte van de operand in bytes
;                     Laagstwaardige byte van het beginadres van de operand
;                     Hoogstwaardige byte van het beginadres van de operand
;
;       Eind      : Operand naar links geroteerd
;                   Overdrachtvlag = Laatste bit dat verschoven werd vanaf de
;                   hoogstwaardige positie
;
;       Registers : Alle
```

```
;
;       Tijd        : 73 basis cycli plus ((20 * lengte) + 27) cycli per
;                     verschuiving
;
;       Grootte     : programma 60 bytes
;                     data        2 bytes plus
;                                 2 bytes op pagina 0

PTR:                  .EQU    0D0H          ; Pointer naar operand op pagina 0

MPRL:
            ; Bewaar het terugkeeradres

                      pla
                      tay
                      pla
                      tax

            ; Haal het aantal bits van de stack

                      pla
                      sta     NBITS

            ; Haal de lengte van de operand van de stack

                      pla
                      sta     LENGTH

            ; Haal het beginadres van de operand van de stack

                      pla
                      sta     PTR
                      pla
                      sta     PTR+1

            ; Zet het terugkeeradres terug op de stack

                      txa
                      pha
                      tya
                      pha

            ; Initialiseer

                      clc                   ; Maak de overdracht schoon
                      lda     LENGTH
                      beq     EXIT          ; Keer terug als LENGTH = 0
                      lda     NBITS
                      beq     EXIT          ; Keer terug als NBITS = 0

            ; Doe dit net zolang tot alle shift uitgevoerd zijn

RLLP:
                      ldy     LENGTH
                      dey
                      lda     (PTR),Y       ; Haal de hoogstwaardige byte van
                                            ; de operand op
                      asl     A             ; Overdrachtvlag = bit 7 van deze
                                            ; byte
                      ldy     #0            ; Y = index naar de laagstwaardige
                                            ; byte
                      ldx     LENGTH        ; X = Aantal bytes
```

```
        ; Roteer een bit naar links

LOOP:
                lda     (PTR),Y         ; Haal de volgende byte op
                rol     A               ; Roteer  BIT 7 = overdracht
                                        ;         overdracht = BIT 0
                sta     (PTR),Y         ; Zet de nieuwe waarde terug
                iny                     ; Verhoog naar de volgende byte
                dex                     ; Verlaag de teller
                bne     LOOP            ; Ga door tot alle bytes verschoven
                                        ; zijn

                dec     NBITS           ; Verlaag de schuif-teller
                bne     RLLP            ; Ga door zolang er nog verschoven
                                        ; moet worden

EXIT:
                rts


; Data
NBITS:          .BLOCK  1               ; Aantal bits dat verschoven moet
                                        ; worden
LENGTH:         .BLOCK  1               ; Lengte van de operand

;
;
; Voorbeeld uitvoering
;
;

SC0710:
        ; Zet het beginadres van de operand op de stack

                lda     AYADR+1
                pha
                lda     AYADR
                pha

        ; Zet de lengte van de operand op de stack

                lda     #SZAY
                pha

        ; Zet het aantal shifts op de stack

                lda     SHIFTS
                pha
                jsr     MPRL            ; Roteer
                brk                     ; Resultaat van
                                        ;
                                        ;  AY = EDCBA987654321H
                                        ;  4 BITS
                                        ;
                                        ;      is
                                        ;
                                        ;  AY = DCBA987654321EH
                                        ;  Overdrachtvlag = 0
                                        ;
                                        ; opgeslagen als
                                        ;
```

```
                                        ;   AY   = 01EH
                                        ;   AY+1 = 032H
                                        ;   AY+2 = 054H
                                        ;   AY+3 = 076H
                                        ;   AY+4 = 098H
                                        ;   AY+5 = 0BAH
                                        ;   AY+6 = 0DCH


                jmp     SC0710


; Data
SZAY:           .EQU    7               ; Lengte van de operand
SHIFTS:         .EQU    4               ; Aantal verschuivingen
AYADR:          .WORD   AY              ; Beginadres van de operand

AY:             .BYTE   21H,43H,65H,87H,0A9H,0CBH,0EDH


                .END
```

String-vergelijking - 8A

Vergelijkt twee strings en zet de overdracht- en de nulvlag overeenkomstig. De nulvlag wordt op 1 gezet als de strings identiek zijn en op 0 als dit niet het geval is. De overdrachtvlag wordt op 0 gezet als de string met het adres hoger in de stack (string 2) groter is dan de andere string (string 1); de overdrachtvlag wordt anders op 1 gezet. De strings zijn maximaal 255 bytes lang en de karakters van de string worden voorafgegaan door een byte die de lengte van de string aangeeft. Als de twee strings tot de lengte van de kortste gelijk zijn, dan wordt de langste geacht groter te zijn.
Procedure: Het programma stelt eerst, aan de hand van de opgegeven lengten, vast welke string de kortste is. Dan vergelijkt het de strings byte-voor-byte gedurende de lengte van de kortste. Als het programma gedurende de lengte van de kortste string twee corresponderende bytes aantreft die niet gelijk zijn, zet het programma de vlaggen door de lengten te vergelijken.

Gebruikte registers: Alle

Looptijd:

1. Als de strings gedurende de lengte van de kortste niet gelijk zijn, is de looptijd ongeveer

   81 + 19 * AANTAL VERGELEKEN KARAKTERS.

   Als de routine, bij voorbeeld, vijf karakters vergelijkt voordat er een verschil wordt aangetroffen, is de looptijd

   81 + 19 * 5 = 81 + 95 = 176 cycli.

2. Als de strings gedurende de lengte van de kortste gelijk zijn, is de looptijd ongeveer

   93 + 19 = lengte van de kortste string

   Als de kortste string bij voorbeeld acht bytes lang is, is de looptijd

   93 + 19 * 8 = 93 + 152 = 245 cycli.

Programmalengte: 52 bytes

Benodigd datageheugen: Vier bytes op pagina 0, twee bytes vanaf adres S1ADR ($00D0_{16}$ in de listing) voor een pointer naar string 1 en twee bytes vanaf adres S2ADR ($00D2_{16}$ in de listing) voor een pointer naar string 2.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van het beginadres van string 2
Hoogstwaardige byte van het beginadres van string 2

Laagstwaardige byte van het beginadres van string 1
Hoogstwaardige byte van het beginadres van string 1

Eindcondities

Vlaggen gezet alsof string 2 van string 1 is afgetrokken, of, als de strings gedurende de lengte van de kortste string gelijk zijn, net alsof de lengte van string 2 van de lengte van string 1 afgetrokken is.

Nulvlag = 1 als de strings gelijk zijn, 0 als ze niet gelijk zijn.

Overdrachtvlag = 0 als string 2 groter is dan string 1, 1 als zij gelijk zijn of string 1 groter is. Als de strings gedurende de lengte van de kortste string gelijk zijn, wordt de langste geacht de grootste te zijn.

Voorbeelden

1. Data: String 1 = 05'PRINT' (05 is de lengte van de string)
String 2 = 03'END' (03 is de lengte van de string)

Resultaat: Nulvlag = 0 (strings zijn niet gelijk)
Overdrachtvlag = 1 (string 2 is niet groter dan string 1)

2. Data: String 1 = 05'PRINT' (05 is de lengte van de string)
String 2 = 02'PR' (02 is de lengte van de string)

Resultaat: Nulvlag = 0 (strings zijn niet gelijk)
Overdrachtvlag = 1 (string 2 is niet groter dan string 1)

De langste string (string 1) wordt geacht groter te zijn. Als bekend moet zijn of string 2 een afkorting van string 1 is, kan subroutine 8C (Zoek de positie van een substring) gebruikt worden om te bepalen of string 2 een deel van string 1 was en bij het eerste karakter begon.

3. Data: String 1 = 05'PRINT' (05 is de lengte van de string)
String 2 = 07'SYSTEEM' (07 is de lengte van de string)

Resultaat; Nulvlag = 0 (strings zijn niet gelijk)
Overdrachtvlag = 0 (string 2 is groter dan string 1)

We gaan er hier vanuit dat de strings uit ASCII-karakters bestaan. Let op dat de byte die aan de string voorafgaat een hexadecimaal getal bevat (de lengte van de string), geen karakter. We hebben deze byte weergegeven als twee hexadecimale digits voor de string; de string zelf is afgebakend met enkele aanhalingstekens.
Let op dat deze routine spaties als elk ander karakter behandelt. Als, bij voorbeeld, de strings uit ASCII-karakters bestaan, zal de routine constateren dat VERREKIJKER groter is dan VERRE KIJKER, aangezien een ASCII K ($4B_{16}$) groter is dan een ASCII spatie (20_{16}).

```
;
;
;       Titel       : Strings vergelijken
;       Naam        : STRCMP
;
;
;       Doel        : Vergelijk twee strings en retourneer
;                     met de C- en de Z-vlag schoon of gezet
;
;
;       Begin       : Top van de stack
;                       Laagstwaardige byte van het terugkeeradres
;                       Hoogstwaardige byte van het terugkeeradres
;                       Laagstwaardige byte van het adres van string 2
;                       Hoogstwaardige byte van het adres van string 2
;                       Laagstwaardige byte van het adres van string 1
;                       Hoogstwaardige byte van het adres van string 1
;
;                       Een string is maximaal 255 bytes lang plus een
;                       lengte-byte die aan de string voorafgaat
;
;       Eind        : Als string 1 = string 2 dan
;                         Z=1, C=1
;                     Als string 1 > string 2 dan
;                         Z=0, C=1
;                     Als string 1 < string 2 dan
;                         Z=0, C=0
;
;       Registers : Alle
;
;       Tijd        : In het ergste geval zijn de strings gelijk.
;                       93 cycli maximale basistijd plus (19 * lengte)
;
;       Grootte     : programma 52  bytes
;                     data       4  bytes op pagina 0


S1ADR:          .EQU    0D0H            ; Pointer naar string 1 op
                                        ; pagina 0
S2ADR:          .EQU    0D2H            ; Pointer naar string 2 op
                                        ; pagina 0


STRCMP:
        ; Haal het terugkeeradres van de stack
                pla
                tay
                pla
                tax

        ; Haal het beginadres van string 2 van de stack
                pla
                sta     S2ADR
                pla
                sta     S2ADR+1

        ; Haal het beginadres van string 1 van de stack
                pla
                sta     S1ADR
                pla
                sta     S1ADR+1
```

```
            ; Zet het terugkeeradres terug op de stack
                  txa
                  pha
                  tya
                  pha

            ; Bepaal welke string het kortste is
                  ldy     #0
                  lda     (S1ADR),Y       ; Haal de lengte van string 1 op
                  cmp     (A2ADR),Y
                  bcc     BEGCMP          ; Als string 2 korter is,
                  lda     (S2ADR),Y       ; gebruik dan zijn lengte

            ;
            ; Vergelijk de twee strings met elkaar tot en met het
            ; laatste karakter van de kortste string
BEGCMP:
                  tax                     ; X = lengte van de kortste
                                          ; string
                  beq     TSTLEN          ; Spring als de lengte 0 is

                  ldy     #1              ; Wijs naar het eerste karakter

CMPLP:
                  lda     (S1ADR),Y
                  cmp     (S2ADR),Y
                  bne     EXIT            ; Spring als de karakters
                                          ; niet gelijk zijn
                                          ;   Z en C worden goed gezet,
                                          ; anders
                  iny                     ;   Ga wijzen naar het
                                          ;   volgende karakter
                  dex                     ;   Verlaag de teller
                  bne     CMPLP           ;   Ga door tot alle
                                          ;   karakters vergeleken zijn

            ; De twee strings zijn gelijk, tot op het laatste karakter
            ; van de kortste string. Gebruik nu de lengte om de vlaggen
            ; goed te zetten
TSTLEN:
                  ldy     #0              ; Vergelijk de lengten
                  lda     (S1ADR),Y
                  cmp     (S2ADR),Y       ; Zet de vlaggen goed

            ; Keer terug
EXIT:
                  rts


;
;
; Voorbeeld uitvoering
;
;

SC0801:
            ; Zet het beginadres van string 1 op de stack
                  lda     SADR1+1
                  pha
                  lda     SADR1
                  pha

            ; Zet het beginadres van string 2 op de stack
```

```
                lda     SADR2+1
                pha
                lda     SADR2
                pha

                jsr     STRCMP          ; Vergelijk
                brk                     ; Resultaat van de
                                        ; vergelijking van
                                        ;   string 1 met string 2
                                        ; is dat string 1 kleiner dan
                                        ; string 2 is, dus
                                        ;    Z = 0, C = 0
                jmp     SC0801

; Test-data, verander deze zelf om andere strings te testen
SADR1:          .WORD   S1
SADR2:          .WORD   S2
S1:             .BYTE   20H,"STRING1                    "
S2:             .BYTE   20H,"STRING2                    "

                .END
```

String-rijging - 8B

Combineert (rijgt) twee strings door de tweede onmiddellijk achter de eerste in het geheugen te plaatsen. Als de rijging een string zou opleveren die groter is dan een aangegeven maximum, rijgt het programma alleen zoveel van string 2 dat de gecombineerde string de maximum lengte krijgt. De overdrachtvlag wordt gewist als alles van string 2 geregen kan worden en op 1 gezet als een deel van string 2 moet afvallen. Beide strings zijn maximaal 255 karakters groot en worden voorafgegaan door een byte die de lengte aangeeft.
Procedure: Het programma gebruikt de lengte van string 1 om te bepalen vanaf waar de karakters bijgevoegd moeten worden. De lengte van string 2 wordt gebruikt om te bepalen hoeveel karakters bijgevoegd moeten worden. Als de som van de lengten het maximum overschrijdt, geeft het programma een overschot aan en verkleint het aantal te rijgen karakters (dit aantal is de maximum lengte minus de lengte van string 1). Vervolgens verplaatst het programma het juiste aantal karakters van string 2 naar het einde van string 1, en zet de overdrachtvlag om aan te geven of er karakters buiten de boot zijn gevallen.

Gebruikte registers: Alle

Looptijd: Ongeveer 40 * AANTAL SAMENGEVOEGDE KARAKTERS plus 164 cycli vast. Het AANTAL SAMENGEVOEGDE KARAKTERS is in het algemeen de lengte van string 2, maar wordt de maximum lengte van string 1 minus de huidige lengte als de samengevoegde string langer zou zijn dan het maximum. Als, bij voorbeeld, AANTAL SAMENGEVOEGDE KARAKTERS is 14_{16} (20_{10}), is de looptijd

40 * 20 + 161 = 800 + 164 = 964 cycli.

Programmalengte: 141 bytes

Benodigd datageheugen: Zeven bytes ergens in RAM, plus vier bytes op pagina 0. De zeven bytes ergens in RAM dienen als tijdelijke opslag voor de maximum lengte van string 1 (1 byte op adres MAXLEN), de lengte van string 1 (1 byte op adres S1LEN), de lengte van string 2 (1 byte op adres S2LEN), een index voor string 1 (1 byte op adres S1IDX), een index voor string 2 (1 byte op adres S2IDX), een rijgteller (1 byte op adres COUNT) en een vlag die aangeeft of de gecombineerde strings een overschot veroorzaakten (1 byte op adres STRGOV). De vier bytes op pagina 0 bevatten pointers naar string 1 (twee bytes vanaf adres S1ADR, adres $00D0_{16}$ in de listing) en naar string 2 (twee bytes vanaf adres S2ADR, adres 00D216 in de listing).

Speciale gevallen:

1. Als de samenvoeging een string zou opleveren die groter is dan de aangegeven maximum lengte, rijgt het programma alleen dat deel van string 2 aan dat de gezamenlijke string de maximale lengte geeft. Als een deel van string 2 moet afvallen, wordt de overdrachtvlag op 1 gezet.
2. Als string 2 een lengte van nul heeft, stopt het programma met de overdrachtvlag gewist (geen fouten) en string 1 ongewijzigd. Dit houdt in dat een lengte van 0 voor elke string ook als nul en niet als 256 wordt geinterpreteerd.

3. Als de oorspronkelijke lengte van string 1 de aangegeven maximum lengte overschrijdt, stopt het programma met de overdrachtvlag op 1 gezet (een fout opgetreden) en string 1 ongewijzigd.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Maximale lengte van string 1
Laagstwaardige byte van het beginadres van string 2
Hoogstwaardige byte van het beginadres van string 2
Laagstwaardige byte van het beginadres van string 1
Hoogstwaardige byte van het beginadres van string 1

Eindcondities

String 2 aan het einde van string 1 geregen en de lengte van string 1 aangepast. Als de nieuwe string de maximum lengte zou overschrijden, zou slechts dat deel van string 2 worden toegevoegd dat de maximale lengte geeft. Als er een deel van string 2 moet afvallen, wordt de overdrachtvlag op 1 gezet. Anders wordt de overdrachtvlag gewist.

Voorbeelden

1. Data: Maximum lengte van string 1 = $0E_{16} = 14_{10}$
String 1 = 07'JOHNSON' (07 is de lengte van de string)
String 2 = 05', DON' (05 is de lengte van de string)

Resultaat: String 1 = 0C'JOHNSON, DON' ($0C_{16} = 12_{10}$ is de lengte van de gecombineerde string, waarbij string 2 achter string 1 is geplakt)
Overdrachtvlag = 0, aangezien de samenvoeging geen string gaf die de maximum lengte overschreed.

2. Data: Maximum lengte van string 1 = $0E_{16} = 14_{10}$
String 1 = 07'JOHNSON' (07 is de lengte van de string)
String 2 = 09', RICHARD' (09 is de lengte van de string)

Resultaat: String 1 = 0E'JOHNSON, RICHA' ($0E_{16} = 14_{10}$ is de maximum lengte, dus zijn de laatste twee karakters van string 2 afgevallen)
Overdrachtvlag = 1, aangezien de samenvoeging een string opleverde die langer was dan de maximum lengte.

Let op dat we de, aan de string voorafgaande byte (die de lengte van de string bevat), in beide voorbeelden weergeven als twee hexadecimale digits.

```
;
;
;
;       Titel      : String-rijger
;       Naam       : CONCAT
;
;
;       Doel       : Maak van twee strings een
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Maximale lengte van string 1
;                      Laagstwaardige byte van het beginadres van string 2
;                      Hoogstwaardige byte van het beginadres van string 2
;                      Laagstwaardige byte van het beginadres van string 1
;                      Hoogstwaardige byte van het beginadres van string 1
;
;                      Een string is maximaal 255 bytes lang plus een
;                      lengte-byte die aan de string voorafgaat
;
;       Eind       : String 1 := String 1 + String 2
;                    Als er geen fouten werden gemaakt dan is de
;                    overdrachtvlag schoon, anders is de overdrachtvlag 1.
;                    Als de nieuwe string groter dan string 1 is dan wordt
;                    string 1 tot het maximum gevuld met de nieuwe string.
;                    Als de lengte van string 1 groter dan de maximum lengte
;                    is dan wordt string 1 niet veranderd
;
;       Registers  : Alle
;
;       Tijd       : Ongeveer 40 * (lengte van string2) plus
;                    161 basis cycli
;
;       Grootte    : programma 141  bytes
;                    data        7  bytes plus
;                                4  bytes op pagina 0
;


S1ADR:              .EQU    0D0H            ; Pointer naar string 1 op
                                            ; pagina 0
S2ADR:              .EQU    0D2H            ; Pointer naar string 2 op
                                            ; pagina 0

CONCAT:
            ; Haal het terugkeeradres van de stack
                    pla
                    tay
                    pla
                    tax

            ; Haal de maximale lengte van string 2 van de stack
                    pla
                    sta     MAXLEN

            ; Haal het beginadres van string 2 van de stack
                    pla
                    sta     S2ADR
                    pla
                    sta     S2ADR+1
```

```
; Haal het beginadres van string 1 van de stack
        pla
        sta     S1ADR
        pla
        sta     S1ADR+1

; Zet het terugkeeradres terug op de stack
        txa
        pha
        tya
        pha

; Bepaal de plaats waarvanaf begonnen kan worden met rijgen
        ldy     #0
        lda     (S1ADR),Y       ; De huidige lengte van string 1
        sta     S1LEN
        sta     S1IDX
        inc     S1IDX           ; Begin aan het eind van string 1
        lda     (S2ADR),Y       ; Haal de lengte van string 2 op
        sta     S2LEN
        lda     #1
        sta     S2IDX           ; Begin met het eerste
                                ; karakter van string 2

; Bepaal hoeveel karakters van string 2 achter string 1
; geplakt moeten worden
        lda     S2LEN           ; De lengte van string 2
        clc
        adc     S1LEN           ; Tel deze op bij de huidige
                                ; lengte van string 1
        bcs     TOOLNG          ; Spring als resultaat > 255
        cmp     MAXLEN          ; Vergelijk met de maximale
                                ; lengte
        beq     LENOK           ; Spring als de lengte
        bcc     LENOK           ; kleiner dan de maximale
                                ; lengte is

; De resulterende string zou te lang worden dus
;   Geef een indicatie dat de grens werd overschreden door
;    STRGOV gelijk te maken aan 0FFH
;   Maak het aantal karakters die aan elkaar geregen moeten
;    worden gelijk aan MAXLEN - S1LEN
;   Maak de lengte van string 1 gelijk aan MAXLEN
TOOLNG:
        lda     #0FFH
        sta     STRGOV          ; Zet op 0FFH
        lda     MAXLEN
        sec
        sbc     S1LEN
        bcc     EXIT            ; Keer terug als de originele
                                ; string 1 groter was dan MAXLEN
        sta     COUNT           ; Zet het aantal karakters goed
        lda     MAXLEN
        sta     S1LEN           ; Zet de lengte van string 1 goed
        jmp     DOCAT           ; Begin te rijgen

; De resulterende lengte is <= MAXLEN dus
;   Maak de lengte van string 1 gelijk aan S1LEN + S2LEN
;   Zet de grensvlag op 0
;   Maak het aantal te rijgen karakters gelijk aan S2LEN
```

```
LENOK:
                sta     S1LEN           ; Bewaar de som van de lengten
                lda     #0
                sta     STRGOV          ; Geen grensoverschrijding
                lda     S2LEN
                sta     COUNT           ; COUNT := S2LEN

            ; Rijg de strings aan elkaar
DOCAT:
                lda     COUNT
                beq     EXIT            ; Spring als we klaar zijn

CATLP:
                ldy     S2IDX
                lda     (S2ADR),Y       ; Het volgende karakter van
                                        ; string 2
                ldy     S1IDX
                sta     (S1ADR),Y       ; Voeg het achter het laatste
                                        ; karakter van string 1
                inc     S1IDX           ; Verhoog de index in string 1
                inc     S2IDX           ; Verhoog de index in string 2
                dec     COUNT           ; Verlaag de teller
                bne     CATLP           ; Ga door tot COUNT = 0

EXIT:
                lda     S1LEN           ; Maak het lengteveld
                ldy     #0              ; van string 1 gelijk aan de
                sta     (S1ADR),Y       ; nieuwe lengte
                lda     STRGOV          ; Haal de grensvlag op
                ror     A               ; Als C-vlag = 1 dan is de
                                        ; grens overschreden
                rts


; Data
MAXLEN:         .BLOCK  1               ; Max. lengte van string 1
S1LEN:          .BLOCK  1               ; Lengte van string 1
S2LEN:          .BLOCK  1               ; Lengte van string 2
S1IDX:          .BLOCK  1               ; Index in string 1
S2IDX:          .BLOCK  1               ; Index in string 2
COUNT:          .BLOCK  1               ; Teller voor het aantal
                                        ; te rijgen karakters
STRGOV:         .BLOCK  1               ; Grensvlag

;
;
; Voorbeeld uitvoering
;
;

SC0802:
            ; Zet het beginadres van string 1 op de stack
                lda     SADR1+1
                pha
                lda     SADR1
                pha

            ; Zet het beginadres van string 2 op de stack
                lda     SADR2+1
                pha
                lda     SADR2
                pha
```

```
        ; Zet de maximale lengte van string 1 op de stack
                lda     #20H
                pha
                jsr     CONCAT          ; Rijg de strings aan elkaar
                brk                     ; Resultaat van
                                        ;    "LASTNAME" en ", FIRSTNAME"
                                        ;
                                        ;        is
                                        ;
                                        ;  S1 = 13H,"LASTNAME, FIRSTNAME"

                jmp     SC0802


; Test-data, verander deze zelf om andere string te proberen
SADR1:          .WORD   S1              ; Beginadres string 1
SADR2:          .WORD   S2              ; Beginadres string 2
S1:             .BYTE   8H              ; Lengte van string 2
                .BYTE   "LASTNAME                          "
                                        ; max 32 bytes
S2:             .BYTE   0BH             ; Lengte van string 2
                .BYTE   ", FIRSTNAME                       "
                                        ; max 32 bytes

                .END
```

Zoekt de positie van een substring - 8C

Zoekt naar het eerste voorkomen van een substring binnen een string. Geeft de index waarop de substring begint terugals deze gevonden is en geeft 0 terug als deze niet gevonden is. Zowel de string als de substring zijn maximaal 255 bytes lang en de karakters van de string worden voorafgegaan door een byte met de lengte van de string. Zodoende kan de begin-index van een substring, als deze is gevonden, nooit kleiner dan 1 of groter dan 255 zijn.
Procedure: Het programma gaat door de string op zoek naar de substring tot er een overeenstemmimg is gevonden, of tot het resterende deel van de string kleiner is dan de substring en de substring er daarom nooit meer in kan passen. Als de substring niet wordt gevonden, wist het programma de accumulator; in het andere geval wordt de begin-index van de substring in de accumulator gezet.

Gebruikte registers: Alle

Looptijd: Afhankelijk van de data, maar in ieder geval 135 cycli, elke overeenstemming van een karakter duurt 47 cycli, iedere niet-overeenstemming van een karakter duurt 50 cycli. De meeste tijd wordt in beslag genomen als de string en de substring steeds gelijk zijn, op het laatste karakter in de substring na. Bij voorbeeld:

String = 'AAAAAAAAB'
Substring = 'AAB'

In dit geval is de looptijd

(STRING-LENGTE - SUBSTRING-LENGTE + 1) * (47 * (SUBSTRING-LENGTE - 1) + 50) + 135

Als, bijvoorbeeld STRING-LENGTE = 9 en SUBSTRING-LENGTE = 3, dan is de looptijd

(9 - 3 + 1) * (47 * (3 - 1) + 50) + 135 = 7 * 144 + 135 = 1008 + 135 = 1143 cycli.

Programmalengte: 124 bytes

Benodigd datageheugen: Zes bytes ergens in RAM plus vier bytes op pagina 0. De zes bytes ergens in RAM dienen voor tijdelijke opslag voor de lengte van de string (een byte op adres SLEN), de lengte van de substring (een byte op adres SUBLEN), een lopende index binnen de string (een byte op adres SIDX), een lopende index binnen de substring (een byte op adres SUBIDX), een zoekteller (een byte op adres COUNT) en een index binnen de string (een byte op adres INDEX). De vier bytes op pagina 0 bevatten pointers naar de substring (twee bytes vanaf adres SUBSTG, $00D0_{16}$ in de listing) en naar de string (twee bytes vanaf adres STRING, $00D2_{16}$ in de listing).

Speciale gevallen:

1. Als of de string of de substring een lengte van 0 heeft, stopt het programma. Hierbij wordt de accumulator gewist, aangevende dat de substring niet gevonden is.
2. Als de substring langer is dan de string, stopt het programma. Hierbij wordt de accumulator op nul gezet, aangevende dat de substring niet gevonden is.
3. Als het programma een index van 1 teruggeeft, kan de substring worden beschouwd als een afkorting van de string, beginnende vanaf het eerste karakter. Een voorbeeld hiervan is string PRINT en substring PR.
4. Als de substring meer dan eens in de string voorkomt, zal het programma alleen de index van de eerste overeenkomst teruggeven (de overeenkomst met de laagste begin-index).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van het beginadres van de substring
Hoogstwaardige byte van het beginadres van de substring
Laagstwaardige byte van het beginadres van de string
Hoogstwaardige byte van het beginadres van de string

Eindcondities

Accumulator bevat index naar de eerste overeenkomst van de substring, als deze is gevonden. Accumulator bevat nul als de substring niet is gevonden.

Voorbeelden

1. Data: String = 23'GEEF SNELHEID IN KILOMETERS PER UUR' (23_{16} = 35_{10} is de lengte van de string)
Substring = 0A'KILOMETERS' ($0A_{16}$ = 10_{10} is de lengte van de substring)

   Resultaat: Accumulator bevat 12_{16} (18_{10}), de index waarop de substring 'KILOMETERS' begint.

2. Data: String = 1D'VERKOOPCIJFERS VOOR JUNI 1985' ($1D_{16}$ = 29_{10} is de lengte van de string)
Substring = 04'JUNI' (04 is de lengte van de substring)

   Resultaat: Accumulator bevat 15_{16} (21_{10}), de index waarop de substring 'JUNI' begint.

3. Data: String = 10'LET Y1 = X1 + R7' (10_{16} = 16_{10} is de lengte van de string)
Substring = 02'R4' (02 is de lengte van de substring).

Resultaat: Accumulator bevat 00, aangezien de substring 'R4' niet voorkomt in de string 'LET Y1 = X1 + R7'.

4. Data: String = 07'RESTORE' (07 is de lengte van de string)
Substring = 03'RES' (03 is de lengte van de substring)

Resultaat: Accumulator bevat 01, de index waarop de substring 'RES' begint. Een index van 01 geeft aan dat de substring een afkorting zou kunnen zijn van de string; zulke afkortingen worden, bij voorbeeld, vaak gebruikt in interactieve programma's (als BASIC interpreters) om opslagruimte en typewerk te besparen.

```
;       Titel      : Vind de positie van een substring in een string
;       Naam       : POS
;
;
;       Doel       : Het vinden van de eerste positie van een substring in een
;                    string. Als de substring geen deel van de string is wordt
;                    0 geretourneerd
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Laagstwaardige byte van het beginadres van de substring
;                      Hoogstwaardige byte van het beginadres van de substring
;                      Laagstwaardige byte van het beginadres van de string
;                      Hoogstwaardige byte van het beginadres van de string
;
;       Eind       : Als de substring in de string zat wordt de eerste positie
;                    van de substring in de string teruggegeven, anders 0
;
;       Registers  : Alle
;
;       Tijd       : Omdat het algoritme zo van de strings afhangt, is het
;                    vrijwel niet mogelijk om een simpele tijdformule op de
;                    schrijven, maar de volgende beweringen gelden
;
;                    135 basiscycli
;                    Elk goed karakter kost 47 cycli
;                    Een fout kost 50 cycli
;
;                    In het ergste geval is de string tot op het laatste
;                    karakter van de substring gelijk, bij voorbeeld
;                            string = "AAAAAAAAAB"
;                         substring = "AAB"
;                    Dan is de tijdformule weer te geven door
;
;
;                    135 basiscycli plus
;                    (lengte(string) - lengte(substring) + 1) *
;                        (((lengte(substring) - 1) * 47) + 50)
;
```

```
;       Grootte    : programma 124  bytes
;                    data        6  bytes plus
;                                4  bytes op pagina 0

SUBSTG:             .EQU    0D0H            ; Pointer naar substring op
                                            ; pagina 0
STRING:             .EQU    0D2H            ; Pointer naar string op
                                            ; pagina 0

POS:
            ; Haal het terugkeeradres van de stack
                    pla
                    tay
                    pla
                    tax

            ; Haal het beginadres van de substring van de stack
                    pla
                    sta     SUBSTG
                    pla
                    sta     SUBSTG+1

            ; Haal het beginadres van de string van de stack
                    pla
                    sta     STRING
                    pla
                    sta     STRING+1

            ; Zet het terugkeeradres terug op de stack
                    txa
                    pha
                    tya
                    pha

            ; Initieer de lengte- en index-variabelen
                    ldy     #0
                    lda     (STRING),Y      ; Lengte van de string
                    beq     NOTFND          ; Keer terug als lengte = 0
                    sta     SLEN
                    lda     (SUBSTG),Y      ; Lengte van de substring
                    beq     NOTFND          ; Keer terug als lengte = 0
                    sta     SUBLEN

            ; Als de substring langer dan de string is, zeg dan dat de
            ; substring niet in de string zit
                    lda     SUBLEN
                    cmp     SLEN
                    beq     LENOK
                    bcs     NOTFND          ; De substring is te lang

            ; Begin te zoeken, ga door tot het overgebleven deel
            ; van de string kleiner is dan de lengte van de substring
LENOK:
                    lda     #1
                    sta     INDEX           ; Begin met het eerste
                                            ; karakter van de string
                    lda     SLEN            ; Ga door tot de lengte van
                                            ; de overgebleven string
                                            ; kleiner is dan de substring
                    sec                     ; COUNT = lengte (string) -
                                            ;       lengte (substring) + 1
                    sbc     SUBLEN
```

```
                sta     COUNT
                inc     COUNT

        ; Zoek naar de substring in de string
SLP1:
                lda     INDEX
                sta     SIDX            ; Begin met als string-index,
                                        ; INDEX
                lda     #1
                sta     SUBIDX          ; Begin met als substring-
                                        ; index, 1

        ; Begin bij INDEX uit te kijken naar de substring
CMPLP:
                ldy     SIDX
                lda     (STRING),Y      ; Volgende karakter van de string
                ldy     SUBIDX

                cmp     (SUBSTG),Y      ; Vergelijk met volgende
                                        ; karakter van de substring
                bne     SLP2            ; Spring als de substring
                                        ; hier niet is
                ldy     SUBIDX
                cpy     SUBLEN          ; Test of we klaar zijn
                beq     FOUND           ; Gevonden!!!
                iny                     ; Ga naar het volgende
                                        ; karakter kijken
                sty     SUBIDX
                inc     SIDX            ; Verhoog de index in de string
                jmp     CMPLP           ; Ga door

        ; Als we hier komen is de substring nog niet gevonden
SLP2:
                inc     INDEX           ; Verhoog de index
                dec     COUNT           ; Verlaag de teller
                bne     SLP1            ; Spring als we nog niet
                                        ; klaar zijn
                beq     NOTFND          ; Anders zit de substring
                                        ; niet in de string
FOUND:
                lda     INDEX           ; Gevonden. A = index in de
                                        ; string
                jmp     EXIT


NOTFND:
                lda     #0              ; Niet gevonden. A = 0
EXIT:
                rts


; Data
SLEN:           .BLOCK  1               ; Lengte van de string
SUBLEN:         .BLOCK  1               ; Lengte van de substring
SIDX:           .BLOCK  1               ; Index in string
SUBIDX:         .BLOCK  1               ; Index in substring
COUNT:          .BLOCK  1               ; Teller
INDEX:          .BLOCK  1               ; Huidige index in de string

;
;
; Voorbeeld uitvoering
```

```
;
;

SC0803:
            ; Zet het beginadres van de string op de stack
                  lda     SADR+1
                  pha
                  lda     SADR
                  pha

            ; Zet het beginadres van de substring op de stack
                  lda     SUBADR+1
                  pha
                  lda     SUBADR
                  pha
                  jsr     POS             ; Zoek de positie van de
                                          ; substring
                  brk                     ; Resultaat van het zoeken
                                          ; naar "AAB" in "AAAAAAAAAB" is
                                          ;     Register A = 8

                  jmp     SC0803


; Data
SADR:             .WORD   STG
SUBADR:           .WORD   SSTG
STG:              .BYTE   0AH             ; Lengte van de string
                  .BYTE   "AAAAAAAAAB                          "
                                          ; Max. 32 karakters
SSTG:             .BYTE   3H              ; Lengte van de substring
                  .BYTE   "AAB                                 "
                                          ; Max. 32 karakters

                  .END
```

Kopieer een substring van een string - 8D

Kopieert een substring van een string, gegeven een begin-index en het aantal te kopieren bytes. De strings zijn maximaal 255 bytes lang en de karakters van de strings worden voorafgegaan door een byte met de lengte van de string. Als de begin-index van de substring nul is (d.w.z. als de substring in de lengte-byte zou beginnen) of voorbij de string wijst, krijgt de substring een lengte van nul en wordt de overdrachtvlag op 1 gezet. Als de substring de maximum lengte zou overschrijden of voorbij het einde van de string zou komen, dan wordt alleen het beschikbare aantal karakters (tot en met het einde van de string) in de substring gekopieerd en de overdrachtvlag op 1 gezet. Als de substring als aangegeven kan worden gevormd, wordt de overdrachtvlag gewist.
Procedure: Het programma stopt onmiddellijk als het aantal te kopieren bytes, de maximum lengte van de substring of de begin-index nul is. Het stopt ook onmiddellijk als het beginadres de lengte van de string overschrijdt. Als niets van het bovenstaande geldt, controleert het programma of het aantal te kopieren bytes ofwel de maximale lengte van de substring, ofwel het aantal beschikbare karakters in de string overschrijdt. Als een van de twee het geval is, verkleint het programma het aantal te kopieren bytes overeenkomstig. Vervolgens kopieert het programma het juiste aantal bytes van de string naar de substring. Het programma wist de overdrachtvlag als de opgegeven substring gevormd kan worden en zet de overdrachtvlag als dit niet het geval is.

Gebruikte registers: Alle

Looptijd: Ongeveer 36 * AANTAL GEKOPIEERDE BYTES plus 200 cycli vast. AANTAL GEKOPIEERDE BYTES is het opgegeven aantal (als er zich geen problemen voordoen) of het beschikbare aantal bytes, of de maximale lengte van de substring als het kopieren voorbij het einde van ofwel de substring ofwel de string zou reiken. Als, bij voorbeeld, AANTAL GEKOPIEERDE BYTES = 12 ($0C_{16}$), is de looptijd

36 * 12 + 200 = 432 + 200 = 632 cycli

Programmalengte: 173 bytes

Benodigd datageheugen: Zes bytes ergens in RAM plus vier bytes op pagina 0. De zes bytes ergens in RAM bevatten de lengte van de string (een byte op adres SLEN), de lengte van de substring (een byte op adres DLEN), de maximum lengte van de substring (een byte op adres MAXLEN), de zoekteller (een byte op adres COUNT), de huidige index binnen de string (een byte op adres INDEX) en een foutvlag (een byte op adres CPYERR). De vier bytes op pagina 0 bevatten pointers naar de string (twee bytes vanaf adres DSTRG, $00D0_{16}$ in de listing) en naar de substring (twee bytes vanaf adres SSTRG, $00D2_{16}$ in de listing).

Speciale gevallen:

1. Als het aantal te kopieren bytes nul is, krijgt de substring een lengte van nul en wordt de overdrachtvlag gewist (geen fout opgetreden).
2. Als de maximum lengte van de substring nul is, krijgt de substring een lengte van nul en wordt de overdrachtvlag op 1 gezet (er is een fout opgetreden).

3. Als de begin-index van de substring nul is, krijgt de substring een lengte van nul en wordt de overdrachtvlag op 1 gezet (er is een fout opgetreden).
4. Als de bronstring niet eens tot de aangegeven index reikt, krijgt de substring een lengte van nul en wordt de overdrachtvlag op 1 gezet (er is een fout opgetreden).
5. Als de substring voorbij het einde van de bronstring zou reiken, worden alle beschikbare karakters in de substring geplaatst en wordt de overdrachtvlag op 1 gezet (er is een fout opgetreden). De beschikbare karakters zijn die vanaf de begin-index tot het einde van de string.
6. Als de substring de opgegeven maximum lengte zou overschrijden, dan plaatst het programma alleen het opgegeven maximum aantal karakters in de substring en zet de overdrachtvlag op 1 (er is een fout opgetreden).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Maximum lengte van de substring (bestemmingsstring)
Laagstwaardige byte van het beginadres van de substring (bestemmingsstring)
Hoogstwaardige byte van het beginadres van de substring (bestemmingsstring)
Aantal te kopieren bytes
Begin-index voor het kopieren
Laagstwaardige byte van het beginadres van de string (bronstring)
Hoogstwaardige byte van het beginadres van de string (bronstring)

Eindcondities

Substring bevat karakters gekopieerd van de string. Als de begin-index nul is, de maximum lengte van de substring nul is, of de als de begin-index voorbij de lengte van de string wijst, dan krijgt de substring een lengte van nul en wordt de overdrachtvlag op 1 gezet. Als de substring voorbij de lengte van de string zou reiken of de opgegeven maximum lengte zou overschrijden, dan worden alleen de beschikbare karakters van de string (tot de maximale lengte van de substring) in de substring gekopieerd; hierbij wordt ook de overdrachtvlag op 1 gezet. Als er zich geen problemen voordoen bij het vormen van de string, wordt de overdrachtvlag gewist.

Voorbeelden

1. Data: String = 10'LET Y1 = R7 + X4' ($10_{16} = 16_{10}$ is de lengte van de string)
Maximum lengte van substring = 2
Aantal te kopieren bytes = 2
Begin-index = 5

Resultaat: Substring = 02'Y1' (2 is de lengte van de substring)
Overdrachtvlag = 0, aangezien er geen problemen waren bij het vormen van de substring.

2. Data: String = 0F'POWELL STR 8657' ($0F_{16} = 15_{10}$ is de lengte van de string)
Maximum lengte van substring = $11_{16} = 17_{10}$
Aantal te kopieren bytes = $0E_{16} = 14_{10}$
Begin-index = 1

Resultaat: Substring = 0A'POWELL STR' ($0A_{16} = 10_{10}$ is de lengte van de substring)
Overdrachtvlag = 1, aangezien er niet voldoende karakters in de string waren om het opgegeven aantal karakters te kopieren.

3. Data: String = 11'1068 WC AMSTERDAM' ($11_{16} = 17_{10}$ is de lengte van de string)
Maximum lengte van de substring = 5
Aantal te kopieren bytes = 9
Begin-index = 9

Resultaat: Substring = 05'AMSTE' (05 = de lengte van de substring)
Overdrachtvlag = 1, aangezien het aantal te kopieren bytes de maximum lengte van de substring overschreed.

```
;       Titel      : Kopieer een substring van een string
;       Naam       : COPY
;
;
;       Doel       : Kopieer een substring van een string als een index en het
;                    aantal bytes gegeven zijn


;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Maximale lengte van de substring.
;                      Laagstwaardige byte van het beginadres van de
;                      bestemmingsstring
;                      Hoogstwaardige byte van het beginadres van de
;                      bestemmingsstring
;                      Aantal te kopieeren bytes
;                      Begin-index voor het kopieeren
;                      Laagstwaardige byte van het beginadres van de uitgangs-
;                       string
;                      Hoogstwaardige byte van het beginadres van de uitgangs-
;                       string

;
;
;                    Een string is maximaal 255 bytes lang plus een
;                    lengte-byte die aan de string voorafgaat.
;
;       Eind       : Bestemmingsstring = Substring van de string
```

```
;                   Als er geen fouten zijn gemaakt
;                       Overdrachtvlag = 0
;                   anders
;                       De volgende voorwaarden veroorzaken een fout
;                       en overdracht=1
;                       Als INDEX = 0 of MAXLEN = 0 of
;                              INDEX > lengte (string) dan heeft de
;                       bestemmingsstring een lengte van 0 karakters
;                       Als INDEX + COUNT > lengte (string) dan wordt de
;                              bestemmingsstring gelijk aan de rest van de
;                              string vanaf INDEX
;
;
;     Registers : Alle
;
;     Tijd      : Ongeveer (36 * COUNT) cycli plus 200 basiscycli
;
;     Grootte   : programma  173  bytes
;                 data         6  bytes plus
;                              4  bytes op pagina 0

DSTRG:              .EQU    0D0H            ; Pointer naar bestemmings-
                                            ; string op pagina 0
SSTRG:              .EQU    0F2H            ; Pointer naar uitgangsstring
                                            ; op pagina 0

COPY:
            ; Haal het terugkeeradres van de stack
                    pla
                    tay
                    pla
                    tax

            ; Haal de maximale lengte van de bestemmingsstring van de stack
                    pla
                    sta     MAXLEN

            ; Haal het beginadres van de bestemmingsstring van de stack
                    pla
                    sta     DSTRG
                    pla
                    sta     DSTRG+1

            ; Haal het aantal te kopieeren bytes van de stack
                    pla
                    sta     COUNT

            ; Haal de index van de substring in de string van de stack
                    pla
                    sta     INDEX

            ; Haal het beginadres van de uitgangsstring van de stack
                    pla
                    sta     SSTRG
                    pla
                    sta     SSTRG+1

            ; Zet het terugkeeradres terug op de stack
                    txa
                    pha
                    tya
                    pha
```

```
; Zet de lengte van de uitgangsstring en de foutvlag op 0
        lda     #0
        sta     DLEN            ; De lengte van de
                                ; bestemmingsstring
        sta     CPYERR          ; De foutvlag

; Test of we 0 bytes moeten kopieeren of dat de maximale
; lengte van de substring 0 is
        lda     COUNT
        beq     OKEXIT          ; Spring als we 0 bytes
                                ; moeten kopieeren

        lda     MAXLEN
        beq     EREXIT          ; Spring als de maximale
                                ; lengte van de substring 0 is

        lda     INDEX
        beq     EREXIT          ; Spring als de begin-index 0 is

; Test of de index van de substring binnen de grenzen van
; string ligt, keer anders terug
        ldy     #0
        lda     (SSTRG),Y       ; Haal de lengte op
        sta     SLEN            ; Bewaar de lengte van de
                                ; uitgangsstring
        cmp     INDEX           ; Vergelijk met de index van

                                ; de substring
        bcc     EREXIT          ; Spring als de index te
                                ; groot is

; Test of we niet verder gaan dan het eind van de string
; met het kopieeren.
;  Als INDEX + COUNT - 1 > lengte (uitgangsstring) dan
;     COUNT = lengte (uitgangsstring) - INDEX + 1
        lda     INDEX
        clc
        adc     COUNT
        bcs     RECALC          ; Spring als INDEX + COUNT > 255
        tax
        dex
        cpx     SLEN
        bcc     CNT1OK          ; Spring als INDEX + COUNT - 1
        beq     CNT1OK          ; =< lengte (uitgangsstring)


; De aanroeproutine vroeg om te veel karakters, retourneer
;  alles tussen de index en het einde van de string
; Dus COUNT := lengte (uitgangsstring) - INDEX+ 1
RECALC:
        lda     SLEN            ; Bereken COUNT opnieuw
        sec
        sbc     INDEX
        sta     COUNT
        inc     COUNT           ; COUNT := lengte
                                ; (uitgangsstring) - INDEX + 1
        lda     #0FFH
        sta     CPYERR          ; Zet de foutvlag

; Test of COUNT <= MAXLEN
; Als dat niet het geval is, maak COUNT dan gelijk aan de
; maximale lengte van de bestemmingsstring
```

```
            ;   Als COUNT > MAXLEN dan
            ;       COUNT = MAXLEN
CNT1OK:
                lda     COUNT           ; COUNT > MAXLEN?
                cmp     MAXLEN
                bcc     CNT2OK          ; Spring als COUNT < MAXLEN
                beq     CNT2OK          ; Spring als COUNT = MAXLEN
                lda     MAXLEN
                sta     COUNT           ; COUNT = MAXLEN
                lda     #0FFH
                sta     CPYERR          ; Zet de foutvlag

            ; Alles is nu goed opgezet, dus kunnen we beginnen met
            ; kopieeren
CNT2OK:
                ldx     COUNT           ; Register X = teller
                beq     EREXIT          ;  Spring als COUNT = 0
                lda     #1              ; Begin met het eerste karakter
                sta     DLEN            ; DLEN = index in
                                        ; bestemmingsstring
                                        ; INDEX = index in uitgangsstring

MVLP:
                ldy     INDEX
                lda     (SSTRG),Y       ; Haal volgend karakter op
                ldy     DLEN
                sta     (DSTRG),Y       ; Kopieer
                inc     INDEX           ; Verhoog de index in de
                                        ; uitgangsstring
                inc     DLEN            ; Verhoog de index in de
                                        ; bestemmingsstring
                dex                     ; Verlaag de teller
                bne     MVLP            ; Ga door totdat COUNT = 0
                dec     DLEN            ; lengte (substring) = index
                                        ; in bestemmingsstring - 1
                lda     CPYERR          ; Test of er fouten zijn
                                        ; gemaakt
                bne     EREXIT          ; Zo ja zet dan
                                        ; de overdracht-vlag

OKEXIT:
                clc
                bcc     EXIT

            ; Hier komen we als er een fout is gemaakt
EREXIT:
                sec


            ; Bewaar de lengte-byte vooraan in de bestemmingsstring
EXIT:
                lda     DLEN
                ldy     #0
                sta     (DSTRG),Y       ; Zet het lengte veld goed
                rts


; Data
SLEN:           .BLOCK  1               ; Lengte van de uitgangsstring
DLEN:           .BLOCK  1               ; Lengte van de bestemmingsstring
                                        ;
MAXLEN:         .BLOCK  1               ; Maximale lengte van de
```

```
                                        ; bestemmingsstring
COUNT:          .BLOCK  1               ; Teller bij het zoeken
INDEX:          .BLOCK  1               ; Huidige index in de
                                        ; uitgangsstring
CPYERR:         .BLOCK  1               ; Foutvlag

;
;
; Voorbeeld uitvoering
;
;

SC0804:
        ; Zet het beginadres van de uitgangsstring op de stack
                lda     SADR+1
                pha
                lda     SADR
                pha

        ; Zet de begin index voor het kopieeren op de stack
                lda     IDX
                pha

        ; Zet het aantal te kopieeren karakters op de stack
                lda     CNT
                pha

        ; Zet het beginadres van de bestemmingsstring op de stack
                lda     DADR+1
                pha
                lda     DADR
                pha

        ; Zet de maximale lengte van de bestemmingsstring op de stack
                lda     MXLEN
                pha
                jsr     COPY            ; Kopieer
                brk                     ; Resultaat van het kopieeren
                                        ; van 3 karakters te beginnen
                                        ; bij karakter 4 van de
                                        ; string "12.345E+10" is 3,"345"


                jmp     SC0804


; Data
IDX:            .BYTE   4               ; Index in de string
CNT:            .BYTE   3               ; Aantal bytes
MXLEN:          .BYTE   20H             ; Maximale lengte
SADR:           .WORD   SSTG
DADR:           .WORD   DSTG
SSTG:           .BYTE   0AH             ; Lengte van de uitgangsstring
                .BYTE   "12.345E+10                     "
DSTG:           .BYTE   0               ; Lengte van de bestemmingsstring
                .BYTE   "                               "

                .END
```

Verwijder een substring uit een string - 8E

Verwijdert een substring uit een string, gegeven een begin-index en een lengte. De string is maximaal 255 bytes lang en de karakters van de string worden voorafgegaan door een byte met de lengte van de string. De overdrachtvlag wordt gewist als de opgegeven verwijdering kan gebeuren. De overdrachtvlag wordt gezet als de begin-index nul is of voorbij de lengte van de string wijst; in beide gevallen blijft de string ongewijzigd. Als de verwijdering voorbij het einde van de string reikt, wordt de overdrachtvlag gezet (op 1) en worden alleen de karakters van de begin-index tot het einde verwijderd.
Procedure: Het programma stopt onmiddellijk als de begin-index of het aantal te verwijderen bytes nul is. Het stopt ook als de begin-index voorbij het einde van de string wijst. Als niets van hierboven geldt, dan controleert het programma of de string voorbij het te verwijderen gebied reikt. Zo niet, dan wordt de string ingekort door eenvoudigweg de nieuwe lengte te zetten op de begin-index minus 1. Als de string wel voorbij het te verwijderen gebied reikt, wordt de string verkleind door de bytes boven het verwijderde gebied naar beneden te verplaatsen. Het programma bepaalt vervolgens de lengte van de nieuwe string en stopt met de Overdrachtvlag gewist als het opgegeven aantal bytes verwijderd is en op 1 gezet als er fouten optraden.

Gebruikte registers: Alle

Looptijd: Ongeveer 36 * AANTAL NAAR BENEDEN VERPLAATSTE BYTES + 165 waarbij AANTAL NAAR BENEDEN VERPLAATSTE BYTES nul is als de string afgekort kan worden en gelijk is aan LENGTE - BEGIN-INDEX - AANTAL TE VERWIJDEREN BYTES + 1 als string gecomprimeerd moet worden.

Voorbeelden

1. STRINGLENGTE = 20_{16} (32_{10})
   BEGIN-INDEX = 19_{16} (25_{10})
   AANTAL TE VERWIJDEREN BYTES = 08

   Aangezien er vanaf index 19_{16} precies nog acht bytes in de string zijn, hoeft de routine de string alleen maar in te korten. Dit duurt

   36 * 0 + 165 = 165 cycli

2. STRINGLENGTE = 40_{16} (64_{10})
   BEGIN-INDEX = 19_{16} (25_{10})
   AANTAL TE VERWIJDEREN BYTES = 08

   Aangezien er 20_{16} (32_{10}) bytes boven het ingekorte gebied zijn, moet de routine deze bytes 8 posities omlaag verplaatsen. De looptijd is

   36 * 32 + 165 = 1152 + 165 = 1317 cycli

Programmalengte: 139 bytes

Benodigd datageheugen: Vijf bytes ergens in RAM, plus twee bytes op pagina 0. De vijf bytes ergens in RAM bevatten de lengte van de string (een byte op adres SLEN), de zoekteller (een byte op adres COUNT), een index in de string (een byte op adres INDEX), de bronindex om te gebruiken tijdens de verplaatsing (een byte op adres SIDX) en een foutvlag (een byte op adres DELERR). De twee bytes op pagina 0 bevatten een pointer naar de string (beginnende op adres STRG, $00D0_{16}$ in de listing).

Speciale gevallen:

1. Als het aantal te verwijderen bytes nul is, stopt het programma, waarbij de overdrachtvlag is gewist (geen fouten) en de string ongewijzigd is gebleven.
2. Als de string niet eens tot de opgegeven begin-index reikt, stopt het programma met de overdrachtvlag op 1 gezet (een fout opgetreden), terwijl de string ongewijzigd is gebleven.
3. Als het aantal te verwijderen bytes het beschikbare aantal overtreft, verwijdert het programma alle bytes vanaf de begin-index tot het einde van de string en stopt met de overdrachtvlag op 1 gezet (een fout opgetreden).

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Aantal te verwijderen bytes
Begin-index van te verwijderen bytes
Laagstwaardige byte van het beginadres van de string
Hoogstwaardige byte van het beginadres van de string

Eindcondities

Substring verwijderd uit string. Als er geen fouten optreden, is de overdrachtvlag gewist. Als de begin-index nul is of voorbij de lengte van de string wijst, wordt de overdrachtvlag gezet en blijft de string ongewijzigd. Als het aantal te verwijderen bytes voorbij het einde van de string reikt, wordt de overdrachtvlag gezet en worden de karakters vanaf de begin-index tot het einde van de string verwijderd.

Voorbeelden

1. Data: String = 21'VERKOPEN VOOR MAART EN APRIL 1980' (21_{16} = 33_{10} is de lengte van de string)
Aantal te verwijderen bytes = 09
Begin-index voor verwijdering = 14_{16} = 20_{10}

Resultaat: String 18'VERKOPEN VOOR MAART 1980' (18_{16} = 24_{10} is de lengte van de string waar tien bytes vanaf het 20e karakter zijn

verwijderd - 'EN APRIL' is verdwenen).
Overdrachtvlag = 0 aangezien er geen problemen optraden.

2. Data: String = 29'DE PRIJS IS FL.3,00 (FL.2,00 VOOR 1 JUNI)' (29_{16} = 41_{10} is de lengte van de string).
Aantal te verwijderen bytes = 48
Begin-index voor verwijdering = 14_{16} = 20_{10}

Resultaat: String = 13'DE PRIJS IS FL.3,00' (13_{16} = 19_{10} is de lengte van de string waarvan alle resterende bytes verwijderd zijn).
Overdrachtvlag = 1, aangezien er minder karakters in de string over waren dan er verwijderd hadden moeten worden.

```
;       Titel      : Verwijder een substring uit een string
;       Naam       : DELETE
;
;
;       Doel       : Verwijder een substring uit een string als de index en
;                    de lengte gegeven zijn
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Het aantal bytes dat verwijderd moet worden
;                      De index in de string, waarvanaf verwijderd moet worden
;                      Laagstwaardige byte van het beginadres van de string
;                      Hoogstwaardige byte van het beginadres van de string
;
;                      Een string is maximaal 255 bytes lang plus een
;                      lengte byte die aan de string voorafgaat
;
;       Eind       : De substring is verwijderd
;                    Als er geen fouten zijn gemaakt dan
;                      overdrachtvlag = 0
;                    anders
;                         de volgende condities veroorzaken een fout, waardoor
;                         de overdrachtvlag gezet wordt:
;                          Als de index 0 of groter dan de lengte van de
;                           string is dan wordt de string niet veranderd
;                          Als de lengte van de substring te groot is dan
;                           worden de karakters vanaf de index tot het einde
;                           van de string verwijderd
;
;       Registers : Alle
;
;       Tijd       : Ongeveer 36 * (lengte v/d string - index - het aantal
;                    karakters dat verwijderd moet worden) plus 165 basiscycli
;
;       Grootte    : programma 139  bytes
;                    data        5  bytes plus
;                                2 bytes op pagina 0

STRG:                .EQU    0D0H            ; Pointer naar string op
                                             ; pagina 0
```

```
DELETE:
        ; Haal het terugkeeradres van de stack
              pla
              tay
              pla
              tax

        ; Haal het aantal te verwijderen bytes van de stack
              pla
              sta     COUNT

        ; Haal de begin-index in string van de stack
              pla
              sta     INDEX

        ; Haal het beginadres van de string van de stack
              pla
              sta     STRG
              pla
              sta     STRG+1

        ; Zet het terugkeeradres terug op de stack
              txa
              pha
              tya
              pha

        ; Initializeer de foutvlag op 0
        ; Haal de lengte van de string op
              ldy     #0
              sty     DELERR
              lda     (STRG),Y        ; De lengte van de string
              sta     SLEN            ; Bewaar de lengte

        ; Test of COUNT en index allebei ongelijk aan 0 zijn
              lda     COUNT
              beq     OKEXIT          ; Spring als er niets ver-
                                      ; wijderd hoeft te worden
              lda     INDEX
              beq     EREXIT          ; Spring als de index 0 is

        ; Test of de begin-index tussen de grenzen van de string
        ; ligt, keer terug als dat niet zo is
              lda     SLEN            ; Is de index O.K?
              cmp     INDEX
              bcc     EREXIT          ; Nee, keer terug

        ; Test of het aantal te verwijderen karakters in de string
        ; aanwezig is, verwijder anders alle karakters tot en met
        ; het einde van de string
              lda     INDEX
              clc
              adc     COUNT
              bcs     TRUNC           ; Verwijder tot en met het
                                      ; einde van de string
              sta     SIDX            ; Bewaar INDEX + COUNT als
                                      ; index in de string
              tax                     ; X = INDEX + COUNT
              dex
              cpx     SLEN
              bcc     CNTOK           ; Spring als INDEX + COUNT - 1
                                      ; < lengte (string) anders
```

```
                                        ; Verwijder rest string
            beq     TRUNC               ; Verwijder zonder foutmelding
            lda     #0FFH
            sta     DELERR              ; Zet de foutvlag

        ; Verwijder de rest van de string. Er hoeft niet geschoven
        ; te worden
TRUNC:
            ldx     INDEX               ; Lengte(string) = INDEX - 1
            dex
            stx     SLEN
            lda     DELERR
            beq     OKEXIT              ; Geen foutmelding
            bne     EREXIT              ; Wel een foutmelding

        ; Verwijder de substring door alle karakters boven het
        ; stuk dat verwijderd moet worden, omlaag te schuiven
        ; tot aan het begin van het te verwijderen gedeelte
CNTOK:
        ; Bereken hoeveel karakters verplaatst moeten worden
        ;  (SLEN - SIDX + 1)
            lda     SLEN
            sec
            sbc     SIDX
            tax
            inx
            beq     OKEXIT              ; Spring als COUNT = 0


MVLP:
            ldy     SIDX
            lda     (STRG),Y            ; Haal het volgende karakter op
            ldy     INDEX
            sta     (STRG),Y            ; Schuif het naar beneden
            inc     INDEX               ; Verhoog de indices
            inc     SIDX
            dex                         ; Verlaag de teller
            bne     MVLP                ; Ga door totdat teller = 0
            ldx     INDEX
            dex                         ; De lengte van de
                                        ; string = INDEX - 1
            stx     SLEN

        ; Alles is goed gegaan
OKEXIT:
            clc
            bcc EXIT

        ; Er is iets misgegaan
EREXIT:
            sec


EXIT:
            lda     SLEN
            ldy     #0
            sta     (STRG),Y            ; Zet de lengte van de nieuwe
                                        ; string goed
            rts


; Data
```

```
SLEN:           .BLOCK  1           ; Lengte van de string
COUNT:          .BLOCK  1           ; Teller
INDEX:          .BLOCK  1           ; Huidige index in de string
SIDX:           .BLOCK  1           ; Index tijdens het schuiven
DEPERR:         .BLOCK  1           ; Foutvlag


;
;
; Voorbeeld uitvoering
;
;

SC0805:
        ; Zet het beginadres van de string op de stack
                lda     SADR+1
                pha
                lda     SADR
                pha

        ; Zet de begin-index op de stack
                lda     IDX
                pha

        ; Zet het aantal te verwijderen karakters op de stack
                lda     CNT
                pha
                jsr     DELETE          ; Verwijder ze
                brk                     ; Resultaat van het
                                        ; verwijderen van 4 karakters
                                        ; te beginnen bij karakter 1
                                        ; van "JOE HANDOVER" is
                                        ; "HANDOVER"

                jmp     SC0805


; Data
IDX:            .BYTE   1               ; Begin-index in de string
CNT:            .BYTE   4               ; Het aantal te verwijderen
                                        ; karakters
SADR:           .WORD   SSTG
SSTG:           .BYTE   12              ; Lengte van de string
                .BYTE   "JOE HANDOVER"


                .END
```

Voeg een substring in een string - 8F

Voegt een substring in een string, gegeven een begin-index. De string en de substring zijn beide maximaal 255 bytes lang en de karakters van de string worden voorafgegaan door een byte die de lengte bevat. De overdrachtvlag wordt gewist als de invoeging zonder problemen plaats kan vinden. De overdrachtvlag wordt gezet als de begin-index nul is of voorbij het einde van de string wijst. In dit laatste geval wordt de substring aan het einde van de string geplaatst. De overdrachtvlag wordt ook gezet als de string met de invoeging de opgegeven maximum lengte zou overschrijden; in dat laatste geval wordt alleen zoveel van de substring aan de string toegevoegd, dat de string de maximum lengte krijgt.
Procedure: Het programma stopt onmiddellijk als de begin-index nul is of als de lengte van de substring nul is. Als aan geen van deze condities wordt voldaan, controleert het programma of de tussenvoeging een string oplevert die groter is dan het maximum. Als dit het geval zou zijn, kapt het programma de substring af. Vervolgens controleert het programma of de begin-index binnen de string valt. Zo niet, dan plakt het programma de substring eenvoudigweg aan de string door deze naar de geheugenlocaties onmiddellijk achter de string te verplaatsen. Als de begin-index binnen de string is, moet het programma eerst ruimte voor de tussenvoeging maken door de resterende karakters hoger in het geheugen te zetten. Deze verplaatsing moet vanaf het hoogste adres beginnen om te voorkomen dat gegevens overschreven worden. Dan kan het programma uiteindelijk de substring in het open gebied voegen. Het programma controleert dan de nieuwe stringlengte en stopt met de overdrachtvlag overeenkomstig gezet (0 als er geen problemen waren en 1 als de begin-index nul was, de substring afgekapt moest worden of het beginadres voorbij de lengte van de string wees).

Gebruikte Registers: Alle

Looptijd: Ongeveer 36 * AANTAL VERPLAATSTE BYTES + 36 * AANTAL TUSSENGEVOEGDE BYTES + 207. AANTAL VERPLAATSTE BYTES is het aantal bytes dat naar een bovenliggende vrije geheugenruimte moet worden verplaatst voor de tussenvoeging. Als de begin-index voorbij het einde van de string wijst, is dit nul aangezien de substring eenvoudigweg aan de string wordt vastgeplakt. Anders is dit STRINGLENGTE - BEGIN-INDEX + 1, aangezien de bytes op of boven de begin-index verplaatst moeten worden.
AANTAL TUSSENGEVOEGDE BYTES is de lengte van de substring als deze niet afgekapt hoeft te worden. Het is de maximum lengte van de string minus de huidige lengte als tussenvoeging van de substring een string zou opleveren die langer dan de maximum lengte is.

Voorbeelden

1. STRINGLENGTE = 20_{16} (32_{10})
   BEGIN-INDEX = 19_{16} (25_{10})
   MAXIMUM LENGTE = 30_{16} (48_{10})
   SUBSTRINGLENGTE = 06

   Met andere woorden, we willen een zes bytes lange substring vanaf het 25e karakter tussenvoegen. Aangezien er acht bytes zijn die naar boven verplaatst moeten worden (20_{16} - 19_{16} + 1

= AANTAL VERPLAATSTE BYTES) en zes bytes die tussengevoegd moeten worden, is de looptijd ongeveer

36 * 8 + 36 * 6 + 207 = 288 + 216 + 207 = 711 cycli.

2. STRINGLENGTE = 20_{16} (32_{10})
   BEGIN-INDEX = 19_{16} (25_{10})
   MAXIMUM LENGTE = 24_{16} (36_{10})
   SUBSTRINGLENGTE = 06

   In tegenstelling tot voorbeeld 1, kunnen hier slechts vier bytes van de substring tussengevoegd worden zonder de maximum lengte van de string te overschrijden. Zodoende, het AANTAL VERPLAATSTE BYTES = 8 en AANTAL TUSSENGEVOEGDE BYTES = 4. De looptijd is ongeveer

   36 * 8 + 36 * 4 + 207 = 288 + 144 + 207 = 639 cycli.

Programmalengte: 212 bytes

Benodigd datageheugen: Zeven bytes ergens in RAM, plus vier bytes op pagina 0. De zeven bytes ergens in RAM bevatten de lengte van de string (een byte op adres SLEN), de lengte van de substring (een byte op adres SUBLEN), de maximum lengte van de string (een byte op adres MAXLEN), de huidige index in de string (een byte op adres INDEX), lopende indexen voor gebruik tijdens de verplaatsing (een byte op adres SIDX en een byte op adres DIDX) en een foutvlag (een byte op adres INSERR). De vier bytes op pagina 0 bevatten pointers naar de substring (twee bytes vanaf adres SUBSTG, $00D0_{16}$ in de listing) en de string (twee bytes vanaf adres STRG, $00D2_{16}$ in de listing).

Speciale gevallen:

1. Als de lengte van de substring (het tussenvoegsel) nul is, stopt het programma met de overdrachtvlag gewist (geen fout) en laat de string ongewijzigd.
2. Als de begin-index voor de tussenvoeging nul is (d.w.z. de tussenvoeging begint in de lengtebyte), stopt het programma met de overdrachtvlag op 1 gezet (er is een fout opgetreden) en de string ongewijzigd.
3. Als de string met de tussengevoegde substring de opgegeven maximum lengte overschrijdt, voegt het programma alleen zoveel karakters tussen, dat de maximum lengte wordt bereikt. De overdrachtvlag wordt op 1 gezet om aan te geven dat de tussenvoeging afgekapt is.
4. Als de begin-index van de tussenvoeging voorbij het einde van de string wijst, plakt het programma de substring aan het einde van de string en geeft een fout aan door de overdracht op 1 te zetten.
5. Als de oorspronkelijke lengte van de string de opgegeven maximum lengte overschrijdt, stopt het programma met de overdrachtvlag op 1 gezet (er is een fout opgetreden) en laat de string ongewijzigd.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het beginadres van de substring
Hoogstwaardige byte van het beginadres van de substring
Maximum lengte van de string
Begin-index waar de substring ingevoegd moet worden
Laagstwaardige byte van het beginadres van de string
Hoogstwaardige byte van het beginadres van de string

Eindcondities

Substring in string gevoegd. Als er zich geen fouten voordoen, is de overdrachtvlag gewist. Als de begin-index of de lengte van de substring nul is, wordt de overdrachtvlag gezet en blijft de string ongewijzigd. Als de begin-index voorbij de lengte van de string wijst, wordt de overdrachtvlag gezet en de substring aan het einde van de string vastgeplakt. Als de string met de tussengevoegde substring de opgegeven maximum lengte zou overschrijden, wordt de overdrachtvlag gezet en worden alleen die karakters van de substring tussengevoegd die de string tot de maximum lengte brengen.

Voorbeelden

1. Data:

   String = 0A'KEES GROOT' ($0A_{16} = 10_{10}$ is de lengte van de string)
   Substring = 09'HANNIBAL ' (09 is de lengte van de substring)
   Maximum lengte van string = $14_{16} = 20_{10}$
   Begin-index = 06

   Resultaat:

   String = 13'KEES HANNIBAL GROOT' ($13_{16} = 19_{10}$ is de lengte van de string met de tussengevoegde substring).
   Overdrachtvlag = 0, aangezien er geen problemen waren.

2. Data:

   String = 0A'KEES GROOT' ($0A_{16} = 10_{10}$ is de lengte van de string)
   Substring = 0C'ROCKEFELLER ' ($0C_{16} = 12_{10}$ is de lengte van de substring)
   Maximum lengte van string = $14_{16} = 20_{10}$
   Begin-index = 06

   Resultaat:

   String = 14'KEES ROCKEFELLEGROOT' ($14_{16} = 20_{10}$ is de lengte van de string met tot de maximum lengte karakters van de substring tussengevoegd).
   Overdrachtvlag = 1, aangezien de substring niet kon worden tussengevoegd zonder de maximum lengte van de string te overschrijden.

```
;
;
;
;       Titel       : Voeg een substring in in een string
;       Naam        : INSERT
;
;
;       Doel        : Voeg een substring toe aan een string bij
;                     een gegeven index
;
;       Begin       : Top van de stack
;                       Laagstwaardige byte van het terugkeeradres
;                       Hoogstwaardige byte van het terugkeeradres
;                       Laagstwaardige byte van het adres van de substring
;                       Hoogstwaardige byte van het adres van de substring
;                       Maximale lengte van de string
;                       Begin-index in de string waar de substring tussen moet
;                       Laagstwaardige byte van het adres van de string
;                       Hoogstwaardige byte van het adres van de string
;
;                       Een string is maximaal 255 bytes lang plus een
;                       lengte-byte die aan de string voorafgaat
;
;
;       Eind        : De substring zit in de string
;                     Als er geen fouten zijn gemaakt dan
;                       Overdrachtvlag = 0
;                     anders
;                          De volgende condities veroorzaken dat de
;                          overdrachtvlag gezet wordt :
;                           Als de index = 0 dan wordt de string niet veranderd
;                           Als de lengte van de string groter dan de maximale
;                            lengte is dan wordt de string niet veranderd
;                           Als de index groter dan de lengte van de string is
;                            dan wordt de substring achter de string geplakt
;                           Als de lengte van de string + de lengte van de sub-
;                            string > maximale lengte is dan wordt net zoveel
;                            van de substring toegevoegd tot de maximale lengte
;                            is bereikt
;
;       Registers : Alle
;
;       Tijd        : Ongeveer
;                       36 * (lengte van de string  - index + 1) +
;                       36 * (lengte van de substring) +
;                       207 basiscycli
;
;       Grootte     : programma 214  bytes
;                     data        7  bytes plus
;                                 4  bytes op pagina 0

SUBSTG:             .EQU    0D0H            ; Pointer naar substring op
                                            ; pagina 0
STRG:               .EQU    0D2H            ; Pointer naar string op
                                            ; pagina 0

INSERT:
            ; Haal het terugkeeradres van de stack
                    pla
                    tay
                    pla
```

```
        tax

; Haal het beginadres van de substring van de stack
        pla
        sta     SUBSTG
        pla
        sta     SUBSTG+1

; Haal de maximale lengte van de string van de stack
        pla
        sta     MAXLEN

; Haal de begin-index voor het invoegen, van de stack
        pla
        sta     INDEX

; Haal het beginadres van de string van de stack
        pla
        sta     STRG
        pla
        sta     STRG+1

; Zet het terugkeeradres terug op de stack
        txa
        pha
        tya
        pha

; Veronderstel dat er geen fouten worden gevonden
        lda     #0
        sta     INSERR

; Haal de lengtes van de substring en de string op
;  Als de lengte van de substring gelijk aan 0 is,
;  keer dan terug zonder foutmelding
        ldy     #0
        lda     (STRG),Y        ; Haal het lengte-veld op
        sta     SLEN
        lda     (SUBSTG),Y      ; Haal het andere lengte-veld op
        sta     SUBLEN
        bne     IDX0
        jmp     OKEXIT          ; Spring als er niets tussen-
                                ; gevoegd hoeft te worden

; Keer terug met een foutmelding als de begin-index 0 is
IDX0:
        lda     INDEX
        bne     CHKLEN
        jmp     EXEXIT

; Test of het resultaat van het tussenvoegen van de sub
; string in de string nog in de string zelf past.
; Als dat niet het geval is neem dan maar een gedeelte van
; de substring en zet de foutvlag aan
CHKLEN:
        lda     SUBLEN          ; De lengte van de substring
        clc
        adc     SLEN
        bcs     TRUNC           ; Spring als
                                ; de nieuwe lengte > 255 is
        cmp     MAXLEN
```

```
        bcc     IDXLEN          ; Spring als lengte < MAXLEN
        beq     IDXLEN          ; Spring als lengte = MAXLEN

; De substring past niet in zijn geheel, bereken dus
; welk stuk van de substring gebruikt kan worden
; Lengte(substring) = MAXLEN - lengte(string)
TRUNC:
        lda     MAXLEN
        sec
        sbc     SLEN
        bcc     EREXIT          ; Keer terug als
                                ; MAXLEN < lengte(string)
        beq     EREXIT          ; Keer terug als de string
                                ; zelf te lang was
        sta     SUBLEN
        lda     #0FFH
        sta     INSERR          ; Zet de foutvlag


; Test of de index binnen de grenzen van de string ligt
;  Als dat niet het geval is, plak de substring dan maar
; aan het eind van de string vast
IDXLEN:
        lda     SLEN            ; Lengte(string)
        cmp     INDEX           ; Vergelijk met de index
        bcs     LENOK           ; Spring als de index O.K. is
        ldx     SLEN            ; Plak de substring aan het
                                ; van de string vast
        inx
        stx     INDEX           ; Begin precies aan het einde
                                ; van de string
        lda     #0FFH
        sta     INSERR          ; Zet de foutvlag
        lda     SLEN            ; Tel lengte(substring) bij
                                ; lengte(string) op
        clc
        adc     SUBLEN
        sta     SLEN
        jmp     MVESUB          ; Schuif alleen, er hoeft
                                ; geen ruimte open te worden
                                ; gemaakt in string

; Maak ruimte in de string, zodat de substring precies in
; die ruimte past, door de karakters vanaf de index tot en
; met het einde van de string op te schuiven
LENOK:
; Bereken hoeveel karakters verschoven moeten worden
;  COUNT = lengte(string) - begin-index + 1
        lda     SLEN
        sec
        sbc     INDEX
        tax
        inx                     ; X = aantal karakters

; Zet de begin-index goed en bereken de index in het
; resultaat
        lda     SLEN
        sta     SIDX            ; Begin-index eindigt bij het
                                ; einde van de originele string
        clc
        adc     SUBLEN
        sta     DIDX            ; Resultaat eindigt verder
                                ; door de substring
```

```
                sta     SLEN            ; Zet de nieuwe lengte meteen
                                        ; goed

OPNLP:
                ldy     SIDX
                lda     (STRG),Y        ; Haal volgende karakter op
                ldy     DIDX
                sta     (STRG),Y        ; Schuif het omhoog
                dec     SIDX            ; Verlaag de indices
                dec     DIDX
                dex                     ; Verlaag de teller
                bne     OPNLP           ; Ga door tot teller = 0

        ; Plaats de substring in het opengemaakte stuk
MVESUB:
                lda     #1
                sta     SIDX            ; Begin met het eerste
                                        ; karakter van de substring
                                        ; Begin bij index in de string
                ldx     SUBLEN          ; X = aantal karakters

MVELP:
                ldy     SIDX
                lda     (SUBSTG),Y      ; Haal volgende karakter op
                ldy     INDEX
                sta     (STRG),Y        ; Zet het op zijn plaats
                inc     SIDX            ; Verhoog de indices
                inc     INDEX
                dex                     ; Verlaag de teller
                bne     MVELP           ; Ga door tot teller = 0
                lda     INSERR          ; Haal de foutvlag op
                bne     EREXIT          ; Er is ergens een fout gevonden


OKEXIT:
                clc                     ; Geen fouten
                bcc     EXIT


EREXIT:
                sec                     ; Foutje


EXIT:
                lda     SLEN
                ldy     #0
                sta     (STRG),Y        ; Zet het lengte-veld goed
                rts


; Data
SLEN:           .BLOCK  1               ; Lengte van de string
SUBLEN:         .BLOCK  1               ; Lengte van de substring
MAXLEN:         .BLOCK  1               ; Maximale lengte van de string
INDEX:          .BLOCK  1               ; Index in de string
SIDX:           .BLOCK  1               ; Index tijdens het schuiven
DIDX:           .BLOCK  1               ; Index tijdens het schuiven
INSERR:         .BLOCK  1               ; Foutvlag
```

```
;
; Voorbeeld uitvoering
;
;

SC0806:
            ; Zet het beginadres van de string op de stack
                  lda     SADR+1
                  pha
                  lda     SADR
                  pha

            ; Zet de begin-index op de stack
                  lda     IDX
                  pha

            ; Zet de maximale lengte op de stack
                  lda     MXLEN
                  pha

            ; Zet het beginadres van de substring op de stack
                  lda     SUBADR+1
                  pha
                  lda     SUBADR
                  pha

                  jsr     INSERT          ; Voeg de substring bij de string
                  brk                     ; Resultaat van het tussen-
                                          ; voegen van "-" in "123456"
                                          ; op positie 1 is
                                          ; "-123456"

                  jmp     SC0806


; Data
IDX:              .BYTE   1               ; Begin-index
MXLEN:            .BYTE   20H             ; Maximale lengte
SADR:             .WORD   STG             ; Beginadres string
SUBADR:           .WORD   SSTG            ; Beginadres substring
STG:              .BYTE   06H             ; Lengte string
                  .BYTE   "123456                        "
SSTG:             .BYTE   1               ; Lengte substring
                  .BYTE   "-                             "


                  .END
```

8-Bits array optelling - 9A

Telt de elementen op van een array ter grootte van een byte en produceert een 16-bits som. De grootte van het array is opgegeven en bedraagt maximaal 255 bytes.
Procedure: Het programma wist eerst beide bytes van de som. Vervolgens telt het de elementen opeenvolgend op bij de laagstwaardige byte van de som, beginnende bij het element met het hoogste adres. Telkens wanneer de optelling een overdracht genereert, verhoogt het programma de hoogstwaardige byte van het element.

Gebruikte registers:	Alle
Looptijd:	Ongeveer 16 cycli per byte plus 39 cycli vast. Als, bij voorbeeld, $(X) = 1A_{16} = 26_{10}$, dan is de looptijd ongeveer $16 * 26 + 39 = 416 + 39 = 455$ cycli
Programmalengte:	30 bytes
Benodigd datageheugen:	Twee bytes op pagina 0, die de pointer naar het array bevatten (vanaf adres ARYADR, $00D0_{16}$ in de listing).
Speciaal geval:	Het programma maakt de som nul en stopt bij een array-grootte van nul.

Begincondities

(A) = Hoogstwaardige byte van het beginadres van het array
(Y) = Laagstwaardige byte van het beginadres van het array
(X) = Grootte van het array in bytes

Eindcondities

(A) = Hoogstwaardige byte van de som
(X) = Laagstwaardige byte van de som

Voorbeeld

Data: Grootte van array (in bytes) = (X) = 08
Array elementen

$F7_{16} = 247_{10}$
$23_{16} = 35_{10}$
$31_{16} = 49_{10}$
$70_{16} = 112_{10}$
$5A_{16} = 90_{10}$
$16_{16} = 22_{10}$
$CB_{16} = 203_{10}$
$E1_{16} = 225_{10}$

Resultaat: Som = $03D7_{16} = 983_{10}$
(A) = Hoogstwaardige byte van som = 03_{16}
(Y) = Laagstwaardige byte van som = $D7_{16}$

```
;       Titel      : 8-bit array optelling
;       Naam       : ASUM8
;
;
;       Doel       : Tel de data uit een array bij elkaar op en produceer een
;                    16-bits resultaat.
;                    Maximale array grootte is 255 bytes.
;
;
;       Begin      : Register A = hoogstwaardige byte van het beginadres van
;                                 het array
;                    Register Y = laagstwaardige byte van het beginadres van
;                                 het array
;                    Register X = lengte van het array in bytes
;
;       Eind       : Register A = hoogstwaardige byte van de som
;                    Register Y = laagstwaardige byte van de som
;
;       Registers  : Alle
;
;       Tijd       : Ongeveer 16 cycli per byte plus 39 basiscycli
;
;       Grootte    : programma 30  bytes
;                    data       2  bytes op pagina 0

ARYADR:              .EQU   0D0H            ; Pointer naar array op
                                            ; pagina 0

ASUM8:
            ; Bewaar het beginadres van het array
                  sty     ARYADR
                  sta     ARYADR+1

            ; Verlaag het beginadres met 1, om efficienter te kunnen
            ; werken
                  tya                        ; Haal de laagstwaardige
                                             ; byte op
                  bne     ASUM8              ; Gelijk aan 0?
                  dec     ARYADR+1           ; Leen dan van de
                                             ; hoogstwaardige byte
ASUM8:              dec     ARYADR           ; Verlaag altijd de
                                             ; laagstwaardige byte

            ; Keer terug als de lengte van het array gelijk aan 0 is
                  txa
                  tay
                  beq     EXIT

            ; Initialisatie
                  lda     #0                 ; Maak de som gelijk aan 0
                  tax


            ; Tel alle elementen bij elkaar op
```

```
SUMLP:
                clc
                adc     (ARYADR),Y      ; Tel de volgende byte bij
                                        ; de laagstwaardige byte
                                        ; van de som op
                bcc     DECCNT
                inx                     ; Verhoog hoogstwaardige
                                        ; byte van de som als de
                                        ; overdracht-vlag gezet was

DECCNT:
                dey                     ; Verlaag de lus-teller
                bne     SUMLP           ; Ga door totdat alle
                                        ; elementen bij elkaar op-
                                        ; geteld zijn


EXIT:
                tay                     ; Register Y = laagstwaardige
                                        ; byte
                txa                     ; Register A = hoogstwaardige
                                        ; byte
                rts


;
;
; Voorbeeld uitvoering
;
;

SC0901:
          ; Zet het beginadres van het array in A,Y
                ldy     BUFADR
                lda     BUFADR+1

          ; Register X = lengte van het array
                ldx     BUFSZ
                jsr     ASUM8           ; Bereken de som
                brk                     ; Resultaat van de optelling
                                        ; van de test-data is
                                        ;  07F8 hexadecimaal
                                        ;
                                        ; Register A = 07H
                                        ; Register Y = F8H

                jmp     SC0901


; Test-data, verander deze zelf om andere waarden te testen

SIZE:           .EQU    010H            ; Lengte van het array
BUFADR:         .WORD   BUF             ; Beginadres van het array
BUFSZ:          .WORD   SIZE            ; Lengte van het array

BUF:            .BYTE   00H             ; De elementen van het array
                .BYTE   11H             ; zijn:
                .BYTE   22H             ;
                .BYTE   33H             ; 0, 17, 34, 51, 68, 85, 102,
                .BYTE   44H             ; 119, 136, 153, 170, 187, 204,
                .BYTE   55H             ; 221, 238, 255
                .BYTE   66H
```

```
.BYTE   77H
.BYTE   88H
.BYTE   99H
.BYTE   0AAH
.BYTE   0BBH
.BYTE   0CCH

.BYTE   0DDH
.BYTE   0EEH
.BYTE   0FFH            ; De som is 07F8 (2040
                        ; decimaal)

.END
```

16-Bits array optelling - 9B

Telt de elementen op van een array ter grootte van een woord, waarbij een 24-bits som teruggegeven wordt. De grootte van het array wordt opgegeven en mag niet meer dan 255 16-bits woorden bedragen. De 16-bits elementen zijn op de gebruikelijke 6502 wijze opgeslagen met de laagstwaardige byte eerst.
Procedure: Het programma wist een 24-bits accumulator in drie bytes van het geheugen en telt vervolgens vanaf het laagste adres de elementen op bij de geheugenaccumulator. De hoogstwaardige byte van de geheugenaccumulator wordt telkens verhoogd als de optelling van de hoogstwaardige byte van een element met de middelste byte van de som een overdracht genereert. Als het array meer dan 1 geheugenpagina in beslag neemt, moet het programma de hoogstwaardige byte van de array-pointer verhogen vooraleer het naar de tweede pagina gaat.

Gebruikte registers:	Alle
Looptijd:	Ongeveer 43 cycli per byte, plus 46 cycli vast. Als, bij voorbeeld, (X) = 12_{16} = 18_{10}, dan is de looptijd ongeveer 43 * 18 + 46 = 774 + 46 = 820 cycli
Programmalengte:	60 bytes
Benodigd datageheugen:	Drie bytes ergens in RAM, plus twee bytes op pagina 0. De drie bytes ergens in RAM bevatten de geheugenaccumulator (vanaf adres SUM); de twee bytes op pagina 0 bevatten een pointer naar het array (vanaf adres ARYADR, $00D0_{16}$ in de listing).
Speciaal geval:	Bij een array-grootte van 0 stopt het programma met de som op nul gezet.

Begincondities

(A) = Hoogstwaardige byte van het beginadres van het array
(Y) = Laagstwaardige byte van het beginadres van het array
(X) = Grootte van het array in 16-bits woorden

Eindcondities

(X) = Hoogstwaardige byte van de som
(A) = Middelste byte van de som
(Y) = Laagstwaardige byte van de som

Voorbeeld

Data: Grootte van array (in 16-bits woorden) = (X) = 08
Array-elementen

$F7A1_{16} = 63.393_{10}$
$239B_{16} = 9.115_{10}$

$31D5_{16} = 12.757_{10}$
$70F2_{16} = 28.914_{10}$
$5A36_{16} = 23.094_{10}$
$166C_{16} = 5.740_{10}$
$CBF5_{16} = 52.213_{10}$
$E107_{16} = 57.607_{10}$

Resultaat: Som = $03DBA1_{16} = 252.833_{10}$
(X) = Hoogstwaardige byte van de som = 03_{16}
(A) = Middelste byte van de som = DB_{16}
(Y) = Laagstwaardige byte van de som = $A1_{16}$

```
;       Titel      : 16-bit array optelling
;       Naam       : ASUM16
;
;
;       Doel       : Tel de data uit een array bij elkaar op en produceer een
;                    24-bits resultaat.
;                    Maximale array grootte is 255 16-bits elementen.
;
;
;       Begin      : Register A = hoogstwaardige byte van het beginadres van
;                                 het array
;                    Register Y = laagstwaardige byte van het beginadres van
;                                 het array
;                    Register X = lengte van het array in 16-bits elementen
;
;       Eind       : Register X = hoogstwaardige byte van de som
;                    Register A = middelste byte van de som
;                    Register Y = laagstwaardige byte van de som
;
;       Registers  : Alle
;
;       Tijd       : Ongeveer 43 cycli per byte plus 46 basiscycli
;
;       Grootte    : programma 60  bytes
;                    data        3  bytes plus
;                                2  bytes op pagina 0

ARYADR:             .EQU   0D0H            ; Pointer naar array op
                                           ; pagina 0


ASUM16:
            ; Bewaar het beginadres van het array
                    sty     ARYADR
                    sta     ARYADR+1

            ; Initialiseer de som en de index variabelen
                    lda     #0
                    sta     SUM            ; Som = 0
                    sta     SUM+1
                    sta     SUM+2
                    tay                    ; Index = 0

            ; Keer terug als de lengte van het array gelijk aan 0 is
                    txa
                    beq      EXIT
```

```
            ; Tel alle elementen bij elkaar op
SUMLP:
                lda     SUM
                clc
                adc     (ARYADR),Y      ; Eerst de laagstwaardige byte
                sta     SUM
                lda     SUM+1
                iny                     ; Verhoog de index
                adc     (ARYADR),Y      ; Tel nu de hoogstwaardige
                                        ;  byte van het array-element
                                        ;  bij SUM op
                sta     SUM+1
                bcc     NXTELM
                inc     SUM+2           ; Als overdrachtvlag = 1
                                        ; verhoog dan ook de
                                        ; hoogstwaardige byte van de
                                        ; som

NXTELM:
                iny                     ; Wijs naar volgend array-
                                        ; element
                bne     DECCNT
                inc     ARYADR+1        ; Wijs nu naar de tweede
                                        ; pagina van het array

DECCNT:
                dex                     ; Verlaag de teller
                bne     SUMLP           ; Ga door tot register X = 0


EXIT:
                ldy     SUM             ; Register Y = laagstwaardige,
                lda     SUM+1           ; Register A = middelste,
                ldx     SUM+2           ; Register X = hoogstwaardige
                                        ;  byte van de som
                rts


; Data
SUM:            .BLOCK  3               ; Tijdelijke accumulator van
                                        ; de som




;
;
; Voorbeeld uitvoering
;
;

SC0902:
            ; Stop het beginadres van de buffer in A, Y
                ldy     BUFADR
                lda     BUFADR+1

            ; Stop de lengte van het array in X
                ldx     BUFSZ
                jsr     ASUM16          ; Tel de elementen bij elkaar
                                        ; op
```

```
                brk                     ; Resultaat van de test-data
                                        ; is 12570
                                        ;
                                        ;
                                        ;  Register X = 0
                                        ;  Registeı A = 31H
                                        ;  Register Y = 1AH
                jmp     SC0902

; Test-data, verander deze zelf om andere waarden te testen

SIZE:           .EQU    010H            ; Lengte van de buffer
BUFADR:         .WORD   BUF             ; Beginadres van de buffer
BUFSZ:          .WORD   SIZE            ; Lengte van de buffer

BUF:
        ; De buffer met de array elementen
                .BYTE   0
                .BYTE   111
                .BYTE   222
                .BYTE   333
                .BYTE   444
                .BYTE   555
                .BYTE   666
                .BYTE   777
                .BYTE   888
                .BYTE   999
                .BYTE   1010
                .BYTE   1111
                .BYTE   1212
                .BYTE   1313
                .BYTE   1414
                .BYTE   1515            ; Som = 12570 = 311AH

                .END
```

Zoek element met maximale byte lengte - 9C

Zoekt het maximum element in een array met ongetekende elementen ter grootte van een byte. De grootte van het array is opgegeven en overschrijdt niet de 255 bytes.
Procedure: Het programma stopt met de overdrachtvlag op 1 gezet als de array-grootte nul is. Als de grootte ongelijk nul is, veronderstelt het programma dat de laatste byte van het array de grootste is en werkt het array dan verder van achteren naar voren door, waarbij het verondersteld grootste element met het huidige element wordt vergeleken. De grootste waarde en de index ervan worden bewaard. Uiteindelijk wist het programma de overdrachtvlag om een geldig resultaat aan te geven.

Gebruikte registers: Alle

Looptijd: Ongeveer 15 tot 23 cycli per byte plus 52 cycli vast. De acht extra cycli worden gebruikt op het moment dat het verondersteld grootste element en de index ervan vervangen moeten worden door het huidige element en bijbehorende index. Als die vervanging, gemiddeld, in de helft van het aantal vergelijkingen voorkomt, is de looptijd ongeveer

38 * ARRAY-GROOTTE/2 + 52 cycli.

Als, bijvoorbeeld, ARRAY-GROOTTE = $18 = 24, dan is de looptijd ongeveer

38 * 12 + 52 = 456 + 52 = 508 cycli.

Programmalengte: 45 bytes

Benodigd datageheugen: Een byte ergens in RAM, plus twee bytes op pagina 0. De ene byte ergens in RAM bevat de index van het grootste element (op adres INDEX). De twee bytes op pagina 0 bevatten een pointer naar het array (vanaf adres ARYADR, $00D0_{16}$ in de listing).

Speciale gevallen:

1. Het programma zet de overdrachtvlag op 1 en stopt als de array-grootte 0 is.
2. Als meer dan een element de grootste ongetekende waarde heeft, keert het programma terug met de kleinst mogelijke index. M.a.w., de index geeft de maximum waarde aan die het dichtst bij het beginadres van het array ligt.

Begincondities

(A) = Hoogstwaardige byte van het beginadres van het array
(Y) = Laagstwaardige byte van het beginadres van het array
(X) = Grootte van het array in bytes

Eindcondities

(A) = Grootste ongetekende element

(Y) = Index naar het grootste ongetekende element
Overdrachtvlag = 0 als het resultaat geldig is, 1 als de grootte van het array 0 is en het resultaat zonder betekenis.

Voorbeeld

Data: Grootte van array (in bytes) = (X) = 08

Array-elementen

$35_{16} = 53_{10}$
$A6_{16} = 166_{10}$
$D2_{16} = 210_{10}$
$1B_{16} = 27_{10}$
$44_{16} = 68_{10}$
$59_{16} = 89_{10}$
$7A_{16} = 122_{10}$
$CF_{16} = 207_{10}$

Resultaat: Het grootste ongetekend element is element nr. 2 ($D2_{16} = 210_{10}$).
(A) = grootste element ($D2_{16}$)
(Y) = index naar grootste element (02)
Overdrachtvlag = 0, aangevende dat de array-grootte ongelijk nul en het resultaat geldig is.

```
;       Titel      : Vind het grootste element in een array met bytes zonder
;                    teken
;       Naam       : MAXELM
;
;
;       Doel       : Vind het grootste element, als het beginadres van een
;                    array
;                    en de lengte van dat array gegeven zijn
;
;
;       Begin      : Register A = hoogstwaardige byte van het beginadres
;                    Register Y = laagstwaardige byte van het beginadres
;                    Register X = lengte van het array in bytes
;
;       Eind       : Als de lengte van het array niet gelijk aan 0 is dan
;                        Overdrachtvlag = 0
;                        Register A = grootste element
;                        Register Y = index naar het element
;                            als het grootste element meerdere keren in het
;                            array voorkomt, dan wijst Y naar de eerste plaats
;                            waar dat element voorkomt anders is de
;                            overdrachtvlag 1
;
;       Registers : Alle
;
;       Tijd       : Ongeveer 15 t/m 23 cycli per byte plus 52 basiscycli
;
;       Grootte    : programma 45  bytes
;                    data        1  byte plus
;                                2  bytes op pagina 0
```

```
ARYADR:         .EQU   0D0H          ; Pointer naar array op
                                     ; pagina 0

MAXELM:
        ; Bewaar het beginadres van het array
                sta     ARYADR+1
                sty     ARYADR

        ; Verlaag het beginadres met 1 om zo de index van
        ; 1 tot SIZE te laten lopen
                tya
                bne     MAX1
                dec     ARYADR+1     ; Leen als het nodig is
MAX1:
                dec     ARYADR       ; Verlaag in ieder geval de
                                     ; laagstwaardige byte

        ; Test of de lengte gelijk aan 0 is en initialiseer de
        ; tijdelijke variabelen
                txa
                beq     EREXIT       ; Keer terug als lengte = 0
                tay                  ; Register Y = lengte en
                                     ; index
                lda     (ARYADR),Y   ; Haal de laatste byte van
                                     ; het array op
                sty     INDEX        ; Bewaar zijn index
                dey
                beq     OKEXIT       ; Keer terug als er slechts 1
                                     ; element is

        ; Loop het array van achter naar voren langs, en vergelijk
        ; de waarden met het huidige maximum dat in Register A staat
MAXLP:
                cmp     (ARYADR),Y
                beq     NEWIDX       ; Verander de index alleen
                                     ; als element := maximum
                bcs     NXTBYT       ; Spring als huidig maximum
                                     ; > ARY[Y]
                                     ; anders ARY[Y] >= maximum
                lda     (ARYADR),Y   ;  verander dus het huidige
                                     ;  maximum
NEWIDX:
                sty     INDEX        ;  Verander de index
NXTBYT:
                dey                  ; Ga naar het volgende
                                     ; element
                bne     MAXLP        ; Ga door totdat het hele
                                     ; array doorlopen is

        ; Keer terug
OKEXIT:
                ldy     INDEX        ; Haal de index naar het
                                     ; grootste element op
                dey                  ; Normaliseer de index tot
                                     ;   (0, SIZE-1)
                clc                  ; Geen fouten gevonden
                rts


EREXIT:
                sec                  ; Fout: geen elementen in
                                     ;  het array
                rts
```

```
; Data
INDEX:          .BLOCK  1               ; Index naar het grootste
                                        ; element

;
;
; Voorbeeld aanroep
;
;

SC0903:
        ; Registerpaar A,Y = beginadres van het array
                lda     AADR+1
                ldy     AADR

        ; Register X = lengte van het array
                ldx     #SZARY
                jsr     MAXELM          ; Vind het grootste element
                brk                     ; Resultaat voor deze test-
                                        ; data is
                                        ;  Register A = FFH (max)
                                        ;  Register Y = 06H (index)

                jmp     SC0903


SZARY:          .EQU    10H             ; Lengte van het array
AADR:           .WORD   ARY             ; Beginadres van het array

ARY:
        ; Het array zelf
                .BYTE   8
                .BYTE   7
                .BYTE   6
                .BYTE   5
                .BYTE   4
                .BYTE   3
                .BYTE   2
                .BYTE   1
                .BYTE   0FFH
                .BYTE   0FEH
                .BYTE   0FDH
                .BYTE   0FCH
                .BYTE   0FBH
                .BYTE   0FAH
                .BYTE   0F9H
                .BYTE   0F8H


                .END
```

Zoek element met minimale byte lengte - 9D

Zoekt het kleinste element in een array met ongetekende elementen ter grootte van een byte. De grootte van het array wordt opgegeven en mag niet meer dan 255 bytes bedragen.
Procedure: Het programma stopt met de overdrachtvlag op 1 gezet als de array-grootte nul is. Als de grootte ongelijk nul is, veronderstelt het programma dat de laatste byte van het array de kleinste is en werkt het array dan verder van achteren naar voren door, waarbij het verondersteld kleinste element met het huidige element wordt vergeleken. De kleinste waarde en de index ervan worden bewaard. Uiteindelijk wist het programma de overdrachtvlag om een geldig resultaat aan te geven.

Gebruikte registers: Alle

Looptijd: Ongeveer 15 tot 23 cycli per byte, plus 52 cycli vast. De acht extra cycli worden gebruikt op het moment dat het verondersteld kleinste element en de index ervan vervangen moeten worden door het huidige element en bijbehorende index. Als die vervanging, gemiddeld, in de helft van het aantal vergelijkingen voorkomt, is de looptijd ongeveer

38 * ARRAY-GROOTTE/2 + 52 cycli.

Als, bij voorbeeld, ARRAY-GROOTTE = 14_{16} = 20_{10}, dan is de looptijd ongeveer

38 * 10 + 52 = 380 + 52 = 432 cycli

Programmalengte: 45 bytes

Benodigd datageheugen: Een byte ergens in RAM, plus twee bytes op pagina 0. Dat ene byte ergens in RAM bevat de index van het kleinste element (op adres INDEX). De twee bytes op pagina 0 bevatten een pointer naar het array (vanaf adres ARYADR, $00D0_{16}$ in de listing).

Speciale gevallen:

1. Het programma zet de overdrachtvlag op 1 en stopt als de array-grootte 0 is.
2. Als meer dan een element de kleinste ongetekende waarde heeft, keert het programma terug met de kleinst mogelijke index. M.a.w., de index geeft de minimum waarde aan die het dichtst bij het beginadres van het array ligt.

Begincondities

(A) = Hoogstwaardige byte van het beginadres van het array
(Y) = Laagstwaardige byte van het beginadres van het array
(X) = Grootte van het array in bytes

Eindcondities

(A) = Kleinste ongetekende element

(Y) = Index naar het kleinste ongetekende element
Overdrachtvlag= 0 als het resultaat geldig is, 1 als de grootte van het array 0 is en het resultaat zonder betekenis.

Voorbeeld

Data: Grootte van array (in bytes) = (X) = 08
Array-elementen
$35_{16} = 53_{10}$
$A6_{16} = 166_{10}$
$D2_{16} = 210_{10}$
$1B_{16} = 27_{10}$
$44_{16} = 68_{10}$
$59_{16} = 89_{10}$
$7A_{16} = 122_{10}$
$CF_{16} = 207_{10}$

Resultaat: Het kleinste ongetekend element is element nr. 3 ($1B_{16} = 27_{10}$).
(A) = kleinste element ($1B_{16}$)
(Y) = index naar kleinste element (03)
Overdrachtvlag = 0, aangevende dat array-grootte is ongelijk nul en het resultaat geldig is.

```
;       Titel      : Vind het kleinste element in een array met bytes zonder
;                    teken
;       Naam       : MINELM
;
;
;       Doel       : Vind het kleinste element, als het beginadres van een
;                    array
;                    en de lengte van dat array gegeven zijn
;
;
;       Begin      : Register A = hoogstwaardige byte van het beginadres
;                    Register Y = laagstwaardige byte van het beginadres
;                    Register X = lengte van het array in bytes
;
;       Eind       : Als de lengte van het array niet gelijk aan 0 is dan
;                         Overdrachtvlag = 0
;                         Register A = kleinste element
;                         Register Y = index naar het element
;                            als het kleinste element meerdere keren in het
                             array voorkomt, dan wijst Y naar de eerste plaats
                             waar dat element voorkomt anders is de over-
                             drachtvlag 1
;
;       Registers : Alle
;
;       Tijd       : Ongeveer 15 t/m 23 cycli per byte plus 52 basiscycli
;
;       Grootte    : programma 45  bytes
;                    data        1  byte plus
;                                2  bytes op pagina 0
```

```
ARYADR:         .EQU   0D0H            ; Pointer naar array op
                                       ; pagina 0

MINELM:
        ; Bewaar het beginadres van het array
                sta    ARYADR+1
                sty    ARYADR

        ; Verlaag het beginadres met 1 om zo de index van
        ; 1 tot SIZE te laten lopen
                tya
                bne    MIN1
                dec    ARYADR+1        ; Leen als het nodig is
MIN1:
                dec    ARYADR          ; Verlaag in ieder geval de
                                       ; laagstwaardige byte

        ; Test of de lengte gelijk aan 0 is en initialiseer de
        ; tijdelijke variabelen
                txa
                beq    EREXIT          ; Keer terug als lengte = 0
                tay                    ; Register Y = lengte en
                                       ; index
                lda    (ARYADR),Y      ; Haal de laatste byte van
                                       ; het array op
                sty    INDEX           ; Bewaar zijn index
                dey
                beq    OKEXIT          ; Keer terug als er slechts 1
                                       ; element is

        ; Loop het array van achter naar voren langs, en vergelijk
        ; de waarden met  het huidige minimum dat in register A
        ; staat
MINLP:
                cmp    (ARYADR),Y
                beq    NEWIDX          ; Verander de index alleen
                                       ; als element = minimum
                bcc    NXTBYT          ; Spring als huidig minimum
                                       ; < ARY[Y]
                                       ; anders ARY[Y] <= minimum
                lda    (ARYADR),Y      ;  verander dus het huidige
                                       ;  minimum
NEWIDX:
                sty    INDEX           ;  Verander de index
NXTBYT:
                dey                    ; Ga naar het volgende
                                       ; element
                bne    MINLP           ; Ga door totdat het hele
                                       ; array doorlopen is


OKEXIT:
                ldy    INDEX           ; Haal de index naar het
                                       ; kleinste element op
                dey                    ; Normaliseer de index tot
                                       ;   (0, SIZE-1)
                clc                    ; Geen fouten gevonden
                rts


EREXIT:
                sec                    ; Fout: geen elementen in
                                       ;  het array
                rts
```

```
; Data
INDEX:          .BLOCK  1               ; Index naar het kleinste
                                        ; element


;
;
; Voorbeeld uitvoering
;
;

SC0904:
        ; Registerpaar A,Y = beginadres van het array
                lda     AADR+1
                ldy     AADR

        ; Register X = lengte van het array
                ldx     #SZARY
                jsr     MINELM          ; Vindt het kleinste element
                brk                     ; Resultaat voor deze test
                                        ; data is
                                        ;  Register A = 01H (min)
                                        ;  Register Y = 07H (index)

                jmp     SC0904


SZARY:          .EQU    10H             ; Lengte van het array
AADR:           .WORD   ARY             ; Beginadres van het array

ARY:
        ; Het array zelf
                .BYTE   8
                .BYTE   7
                .BYTE   6
                .BYTE   5
                .BYTE   4
                .BYTE   3
                .BYTE   2
                .BYTE   1
                .BYTE   0FFH
                .BYTE   0FEH
                .BYTE   0FDH
                .BYTE   0FCH
                .BYTE   0FBH
                .BYTE   0FAH
                .BYTE   0F9H
                .BYTE   0F8H


                .END
```

Binair zoeken - 9E

Doorzoekt een array met ongetekende elementen ter grootte van een byte naar een bepaalde waarde. Verondersteld wordt dat het array met het kleinste element op het begin (laagste) adres is gerangschikt. Geeft de index naar de waarde en de overdrachtvlag gewist terug als de waarde is gevonden; geeft een gezette overdrachtvlag (op 1) als de waarde niet is gevonden. De grootte van het array wordt opgegeven en mag niet meer dan 255 bytes bedragen. Bij deze binaire zoekmethode wordt de waarde vergeleken met het middelste element van het resterende deel van het element; als de twee niet gelijk zijn, wordt dat deel dat de waarde in elk geval niet kan bevatten (door de ordening) verder genegeerd. Dit proces wordt daarna herhaald.
Procedure: Het programma houdt boven- en ondergrenzen bij die het deel van het array aangeven dat nog steeds doorzocht wordt. Bij elke nieuwe poging is de index het gemiddelde van de boven- en ondergrens. Het programma vergelijkt de waarde en het element van de proefindex; als de twee niet gelijk zijn, verwerpt het programma dat deel van het array dat het element niet meer kan bevatten. Ofte wel, als de waarde groter is dan het element van de proefindex, dan wordt het deel onder de proefindex weggeworpen. Het programma stopt als het een overeenstemming vindt of als er geen elementen meer zijn die doorzocht kunnen worden (ofte wel als het deel van het array dat wordt onderzocht niets meer bevat). Het programma zet de overdracht vlag op 1 als de waarde wordt gevonden en op 0 als dit niet het geval is.
In het geval van voorbeeld 1 zoals dat verderop te zien is (de waarde is $0D_{16}$), werkt deze procedure als volgt:
Bij de eerste poging is de ondergrens nul en de bovengrens de lengte van het array minus 1 (aangezien we vanaf nul indexeren). Dus

ONDERGRENS = 0
BOVENGRENS = LENGTE - 1 = $0F_{16} = 15_{10}$
GOK = (BOVENGRENS + ONDERGRENS)/2 = 07 (het resultaat wordt naar beneden afgerond)
ARRAY (GOK) = ARRAY (7) = $10_{16} = 16_{10}$

Aangezien onze waarde ($0D_{16}$) kleiner is dan ARRAY (7), heeft het geen zin om te kijken naar de elementen met een index van 7 of meer. Dus hebben we

ONDERGRENS = 0
BOVENGRENS = GOK - 1 = 06
GOK = (BOVENGRENS + ONDERGRENS)/2 = 03
ARRAY (GOK) = ARRAY (3) = 07

Aangezien onze waarde ($0D_{16}$) groter is dan ARRAY (3), heeft het geen zin te kijken naar de elementen met een index van 3 of minder. Dus hebben we

ONDERGRENS = GOK + 1 = 04
BOVENGRENS = 06
GOK = (BOVENGRENS + ONDERGRENS)/2 = 05
ARRAY (GOK) = ARRAY (5) = 09

Aangezien onze waarde ($0D_{16}$) groter is dan ARRAY (5), heeft het geen zin te kijken naar de elementen met een index van 5 of minder. Dus hebben we

ONDERGRENS = GOK + 1 = 06
BOVENGRENS = 06

GOK = (BOVENGRENS + ONDERGRENS)/2 = 06
ARRAY (GOK) = ARRAY (6) = $0D_{16}$

Aangezien onze waarde ($0D_{16}$) gelijk is aan ARRAY (6), hebben we het element gevonden. Als daarentegen onze waarde $0E_{16}$ zou zijn geweest, was de nieuwe ondergrens 07 geworden en waren er geen elementen in het resterende deel van het array meer geweest, om doorzocht te worden.

Gebruikte registers:	Alle
Looptijd:	Ongeveer 52 cycli per herhaling, plus 80 cycli vast. Binair zoeken vereist ongeveer 2log N herhalingen, waarbij N de grootte van het array (aantal elementen) is. Als, bij voorbeeld, N=32, zal het binaire zoeken ongeveer $^{2}\log 32 = 5$ herhalingen nodig hebben. De looptijd zal dan ongeveer zijn 52 * 5 + 80 = 260 + 80 = 340 cycli.
Programmalengte:	89 bytes
Benodigd datageheugen:	Drie bytes ergens in RAM, plus twee bytes op pagina 0. De drie bytes ergens in RAM bevatten de gezochte waarde (een byte op adres VALUE), de ondergrens van het doorzochte gebied (een byte op adres LBND) en de bovengrens van het doorzochte gebied (een byte op adres UBND). De twee bytes op pagina 0 bevatten een pointer naar het array (vanaf adres ARYADR, $00D0_{16}$ in de listing).
Speciaal geval	Het programma zet de overdrachtvlag op 1 en stopt bij een grootte of lengte van nul. M.a.w., de lengte wordt verondersteld nul te zijn en de waarde zal zeker niet gevonden kunnen worden.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Te vinden waarde
Grootte van de array (in bytes)
Laagstwaardige byte van het beginadres van het array (adres van het kleinste ongetekende element)
Hoogstwaardige byte van het beginadres van het array (adres van het kleinste ongetekende element)

Eindcondities

Overdracht = 0 als de waarde is gevonden, overdracht = 1 als dit niet het geval is. Als de waarde gevonden is, bevat (A) de index naar de waarde in het element.

Voorbeelden

Lengte van het array = 10_{16} = 16_{10}
Elementen van het array zijn 01_{16}, 02_{16}, 05_{16}, 07_{16}, 09_{16}, 09_{16}, $0D_{16}$, 10_{16}, $2E_{16}$, 37_{16}, $5D_{16}$, $7E_{16}$, $A1_{16}$, $B4_{16}$, $D7_{16}$, $E0_{16}$

1. Data: Te vinden waarde = $0D_{16}$

   Resultaat: Overdrachtvlag = 0, waarde is gevonden
   (A) = 06, de index van de waarde in het element.

2. Data: Te vinden waarde = $9B_{16}$

   Resultaat: Overdrachtvlag = 1, waarde niet gevonden.

```
;       Titel      : Binair zoeken
;       Naam       : BINSH
;
;
;       Doel       : Zoek in een geordend array van bytes zonder teken naar
;                    een bepaalde waarde
;                    Het array is maximaal 255 elementen groot
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van terugkeeradres
;                      Hoogstwaardige byte van terugkeeradres
;                      De gezochte waarde
;                      De lengte van het array in bytes
;                      Laagstwaardige byte van het begin adres van het array
;                      Hoogstwaardige byte van het begin adres van het array
;
;       Eind       : Als de waarde gevonden is dan
;                        Overdrachtvlag = 0
;                        Register A = index in het array naar de waarde
;                    anders
;                        Overdrachtvlag = 1
;
;       Registers  : Alle
;
;       Tijd       : Ongeveer 52 cycli voor iedere keer dat de zoek-lus wordt
;                    doorlopen plus 80 basis cycli
;                    Voor het vinden van een waarde is bij binair zoeken
;                    gemiddeld 2 log N slagen nodig, waarbij N het aantal
;                    elementen in het array is
;
;       Grootte    : programma 89  bytes
;                    data        3  bytes plus
;                                2  bytes op pagina 0

ARYADR:              .EQU   0D0H            ; Pointer naar array op
                                            ; pagina 0

BINSH:
```

```
        ; Haal het terugkeeradres van de stack af
                pla
                tay
                pla
                tax

        ; Haal de waarde die gezocht wordt van de stack af
                pla
                sta     VALUE

        ; Haal de lengte van het array van de stack af
                pla
                sta     UBND

        ;Haal het beginadres van het array van de stack af
                pla
                sta     ARYADR
                pla
                sta     ARYADR+1

        ; Zet het terugkeeradres terug op de stack
                txa
                pha
                tya
                pha

        ; Test of de lengte ongelijk aan 0 is
                ldx     UBND
                beq     NOTFND          ; Spring als de lengte 0 is

        ; Zet de boven- en de benedengrens van het te doorzoeken
        ; deel van het array goed
                dex
                stx     UBND            ; Bovengrens = lengte - 1
                lda     #0
                sta     LBND            ; Ondergrens = 0

        ; Zoek-lus
        ;  De volgende index zit halverwege de boven- en de beneden-
        ;  grens. Bereken deze index
NXTBYT:
                lda     UBND
                clc
                adc     LBND            ; Tel de boven- en de beneden-
                                        ; grens bij elkaar op
                ror     a               ; Deel door 2 (rond naar
                                        ; beneden af)
                tay                     ; Y is de nieuwe index

        ; Als de index groter dan de bovengrens is, zit het
        ; element niet in het array
                cpy     UBND
                beq     TSTLB           ; Spring als
                                        ;     index = bovengrens
                bcs     NOTFND          ; Spring als
                                        ;     index > bovengrens

        ; Als de index kleiner is dan de ondergrens, zit het
        ; element niet in het array
TSTLB:
                cpy     LBND
                bcc     NOTFND          ; Spring als
                                        ;     index < ondergrens
```

```
            ; Test of we het element gevonden hebben
                  lda     VALUE
                  cmp     (ARYADR),Y
                  bcc     SMALL           ; Spring als
                                          ;      element < ARYADR[Y]
                  beq     FND             ; Spring als we het element
                                          ; gevonden hebben

            ; De waarde is groter dan ARYADR[Y], dus maak de onder-
            ; grens gelijk aan Y + 1 (de waarde kan alleen verderop
            ; leggen)
                  iny
                  sty     LBND
                  bne     NXTBYT          ; Ga door als er geen over-
                                          ; loop is opgetreden

                  beq     NOTFND          ; Overloop betekent dat de
                                          ; ondergrens te groot is
                                          ; geworden

            ; De waarde is kleiner dan ARYADR[Y], maak dus de boven-
            ; grens gelijk aan Y - 1 (de waarde kan alleen maar verder
            ; naar beneden liggen)
SMALL:
                  dey
                  sty     UBND
                  cpy     #0FFH
                  bne     NXTBYT          ; Ga door als er geen onder-
                                          ; loop is opgetreden

                  beq     NOTFND          ; Anders, keer terug


FND:
            ; We hebben de waarde gevonden
                  clc                     ; Geef aan dat de waarde
                                          ; gevonden is
                  tya                     ; Register A = index
                  rts


NOTFND:
            ; De waarde is niet gevonden
                  sec                     ; Geef aan dat de waarde niet
                                          ; gevonden is
                  rts


; Data
VALUE:            .BLOCK  1               ; De te vinden waarde
LBND:             .BLOCK  1               ; De bovengrens
UBND:             .BLOCK  1               ; De ondergrens

;
;
; Voorbeeld uitvoering
;
;

SC0905:
            ; Zoek naar een waarde die in het array zit
```

```
        ; Zet het begin adres van het array op de stack
                lda     BFADR+1
                pha
                lda     BFADR
                pha

        ; Zet de lengte van het array op de stack
                lda     BFSZ
                pha

        ; Zet de te zoeken waarde op de stack
                lda     #7
                pha
                jsr     BINSH           ; Zoek
                brk                     ; Resultaat van het zoeken
                                        ; naar 07H in het array
                                        ; is
                                        ; Register A = 4
                                        ; Overdrachtvlag = 0


        ; Zoek naar een waarde die niet in het array zit

        ; Zet het begin adres van het array op de stack
                lda     BFADR+1
                pha
                lda     BFADR
                pha

        ; Zet de lengte van het array op de stack
                lda     BFSZ
                pha

        ; Zet de te zoeken waarde op de stack
                lda     #0
                pha
                jsr     BINSH           ; Zoek
                brk                     ; Resultaat van het zoeken
                                        ; naar 00H is het array is
                                        ;
                                        ; Overdrachtvlag = 1

                jmp     SC0905


; Data
SIZE:           .EQU    010H            ; Lengte van de buffer
BFADR:          .WORD   BF              ; Beginadres van de buffer
BFSZ:           .BYTE   SIZE            ; Lengte van de buffer
BF:
        ; De buffer
                .BYTE   1
                .BYTE   2
                .BYTE   4
                .BYTE   5
                .BYTE   7
                .BYTE   9
                .BYTE   10
                .BYTE   11
                .BYTE   23
                .BYTE   50
```

```
.BYTE   81
.BYTE   123
.BYTE   191
.BYTE   199
.BYTE   250
.BYTE   255

.END
```

Bubble Sort - 9F

Sorteert een array van ongetekende elementen ter grootte van een byte in oplopende volgorde door gebruik te maken van een bubble-sort algoritme. Een uitvoering van dit algoritme verplaatst het grootste overblijvende element naar boven door het met alle andere elementen te vergelijken en, waar nodig, onderweg omwisselingen uit te voeren. Het algoritme gaat door tot alle elementen doorlopen zijn of tot het een iteratie heeft uitgevoerd en er geen omwisselingen nodig waren. De grootte van het element wordt opgegeven en mag niet meer dan 255 bytes bedragen.
Procedure: Het programma bekijkt eerst het hele array. Het onderzoekt paren van elementen en wisselt ze onderling als ze niet in de goede volgorde staan. Hierbij wordt een vlag gezet om aan te geven dat er een omwisseling heeft plaats gevonden. Aan het einde van een iteratie controleert het programma de omwisselingsvlag om te zien of het array al op volgorde is gebracht. Zo niet, dan wordt de handeling herhaald, nadat het aantal te onderzoeken elementen met 1 is verlaagd. Dit omdat het grootste resterende element naar de top is geborreld. Het programma stopt onmiddellijk als de lengte van het array kleiner dan twee is, omdat er dan geen ordening nodig is. Let op dat het aantal paren altijd een kleiner is dan het aantal te onderzoeken elementen, omdat het laatste element geen element boven zich heeft.

Gebruikte registers: Alle

Looptijd: Ongeveer

$$34 * N * N + 25 * N + 70$$

cycli, waarbij N de grootte (lengte) van het array in bytes is. Als, bijvoorbeeld, $N = 20_{16}$ (32_{10}), dan is de looptijd ongeveer

$$34 * 32 * 32 + 25 * 32 + 70 = 34 * 1024 + 870 = 34.816 + 870 = 35.686$$ cycli.

Programmalengte: 79 bytes

Benodigd datageheugen: Twee bytes ergens in RAM, plus vier bytes op pagina 0. De twee bytes ergens in RAM bevatten de lengte van het array (een byte op adres LEN) en de omwisselingsvlag (een byte op adres XCHGFG). De vier bytes op pagina 0 bevatten pointers naar het eerste en tweede element van het array (twee bytes vanaf adres A1ADR, $00D0_{16}$ in de listing en twee bytes vanaf adres A2ADR, $00D2_{16}$ in de listing).

Speciaal geval: Het programma stopt zonder te sorteren bij een lengte (of grootte) van 00 of 01.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Lengte (grootte) van het array in bytes
Laagstwaardige byte van het beginadres van het array
Hoogstwaardige byte van het beginadres van het array

Eindcondities

Array in oplopende volgorde gesorteerd, aangenomen dat elementen ongetekende bytes waren. Zodoende staat de kleinste ongetekende byte nu op het beginadres.

Voorbeeld

Data: Lengte (grootte) van array = 06
Elementen = 35_{16}, $6A_{16}$, $2B_{16}$, $3E_{16}$, $D4_{16}$, $4F_{16}$

Resultaat: Na de eerste iteratie hebben we

35_{16}, $2B_{16}$, $3E_{16}$, $6A_{16}$, $4F_{16}$, $D4_{16}$

Het grootste element staat nu aan het einde van het array en hoeft niet meer bekeken te worden

Na de tweede iteratie hebben we

$2B_{16}$, 35_{16}, $3E_{16}$, $4F_{16}$, $6A_{16}$, $D4_{16}$

Het op een na grootste element bevindt zich nu in de juiste positie en hoeft niet meer bekeken te worden.

De derde iteratie laat het array ongewijzigd, omdat de elementen al in oplopende volgorde staan.

```
;       Titel     : Bubble-sort
;       Naam      : BUBSRT
;
;
;       Doel      : Sorteer een array bytes zonder teken in volgorde van
;                   grootte met behulp van het bubble-sort algoritme
;                   Een array mag maximaal 255 bytes lang zijn
;
;
;       Begin     : Top van de stack
;                     Laagstwaardige byte van terugkeeradres
;                     Hoogstwaardige byte van terugkeeradres
;                     Lengte van het array
;                     Laagstwaardige byte van het beginadres van het array
;                     Hoogstwaardige byte van het beginadres van het array
;
;       Eind      : Het array is gesorteerd in oplopende volgorde
;
;       Registers : Alle
;
;       Tijd      : Ongeveer (34 * N * N) + (25 * N) cycli plus 70 basis
;                   cycli, waarbij N de lengte van het array representeerd
;
;       Grootte   : programma 79  bytes
```

```
;                 data        2  bytes plus
;                             4  bytes op pagina 0

A1ADR:              .EQU   0D0H            ; Adres van eerste element
A2ADR:              .EQU   0D2H            ; Adres van tweede element

BUBSRT:
            ; Haal het terugkeeradres van de stack af
                  pla
                  tay
                  pla
                  tax

            ; Haal de lengte van het array van de stack af
                  pla
                  sta     LEN

            ; Haal het beginadres van het array van de stack af
                  pla
                  sta     A1ADR          ; Laagstwaardige byte
                  clc
                  adc     #1
                  sta     A2ADR          ; Tweede element ligt 1
                                         ; positie verder in het array

                  pla
                  sta     A1ADR+1        ; Hoogstwaardige byte
                  adc     #0
                  sta     A2ADR+1        ; Zet de hoogstwaardige
                                         ; byte van de pointer naar
                                         ; het tweede element goed

            ; Zet het terugkeeradres terug op de stack
                  txa
                  pha
                  tya
                  pha

            ; Test of de lengte van het array groter dan 1 is
                  lda     LEN
                  cmp     #2
                  bcc     DONE           ; Spring als de lengte
                                         ; kleiner dan 2 is

                  dec     LEN            ; Verlaag de lengte met 1
                                         ; want het laatste element
                                         ; heeft geen opvolger

            ; Begin te sorteren
SRTLP:
                  ldx     LEN            ; X = Teller van de binnenste
                                         ;     lus
                  ldy     #0             ; Y = Begin-index
                  sty     XCHGFG         ; Zet de verwisselvlag op 0
INLOOP:
                  lda     (A2ADR),Y
                  cmp     (A1ADR),Y      ; Vergelijk de twee elementen
                  bcs     AFTSWP         ; Spring als tweede >= eerste
                                         ; element
                                         ; Tweede < eerste element
                  pha                    ;  verwissel dus de elementen
                  lda     (A1ADR),Y      ; Zet het tweede element
```

```
                sta     (A2ADR),Y       ; op de plaats van het eerste
                                        ; element
                pla
                sta     (A1ADR),Y       ; Zet het eerste element op
                                        ; de plaats van het tweede
                                        ; element
                lda     #1
                sta     XCHGFG          ; Er zijn elementen ver-
                                        ; wisseld, zet dus de vlag


AFTSWP:
                iny                     ; Ga naar het volgende
                                        ; elementenpaar wijzen
                dex                     ; Verlaag de teller van de
                                        ; binnenste lus
                bne     INLOOP          ; Ga door zolang de binnenste
                                        ; lus nog niet ten eind is

        ; De binnenste lus is beeindigd , als er niets is ver-
        ; wisseld zijn we klaar
                lda     XCHGFG
                beq     DONE            ; We zijn klaar
                dec     LEN             ; Verlaag de teller van de
                                        ; buitenste lus
                bne     SRTLP           ; Ga door tot ook de
                                        ; buitenste lus klaar is


DONE:
                rts

: Data
LEN:            .BLOCK  1               ; Lengte van het array
XCHGFG:         .BLOCK  1               ; De verwisselvlag
                                        ;   1 = Er zijn elementen
                                        ;       verwisseld
                                        ;   0 = Er zijn geen
                                        ;       elementen verwisseld

;
;
; Voorbeeld uitvoering
;
;

SC0906:
        ; Zet het beginadres van het array op de stack
                lda     BFADR+1
                pha
                lda     BFADR
                pha

        ; Zet de lengte van het array op de stack
                lda     BFSZ
                pha
                jsr     BUBSRT          ; Sorteer het array
                brk                     ; Resultaat is
                                        ; 0,1,2,3, .. ,14,15

                jmp     SC0906
```

```
; Data
SIZE:           .EQU    010H        ; Lengte van de buffer
BFADR:          .WORD   BF          ; Beginadres van de buffer
BFSZ:           .BYTE   SIZE        ; Lengte van de buffer
BF:
        ; De buffer zelf
                .BYTE   15
                .BYTE   14
                .BYTE   13
                .BYTE   12
                .BYTE   11
                .BYTE   10
                .BYTE   9
                .BYTE   8
                .BYTE   7
                .BYTE   6
                .BYTE   5
                .BYTE   4
                .BYTE   3
                .BYTE   2
                .BYTE   1
                .BYTE   0

                .END
```

RAM-test - 9G

Voert een test uit op een gebied in RAM, aangegeven met een beginadres en een lengte in bytes. Schrijft de waarden 00, FF_{16}, AA_{16} (10101010_2) en 55_{16} (01010101_2) in elke byte en controleert of ze goed teruggelezen kunnen worden. Plaatst een enkelvoudig 1-bit in iedere positie van elke byte en kijkt of ook dat goed teruggelezen kan worden. Wist de overdrachtvlag als alle tests uitgevoerd kunnen worden. Als het programma een fout vindt, zet het de overdrachtvlag, geeft het adres terug waar de fout zich voordeed en de in de test gebruikte waarde, en stopt.
Procedure: Het programma voert de enkele-waardetests (met 00, FF_{16}, AA_{16} en 55_{16}) uit door het geheugengebied eerst te vullen en dan elke byte met de opgegeven waarde te controleren. Het vullen van het hele gebied zou voor voldoende vertraging tussen schrijven en lezen moeten zorgen om een fout te kunnen vinden in het vasthouden van de data (mogelijk veroorzaakt door een onjuist ontworpen refresh-circuit). Het programma voert dan een lopende bittest uit, beginnende bij bit 7. Hierbij schrijft het de data in het geheugen en probeert ze onmiddellijk terug te lezen om het te vergelijken. In alle tests werkt het programma eerst hele pagina's af om dan resterende delen van pagina's te testen. Zodoende kan het programma volstaan met 8-bits in plaats van 16-bits tellers. Deze benaderingswijze verkleint de looptijd, maar vergroot het geheugengebruik vergeleken met de methode om het hele geheugengebied in een lus te doorwerken. Merk op dat het programma, als er een fout wordt gevonden, de overdrachtvlag op 1 zet, het adres van de fout en de in de test gebruikte waarde teruggeeft, en stopt. Als alle tests zonder foutmeldingen verlopen, wordt de overdrachtvlag gewist voordat het programma stopt.

Gebruikte registers: Alle

Looptijd: Ongeveer 245 cycli per geteste byte, plus 650 cycli vast. Het testen, bij voorbeeld, van een geheugengebied van 400_{16} = 1024 bytes zou dus ongeveer

245 * 1024 + 650 = 250.880 + 650 = 251.530

cycli duren.

Programmalengte: 229 bytes

Benodigd datageheugen: Zes bytes ergens in RAM, plus twee bytes op pagina 0. De zes bytes ergens in RAM bevatten het adres van het eerste element (twee bytes vanaf adres ADDR), de lengte van het te testen gebied (twee bytes vanaf adres LEN) en de tijdelijke lengte (twee bytes vanaf adres TLEN). De twee bytes op pagina 0 bevatten een pointer naar het getest gebied (vanaf adres TADDR, $00D0_{16}$ in de listing).

Speciale gevallen:

1. Het programma stopt met de overdrachtvlag gewist (geen fouten) als de lengte van het getestte gebied 0000 is.
2. Aangezien de routine alle bytes in het geteste gebied verandert, zal het testen van een gebied waar de routine of de tijdelijke opslag zich bevindt, onvoorspelbare resultaten hebben.
   Dit betekent, dat niet het hele geheugen door deze routine getest kan worden, omdat dat zou betekenen dat de routine zijn eigen tijdelijke opslaggeheugen zou moeten testen.

3. Een poging ROM-gebied te testen leidt tot een foutmelding zodra het programma probeert een ROM-locatie met een andere waarde te laden dan erin stond.

Begincondities

Volgorde op stack (van boven naar beneden)

Laagstwaardige byte van het terugkeeradres
Hoogstwaardige byte van het terugkeeradres
Laagstwaardige byte van de grootte (lengte) van het gebied in bytes
Hoogstwaardige byte van de grootte (lengte) van het gebied in bytes
Laagstwaardige byte van het beginadres van het te testen gebied
Hoogstwaardige byte van het beginadres van het te testen gebied

Eindcondities

1. Als er een fout is gevonden:

   Overdrachtvlag = 1
   (A) = Hoogstwaardige byte van het adres van de gevonden fout
   (Y) = Laagstwaardige byte van het adres van de gevonden fout
   (X) = Verwachte waarde (waarde die in de test werd gebruikt)

2. Als er geen fout is gevonden:

   Overdrachtvlag = 0
   Alle bytes in het testgebied bevatten 00

Voorbeeld

Data: Beginadres = 0380_{16}
Lengte (grootte) van gebied = 0200_{16}

Resultaat: Getest gebied vormen de 0200_{16} bytes vanaf adres 0380_{16}, dus de adressen 0380_{16} tot en met $057F_{16}$. De volgorde van de tests is:

1. Schrijf en lees 00
2. Schrijf en lees FF_{16}
3. Schrijf en lees AA_{16} (10101010_2)
4. Schrijf en lees 55_{16} (01010101_2)
5. Lopende bittest, beginnende bij bit 7 en schuivend naar rechts. Dus de test begint bij 80_{16} (10000000_2) en verplaatst het 1-bit een positie naar rechts bij elke volgende test van een enkele byte.

```
;       Titel      : Ram test
;       Naam       : RAMTST
;
;
;       Doel       : Test het RAM geheugen
;                      1)    Schrijf overal 00 hexadecimaal in en test
;                      2)    Schrijf overal FF hexadecimaal in en test
;                      3)    Schrijf overal AA hexadecimaal in en test
;                      4)    Schrijf overal 55 hexadecimaal in en test
;                      5)    Schuif een bit door alle geheugen posities
;
;                    Zodra het programma een fout vindt, stopt het
;                    ogenblikkelijk met de overdrachtvlag gezet en geeft aan
;                    waar de fout is aangetroffen en welke waarde in de test
;                    werd gebruikt
;
;
;       Begin      : Top van de stack
;                      Laagstwaardige byte van het terugkeeradres
;                      Hoogstwaardige byte van het terugkeeradres
;                      Laagstwaardige byte van de lengte in bytes
;                      Hoogstwaardige byte van de lengte in bytes
;                      Laagstwaardige byte van start adres van het testgebied
;                      Hoogstwaardige byte van start adres van het testgebied
;
;       Eind       : Als er geen fouten zijn gevonden dan
;                        Overdrachtvlag = 0
;                        Alle bytes van het testgebied bevatten 00
;                        hexadecimaal
;                    anders
;                        Overdrachtvlag = 1
;                        Register A = hoogstwaardige byte van het adres waar
;                                     de fout optrad
;                        Register Y = laagstwaardige byte van het adres waar
;                                     de fout optrad
;                        Register X = de waarde die verwacht werd
;
;
;       Registers : Alle
;
;       Tijd       : Ongeveer 245 cycli per byte plus 650 basiscycli
;
;       Grootte    : programma 288  bytes
;                    data        6  bytes plus
;                                2  bytes op pagina 0

TADDR:              .EQU    0D0H            ; Pointer naar het te testen
                                            ; gebied op pagina 0


RAMTST:
            ; Haal het terugkeeradres van de stack af
                    pla
                    tay
                    pla
                    tax

            ; Haal de lengte van het te testen gebied van de stack af
                    pla
                    sta     LEN
                    pla
                    sta     LEN+1
```

```
; Haal het beginadres van het te testen gebied van de
; stack af
        pla
        sta     ADDR
        pla
        sta     ADDR+1

; Zet het terugkeeradres weer terug op de stack
        txa
        pha
        tya
        pha

; Kijk of de lengte van het te testen gebied groter dan 0 is
        lda     LEN
        ora     LEN+1
        beq     EXITOK          ; Als het 0 is hoeven we niet
                                ; te testen

; Vul het te testen gebied met FFH en vergelijk
        lda     #0FFH
        jsr     FILCMP
        bcs     EXITER          ; Spring als een fout
                                ; gevonden is

; Vul het te testen gebied met AAH en vergelijk
;     (AAH is 10101010 binair)
        lda     #0AAH
        jsr     FILCMP
        bcs     EXITER          ; Spring als een fout
                                ; gevonden is

; Vul het te testen gebied met 55H en vergelijk
;     (55H is 01010101 binair)
        lda     #55H
        jsr     FILCMP
        bcs     EXITER          ; Spring als een fout
                                ; gevonden is

; Vul het te testen gebied met 0 en vergelijk
        lda     #0
        jsr     FILCMP
        bcs     EXITER          ; Spring als een fout
                                ; gevonden is

; Schuif een bit door alle bytes van het te testen gebied
; en vergelijk
        jsr     ITEMPS          ; Initialiseer

; Loop door alle pagina's van 256 bytes
        ldx     TLEN+1          ; Test hoeveel hele pagina's
                                ;  doorlopen moeten worden
        beq     WLKPRT          ; Spring als er geen hele
                                ;  pagina's doorlopen moeten

                                ;  worden
        ldy     #0              ; Register Y = index


WLKLP:
        lda     #80H            ; Maak A gelijk aan
                                ;     10000000 binair
```

```
WLKLP1:
                sta     (TADDR),Y       ; Zet het patroon in het
                                        ;  geheugen
                cmp     (TADDR),Y       ; Test of het goed is aange-
                                        ;  komen
                bne     EXITER          ; Zo niet, spring dan terug
                lsr     A               ; Schuif het bit 1 positie op
                bne     WLKLP1          ; Ga door tot we met deze byte
                                        ; klaar zijn
                sta     (TADDR),Y       ; Zet A op dezelfde plaats
                iny                     ; Verhoog de index
                bne     WLKLP           ; Ga door totdat deze pagina
                                        ; klaar is
                inc     TADDR+1         ; Ga naar de volgende pagina
                dex                     ; Verlaag de pagina-teller
                bne     WLPLP           ; Ga door totdat alle hele
                                        ; pagina's getest zijn

        ; Test de laatste gedeeltelijke pagina
WLKPRT:
                ldx     TLEN            ; X = het aantal nog te testen
                                        ;     bytes
                beq     EXITOK          ; Spring als X = 0
                ldy     #0              ; Zet de index op 0

WLKLP2:
                lda     #80H            ; Zet A op 10000000 binair

WLKLP3:
                sta     (TADDR),Y       ; Zet het testpatroon in het
                                        ;  geheugen
                cmp     (TADDR),Y       ; Vergelijk het met de weg-
                                        ; geschreven waarde
                bne     EXITER          ; Spring als ze niet gelijk zijn
                lsr     A               ; Schuif het bit 1 positie op
                bne     WLKLP3          ; Ga door tot we met deze byte
                                        ; klaar zijn
                sta     (TADDR),Y       ; Zet A nogmaals in het geheugen
                iny                     ; Ga naar de volgende byte wijzen
                dex                     ; Verlaag de byte-teller
                bne     WLKLP2          ; Ga door tot we ook de
                                        ; gedeeltelijke pagina getest
                                        ; hebben

EXITOK:
                sec                     ; Geen fouten
                rts

EXITER:
                jsr     ERROR           ; Er is ergens een foute
                                        ; waarde teruggelezen
                rts


; Subroutine : FILCMP
; Doel       : Vul het geheugenblok met een waarde en kijk daarna of de
;              ingevulde waarde er daadwerkelijk ook staat
; Begin      : Register A = waarde
;              ADDR = beginadres van het geheugenblok
;              LEN = lengte van het geheugenblok
```

```
; Einde       : Als er geen fouten zijn gevonden
;                 Overdrachtvlag = 0
;               anders
;                 Overdrachtvlag = 1
;                 Register A = hoogstwaardige byte van het adres waar het
;                              fout ging
;                 Register Y = laagstwaardige byte van het adres waar het
;                              fout ging
;                 Register X = Verwachte waarde
; Registers   : Alle
;

FILCMP:
                  jsr     ITEMPS          ; Initialiseer

            ; Vul het te testen geheugen blok met de waarde in A
            ; Vul eerst de hele pagina's
                  ldx     TLEN+1
                  beq     FILPRT
                  ldy     #0              ; Begin met index 0

FILLP:
                  sta     (TADDR),Y       ; Zet de waarde in het
                                          ; geheugen
                  iny                     ; Verhoog de index
                  bne     FILLP           ; Ga door tot we deze pagina
                                          ; verwerkt hebben
                  inc     TADDR+1         ; Ga door met de volgende
                                          ; pagina
                  dex
                  bne     FILLP           ; Ga door totdat alle hele
                                          ; pagina's gedaan zijn

            ; Vul het resterende deel van de laatste pagina
FILPRT:
                  ldx     TLEN
                  ldy     #0              ; Index = 0

FILLP1:
                  sta     (TADDR),Y       ; Zet de waarde in het
                                          ; geheugen
                  iny
                  dex
                  bne     FILLP1          ; Ga door tot alle bytes die
                                          ; op deze pagina getest
                                          ; moesten worden, gevuld
                                          ; zijn

            ; Vergelijk nu het blok geheugen weer met de waarde in A
CMPARE:
                  jsr     ITEMPS          ; Initialiseer

                  ldx     TLEN+1
                  beq     CMPRT
                  ldy     #0              ; Index = 0


CMPLP:
                  cmp     (TADDR),Y       ; Kan de bewaarde waarde
                                          ; teruggelezen worden?
                  bne     CMPER           ; Spring als dit niet zo is
                  iny
```

```
                bne     CMPLP           ; Ga door tot we deze pagina
                                        ; getest hebben
                inc     TADDR+1         ; Ga door met de volgende
                                        ; pagina
                dex                     ; Verlaag de pagina-teller
                bne     CMPLP           ; Ga door tot we alle hele
                                        ; pagina's gedaan hebben


CMPRT:
          ; Test nu nog even de laatste deelpagina
                ldx     TLEN
                ldy     #0


CMPLP1:
                cmp     (TADDR),Y       ; Vergelijk
                bne     CMPER           ; Spring als dit fout ging
                iny
                dex
                bne     CMPLP1          ; Ga door tot alle bytes
                                        ; getest zijn


CMPOK:
                clc                     ; Overdrachtvlag = 0
                                        ; Geen fouten
                rts


CMPER:
                jsr ERROR               ; Er is iets fout gegaan
                rts


;
; Subroutine : ITEMPS
; Doel       : Initialiseer de tijdelijke variabelen
; Begin      : ADDR = adres van het geheugenblok
;              LEN = aantal bytes in het geheugenblok
; Einde      : TADDR = ADDR
;              TLEN  = LEN
; Registers  : Y, P
;

ITEMPS:
                ldy     ADDR
                sty     TADDR
                ldy     ADDR+1
                sty     TADDR+1

                ldy     LEN
                sty     TLEN
                ldy     LEN+1
                sty     TLEN+1
                rts
;
; Subroutine : ERROR
; Doel       : Zet de registers goed voor de fout-indicatie
; Begin      : Register A = verwachte byte
;              TADDR      = basisadres
;              Register Y = index in het geheugenblok
```

```
; Einde       : Register X = verwachte byte
;               Register A = hoogstwaardige byte van het adres waar de
;                            fout ontdekt werd
;               Register Y = laagstwaardige byte van het adres waar de
;                            fout ontdekt werd
; Registers   : Allemaal
;

ERROR:
                    tax                     ; Register X = verwachte byte
                    tya                     ; Haal de index op
                    clc                     ; Maak de overdracht schoon
                    adc     TADDR           ; Bereken het adres waar de
                    tay                     ; fout is opgetreden
                    lda     TADDR+1
                    adc     #0
                    sec                     ; Overdrachtvlag = 1
                                            ; Er is iets fout gegaan
                    rts


; Data
ADDR:               .BLOCK  2               ; Adres van het eerste element
LEN:                .BLOCK  2               ; Lengte
TLEN:               .BLOCK  2               ; Tijdelijke bewaarplaats voor
                                            ; de lengte


;
;
; Voorbeeld uitvoering
;
;

SC0907:
            ; Zet het beginadres op de stack
                    lda     ADR+1
                    pha
                    lda     ADR
                    pha

            ; Zet de lengte van het te testen blok op de stack
                    lda     SZ+1
                    pha
                    lda     SZ
                    pha

                    jsr     RAMTST          ; Voer de tests uit
                    brk                     ; Resultaat
                                            ;  Overdrachtvlag is hopelijk
                                            ;  0

                    jmp     SC0907

; Data
ADR:                .WORD   2000H
SZ:                 .WORD   1010H


                    .END
```

Sprongtabel - 9H

Verplaatst de controle naar een adres dat aan de hand van een index uit een tabel wordt gekozen. De adressen zijn op de gebruikelijke 6502 wijze opgeslagen (laagstwaardige byte eerst), vanaf adres TABLE. De grootte van de tabel (aantal adressen) is een constante LENSUB, die kleiner dan of gelijk aan 128 moet zijn. Als de index groter is dan of gelijk is aan LENSUB, geeft het programma de controle meteen weer terug, met de overdrachtvlag op 1 gezet.
Procedure: Het programma controleert eerst of de index groter is dan of gelijk is aan de grootte van de tabel (LENSUB). Als dit het geval is, geeft het programma de controle terug, na de overdrachtvlag op 1 te hebben gezet. Als dit niet zo is, haalt het programma het juiste beginadres van de subroutine uit de tabel, slaat het in het geheugen op en springt er indirect naar toe.

Gebruikte registers:	A, P
Looptijd:	31 cycli vast, naast de tijd die nodig is om de subroutine uit te voeren.
Programmalengte:	23 bytes plus 2 * LENSUB bytes voor de tabel met beginadressen. LENSUB is het aantal subroutines.
Benodigd datageheugen:	Twee bytes ergens in RAM (vanaf adres TMP) om het verkregen indirecte adres uit de tabel in op te slaan.
Speciaal geval	Als (A) groter is dan of gelijk is aan LENSUB, stopt het programma meteen met de overdrachtvlag op 1 gezet.

Begincondities

(A) = index

Eindcondities

Als (A) groter is dan LENSUB, een onmiddellijke terugkeer met overdrachtvlag = 1. Anders de controle overgedragen aan de juiste subroutine alsof er een geindexeerde aanroep van een subroutine heeft plaatsgevonden. Het terugkeeradres blijft boven in de stack staan.

Voorbeeld

Data:	LENSUB (grootte van de subroutine-tabel) = 03 Tabel bestaat uit adressen SUB0, SUB1 en SUB2 Index = (A) = 02
Resultaat:	Controle overgedragen aan adres SUB2 (PC = SUB2)

```
;       Titel      : Sprongtabel
;       Naam       : JTAB
;
;
;       Doel       : Gegeven een index in de sprongtabel, spring naar de
;                    bijbehorende subroutine
;
;
;       Begin      : Register A bevat het nummer van de subroutine en kan 0
;                    tot en met LENSUB-1 bevatten
;                    LENSUB moet kleiner of gelijk aan 128 zijn
;
;       Eind       : Als het een legaal subroutine nummer is dan
;                        Voer de subroutine uit
;                    anders
;                        Overdrachtvlag = 1
;
;       Registers  : A, P
;
;       Tijd       : 31 cycli plus de executietijd van de subroutine
;
;       Grootte    : programma 23  bytes plus de lengte van de tabel
;                    (2*LENSUB)
;                    data       2  bytes

JTAB:
                     cmp     #LENSUB
                     bcs     JTABER          ; Spring als A te groot is
                     asl     A               ; BYTEs --> WOORDen
                     tay
                     lda     TABLE,Y         ; Zet beginadres tijdelijk
                                             ; even weg
                     sta     TMP
                     lda     TABLE+1,Y
                     sta     TMP+1
                     jmp     (TMP)           ; Spring met een omweg naar
                                             ; de routine toe


JTABER:
                     sec                     ; Geen goede routine
                     rts


LENSUB:              .EQU    3
TABLE:               .WORD   SUB1
                     .WORD   SUB2
                     .WORD   SUB3


TMP:                 .BLOCK  2               ; Tijdelijke bewaarplaats van
                                             ; het beginadres

            ; De drie subroutines in de tabel

SUB1:
                     lda     #1
                     rts

SUB2:
                     lda     #2
                     rts
```

```
SUB3:
                    lda     #3
                    rts


;
;
; Voorbeeld uitvoering
;
;

SC0908:
                    lda     #0
                    jsr     JTAB
                    brk                     ; Voer routine 0 uit,
                                            ; Register A = 1

                    lda     #1
                    jsr     JTAB
                    brk                     ; Voer routine 1 uit,
                                            ; Register A = 2

                    lda     #2
                    jsr     JTAB
                    brk                     ; Voer routine 2 uit,
                                            ; Register A = 3

                    lda     #3
                    jsr     JTAB
                    brk                     ; Probeer routine 3 uit te
                                            ; voeren,
                                            ; Overdrachtvlag is gezet

                    jmp     SC0908


                    .END
```

Lees een regel karakters van een terminal - 10A

Leest ASCII-karakters van een terminal en bewaart ze in een buffer tot het een 'carriage-return' karakter tegenkomt. Definieert de controlekarakters 'Control H' (08 hex) die het laatst ingevoerde karakter verwijdert en 'Control X' (18 hex) die alle karakters in de buffer verwijdert. Stuurt een bel-karakter (07 hex) als de buffer vol raakt. Kaatst elk karakter dat in de buffer wordt geplaatst terug. Stuurt een nieuwe-regelreeks (meestal carriage-return/wagen terug, line-feed/regel omlaag) naar de terminal voordat het stopt.
RDLINE veronderstelt de aanwezigheid van de volgende, systeem-afhankelijke subroutines:

1. RDCHAR leest een enkel karakter uit de terminal en plaatst het in de accumulator.
2. WRCHAR zendt het karakter in de accumulator naar de terminal
3. WRNEWL zendt een nieuwe-regelreeks (meestal bestaande uit carriage-return- en line-feed karakters) naar de terminal.

Verondersteld wordt dat deze subroutines de inhoud van alle gebruikersregisters veranderen.
RDLINE is bedoeld als voorbeeld van een typische terminal-invoer-afhandelingsroutine. De specifieke controlekarakters en I/O-subroutines in een werkelijk systeem zullen, uiteraard, computer-afhankelijk zijn. Een uitgewerkt voorbeeld in de listing beschrijft een Apple // computer met de volgende specificaties:

1. Het ingangspunt voor de routine die een karakter van het toetsenbord leest is $FD0C_{16}$. Bij terugkeer uit deze routine is bit 7 gezet en moet dus gewist worden voor gewone ASCII handelingen.

2. Het ingangspunt voor de routine die een karakter naar de monitor stuurt is $FDED_{16}$. Deze routine vereist dat bit 7 van het karakter (in de accumulator) gezet is.

3. Het ingangspunt voor de routine die het goede nieuwe-regelkarakter (een carriage-return) geeft, is $FD8E_{16}$.

4. Een 08_{16} karakter verplaatst de cursor een positie naar links.

Een standaardwerk dat de Apple // computer beschrijft is *Apple // User's Guide,* Berkeley: Osborne/McGraw-Hill, 1981.

Procedure: Het programma leest eerst met de RDCHAR-routine een karakter in en stopt als dit een carriage-return is. Als dit niet het geval is, controleert het programma op de bijzondere karakters Control-H en Control-X. Het programma reageert op Control-H door de bufferindex met een te verlagen en een 'backspace' string (cursor naar links, spatie, cursor naar links) naar de terminal te sturen als er iets in de buffer is. Het programma reageert op Control X hetzelfde als met Control H, alleen herhaalt het de handeling tot de buffer leeg is. Als het geen bijzonder karakter is, controleert het programma of de buffer vol is. Is dit het geval, dan stuurt het programma een bel-karakter naar de terminal en gaat verder. Als de buffer (nog) niet vol is, slaat het programma het karakter op in de buffer, echoot het naar de terminal en verhoogt de buffer met een. Voordat het programma stopt, stuurt het via de WRNEWL een nieuwe-regelreeks naar de terminal.

Gebruikte registers: Alle

Looptijd: Ongeveer 67 cycli om een gewoon karakter in de buffer te plaatsen, niet meegerekend de looptijd van de RDCHAR of WRCHAR routines.

Programmalengte: 138 bytes

Benodigd datageheugen: Vier bytes ergens in RAM, plus twee bytes op pagina 0. De vier bytes ergens in RAM bevatten de bufferindex (een byte op adres BUFIDX), de bufferlengte (een byte op adres BUFLEN), de teller voor de 'backspace' routine (een byte op adres INDEX). De twee bytes op pagina 0 bevatten een pointer naar de invoerbuffer (vanaf adres BUFADR, 00D016 in de listing).

Speciale gevallen:

1. Het intoetsen van Control-H (verwijder een karakter) of Control-X (verwijder de hele regel) als er niets in de buffer zit, heeft geen effect op de buffer. Evenmin wordt er iets naar de terminal gestuurd.
2. Als het programma een gewoon karakter ontvangt en de buffer vol is, dan stuurt het een bel-karakter naar de terminal (die daarop een akoestisch signaal laat horen), gooit het ontvangen karakter weg en gaat verder met de normale handelingen.

Begincondities

(A) = Hoogstwaardige byte van het beginadres van de buffer
(Y) = Laagstwaardige byte van het beginadres van de buffer
(X) = Lengte (grootte) van de buffer in bytes

Eindcondities

(X) = Aantal karakters in de buffer

Voorbeelden

1. Data: Regel (van toetsenbord) is 'ENTERcr'

   Resultaat: Bufferindex = 5 (lengte van de regel)
   Buffer bevat 'ENTER'
   'ENTER' terug geechood naar de terminal, gevolgd door een nieuwe-regelreeks (meestal carriage-return en line-feed, of alleen carriage-return).
   Merk op dat 'cr' (carriage-return) niet in de buffer voorkomt.

2. Data: Regel (van toetsenbord) is

   'DMcontrolHNcontrolXENTETcontrolHRcr'.

Resultaat: Bufferindex = 5 (lengte van de feitelijke regel)
Buffer bevat 'ENTER'
'ENTER' terug geechood naar de terminal, gevolgd door een nieuwe-regelreeks (meestal carriage-return en line-feed, of alleen carriage-return).

De volgorde van de handelingen is als volgt:

Ingetoetst karakter	Buffer voor	Buffer na
D	leeg	'D'
M	'D'	'DM'
Control-H	'DM'	'D'
N	'D'	'DN'
Control-X	'DN'	leeg
E	leeg	'E'
N	'E'	'EN'
T	'EN'	'ENT'
E	'ENT'	'ENTE'
T	'ENTE'	'ENTET'
Control-H	'ENTET'	'ENTE'
R	'ENTE'	'ENTER'
cr	'ENTER'	'ENTER'

Dit is wat er gebeurde:

a. De gebruiker typt 'D', 'M'.
b. Hij/zij ziet dat 'M' fout is (moet een 'N' zijn), toetst Control-H om de 'M' te verwijderen' en toetst 'N'.
c. De gebruiker ziet dan dat de eerste 'D' ook al fout was (had een 'E' moeten zijn). Omdat het te verwijderen karakter niet het laatste is, typt de gebruiker Control-X om de hele regel te verwijderen en typt dan 'ENTET'
d. Nu ziet de gebruiker dat ook de tweede 'T' fout is (moet een 'R' zijn), toetst Control-H om de 'T' te verwijderen en toetst 'R'.
e. De gebruiker drukt op carriage-return om de regel te besluiten.

```
;       Titel       : Lees regel
;       Naam        : RDLINE
;
;
;       Doel        : Lees karakters van een input-medium in tot een carriage
;                     return wordt gelezen. RDLINE definieert de volgende
;                     speciale karakters:
;                       -  Control H -- Verwijder het vorige karakter
;                       -  Control X -- Verwijder alle karakters
;
;
;       Begin       : Register A = hoogstwaardige byte van het adres van de
;                     buffer
;                     Register Y = laagstwaardige byte van het adres van de
;                     buffer
;                     Register X = lengte van de buffer
```

```
;
;       Eind       : Register X = het aantal karakters in de buffer
;
;       Registers  : Alle
;
;       Tijd       : Niet van toepassing
;
;       Grootte    : programma 138  bytes
;                    data        4  bytes plus
;                                2  bytes op pagina 0


BUFADR:             .EQU    0D0H        ; Adres van de input buffer

DELKEY:             .EQU    018H        ; Verwijder regel karakter,
BSKEY:              .EQU    08H         ; backspace karakter,
CRKEY:              .EQU    0DH         ; carriage return karakter en
SPACE:              .EQU    020H        ; spatie karakter op het
                                        ; toetsenbord
BELL:               .EQU    07H         ; Karakter op het toetsenbord
                                        ; waardoor de bel rinkelt


RDLINE:
          ; Bewaar de parameters
                    sta     BUFADR+1
                    sty     BUFADR
                    stx     BUFLEN

          ; Zet de buffer-index op nul
INIT:
                    lda     #0
                    sta     BUFIDX

          ; Lees-lus
          ;  Lees karakters in tot een carriage return wordt gelezen
RDLOOP:
                    jsr     RDCHAR      ; Lees een karakter

          ; Keer terug als het een carriage return is
                    cmp     #CRKEY

                    beq     EXITRD

          ; Verwijder het laatste karakter als er een backspace
          ; is gelezen
                    cmp     #BSKEY
                    bne     RDLP1       ; Spring als het geen back-space is
                    jsr     BACKSP      ; Haal laatste karakter weg
                    jmp     RDLOOP      ; Ga door met lezen van karakters

          ; Gooi de regel weg als het een 'delete line' karakter is
RDLP1:
                    cmp     #DELKEY
                    bne     RDLP2       ; Spring als het een andere
                                        ; toets is

DEL1:
                    jsr     BACKSP      ; Haal een karakter weg
                    lda     BUFIDX
                    bne     DEL1        ; Ga door tot buffer leeg is
                    beq     RDLOOP      ; Ga door met inlezen van karakters
```

```
        ; Als we hier aankomen is het een normaal karakter
        ; Als de buffer vol is, laat dan de bel horen, anders
        ; voeg het karakter toe aan de buffer en laat het op
        ; het scherm zien
RDLP2:
                ldy     BUFIDX          ; Is de buffer vol?
                cpy     BUFLEN
                bcc     STRCH           ; Spring als dit niet zo is
                lda     #BELL
                jsr     WRCHAR          ; Laat de bel horen
                jmp     RDLOOP          ; Ga door met inlezen van de regel


STRCH:
                sta     (BUFADR),Y      ; Voeg het karakter toe aan
                                        ; de buffer
                jsr     WRCHAR          ; En laat het op het scherm zien
                inc     BUFIDX          ; Verhoog index in de buffer
                jmp     RDLOOP          ; Ga door met inlezen van karakters


EXITRD:
                jsr     WRNEWL          ; Laat de cursor naar de
                                        ; volgende regel gaan
                ldx     BUFIDX          ; Keer terug met de lengte
                                        ; van de regel in X
                rts


;*************************************************************************
;*                                                                       *
;*          De volgende subroutines zijn machine-afhankelijk,            *
;*          In deze voorbeelden gaan we uit van de Apple II              *
;*                                                                       *
;*************************************************************************


;
; Subroutine : RDCHAR
; Doel       : Lees een karakter, maar laat het niet op het uitvoer-medium
;              zien
; Begin      : Niets nodig
; Einde      : Register A = karakter
; Registers  : Allemaal
;

RDCHAR:
                jsr     0FD0CH          ; Apple Monitor:
                                        ;   Lees-toetsenbord-routine
                and     #01111111B      ; Zet bit 7 op 0
                rts


;
; Subroutine : WRCHAR
; Doel       : Stuur een karakter naar het uitvoer-medium
; Begin      : Register A = karakter
; Einde      : Niets
; Registers  : Allemaal
```

```
WRCHAR:
                ora     #10000000B      ; Zet bit 7 op 1
                jsr     0FDEDH          ; Apple Monitor:
                                        ;   Karakter-uitvoer-routine
                rts


;
; Subroutine : WRNEWL
; Doel       : Stuur de machine-afhankelijke einde van de regel karakters
;              naar het uitvoer-medium.
;              Voor de Apple II is dit een enkele carriage return, maar
;              normaal is dit een carriage return gevolgd door een line-feed
;              karakter
; Begin      : Niets
; Einde      : Niets
; Registers  : Allemaal
;

WRNEWL:
                jsr     0FD8EH          ; Apple Monitor:
                                        ;   Stuur een 'carriage
                                        ;   return' naar het scherm
                rts


;
; Subroutine : BACKSP
; Doel       : Verwijder het vorige karakter
; Begin      : BUFIDX = index naar de eerstvolgende open plaats in de buffer
; Einde      : Het karakter is verwijderd uit de buffer
; Registers  : Allemaal
;

BACKSP:
        ; Test of de buffer leeg is
                lda     BUFIDX
                beq     EXITBS          ; Keer terug als de buffer leeg is

        ; De buffer is niet leeg, verlaag dus de index in de buffer
                dec     BUFIDX

        ; Verwijder het karakter van het uitvoer-medium, door het
        ; met een spatie te overschrijven
                lda     #LENBSS
                sta     COUNT           ; Lengte van uit te sturen string
                lda     #0
                sta     INDEX           ; Begin bij het begin van de string

BSLOOP:
                lda     COUNT
                beq     EXITBS          ; Keer terug als alle
                                        ; karakters verstuurd zijn
                ldy     INDEX
                lda     BSSTRG,Y        ; Haal het volgende karakter op
                jsr     WRCHAR          ; Stuur het naar het output- medium
                inc     INDEX
                dec     COUNT
                jmp     BSLOOP


EXITBS:
                rts
```

```
CSRLFT:         .EQU    08H             ; Karaktercode die de cursor
                                        ; een positie naar links laat gaan
LENBSS:         .EQU    3               ; Lengte van de string
BSSTRG:         .BYTE   CSRLFT,SPACE,CSRLFT   ; De string zelf

; Data
BUFIDX:         .BLOCK  1               ; Index naar het volgende
                                        ; karakter in de buffer
BUFLEN:         .BLOCK  1               ; Lengte van de buffer
COUNT:          .BLOCK  1               ; Teller voor de backspace- routine
INDEX:          .BLOCK  1               ; Indexvariabele voor de
                                        ; backspace-routine


;
;
; Voorbeeld uitvoering
;
;

SC1001:
        ; Lees een regel
                lda     #"?"            ; Zet een prompt op het scherm
                jsr     WRCHAR
                lda     ADRBUF+1        ; Haal het beginadres van de buffer
                                        ;op
                ldy     ADRBUF
                ldx     #LINBUF         ; Haal de lengte van de buffer op
                jsr     RDLINE          ; Lees een regel

        ; Print de regel
                stx     CNT             ; Bewaar het aantal karakters dat
                                        ; in de buffer zit
                lda     #0
                sta     IDX

TLOOP:
                lda     CNT
                bne     TLOOP1          ; Spring als er nog meer
                                        ; karakters te versturen zijn
                jsr     WRNEWL          ; Ga met de cursor naar de
                                        ; volgende regel
                jmp     SC1001          ; En ga door met lezen


TLOOP1:
                ldy     IDX
                lda     INBUFF,Y        ; Haal een karakter op
                jsr     WRCHAR          ; Zet het op het scherm
                inc     IDX             ; Verhoog de index
                dec     CNT             ; Verlaag de teller
                jmp     TLOOP


; Data
IDX:            .BLOCK  1               ; Index
CNT:            .BLOCK  1               ; Teller
ADRBUF:         .WORD   INBUFF          ; Adres van de buffer
LINBUF:         .EQU    10H             ; Lengte van de buffer
INBUFF:         .BLOCK  LINBUF          ; Definieer de buffer zelf


                .END
```

Schrijf een regel karakters naar een uitvoer-apparaat - 10B

Schrijft karakters naar een uitvoer-apparaat, gebruik makend van de computer-afhankelijke WRCHAR routine. WRCHAR schrijft het karakter in de accumulator op het uitoer-apparaat. Gaat door tot het een buffer met gegeven lengte en beginadres heeft geleegd. Deze subroutine is bedoeld als voorbeeld voor een typische uitvoer-sturingsroutine. De specifieke I/O-routines zullen, uiteraard, systeem-afhankelijk zijn. In dit specifieke voorbeeld is de Apple // computer met de volgende kenmerken beschreven:

1. Het ingangspunt voor de routine die een karakter naar de monitor zendt is $FDED_{16}$.

2. Het te schrijven karakter moet met bit 7 op 1 gezet in de accumulator geplaatst worden.

Procedure: Het programma stopt meteen als de bufferlengte nul is. Zo niet, dan stuurt het karakters een-voor-een naar het output-apparaat zolang de buffer niet leeg is. Het programma slaat alle tijdelijke gegevens in het geheugen op in plaats van in registers om afhankelijkheid van de WRCHAR routine te voorkomen.

Gebruikte registers:	Alle
Looptijd:	24 cycli vast, plus 25 cycli per byte (naast de looptijd van de WRCHAR subroutine)
Programmalengte:	37 bytes
Benodigd datageheugen:	Twee bytes ergens in RAM, plus twee bytes op pagina 0. De twee bytes ergens in RAM bevatten de buffer-index (een byte op adres BUFIDX) en de bufferlengte (een byte op adres BUFLEN). De twee bytes op pagina 0 bevatten een pointer naar de uitvoer-buffer (vanaf adres BUFADR, $00D0_{16}$ in de listing).
Speciaal geval	Een bufferlengte van nul zorgt ervoor dat het programma onmiddellijk stopt zonder karakters naar het uitvoer-apparaat te sturen.

Begincondities

(A) = Hoogstwaardige byte van het beginadres van de buffer
(Y) = Laagstwaardige byte van het beginadres van de buffer
(X) = Lengte (grootte) van de buffer in bytes

Eindcondities

Geen

Voorbeeld

Data:	Bufferlengte = 5 Buffer bevat 'ENTER'

Resultaat: 'ENTER' naar uitvoer-apparaat gestuurd

```
;       Titel      : Schrijf regel
;       Naam       : WRLINE
;
;
;       Doel       : Stuur een regel karakters naar het uitvoer-medium
;
;
;       Begin      : Register A = hoogstwaardige byte van het adres van de
;                                 buffer
;                    Register Y = laagstwaardige byte van het adres van de
;                                 buffer
;                    Register X = lengte van de buffer in bytes
;
;       Eind       : Niet van toepassing
;
;       Registers  : Alle
;
;       Tijd       : 24 basiscycli plus
;                    (25 + uitvoertijd van WRCHAR) cycli per byte
;
;       Grootte    : programma 37  bytes
;                    data        2  bytes plus
;                                2  bytes op pagina 0

BUFADR:             .EQU    0D0H          ; Adres van de uitvoer-buffer

WRLINE:
          ; Bewaar de parameters
                    sta     BUFADR+1
                    sty     BUFADR
                    stx     BUFLEN
                    beq     EXIT

          ; Zet de index in de buffer op 0
                    lda     #0
                    sta     BUFIDX

WRLOOP:
                    ldy     BUFIDX
                    lda     (BUFADR),Y    ; Haal het volgende karakter op
                    jsr     WRCHAR        ; Stuur het naar het scherm
                    inc     BUFIDX        ; Verhoog de index in de buffer
                    dec     BUFLEN        ; Verlaag de lengte van de buffer
                    bne     WRLOOP        ; Ga door als we nog niet alle
                                          ; karakters op het scherm hebben
                                          ; staan

;
EXIT:
                    rts

;*************************************************************************
;*                                                                       *
;*             De volgende subroutine is machine-afhankelijk,            *
;*             In deze voorbeelden gaan we uit van de Apple II           *
;*                                                                       *
;*************************************************************************

;
```

```
;
; Subroutine : WRCHAR
; Doel       : Stuur een karakter naar het uitvoer-medium
; Begin      : Register A = karakter
; Einde      : Niets
; Registers  : Alle
;

WRCHAR:
                ora     #10000000B   ; Zet bit 7 op 1
                jsr     0FDEDH       ; Apple Monitor:
                                     ;   Zet een karakter op het scherm
                rts


; Data
BUFIDX:         .BLOCK  1            ; Index naar het volgende
                                     ; karakter in de buffer
BUFLEN:         .BLOCK  1            ; Lengte van de buffer


;
;
; Voorbeeld uitvoering
;
;

SC1002:
        ; Lees een regel met behulp van de Apple Monitor routine
        ; GETLN (0FD6AH)
        ;  33H  = adres van de Apple prompt
        ;  200H = bufferadres
                lda     #"?" or 80H  ; Gebruik ? als prompt, met
                                     ; bit 7 op 1
                sta     033H         ; Zet het prompt-karakter op
                                     ; zijn plaats
                jsr     0FD6AH       ; Apple Monitor: Lees een regel
                stx     LENGTH       ; Lengte van de regel staat
                                     ; in register X


        ; Stuur deze regel naar het uitvoer-medium
                lda     #02H         ; Hoogstwaardige byte van
                                     ; het adres van de buffer
                ldy     #0           ; Laagstwaardige byte van
                                     ; het adres van de buffer
                ldx     LENGTH       ; Lengte van de buffer
                jsr     WRLINE       ; Zet de regel op het scherm
                jsr     0FD8EH       ; Apple Monitor:
                                     ;   Verplaats de cursor naar
                                     ;   het begin van de volgende regel

                jmp     SC1002


; Data
LENGTH:         .BLOCK  1


                .END
```

Genereer even pariteit - 10C

Genereert even pariteit voor een zeven-bits karakter en plaatst het in bit 7. Even pariteit voor een zeven-bits karakter is een bit dat het totale aantal 1-bits in de byte even maakt.
Procedure: Het programma genereert even pariteit door het aantal 1-bits in de zeven laagstwaardige bits van de accumulator te tellen. Dit tellen gebeurt door de data logisch naar links te schuiven en de teller met 1 te verhogen als het bit dat in de Carry wordt geschoven 1 is. Het laagstwaardige bit van de teller is een evenpariteitsbit; het programma besluit door dat bit eerst in de Carry en dan naar bit 7 van de oorspronkelijke data te schuiven.

Gebruikte registers:	A, F
Looptijd:	Maximaal 114 cycli. Hangt af van het aantal 1-bits in de data en hoe snel de reeks logische schuivingen de data nul maakt. Het programma stopt zodra de resterende bits allemaal nul zijn. De looptijd is dus korter als de laagstwaardige bits allemaal nul zijn.
Programmalengte:	39 bytes
Benodigd datageheugen:	Een byte ergens in RAM (op adres VALUE) voor de data

Begincondities

Data in de accumulator (bit 7 wordt niet gebruikt)

Eindcondities

Data met even-pariteit in bit 7 van de accumulator.

Voorbeelden

1. Data: (A) = 42_{16} = 01000010_2 (ASCII B)

   Resultaat: (A) = 42_{16} = 01000010_2 (ASCII B met bit 7 gewist)

   De even-pariteit is 0, omdat 01000010_2 een even aantal 1-bits heeft

2. Data: (A) = 43_{16} = 01000011_2 (ASCII C)

   Resultaat: (A) = $C3_{16}$ = 11000011_2 (ASCII C met bit 7 gezet)

```
;       Titel       : Genereer even pariteit
;       Naam        : GEPRTY
;
;
;       Doel        : Genereer even pariteit in bit 7 voor een 7-bits karakter
;
```

```
;
;       Begin      : Register A = karakter
;
;       Eind       : Register A = karakter met even pariteit
;
;       Registers  : A, F
;
;       Tijd       : 114 cycli maximaal
;
;       Grootte    : programma 39  bytes
;                    data       1  byte

GETPRTY:
            ; Bewaar het karakter
                    sta     VALUE

            ; Bewaar de X en Y registers
                    pha
                    txa
                    pha
                    tya

            ; Tel het aantal bits dat op 1 staat in bits 0 tot en met 6
            ; van het karakter
                    ldy     #0          ; Initialiseer
                    lda     VALUE       ; Haal de data op
                    asl     A           ; Vergeet bit 7, zet het
                                        ;  volgende bit in bit 7
                    sta     VALUE
GETLOOP:
                    bpl     SHFT        ; Spring als bit 7 = 0
                    iny                 ; Anders verhoog het aantal
                                        ;  bits dat op 1 staat met 1

SHFT:
                    asl     A
                    bne     GETLOOP     ; Spring als er meerdere 1-bits
                                        ;  in de byte zitten

                    tya                 ; Bit 0 = aantal 1-bits in de byte
                    lsr     A           ; Schuif de pariteit in de carry
                    lda     VALUE
                    ror     A           ; Roteer eenmaal, zodat de
                                        ;  pariteit in bit 7 komt te staan
                    sta     VALUE

            ; Zet de oude waarden van X en Y terug
                    pla
                    tay

                    pla
                    tax
                    lda     VALUE       ; Haal de waarde met de pariteit op
                    rts

; Data
VALUE:              .BLOCK  1           ; Opslag van de data

;
;
; Voorbeeld uitvoering
;
;
```

```
; Genereer pariteit voor waarden vanaf 0 tot en met 127 en bewaar de
; pariteit in een buffer

SC1003:
                ldx     #0

SC1LP:
                txa
                jsr     GEPRTY          ; Genereer even pariteit
                sta     BUFFER,X         ; Stop de nieuwe waarde in de buffer
                inx
                cpx     #80H
                bne     SC1LP            ; Spring als we nog niet klaar zijn
                brk


; Data
BUFFER:         .BLOCK  128


                .END
```

Controleer pariteit - 10D

Zet de overdrachtvlag op 0 als een databyte even pariteit heeft en op 1 als dat databyte oneven pariteit heeft. Een byte heeft even pariteit als het een even aantal 1-bits heeft en oneven pariteit als het aantal 1-bits oneven is.
Procedure: Het programma telt het aantal 1-bits door de data logisch naar links te schuiven en een teller te verhogen als het bit dat in de overdrachtvlag is geschoven 1 is. Het programma stopt zodra de geschoven data nul wordt (omdat nul duidelijk geen 1-bits kan bevatten). Het laagstwaardige bit van de teller is 0 als de databyte een even aantal 1-bits heeft en is 1 als er een oneven aantal 1-bits in de databyte is. Het programma besluit met het laagstwaardige bit van de teller in de overdrachtvlag te schuiven.

Gebruikte registers:	A, F
Looptijd:	Maximaal 111 cycli. Hangt af van het aantal 1-bits in de data en hoe snel de reeks logische schuivingen de data nul maakt. Het programma stopt zodra de resterende bits allemaal nul zijn. De looptijd is dus korter als de laagstwaardige bits allemaal nullen zijn.
Programmalengte:	25 bytes
Benodigd datageheugen:	Een byte ergens in RAM (op adres VALUE) voor de data

Begincondities

Databyte in de accumulator (bit 7 is in het genereren van de pariteit opgenomen)

Eindcondities

Overdrachtvlag = 0 als de pariteit van de databyte even is, 1 als deze oneven is.

Voorbeelden

1. Data: (A) = 42_{16} = 01000010_2 (ASCII B)

Resultaat: Overdrachtvlag = 0, aangezien 42_{16} (01000010_2) een even aantal (2) 1-bits heeft.

2. Data: (A) = 43_{16} = 01000011_2 (ASCII C)

Resultaat: Overdrachtvlag = 1, aangezien 43_{16} (01000011_2) een oneven aantal (3) 1-bits heeft.

```
;       Titel      : Test pariteit
;       Naam       : CKPRTY
;
;
;       Doel       : Test de pariteit van een byte
;
;
;       Begin      : Register A = Byte met pariteit in bit 7
;
;       Eind       : Overdrachtvlag = 0 als de pariteit even was
;                    Overdrachtvlag = 1 als de pariteit oneven was
;
;       Registers  : A, F
;
;       Tijd       : 111 cycli maximaal
;
;       Grootte    : programma 25  bytes
;                    data       1  bytes

CKPRTY:
          ; Bewaar de data
                    sta     VALUE

          ; Bewaar registers X en Y
                    txa
                    pha
                    tya
                    pha

          ; Tel het aantal bits dat 1 is in VALUE
                    ldy     #0            ; Het aantal 1-bits is 0
                    lda     VALUE

CKLOOP:
                    bpl     SHFT          ; Spring als het volgende bit
                                          ;  0 is (bit 7)
                    iny                   ; Verhoog anders het aantal 1-bits

SHFT:
                    asl     A             ; Schuif volgend bit in bit 7
                    bne     CKLOOP        ; Ga door totdat alle bits 0 zijn

                    tya
                    lsr     A             ; Overdrachtvlag = laagst-
                                          ;  waardige byte van het
                                          ;  aantal bits dat op 1 staat

          ; Zet de oude waarden van X en Y weer goed en keer terug
                    pla
                    tay
                    pla
                    tax
                    rts

 ; Data
 VALUE:             .BLOCK  1


 ;
 ;
 ; Voorbeeld uitvoering
 ;
 ;
```

```
; Test de pariteit voor waarden vanaf 0 tot 255 en bewaar het testresultaat
; in een buffer
;
;     BUFFER[WAARDE] = 0 voor even pariteit
;     BUFFER[WAARDE] = 1 voor oneven pariteit


SC1004:
                ldx     #0
SCLP:
                txa
                jsr     CKPRTY          ; Test pariteit
                lda     #0
                rol     A               ; Zet de pariteit in bit 0
                sta     BUFFER,X        ; Bewaar het resultaat
                inx                     ; Verhoog de waarde
                bne     SCLP            ; Ga door tot we alle waarden
                                        ;  getest hebben
                brk
                jmp     SC1004


; Data
BUFFER:         .BLOCK  256


                .END
```

CRC-16 controle en uitvoering - 10E

Voert een 16-bits 'cyclic redundancy check' (CRC) uit, gebaseerd op het IBM Binaire Synchrone Communicaties (BSC of Bisync) protocol. Het gebruikt de polynoom $X^{16} + X^{15} + X^2 + 1$ om de CRC te genereren. Het ingangspunt ICRC16 initialiseert de CRC op 0 en de polynoom op het juiste bit-patroon. Het ingangspunt CRC16 combineert de vorige CRC met de CRC die vanuit de volgende databyte is gegenereerd. Het ingangspunt GCRC16 geeft de CRC terug.
Procedure: Subroutine ICRC16 initialiseert de CRC op nul en de polynoom op de juiste waarde (een in elke bit-positie overeenkomstig een macht van X die in de polynoom staat). Subroutine CRC16 vernieuwt de CRC overeenkomstig een gespecificeerde data byte. De routine vernieuwt de CRC door de data en de CRC een bit naar links te schuiven en de CRC te 'exclusive OR-en' met de polynoom als de exclusieve OR van het databit en het hoogstwaardige bit van CRC 1 is. Subroutine CRC16 laat de CRC in de geheugenlocaties CRC (laagstwaardige byte) en CRC+1 (hoogstwaardige byte) achter. Subroutine GCRC16 laadt de CRC in de accumulator (hoogstwaardige byte) en indexregister Y (laagstwaardige byte).

Gebruikte Registers:

1. Bij ICRC16: A, F
2. Bij CRC16: Geen
3. Bij GCRC16: A, F, Y

Looptijd:

1. Voor ICRC16: 28 cycli
2. Voor CRC16: Minimaal 302 cycli als er geen 1-bits gegenereerd worden en de polynoom en de CRC nooit een exclusieve OR hoeven ondergaan. 19 extra cycli voor elke keer dat dit wel moet gebeuren. Zodoende is de maximale looptijd 302 + 19 * 8 = 454 cycli.
3. Voor GCRC16: 14 cycli

Programma Grootte:

1. Voor ICRC16: 19 bytes
2. Voor CRC16: 53 bytes
3. Voor GCRC16: 7 bytes

Benodigd datageheugen: Vijf bytes ergens in RAM voor de CRC (twee bytes vanaf adres CRC), de polynoom (twee bytes vanaf adres PLY) en de databyte (een byte op adres VALUE).

Begincondities

1. Voor ICRC16: geen
2. Voor CRC16: databyte in de accumulator, vorige CRC in geheugenlocaties CRC (laagstwaardige byte) en CRC+1 (hoogstwaardige byte), CRC-polynoom in geheugenlocaties PLY (laagstwaardige byte) en PLY+1 (hoogstwaardige byte).
3. Voor GCRC16: CRC in geheugenlocaties CRC (laagstwaardige byte) en CRC+1 (hoogstwaardige byte).

Eindcondities

1. For ICRC16: nul (initiele CRC-waarde) in geheugenlocaties CRC (laagstwaardige byte) en CRC+1 (hoogstwaardige byte). CRC-polynoom in geheugenlocaties PLY (laagstwaardige byte) en PLY+1 (hoogstwaardige byte).
2. Voor CRC16: CRC met huidige databyte verwerkt in geheugenlocaties CRC (laagstwaardige byte) en CRC+1 (hoogstwaardige byte).
3. Voor GCRC16: CRC in de accumulator (hoogstwaardige byte) en indexregister Y (laagstwaardige byte).

Voorbeelden

1. Het genereren van een CRC.

   Roep ICRC16 aan om de polynoom te initialiseren en de CRC vanaf nul te laten beginnen.
   Roep CRC16 aan om de CRC te vernieuwen voor elke databyte die bij de CRC betrokken moet worden.
   Roep GCRC16 aan om de uiteindelijke CRC te verkrijgen (hoogstwaardige byte in A, laagstwaardige byte in Y).

2. Het controleren van een CRC

   Roep ICRC16 aan om de polynoom te initialiseren en de CRC vanaf nul te laten beginnen.
   Roep CRC16 aan om de CRC te vernieuwen voor elke data byte (inclusief de opgeslagen CRC) om deze te kunnen controleren.
   Roep GCRC16 aan om de uiteindelijke CRC te verkrijgen (hoogstwaardige byte in A, laagstwaardige byte in Y). Als er geen fouten zijn opgetreden, horen beide bytes nul te zijn.

Merk op dat alleen subroutine ICRC16 afhankelijk is van de specifieke CRC-polynoom die wordt gebruikt. Voor het wijzigen van de polynoom hoeven alleen de data die ICRC16 in de geheugenlocaties PLY (laagstwaardige byte) en PLY+1 (hoogstwaardige byte) laadt, veranderd te worden.

LITERATUUR

J.E. McNamara, Techical Aspects of Communications, Digital Equipment Corp., Maynard, Mass., 1977. Dit boek bevat een uitleg met betrekking tot het CRC gebeuren. Verder worden de diverse commuinicatie protocols behandeld.

```
;       Titel      : Genereer CRC-16-code
;       Naam       : CRC16
;
;
;       Doel       : Genereer een 16-bits CRC gebaseerd op het synchrone
;                    binaire communicatieprotocol van IBM
;                    De CRC is gebaseerd op de volgende polynoom
;                          (^ betekent "tot de macht")
;
;                              X^16 + X^15 + X^2 + 1
;
;                    Om een CRC-code te genereren:
;                      1)  Roep ICRC16 aan om de CRC-code en de CRC-polynoom
;                          te initialiseren
;                      2)  Roep CRC16 aan voor iedere byte data waarvoor de CRC
;                          moet worden berekend
;                      3)  Roep GCRC16 aan om de resulterende CRC-code te
;                          krijgen,
;                          die daarna aan de data toegevoegd moet worden met
;                          de hoogstwaardige byte voorop
;
;                    Om een CRC-code te testen:
;                      1)  Roep ICRC16 aan om de CRC-code te initialiseren
;                      2)  Roep CRC16 aan voor iedere byte data, en de vorige
;                          twee gegenereerde CRC-bytes
;                      3)  Roep GCRC16 aan om de CRC-code te krijgen. Als er
;                          geen fouten zijn gemaakt levert dit 0 op
;
;
;       Begin      : Register A = Data byte
;
;       Eind       : CRCLO en CRCHI zijn aangepast
;                    Register A = data byte
;
;       Registers  : Geen
;
;       Tijd       : 302 cycli minimaal als er geen bits aan gezet worden
;                    454 cycli maximaal als alle bits 1 worden
;
;       Grootte    : programma  53  bytes
;                    data          5  bytes

CRC16:
          ; Bewaar de data byte
                    sta     VALUE

          ; Bewaar alle registers
                    php
                    pha
                    tya
                    pha
                    txa
                    pha


          ; Loop alle 8 bits langs om de CRC te genereeren
                    ldx     #8              ; 8 bits in een byte

CRCLP:
                    asl     VALUE           ; Bit 7 naar de overdrachtvlag
                    ror     A               ; overdrachtvlag naar bit 7
                    and     #10000000B      ; Maskeer alle andere bits
```

```
                eor     CRC+1           ; Exclusive or van bit 7 met
                                        ;  bit 16 van de CRC
                asl     CRC             ; Schuif CRC 1 bit naar links
                                        ;  (eerst de laagstwaardige
                                        ;  daarna de hoogstwaardige
                                        ;  byte )
                rol     A
                bcc     CRCLP1          ; Spring als de hoogstwaardige
                                        ;  byte van de CRC  1 is

        ; Bit 7 is 1, voer nu een ex-or uit van CRC met de
        ; polynoom
                tay                     ; Bewaar hoogstwaardige byte
                                        ; van de CRC
                lda     CRC
                eor     PLY             ; Exclusive or van de laagst-
                                        ; waardige byte van de CRC met
                                        ; de polynoom
                sta     CRC
                tya
                eor     PLY+1           ; Idem voor de hoogstwaardige
                                        ;  byte

CRCLP1:
                sta     CRC+1           ; Bewaar de hoogstwaardige byte
                dex
                bne     CRCLP           ; Spring als we nog niet alle
                                        ;  8 bytes hebben gehad

        ; Zet de registers weer terug op hun oude waarde en keer terug
                pla
                tax
                pla
                tay
                pla
                plp
                rts

;
; Subroutine : ICRC16
; Doel       : Initieer CRCHI, CRCLO, PLYHI, PLYLO
; Begin      : Geen
; Einde      : CRC en polynoom geinitialiseerd
; Registers  : A, F
;

ICRC16:
                lda     #0
                sta     CRC             ; CRC = 0
                sta     CRC+1
                lda     #5
                sta     PLY             ; PLY = 8005H
                                        ;  8005H is de uitomst van
                                        ;   x^16 + x^15 + x^2 + 1
                                        ;  (1 voor iedere positie
                                        ;  waarvoor een macht in de

                                        ;  formule voorkomt)

                lda     #80H
                sta     PLY+1
                rts
```

```
;
; Subroutine : GCRC16
; Doel       : Haal de CRC-waarde op
; Begin      : Geen
; Einde      : Register A = hoogstwaardige byte van CRC16
;              Register Y = laagstwaardige byte van CRC16
; Registers  : A, F, Y
;

GCRC16:
                lda     CRC+1           ; A = hoogstwaardige byte
                ldy     CRC             ; Y = laagstwaardige byte
                rts

; Data
VALUE:          .BLOCK  1               ; Databyte
CRC:            .BLOCK  2               ; CRC-waarde
PLY:            .BLOCK  2               ; Waarde van de polynoom

;
;
; Voorbeeld aanroepen
;
;

SC1005:
        ; Genereer een CRC voor de waarde 1 en test deze
                jsr     ICRC16
                lda     #1
                jsr     CRC16           ; Genereer een CRC-waarde
                jsr     GCRC16
                tax                     ; Bewaar hoogstwaardige byte
                                        ; van CRC in X
                jsr     ICRC16          ; Initialiseer opnieuw
                lda     #1
                jsr     CRC16           ; Test de CRC door opnieuw
                                        ;  voor 1 de CRC te berekenen
                txa
                jsr     CRC16           ; Neem de CRC-waarde mee in het
                                        ; het resultaat
                tya
                jsr     CRC16
                jsr     GCRC16
                brk                     ; De CRC zou nu 0 moeten zijn

        ; Genereer een CRC voor waarden tussen 0 en 255 en test deze

                jsr     ICRC16
                ldx     #0

GENLP:
                txa                     ; Haal de volgende byte op
                jsr     CRC16           ; Verwerk deze
                inx
                bne     GENLP           ; Ga door zolang er nog
                                        ;  waarden gedaan moeten worden

                jsr     GCRC16          ; Haal de CRC-waarde op
                sta     CRCVAL+1        ; Bewaar de CRC-waarde
                sty     CRCVAL
```

```
        ; Test nu de gegenereerde CRC door hem opnieuw te creeren
                jsr     ICRC16
                ldx     #0
CHKLP:
                txa
                jsr     CRC16
                inx
                bne     CHKLP

        ; Neem nu ook de bewaarde CRC mee in de test
                lda     CRCVAL+1
                jsr     CRC16           ; Hoogstwaardige byte eerst
                lda     CRCVAL
                jsr     CRC16           ; Dan de laagstwaardige byte

                jsr     GCRC16          ; Haal de CRC-waarde op
                brk                     ; De waarde zou 0 moeten zijn

                jmp     SC1005


; Data
CRCVAL:         .BLOCK  2


                .END
```

I/O-apparaattabel-verwerker - 10F

Voert op een apparaat-onafhankelijke wijze in- en uitvoer uit met behulp van I/O-controleblokken en een I/Oapparaattabel. De I/O-apparaattabel bestaat uit een geschakelde lijst; elk element bevat een schakel naar het volgende element, het apparaatnummer en beginadressen voor routines die het apparaat initialiseren, de invoer-status ervan vaststellen, data ervan lezen, de uitvoer-status ervan vaststellen en data ernaartoe schrijven. Een I/O-controleblok is een array dat het apparaatnummer, het handelingsnummer, de status van het apparaat, het beginadres van de buffer van het apparaat en de lengte van deze buffer bevat. De gebruiker moet IOHDLR voorzien van het adres van een geschikt I/O-controleblok en de data als er slechts een byte geschreven hoeft te worden. IOHDLR geeft een kopie van de status-byte en de data terug als er slechts een byte gelezen is.
Deze subroutine is bedoeld als een voorbeeld van hoe invoer en uitvoer op een systeem-onafhankelijke manier verwerkt kunnen worden. De I/O-apparaattabel moet opgebouwd worden met de subroutines INITIO, die de apparaatlijst leeg initialiseert, en ADDDL, die een apparaat aan de lijst toevoegt. Een specifiek voorbeeld voor de Apple //, zet de Apple // eenheid op als apparaat 1 en de printer als apparaat 2; een test routine leest een regel van de computer en echoot de regel naar computer en printer.
Een algemeen programma zal de in- en uitvoer uitvoeren door een I/O-controleblok te verkrijgen of samen te stellen en dan IOHDLR aan te roepen. Subroutine IOHDLR bepaalt dan welk apparaat gebruikt moet worden en hoe de controle door middel van de I/O-apparaattabel aan het sturingsprogramma van het apparaat kan worden overgedragen.
Procedure: Het programma initialiseert eerst de status-byte op 0, waarmee wordt aangegeven dat er geen fouten zijn. Het zoekt dan in de apparaattabel naar het apparaatnummer van het I/O-controleblok. Als in de tabel geen overeenkomst wordt gevonden, stopt het programma met het juiste foutnummer in de status-byte. Als het programma een apparaat met het goede apparaatnummer vindt, controleert het of er een legale handeling wordt gevraagd en geeft, als dit het geval is, de controle over aan de routine die in de apparaattabel staat. Die routine moet daarna de controle teruggeven aan het programma dat de I/O aanriep. Als de handeling illegaal is (het handelingsnummer is te groot of het beginadres van de routine is nul), dan keert het programma terug met foutmelding in de status-byte.
Subroutine INITDL initialiseert de apparaatlijst, waarbij de eerste schakel op nul gezet wordt.
Subroutine ADDL voegt een element bij de apparaatlijst, waarbij het adres ervan aan het hoofd van de lijst wordt gezet en de schakel naar het oude hoofd van de lijst komt te wijzen.

Gebruikte registers:

1. Bij IOHDLR: Alle
2. Bij INITL: A, F
3. Bij ADDL: Alle

Looptijd:

1. Voor IOHDLR: 93 cycli vast plus 59 cycli voor elke niet geslaagde vergelijking van een apparaatnummer
2. Voor INITL: 14 cycli
3. Voor ADDDL: 48 cycli

Programmalengte:

1. Voor IOHDLR: 101 bytes
2. Voor INITL: 9 bytes
3. Voor ADDDL: 21 bytes

Benodigd datageheugen. Drie bytes ergens in RAM, plus zes bytes op pagina 0. De drie bytes ergens in RAM bevatten een indirect adres dat wordt gebruikt om naar een I/O-subroutine te springen (twee bytes vanaf adres OPADR) en het X-register (een byte op adres SVXREG). De zes bytes op pagina 0 bevatten de beginadressen van het I/O-controleblok (twee bytes vanaf adres IOCB), het hoofd van de apparatenlijst (twee bytes vanaf adres DVLST) en het beginadres van het huidige apparaattabel-element (twee bytes vanaf adres CURDEV).

Begincondities

1. Voor IOHDLR:

   (A) = Hoogstwaardige byte van het beginadres van het invoer/uitvoer- controleblok
   (Y) = Laagstwaardige byte van het beginadres van het invoer/uitvoer- controleblok
   (X) = Databyte als er slechts een byte geschreven hoeft te worden.

2. Voor INITL: geen

3. Voor ADDDL:

   (A) = Hoogstwaardige byte van het beginadres van een element van de apparaattabel
   (Y) = Laagstwaardige byte van het beginadres van een element van de apparaattabel

Eindcondities

1. Voor IOHDLR:

   (A) = I/O-controleblok status-byte als er een fout is opgetreden; anders draagt het programma de controle over aan de juiste I/O-sturingsroutine.
   (X) = Databyte als de handeling uit het lezen van een byte bestond.

2. Voor INITL:

   Hoofd van de apparaatlijst (adressen DVLST en DVLST+1) gewist om een lege lijst aan te geven.

3. Voor ADDDL:

   Apparaattabel-element bij de lijst gevoegd

Voorbeeld

Het gegeven voorbeeld heeft de volgende structuur:

INVOER/UITVOER-HANDELINGEN

Handelings-nummer	Handeling
0	Initialiseer apparaat
1	Bepaal invoer-status
2	Lees 1 byte uit invoer-apparaat
3	Lees N bytes uit invoer-apparaat (normaal 1 regel)
4	Bepaal uitvoer-status
5	Schrijf 1 byte naar uitvoer-apparaat
6	Schrijf N bytes naar uitvoer-apparaat (normaal 1 regel)

INVOER/UITVOER-CONTROLE BLOK

Index	Inhoud
0	Apparaatnummer
1	Handelingsnummer
2	Status
3	Laagstwaardige byte van beginadres buffer
4	Hoogstwaardige byte van beginadres buffer
5	Laagstwaardige byte van bufferlengte
6	Hoogstwaardige byte van bufferlengte

APPARAATTABEL-ELEMENT

Index	Inhoud
0	Laagstwaardige byte van schakelveld (beginadres van volgende element)
1	Hoogstwaardige byte van schakelveld (beginadres van volgende element)
2	Apparaatnummer
3	Laagstwaardige byte van beginadres van apparaat initialisatieroutine
4	Hoogstwaardige byte van beginadres van apparaat initialisatieroutine
5	Laagstwaardige byte van beginadres van routine die de invoer-status vaststelt
6	Hoogstwaardige byte van beginadres van routine die de invoer-status vaststelt
7	Laagstwaardige byte van beginadres van invoer-sturingsroutine (lees 1 byte)
8	Hoogstwaardige byte van beginadres van invoer-sturingsroutine (lees 1 byte)
9	Laagstwaardige byte van beginadres van invoer-sturingsroutine (N bytes of 1 regel)
10	Hoogstwaardige byte van beginadres van invoer-sturingsroutine (N bytes of 1 regel)
11	Laagstwaardige byte van beginadres van routine die de uitvoer-status vaststelt

12	Hoogstwaardige byte van beginadres van routine die de uitvoer-status vaststelt
13	Laagstwaardige byte van beginadres van uitvoer-sturingsroutine (schrijf 1 byte)
14	Hoogstwaardige byte van beginadres van uitvoer-sturingsroutine (schrijf 1 byte)
15	Laagstwaardige byte van beginadres van uitvoer-sturingsroutine (schrijf N bytes of 1 regel)
16	Hoogstwaardige byte van beginadres van uitvoer-sturingsroutine (schrijf N bytes of 1 regel)

Als een handeling irrelevant is of niet gedefinieerd is voor een bepaald apparaat (bij voorbeeld een routine die de uitvoer-status vaststelt voor een toetsenbord of een invoer-sturingsroutine voor een printer), moet het corresponderende beginadres in de apparaattabel op nul (d.i. 0000_{16}) gezet worden.

STATUS-WAARDEN

Waarde	Beschrijving
0	Geen fouten
1	llegaal apparaatnummer (apparaat komt niet voor)
2	Data beschikbaar vanuit invoer-apparaat, dergelijke handeling niet voor I/O
3	Uitvoer-apparaat klaar

```
;     Titel      : I/O-apparatentabel-verwerker
;
;     Naam       : IOHDLR
;
;
;     Doel       : Voer een I/O-operatie uit met behulp van 1 aanroep-
                   routine, in plaats van voor elke I/O-operatie de
                   specifieke routine voor het uitvoer-apparaat aan te
                   roepen. Dit kan alleen als apparaten op dezelfde manier
                   aangesproken worden met behulp van een I/O Controle Blok
                   (IOCB) en een tabel waarin de adressen staan van de
                   specifieke routines voor dat apparaat. Met deze routines
                   kunt u het volgende doen:
;
;                  operatienummer              omschrijving
;
;                          0                    Initialiseer een apparaat
;                          1                    Status van de invoer
;                                               bekijken
;                          2                    Lees 1 byte
;                          3                    Lees N bytes
;                          4                    Status van de uitvoer
;                                               bekijken
;                          5                    Schrijf 1 byte
;                          6                    Schrijf N bytes
;
;
;                  Operaties die aan deze lijst toegevoegd zouden kunnen
;                  worden, zijn Open, Sluit af, Verwijder, Hernoem en Voeg
```

```
;               toe. Met deze subroutines zouden dan ook apparaten als
;               floppy-disks aangesproken kunnen worden.
;
;               Een IOCB ziet er als volgt uit :
;
;                       IOCB + 0 = nummer van het apparaat
;                       IOCB + 1 = nummer van de operatie
;                       IOCB + 2 = status
;                       IOCB + 3 = laagstwaardige byte van het beginadres
;                                  van de buffer
;                       IOCB + 4 = hoogstwaardige byte van het beginadres
;                                  van de buffer
;                       IOCB + 5 = laagstwaardige byte van de lengte van
;                                  de buffer
;                       IOCB + 6 = hoogstwaardige byte van de lengte van
;                                  de buffer
;
;               De apparatentabel wordt geimplementeerd als een
;               gekoppelde lijst.
;               Er zijn twee routines die deze lijst bijhouden:
;               INITIO die de apparatuur-lijst initialiseer als een lege
;               lijst, en ADDDL die een apparaat aan de lijst toevoegd.
;               Een tabel van een apparaat heeft de volgende vorm :
;
;             DVTBL + 0 = laagstwaardige byte van het koppelveld
;             DVTBL + 1 = hoogstwaardige byte van het koppelveld
;             DVTBL + 2 = nummer van het apparaat
;             DVTBL + 3 = laagstwaardige byte van initialiseer-routine
;             DVTBL + 4 = hoogstwaardige byte van initialiseer-routine
;             DVTBL + 5 = laagstwaardige byte van invoerstatus-routine
;             DVTBL + 6 = hoogstwaardige byte van invoerstatus-routine
;             DVTBL + 7 = laagstwaardige byte van lees 1 byte routine
;             DVTBL + 8 = hoogstwaardige byte van lees 1 byte routine
;             DVTBL + 9 = laagstwaardige byte van lees N bytes routine
;             DVTBL + 10= hoogstwaardige byte van lees N bytes routine
;             DVTBL + 11= laagstwaardige byte van uitvoerstatus-routine
;             DVTBL + 12= hoogstwaardige byte van uitvoerstatus-routine
;             DVTBL + 13= laagstwaardige byte van schrijf 1 byte routine
;             DVTBL + 14= hoogstwaardige byte van schrijf 1 byte routine
;             DVTBL + 15= laagstwaardige byte van schrijf N bytes routine
;             DVTBL + 16= hoogstwaardige byte van schrijf N bytes routine
;
;
;     Begin       : Register A = hoogstwaardige byte van IOCB
;                   Register Y = laagstwaardige byte van IOCB
;                   Register X = als er geschreven moet worden, wordt er geen
;                                buffer gebruikt, maar slechts een byte
;                                waarin de gegevens staan die verstuurd
;                                moeten worden
;
;
;
;     Einde       : Register A = kopie van de IOCB statusbyte
;                   Register Y = als er gelezen moest worden, wordt er geen
;                                buffer gebruikt, maar slechts een byte
;                                waarin de gegevens staan die ingelezen
;                                werden
;
;                   De status-byte van de IOCB is 0 als de operatie succesvol
;                   beeindigd is; anders staat er de foutcode in.
;
```

```
;                          statuswaarde          omschrijving
;
;                                0               Geen fouten
;                                1               Fout apparaatnummer
;                                2               Invoergegevens klaar, foute
;                                                operatie
;                                3               Uitvoer klaar
;
;
;     Registers  : Alle
;
;     Tijd       : 93 cycli minimaal plus 59 cycli voor ieder apparaat dat
;                  in de tabel is opgenomen, maar dat niet het apparaat is
;                  wat we aan willen spreken
;
;     Grootte    : Programma 131 bytes
;                  Data        3 bytes plus
;                              6 bytes op pagina 0
;
;

; Vaste IOCB- en tabelwaarden
IOCBDN:             .EQU    0           ; IOCB apparaatnummer
IOCBOP:             .EQU    1           ; IOCB operatienummer
IOCBST:             .EQU    2           ; IOCB status
IOCBBA:             .EQU    3           ; IOCB bufferadres
IOCBBL:             .EQU    5           ; IOCB bufferlengte
DTLNK:              .EQU    0           ; Tabel koppelveld
DTDN:               .EQU    2           ; Tabel apparaatnummer
DTSR:               .EQU    3           ; Tabel begin van de routines


; Nummers behorende bij de I/O-operaties
NUMOP:              .EQU    7           ; Operatienummer
INIT:               .EQU    0           ; Initialisering
ISTAT:              .EQU    1           ; Invoerstatus
R1BYTE:             .EQU    2           ; Lees 1 byte
RNBYTE:             .EQU    3           ; Lees N bytes
OSTAT:              .EQU    4           ; Uitvoerstatus
W1BYTE:             .EQU    5           ; Schrijf 1 byte
WNBYTE:             .EQU    6           ; Schrijf N bytes

; Constanten op pagina 0
IOCBA:              .EQU    0D0H        ; Adres van het IOCB
DVLST:              .EQU    0D2H        ; Adres van de lijst met
                                        ;  apparaten
CURDEV:             .EQU    0D4H        ; Beginadres van het huidige
                                        ;  apparaat in de tabel


IOHDLR:
          ; Bewaar IOCB-adres en de waarde van X
                    sta     IOCBA+1
                    sty     IOCBA
                    stx     SVXREG

          ; Initialiseer status-byte op 0 (geen fouten)
                    ldy     #IOCBST
                    lda     #0
                    sta     (IOCBA),Y     ; STATUS = 0
```

```
        ; Zoek naar het gewenste apparaat in de apparatenlijst
                lda     DVLST           ; Begin met het eerste apparaat
                sta     CURDEV
                lda     DVLST+1
                sta     CURDEV+1

        ; Haal het nummer van het gewenste apparaat op
                ldy     #IOCBDN
                lda     (IOCBA),Y
                tax                     ; Zet de waarde in X

SRCHLP:
        ; Test of we het einde van de lijst nog niet hebben bereikt
                lda     CURDEV
                ora     CURDEV+1
                beq     BADDN           ; Koppelveld = 0  -> er zijn
                                        ;  geen andere apparaten meer

        ; Test of dit het gewenste apparaat is
                txa
                ldy     #DTDN
                cmp     (CURDEV),Y      ; Vergelijk het nummer van dit
                                        ;  apparaat met  X
                beq     FOUND           ; Spring als de waarden gelijk
                                        ;  zijn

        ; Ga door naar het adres van de volgende tabelingang
        ;       Maak het huidige apparaat gelijk aan LINK
                ldy     #DTLNK
                lda     (CURDEV),Y      ; Haal de laagstwaardige byte
                                        ; op
                pha                     ; En zet het op de stack
                iny
                lda     (CURDEV),Y      ; Nu de hoogstwaardige byte
                sta     CURDEV+1
                pla                     ; Haal de laagstwaardige byte
                                        ; weer van de stack
                sta     CURDEV
                jmp     SRCHLP          ; Ga door met zoeken


        ; We hebben het apparaat in de tabel gevonden, spring dus
        ; naar de juiste I/O-routine, als ten minste het nummer van
        ; de operatie geldig is
FOUND:          ldy     #IOCBOP
                lda     (IOCBA),Y       ; Haal het nummer van de
                                        ;  operatie op
                cmp     #NUMOP          ; Test of het een geldig
                                        ;  nummer is
                bcs     BADOP           ; Spring als het een slecht
                                        ;  nummer was

        ; Haal het adres van de operatie op
                asl     A               ; Vermenigvuldig met 2 om als
                                        ;  index in de adreslijst te
                                        ;  dienen
                clc
                adc     #DTSR           ; Tel de offset erbij op
                tay                     ; Y is de indexvariabele
                lda     (CURDEV),Y
                sta     OPADR           ; Bewaar de laagstwaardige byte
                iny
```

```
                    lda     (CURDEV),Y
                    sta     OPADR+1         ; Bewaar de hoogstwaardige byte
                    ora     OPADR           ; Test of het adres ongelijk
                                            ;  aan 0 is
                    beq     BADOP           ; Spring als het een illegaal
                                            ;  adres is
                    ldx     SVXREG          ; Zet de bewaarde waarde van X
                                            ;  terug in X
                    jmp     (OPADR)         ; Spring naar de gewenste routine

BADDN:
                    lda     #1              ; Foutcode 1 -- dat apparaat
                                            ;  is niet aanwezig
                    bne     EREXIT

BADOP:
                    lda     #2              ; Foutcode 2 -- die operatie
                                            ;  is niet beschikbaar

EREXIT:
                    ldy      #IOCBST
                    sta     (IOCBA),Y       ; Bewaar de foutcode
                    rts


;
;       Subroutine : INITDL
;
;       Doel       : Initialiseer de apparatentabel als een lege lijst
;
;       Begin      : Geen
;
;       Eind       : Een lege apparatentabel
;
;       Registers  : A, F
;

INITDL:
                    lda     #0
                    sta     DVLST
                    sta     DVLST+1
                    rts


;
;       Subroutine : ADDDL
;
;       Doel       : Voeg een apparaat toe aan de lijst
;
;       Begin      : Register A = hoogstwaardige adres van een tabelingang
;                    Register Y = laagstwaardige adres van een tabelingang
;
;       Einde      : Het apparaat is in de tabel opgenomen
;
;       Registers  : Allemaal

ADDDL:
            ; X, Y = nieuwe tabelingang
                    tax

            ; Zet de huidige kop van de tabel op de stack
                    lda     DVLST+1
```

```
                pha                     ; Zet de hoogstwaardige byte
                                        ; van de huidige kop van de
                                        ; tabel op de stack
                lda     DVLST
                pha                     ; De laagstwaardige byte ook

        ; Maak de nieuwe tabelingang de nieuwe kop van de lijst
                sty     DVLST
                stx     DVLST+1

        ; Koppel de nieuwe kop van de tabel aan de oude kop vast
                pla
                ldy     #0
                sta     (DVLST),Y       ; Bewaar de laagstwaardige byte
                pla
                iny
                sta     (DVLST),Y       ; De hoogstwaardige byte ook
                rts

; Data
OPADR:          .BLOCK  2               ; Adres van de routine
                                        ;  behorende bij de operatie
SVXREG:         .BLOCK  1               ; Tijdelijke bewaarplaats van
                                        ;  de inhoud van X


;
;
; Voorbeeld uitvoering
;
;       Deze routines maken een apparatentabel met als eerste apparaat de
;       Apple console en als tweede apparaat de Apple printer (in slot 1). De
;       test- routine gaat dan een regel inlezen van de console, en stuurt
;       daarna deze regel naar de printer en de console.

CR:             .EQU    08DH            ; Apple II wagen-terug karakter
CBUF:           .EQU    0D6H            ; Beginadres I/O-buffer

SC1006:
        ; Initialiseer de tabel
                jsr     INITDL

        ; Voeg de console toe aan de tabel
                lda     CONDVA+1
                ldy     CONDVA
                jsr     ADDDL           ; Voeg de console toe
                lda     #INIT           ; Initialiseer operatie
                sta     IOCB+IOCBOP
                lda     #1              ; Apparaatnummer = 1
                sta     IOCB+IOCBDN
                lda     AIOCB+1
                ldy     AIOCB
                jsr     IOHDLR          ; Initialiseer de console

        ; Voeg de Apple printer toe als apparaat 2
                lda     PRTDVA+1
                ldy     PRTDVA
                jsr     ADDDL           ; Voeg de printer toe
                lda     #INIT           ; Initialiseer operatie
                sta     IOCB+IOCBOP
                lda     #2              ; Apparaatnummer = 2
                sta     IOCB+IOCBDN
```

```
                lda     AIOCB+1
                ldy     AIOCB
                jsr     IOHDLR          ; Initialiseer de printer

        ; Lees regels van de console en stuur ze naar het scherm
        ;  en de printer totdat een lege regel wordt ingetikt
TSTLP:
                lda     #1              ; I/O op de console
                sta     IOCB+IOCBDN
                lda     #RNBYTE         ; Lees N bytes operatie
                sta     IOCB+IOCBOP
                lda     #LENBUF
                sta     IOCB+IOCBBL     ; Zet de bufferlengte goed
                lda     #0
                sta     IOCB+IOCBBL+1
                lda     AIOCB+1
                ldy     AIOCB
                jsr     IOHDLR          ; Lees een regel in

        ; Stuur de regel naar de console
        ;  Het apparaat is nog steeds gelijk aan hierboven
                lda     #WNBYTE         ; Schrijf N bytes operatie
                sta     IOCB+IOCBOP
                lda     AIOCB+1
                ldy     AIOCB
                jsr     IOHDLR          ; Schrijf de regel op de
                                        ;  console

        ; Stuur een wagen-terug karakter naar de console
                ldx     #CR
                lda     #W1BYTE         ; Schrijf 1 byte
                sta     IOCB+IOCBOP
                lda     AIOCB+1
                ldy     AIOCB
                jsr     IOHDLR          ; Schrijf een karakter

        ; Stuur de regel naar de printer
                lda     #2              ; I/O op de printer
                sta     IOCB+IOCBDN
                lda     #WNBYTE         ; Schrijf N bytes
                sta     IOCB+IOCBOP
                lda     AIOCB+1
                ldy     AIOCB
                jsr     IOHDLR          ; Schrijf de regel op de
                                        ;  printer

        ; Stuur een wagen-terug karakter naar de printer
                ldx     #CR
                lda     #W1BYTE         ; Schrijf 1 byte
                sta     IOCB+IOCBOP
                lda     AIOCB+1
                ldy     AIOCB
                jsr     IOHDLR          ; Schrijf een karakter

                lda     IOCB+IOCBBL     ; Haal de laagstwaardige byte
                                        ; op
                ldy     #1
                ora     IOCB+IOCBBL,Y; Logische-OR met de hoogst-
                                        ; waardige byte
                bne     TSTLP           ; Spring als de regel niet
                                        ;  leeg was
                brk
```

```
                jmp     SC1006

; IOCB voor het uitvoeren van I/O-instructies
AIOCB:          .WORD   IOCB
IOCB:
                .BLOCK  1               ; Apparaatnummer
                .BLOCK  1               ; Operatienummer
                .BLOCK  1               ; Status
                .WORD   BUFFER          ; Adres van de buffer
                .WORD   LENBUF          ; Lengte van de buffer

; De buffer zelf
LENBUF:         .EQU    127
BUFFER:         .BLOCK  LENBUF


; Apparatentabel-ingangen
CONDVA:         .WORD   CONDV           ; Adres van de console-
                                        ;  gegevens
CONDV:
                .WORD   0               ; Koppelveld
                .BYTE   1               ; Apparaatnummer = 1
                .WORD   CINIT           ; Initialiseer
                .WORD   CINSTAT         ; Invoerstatus
                .WORD   CIN             ; Lees 1 byte
                .WORD   CINN            ; Lees N bytes
                .WORD   COSTAT          ; Uitvoerstatus
                .WORD   COUT            ; Schrijf 1 byte
                .WORD   COUTN           ; Schrijf N bytes

PRTDVA:         .WORD   PRTDV           ; Adres van de printer-
                                        ;  gegevens
PRTDV:
                .WORD   0               ; Koppelveld
                .BYTE   2               ; Apparaatnummer = 2
                .WORD   PINIT           ; Initialiseer
                .WORD   0               ; Geen invoerroutines op
                .WORD   0               ; een printer
                .WORD   0
                .WORD   POSTAT          ; Uitvoerstatus
                .WORD   POUT            ; Schrijf 1 byte
                .WORD   POUTN           ; Schrijf N bytes

;
;
;       CONSOLE I/O-ROUTINES
;
;

; Initialiseer de console

CINIT:
                lda     #0              ; Geen fouten
                rts                     ; Console heeft geen
                                        ;  initialisatie nodig

; Invoerstatus op de console

CISTAT:
                lda     0C000H          ; Haal de toetsenbord-status-
                                        ;  byte op
                bpl     CNONE           ; Spring als er geen karakter
                                        ;  aanwezig is
```

```
                lda     #2          ; Geef aan dat er een karakter
                                    ;  aanwezig is
                bne     CIS1        ; Spring altijd
CNONE:
                lda     #0          ; Niet gereed
CIC1:
                ldy     #IOCBST
                sta     (IOCBA),Y   ; Bewaar de status
                rts

; Lees 1 byte van de console

CIN:
                lda     C000H
                bpl     CIN         ; Wacht tot er een karakter
                                    ;  aanwezig is
                txa                 ; Bewaar het in X
                lda     #0          ; Geen fouten gemaakt
                rts


; Lees N bytes van de console

CINN:
        ; Lees een regel met behulp van de Apple Monitor GETLN
        ;  routine op 0FD6AH
        ;       33H = prompt-locatie
        ;       200H = adres van de buffer

                lda     #"?" OR 80H
                sta     033H        ; Zet het Apple prompt-karakter

                jsr     0FD6AH      ; Apple Monitor :
                                    ;   Lees een regel van de
                                    ;    console

        ; Test of de ingelezen regel wel in de buffer past
                ldy     #IOCBBL+1
                lda     (IOCBA),Y   ; Haal de hoogstwaardige byte
                                    ; op
                bne     CINN1       ; Spring als hij ongelijk aan
                                    ;  0 is
                dey
                txa
                cmp     (IOCBA),Y
                bcc     CINN1       ; Spring als het aantal
                                    ;  ingelezen karakters kleiner
                BEQ     CINN1       ; dan of gelijk aan de buffer-
                                    ;  lengte is
                lda     (IOCBA),Y
                tax                 ; Kopieer net zoveel karakters
                                    ;  als er in onze buffer passen

CINN1:
                txa
                sta     (IOCBA),Y   ; Maak de bufferlengte gelijk
                                    ;  aan het aantal ingelezen
                                    ;  karakters
                lda     #0
                iny
                sta     (IOCBA),Y   ; Maak de hoogstwaardige byte
                                    ; 0
```

```
        ; Verplaats de data van de Apple buffer op 200H naar onze
        ;  eigen buffer
                ldy     #IOCBBA         ; Haal het adres van onze
                                        ;  buffer op
                lda     (IOCBA),Y
                sta     CBUF            ; En bewaar dat adres in CBUF
                iny
                lda     (IOCBA),Y
                sta     CBUF+1
                txa
                beq     CINN3           ; Spring als het aantal te
                                        ;  kopieeren bytes = 0
                ldy     #0

        ; Verplaats nu werkelijk de data
CINN2:
                lda     200H,Y          ; Haal een byte op
                sta     (CBUF),Y        ; Stop hem in onze buffer
                iny                     ; Verhoog de index
                dex                     ; Verlaag de teller
                bne     CINN2           ; Ga door totdat teller = 0

        ; Geen fouten gemaakt, dus A = 0
CINN3:
                lda     #0
                rts

; Uitvoerstatus van de console

COSTAT:
                lda     #3              ; Altijd klaar om iets uit te
                                        ; sturen

                rts


; Schrijf 1 byte naar de console

COUT:
                txa
COUT1:
                jsr     0FDEDH          ; Apple Monitor:
                                        ;  Schrijf 1 karakter naar de
                                        ;  console
                lda     #0              ; Geen fouten
                rts

COUT1A:         .WORD   COUT1

; Schrijf N bytes naar de console

COUTN:
                lda     COUT1A+1        ; A,Y = adres van de uitvoer-
                                        ;  routine
                ldy     COUT1A
                jsr     OUTN            ; Roep de uitvoer-routine aan
                lda     #0              ; Geen fouten
                rts

;
;
;     PRINTER I/O-ROUTINES
```

```
;       Veronderstel dat de printerkaart in slot 1 zit
;
;


; Initialiseer de printer
PINIT:
                    lda     #0          ; De printer heeft geen
                                        ;  initialisatie nodig
                    rts                 ; Geen fouten


; Printer-uitvoerstatus

POSTAT:
                    lda     #0          ; Neem aan dat de printer
                                        ;  altijd klaar is om iets te
                                        ;  printen
                    rts

; Schrijf 1 byte naar de printer

POUT:
                    txa
POUT1:
                    jsr     0C107H      ; Apple Monitor:
                                        ;  Schrijf een karakter naar
                                        ;  de printer
                    lda     #0          ; Geen fouten
                    rts


POUT1A:             .WORD   POUT1       ; Adres van de karakter-uit-
                                        ;  voer-routine


; Schrijf N karakters naar de printer

POUTN:
                    lda     POUT1A+1    ; A,Y = adres van de karakter-
                                        ;  uitvoer-routine
                    ldy     POUT1A
                    jsr     OUTN        ; Schrijf de karakters naar
                                        ;  de printer
                    lda     #0          ; Geen fouten
                    rts



;
;
;       Subroutine : OUTN
;
;       Doel       : Schrijf N karakters naar een uitvoer medium
;
;       Begin      : Register A = hoogstwaardige byte van het adres van de
;                                 uitvoer-routine
;                    Register Y = laagstwaardige byte van het adres van de
;                                 uitvoer-routine
;                    IOCBA      = beginadres van een IOCB
;
```

```
;       Einde       : De data is weggeschreven
;
;       Register    : Allemaal
;
;

OUTN:
        ; Bewaar het adres van de uitvoer-routine
                sta     COSR+1
                sty     COSR

        ; Haal het adres van de uitvoer-buffer op en bewaar het op
        ; pagina 0
                ldy     #IOCBBA
                lda     (IOCBA),Y
                sta     CBUF
                iny
                lda     (IOCBA),Y
                sta     CBUF+1

        ; Haal de bufferlengte uit de IOCB, keer terug als deze 0 is
                ldy     #IOCBBL
                lda     (IOCBA),Y
                sta     BUFLEN
                iny
                lda     (IOCBA),Y
                sta     BUFLEN+1
                ora     BUFLEN
                beq     OUT3            ; Spring als de lengte van de
                                        ;  buffer 0 is

        ; Begin aan het begin van de buffer
                lda     #0
                sta     IDX
OUTLP:
                ldy     IDX
                lda     (CBUF),Y        ; Haal het volgende karakter
                                        ;  op
                jsr     LP0             ; Schrijf het weg
                jsr     LP1

LP0:            jmp     (COSR)          ; Schrijf het karakter met
                                        ;  behulp van de meegegeven
                                        ;  routine

LP1:
        ; Verhoog de index naar het volgende karakter in de buffer
                inc     IDX
                bne     LP2
                inc     CBUF+1          ; Verhoog hoogstwaardige byte
                                        ; als dat nodig is

        ; Verlaag de lengte van de buffer en ga door als de lengte
        ;  ongelijk aan 0 is

LP2:
                lda     BUFLEN
                bne     DECLS
                dec     BUFLEN+1        ; Leen van de hoogstwaardige
                                        ;  byte
DECLS:
                dec     BUFLEN          ; Verlaag deze altijd
```

```
                bne     OUTLP
                lda     BUFLEN+1
                bne     OUTLP           ; Ga door totdat alle karakters
                                        ; verzonden zijn

OUT3:
                rts


; Data
COSR:           .WORD   0               ; Adres van de uitvoerroutine
                                        ;  voor 1 karakter
BUFLEN:         .WORD   0               ; Lengte van de buffer
IDX:            .BYTE   0               ; Index in de buffer


                .END
```

Initialiseer I/O-poorten - 10G

Initialiseert een set I/O-poorten vanuit een array met poortadressen en initiele waarden. Er worden voorbeelden gegeven hoe programmeerbare I/O-apparaten als de 6520 Peripheral Interface Device Adapter (PIA), de 6522 Versatile Interface Adapter (VIA), het 6530 Multifunction Device, het 6532 Multifunction Device,de 6551 Asynchronous Communications Device Adapter en de 6850 Asynchronous Communications Device Adapter geinitialiseerd kunnen worden.
Deze subroutine is bedoeld als een algemene methode om I/O-poorten te initialiseren. De initialisatie kan zich uitstrekken tot datapoorten, data-richtingsregisters die bepalen of bits in- dan wel uitvoer zijn, controle- of commando registers die de werkingsmodus van programmeerbare apparaten bepalen, tellers (bij timers), prioriteitsregisters en andere externe registers of opslaglocaties.
Enkele handelingen die de gebruiker met deze routine kan uitvoeren zijn:

1. Bepaal bidirectionele I/O-lijnen als in- of uitvoer.
2. Initialiseer uitvoerpoorten op bekende beginwaarden.
3. Schakel interrupts van randchips in of uit.
4. Bepaal werkingsmodi; zijn poorten al dan niet gelatcht, moeten er strobes gegenereerd worden, wat heeft welke prioriteit, moeten timers voortdurend of op afroep werken, etc.
5. Laad initiele waarden in tellers.
6. Kies bit (baud) snelheden voor communicatie.
7. Wis of reset apparaten die niet aan de algemene systeem-resetlijn gekoppeld zijn.
8. Initialiseer prioriteitsregisters of ken initiele prioriteiten toe aan interrupts of andere handelingen.
9. Initialiseer vectoren zoals deze worden gebruikt bij het afhandelen van interrupts, DMA-verzoeken en andere invoer.

Procedure: Het programma lust door het opgegeven aantal poorten, waarbij het het poortadres en de initiele waarde uit het array haalt en de initiele waarde in het poortadres opslaat. Deze procedure is niet afhankelijk van het type apparaat dat in het I/O-deel wordt gebruikt, of van het aantal apparaten. Initialisatieprocedures voor apparaten kunnen worden bijgevoegd of verwijderd door middel van de juiste wijzigingen in het array en de parameters van de routine; de routine zelf blijft ongewijzigd.

Gebruikte registers:	Alle
Looptijd:	16 cycli vast, plus 52 cycli per poort. Als, bij voorbeeld, AANTAL POORTEN = 10, dan is de looptijd 52 * 10 + 16 = 520 + 16 = 536 cycli
Programmalengte:	40 bytes plus de grootte van de tabel (drie bytes per onderdeel)
Benodigd datageheugen:	Vier bytes op pagina 0, twee voor een pointer naar het array (vanaf adres ARYADR, $00D0_{16}$ in de listing) en twee voor een pointer naar de poort (vanaf adres PRTADR, $00D2_{16}$ in de listing).

Begincondities

(A) = Hoogstwaardige byte van het beginadres van het array met poorten en initiele waarden
(Y) = Laagstwaardige byte van het beginadres van het array met poorten initiele waarden

(X) = Aantal array-elementen (aantal te initialiseren poorten)

Eindcondities

Alle poorten geinitialiseerd.

Voorbeeld

Data: Aantal te initialiseren poorten = 3
Array elementen zijn:

Hoge byte van adres poort 1
Lage byte van adres poort 1
Initiele waarde voor poort 1

Hoge byte van adres poort 2
Lage byte van adres poort 2
Initiele waarde voor poort 2

Hoge byte van adres poort 3
Lage byte van adres poort 3
Initiele waarde voor poort 3

Resultaat: Initiele waarde voor poort 1 in adres van poort 1 opgeslagen

Initiele waarde voor poort 2 in adres van poort 2 opgeslagen
Initiele waarde voor porrt 3 in adres van poort 3 opgeslagen

Merk op dat elk element van het array uit 3 bytes bestaat met de volgende inhoud:

Laagstwaardige byte van poortadres
Hoogstwaardige byte van poortadres

Initiele waarde voor poort

```
;       Titel       : Initialiseer I/O-poorten
;
;       Naam        : IPORTS
;
;
;       Doel        : Initialiseer I/O-poorten met behulp van een array van
;                     poortadressen en -waarden
;
;       Begin       : Register A = hoogstwaardige byte van het adres van het
;                                  array
;                     Register Y = laagstwaardige byte van het adres van het
;                                  array
;                     Register X = nummer van de te initialiseren poort
;
;                     Het array bestaat uit elementen (3 bytes groot)
```

```
;                   array+0 = hoogstwaardige adres van poort 1
;                   array+1 = laagstwaardige adres van poort 1
;                   array+2 = waarde die in adres van poort 1 gezet moet
;                             worden
;                   array+3 = hoogstwaardige adres van poort 2
;                   array+4 = laagstwaardige adres van poort 2
;                   array+5 = waarde die in adres van poort 2 gezet moet
;                             worden
;                                                .
;                                                .
;                                                .
;                                                .
;
;
;     Einde       : Geen
;
;     Registers   : Alle
;
;     Tijd        : 16 basiscycli plus
;                   52 cycli per te initialiseren poort
;
;     Grootte     : Programma 40 bytes
;                   Data        2 bytes op pagina 0
;
;


ARYADR:             .EQU    0D0H        ; Adres van het array
PRTADR:             .EQU    0D2H        ; Adres van de poort


IPORTS:
          ; Bewaar het beginadres van het array
                    sta     ARYADR+1
                    sty     ARYADR

          ; Keer terug als het aantal te initialiseren poorten 0 is
                    txa
                    beq     EXITIP

          ; Loop nu het array na en verstuur de waarden naar de bij-
          ;  behorende poorten totdat alle poorten geinitialiseerd zijn

LOOP:
          ; Haal het adres van de poort uit het array en bewaar het
                    ldy     #0
                    lda     (ARYADR),Y  ; Haal het laagstwaardige byte
                                        ; van het adres van de poort op
                    sta     PRTADR
                    iny
                    lda     (ARYADR),Y  ; Hoogstwaardige byte
                    sta     PRTADR+1

          ; Haal de waarde die verstuurd moet worden op en stuur haar
          ;  naar de poort
                    iny
                    lda     (ARYADR),Y  ; Haal de waarde op
                    ldy     #0
                    sta     (PRTADR),Y  ; Stuur haar naar de poort

          ; Ga wijzen naar het volgende element van het array
                    lda     ARYADR
```

```
                clc
                adc     #3              ; Verhoog met 3
                sta     ARYADR
                bcc     LOOP1
                inc     ARYADR+1        ; Verhoog de hoogstwaardige
                                        ; byte als de overdrachtvlag
                                        ; 1 was
LOOP1:
        ; Verlaag het aantal poorten met 1 en keer terug als alle
        ;  poorten geinitialiseerd zijn
                dex
                bne     LOOP

EXITIP:
                rts



;
;
;       Voorbeeld uitvoering
;
;

;
;       Initialiseer
;               6520    PIA
;               6522    VIA
;               6530    ROM/RAM/IO/TIMER
;               6532    RAM/IO/TIMER
;               6850    serieele poort (ACIA)
;               6851    serieele poort (ACIA)

SC1007:
                lda     ADRARY+1
                ldy     ADRARY
                ldx     SZARY
                jsr     IPORTS          ; Initialiseer de poorten
                brk


ARRAY:

      ; Initialiseer 6520

      ;  Neem aan dat de registers staan vanaf adres 2000H
      ;       Poort A = invoer
      ;       CA1     = Data gereed, wordt gezet door een laag
      ;                 naar hoog puls, geen interrupts
      ;       CA2     = Data in orde lijn

              .WORD   2001H           ; 6520 Controleregister A
              .BYTE   00000000B       ; Volgende toegang is in het
                                      ;  data-richtingregister
                                      ;  (= dataregister)
              .WORD   2000H           ; 6520 Dataregister A
              .BYTE   00000000B       ; Invoer bestaat uit alle bits
              .WORD   2001H           ; 6520 Controleregister A
              .BYTE   00100110B       ; Zet CA1,CA2 goed en plaats
                                      ;  bit 2 op het dataregister

      ;       Poort B = uitvoer
```

```
;       CB1     = Data in orde lijn
;       CB2     = Data gereed, wordt laag als er data in
;                 register B komen en hoog als er een hoog
;                 naar laag puls op CB1 wordt gezet
          .WORD   2003H         ; 6520 Controleregister B
          .BYTE   00000000B     ; Volgende toegang is in het
                                ;  data-richtingregister
                                ;  (= dataregister)
          .WORD   2002H         ; 6520 Dataregister B
          .BYTE   11111111B     ; Alle bits = uitvoer
          .WORD   2003H
          .BYTE   00100100B     ; Zet CA1,CA2 goed en plaats
                                ;  bit 2 op het dataregister

; Initialiseer 6522

    ; Neem aan dat de registers staan vanaf 2010H
    ;       Poort A = bits 0..3 = uitvoer, bits 4..7 = input
    ;       CA1 en CA2 worden niet gebruikt
    ;       Poort B = gebufferde invoer
    ;       CB1     = Data gereed, wordt gezet door een laag
    ;                 naar hoog puls
    ;       CB2     = data in orde lijn
          .WORD   2013H         ; Data-richtingregister A
          .BYTE   00001111B     ; Bits 0..3 = uitvoer
                                ; Bits 4..7 = invoer
          .WORD   2012H         ; Data-richting register B
          .BYTE   00000000B     ; Bits 0..7 = invoer
          .WORD   201CH         ; Randapparatuur-controle-
                                ;  register
          .BYTE   10010000B     ; Zet CB1 en CB2 goed
          .WORD   201BH         ; Hulpcontroleregister
          .BYTE   00000010B     ; Poort B buffert de invoer-
                                ;  data

; Initialiseer 6530

    ; Neem aan dat de registers staan vanaf adres 2020H
    ;       Poort A = uitvoer
    ;       Poort B = invoer
          .WORD   2021H         ; Data-richtingregister A
          .BYTE   11111111B     ; Bits 0..7 = uitvoer
          .WORD   2023H         ; Data-richtingregister B
          .BYTE   00000000B     ; Bits 0..7 = invoer


; Initialiseer 6532

    ; Neem aan dat de registers beginnen vanaf 2030H
    ;       Poort A = bits 0..6 = uitvoer
    ;                 bit 7 = invoer op poort B aanwezig
    ;       Poort B = invoer
          .WORD   2031H         ; Data-richtingregister A
          .BYTE   01111111B     ; Bits 0..6 = uitvoer
                                ; Bit 7 = invoer
          .WORD   2033H         ; Data-richtingregister B
          .BYTE   00000000B     ; BITS 0..7 = invoer

; Initialiseer 6551

    ; Neem aan dat de registers beginnen vanaf 2040H
    ;       8 bits data, geen pariteit
```

```
        ;       1 stop bit
        ;       9600 baud van de baud-rate generator die op het board zit
        ;       Geen interrupts
                .WORD   2041H           ; Reset de 6551 door het status
                                        ; register te beschrijven
                .BYTE   0               ; Maakt niet uit wat hier staat
                .WORD   2042H           ; Controle register
                .BYTE   10011110B       ; 1 stop, 8 bits data, intern 9600
                                        ; baud
                .WORD   2043H           ; Commandoregister
                .BYTE   00000011B       ; Geen pariteit, geen echo, geen
                                        ; interrupts van de ontvanger, DTR
                                        ; laag

    ; Initialiseer 6850

        ; Neem aan dat de registers beginnen vanaf adres 2050H
        ;       8 bits, geen pariteit
        ;       1 stop bit
        ;       deel de master-clock door 1
        ;       geen interrupts
                .WORD   2050H           ; Schrijf naar het controle-
                                        ;  register
                .BYTE   00000011B       ; Reset
                .WORD   2050            ; Controleregister
                .BYTE   00010101B       ; Geen interrupts, RTS laag,
                                        ;  8 bits, 1 stop, deel door
                                        ;  1


ENDARY:                                 ; Einde van het array
ADRARY:         .WORD   ARRAY           ; Beginadres van het array
SZARY:          .BYTE   (ENDARY - ARRAY) / 3
                                        ; Het aantal poorten dat
                                        ;  geinitialiseerd moet worden

                .END
```

Vertraag milliseconden - 10H

Geeft een vertraging van tussen 1 en 255 milliseconden, afhankelijk van de door de gebruiker gegeven parameter. De gebruiker moet de waarde MSCNT berekenen voor de eigen computer. De formule hiervoor is

MSCNT = (100 / KLOKSNELHEID - 10) / 5 = 200 / KLOKSNELHEID - 2

KLOKSNELHEID is het aantal microseconden per klokperiode voor een specifieke computer (1 voor KIM-1, SYM-1 en AIM-65, 0.9799269 voor Apple //tm).
Procedure: Het programma telt eenvoudigweg de indexregisters af gedurende de tijd die bepaald wordt door de, door de gebruiker meegegeven, waarde. Een
paar NOP-instructies zorgen voor extra vertraging met betrekking tot de aanroep- en terugkeer-instructie.

Gebruikte registers:	X, Y, P
Looptijd:	1 milliseconde * (Y). Als (Y) = 0, is de minimumtijd 17 cycli, inclusief een JSR-instructie.
Programmalengte:	156 bytes
Benodigd datageheugen:	Geen
Speciaal geval:	(Y) = 0 veroorzaakt een terugkeer na een minimale looptijd van 17 cycli, inclusief een JSR-instructie. (Y) = 0 en (X) is ongewijzigd.

Begincondities

(Y) = Aantal milliseconden vertraging (1 tot 255)

Eindcondities

Keert terug na opgegeven vertraging met (X) = (Y) = 0

Voorbeeld

Data: (Y) = aantal milliseconden = $2A_{16} = 42_{10}$

Resultaat: Programmatische vertraging van $2A_{16}$ (42_{10}) milliseconden, aangenomen dat de gebruiker de juiste waarde voor MSCNT heeft gegeven.

```
;       Titel       : Wacht een paar milliseconden
;
;       Naam        : DELAY
;
;
;       Doel        : Wacht van 1 tot 255 milliseconden
```

```
;
;     Begin       : Register Y = aantal milliseconden dat we moeten wachten
;
;     Einde       : Keert terug naar de routine die ons aanriep als we het
;                   opgegeven aantal milliseconden hebben gewacht
;
;
;     Registers   : X, Y, P
;
;     Tijd        : 1 milliseconde * Register Y
;                   Als Y = 0 dan is de tijd minimaal 17 cycli inclusief de
;                   tijd die nodig is om de JSR-instructie uit te voeren
;
;     Grootte     : Programma 29 bytes
;                   Data         GEEN
;

; De volgende formule moet worden gebruikt om de wachttijden te berekenen
;
;     MSCNT = 200 / cyclus-tijd - 2     waarbij cyclus-tijd  de clock-
;             frequentie van de gebruikte computer is
;
;
;     Bijvoorbeeld : KIM, SYM en AIM hebben een klok-frequentie van 1 MHz
;                       en dus MSCNT = 198
;
;                    Apple heeft een klok-frequentie van 1.023 MHz
;                       en dus MSCNT = 202
;
; We moeten echter een correctie maken om de tijd van de laatste keer dat de
; wachtlus wordt doorlopen te corrigeren. Daardoor trekken we in de laatste
; lus 3 van MSCNT af om 1 milliseconde minus de tijd, nodig voor de lus, te
; wachten

;.   De tijd nodig voor de lus is :
;
;                  6 cycli --> jsr DELAY
;                  2 cycli --> cpy #0
;                  2 cycli --> beq EXIT   (neem aan, niet uitgevoerd)
;                  2 cycli --> nop
;                  2 cycli --> cpy #1
;                  3 cycli --> bne DELAYA  (wel uitgevoerd)
;                  2 cycli --> dey
;                 -1 cyclus -->   de laatste bne DELAY1 wordt niet
;                                 uitgevoerd
;                  2 cycli --> ldx #MSCNT2
;                 -1 cyclus -->   de laatste bne DELAY2 wordt niet
;                                 uitgevoerd
;                  6 cycli --> rts
;              ---- --------------------   +
;                 25 cycli voor de laatste lus
; 1 MHz klok
;     MSCNT:       .EQU    0C6H          ; 198 maal door DELAY1

; We gaan uit van een Apple met een 1,023 MHz klok-frequentie
MSCNT:             .EQU    0CAH          ; 202 maal door DELAY1


DELAY:
                   cpy     #0            ; 2 cycli
                   beq     EXIT          ; 2 cycli (keer terug als
                                         ;  DELAY = 0)
```

```
                nop                     ; 2 cycli (om het totaal op
                                        ;  25 te brengen)

        ; Als DELAY = 1 milliseconde ga dan naar LAST1
        ;  Dit stukje programma kost 5 cycli of er wordt gesprongen
        ;   of niet
                cpy     #1              ; 2 cycli
                bne     DELAYA          ; 3 cycli (als er gesprongen
                                        ;  moet worden, anders 2)
                jmp     LAST1           ; 3 cycli

        ; Wacht 1 maal (Y - 1)  milliseconden
DELAYA:
                dey                     ; 2 cycli (verlaag bijvoorbaat
                                        ; al Y)
DELAY0:
                ldx     #MSCNT          ; 2 cycli
DELAY1:
                dex                     ; 2 cycli
                bne     DELAY1          ; 3 cycli
                nop                     ; 2 cycli
                nop                     ; 2 cycli
                dey                     ; 2 cycli
                bne     DELAY0          ; 3 cycli

LAST1:
        ; Wacht de laatste keer 25 cycli minder dan nodig is om
        ;  de aanroep, terugkeer en routinetijd te compenseren
                ldx     #MSCNT-3        ; 2 cycli
DELAY2:
                dex                     ; 2 cycli
                bne     DELAY2          ; 3 cycli
EXIT:
                rts                     ; 6 cycli


;
;
; Voorbeeld uitvoering
;
;

SC1008:
        ; Wacht 10 seconden
        ;  Eigenlijk: wacht 40 maal  250 miliseconden
                lda     #40
                sta     COUNT

        ; Wacht 1/4 seconde
QTRSCD:
                ldy     #250            ; 250 milliseconden
                jsr     DELAY
                dec     COUNT
                bne     QTRSCD

                brk                     ; Stop na 10 seconden
                jmp     SC1008

; Data
COUNT:          .BYTE   0

                .END
```

Interrupt-gestuurde invoer/uitvoer zonder buffer voor een 6850 ACIA - 11A

Voert interrupt-gestuurde invoer en uitvoer uit op een 6850 ACIA en enkelvoudige-karakter invoer- en uitvoer-buffers. Bestaat uit de volgende subroutines:

1. INCH leest een karakter uit de invoer-buffer
2. INST bepaalt of er een karakter in de invoer-buffer beschikbaar is
3. OUTCH schrijft een karakter naar de uitvoer-buffer
4. OUTST bepaalt of de uitvoer-buffer vol is
5. INIT initialiseert de 6850 ACIA, de interrupt-vectoren en de software-vlaggen (die worden gebruikt om data tussen het hoofdprogramma en de interrupt-afhandelingsroutine te verplaatsen)
6. IOSRVC bepaalt welke interrupt er werd gegeven en voorziet in de bijbehorende invoer- of uitvoer-service. Als reactie op een invoer-interrupt, leest de routine het karakter uit de ACIA en stopt het in de invoer-buffer. Als reactie op een uitvoer-interrupt, schrijft de routine het karakter van de uitvoer-buffer in de ACIA.

De voorbeelden beschrijven een 6850 ACIA op een Apple // serieel I/O-bord in slot 1.

Procedure:

1. INCH wacht tot er een karakter beschikbaar is, wist de Data Ready vlag (RECDF) en laadt het karakter in de accumulator.
2. INST zet de overdrachtvlag vanuit de Data Ready vlag (geheugenlocatie RECDF).
3. OUTCH wacht tot de karakter-buffer leeg is, plaatst het karakter in de buffer en zet de Character Available (Karakter Beschikbaar) vlag (TRNDF).
4. OUTST zet de overdrachtvlag volgens de Character Available vlag (geheugenlocatie TRNDF).
5. INIT wist de software-vlaggen, zet de interrupt-vector op, reset de ACIA (een hoofd-reset, omdat de ACIA geen reset-invoer heeft) en initialiseert de ACIA door de correcte waarde in het controleregister te stoppen (invoer-interrupts ingeschakeld, uitvoer-interrupts uitgeschakeld).
6. IOSRVC bepaalt of de gegeven interrupt een invoer-interrupt (bit 0 van het ACIA statusregister = 1), een uitvoer-interrupt (bit 1 van het ACIA statusregister = 1) of het produkt van een ander apparaat was. Als de invoer-interrupt werd gegeven, leest het programma de data, slaat ze in het geheugen op en zet de Data Ready (Data Klaar) vlag (RECDF). Als de uitvoer-interrupt werd gegeven, kijkt het programma of er een karakter beschikbaar is. Zo niet, dan schakelt het programma de uitvoer-interrupt eenvoudigweg uit. Als er wel een karakter beschikbaar is, stuurt het programma dit naar de ACIA, wist de Character Available vlag (TRNDF) en schakelt zowel de in- als de uitvoer-interrupt weer in.

Het enige bijzondere probleem bij deze routines doet zich voor als er een uitvoer-interrupt plaatsvindt terwijl er geen karakter beschikbaar is. We kunnen de interrupt niet negeren, anders blijft deze zichzelf onophoudelijk aandienen. Er ontstaat dan een oneindige lus. De oplossing ligt in het uitschakelen van de uitvoer-interrupts. Maar nu creeren we een nieuw probleem voor als er een karakter klaar is om verzonden te worden. Als we namelijk de uitvoer-interrupts uitgeschakeld hebben, kunnen we niet aan de hand van een interrupt vaststellen of de ACIA klaar is om te verzenden. Ook dit probleem kunnen we oplossen,namelijk door een extra, niet-interrupt-gestuurde ingang te creeren naar de routine die een karakter naar de ACIA stuurt. Omdat deze ingang niet wordt veroorzaakt door een interrupt, moet gecontroleerd worden of het uitvoer-register van de ACIA werkelijk leeg is, voordat het karakter verstuurd kan worden.

De bijzondere reeks van handelingen is als volgt:

1. Er vindt een uitvoer-interrupt plaats, voordat een nieuwe karakter beschikbaar is (ofwel, de ACIA is nu gereed voor data). De reactie hierop is de uitvoer-interrupt uitschakelen, aangezien er nog geen karakter te versturen is. Merk op dat deze reeks niet in het begin voorkomt, omdat INIT de uitvoer-interrupt uitschakelt. Anders zou de uitvoer-interrupt meteen gegeven worden, omdat de ACIA duidelijk leeg begint en daarom klaar is om data te versturen.

2. Uitvoer-karakter is beschikbaar. Dit betekent dat het systeem nu een karakter heeft dat verstuurd kan worden. Maar gaan zitten wachten op een uitvoer-interrupt heeft geen zin, omdat deze uitgeschakeld is.

3. Het hoofdprogramma roept de routine (OUTDAT) aan die de data naar de ACIA stuurt. Bij het controleren van het ACIA-statusregister blijkt dat de ACIA inderdaad klaar is om een karakter te versturen (we weten dat de ACIA klaar is omdat we de uitvoer-interrupt ontvingen). De routine stuurt dan het karakter en schakelt de interrupts weer in.

Het voornaamste probleem hierbij is dat het uitvoer-apparaat afhandeling kan eisen voordat de computer er klaar voor is. Anders gezegd: de apparaten kunnen data ontvangen, maar de computer heeft niets om te versturen. In het bijzonder is er een initialisatieprobleem dat wordt veroorzaakt door uitvoer-interrupts die blijven 'aandringen' en om afhandeling vragen. Invoer-apparaten daarentegen vragen alleen afhandeling als zij data hebben. Zij starten op in de 'not ready' (niet klaar) stand. Anders gezegd: het invoer-apparaat heeft bij de opstart geen data, terwijl de computer klaar is om data te ontvangen. Output apparaten zorgen daarom voor meer initialisatie en ordeningsproblemen dan invoer-apparaten.
De oplossing kan echter tot een merkwaardige situatie leiden. Laten we aannemen dat het systeem enige data klaar heeft voor uitvoer-maar de ACIA er nog niet klaar voor is. Het systeem moet dan, met uitgeschakelde interrupts, wachten tot de ACIA klaar is. Dus met andere woorden een interrupt-gestuurde routine moet de interrupts uitschakelen en almaar wachten terwijl het uitvoer-apparaat in de gaten wordt gehouden. Dit nadeel kunnen we ondervangen door een software-vlag bij te houden. Deze vlag wordt omgezet als de uitvoer-interrupt werd gegeven op een moment dat er geen karakter was. Dan zou het systeem de software-vlag kunnen controleren en vaststellen of er al een uitvoer-interrupt is gegeven (zie subroutine 11C).

Gebruikte registers:

1. INCH: A, F, Y
2. INST: A, F
3. OUTCH: A, F, Y
4. OUTST: A, F
5. INIT: A, F

Looptijd:

1. INCH: 33 cycli als er een karakter beschikbaar is
2. INST: 12 cycli
3. OUTCH: 92 cycli als de uitvoer-buffer leeg is en de ACIA klaar is om data te versturen
4. OUTST 12 cycli
5. INIT 73 cycli
6. IOSRVC 39 cycli om een invoer-interrupt af te handelen, 59 cycli om een uitvoer-interrupt af te handelen,

24 cycli om vast te stellen dat interrupt van een ander apparaat kwam

Programmalengte: 168 bytes

Benodigd datageheugen: Zes bytes ergens in RAM. Een byte voor de ontvangen data (op adres RECDAT), een byte voor de data-ontvangstvlag (op adres RECDF), een byte voor de te versturen data (een byte op adres TRNDAT), een byte voor de data-verstuurd vlag (op adres TRNDF) en twee bytes voor het adres van de volgende interrupt-afhandelingsroutine (vanaf adres NEXTSR).

Begincondities

1. INCH: Geen
2. INST: Geen
3. OUTCH: Te versturen karakter in accumulator
4. OUTST: Geen
5. INIT: Geen

Eindcondities

1. INCH: Karakter in accumulator
2. INST: Overdrachtvlag = 0 als geen karakter beschikbaar is; 1 als dit wel het geval is.
3. OUTCH: Geen
4. OUTST: Overdrachtvlag = 0 als uitvoer-buffer leeg is; 1 als de buffer vol is.
5. INIT: Geen

```
;       Titel :         Eenvoudige, interrupt-gestuurde, in- en uitvoer met
;                       behulp van een 6850 ACIA en een buffer van 1 karakter
;
;       Naam :          SINTIO
;
;
;       Doel :          Dit programma bestaat uit vijf subroutines die samen voor
;                       interrupt-gestuurde in- en uitvoer zorgen, met behulp van
;                       een 6850 ACIA
;
;                       INCH
;                               Lees een karakter
;
;                       INST
;                               Bepaal de invoer-status (kijk of de invoer-buffer
;                               leeg is)
;
;                       OUTCH
;                               Schrijf een karakter
;
;                       OUTST
;                               Bepaal de uitvoer-status (kijk of de uitvoer-
;                               buffer leeg is)
;
```

```
;                   INIT
;                           Initialiseer
;
;
;     Begin :
;                   INCH
;                           Geen parameters
;
;                   INST
;                           Geen parameters
;
;                   OUTCH
;                           Register A = uit te sturen karakter
;
;                   OUTST
;                           Geen parameters
;
;                   INIT
;                           Geen parameters
;
;
;     Einde :
;                   INCH
;                           Register A = karakter
;
;                   INST
;                           Overdrachtvlag = 0 als de invoer-buffer leeg is
;                           Overdrachtvlag = 1 als er een karakter in de
                            invoer buffer zit
;
;
;                   OUTCH
;                           Geen parameters
;
;                   OUTST
;                           Overdrachtvlag = 0 als de uitvoer-buffer leeg is

;                           Overdrachtvlag = 1 als er een karakter in de
                            uitvoer buffer zit
;
;
;                   INIT
;                           Geen parameters
;
;
;     Registers :
;                   INCH
;                           A, F, Y
;
;                   INST
;                           A, F
;
;                   OUTCH
;                           A, F, Y
;
;                   OUTST
;                           A, F
;
;                   INIT
;                           A, F
;
```

```
;
;     Tijd :
;                    INCH
;                            33 cycli als er een karakter aanwezig is
;
;                    INST
;                            12 cycli
;
;                    OUTCH
;                            92 cycli als de invoer-buffer leeg is en de ACIA
;                            klaar is om een karakter te versturen
;
;                    OUTST
;                            12 cycli
;
;                    INIT
;                            73 cycli
;
;                    IOSRVC
;                            24 cycli minimaal, als de interrupt niet door ons
;                             gegenereerd is
;                            39 cycli om een invoer-interrupt af te handelen
;                            59 cycli om een uitvoer-interrupt af te handelen
;
;     Grootte :
;                    Programma 168 bytes
;                    Data      6   bytes
;

; Voorbeeld van 6850 ACIA definities voor een Apple serieele kaart in slot 1
ACIASR:             .EQU    0C094H        ; ACIA statusregister
ACIADR:             .EQU    0C095H        ; ACIA dataregister
ACIACR:             .EQU    0C094H        ; ACIA controleregister
IRQVEC:             .EQU    03FEH         ; Apple IRQ vectoradres

; Lees een karakter
INCH:
                    jsr     INST          ; Haal de input-status op
                    bcc     INCH          ; Wacht als er geen karakter

                                          ; in de invoer-buffer zit
                    php                   ; Bewaar de huidige status van
                                          ; de interrupts
                    sei                   ; Zet de interrupts af
                    lda     #0
                    sta     RECDF         ; Geef aan dat de buffer nu leeg is
                    lda     RECDAT        ; Haal het karakter uit de buffer
                    plp                   ; Zet de interruptvlag terug
                    rts

; Keer terug met de invoer-status (overdrachtvlag is 1 als er een karakter
; in de invoer-buffer aanwezig is)

INST:
                    lda     RECDF         ; Haal de waarde van de data-
                                          ; aanwezig vlag
                    lsr     A             ; Zet de overdrachtvlag met
                                          ; behulp van de vlag
                    rts

; Schrijf een karakter
```

```
OUTCH:
                    php                     ; Haal de interrupt-status op
                    pha                     ; Bewaar het uit te sturen
                                            ; karakter

          ; Wacht tot de uitvoer-buffer leeg is, en zet dan het
          ; nieuwe karakter in de buffer
WAITOC:
                    jsr     OUTST           ; Haal de uitvoer-status op
                    bcs     WAITOC          ; Wacht tot de buffer leeg is
                    sei                     ; Zet de interrupts af
                    pla                     ; Haal het nieuwe karakter op
                    sta     TRNDAT          ; Zet het in de buffer
                    lda     #0FFH           ; Geef aan dat de buffer gevult is
                    sta     TRNDF
                    jsr     OUTDAT          ; Stuur het karakter naar de
                                            ; uitvoerpoort
                    plp                     ; Zet de interruptvlag terug
                    rts

; Uitvoer-status (overdrachtvlag = 1 als de buffer vol is)

OUTST:
                    lda     TRNDF
                    lsr     A
                    rts

; Initialiseer
INIT:
                    php                     ; Bewaar de huidig waarde van
                                            ; de interruptvlag
                    sei                     ; Zet de interrupt uit tijdens
                                            ; het initializeren

          ; Initialiseer de softwarevlaggen
                    lda     #0
                    sta     RECDF           ; Geen data in de buffer aan- wezig
                    sta     TRNDF           ; Uitvoer-buffer is leeg

          ; Bewaar de huidige IRQ-vector in NEXTSR
                    lda     IRQVEC
                    sta     NEXTSR
                    lda     IRQVEC+1
                    sta     NEXTSR+1

          ; Laat de IRQ-vector naar onze invoer-afhandelroutine wijzen
                    lda     AIOS
                    sta     IRQVEC
                    lda     AIOS+1
                    sta     IRQVEC+1

          ; Initialiseer de 6850
                    lda     #011B
                    sta     ACIACR          ; Reset de ACIA
                    lda     #10010001B
                    sta     ACIACR          ; Initialiseer de ACIA modus tot
                                            ;
                                            ;    deel door 16
                                            ;    8 databits
                                            ;    2 stopbits
                                            ;    geen uitvoer-interrupts!
                                            ;    wel invoer-interrupts
```

```
                plp                     ; Zet de interruptvlag terug
                rts

AIOS:           .WORD   IOSRVC          ; Adres van de invoer-afhandel-
                                        ; routine

IOSRVC:
                pha                     ; Bewaar de inhoud van register A
                cld                     ; Zet de processor in binaire mode

        ; Haal de status van de ACIA op
        ;       Bit 0 = 1 als het een invoer-interrupt was
        ;       Bit 1 = 1 als het een outvoer-interrupt was
                lda     ACIASR
                lsr     A               ; Bit 0 naar de overdrachtvlag
                bcs     IINT            ; Spring als het een invoer-
                                        ; interrupt is
                lsr     A               ; Bit 1 naar de overdrachtvlag
                bcs     OINT            ; Spring als het een uitvoer-
                                        ; interrupt is

        ; Deze interrupt is niet gegenereerd door de ACIA
                pla
                jmp     (NEXTSR)        ; Ga naar de volgende
                                        ; afhandel routine

; Handel een invoer-interrupt af
IINT:
                lda     ACIADR          ; Haal het karakter op
                sta     RECDAT          ; Zet het in de buffer
                lda     #0FFH           ; Geef aan dat er een karakter
                                        ; in de buffer zit
                sta     RECDF
                jmp     EXIT            ; Keer terug naar de plaats
                                        ; waar we de interrupt kregen

; Handel een uitvoer-interrupt af
OINT:
                lda     TRNDF           ; Kijk of er een karakter
                                        ; klaar is om verzonden te
                                        ; worden
                beq     NODATA          ; Spring als er niets
                                        ; verzonden hoeft te worden
                jsr     OUTDT1          ; Verstuur het karakter
                jmp     EXIT            ; Keer terug naar de plaats
                                        ; waar we de interrupt kregen

; Als er een uitvoer-interrupt werd verstuurd zonder dat er een karakter in
;  de buffer aanwezig is, moeten we de uitvoer-interrupt afzetten om niet in
;  een oneindige lus te geraken.
;  Als er weer een karakter in de buffer aanwezig is, roepen we de uitvoer
;  routine, OUTDAT aan, die eerst de status van de ACIA  bekijkt voordat
;  hij het karakter mag versturen. De uitvoerroutine moet ook de uitvoer-
;  interrupt weer aanzetten nadat het karakter is verzonden. Dit heft het
;  probleem op dat een niet afgehandelde uitvoer-interrupt zichzelf
;  herhaaldelijk veroorzaakt, terwijl er nog steeds voor gezorgd wordt dat
;  uitvoer-interrupts worden herkend en dat het karakter niet wordt
;  verzonden naar een ACIA die niet klaar is om het karakter te versturen.
;  Het probleem hier is dus dat een uitvoer-medium een interrupt kan
;  genereeren terwijl de computer nog niets te versturen heeft. (Een invoer-
;  medium heeft al data als het een interrupt stuurt.)
```

```
NODATA:
                    lda     #10010001B   ; Zet de uitvoer-interrupt af,
                                         ;  zet de invoer-interrupt aan
                                         ;  8 databits
                                         ;  2 stopbits
                                         ;  deel door 16 klok
                    sta     ACIACR       ; Zet de uitvoer-interrupt af

EXIT:
                    pla                  ; Zet register A weer goed
                    rti                  ; Keer terug van de interrupt


;
;       Subroutine : OUTDAT, OUTDT1 (OUTDAT is het beginpunt voor een routine
;                    die niet door een interrupt werd aangeroepen)
;
;       Doel       : Stuur een karakter naar de ACIA
;
;       Begin      : TRNDAT = karakter dat verstuurd moet worden
;
;       Eind       : Geen
;
;       Registers  : A, F
;

; Beginpunt voor niet door interrupt gestuurde routines
; Wacht tot de ACIA klaar is om het karakter te versturen
OUTDAT:
                    lda     ACIASR
                    and     #00000010B   ; Test of het ACIA uitvoer-
                                         ; register leeg is
                    beq     OUTDAT       ; Spring als het niet leeg is

OUTDT1:
                    lda     TRNDAT       ; Haal het karakter op
                    sta     ACIADR       ; Verstuur het
                    lda     #0
                    sta     TRNDF        ; Geef aan dat de buffer leeg is
                    lda     #10110001B
                    sta     ACIACR       ; Zet zowel de ACIA invoer-

                                         ; als uitvoer-interrupts aan,
                                         ;  8 databits, 2 stopbits,
                                         ;  deel door 16 klok
                    rts

; Data
RECDAT:             .BLOCK  1            ; Invoer-buffer
RECDF:              .BLOCK  1            ; Invoer-buffer statusvlag
                                         ;  0 = geen karakter aanwezig
                                         ;  FFH = buffer vol
TRNDAT:             .BLOCK  1            ; Uitvoer-buffer
TRNDF:              .BLOCK  1            ; Uitvoer-buffer statusvlag
                                         ;  0 = geen karakter aanwezig
                                         ;  1 = buffer vol
NEXTSR:             .BLOCK  2            ; Adres van de volgende
                                         ; interrupt-afhandelroutine

;
;
; Voorbeeld uitvoering
;
```

```
SC1101:
                jsr     INIT            ; Initialiseer
                cli                     ; Zet de interrupts aan

        ; Simpel voorbeeldje
LOOP:
                jsr     INCH            ; Lees een karakter
                pha
                jsr     OUTCH           ; Stuur het weer uit
                pla
                cmp     #1BH            ; Is het een escape-karakter
                bne     LOOP            ; Spring terug als het geen
                                        ; escape-karakter is
                brk

        ; Een asynchroon voorbeeld
        ;  Stuur constant een 'A' naar de console, maar handel ook
        ;  alle invoer af (d.w.z. stuur de invoer ook naar de console).
ASYNLP:
        ; Als het uitvoer-medium niet bezig is, stuur er dan een
        ; 'A' naartoe
                jsr     OUTST           ; Is het uitvoer-medium bezig?
                bcs     ASYNLP          ; Spring als het bezig is
                lda     #"A"
                jsr     OUTCH           ; Verstuur het karakter

        ; Als er een karakter in de invoer-buffer zit, haal het dan op
                jsr     INST            ; Is er een karakter aanwezig?
                bcc     ASYNLP          ; Spring als er geen karakter
                                        ; In de buffer zit
                jsr     INCH            ; Lees een karakter
                cmp     #1BH            ; Is het een escape-karakter?
                beq     DONE            ; Spring als het een escape-
                                        ; karakter is
                jsr     OUTCH           ; Stuur het uit
                jmp     ASYNLP          ; en ga door

DONE:
                brk
                jmp     SC1101

                .END
```

Interrupt-gestuurde invoer/uitvoer zonder buffer voor een 6522 VIA - 11B

Voert interrupt-gestuurde invoer en uitvoer uit op een 6522 VIA en enkelvoudige-karakter in- en uitvoer-buffers. Bestaat uit de volgende subroutines:

1. INCH leest een karakter uit de invoer-buffer
2. INST bepaalt of er een karakter in de invoer-buffer beschikbaar is
3. OUTCH schrijft een karakter naar de uitvoer-buffer
4. OUTST bepaalt of de uitvoer-buffer vol is
5. INIT initialiseert de 6522 VIA, de interrupt vectoren en de software-vlaggen.
6. IOSRVC bepaalt welke interrupt er werd gegeven en voorziet in de bijbehorende invoer-of uitvoer-service (dat wil zegge, de routine leest een karakter uit de VIA en stopt dit karakter in de uitvoer-buffer in antwoord op een invoer-interrupt en schrijft een karakter uit de uitvoer-buffer in de VIA in antwoord op een uitvoer-interrupt).

De voorbeelden beschrijven een 6522 VIA die op een Apple // is aangesloten.

Procedure:

1. INCH wacht tot er een karakter in de invoer-buffer beschikbaar is, wist de Data Ready vlag (RECDF) en laadt het karakter in de accumulator.
2. INST zet de overdrachtvlag vanuit de Data Ready vlag (geheugenlocatie RECDF).
3. OUTCH wacht tot de uitvoer-buffer leeg is, plaatst het karakter (dat in de accumulator zit) in de buffer en zet de karakter-aanwezig (buffer vol) vlag (TRNDF). Als er een niet afgehandelde uitvoer-interrupt plaatsvond (ofwel het uitvoer-apparaat vroeg om afhandeling op een moment dat er geen data beschikbaar was), stuurt OUTCH de data feitelijk naar de VIA.
4. OUTST zet de overdrachtvlag volgens de karakter-aanwezig-vlag (geheugenlocatie TRNDF).
5. INIT wist de software -laggen, zet de interrupt-vector op en initialiseert de 6522 VIA. Het maakt van poort A een invoer-poort en van poort B een uitvoer-poort. Controlelijnen CA1 en CB2 worden laag-naar-hoog actief, controlelijn CA2 een korte uitvoer-puls om invoer-bevestiging aan te geven (actief-laag nadat de centrale processor de data heeft gelezen) en controlelijn CB2 wordt een schrijf-strobe (actief-laag nadat de centrale processor de data heeft geschreven, en blijft actief-laag tot de randapperatuur weer gereed is). INIT schakelt ook de invoer-interrupt op CA1 en de uitvoer-interrupt op CB1 uit.
6. IOSRVC bepaalt of de gegeven interrupt een invoer-interrupt (bit 1 van het VIA interrupt-vlagregister = 1), een uitvoer-interrupt (bit 4 van het VIA interrupt vlag register = 1) of het produkt van een ander apparaat was. Als de invoer-interrupt werd gegeven, leest het programma de data, slaat ze in de invoer-buffer op en zet de Data Ready (Data Klaar) vlag (RECDF). Als de uitvoer-interrupt werd gegeven, kijkt het programma of er een karakter beschikbaar is. Zo niet, dan schakelt het programma de uitvoer-interrupt eenvoudigweg uit en wist de vlag (OIE) die aangeeft dat het uitvoer-apparaat inderdaad klaar (m.a.w. er is een uitvoer-interrupt gegeven op een moment dat er geen katakter beschikbaar was). Als er wel een karakter beschikbaar is, stuurt het programma dit vanuit de uitvoer-buffer naar de VIA, wist de karakter-aanwezig-vlag (TRNDF), zet de Output Interrupt Vlag (OIE) en schakelt zowel de in- als de uitvoer-interrupt weer in.

Het enige bijzondere probleem bij deze routines doet zich voor als er een uitvoer-interrupt wordt geven terwijl er geen karakter aanwezig is. We kunnen de interrupt niet negeren, anders blijft deze zichzelf onophoudelijk aandienen. Er ontstaat dan een oneindige lus. De oplossing hiervoor is

eenvoudigweg de interrupt te neutraliseren door het dataregister in poort B te lezen. Maar nu creeeren we een nieuw probleem voor als het hoofdprogramma data klaar heeft om te laten verzenden. De interrupt die aangeeft dat het uitvoer-apparaat klaar is, is al gegeven (en gewist), dus heeft het geen zin daar op te blijven wachten. De oplossing hiervoor is een extra vlag die (met een 0) aangeeft dat de uitvoer-interrupt is gegeven zonder afgehandeld te zijn. Wij noemen dit de OIE-vlag, output-inputerrupt-vlag. De initialisatier-routine zet deze eerst (aangezien het uitvoer-apparaat niet om afhandeling heeft gevraagd) en de uitvoer-afhandelingsroutine wist de vlag als er een uitvoer-interrupt plaatsvindt die niet afgehandeld kan worden (geen data beschikbaar). De routine zet de vlag nadat het data naar de VIA heeft gestuurd (voor het geval de vlag gewist mocht zijn). Nu kan de uitvoer-routine OUTCH de OIE controleren om vast te stellen of er al een interrupt is gegeven (0 als dit het geval is, FF_{16} indien niet).
Merk op dat we een VIA-interrupt kunnen wissen zonder feitelijk data te versturen. Dit kan niet met de 6850 ACIA (zie de subroutines 11A en 11C), dus verschillen de handelingen enigszins. Dit probleem van niet afgehandelde interrupts doet zich alleen voor bij uitvoer-apparaten, aangezien invoer-apparaten alleen afhandeling vragen als zij data klaar hebben om af te geven.

Gebruikte registers:

1. INCH A, F, Y
2. INST A, F
3. OUTCH A, F, Y
4. INIT A, F

Looptijd:

1. INCH 33 cycli als er een karakter beschikbaar is
2. INST 12 cycli
3. OUTCH 83 cycli als de uitvoer-buffer leeg is en de VIA klaar is om data te versturen
4. OUTST 12 cycli
5. INIT 93 cycli
6. IOSRVC 43 cycli om een invoer-interrupt af te handelen, 81 cycli om een uitvoer-interrupt af te handelen, 24 cycli om vast te stellen dat interrupt van een ander apparaat kwam

Programmalengte: 194 bytes

Benodigd datageheugen: Zeven bytes ergens in RAM. Een byte voor de ontvangen data (op adres RECDAT), een byte voor de data-ontvangst-vlag (op adres RECDF), een byte voor de te versturen data (op adres TRNDAT), een byte voor de data-verzonden-vlag (op adres TRNDF), een byte voor de output-interrupt-vlag (op adres OIE) en twee bytes voor het adres van de volgende interrupt-afhandelingsroutine (vanaf adres NEXTSR).

Begincondities

1. INCH: Geen
2. INST: Geen
3. OUTCH: Te versturen karakter in accumulator
4. OUTST: Geen
5. INIT: Geen

Eindcondities

1. INCH: Karakter in accumulator
2. INST: Overdrachtvlag = 0 als geen karakter beschikbaar is, 1 als dit wel het geval is.
3. OUTCH: Geen
4. OUTST: Overdrachtvlag = 0 als uitvoer-buffer leeg is, 1 als de buffer vol is.
5. INIT: Geen

```
;       Titel :         Eenvoudige, interrupt-gestuurde, in- en uitvoer met
;                       behulp van een 6522 VIA en een buffer van een karakter
;
;       Naam :          PINTIO
;
;
;       Doel :          Dit programma bestaat uit vijf subroutines die samen voor
;                       interrupt-gestuurde in- en uitvoer zorgen, met behulp van
;                       een 6522 VIA
;
;                       INCH
;                               Lees een karakter
;
;                       INST
;                               Bepaal de invoer-status (kijk of de invoer-buffer
;                               leeg is)
;
;                       OUTCH
;                               Schrijf een karakter
;
;                       OUTST
;                               Bepaal de uitvoer-status (kijk of de uitvoer-
;                               buffer leeg is)
;
;                       INIT
;                               Initialiseer
;
;
;       Begin :
;                       INCH
;                               Geen parameters
;
;                       INST
;                               Geen parameters
;
;                       OUTCH
;                               Register A = uit te sturen karakter
;
;                       OUTST
;                               Geen parameters
;
;                       INIT
;                               Geen parameters
;
;
;       Einde :
;                       INCH
;                               Register A = karakter
;
```

```
;                   INST
;                           Overdrachtvlag = 0 als de invoer-buffer leeg is
;                           Overdrachtvlag = 1 als er een karakter in de
;                                              invoer-buffer zit
;
;
;                   OUTCH
;                           Geen parameters
;
;                   OUTST

;                           Overdrachtvlag = 0 als de uitvoer-buffer leeg is
;                           Overdrachtvlag = 1 als er een karakter in de
;                                              uitvoer-buffer zit
;
;                   INIT
;                           Geen parameters
;
;
;     Registers :
;                   INCH
;                           A, F, Y
;
;                   INST
;                           A, F
;
;                   OUTCH
;                           A, F, Y
;
;                   OUTST
;                           A, F
;
;                   INIT
;                           A, F
;
;
;     Tijd :
;                   INCH
;                           33 cycli als er een karakter aanwezig is
;
;                   INST
;                           12 cycli
;
;                   OUTCH
;                           83 cycli als de invoer-buffer leeg is en de VIA
;                           klaar is om een karakter te versturen
;
;                   OUTST
;                           12 cycli
;
;                   INIT
;                           93 cycli
;
;                   IOSRVC
;                           24 cycli minimaal, als de interrupt niet door ons
;                            gegenereerd is
;                           43 cycli om een invoer-interrupt af te handelen
;                           81 cycli om een uitvoer-interrupt af te handelen
;
;     Grootte :
;                   Programma 194 bytes
;                   Data        7 bytes
;
```

```
; Voorbeeld van 6522 VIA definities
VIA:            .EQU    0C090H      ; Begin adres VIA
VIABDR:         .EQU    VIA         ; VIA poort B dataregister
VIAADR:         .EQU    VIA+1       ; VIA poort A dataregister,
                                    ;  met handshaking
VIABDD:         .EQU    VIA+2       ; VIA poort B richtings-
                                    ;  register
VIAADD:         .EQU    VIA+3       ; VIA poort A richtings-
                                    ;  register
VIAACR:         .EQU    VIA+11      ; VIA hulpcontroleregister

VIAPCR:         .EQU    VIA+12      ; VIA randapparatuur controle-
                                    ;  register
VIAIFR:         .EQU    VIA+13      ; VIA interruptvlag-register
VIAIER:         .EQU    VIA+14      ; Register om VIA interrupts
                                    ;  aan te zetten

IRQVEC:         .EQU    03FEH       ; Apple IRQ vectoradres

; Lees een karakter
INCH:
                jsr     INST        ; Haal de input-status op
                bcc     INCH        ; Wacht als er geen karakter
                                    ; in de invoer-buffer zit
                php                 ; Bewaar de huidige status van
                                    ; de interrupts
                sei                 ; Zet de interrupts af
                lda     RECDAT      ; Haal het karakter uit de buffer
                lda     #0
                sta     RECDF       ; Geef aan dat de buffer nu leeg is
                lda     RECDAT      ; Haal het karakter uit de buffer
                plp                 ; Zet de interruptvlag terug
                rts

; Keer terug met de invoer-status (overdrachtvlag is 1 als er een karakter
; in de invoer-buffer aanwezig is

INST:
                lda     RECDF       ; Haal de waarde van de data-
                                    ; aanwezig vlag
                lsr     A           ; Zet de overdrachtvlag met
                                    ; behulp van de vlag
                rts

; Schrijf een karakter
OUTCH:
                php                 ; Haal de interrupt-status op
                pha                 ; Bewaar het uit te sturen karakter

        ; Wacht tot de uitvoer-buffer leeg is, en zet dan het
        ; nieuwe karakter in de buffer
WAITOC:
                jsr     OUTST       ; Haal de uitvoer-status op
                bcs     WAITOC      ; Wacht tot de buffer leeg is
                sei                 ; Zet de interrupts af
                pla                 ; Haal het nieuwe karakter op
                sta     TRNDAT      ; Zet het in de buffer
                lda     #0FFH       ; Geef aan dat de buffer gevuld is
                sta     TRNDF
                lda     OIE         ; Heeft het uitvoer-medium al
                                    ;  om aandacht gevraagd?
                bne     OUTCH1      ; Nee, wacht op een interrupt
```

```
                jsr     OUTDAT          ; Stuur het karakter naar de
                                        ; uitvoerpoort
OUTCH1:
                plp                     ; Zet de interruptvlag terug
                rts

; Uitvoer-status (overdrachtvlag = 1 als de buffer vol is)
OUTST:
                lda     TRNDF
                lsr     A
                rts

; Initialiseer
INIT:
                php                     ; Bewaar de huidig waarde van
                                        ; de interruptvlag
                sei                     ; Zet de interrupt uit tijdens
                                        ; het initializeren

        ; Initialiseer de softwarevlaggen
                lda     #0
                sta     RECDF           ; Geen data in de buffer aanwezig
                sta     TRNDF           ; Uitvoer-buffer is leeg
                lda     #0FFH           ; Uitvoer-medium heeft nog
                                        ;  niet om aandacht gevraagd
                sta     IOE

        ; Bewaar de huidige IRQ-vector in NEXTSR
                lda     IRQVEC
                sta     NEXTSR
                lda     IRQVEC+1
                sta     NEXTSR+1

        ; Laat de IRQ-vector naar onze invoer-afhandelroutine wijzen
                lda     AIOS
                sta     IRQVEC
                lda     AIOS+1
                sta     IRQVEC+1

        ; Initialiseer de 6522 VIA
                lda     #00000000B
                sta     VIAADD          ; Poort A wordt invoer-poort
                lda     #11111111B
                sta     VIABDD          ; Poort B wordt uitvoer-poort
                lda     #10001010B
                sta     VIAPCR          ; Zet poort A op
                                        ;  Interrupt als CA1 van laag
                                        ;   naar hoog gaat (bit 0 = 1)
                                        ;  Stuur een lage puls naar
                                        ;   CA2 (bits 1..3 = 101)
                                        ; Zet poort B op
                                        ;  Interrupt als CB1 van laag
                                        ;   naar hoog gaat (bit 4 = 1)
                                        ;  Handshake-uitvoermodus
                                        ;   (bits 5..7 = 001)
                lda     #00000001B
                sta     VIAACR          ; Zet het hulpregister goed
                                        ;  om invoer-latching op poort
                                        ;  A aan te zetten
                lda     #00010010B
                sta     VIAIER          ; Zet het interrupt-register
                                        ;  om interrupts op CA1
                                        ;  (bit 1) en CB1 (bit 4)
```

```
                                        ;  goed te vinden
                plp                     ; Zet de interruptvlag terug
                rts

AIOS:           .WORD   IOSRVC          ; Adres van de invoer-afhandel-
                                        ; routine

IOSRVC:
                pha                     ; Bewaar de inhoud van register A
                cld                     ; Zet de processor in binaire mode

        ; Haal de status van de VIA op

        ;       Bit 1 = 1 als het een invoer-interrupt was
        ;       Bit 4 = 1 als het een outvoer-interrupt was
                lda     VIAIFR
                and     #10B            ; Test bit 1
                bne     IINT            ; Spring als het een invoer -
                                        ; interrupt is
                lda     VIAIFR
                and     #1000B          ; Test bit 4
                bne     OINT            ; Spring als het een uitvoer-
                                        ; interrupt is

        ; Deze interrupt is niet gegenereerd door de VIA
                pla
                jmp     (NEXTSR)        ; Ga naar de volgende afhandel-
                                        ; routine

; Handel een invoer-interrupt af
IINT:
                lda     VIAADR          ; Haal het karakter op
                                        ;  (Stuur pulsen naar CA2
                                        ;  om om handshaking te vragen
                                        ;  en zet de interruptvlag)
                sta     RECDAT          ; Zet het in de buffer
                lda     #0FFH           ; Geef aan dat er een karakter
                                        ; in de buffer zit
                sta     RECDF
                jmp     EXIT            ; Keer terug naar de plaats
                                        ; waar we de interrupt kregen

; Handel de uitvoer-interrupt af
; We kunnen een 6522 interrupt wissen door het data register te lezen.
; Daarmee kunnen we dus een uitvoer-interrupt wissen, zonder de interrupt te
; behandelen of uit te zetten.  Als we dit echter doen, moeten we een vlag
; hebben (OIE) die aangeeft dat er een uitvoer-interrupt is geweest, maar
; nog niet verwerkt.  OUTCH kan dan de OIE-vlag gebruiken om te bepalen of
; hij moet wachten om een uitvoer-interrupt te sturen, of dat hij meteen
; kan beginnen met versturen.

OINT:
                lda     TRNDF           ; Kijk of er een karakter
                                        ; klaar is om verzonden te worden
                beq     NODATA          ; Spring als er niets
                                        ; verzonden hoeft te worden
                jsr     OUTDAT          ; Verstuur het karakter
                jmp     EXIT            ; Keer terug naar de plaats
                                        ; waar we de interrupt kregen

NODATA:
                lda     VIABDR          ; Lees het dataregister van
                                        ;  poort B om de interrupt te wissen
```

```
                    lda     #0              ; Geef aan dat er een
                                            ;  interrupt is geweest die
                                            ;  nog niet verder afgehandeld is
                    sta     OIE
EXIT:
                    pla                     ; Zet register A weer goed
                    rti                     ; Keer terug van de interrupt


;
;       Subroutine : OUTDAT
;
;       Doel       : Stuur een karakter naar de VIA
;
;       Begin      : TRNDAT = karakter dat verstuurd moet worden
;
;       Eind       : Geen
;
;       Registers  : A, F
;

OUTDAT:
                    lda     TRNDAT          ; Haal het karakter op
                    sta     VIABDR          ; Verstuur het
                    lda     #0
                    sta     TRNDF           ; Geef aan dat de buffer leeg is
                    lda     #OFFH
                    sta     OIE             ; Geef aan dat er nog wat af-
                                            ;  gehandeld moet worden
                    rts

; Data
RECDAT:             .BLOCK  1               ; Invoer-buffer
RECDF:              .BLOCK  1               ; Invoer-buffer-statusvlag
                                            ;  0 = geen karakter aanwezig
                                            ;  FFH = buffer vol
TRNDAT:             .BLOCK  1               ; Uitvoer-buffer
TRNDF:              .BLOCK  1               ; Uitvoer-buffer statusvlag
                                            ;  0 = geen karakter aanwezig
                                            ;  1 = buffer vol
; Data
RECDAT:             .BLOCK  1               ; Invoer-buffer
RECDF:              .BLOCK  1               ; Invoer-buffer-statusvlag
                                            ;  0 = geen karakter aanwezig
                                            ;  FFH = buffer vol
TRNDAT:             .BLOCK  1               ; Uitvoer-buffer
TRNDF:              .BLOCK  1               ; Uitvoer-buffer statusvlag
                                            ;    0 = geen karakter aanwezig
                                            ;  FFH = buffer vol
OIE:                                        ; Uitvoer-interruptvlag
                                            ;    0 = interrupt is nog niet
                                            ;        afgehandeld
                                            ;  FFH = interrupt afgehandeld
NEXTSR:             .BLOCK  2               ; Adres van de volgende
                                            ; interrupt-afhandelroutine

;
;
; Voorbeeld uitvoering
;
;
```

```
SC1102:
                jsr     INIT            ; Initialiseer
                cli                     ; Zet de interrupts aan

        ; Simpel voorbeeldje
LOOP:
                jsr     INCH            ; Lees een karakter
                pha
                jsr     OUTCH           ; Stuur het weer uit
                pla
                cmp     #1BH            ; Is het een escape-karakter
                bne     LOOP            ; Spring terug als het geen
                                        ; escape-karakter is
                brk

        ; Een asynchroon voorbeeld
        ;  Stuur constant een 'A' naar de console, maar handel ook
        ;  alle invoer af (d.w.z. stuur de invoer ook naar de
        ;  console).
ASYNLP:
        ; Als het uitvoer-medium niet bezig is, stuur er dan een
        ; 'A' naartoe
                jsr     OUTST           ; Is het uitvoer-medium bezig?
                bcs     ASYNLP          ; Spring als het bezig is
                lda     #"A"
                jsr     OUTCH           ; Verstuur het karakter

        ; Als er een karakter in de invoer-buffer zit, haal het dan op
                jsr     INST            ; Is er een karakter aanwezig?
                bcc     ASYNLP          ; Spring als er geen karakter
                                        ; In de buffer zit
                jsr     INCH            ; Lees een karakter
                cmp     #1BH            ; Is het een escape-karakter?
                beq     DONE            ; Spring als het een escape-
                                        ; karakter is
                jsr     OUTCH           ; Stuur het weer uit
                jmp     ASYNLP          ; en ga door

DONE:
                brk
                jmp     SC1102

                .END
```

Interrupt-gestuurde invoer/uitvoer met buffer voor een 6850 ACIA - 11C

Voert interrupt-gestuurde in- en uitvoer- uit op een 6850 ACIA en meervoudige-karakterbuffers. Bestaat uit de volgende subroutines:

1. INCH leest een karakter uit de invoer-buffer
2. INST bepaalt of er een karakter in de invoer-buffer beschikbaar is
3. OUTCH schrijft een karakter naar de uitvoer-buffer
4. OUTST bepaalt of de uitvoer-buffer vol is
5. INIT initialiseert de 6850 ACIA en de buffers.
6. IOSRVC bepaalt welke interrupt er werd gegeven en voorziet in de bijbehorende invoer- of uitvoer-service.

Procedure:

1. INCH wacht tot er een karakter beschikbaar is, haalt het karakter uit de kop van de invoer-buffer, verplaatst de kop van de invoer-buffer een plaats omhoog en verlaagt de invoer-bufferteller met 1.
2. INST zet deoverdrachtvlag op 0 als de invoer-bufferteller nul is en op 1 als deze niet-nul is.
3. OUTCH wacht tot er een lege plaats in de uitvoer-buffer is (d.w.z. tot de uitvoer-buffer niet vol is), plaatst het karakter in de staart van de uitvoer-buffer, verplaatst de staart van de buffer een positie omhoog en verhoogt de uitvoer-bufferteller met 1.
4. OUTST zet de overdrachtvlag op 1 als de uitvoer-buffer teller gelijk is aan de lengte van de buffer, en op 0 als dit niet zo is.
5. INIT wist de buffertellers, zet zowel de koppen als de staarten van de buffers op nul, zet de interrupt-vector op, reset de ACIA door een hoofd-reset op het controleregister uit te voeren (de ACIA heeft geen reset-invoer) en plaatst de ACIA in de vereiste werkstand door de juiste waarde in het controleregister op te slaan. INIT schakelt de invoer-interrupts in en de uitvoer-interrupts uit. Het wist echter wel de uitvoer-interrupt- inschakelvlag, waarmee wordt aangegeven dat de ACIA klaar is om data te versturen, ofschoon het geen uitvoer-interrupt kan geven.
6. IOSRVC bepaalt of de gegeven interrupt een invoer-interrupt (bit 0 van het ACIA status register = 1), een uitvoer-interrupt (bit 1 van het ACIA-statusregister = 1) of het produkt van een ander apparaat was. Als de invoer-interrupt werd gegeven, leest het programma de data en stelt vast of er ruimte voor is in de buffer. Is er ruimte, dan slaat de processor het karakter op in de staart van de invoer-buffer, verplaatst de staart van de buffer een positie naar boven en verhoogt de invoer-bufferteller met 1. Als de uitvoer-interrupt werd gegeven, kijkt het programma of er data in de uitvoer-buffer zijn. Zo niet, dan schakelt het programma de uitvoer-interrupt uit (zodat de interrupt niet onophoudelijk wordt gegeven) en wist een uitvoer- interrupt-vlag die aangeeft dat de ACIA klaar is, omdat de hoofdroutine dit niet aan de interrupt kan zien. Als er data in de uitvoer-buffer beschikbaar zijn, haalt het programma een karakter uit de kop van de buffer, zendt dit naar de ACIA, verplaatst de kop van de uitvoer-buffer een positie omhoog en verlaagt de bufferteller met 1. Het schakelt dan zowel de in- als de uitvoer-interrupt weer in en zet de uitvoer-interrupt-vlag (voor het geval deze eerder gewist mocht zijn).

Het nieuwe probleem dat zich voordoet bij het gebruik van meervoudige-karakter-buffers is het beheer van geordende rijen. Het hoofdprogramma moet de data in dezelfde volgorde lezen als waarin de invoer-interrupt- afhandelingsroutine deze ontvangt. Hetzelfde geldt voor de uitvoer-interrupt- afhandelingsroutine; deze moet de data in dezelfde volgorde versturen als waarin het

hoofdprogramma ze opslaat. Zodoende moeten we aan de volgende voorwaarden voldoen om invoer- af te handelen:

1. Het hoofdprogramma moet weten of er iets in de invoer-buffer is.

2. Als de invoer-buffer niet leeg is, moet het hoofdprogramma weten waar het 'oudste' karakter (het karakter dat het eerst werd ontvangen) zich bevindt.

3. De invoer-interrupt-afhandelingsroutine moet weten of de invoer-buffer vol is.

4. Als de invoer-buffer niet vol is, moet de invoer-interrupt- afhandelingsroutine weten waar de volgende lege plaats is. Anders gezegd: de routine moet weten waar het nieuwe karakter opgeslagen moet worden.

De uitvoer-interrupt-afhandelingsroutine en het hoofdprogramma hebben een soortgelijke reeks voorwaarden voor de uitvoer-buffer, hoewel de rollen van zender en ontvanger omgedraaid zijn.
We kunnen aan voorwaarden 1 en 3 voldoen door een teller (ICNT) bij te houden. INIT initialiseert ICNT op nul, de interrupt-afhandelingsroutine telt er telkens 1 bij op als het een karakter ontvangt (vooropgesteld dat de buffer niet vol is) en het hoofdprogramma trekt er telkens 1 vanaf als het een karakter uit de buffer haalt (vooropgesteld dat de buffer niet leeg is). Zodoende kan het hoofdprogramma bepalen of de invoer-buffer vol is door te cotroleren of ICNT gelijk is aan de grootte van de buffer.
We kunnen aan voorwaarden 2 en 4 voldoen door twee indexen bij te houden: IHEAD en ITAIL. Deze zijn als volgt gedefinieerd:

1. ITAIL is de index van de volgende lege plaats in de buffer.

2. IHEAD is de index van het oudste karakter in de buffer.

INIT initialiseert IHEAD en ITAIL op nul. Telkens wanneer de interrupt- afhandelingsroutine een karakter ontvangt, plaatst het dit karakter in de buffer op index ITAIL en verhoogt het ITAIL met 1 (ervan uitgaande dat de buffer niet vol is). Telkens wanneer het hoofdprogramma een karakter leest, verwijdert het dit karakter uit de buffer op index IHEAD en verhoogt het IHEAD met 1 (vooropgesteld dat de buffer niet leeg is). Zodoende 'achtervolgt' IHEAD ITAIL door de buffer; de afhandelingsroutine slaat karakters in het ene eind (de staart) op, terwijl het hoofdprogramma de karakters uit het andere eind (de kop) haalt. Het bezette gebied van de buffer kan dus overal beginnen en eindigen. Als of IHEAD of ITAIL aan het fysieke einde van de buffer komt, zetten we deze eenvoudigweg op nul. Zodoende staan we 'omwikkeling' (wrap-around) op de buffer toe. Anders gezegd: het bezette deel van de buffer kan bij het einde beginnen (bij voorbeeld op byte nr. 195 van een 200-bytes buffer) en doorlopen tot het begin (bij voorbeeld op byte nr. 10). In dit geval zou IHEAD 195 en ITAIL 10 zijn en zou de buffer 15 karakters bevatten die bytes #195 tot en met 199 en 0 tot en met 9 beslaan.

Gebruikte registers:

1.	INCH	A, F, Y
2.	INST	A, F
3.	OUTCH	A, F, Y
4.	OUTST	A, F
5.	INIT	A, F

Looptijd:

1.	INCH	70 cycli als er een karakter beschikbaar is

2. INST 18 cycli
3. OUTCH minimaal 75 cycli, maximaal 105 cycli als de uitvoer-buffer niet vol is en de ACIA klaar is om te versturen
4. OUTST 12 cycli
5. INIT 89 cycli
6. IOSRVC 73 cycli om een invoer-interrupt af te handelen, 102 cycli om een uitvoer-interrupt af te handelen, 27 cycli om vast te stellen dat interrupt van een ander apparaat kwam

Programmalengte: 258 bytes

Benodigd datageheugen: Zeven bytes ergens in RAM, plus de invoer- en uitvoer-buffers. De zeven bytes ergens in RAM bevatten de invoer-bufferteller (een byte op adres ICNT), de index naar de kop van de invoer-buffer (een byte op adres IHEAD), de index naar de staart van de invoer-buffer (een byte op adres ITAIL), de uitvoer-bufferteller (een byte op adres OCNT), de index naar de kop van de uitvoer-buffer (een byte op adres OHEAD), de index naar de staart van de uitvoer-buffer (een byte op adres OTAIL) en een byte voor de output-interrupt-vlag (op adres OIE). De invoer-buffer begint op adres IBUF; de grootte ervan wordt bepaald door IBSZ. De uitvoer-buffer begint op OBUF; de grootte hiervan wordt bepaald door OBSZ.

Begincondities

1. INCH: Geen
2. INST: Geen
3. OUTCH: Te versturen karakter in accumulator
4. OUTST: Geen
5. INIT: Geen

Eindcondities

1. INCH: Karakter in accumulator
2. INST: Overschotvlag = 0 als geen karakter beschikbaar is, 1 als er wel karakters beschikbaar zijn.
3. OUTCH: Geen
4. OUTST: Overschotvlag = 0 als uitvoer-buffer niet vol is, 1 als de buffer vol is.
5. INIT: Geen

```
;
;     Titel :      Eenvoudige, interrupt-gestuurde, in- en uitvoer met
;                   behulp  van  een 6850 ACIA en een buffer van meerdere
;                   karakters
;
;     Naam :       SINTB
;
```

```
;
;       Doel :          Dit programma bestaat uit vijf subroutines die samen voor
;                       interrupt-gestuurde in- en uitvoer zorgen, met behulp van
;                       een 6850 ACIA
;
;                       INCH
;                               Lees een karakter
;
;                       INST
;                               Bepaal de invoerstatus (kijk of de invoer-buffer
;                               leeg is)
;
;                       OUTCH
;                               Schrijf een karakter
;
;                       OUTST
;                               Bepaal de uitvoerstatus (kijk of de uitvoer-
;                               buffer leeg is)
;
;                       INIT
;                               Initialiseer
;
;
;       Begin :
;                       INCH
;                               Geen parameters
;
;                       INST
;                               Geen parameters
;
;                       OUTCH
;                               Register A = uit te sturen karakter
;
;                       OUTST
;                               Geen parameters
;
;                       INIT
;                               Geen parameters
;
;
;       Einde :
;                       INCH
;                               Register A = karakter
;
;                       INST
;                               Overdrachtvlag = 0 als de invoer-buffer leeg is
;                               Overdrachtvlag = 1 als er een karakter in de
;                               invoer-
;                               buffer zit
;
;                       OUTCH
;                               Geen parameters
;
;                       OUTST
;                               Overdrachtvlag = 0 als de uitvoer-buffer leeg is
;                               Overdrachtvlag = 1 als er een karakter in de
;                               uitvoer-
;                               buffer zit
;
;                       INIT
;                               Geen parameters
;
```

```
;
;       Registers :
;                       INCH
;                               A, F, Y
;
;                       INST
;                               A, F
;
;                       OUTCH
;                               A, F, Y
;
;                       OUTST
;                               A, F
;
;                       INIT
;                               A, F
;
;
;       Tijd :
;                       INCH
;                               70 cycli als er een karakter aanwezig is
;
;                       INST
;                               18 cycli
;
;                       OUTCH
;                               75 cycli minimaal als de uitvoer-buffer niet vol
;                               is
;                               en de ACIA klaar is om een karakter te versturen
;
;                       OUTST
;                               12 cycli
;
;                       INIT
;                               89 cycli
;
;                       IOSRVC
;                               27 cycli minimaal, als de interupt niet door ons
;                                gegenereerd is
;                               73 cycli om een invoer-interrupt af te handelen
;                               102 cycli om een uitvoer-interrupt af te handelen
;
;       Grootte :
;                       Programma 258 bytes
;                       Data        7 bytes plus de lengte van de buffers
;
;
;       Buffers :
;                       De routines verwachten twee buffers die beginnen bij de
;                       adressen IBUF en OBUF. De lengte van de buffers in bytes
;                       staan in IBSZ en OBSZ. Voor de invoer-buffer is IHEAD de
;                       index naar het oudste karakter (het volgende karakter dat
;                       het hoofd programma te lezen krijgt), ITAIL wijst naar de
;                       eerstvolgende vrije plaats in de buffer (de positie die
;                       het eerst door de afhandel-routine gevuld dient te
;                       worden)  en ICNT is het aantal karakters dat op dat
;                       moment in de buffer zit. Voor de uitvoer-buffer is OHEAD
;                       de index naar het
;                       oudste karakter in de buffer (het volgende karakter dat
;                       verstuurd moet worden), OTAIL wijset naar de
;                       eerstvolgende
;                       vrije plaats in de buffer (de positie die het hoofd
```

```
;                   programma het eerst dient te vullen)  en OCNT het aantal
;                   karakters dat in de buffer zit.
;
;     N.B. :
;                   Beide buffers werken volgens het wrap-around mechanisme
;                   (Als een index het eind van de buffer overschrijdt gaat
;                   hij wijzen vanaf het begin van de buffer), zodat het
;                   huidige gevulde gebied overal in de buffer kan beginnen
;                   en
;                   door kan gaan tot het einde van de buffer en weer bij het
;                   begin van de buffer door kan gaan. Bij voorbeeld, als de
;                   uitvoer-buffer 40H bytes lang is, kan het gevulde gebied
;                   lopen van OBUF+32H (OHEAD = 32H) tot OBUF+10H
;                   (OTAIL = 11H). Er zijn dan dus 19H bytes gevuld in de
;                   buffer, 8 bytes van adres OBUF+32H tot en met OBUF+39H en
;                   17 bytes vanaf adres OBUF tot en met OBUF+10H. De buffer
;                   lijkt dus een beetje op een televisiebeeld, waarbij de
;                   verticale afbuiging versteld is, waardoor het beeld net
;                   voor het einde van het scherm begint en boven aan het
;                   scherm doorgaat.


; Voorbeeld van 6850 ACIA definities voor een Apple serieele kaart in slot 1
ACIASR:             .EQU    0C094H      ; ACIA statusregister
ACIADR:             .EQU    0C095H      ; ACIA dataregister
ACIACR:             .EQU    0C094H      ; ACIA controleregister
IRQVEC:             .EQU    03FEH       ; Apple IRQ vectoradres

; Lees een karakter
INCH:
                    jsr     INST        ; Haal de invoerstatus op
                    bcc     INCH        ; Wacht als er geen karakter
                                        ; in de invoer-buffer zit
                    php                 ; Bewaar de huidige status van
                                        ; de interrupts
                    sei                 ; Zet de interrupts af
                    ldy     IHEAD
                    lda     IBUF,Y      ; Haal het huidige karakter op
                    iny
                    cpy     #IBSZ       ; Is wrap-around nodig?
                    bcc     INCH1       ; Spring als het niet nodig is
                    ldy     #0          ; Ga wijzen naar het begin van
                                        ;  de invoer-buffer
INCH1:
                    sty     IHEAD       ; Bewaar de nieuwe index
                    dec     ICNT        ; Er zit nu een karakter
                                        ;  minder in de buffer
                    plp                 ; Zet de interruptvlag
                                        ; terug
                    rts

; Keer terug met de invoerstatus (overdrachtvlag is 1 als er een karakter in
; de invoer-buffer aanwezig is
INST:
                    clc                 ; Maak de overdracht schoon
                                        ;  (neem aan dat er geen
                                        ;  karakters in de buffer
                                        ;  zitten)
                    lda     ICNT
                    beq     INST1       ; Spring als er geen karakters
                                        ;  in de buffer zitten
```

```
                sec                     ; Geef aan dat er minstens een
                                        ;  karakter in de buffer zit
INST1:
                rts


; Schrijf een karakter
OUTCH:
                php                     ; Haal de interruptstatus op
                pha                     ; Bewaar het uit te sturen
                                        ; karakter

        ; Wacht tot de uitvoer-buffer leeg is, en zet dan het
        ; nieuwe karakter in de buffer
WAITOC:
                jsr     OUTST           ; Haal de uitvoerstatus op
                bcs     WAITOC          ; Wacht tot de buffer leeg is
                sei                     ; Zet de interrupts af
                pla                     ; Haal het nieuwe karakter op
                ldy     OTAIL
                sta     OBUF,Y          ; Stop het karakter in de
                                        ;  buffer
                iny
                cpy     #OBSZ           ; Gaan we niet over het eind
                                        ;  van de buffer heen?
                bcc     OUTCH1
                ldy     #0              ; Zo ja, ga wijzen naar het
                                        ;  begin van de buffer
OUTCH1:
                sty     OTAIL
                inc     OCNT            ; Er zit nu een karakter meer
                                        ;  in de buffer
                lda     OIE             ; Zijn de interrupts uitgezet,
                                        ;  maar is de ACIA klaar ?
                bne     OUTCH2          ; Nee, keer dus terug
                jsr     OUTDAT          ; Stuur een karakter naar de
                                        ;  poort en zet de interrupts
                                        ;  aan
OUTCH2:
                plp                     ; Zet de interrupt-flag
                                        ; terug
                rts

; Uitvoerstatus (overdrachtvlag = 1 als de buffer vol is)
OUTST:
                lda     OCNT
                cmp     #OBSZ           ; Als OCNT >= OBSZ dan
                                        ;  buffer is vol ->
                                        ;    overdracht = 1
                                        ; anders
                                        ;  er is nog plaats voor een
                                        ;  karakter -> overdracht = 0
                rts

; Initialiseer
INIT:
                php                     ; Bewaar de huidig waarde van
                                        ; de interruptvlag
                sei                     ; Zet de interrupt uit tijdens
                                        ; het initialiseren
```

```
        ; Initialiseer de softwarevlaggen
                lda     #0
                sta     ICNT            ; Er is nog geen invoer
                sta     IHEAD
                sta     ITAIL
                sta     OCNT            ; Er is nog geen uitvoer
                sta     OHEAD
                sta     OTAIL
                sta     OIE             ; De ACIA is klaar om een
                                        ;  karakter te versturen!

        ; Bewaar de huidige IRQ-vector in NEXTSR
                lda     IRQVEC
                sta     NEXTSR
                lda     IRQVEC+1
                sta     NEXTSR+1

        ; Laat de IRQ-vector naar onze invoer-afhandelroutine
        ; wijzen
                lda     AIOS
                sta     IRQVEC
                lda     AIOS+1
                sta     IRQVEC+1

        ; Initialiseer de 6850
                lda     #011B
                sta     ACIACR          ; Reset de ACIA
                lda     #10010001B
                sta     ACIACR          ; Initialiseer de ACIA modus
                                        ; tot
                                        ;
                                        ;    deel door 16
                                        ;    8 databits
                                        ;    2 stopbits
                                        ;    geen uitvoer-interrupts!
                                        ;    wel invoer-interrupts

                plp                     ; Zet de interruptvlag terug
                rts

AIOS:           .WORD   IOSRVC          ; Adres van de invoer-afhandel-
                                        ; routine

IOSRVC:
                pha                     ; Bewaar de inhoud van
                                        ; register A
                cld                     ; Zet de processor in binaire
                                        ; modus

        ; Haal de status van de ACIA op
        ;       Bit 0 = 1 als het een invoer-interrupt was
        ;       Bit 1 = 1 als het een outvoer-interrupt was
                lda     ACIASR
                lsr     A               ; Bit 0 naar de overdrachtvlag
                bcs     IINT            ; Spring als het een invoer-
                                        ; interrupt is
                lsr     A               ; Bit 1 naar de overdrachtvlag
                bcs     OINT            ; Spring als het een uitvoer-
                                        ; interrupt is

        ; Deze interrupt is niet gegenereerd door de ACIA
                pla
```

```
                jmp     (NEXTSR)        ; Ga naar de volgende
                                        ; afhandelroutine

; Handel een invoer-interrupt af
IINT:
                tya
                pha                     ; Bewaar de inhoud van
                                        ;  register Y

        ; Haal het karakter op en stop het, mits er daarvoor ruimte
        ; is, in de buffer
                lda     ACIADR          ; Haal het karakter op
                ldy     ICNT            ; Is er nog ruimte in de
                                        ;  buffer?
                cpy     #IBSZ
                bcs     EXIT            ; Geen ruimte, ga terug
                ldy     ITAIL
                sta     IBUF,Y          ; Stop het karakter in de
                                        ;  buffer
                iny                     ; Verhoog de index
                cpy     #IBSZ           ; Gaan we over het einde van
                                        ;  de buffer heen?
                bcc     IINT1           ; Indien niet, spring dan
                ldy     #0              ; Ga wijzen naar het begin van
                                        ;  de buffer
IINT1:
                sty     ITAIL           ; Bewaar de nieuwe index
                inc     ICNT            ; Er zit nu een karakter meer
                                        ;  in de buffer
                jmp     EXIT                    : Keer terug

; Handel een uitvoer-interrupt af
OINT:
                tya
                pha

                lda     OCNT            ; Is de buffer nog niet leeg?
                beq     NODATA          ; Spring als er niets
                                        ; verzonden hoeft te worden
                jsr     OUTDAT          ; Verstuur het karakter
                jmp     EXIT            ; Keer terug naar de plaats
                                        ; waar we de interrupt kregen

NODATA:
                lda     #10010001B      ; Zet de uitvoer-interrupt af,
                                        ;  zet de invoer-interrupt aan
                                        ;  8 databits
                                        ;  2 stopbits
                                        ;  deel door 16 klok
                sta     ACIACR          ; Zet de uitvoer-interrupt af

EXIT:
                pla
                tay                     ; Zet register Y weer goed
                pla                     ; Zet register A weer goed
                rti                     ; Keer terug van de interrupt


;
;       Subroutine : OUTDAT
;
;       Doel       : Stuur een karakter naar de ACIA
```

```
;       Begin       : OHEAD is de index in OBUF waar het karakter dat verstuurd
;                     moet worden, staat
;
;       Eind        : Geen
;
;       Registers   : A, F
;

OUTDAT:
                      lda     ACIASR
                      and     #00000010B    ; Test of het ACIA uitvoer-
                                            ; register leeg is
                      beq     OUTDAT        ; Spring als het niet leeg is
                      ldy     OHEAD
                      lda     OBUF,Y        ; Haal het karakter op
                      sta     ACIADR        ; Verstuur het
                      iny                   ; Verhoog de index in de
                                            ;  buffer
                      cpy     OBSZ          ; Gaan we niet over het einde
                                            ;  van de buffer heen?
                      bcc     OUTD1         ; Nee, ga door
                      ldy     #0            ; Ja, ga wijzen naar het
                                            ;  begin van de buffer
OUTDT1:
                      sty     OHEAD         ; Bewaar de nieuwe index
                      dec     OCNT          ; Er zit een karakter minder
                                            ;  in de buffer
                      lda     #10110001B
                      sta     ACIACR        ; Zet zowel de ACIA invoer-
                                            ; als uitvoer-interrupts aan,
                                            ;  8 databits, 2 stopbits,
                                            ;  deel door 16 klok
                      lda     #0FFH
                      sta     OIE           ; Geef aan dat de uitvoer-
                                            ;  interrupts aan staan
                      rts

; Data
ICNT:                 .BLOCK  1             ; Teller voor het aantal
                                            ;  karakters in de invoer-
                                            ;  buffer
IHEAD:                .BLOCK  1             ; Wijzer naar de kop van de
                                            ;  invoer-buffer
ITAIL:                .BLOCK  1             ; Wijzer naar de staart van de
                                            ;  invoer-buffer

OCNT:                 .BLOCK  1             ; Teller voor het aantal
                                            ;  karakters in de invoer-
                                            ;  buffer
OHEAD:                .BLOCK  1             ; Wijzer naar de kop van de
                                            ;  uitvoer-buffer
OTAIL:                .BLOCK  1             ; Wijzer naar de staart van de
                                            ;  uitvoer-buffer

OIE:                  .BLOCK  1             ; Vlag om te zien of de
                                            ;  uitvoer-interrupts aan
                                            ;  staan

IBSZ:                 .EQU    80            ; Grootte van de invoer-buffer
IBUF:                 .BLOCK  IBSZ          ; De invoer-buffer
OBSZ:                 .EQU    80            ; Grootte van de uitvoer-
                                            ;  buffer
```

```
OBUF:           .BLOCK  OBSZ        ; De uitvoer-buffer

NEXTSR:         .BLOCK  2           ; Adres van de volgende
                                    ; interrupt-afhandelroutine


;
;
; Voorbeeld uitvoering
;
;

SC1103:
                jsr     INIT        ; Initialiseer
                cli                 ; Zet de interrupts aan

        ; Simpel voorbeeldje
LOOP:
                jsr     INCH        ; Lees een karakter
                pha
                jsr     OUTCH       ; Stuur het weer uit
                pla
                cmp     #1BH        ; Is het een escape-karakter
                bne     LOOP        ; Spring terug als het geen
                                    ; escape-karakter is
                brk

        ; Een asynchroon voorbeeld
        ;  Stuur constant een 'A' naar de console, maar handel ook
        ;  alle invoer af (d.w.z. stuur de invoer ook naar de
        ;  console).
ASYNLP:
        ; Als het uitvoer-medium niet bezig is, stuur er dan een
        ; 'A' naartoe
                jsr     OUTST       ; Is het uitvoer-medium bezig?
                bcs     ASYNLP      ; Spring als het bezig is
                lda     #"A"
                jsr     OUTCH       ; Verstuur het karakter

        ; Als er een karakter in de invoer buffer zit, haal het dan
        ; op
                jsr     INST        ; Is er een karakter aanwezig?
                bcc     ASYNLP      ; Spring als er geen karakter
                                    ; In de buffer zit
                jsr     INCH        ; Lees een karakter
                cmp     #1BH        ; Is het een escape-karakter?
                beq     DONE        ; Spring als het een escape-
                                    ; karakter is
                jsr     OUTCH       ; Stuur het weer uit
                jmp     ASYNLP      ; en ga door

DONE:
                brk

                .END
```

Real-time klok en kalender - 11D

Houdt een 24-uurs klok die de tijd van de dag en de datum aangeeft bij, gebaseerd op een real-time klok-interrupt. Bestaat uit de volgende subroutines:

1. CLOCK geeft het beginadres van de klokvariabelen terug.

2. ICLK initialiseert de klok-interrupt en initialiseert de klokvariabelen op hun standaard waarden.

3. CLKINT past de klok na elke interrupt aan (verondersteld dat de interrupts een interval van een tik hebben).

Een uitgebreid voorbeeld in de listing beschijft een tijdaanduidingsroutine voor de Apple // computer. De routine vraagt de gebruiker om de begintijd en -datum. Vervolgens toont ze continu de tijd en datum op het midden van het monitorscherm. De routine gaat ervan uit dat het interruptbord in slot 2 zit.

Procedure:

1. CLOCK laadt het beginadres van de klokvariabelen in de accumulator (hoogstwaardige byte) en indexregister Y (laagstwaardige byte). De klok- variabelen worden in deze volgorde (laagste adres eerst) opgeslagen: tikken, seconden, minuten, uren, dagen, maanden, laagstwaardige byte van jaar, hoogstwaardige byte van jaar.
2. ICLK laadt de klokvariabelen met hun standaard waarden (acht bytes vanaf adres DFLTS) en initialiseert de klok-interrupt (meestal systeem-afhankelijk).
3. CLKINT verlaagt de resterende tikteller en past de rest van de klok, indien nodig, aan. Uiteraard moet het aantal seconden en minuten minder dan 60 zijn en het aantal uren minder dan 24. De dag van de maand moet kleiner zijn dan of gelijk zijn aan de laatste dag van de huidige maand; een array met de laatste dag van elke maand begin op adres LASTDY. Tijdens de maand februari (maand 2) moet het programma controleren of het huidige jaar een schrikkeljaar is. Hiervoor moet gekeken worden of de twee laagstwaardige bits van geheugenlocatie YEAR beide nul zijn. Als het huidige jaar inderdaad een schrikkeljaar is, is de 29e en niet de 28e de laatste dag van februari. Het maandnummer mag 12 (december) niet overschrijden, anders is er een overdracht naar het jaarnummer nodig. Het programma moet de variabelen goed her-initialiseren als er overdrachten plaatsvinden; dus TICK naar DTICK; seconden, minuten en uren naar nul; dag en maand naar 1 (respectievelijk de eerste dag en januari).

Gebruikte registers:

1. CLOCK: A, F, Y
2. ICLK: A, Y
3. CLKINT: Geen

Looptijd:

1. CLOCK: 14 cycli
2. ICLK: 166 cycli
3. CLKINT: 33 cycli als alleen TICK verlaagd hoeft te worden, 184 cycli maximaal als er een nieuw jaar moet verschijnen

Programmalengte:

1. CLOCK: 7 bytes
2. ICLK: 39 bytes
3. CLKINT: 145 bytes

Benodigd datageheugen: Achttien bytes ergens in RAM. Dit is inclusief acht bytes voor de klokvariabelen (vanaf adres ACVAR), acht bytes voor de standaard (default) waarden (vanaf adres DFLTS) en twee bytes voor het adres van de volgende afhandelingsroutine (vanaf adres NEXTSR).

Begincondities

1. CLOCK: Geen
2. ICLK: Geen
3. CLKINT: Geen

Eindcondities

1. CLOCK: Hoogstwaardige byte van beginadres klokvariabelen in accumulator, laagstwaardige byte in index register Y.
2. ICLK: Geen
3. CLKINT: Geen

Voorbeelden

Deze voorbeelden veronderstellen dat de tikfrequentie DTICK Hz is (kleiner dan 256 Hz - typische waarden zouden 50, 60 of 100 Hz zijn) en dat de klok en de kalender zijn opgeslagen in de geheugenlocaties

TICK	aantal resterende tikken voordat er een overdracht plaatsvindt. Afgeteld vanuit DTICK
SEC	seconden (0 tot en met 59)
MIN	minuten (0 tot en met 59)
HOUR	uren van dag (0 tot en met 23)
DAY	dag van maand (1 tot en met 28, 30 of 31, afhankelijk van maand)
MONTH	maand van jaar (1 tot en met 12 voor januari tot en met december)
YEAR & YEAR+1	huidige jaar

1. Beginwaarden zijn 7 maart, 1982. 11:59.59 n.m. en 1 tik over
Anders gezegd:
(TICK) = 1
(SEC) = 59
(MIN) = 59
(HOUR) = 23
(DAY) = 07
(MONTH) = 03
(YEAR) = 1982

Resultaat (na de tik): 8 maart 1982 12:00.00 en DTICK tikt.

Anders gezegd:

(TICK) = DTICK
(SEC) = 0
(MIN) = 0
(HOUR) = 0
(DAY) = 08
(MONTH) = 03
(YEAR) = 1982

2. Beginwaarden zijn 31 dec. 1982. 11:59.59 en 1 tik over
Anders gezegd:

(TICK) = 1
(SEC) = 59
(MIN) = 59
(HOUR) = 23
(DAY) = 31
(MONTH) = 12
(YEAR) = 1982

Resultaat (na tik): 1 januari 1983, 12:00.00 v.m. en DTICK tikt.
Anders gezegd:

(TICK) = DTICK
(SEC) = 0
(MIN) = 0
(HOUR) = 0
(DAY) = 1
(MONTH) = 1
(YEAR) = 1983

```
;
;       Titel :         Constant bijgewerkte klok en kalender
;
;       Naam :          CLOCK
;
;
;       Doel :          Dit programma houdt de datum en tijd bij met behulp van
;                       de klok-interrupt
;
;                       CLOCK
;                               Keer terug met het adres van de klokvariabelen
;
;                       ICLK
;                               Initialiseer de klok-interrupts
;
;       Einde :
;                       CLOCK
;                               Register A = Hoogstwaardige byte van het adres
;                                            van de klokvariabelen
;                               Register Y = Laagstwaardige byte van het adres
;                                            van de klokvariabelen
;
;                       ICLK
;                               Geen
;
```

```
;
;       Registers :  Alle
;
;       Tijd :
;                    CLOCK
;                            14 cycli
;
;                    ICLK
;                            166 cycli
;
;                    CLKINT
;                            22 cycli als interrupt niet van ons afkomstig is
;                            33 cycli normaal
;                           184 cycli maximum als we over moeten gaan op een
;                                     nieuw jaar
;
;       Grootte :
;                    Programma 191 bytes
;                    Data       18 bytes
;

IRQVEC:              .EQU    03FEH          ; Apple IRQ-vector
CLKPRT:              .EQU    0C0A0H         ; Slot 2 IO-locatie van een
                                            ;  interrupt-board
CLKIM:               .EQU    01H            ; Bit 0 = vraag om een
                                            ;  interrupt-bit
TRUE:                .EQU    0FFH           ; TRUE = ongelijk aan 0
FALSE:               .EQU    0              ; FALSE = gelijk aan 0


; Keer terug met het adres van de klokvariabelen
CLOCK:
                     lda     ACVAR+1
                     ldy     ACVAR
                     rts

; Initialiseer de klok-interrupt
ICLK:
                     php                    ; Bewaar de vlaggen
                     sei                    ; Zet de interrupts uit

            ; Initialiseer de klokvariabelen op hun systeem-gekozen
            ;  waarden
                     ldy     #8

ICLK1:
                     lda     DFLTS-1,Y
                     sta     CLKVAR-1,Y
                     dey
                     bne     ICLK1

            ; Bewaar de waarde van de huidige IRQ-vector
                     lda     IRQVEC
                     sta     NEXTSR
                     lda     IRQVEC+1
                     sta     NEXTSR+1

            ; Laat de IRQ-vector wijzen naar onze routine
                     lda     ACINT
                     sta     IRQVEC
                     lda     ACINT+1
                     sta     IRQVEC+1
```

```
        ; Hier moet code staan om de interrupt-hardware te
        ; initialiseren


        ; Keer terug
                plp                     ; Zet de interrupt-vlaggen
                                        ;  terug
                rts

; Handel de klok-interrupt af
CLKINT:
                pha                     ; Bewaar de inhoud van A
                cld                     ; Zet de processor in binaire
                                        ;  modus

        ; Test of deze interrupt voor ons bestemd was
        ;       Dit is alleen maar een voorbeeld
                lda     CLKPRT
                and     #CLKIM          ; Kijk naar het interrupt-bit
                bne     OURINT          ; Spring als de interrupt
                                        ;  voor ons bestemd was
                pla                     ; Zet register A weer goed
                jmp     (NEXTSR)        ; Ga naar de volgende
                                        ;  interrupt-afhandelroutine


        ; Verwerk de interrupt
OURINT:
                dec     TICK
                bne     EXIT1           ; Keer terug, als de tikken-
                                        ;  teller nog niet gelijk aan
                                        ;  0 is
                lda     DTICK
                sta     TICK            ; Zet de originele tikwaarde
                                        ;  terug in TICK

        ; Bewaar ook de registers X en Y
                tya
                pha
                txa
                pha

        ; Verhoog het aantal seconden met een
                inc     SEC
                lda     SEC
                cmp     #60
                bcc     EXIT            ; Spring terug als er nog geen
                                        ; minuut om is
                ldy     #0
                sty     SEC             ; Zet de secondenteller op 0

        ; Verhoog het aantal minuten met een
                inc     MIN
                lda     MIN
                cmp     #60             ; Spring terug als er nog geen
                                        ;  uur om is
                bcc     EXIT
                sty     MIN             ; Zet de minutenteller op 0

        ; Verhoog het aantal uren met een
                inc     HOUR
                lda     HOUR
```

```
                cmp     #24
                bcc     EXIT            ; Spring terug als er nog geen
                                        ;  dag voorbij is
                sty     HOUR

        ; Verhoog het aantal dagen met een
                inc     DAY
                lda     DAY
                ldx     MONTH
                cmp     LASTDY-1,X
                bcc     EXIT            ; Spring terug als er nog geen
                                        ;  maand voorbij is

        ; Ga door naar de volgende maand
                cpx     #2
                bne     INCMTH          ; Spring als we niet in de
                                        ;  maand februari zitten
                lda     YEAR
                and     #00000011B
                bne     INCMTH          ; Spring als het geen
                                        ;  schrikkeljaar is

        ; Het is een schrikkeljaar, februari heeft dus 29
        ;  dagen in plaats van 28 dagen
                lda     DAY
                cmp     #29
                beq     EXIT            ; Spring terug als 29 februari
                                        ;  nog niet voorbij is

INCMTH:
                ldy     #1
                sty     DAY             ; Zet het dagnummer op 1
                inc     MONTH           ; Verhoog het maandnummer
                lda     MONTH
                cmp     #13
                bcc     EXIT            ; Spring terug als er nog geen
                                        ;  jaar voorbij is,
                sty     MONTH           ; anders is het nu januari
        ; Een nieuw jaar
                inc     YEAR            ; Het volgende jaar
                bne     EXIT
                inc     YEAR+1          ; Verhoog de hoogstwaardige
                                        ;  byte

EXIT:
        ; Zet de registers terug naar hun oorspronkelijke waarden
                pla
                tax
                pla
                tay

EXIT1:
                pla
                rti                     ; Keer terug van een interrupt

; Array van de laatste dag van iedere maand
LASTDY:
                .BYTE   31              ; januari
                .BYTE   28              ; februari
                .BYTE   31              ; maart
                .BYTE   30              ; april
                .BYTE   31              ; mei
```

```
                    .BYTE   30          ; juni
                    .BYTE   31          ; juli
                    .BYTE   31          ; augustus
                    .BYTE   30          ; september
                    .BYTE   31          ; oktober
                    .BYTE   30          ; november
                    .BYTE   31          ; december

; Klokvariabelen
ACVAR:              .WORD   CLKVAR
CLKVAR:
TICK:               .BLOCK  1           ; Tikken over in de huidige
                                        ;  seconde
SEC:                .BLOCK  1           ; Seconden
MIN:                .BLOCK  1           ; Minuten
HOUR:               .BLOCK  1           ; Uren
DAY:                .BLOCK  1           ; Dagen
MONTH:              .BLOCK  1           ; Maand
YEAR:               .WORD   0           ; Jaar

; Systeem-gekozen waarden
DFLTS:
DTICK:              .BYTE   60          ; Klok-frequentie is 60 Hz
DSEC:               .BYTE   0
DMIN:               .BYTE   0
DHR:                .BYTE   0
DDAY:               .BYTE   1
DMTH:               .BYTE   1
DYEAR:              .WORD   1981        ; 00:00:00 1 JAN 1981

NEXTSR:             .BLOCK  2           ; Adres van de volgende
                                        ;  interrupt-afhandelroutine
ACINT:              .WORD   CLKINT      ; Adres van de klok-interrupt-
                                        ;  routine

;
;
;     Voorbeeld uitvoering
;
;                 Deze routine vraagt de gebruiker om een begindatum en
;                 tijd en slaat deze op. Daarna wordt continu de datum en
;                 de tijd midden op het scherm gezet, tot de gebruiker een
;                 escape-karakter invoert. Daarna wordt weer om een nieuwe
;                 datum en tijd gevraagd en gaat het programma weer door.
;
;

; CLK variabelen posities
OTICK:              .EQU    0
OSEC:               .EQU    1
OMIN:               .EQU    2
OHR:                .EQU    3
ODAY:               .EQU    4
OMTH:               .EQU    5
OYEAR:              .EQU    6

CVARS:              .EQU    0D0H        ; Tijdelijke variabele voor
                                        ;  het adres van de klok-
                                        ;  variabelen op pagina 0

; In dit voorbeeld gaan we uit van de Apple
ESC:                .EQU    1BH         ; Het escape-karakter
```

```
; Apple Monitor aanknopingspunten
CH:             .EQU    24H         ; Horizontale cursor-positie
CV:             .EQU    25H         ; Verticale cursor-positie
HOME:           .EQU    0FC58H      ; Home-routine
VTAB:           .EQU    0FC22H      ; Vtab-routine
RCHAR:          .EQU    0FD0CH      ; Karakter-invoerroutine
COUT:           .EQU    0FDEDH      ; Karakter-uitvoerroutine
GETLN1:         .EQU    0FD6FH      ; Lees een regel in routine
                                    ;  met onze eigen prompt

SC1104:
                jsr     ICLK        ; Initialiseer

        ; Vraag om een datum en een tijd   MM/DD/YY HR:MIN:SEC
        ;  Zet een prompt op het scherm

PROMPT:
                jsr     HOME        ; Maak het scherm schoon en
                                    ;  zet de cursor linksboven
                                    ;  op het scherm
                lda     #0
                sta     MSGIDX

PMTLP:
                ldy     MSGIDX
                lda     MSG,Y
                beq     RDTIME      ; Spring als de hele vraag op
                                    ;  het scherm staat
                inc     MSGIDX      ; Ga wijzen naar het volgende
                                    ;  karakter
                jsr     WRCHAR      ; Druk een karakter af
                jmp     PMTLP

        ; Lees de tijd in

RDTIME:
                jsr     RDLINE      ; Lees een regel in
                                    ; De invoer komt te staan in
                                    ;  de Apple buffer vanaf adres
                                    ;  200H, de lengte staat in X

        ; Haal het adres van de klokvariabelen op
                jsr     CLOCK       ; Haal het adres op
                sta     CVARS+1
                sty     CVARS       ; Bewaar het adres

        ; Initialiseer de variabelen voor het inlezen van de nummers
                stx     LLEN        ; Bewaar de lengte
                lda     #0
                sta     LIDX        ; Initialiseer index in de
                                    ;  buffer

        ; Lees de maand in
                jsr     NXTNUM      ; Haal het volgende nummer uit
                                    ;  de buffer
                ldy     #OMTH
                sta     (CVARS),Y   ; Zet de maand goed

        ; Lees de dag in
                jsr     NXTNUM
```

```
        ldy     #ODAY
        sta     (CVARS),Y

; Lees het jaar in
        jsr     NXTNUM
        ldy     #OYEAR
        sta     (CVARS),Y
        clc
        adc     CEN20           ; Tel 1900 bij het nummer op
        sta     (CVARS),Y       ; Laagstwaardige byte
        lda     CEN20+1
        adc     #0
        iny
        sta     (CVARS),Y       ; Hoogstwaardige byte

; Lees het uur in
        jsr     NXTNUM
        ldy     #OHR
        sta     (CVARS),Y

; Lees het aantal minuten in
        jsr     NXTNUM
        ldy     #OMIN
        sta     (CVARS),Y

; Lees het aantal seconden in
        jsr     NXTNUM
        ldy     #OSEC
        sta     (CVARS),Y

; Zet de interrupts aan
        cli                     ; Zet de interrupts aan

; Maak het scherm schoon en plaats de cursor linksboven
        jsr     HOME

; Druk iedere seconde opnieuw de tijd en datum af
;  op regel 12, kolom 12
LOOP:
        lda     #11
        sta     CV              ; Zet de horizontale en
        sta     CH              ;  verticale cursor-positie
        jsr     VTAB            ; Zet de cursor op de goede
                                ;  plaats

; Druk de maand af
        ldy     #OMTH
        lda     (CVARS),Y
        jsr     PRTNUM          ; Druk het nummer af
        lda     #'/'
        jsr     WRCHAR          ; Druk een '/' af

; Druk de dag af
        ldy     #ODAY
        lda     (CVARS),Y
        jsr     PRTNUM
        lda     #'/'
        jsr     WRCHAR

; Druk het jaar af
        ldy     #OYEAR
        lda     (CVARS),Y
```

```
        sec
        sbc     CEN20           ; Druk b.v. 1985 als 85 af
        jsr     PRTNUM

; Druk een spatie af als scheider tussen de datum en de tijd
        lda     #' '
        jsr     WRCHAR

; Druk het uur af
        ldy     #OHR
        lda     (CVARS),Y
        jsr     PRTNUM
        lda     #':'            ; Druk een ':' af
        jsr     WRCHAR

; Druk de minuten af
        ldy     #OMIN
        lda     (CVARS),Y
        jsr     PRTNUM
        lda     #':'
        jsr     WRCHAR

; Druk de seconden af
        ldy     #OSEC
        lda     (CVARS),Y
        jsr     PRTNUM

; Wacht tot de secondenteller verandert
;  Stop als de gebruiker een toets indrukt

        ldy     #OSEC
        lda     (CVARS),Y
        sta     CURSEC          ; Bewaar de huidige waarde van
                                ;  de secondenteller

WAIT:
; Kijk of er een toets wordt ingedrukt
        jsr     KEYPRS
        bcs     RDKEY           ; Spring als er een toets is
                                ;  ingedrukt
        lda     (CVARS),Y
        cmp     CURSEC
        beq     WAIT            ; Wacht tot de twee waarden
                                ;  ongelijk zijn
        jmp     LOOP            ; Ga door

; De gebruiker heeft een toets ingedrukt
;   Vraag opnieuw om een datum en tijd als de escape-toets
;   was ingedrukt
;  Stop op iedere andere toets
RDKEY:
        jsr     RDKEY           ; Haal het karakter op
        cmp     #ESC            ; Is het een escape?
        beq     DONE            ; Spring als het geen escape
                                ;  was
        jmp     PROMPT          ; Het was een escape

DONE:
        lda     #0
        sta     CH              ; De nulde kolom
        lda     #12
        sta     CV              ; De dertiende regel
```

```
                    jsr     VTAB          ; Zet de cursor goed
                    brk
                    jmp     SC1104

;
;       Subroutine : KEYPRS
;
;       Doel :       Kijk of de gebruiker een toets indrukt
;
;       Begin :      Geen
;
;       Eind       : Als de gebruiker een toets heeft ingedrukt
;                          Overdrachtvlag = 1
;                      anders
;                          Overdrachtvlag = 0
;
;       Registers :  P

KEYPRS:
                    pha
                    lda     0C000H        ; Kijk naar de Apple Keyboard
                                          ; poort
                    asl     A             ; Overdrachtvlag = bit 7
                    pla
                    rts


;
;       Subroutine : RDCHAR
;
;       Doel       : Lees een karakter
;
;       Begin      : Geen
;
;       Einde      : Register A = ingelezen karakter
;
;       Registers  : A, P
;

RDCHAR:
                    pha                   ; Bewaar de inhoud van de
                    tya                   ;  registers A, X, Y
                    pha
                    txa
                    pha

                    jsr     RCHAR         ; Apple Monitor RDCHAR routine
                    tsx
                    and     #01111111B    ; Wis bit 7
                    sta     103H,X        ; Bewaar het karakter op de
                                          ;  stack zodat het in register
                                          ;  A kan worden teruggezet

                    pla                   ; Zet de waarden van A, X, Y
                    tax                   ;  terug
                    pla
                    tay
                    pla
                    rts

;
;       Subroutine : WRCHAR
```

```
;
;       Doel        : Schrijf een karakter
;
;       Begin       : Register A = karakter
;
;       Einde       : Geen
;
;       Registers   : P
;

WRCHAR:
                    pha
                    tya
                    pha
                    txa
                    pha

                    tsx
                    lda     103H,X          ; Haal register A van de stack
                                            ;  af
                    ora     #10000000B      ; Zet bit 7
                    jsr     COUT            ; Apple Monitor
                                            ;  druk karakter af routine

                    pla                     ; Zet A, Y, X terug
                    tax
                    pla
                    tay
                    pla
                    rts


;
;       Subroutine : RDLINE
;
;       Doel        : Lees een regel in via de Apple Monitor
;
;       Begin       : Geen
;
;       Einde       : Register X = lengte van de regel
;
;       Registers   : Allemaal
;


RDLINE:
                    jsr     GETLN1
                    rts


;
;       Subroutine : NXTNUM
;
;       Doel        : Haal een nummer van de invoerregel.
;                     Keer terug met 0 als er geen nummer te vinden was.
;
;       Begin       : LLEN = lengte van de regel
;                     LLIDX = index in de regel naar het volgende karakter
;
;       Einde       : Register A = Laagstwaardige byte van het nummer
;                     Register Y = Hoogstwaardige byte van het nummer
;
;       Registers   : Allemaal
```

```
NXTNUM:
                lda     #0
                sta     NUM
                sta     NUM+1           ; Nummer is nu 0

        ; Wacht tot een decimaal cijfer is gevonden
        ;  (een karakter tussen 30H en 39H)
                jsr     GETCHR          ; Haal een volgend karakter op
                bcs     EXITNN          ; Keer terug als het einde van
                                        ; de regel is bereikt
                cmp     #'0'
                bcc     NXTNUM          ; Wacht als het karakter
                                        ;  kleiner dan een '0' is
                cmp     #'9'+1
                bcs     NXTNUM          ; Wacht als het karakter
                                        ;  groter dan een '9' is

        ; We hebben een nummer gevonden
GETNUM:
                pha                     ; Bewaar het karakter

        ; Vermenigvuldig het nummer met 10
                lda     NUM
                asl     A
                rol     NUM+1
                sta     NUM             ; NUM = Laagstwaardige byte
                                        ;  NUM * 2
                ldx     NUM+1           ; X = Hoogstwaardige byte
                                        ;  NUM * 2
                asl     A
                rol     NUM+1
                asl     A               ; A = Laagstwaardige byte
                                        ;  NUM * 8
                rol     NUM+1           ; Hoogstwaardige byte NUM =
                                        ;  NUM * 8
                clc                     ; (NUM * 8) + (NUM * 2) =
                                        ;  NUM * 10
                adc     NUM
                sta     NUM
                txa
                adc     NUM+1
                sta     NUM+1

        ; Tel het karakter bij het nummer op
                pla                     ; Haal het karakter van de
                                        ;  stack
                and     #00001111B      ; Karakter genormaliseerd tot
                                        ;   0 .. 9
                clc
                adc     NUM
                sta     NUM
                bcc     GETNM1
                inc     NUM+1
GETNM1:
        ; Haal het volgende karakter op
                jsr     GETCHR
                bcs     EXITNN          ; Keer terug als het einde van
                                        ;  de regel is bereikt
                cmp     #'0'            ; Keer terug als het karakter
                                        ;  kleiner dan ' ' is
                bcc     EXITNN
```

```
                cmp     #'9'+1          ; Keer terug als het karakter
                                        ;  groter dan '9' is
                bcc     GETNUM

EXITNN:
                lda     NUM             ; Keer terug met het nummer
                ldy     NUM+1
                rts


;
;       Subroutine : GETCHR
;
;       Doel       : Haal een karakter voor de regel op
;
;       Begin      :
;                    LIDX = index naar het volgende karakter
;                    LLEN = lengte van de regel
;
;       Einde      : Als er geen karakters meer zijn dan
;                            overdracht = 1
;                    anders
;                            overdracht = 0
;                            Register A = karakter
;
;       Registers  : Allemaal
;

GETCHR:
                lda     LIDX
                cmp     LLEN
                bcs     EXITGC          ; Keer terug, met overdrachtvlag
                                        ;  gezet om aan te geven dat
                                        ;  het einde van de regel
                                        ;  bereikt is, anders is
                                        ;  overdrachtvlag = 0

                tay
                lda     200H,Y          ; Haal het karakter op
                and     #01111111B      ; Wis bit 7
                iny                     ; Ga wijzen naar het volgende
                                        ;  karakter
                sty     LIDX
                                        ; De overdrachtvlag is nog
                                        ;  steeds 0

EXITGC:
                rts


;
;       Subroutine : PRTNUM
;
;       Doel       : Druk een nummer af (vanaf 0 tot en met 99)
;
;       Begin      : A = getal dat afgedrukt moet worden
;
;       Einde      : Geen
;
;       Registers  : Allemaal
```

```
PRTNUM:
                ldy     #'0'-1          ; Y = karakter voor '0'
                                        ;  In Y worden de tientallen
                                        ;  berekend
                sec
DIV10:
                iny                     ; Verhoog de tientallen
                sbc     #10             ; Trek 10 van de eenheden af
                bcs     DIV10
                adc     #10+'0'         ; Maak er een karakter van

        ; Register A = eenheden
        ; Register Y = tientallen
                tax                     ; Bewaar de eenheden
                tya
                jsr     WRCHAR          ; Druk de tientallen af
                txa
                jsr     WRCHAR          ; Druk de eenheden af
                rts


; Data
CR:             .EQU    0DH             ; ASCII wagen terug karakter
MSG:            .BYTE   'ENTER DATE AND TIME ',CR,
                        '(MM/DD/YR HR:MN:SC) ? ',0
                                        ; Af te drukken string
MSGIDX:         .BLOCK  1               ; Index in de string
NUM:            .BLOCK  2               ; Nummer
LLEN:           .BLOCK  1               ; Lengte van de invoerregel
LIDX:           .BLOCK  1               ; Index in de invoerregel
CEN20:          .WORD   1900            ; Twintigste eeuw
CURSEC:         .BLOCK  1               ; Huidige seconde


                .END
```

Appendix A 6502 Overzicht 6502 Instructie Set

Tabel A-1. 6502 Instructies in alfabetische volgorde

instructies		IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM			IMPLIED			IND X			IND Y			Z PAGE X			ABS Y			ABS X			RELATIVE			INDIRECT			Z PAGE Y			conditie-codes					
MNEMONIC	OPERATION	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	OP	N	#	N	Z	C	I	D	V
ADC	A + M + C → A (4) (1)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3										✓	✓	✓	–	–	✓
AND	A ∧ M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3										✓	✓	–	–	–	–
ASL	C ← [7 0] ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3													✓	✓	✓	–	–	–
BCC	BRANCH ON C=0 (2)																															90	2	2							–	–	–	–	–	–
BCS	BRANCH ON C=1 (2)																															B0	2	2							–	–	–	–	–	–
BEQ	BRANCH ON Z=1 (2)																															F0	2	2							–	–	–	–	–	–
BIT	A ∧ M				2C	4	3	24	3	2																															M7	✓	–	–	–	M6
BMI	BRANCH ON N=1 (2)																															30	2	2							–	–	–	–	–	–
BNE	BRANCH ON Z=0 (2)																															D0	2	2							–	–	–	–	–	–
BPL	BRANCH ON N=0 (2)																															10	2	2							–	–	–	–	–	–
BRK	(See Fig. A-1)													00	7	1																									–	–	–	–	–	–
BVC	BRANCH ON V=0 (2)																															50	2	2							–	–	–	–	–	–
BVS	BRANCH ON V=1 (2)																															70	2	2							–	–	–	–	–	–
CLC	0 → C													18	2	1																									–	–	0	–	–	–
CLD	0 → D													D8	2	1																									–	–	–	–	0	–
CLI	0 → I													58	2	1																									–	–	–	0	–	–
CLV	0 → V													B8	2	1																									–	–	–	–	–	0
CMP	A-M (1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3										✓	✓	✓	–	–	–
CPX	X-M	E0	2	2	EC	4	3	E4	3	2																															✓	✓	✓	–	–	–
CPY	Y-M	C0	2	2	CC	4	3	C4	3	2																															✓	✓	✓	–	–	–
DEC	M-1 → M				CE	6	3	C6	5	2													D6	6	2	DE	7	3													✓	✓	–	–	–	–
DEX	X-1 → X													CA	2	1																									✓	✓	–	–	–	–
DEY	Y-1 → Y													88	2	1																									✓	✓	–	–	–	–
EOR	A ∨ M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3										✓	✓	–	–	–	–
INC	M + 1 → M				EE	6	3	E6	5	2													F6	6	2	FE	7	3													✓	✓	–	–	–	–
INX	X + 1 → X													E8	2	1																									✓	✓	–	–	–	–
INY	Y + 1 → Y													C8	2	1																									✓	✓	–	–	–	–
JMP	JUMP TO NEW LOC				4C	3	3																												6C	5	3				–	–	–	–	–	–
JSR	(See Fig. A-2) JUMP SUB				20	6	3																																		–	–	–	–	–	–
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3										✓	✓	–	–	–	–

Tabel A-1. 6502 Instructies in alfabetische volgorde (vervolg)

MNEMONIC	OPERATION	IMMEDIATE OP	IMMEDIATE N	IMMEDIATE #	ABSOLUTE OP	ABSOLUTE N	ABSOLUTE #	ZERO PAGE OP	ZERO PAGE N	ZERO PAGE #	ACCUM OP	ACCUM N	ACCUM #	IMP OP	IMPLIED OP	IMPLIED N	IMPLIED #	(IND, X) OP	(IND, X) N	(IND, X) #	(IND), Y OP	(IND), Y N	(IND), Y #	Z. PAGE, X OP	Z. PAGE, X N	Z. PAGE, X #	ABS. X OP	ABS. X N	ABS. X #	ABS. Y OP	ABS. Y N	ABS. Y #	RELATIVE OP	RELATIVE N	RELATIVE #	INDIRECT OP	INDIRECT N	INDIRECT #	Z. PAGE, Y OP	Z. PAGE, Y N	Z. PAGE, Y #	CONDITION CODES N	Z	C	I	D	V
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2																			BE	4	3								B6	4	2	✓	✓	–	–	–	–
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2														B4	4	2	BC	4	3												✓	✓	–	–	–	–
LSR	0 → 7 → 0 → C				4E	6	3	46	5	2	4A	2	1											56	6	2	5E	7	3												0	✓	✓	–	–	–
NOP	NO OPERATION														EA	EA	2	1																								–	–	–	–	–	–
ORA	A ∨ M → A	09	2	2	0D	4	3	05	3	2								01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3									✓	✓	–	–	–	–
PHA	A → Ms S-1 → S														48	48	3	1																								–	–	–	–	–	–
PHP	P → Ms S-1 → S														08	08	3	1																								–	–	–	–	–	–
PLA	S + 1 → S Ms → A														68	68	4	1																								✓	✓	–	–	–	–
PLP	S + 1 → S Ms → P														28	28	4	1																								(RESTORED)					
ROL	7 ← 0 ← C ←				2E	6	3	26	5	2	2A	2	1											36	6	2	3E	7	3												✓	✓	✓	–	–	–
ROR	→ C → 7 → 0 →				6E	6	3	66	5	2	6A	2	1											76	6	2	7E	7	3												✓	✓	✓	–	–	–
RTI	(See Fig. A-1) RTRN INT														40	40	6	1																								RESTORED)					
RTS	(See Fig. A-2) RTRN SUB														60	60	6	1																								–	–	–	–	–	–
SBC	A-M-$\bar{C}$ → A (1)	E9	2	2	ED	4	3	E5	3	2									E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3								✓	✓	(3)	–	–	✓
SEC	1 → C														38	38	2	1																								–	–	1	–	–	–
SED	1 → D														F8	F8	2	1																								–	–	–	–	1	–
SEI	1 → I														78	78	2	1																								–	–	–	1	–	–
STA	A → M				8D	4	3	85	3	2									81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3									–	–	–			
STX	X → M				8E	4	3	86	3	2																													96	4	2	–	–	–	–	–	–
STY	Y → M				8C	4	3	84	3	2														94	4	2																–	–		–	–	
TAX	A → X														AA	AA	2	1																								✓	✓	–	–	–	–
TAY	A → Y														A8	A8	2	1																								✓	✓	–	–	–	–
TSX	S → X														BA	BA	2	1																								✓	✓	–	–	–	–
TXA	X → A														8A	8A	2	1																								✓	✓	–	–	–	–
TXS	X → S														9A	9A	2	1																								–	–	–	–	–	–
TYA	Y → A														98	98	2	1																								✓	✓	–	–	–	–

(1) Bij overschrijding paginagrens 1 bij "n" optellen
(2) Bij vertakking binnen zelfde pagina 1 bij "n" optellen
Bij vertakking naar andere pagina 2 bij "n" optellen
(3) Niet carry = leen
(4) In decimale mode is nulvlag ongeldig, controleer accumulator voor resultaat gelijk 0

x index x
y index y
a accumulator
m effectief geheugenadres
m1 stapelwijzer geheugenadres

+ optellen
- aftrekken
and
or
exlusief or
gewijzigd
niet gewijzigd

m7 geheugenbit 7
m6 geheugenbit 6
n aantal klokcycli
aantal bytes

Tabel A-2. 6502 operatie codes in numerieke volgorde

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	LSD / MSD
0	BRK	ORA IND, X				ORA Z. Page	ASL Z. PAGE		PHP	ORA IMM	ASL A			ORA ABS	ASL ABS		0
1	BPL	ORA-IND, Y				ORA-Z.Page,X	ASL-Z. Page, X		CLC	ORA-ABS. Y				ORA-ABS. X	ASL-ABS. X		1
2	JSR	AND IND, X			BIT-Z Page	AND Z. Page	ROL Z. Page		PLP	AND IMM	ROL A		BIT ABS	AND ABS	ROL ABS		2
3	BMI	AND IND, Y				AND-Z.Page,X	ROL-Z.Page,X		SEC	AND-ABS. Y				AND-ABS. X	ROL ABS. X		3
4	RTI	EOR IND, X				EOR Z.Page	LSR Z.Page		PHA	EOR IMM	LSR A		JMP ABS	EOR ABS	LSR ABS		4
5	BVC	EOR-IND, Y				EOR-Z.Page,X	LSR-Z.Page,X		CLI	EOR-ABS. Y				EOR-ABS. X	LSR ABS. X		5
6	RTS	ADC IND, X				ADC Z.Page	ROR Z. Page		PLA	ADC IMM	ROR A		JMP IND	ADC ABS	ROR ABS		6
7	BVS	ADC-IND, Y				ADC-Z.Page,X			SEI	ADC-ABS. Y				ADC-ABS. X			7
8		STA IND, X			STY Z. Page	STA Z.Page	STX Z.Page		DEY		TXA		STY ABS	STA ABS	STX ABS		8
9	BCC	STA-IND, Y			STY-Z.Page,X	STA-Z.Page,X	STX-Z.Page,Y		TYA	STA-ABS. Y	TXS			STA ABS. X			9
A	LDY IMM	LDA IND, X	LDX IMM		LDY Z.Page	LDA Z.Page	LDX Z.Page		TAY	LDA IMM	TAX		LDY ABS	LDA ABS	LDX ABS		A
B	BCS	LDA-IND, Y			LDY-Z.Page,X	LDA-Z.Page,X	LDX-Z.Page,Y		CLV	LDA-ABS. Y	TSX		LDY ABS. X	LDA-ABS. X	LDX-ABS. Y		B
C	CPY IMM	CMP IND, X			CPY Z.Page	CMP Z.Page	DEC Z.Page		INY	CMP IMM	DEX		CPY ABS	CMP ABS	DEC ABS		C
D	BNE	CMP-IND, Y				CMP-Z.Page,X	DEC-Z.Page,X		CLD	CMP-ABS. Y				CMP-ABS. X	DEC-ABS. X		D
E	CPX IMM	SBC IND, X			CPX Z.Page	SBC Z.Page	INC Z.Page		INX	SBC IMM	NOP		CPX ABS	SBC ABS	INC ABS		E
F	BEQ	SBC-IND, Y				SBC-Z.Page,X	INC-Z.Page,X		SED	SBC-ABS. Y				SBC-ABS. X	INC-ABS. X		F

Tabel A-3. Overzicht van 6502 Adresseermodes

imm - onmiddelijke adressering - de operand is vervat in het tweede byte van de instructie

abs - absolute adressering - het tweede byte van de instructie bevat de 8 minst significante adresbits. het derde byte bevat de 8 meest significante adresbits.

0-pagina, pagina nul-adressering - het tweede byte bevat de 8 adresbits van het effectieve adres. de acht meest significante adresbits zijn nul.

a, accumulator-adressering - een byte instructie die op de accumulator werkt.

pagina nul,x, pagina nul,y - pagina nul geindexeerd - het tweede byte van de instructie wordt opgeteld (zonder rekening te houden met carry) om het minst significante deel van het effectieve adres te vormen. het meest significante deel van het effectieve adres is nul.

abs x, abs y, absoluut geindexeerd - het effectieve adres wordt gevormd door de index bij tweede en derde byte van de instructie op te tellen.

(ind,x) geindexeerd indirect - het tweede byte van de instructie wordt zonder carry opgeteld bij index x. het resultaat wijst naar een locatie op pagina nul die de acht minst significante bits van het effectieve geheugen adres bevat. het volgende byte bevat de acht meest significante bits.

(ind),y, indirect geindexeerd - het tweede byte van de instructie wijst naar een locatie in pagina 0. de inhoud van deze geheugenlocatie wordt opgeteld bij y index, waarbij het resultaat de acht minst significante adresbits vormen. Een carry van deze operatie wordt bij de inhoud van het volgende pagina nul adres opgeteld, waarbij de acht meest significante adresbits gevormd worden.

Tabel A-4. 6502 Assembler Besturingsaanwijzingen, Labels en Speciale Tekens

Assembler besturingsaanwijzingen

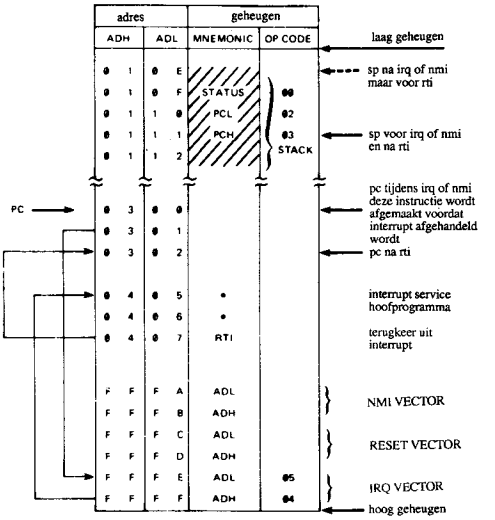
.opt - specificeer opties voor assembleren
opties zijn: (eerst getoonde opties worden door systeem gekozen).
noc (cou of cnt) - maak geen lijst van alle instructies en hun gebruik.
nog (gen) - genereer maximaal 1 regel code voor ascii teksten.
xre (nox) - produceer een kruisreferentielijst in de symbolentabel.
err (noe) - creeer een foutmeldingsbestand.
mem (nom) - creeer een assembler objectuitvoerbestand.
lis (nol) - produceer een volledige assembleer lijst.
.byte - plaats een enkel byte in het geheugen, gelijk aan de gespecificeerde operand.
.word - plaats een adres (twee bytes) in het geheugen, aan de hand van de gespecificeerde-operand.
.dbyte - plaats twee bytes in het geheugen, gespecificeerd door de operand.
.skip - sla een gespecificeerd aantal regels over.
.page - ga met de lijst verder op een nieuwe pagina en verander titel.
.end - definieer het einde van een bronprogramma.
*= - definieer een nieuwe programmateller stand.

labels

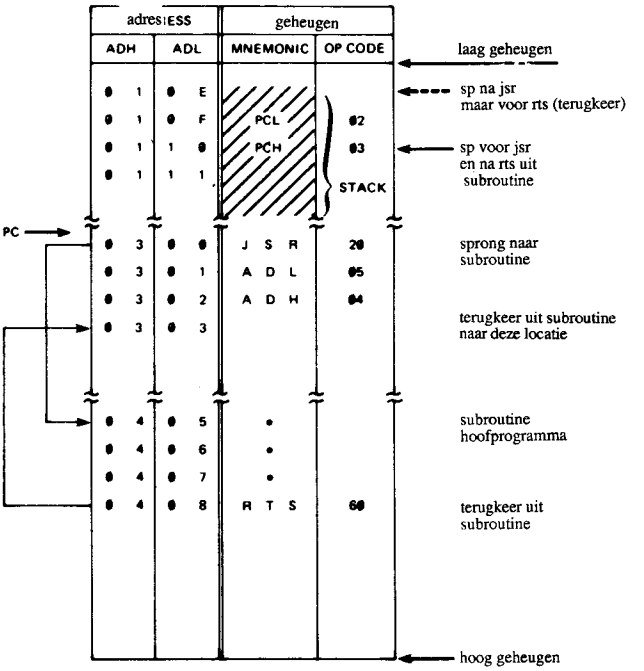
labels vormen het eerste veld en moeten worden gevolgd door minstens een spatie of dubbel punt (:).
labels kunnen tot 6 alfanumerieke tekens lang zijn, en moeten met een alfabetisch teken beginnen.
a, x, y, s, p en de 56 op codes zijn gereserveerd en kunnen niet als labels worden gebruikt.
label = expressie kan worden gebruikt om een waarde aan een label toe te kennen.
label *=*+ kan worden gebruikt om stukken geheugen te reserveren.

Speciale prefix karakters

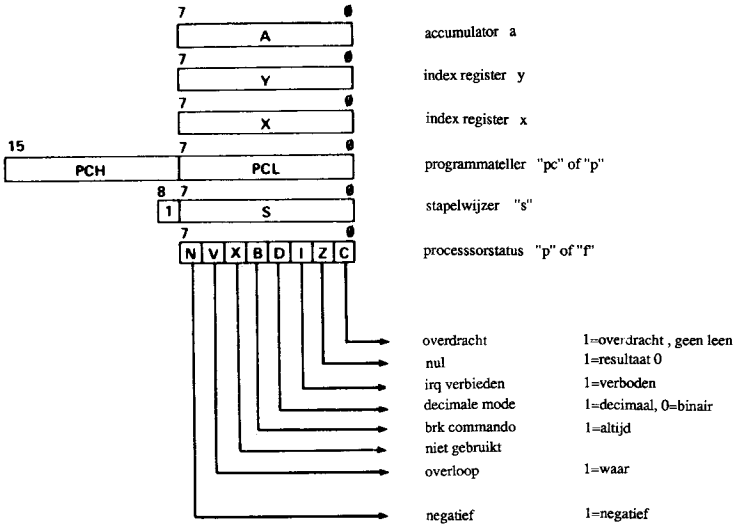
. geeft assemblerbesturingsaanwijzing aan.
# specificeert onmiddelijke adressering.
$ geeft een hexadecimaal getal aan.
@ geeft een octaal getal aan.
% geeft een binair getal aan.
' geeft een ascii tekst teken aan.
() geeft indirecte adressering aan.
; geeft een eropvolgend commentaar aan.
< specificeert minst significante helft 16bit waarde.
> specificeert meest significante helft 16bit waarde.



Figuur A-1. Antwoord op IRQ en NMI invoer en werking van de RTI en BRK instructies.

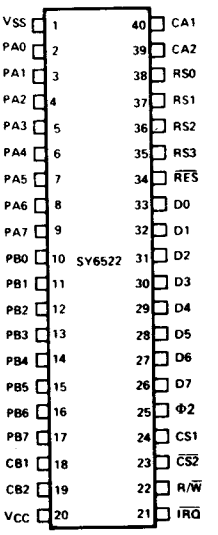


Figuur A-2. Werking van JSR en RTS instructies.

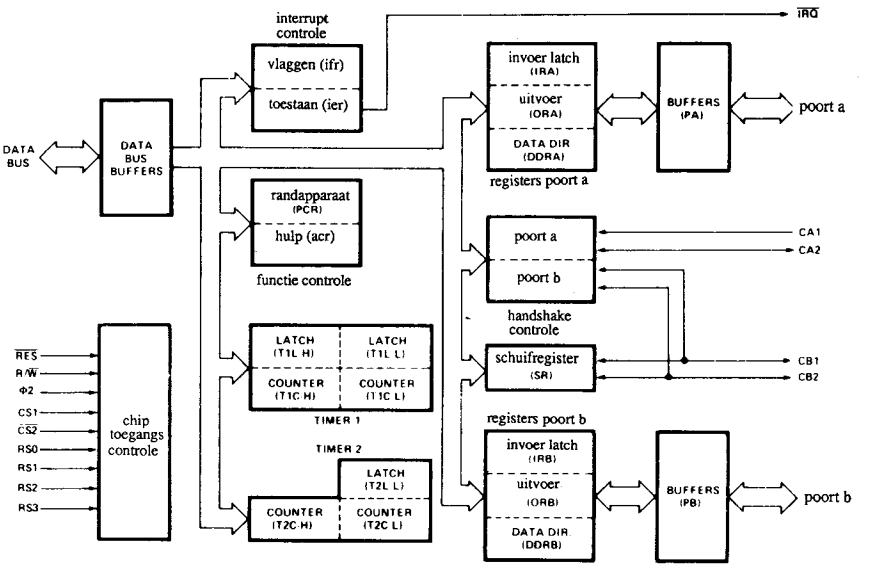


Figuur A-3. Programmeermodel van de 6502 microprocessor.

Appendix B. **Programmeer Overzicht van de 6522 Versatile Interface Adapter (VIA).**



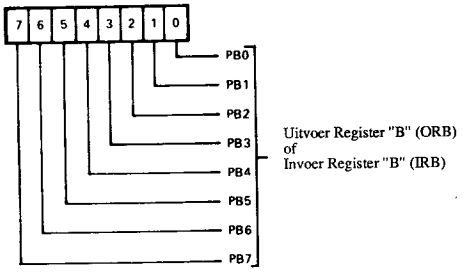
Figuur B-1. 6522 pen toewijzing.



Figuur B-2 Blokschema van de 6522 interface (VIA)

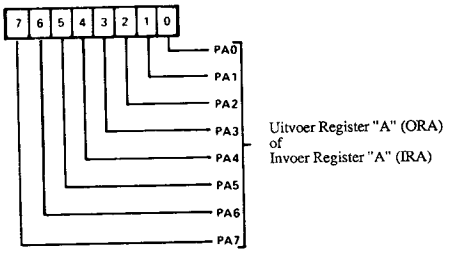
Tabel B-1 Interne registers van de 6522

Register nummer	RS code				Register naam.	Beschrijving.	
	RS3	RS2	RS1	RS0		Schrijf	Lees
0	0	0	0	0	ORB/IRB	Uitvoer Register "B"	Invoer Register "B"
1	0	0	0	1	ORA/IRA	Uitvoer Register "A"	Invoer Register "A"
2	0	0	1	0	DDRB	Data Richtingsregister "B"	
3	0	0	1	1	DDRA	Data Richtingsregister "A"	
4	0	1	0	0	T1C-L	T1 Latch minst sign.	T1 Teller minst sign.
5	0	1	0	1	T1C-H	T1 Teller meest sign.	
6	0	1	1	0	T1L-L	T1 Latch minst sign.	
7	0	1	1	1	T1L-H	T1 Latch meest sign.	
8	1	0	0	0	T2C-L	T2 Latch minst sign.	T2 Teller minst sign.
9	1	0	0	1	T2C-H	T2 Teller meest sign.	
10	1	0	1	0	SR	Schuifregister	
11	1	0	1	1	ACR	Hulp Controle Register	
12	1	1	0	0	PCR	Randapp. Controle Register	
13	1	1	0	1	IFR	Interrupt Vlag Register	
14	1	1	1	0	IER	Interrupt Toestaan Register	
15	1	1	1	1	ORA/IRA	Als reg.1 zonder "handshake"	



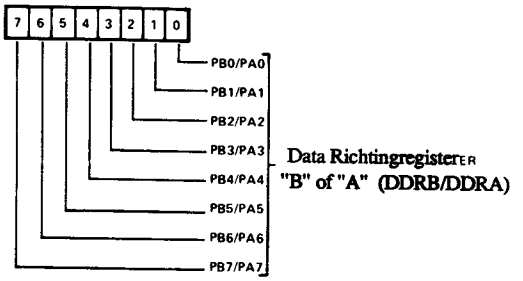
Pen Data Richting	SCHRIJF	LEES
DDRB = "1" (uitvoer)	processor schrijft in ORB	processor leest uitvoer register bit in ORB. pen niveau heeft geen effect
DDRB = "0" (invoer) (zonder latching)	processor schrijft in ORB maar zonder niveau effect totdat DDRB verandert	processor leest invoer op pen PB.
DDRB = "0" (invoer) (met latching)		processor leest IRB bit, dat niveau van pen PB heeft bij laatste actieve overgang van CB1

Figuur B-3. Uitvoer Register B en Invoer Register B (Register 0).



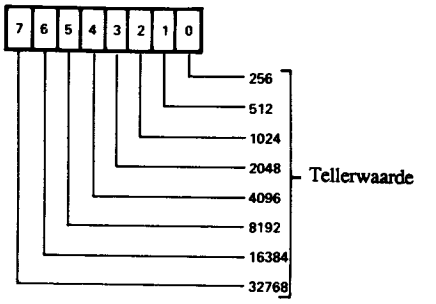
Pen Data Richting	SCHRIJF	LEES
DDRA = "1" (uitvoer) (zonder invoer latching	processor schrijft uitvoer niveau (ORA)	processor leest niveau op pen PA
DDRA = "1" (uitvoer) (met invoer latching)		processor leest IRA bit dat het niveau van pen PA aangeeft bij laatste overgang van CA1
DDRA = "0" (invoer) (zonder invoer latching)	processor schrijft in ORA maar zonder effect op niveau totdat DDRA verandert	processor leest invoer niveau op pen PA.
DDRA = "0" (invoer) (met invoer latching)		processor leest IRA bit, dat niveau van pen PA bij laatste actieve overgang van CA1.

Figuur B-4 Uitvoer register A en invoer register A (register 1)



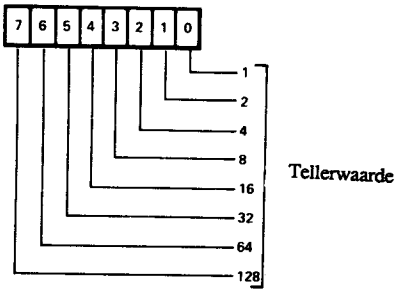
"0" overeenkomende pen PA/PB is een invoer (hoge impedantie).
"1" overeenkomende pen PA/PB is een uitvoer waarvan het niveau bepaald wordt door het ORA/ORB registerbit.

Figuur B-5. Data Richtingregisters B (Register 2) en A (Register 3)



Schrijf - 8 bits worden in meest sign. deel latches T1 geladen. Ook worden nu de beide latches naar de teller gekopieerd en wordt T1 interrupt vlag teruggezet.
Lees - 8 bits uit meest sign. deel teller T1 worden door processor gelezen.

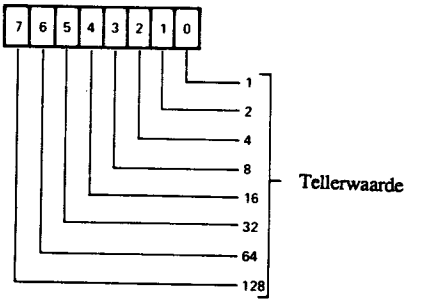
Figuur B-7. Timer 1 Teller hoogwaardig (Register 5)



Tellerwaarde

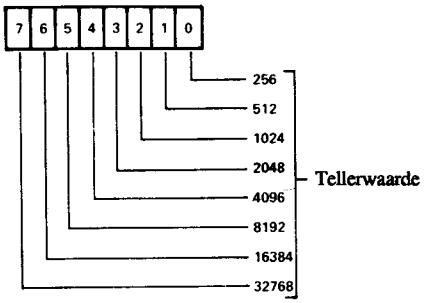
Schrijf - 8 bits worden in minst sign. deel latches T1 geladen. De inhoud wordt naar overeenkomstig deel teller gebracht wanneer het meest significante deel teller (reg.5) geladen wordt.
Lees - 8 bits uit minst sign. deel teller T1 worden door processor gelezen. Ook wordt T1 Interrupt Vlag teruggezet (bit 6 van Interrupt Vlag Register).

Figuur B-6. Timer 1 Teller laagwaardig (Register 4)



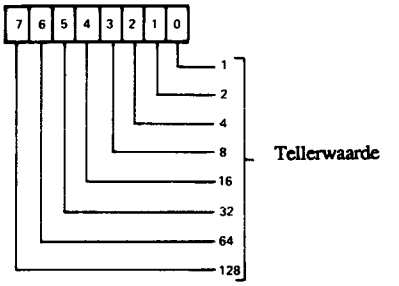
Schrijf - 8 bits worden in minst sign. deel latches T1 geladen. De operatie is hetzelfde als schrijven in register 4.
Lees - 8 bits uit minst sign. deel latches T1 worden door processor gelezen. Nu wordt de T1 Interrupt Vlag niet teruggezet (bit 6 van Interrupt Vlag Register).

Figuur B-8. Timer 1 Teller laagwaardig (Register 6)



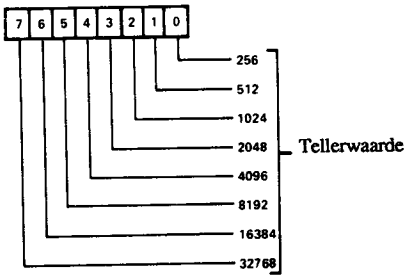
Schrijf - 8 bits worden in meest sign. deel latch T1 geladen. In tegenstelling tot de reg. 4 operatie vindt geen latcl naar-teller overdracht plaats.
Lees - processor leest 8 meest sign. bits latch T1

Figuur B-9. Timer 1 Latches hoogwaardig (Register 7)



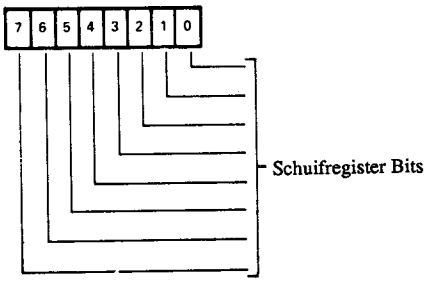
Schrijf - 8 bits worden in minst sign. deel latch T2 geladen.
Lees - processor leest minst sign. deel teller T2. T2 Interrupt Vlag wordt teruggezet.

Figuur B-10. Timer 2 Teller laagwaardig (Register 8)



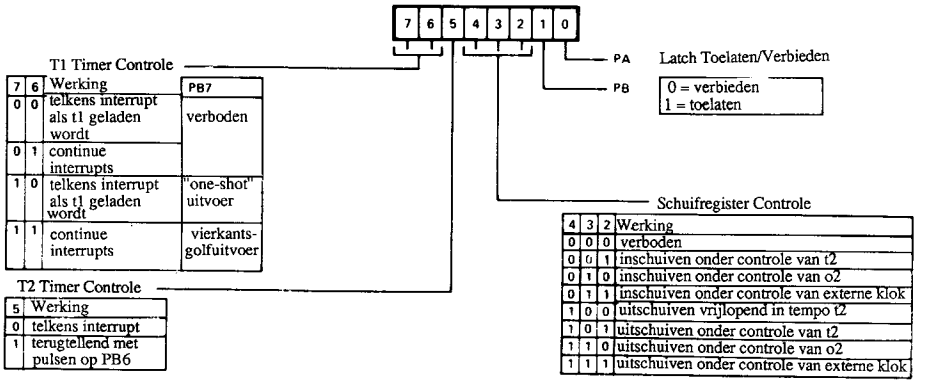
Schrijf - 8 bits worden in meest sign. deel teller T2 geladen. Nu wordt ook de inhoud van de minst sign. latch naar teller gebracht, en wordt T2 Interrupt Vlag teruggezet.
Lees - processor leest 8 meest sign. bits van T2.

Figuur B-11. Timer 2 Teller hoogwaardig (Register 9)

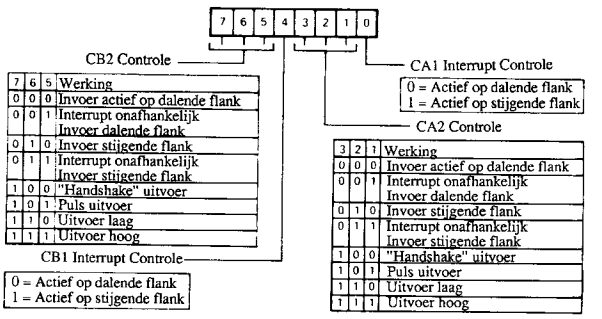


Opmerkingen:
1. Bij uitschuiven is bit 7 het eerste bit en wordt gelijk teruggeschoven naar bit 0.
2. Bij inschuiven, worden bits bij bit 0 naar binnen geschoven en lopen naar bit 7.

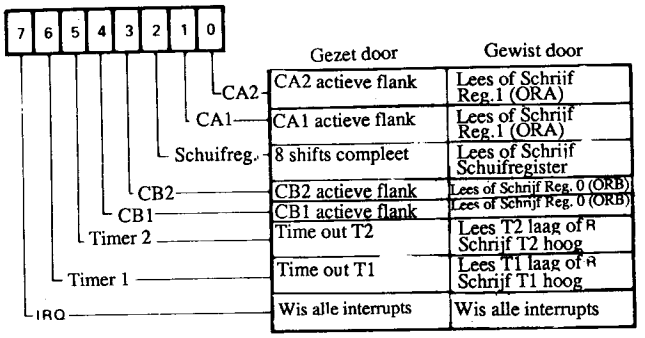
Figuur B-12 Schuifregister (Register 10)



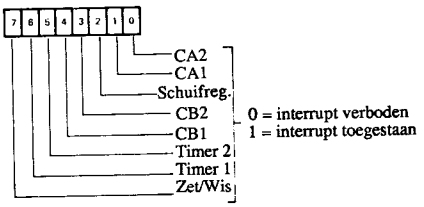
Figuur B-13 Hulp controle register



Figuur b-14 Controle register voor randapparatuur



Figuur B-15. Interrupt Vlag Register (Register 13)



Opmerkingen:

1. Is bit 7 een "0", dan stopt elke "1" in bits 0-6 de corresponderende interrupt.
2. Is bit 7 een "1", dan staat elke "1" in bits 0-6 de corresponderende interrupt toe.
3. Wordt dit register uitgelezen, dan zal bit 7 een "1" zijn, en alle andere bits hun toegestaan/verboden toestand aangeven.

Figuur B-16. Interrupt Toelatings Register (Register 14)

Appendix C ASCII Karakter Set.

LSD \ MSD		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENG	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	}
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	–	=	M	]	m	{
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	VS	/	?	O	_	o	DEL

Verklarende Woordenlijst

A

Aanhouden (van een taak). Stopt de uitvoering en houdt de status van de taak vast tot een nader tijdstip.

Aanroep (van een subroutine). Verplaatst de controle naar de subroutine, waarbij de informatie om het huidige programma te hervatten wordt behouden.

Aansturingsroutine. Zie *I/O-aansturingsroutine*.

Absolute adressering. Een adresseringsmodus waarin de instructie het feitelijke adres bevat dat voor de uitvoering ervan nodig is. In 6502-termen staat absolute adressering voor een type directe adressering waarbij de instructie een volledig 16-bits adres bevat. Dit in tegenstelling tot pagina 0-adressering, waarbij de instructie slechts een 8-bits adres op pagina 0 bevat.

Absolute geindexeerde adressering. Een vorm van geindexeerd adresseren, waarbij de instructie een volledig 16-bits basisadres bevat.

Absoluut adres. Een adres dat een opslaglocatie of apparaat aangeeft zonder gebruik te maken van een basis, een offset of een andere factor. Zie ook *Effectief adres, Relatieve offset*.

Accumulator. Een register dat zonder nadere aanduiding de bron van een enkel gegeven en de bestemming van de meeste rekenkundige en logische handelingen is.

Achteraf-verhogen. Het verhogen van een adresregister nadat het is gebruikt.

Achteraf-verlagen. Het verlagen van een adresregister nadat het is gebruikt.

ACIA. (Asynchronous Communications Interface Adapter). Een serieel interface apparaat. Veel gebruikte ACIA's in 6502-computers zijn de 6551 en 6850 apparaten. Zie ook *UART*.

Adres. De herkenningscode die een geheugenlocatie, een input-poort of een output-poort van een andere onderscheidt en die gebruikt kan worden om een specifieke te selecteren.

Adresregister. Een register dat een geheugenadres bevat.

Adresruimte. De gehele reeks van adressen waaraan een specifieke computer kan refereren.

Adresseringsmodi. De methoden die bij het uitvoeren van een instructie gebruikt kunnen worden om de adressen te specificeren. Algemene adresseringsmodi zijn direct, onmiddellijk, geindexeerd, indirect en relatief.

Afsluiter (tail). De locatie van het oudste voorwerp in een wachtlijst. Dus het eerste gegeven.

ALU. Zie *Rekenkundige-logische eenheid*.

Apparaat-adres. Het adres van poort die in verbinding is met een in- uitvoerapparaat.

Aritmetische schuiving. Zie *Rekenkundige schuiving.*

Array. Een verzameling van aan elkaar gerelateerde gegevens, gewoonlijk in opeenvolgende adressen opgeslagen.

ASCII (American Standard Code for Information Interchange). Een 7-bits karakter-code, veel gebruikt in computers en communicatie.

Assembler. Een computerprogramma dat in assembly taal geschreven programma's in vorm (machine-taal) vertaalt die de computer direct kan uitvoeren. De assembler vertaalt mnemonic-opdrachtcodes en namen in hun numerieke equivalenten en kent geheugenlocaties toe aan data en instructies.

Assembly taal. Een computertaal waarbij de programmeur mnemonic-opdrachtcodes, labels en namen kan gebruiken om aan hun numerieke equivalenten te refereren.

Asynchroon. Handelend zonder gebruik te maken van een algemene timing-bron. Of wel met een onregelmatige interval.

Automatische modus (van een randapparatuur-chip). Een werk-modus waarin de randapparatuur-chip controle-signalen automatisch produceert, zonder specifiek ingrijpen van een programma.

Automatische verhoging. Het automatische verhogen van een adresregister als onderdeel van een instructie die het register gebruikt.

Automatische verlaging. Het automatische verlagen van een adresregister als onderdeel van een instructie die het register gebruikt.

B

Basis adres. Het adres in het geheugen waarop een array of tabel begint. Ook beginadres of basis genoemd.

Baud. Een snelheids-eenheid waarop seriele data wordt verstuurd. Het staat voor het aantal bits per seconde, maar inclusief de data-bits en de bits die worden gebruikt voor synchronisatie, fout-controle en andere doeleinden. Veel gebruikte baud-rates zijn 110, 300, 1200, 2400, 4800 en 9600.

Baudrate-generator. Een apparaat dat de juiste tijd-intervals genereert tussen bits voor seriele data-transmissie.

BCD (Binary-Coded Decimal). Een weergave van decimale getallen, waarbij elk decimaal cijfer afzonderlijk als een binair getal is gecodeerd.

Bestand. Een verzameling van aan elkaar gerelateerde gegevens die wordt behandeld als een eenheid om haar te kunnen opslaan en terug te halen.

Besturingssysteem. Zie *Operating-systeem.*

Bewerkingsloze opdracht (no-op). Een instructie die niets doet dan de programmateller verhogen.

Bibliotheek-programma. Een programma dat onderdeel is van een verzameling programma's en in een standaard-formaat is geschreven en gedocumenteerd.

Bidirectioneel. In staat om signalen in beide richtingen te transporteren.

Binair zoeken of *tweedelingszoekproces.* Een manier van zoeken, waarbij de set te doorzoeken gegevens bij elke herhaling in twee gelijke (of bijna gelijke) delen wordt gedeeld, waarna wordt bekeken met welk deel moet worden doorgegaan.

Binary-Coded Decimal. Zie *BCD.*

Bit test. Een handeling die bepaalt of een bit 0 of 1 is. Refereert gewoonlijk aan een logische AND-handeling met het juiste masker.

Blok. Een hele groep of deel, zoals een registerset of een deel van het geheugen.

Blok-vergelijking. Een handeling waarbij door een geheugenblok wordt gezocht tot het gewenste gegeven is gevonden of het hele blok bekeken is.

Blok-verplaatsing. Het verplaatsen van het ene geheugengebied naar het andere verplaatsen van een hele groep gegevens.

Blokkeertijd (time-out). Een periode waarin geen acitviteit is toegestaan, een non-actieve periode.

Booleaanse variabele. Een variabele die slechts twee waarden kan hebben. Deze kunnen worden weergegeven als waar en niet-waar of als 1 en 0. Zie ook *Vlag.*

Borrow. Zie *Leen-bit.*

Bounce. Tussen verschillende statussen bewegen voordat een uiteindelijke status wordt bereikt.

Break instructie. Zie *Zekering.*

Breakpoint. Een conditie die door de gebruiker wordt gespecificeerd om de uitvoering van een programma tijdelijk te onderbreken. Breakpoints worden gebruikt als een hulpmiddel bij het opsporen en verbeteren van fouten in een programma. Breakpoints zetten is de term voor het aangeven van de condities waaronder de uitvoering stopt. Het opheffen van deze condities heet breakpoints verwijderen.

Bron-programma. Zie *Source-code.*

BSC (Binary Synchronous Communications of BISYNC). Een ouder lijn-protocol dat vaak wordt gebruikt door IBM combuters en terminals.

Bubblesort of *Hinkelsortering.* Een sorteer-techniek die zich door een array heen werkt en elk paar elementen omwisselt als zij in de verkeerde volgorde staan.

Buffer-index. De index van het volgende beschikbare adres in een buffer.

Buffer leeg. Een signaal dat wordt gegeven wanneer de data die in een buffer of register gezet zijn, naar hun uiteindelijke bestemming zijn verplaatst.

Buffer-pointer. Een opslag-locatie die het volgende beschikbare adres in een buffer bevat.

Buffer vol. Een signaal dat wordt gegeven wanneer een buffer of register volledig is bezet met data die nog niet naar hun uiteindelijke bestemming zijn verplaatst.

Bug. Een fout.

Byte. Een eenheid van acht bits. Kan worden omschreven als een eenheid die bestaat uit een hoge nibble of digit (de vier hoogstwaardige bits) en een lage nibble of digit (de vier laagstwaardige bits).

Byte-lengte. Een lengte van acht bits per voorwerp.

C

Call. Zie *Aanroep.*

Carry. Zie *Overdracht-bit.*

Carry-vlag. Zie *Overdrachtvlag.*

CASE-opdracht. Een opdracht in een hogere programmeertaal die de computer aangeeft een subroutine uit een reeks subroutines uit te voeren, afhankelijk van de waarde van een variabele. Of wel, de computer voert de eerste subroutine uit als de variabele de eerst gespecificeerde waarde heeft, enzovoorts. De berekende GO TO heeft een soortgelijke functie in FORTRAN.

Centrale Verwerkingseenheid (Central Processing Unit - CPU). De centrale verwerkingseenheid van de computer die de handelingen regelt, opdrachten haalt en uitvoert en rekenkundige en logische berekeningen uitvoert.

Checksum. Bestaat uit een logische som die in blok data wordt gevoegd om te waken voor opname- en transmissie-fouten. Ook longitudinal parity of longitudinal redundancy check (LRC) genoemd.

Combo-chip. Zie *Multifunctie apparaat.*

Commentaar. Een deel van een programma dat slechts als documentatie dient. Commentaar wordt vertaald noch uitgevoerd; het is enkel in de programma-listing opgenomen.

Complement. Inverteren. Zie ook *Eencomplement, Tweecomplement.*

Controleregister. Ook commando-register. Een register waarvan de inhoud aangeeft wat de status van een overdracht of de werk-modus van een apparaat is.

Controlesignaal. Een signaal dat de richting van een I/O-overdracht aangeeft of de werk-modus van een randapparaat bepaalt.

Controletotaal. Zie *Checksum.*

Cyclic redundancy check (CRC). Een code bedoeld om fouten op te sporen. De code wordt door een polynoom gegenereerd en kan aan een blok data of een geheugengebied worden toegevoegd.

Cyclisch redundancy controle-teken. Zie *Cyclic redundancy check* (CRC)

D

Data klaar (data ready). Een signaal dat actief is als er nieuwe data voor de ontvanger klaar zijn. Identiek aan valid data.

Data ontvangen (data accepted). Een signaal dat actief is wanneer de meest recente data met succes overgedragen is.

Dataregister. De feitelijke in- en uitvoerpoort van een PIA of een VIA. Ook wel output-register of randapparatuur-register genoemd.

Datarichtings-register. Een register dat bepaalt of bidirectionele I/O-lijnen als in- of als uitvoer worden gebruikt. In sommige afbeeldingen als DDR afgekort.

DDCMP (Digital Data Communications Message Protocol). Een veelgebruikt protocol dat elke vorm van fysische data-overdracht (synchroon en asynchroon, serieel en parallel) ondersteunt.

Debounce. Het elimineren van snelle signaal-schommelingen die veel voorkomen bij het omzetten van mechanische schakelingen.

Debounce tijd. De tijd die nodig is om signaal-schommelingen te elimineren.

Debugging. Zie *Fouten opsporen.*

Device-adres. Zie *Apparaatadres.*

Diagnostisch. Een programma dat de werking van een apparaat controleert en de bevindingen hiervan rapporteert.

Digit schuiving. Een schuiving van een enkele BCD-digit-positie of vier bit-posities.

Directe adressering. Een adresseringsmodus waarbij de opdracht het adres bevat dat voor de uitvoering ervan nodig is. De 6502 processor kent twee vormen van directe adressering: pagina 0-adressering (dat alleen een 8-bits adres op pagina 0 nodig heeft) en absolute adressering (dat een volledig 16-bits adres in twee geheugenbytes nodig heeft).

Disable. Zie *Uitschakelen.*

Doel-code. Zie *Object-code.*

Dubbel woord. Een eenheid van 32 bits.

Dump. Een toepassing die de inhoud van een deel van het geheugen of een groep registers op een output-apparaat laat zien.

Dynamische toewijzing (van geheugen). De toewijzing van een beschikbaar deel van het geheugen aan een subroutine op het moment dat deze wordt aangeroepen. Dit staat tegenover de statische toewijzing van een vastgesteld deel van het geheugen aan elke subroutine. Dynamische

toewijzing beperkt vaak het geheugengebruik omdat subroutines sommige gebieden kunnen delen. Het vereist over het algemeen echter extra uitvoeringstijd en overhead bij het beheren van geheugen.

E

EBCDIC (Expanded Binary-Coded Decimal Interchange Code). Een 8-bits karakter code die in veel grotere computers wordt gebruikt.

Echo. Kaatst verzonden informatie terug naar de verzender; geeft de informatie terug aan de terminal die haar heeft verzonden.

Editor. Een programma dat tekst verwerkt en de gebruiker toestaat correcties aan te brengen, aan te vullen, te verwijderen en andere wijzigingen door te voeren.

Eencomplement. Een bity-voor-bit logisch complement van een getal. Het complement wordt verkregen door elk 0-bit door een 1-bit en elk 1-bit door een 0-bit te vervangen.

Effectief adres. Het feitelijke adres dat door een opdracht wordt gebruikt om data op te halen of op te slaan.

EIA RS-232. Zie *RS-232*.

Even pariteit. Een 1-bit fout-opsporingscode die het totale aantal 1-bits in een data-eenheid (inclusief het pariteits-bit) even maakt.

EXCLUSIEVE OR-functie. Een logische functie die waar is als een van de twee inputs waar is, maar niet beide waar zijn. Zodoende is de functie waar als de inputs niet gelijk zijn (of wel, als een van de twee een logische 1 en de ander een logische 0 is).

Externe verwijzing. Het in een programma gebruik maken van een naam die in een ander programma wordt gedefinieerd.

F

F-register. Zie *Processor status register*.

Fouten opsporen. Het proces van het vinden en corrigeren van fouten.

Fouten-opsporings-code. Een programma dat helpt bij het vinden en corrigeren van fouten in een gebruikersprogramma. Enkele versies worden omschreven als ddt (dynamic debugging tools), naar de beruchte insecticide.

Functie-programma. Een programma dat meestal door de computer-fabrikant wordt meegeleverd of onderdeel van een operating-systeem is en standaard of veelgebruikte handelingen uitvoert. Bij voorbeeld sorteren, data van het ene formaat in het andere converteren en bestanden kopieeren.

Functie-toets. Een toets die het systeem opdracht geeft een procedure of functie (bij voorbeeld het wissen van het beeldscherm) uit te voeren.

Fysiek apparaat. Een feitelijk in- of uitvoerapparaat. Dit in tegenstelling tot een logisch apparaat.

G

Gecomprimeerd decimaal. Een binair-gecodeerd decimaal formaat waarbij elk 8-bits byte twee decimale digits bevat.

Gegevensverbinding-controle. Een reeks afspraken met betrekking tot formaat en timing van data-uitwisseling tussen communicerende systemen. Ook protocol genoemd.

Geheugen-capaciteit. Het totale aantal unieke geheugen-adressen (meestal uitgedrukt in bytes) dat op een specifieke computer aangesproken kan worden.

Geindexeerde adressering. Een adresseringsmodus waarbij het adres wordt aangepast door de inhoud van een indexregister om het effectieve adres (het feitelijke adres dat wordt gebruikt) te bepalen.

Geindexeerde indirecte adressering. Een adresseringsmodus waarbij het effectieve adres wordt bepaald door vanaf het basisadres te indexeren en het geindexeerde adres indirect te gebruiken. Dit wordt ook preindexering genoemd, omdat de indexering gebeurt voor het indirecte adresseren. Uiteraard dient de array die op het gegeven basisadres begint, adressen bevatten die indirect kunnen worden gebruikt.

Geinverteerde Leen. Een bit dat op 0 is gezet als een aftrekking een negatief resultaat oplevert en op 1 is gezet als het een positief of nul-resultaat oplevert. Een geinverteerde leen kan worden gebruikt als een echte leen, met die uitzondering dat het complement van de waarde ervan (dat is 1 minus de waarde) in de verlenging naar langere getallen moet worden gebruikt.

Geketende lijst (linked list). Een lijst waarin elk gegeven een wijzer naar het volgende voorwerp bevat. Ook keten of gelinkte lijst genoemd.

Gelinkte lijst. Zie Geketende lijst.

Gepagineerd adres. De identificatie-code die een specifiek adres op een bepaalde pagina aangeeft. In 6502-termen wordt dit gevormd door de acht laagstwaardige bits van een geheugenadres.

Geprogrammeerde in- en uitvoer. In- en uitvoer die onder programma-besturing wordt uitgevoerd, zonder gebruik te maken van interrupts of andere bijzondere hardware-matige technieken.

Geregen programma. Een programma dat uit subroutines bestaat. De subroutines verplaatsen elke nadat er een is afgesloten, automatisch de controle naar de volgende routine.

Gerichte interrupt. Een interrupt die een identificatie-code (of vector) produceert die de CPU kan gebruiken om de controle aan de juiste afhandelingsroutine over te dragen.

Getekend getal. Een getal waarvan een of meerdere bits weergeven of het negatief dan wel positief is. In de meeste gevallen wordt het hoogste bit gebruikt om het teken weer te geven (0 = positief, 1 = negatief).

Globaal. Een universele variabele. In meer dan een deel van een programma gedefinieerd in plaats van dat zij alleen lokaal wordt gebruikt.

H

Handbediende modus (van een randapparatuur-chip). Een werk-modus waarbij de chip alleen dan controle-signalen produceert als een programma daartoe specifiek de opdracht heeft gegeven.

Handelingscode. Dat deel van een opdracht dat de uit te voeren handeling aangeeft.

Handshake. Een asynchrone overdracht waarbij zender en ontvanger vastgestelde signalen uitwisselen om te synchroniseren en de status van de data-overdracht aan te geven. In het algemeen geeft de zender aan dat er nieuwe data beschikbaar zijn; de ontvanger leest de data en geeft aan dat hij klaar is voor meer.

Hardware stack. Een stack die de computer automatisch bijhoudt als deze opdrachten uitvoert die van de stack gebruikmaken.

Header. Zie *hoofd.*

Hexadeximaal (of hex). Een op 16 gebaseerd getallenstelsel. De cijfers ervan zijn de getallen 0 tot en met 9, gevolgd door de letters A tot en met F.

Hogere programmeertaal. Een programmeertaal die meer op het oplossen van problemen is gericht, dan op gemakkelijke omzetting in computer-opdrachten. Een compiler of interpreter (vertolker) vertaalt het in de hogere programmeertaal geschreven programma in een vorm die de computer kan uitvoeren. Tot de gangbare talen behoren BASIC, COBOL, FORTRAN en Pascal.

Hoofd (van een queue of wachtlijst). De locatie van het gegeven dat het laatst aan de wachtlijst is toegevoegd.

Hoogstwaardig bit. Het meest linkse bit in een groep bits. Of wel bit 7 van een byte of bit 15 van een 16-bits woord.

I

Index. Een gegeven dat wordt gebruikt om een specifiek element van een array of tabel aan te geven.

Indexregister. Een register dat kan worden gebruikt om geheugenadressen te veranderen.

Indirecte adressering. Een adresseringsmodus waarbij het effectieve adres de inhoud is van het adres dat in de instructie is opgenomen, in plaats van het adres zelf.

Indirecte geindexeerde adressering. Een adresseringsmodus waarbij het effectieve adres wordt bepaald door eerst indirect het basisadres te halen om vervolgens vanaf dat basisadres te indexeren. Ook wel postindexering genoemd, omdat de indexering plaatsvindt nadat het basisadres indirect is verkregen.

Indirecte sprong. Een sprong-instructie die de controle niet overdraagt aan een vast adres, maar aan een adres dat in een register of geheugenlocatie is opgeslagen.

Input- en Output-controleblok (IOCB). Een reeks geheugenlocaties die de informatie bevatten om een I/O-apparaat te bedienen. Over het algemeen staan hierin de gegevens of de adressen van subroutines die bij voorbeeld een enkele eenheid van data overdragen of de status van een apparaat vaststellen.

Input- en Output-controlesysteem. Een set computer-routines die de uitvoering van I/O-handelingen beheren.

Inschakelen. Staat de doorgang van een activiteit of de herkenning van een signaal (bij voorbeeld een interrupt) toe.

Instructie-cyclus. Het proces van het halen, decoderen en uitvoeren van een instructie.

Instructie-lengte. Het benodigde geheugen om een complete instructie op te slaan.

Instructie-set. De set algemeen bruikbare instructies in een bepaalde computer. De set inputs waarop de CPU een bekende respons levert, wanneer zij gehaald, gedecodeerd en uitgevoerd worden.

Instructie-uitvoeringstijd. De tijd die nodig is om een instructie te halen, te decoderen en uit te voeren.

Interpoleren. Het vaststellen van waarden van een functie op punten die tussen gegeven punten liggen.

Interrupt (onderbreking). Een signaal dat de normale volgorde van handelingen en overdrachten van een computer tijdelijk stopzet en de controle aan een speciale routine overdraagt.

Interrupt-afhandelingsroutine. Een programma dat de benodigde handelingen verricht als er een interrupt is gegeven.

Interrupt-gestuurd. Voor uitvoering afhankelijk van interrupts, kan niets doen tot er een interrupt wordt gegeven.

Interrupt-prioriteitensysteem. Een interrupt-systeem waarbij sommige interrupts voorrang hebben op andere. De interrupts moet voorrang zullen eerst worden afgehandeld of kunnen een de afhandelingsroutines van andere interrupts onderbreken.

Interrupt-verzoek. Een signaal dat actief is wanneer een randapparaat om afhandeling vraagt. Vaak gebruikt om een interrupt van de CPU te veroorzaken. Zie ook Interruptvlag.

Interruptmasker (of interrupt-inschakeling). Een bit dat bepaalt of interrupts herkend zullen worden. Een maskeer- of uitschakel-bit moet gewist worden om interrupts toe te staan. Een Inschakel-bit, daarentegen, moet gezet zijn om hetzelfde te bereieken.

Interruptvlag. Een bit in het in- en uitvoer-deel dat wordt gezet als er iets gebeurt dat afhandeling door de CPU vereist. Typerende voorbeelden van dergelijke gebeurtenissen zijn onder meer het sturen van een signaal over een controle-lijn en het bereiken van een bepaalde waarde door een timer.

IOCB. Zie *Input- en Output-controleblok.*

IOCS. Zie *Input- en Output-controlesysteem.*

I/O-apparatentabel. Een tabel die de verbinding legt tussen de logische apparaten waaraan een programma refereert en de fysieke apparaten die voor de feitelijke data-overdracht worden gebruikt. Een I/O-tabel moet in het geheugen worden opgenomen om een programma aan logische apparaten te laten refereren op een computer met specifieke set van feitelijke (fysieke) apparaten. De I/O-apparatentabel zou bij voorbeeld de begin-adressen kunnen bevatten van routines die de verscheidenen apparaten bedienen.

I/O-device-tabel. Zie *I/O-apparatentabel.*

I/O-aansturingsroutine. Een programma dat de overdracht naar en van een I/O-apparaat regelt. Ook 'driver' of 'I/O-gebruiksprogramma' genoemd. De aansturingsroutine moet niet alleen de feitelijke overdracht van data regelen, maar moet ook kunnen initialiseren en de status en controle kunnen beheren.

K

Klaar voor data (ready for data). Een signaal dat actief is als de ontvanger klaar is om meer data te ontvangen.

Kladgeheugen. Een geheugengebied dat in het bijzonder geschikt is voor de opslag van variabelen of tussentijdse resultaten. In 6502 computers wordt pagina 0 meestal als kladgeheugen gebruikt.

Klok. Een regelmatig timing-signaal dat de signaal-overdrachten in systeem beheerst.

L

Laagstwaardig bit. Het meest rechtse bit in een groep bits. Dus bit 0 in een byte of een 16-bits woord.

Label. Een naam die is toegevoegd aan een instructie of statement in een programma en de geheugenlocatie aangeeft van de machinetaal-code of toewijzing die door van instructie of het statement is geproduceerd.

Lagere programmeertaal. Een computertaal waarbij elk statement in een enkele machinetaal-instructie wordt vertaald.

Latch. Een luik dat de inhoud bewaart tot er nieuwe data specifiek ingekomen zijn.

Leading Edge (van een binaire puls). De rand die het begin van een puls aangeeft.

Leen-bit (borrow). Een bit dat op 1 is gezet als een aftrekking een negatief resultaat oplevert en op 0 is gezet als het resultaat positief of nul is. Het leen-bit wordt veel gebruikt om getallen af te trekken die te lang zijn om in een enkelvoudige handeling te bewerken.

LIFO (last-in, first-out) geheugen. Een geheugen dat is geordend in de volgorde waarop de elementen er in werden opgeslagen. De elementen kunnen alleen in dezelfde volgorde worden teruggehaald als waarin zij werden opgeslagen. Zie ook Stack.

Lijst. Een geordende set gegevens.

Lijsthoofd (Queue header). Een set van geheugenlocaties die de huidige locatie en de status van een wachtlijst beschrijven.

Linearisatie. De wiskundige benadering van een functie door een rechte lijn tussen twee punten waarvan de waarden bekend zijn.

Logisch apparaat. Het input- of uitvoerapparaat waaraan een programma refereert. Het feitelijke of fysieke adres wordt bepaald door het logische apparaat in een I/O-apparatentabel op te zoeken. Deze tabel de feitelijke I/O-adressen (of beginadressen van I/O-aansturingsroutines) die corresponderen met het logische apparaatnummer.

Logisch device. Zie *Logisch apparaat.*

Logische schuiving. Een schuif-handeling waarbij nullen aan het eind worden toegevoegd wanneer de originele data geschoven worden.

Logische som. Een binaire som zonder overdrachten tussen bit-posities. Zie ook Checksum, EXCLUSIEVE OR-functie.

Longitudinal redundancy check (LCR). Zie *Checksum.*

Lopende bit-test. Een procedure waarbij een enkel 1-bit door elke bit-positie in een bepaald geheugengebied wordt verplaatst en getest wordt of het correct teruggelezen kan worden.

M

Machine-taal. De programmeertaal die de computer direct, zonder vertaling kan uitvoeren (numerieke conversies uitgezonderd).

Mark. De 1- of aan-status op een seriele data-communicatielijn.

Maskeerbare interrupt. Een interrupt die het systeem kan uitschakelen (onderdrukken).

Masker. Een bit-patroon dat een of meer bits uit een reeks bits isoleert.

Meerderheids-logica. Een logische functie die waar is als meer dan de helft van de inputs waar is.

Meertakenverwerking (multitasking). Gebruikt om meerdere taken tegelijkertijd uit te voeren. Gewoonlijk gebeurt dit door gedurende een bepaalde tijd aan een taak te werken en taken aan te houden die op invoer, uitvoer, de afronding van andere taken of externe gebeurtenissen wachten.

Micro-computer. Een computer die een micro-processor als centrale verwerkingseenheid heeft.

Micro-processor. Een complete centrale verwerkingseenheid voor een computer, bestaande uit een enkele of weinig geintegreerde circuits.

Mnemonisch symbool. Een naam die de feitelijke betekenis of het feitelijke doel waaraan zij refereert aanduidt.

Modem (Modulator/demodulator). Een apparaat dat een draaggolf-frequentie toevoegt of verwijdert en het daarmee mogelijk maakt om data te ontvangen of te versturen over een kanaal met een hogee frequentie.

Modulair programmeren. Een programmeer-techniek waarbij het programma als geheel is opgebouwd uit logisch afzonderlijke delen of modules.

Module. Een deel van een programma.

Monitor. Een programma waarmee de gebruiker programma's en data kan invoeren, programma's kan starten, de inhoud van de computer en registers kan bekijken en de randapparatuur van de computer kan gebruiken. Zie ook *Operating Systeem.*

Multifunctie device. Zie *Multifunctie-apparaat.*

Multifunctie-apparaat. Een apparaat dat meer dan een functie in een computer-systeem vervult. De term slaat in het algemeen op apparaten die geheugen, in- en uitvoer-poorten, timers enzovoorts bevatten. Dit kunnen bijvoorbeeld 6530, 6531 en 6532 apparaten zijn.

Multi-tasking. Zie *Meertakenverwerking.*

Murphy's Wet. De beroemde uitspraak dat „alles fout gaat dat fout kan gaan".

N

Naslag-tabel. Zie *Opzoek-tabel.*

Negatiefvlag. Zie *Tekenvlag.*

Negatieve logica. Een circuit waarbij een logische nul de actieve stand af AAN-stand vertegenwoordigt.

Negative edge. Een signaal dat in waarde van 1 naar 0 terugloopt.

Negativeren. Levert het tweecomplement (negatief) van een getal.

Negencomplement. Het resultaat van een bewerking waarbij een decimaal getal van een getal wordt afgetrokken dat een negen in elke cijfer-positie heeft.

Nesten. Het in een hierarchische manier opbouwen van programma's. Hierbij is het ene niveau in het andere vervat, en zo verder. Het nest-niveau is het aantal overdrachten van controle dat nodig is om een bepaald deel van een programma te bereiken, zonder naar een hoger deel van het programma terug te keren.

Nibble (of nybble). Een eenheid van vier bits. Een byte (acht bits) kan omschreven worden als een voorwerp dat bestaat uit een hoge nibble (de vier hoogstwaardige bits) en een lage nibble (de vier laagstwaardige bits).

Niet-maskeerbare interrupt. Een interrupt die niet uitgeschakeld (onderdrukt) kan worden binnen de CPU.

Niet-vluchtig geheugen. Een geheugen dat de inhoud niet verliest als de stroom wordt uitgeschakeld.

No-op. Zie *Bewerkingsloze opdracht.*

Normalisatie (van getallen). Het aanpassen van een getal aan een regulier of standaard formaat. Een typisch voorbeeld hiervan is het aanpassen van een binair deel zodat het hoogstwaardige bit 1 is.

Nulvlag. Een valg die 1 is als de laatste handeling nul als resultaat opleverde en 1 is als dit niet het geval was.

O

Object-code (of doelprogramma). Het programma dat het product is van een vertaalprogramma, bij voorbeeld een assembler. Meestal is het een machinetaalprogramma dat klaar is om te gebruiken.

Offset. Afstand van een beginadres of beginpunt.

Omwikkeling (wrap-around). Een vorm van organiseren waarbij het lijkt alsof de uiteinden op elkaar aansluiten. Een geheugengebied kent omwikkeling als de handelingen ervan uit gaan dat paginagrenzen doorlopend zijn.

Onafhankelijke modus (van een parallelle interface). Een werk-modus waarbij status- en controlesignalen die bij een parallelle I/O-poort horen, onafhankelijk van data-overdrachten door die poort, gebruikt kunnen worden.

Onderbreking. Zie *Interrupt*

Onderhoud (van programma's). Het vernieuwen en aanpassen van gebruikte programma's.

Onderloop. Zie *Underflow*

One-shot. Een apparaat dat als reactie op een puls-input, een puls-output geeft, waarvan de tijdsduur bekend is. Een timer werkt in de one-shot modus als deze het einde aangeeft van een enkele interval waarvan de lengte bekend is.

Oneindige lus. Een instructie die de controle aan zichzelf overdraagt en daardoor oneindig wordt uitgevoerd (of tot een hardware-signaal een interrupt geeft).

Oneven pariteit. Een 1-bit fout-opsporingscode die het totale aantal 1-bits in een data-eenheid (inclusief het pariteits-bit) oneven maakt.

Ongepakte decimale vorm. Een binair-gecodeerd decimaal formaat waarbij elke 8-bits byte slechts een decimaal cijfer bevat.

Ongetekend getal. Een getal waarvan alle bits gebruikt worden om grootte aan te geven.

Onmiddelijke adressering. Een adresseringsmodus waarbij de data die een instructie nodig heeft, onderdeel van de instructie is. De data staan onmiddelijk achter de handelingscode in het geheugen.

Onvertraagd. Zie *Real-time.*

Op-code. Zie *Handelingscode.*

Opdracht. Zie *Instructie.*

Openen. Een bestand gereed maken voor gebruik. Over het algemeen moet de gebruiker een bestand eerst openen om er mee te kunnen gaan werken.

Operating-systeem (OS). Een computerprogramma dat de algemene handelingen van een computer beheert. Het wijst bijvoorbeeld geheugenplaatsen voor programma's en data aan, bepaalt de volgorde waarin programma's worden uitgevoerd, handelt interrupts af en beheert het algemene in- en uitvoer-systeem. Ook wel monitor of hoofd-uitvoeringsprogramma genoemd, hoewel de term monitor over het algemeen gebruikt wordt voor een eenvoudig operating-systeem met beperkte functies.

Opzoek-tabel. Een data-array dat dusdanig georganiseerd is, dat het antwoord op een probleem bepaald kan worden door slechts het juiste element op te zoeken (zonder enige berekeningen).

OS. Zie *Operating-systeem.*

Output-register. De feitelijke input/output-poort in een PIA of VIA. Ook wel dataregister of randapparatuur-register genoemd.

Overdracht-bit. Een bit dat 1 is als een optelling in de volgende bit-positie overloopt.

Overdrachtvlag. Een vlag die 1 is als de laatste handeling een overdracht vanuit het hoogstwaardige bit genereerde. De vlag is 0 als dit niet het geval is.

Overloop (van een stack). De beschikbare hoeveelheid geheugenruimte die aan een stack is toegewezen, overschrijden.

P

P-register. Zie *Processor statusregister*

Packed decimal. Zie *Gecomprimeerde decimaal*

Pagina. Een onderdeel van het geheugen. Voor de 6502 geldt dat een pagina uit 256 bytes bestaat, waarin alle adressen dezelfde acht hoogstwaardige bits (of pagina-nummer) hebben. Pagina C6, bijvoorbeeld, bestaat uit de geheugenadressen C600 tot en met C6FF.

Pagina nul. In 6502-termen de laagste 256 geheugenadressen (de adressen 0000 tot en met 00FF).

Pagina-nul-adressering. In 6502-termen is dit een vorm van directe adressering waarbij de instructie slechts een 8-bits adres op pagina 0 bevat. dit betekent dat er zonder verdere aanduiding van uit wordt gegaan dat nul de hoogstwaardige byte van het directe adres is. Daarom hoeft deze niet in de instructie te worden opgenomen.

Pagina-nul-geindexeerde adressering. Een vorm van geindexeerde adressering waarbij de instructie een basisadres op pagina 0 bevat. Of wel, zonder verdere aanduiding wordt ervan uit gegaan dat nul de hoogstwaardige byte van het directe adres is. Daarom hoeft deze niet in de instructie te worden opgenomen.

Pagina-nummer. De identificatie-code die een specifieke geheugenpagina karakteriseert. Voor de 6502 geldt dat deze code wordt gevormd door de acht hoogstwaardige bits van een geheugenadres.

Parallelle interface. Een interface tussen een CPU en in- en uitvoer-apparaten die data parallel (meer dan een bit tegelijkertijd) kunnen verwerken.

Parameter. Een voorwerp of gegeven dat aan een subroutine of programma ter beschikking moet worden gesteld om hen goed te kunnen laten werken.

Parameters doorgeven. Het ter beschikking stellen van parameters aan een subroutine.

Pariteit. Een 1-bit fout-opsporingscode die het totale aantal 1-bits in een data-eenheid (inclusief het pariteits-bit) oneven (oneven pariteit) of even (even paraiteit) maakt. Ook verticale pariteit of verticale redundantiteitscontrole (VRC) genoemd.

Peripheral ready. Zie *Randapparaat klaar.*

PIA (Peripheral Interface Adapter). De meest gebruikte benaming voor het 6520 of 6820 apparaat, dat uit twee bidirectionele 8-bits I/O-poorten, twee statuslijnen en twee bidirectionele status- of controle-lijnen bestaat. De 6821 is een soortgelijk apparaat.

Planningsprogramma. Een programma dat bepaalt wanneer andere andere programma's gestart en beeindigd moeten worden.

Pointer. Een opslag-locatie die het adres van een gegeven bevat, in plaats van het gegeven zelf.

Polling interrupt system. Zie *Polsend interrupt systeem*

Polling. Zie *Polsen.*

Polsen. Vaststellen welke I/O-apparaten gereed zijn door de status van elk apparaat afzonderlijk te bekijken.

Polsend interrupt systeem. Een interrupt-systeem waarbij het programma de bron van een specifieke interrupt vaststelt door een voor een de mogelijke bronnen van een interrupt te onderzoeken.

Poort. De elementair adresseerbare eenheid van het in- en uitvoer-onderdeel van een computer.

Poppen. Verwijdert een operand van een stack.

Positive edge (van een binaire puls). Een signaal dat in waarde van 0 naar 1 oploopt.

Postdecrementing. Zie *Achteraf-verlagen.*

Postincrementing. Zie *Achteraf-verhogen.*

Postindexeren. Zie *Indirect geindexeerde adressering.*

Predecrementing. Zie *Vooraf-verlagen.*

Preincrementing. Zie *Vooraf-verhogen.*

Processor statusregister. Een register dat de huidige status van een computer definieert en vaak verscheidene bits bevat die interne condities aangeven. Dit register wordt ook wel conditiecode-register, vlagregister, statusregister en status-woord genoemd.

Programma-gerelateerd adresseren. Een vorm van relatief adresseren waarbij de programmateller het basisadres is. Het gebruik van deze vorm van programmeren maakt het makkelijk programma's van het ene naar het andere adres te verplaatsen.

Programmateller (PC of P register). Een register dat het adres bevat van de volgende instructie die gehaald moet worden.

Programmeerbaar I/O-apparaat. Een I/O-apparaat waarvan de werk-stand programmatisch vastgesteld kan worden door registers te laden.

Programmeerbare randapparatuur-chip. Een chip die in verscheidene modi kan werken; de huidige werk-stand kan programmatisch worden vastgesteld door registers te laden.

Programmeerbare timer. Een apparaat dat onder programma-controle verscheidene timing-taken verrichten, waaronder het creeren van pauzes.

Protocol. Zie *Gegevensverbinding-controle.*

Pseudo-bewerking (of pseudo-op of pseudo-instrctie). Een machinetaal-werkcode die de assembler vertelt een actie te ondernemen, maar niet resulteert in het genereren van een machinetaal-instructie.

Pseudo-operation. Zie *Pseudo-bewerking.*

Pullen. Verwijdert een operand van een stack, identiek aan pop.

Pushen. Slaat een operand op de stack op.

Q

Queue. Wachtlijst van een reeks taken, opslag-locaties of andere gegevens die op een first-in, first-out worden gebruikt. Of wel het gegeven dat het eerst in de queue is gezet wordt er ook het eerst weer uitgehaald.

Queue-header. Een set opslag-locaties die de huidige locatie en status van de queue weergeven.

R

RAM. Zie *Random-access memory.*

Randapparaat klaar. Een signaal dat actief is wanneer een randapparaat meer data kan ontvangen.

Randapparaat-interface. Een van de versies van parallelle interfaces uit de 6500-familie. Voorbeelden hiervan zijn de 6520, 6522, 6530 en 6532 apparaten.

Randapparaat-register. De feitelijke in- en output-poort van een PIA of VIA. Ook wel dataregister of output-register genoemd.

Random-access memory (RAM). Een geheugen dat met gewone handelingen zowel gelezen als gewijzigd (beschreven) kan worden.

Read-only memory (ROM). Een geheugen dat gelezen maar met normale handelingen niet beschreven kan worden.

Ready for data. Zie *Klaar voor data.*

Real-time. Synchroon met het feitelijke optreden van gebeurtenissen.

Real-time klok. Een apparaat dat met regelmatige tussenperioden een interrupt aan de CPU geeft.

Real-time operating-systeem. Een operating-systeem dat als beheerder kan dienen van programma's die real-time moeten draaien. Wordt ook wel real-time uitvoeringsprogramma of real-time monitor genoemd.

Reentrant. Een programma of routine met de mogelijkheid om gelijktijdig uitgevoerd kan worden, terwijl dezelfde routine een interrupt heeft gekregen of anderszins tijdelijk tot stoppen is gebracht.

Register. Een opslag-locatie binnen de CPU.

Rekenkundige schuiving. Een schuif-handeling die de waarde van het teken-bit (hoogstwaardige bit) bewaart. Bij een schuiving naar rechts wordt het teken-bit hierdoor in de volgende bit-posities geschoven (een zogenoemde teken-verlenging).

Rekenkundige-logische eenheid. Een apparaat dat verscheidene rekenkundige of logische functies kan uitvoeren; functie-inputs geven aan welke functie tijdens een specifieke cyclus wordt uitgevoerd.

Relatieve adressering. Een adresseringsmodus waarbij het in de instructie opgegeven adres de offset van basisadres is.

Relatieve offset. Het verschil tussen het feitelijk adres dat in een instructie wordt gebruikt en de huidige waarde van de programma-teller.

Relocatable. Zie *Verplaatsbaar.*

Return. Zie *Terugkeer.*

Rijen-volgorde. Het in lineair geheugen opslaan van elementen van multi-dimensionaal array, door de indexen vanaf de meest rechtse te veranderen. Dit houdt in dat, gesteld de elementen A (I,J,K) zijn en met A (0,0,0) beginnen, de volgorde A (0,0,0), A (0,0,1), ..., A (0,1,0), A (0,1,1), ... is. De tegenovergestelde techniek wordt kolommen-volgorde (column major order) genoemd.

RIOT. (ROM/I/O/timer of RAM/I/O/timer). Een apparaat dat geheugen (ROM of RAM), I/O-poorten en timers bevat.

ROM. Zie *Read-only memory.*

Rondschuifbewerking. Zie *Roteren.*

Roteren. Een schuif-bewerking data behandelt alsof zij in cirkel georganiseerd zijn. Dus alsof het hoogstwaardige bit en het laagstwaardige bit of wel direct of wel via het Overdracht-bit met elkaar verbonden zijn.

Row major order. Zie *Rijen-volgorde.*

RRIOT. ROM/RAM/I/O/timer. Een apparaat dat read-only memory, read/write memory, I/O-poorten en timers bevat.

RS-232 (of EIA RS-232). Een standaard interface voor de overdracht van serie~le digitale data. Deze standaard wordt ondersteund door de Electronic Industries Assocation of Washington, en is gedeeltelijk vervangen door RS-449.

S

Samenvoeging. Aan elkaar ketenen, aaneenrijgen of samenvoegen in reeksen. Bij het bewerken van strings betekent dit het plaatsen van de ene string achter de andere.

Scheduler. Zie *Planningsprogramma.*

Schuif-instructie. Een instructie die alle bits van de data een bepaald aantal bits verplaatst, net als bij een schuif-register.

Scratchpad. Zie *Kladgeheugen.*

SDLC (synchrone gegevensverbinding controle). De opvolger van het BSC-protocol voor IBM-computers en terminals.

Semafoor. Zie *Vlag.*

Serieel. Een bit per keer.

Seriele interface. Een interface tussen een CPU en in- of uitvoerapparaten die data serieel verwerken. Seriele interfaces die veel gebruikt worden in 6502-computers zijn de 6551 en 6850 devices. Zie ook *UART.*

Shift instructie. Zie *Schuif-opdracht.*

Software-stack. Een stack die door middel van doelgerichte instructies wordt bijgehouden. Dit in tegenstelling tot een hardware-stack, die automatisch door de computer wordt bijgehouden.

Software-interrupt. Zie *Zekering.*

Software-vertraging. Een programma dat geen andere functie heeft dan tijd te verspillen.

Source-code of *bron-programma.* Een computer-programma dat in assembler-taal of een hogere taal is geschreven.

Space. De nul-stand op een seriele datacommunicatie-lijn.

Spring-instructie (of vertak-instructie). Een instructie die een nieuwe waarde in de programma-teller stopt en zo de gewone enkelvoudige-stap verhoging verlaat. Spring-instructies kunnen voorwaardelijk zijn. Dit houdt in dat de nieuwe waarde alleen de programma-teller kan worden gezet als aan een bepaalde voorwaarde wordt voldaan.

Sprong-tabel. Een tabel die bestaat uit beginadressen van uitvoerbare routines, gebruikt om de controle aan een van hen over te dragen.

Staartvlag (van een wachtlijst). De locatie van het oudste voorwerp in een wachtlijst, of wel het eerst ingevoerde voorwerp.

Stack. Een geheugendeel dat alleen op een last-in, first-out wijze betreden kan worden. Dit houdt in dat data alleen aan de top van de stack toegevoegd of weggehaald kunnen worden. Nieuwe data worden boven op de oude data gezet en het weghalen van data betekent dat het onderliggende voorwerp de nieuwe top wordt.

Stack-pointer. Zie *Stack-wijzer.*

Stack-top. Het adres dat het gegeven bevat dat het laatst aan de stack werd toegevoegd.

Stack-wijzer. Een register dat het adres van de stack-top bevat. De stack-wijzer van de 6502 bevat het adres van de eerstvolgende beschikbare (lege) stack-locatie op pagina 1.

Standaard (of 8, 4, 2, 1) *BCD.* Een BCD weergave waarbij de bit-posities hetzelfde gewicht als bij gewone binaire getallen hebben.

Standaard teletypewriter. Een teletypewriter die asynchroon met een snelheid van 10 karakters per seconde werkt.

Stap-voor-stap. Zie *One-shot.*

Start-bit. Een 1-bit-signaal dat het begin aangeeft van een data-transmissie door een asynchroon apparaat.

Statische afwijking. Zie *Offset.*

Statische geheugentoewijzing. De toewijzing van vaste opslag-gebieden voor data en programma's. Dit in tegenstelling tot dynamische toewijzing, waarbij de opslaggebieden worden toegewezen op het moment dat ze nodig zijn.

Statusregister. Een register waarvan de inhoud de huidige status of werkstand van een apparaat aangeeft. Zie ook *Processor-statusregister*.

Statussignaal. Een signaal dat de huidige status van een overdracht of de werkstand van een apparaat aangeeft.

Stop-bit. Een 1-bit-signaal dat het einde van data-overdracht door een asynchroon apparaat aangeeft.

String. Een array (groep data) dat uit karakters bestaat.

String-functies. Procedures die het de gebruiker mogelijk maken om met data te werken die uit karakters in plaats van getallen bestaan. Typische functies zijn tussenvoeging, verwijdering, samenvoeging, zoeken en vervanging.

Strobe. Een signaal dat een andere groep signalen identificeert of omschrijft. Het kan gebruikt worden om een buffer, latch of register te beheren.

Stroomuitval-onderbreking. Een interrupt die de CPU over een dreigende stroomuitval informeert.

Subroutine. Een subprogramma dat vanuit meer dan een plaats in een hoofdprogramma uitgevoerd (aangeroepen) kan worden.

Subroutine-aanroep. Verplaatst de controle naar een subroutine, terwijl de informatie om het huidige programma te hervatten, bewaard blijft. Een aanroep verschilt van een spring- of vertak-instructie, omdat een aanroep informatie over zijn oorsprong bewaart. Spring- en vertak-instructies doen dit niet.

Subroutine-ketening. Het mechanisme waarbij de computer de informatie om het huidige programma te hervatten vasthoudt, nadat hij de uitvoering van een subroutine afgerond heeft.

Synchronisatie (of sync) *karakter*. Een karakter dat alleen wordt gebruikt om zender en ontvanger te synchroniseren.

Synchroon. Handelen volgens een algemene timings-bron of klok. Of wel met regelmatige tussen-perioden.

Systeem-programmatuur. Programma's die administratieve functies uitvoeren of assisteren bij het ontwikkelen van andere programmatuur, maar feitelijk niets van de uiteindelijke werkbelasting van de computer uitvoeren.

T

Taak. Zie *Task*.

Tail. Zie *Staartvlag*.

Task. Een zelfstandig programma dat, beheerd door een supervisie-programma, als onderdeel van een totaal-programma kan dienen.

Task-status. De groep paramaters die de huidige status van een task bepaalt. Een task kan aangehouden en hervat worden zolang de status ervan, bewaard en hersteld wordt.

Teken-functie. Een functie die 0 is als haar parameter positief is en 1 als haar parameter negatief is.

Tekenverlenging. Het proces waarbij het teken-bit (het hoogstwaardige bit) naar rechts gekopieerd wordt, als bij een rekenkundige schuiving. Tekenverlenging bewaart het teken wanneer tweecomplement getallen gedeeld of genormaliseerd worden.

Tekenvlag. Een vlag die het hoogstwaardige bit van het resultaat van de vorige bewerking bevat. Ze wordt soms een negatiefvlag genoemd, omdat een waarde van 1 een negatief getekend getal aangeeft.

Teletypewriter. Een apparaat dat een toetsenbord en een seriele printer bevat en dat vaak gebruikt wordt bij communicatie en bij computers. Ook wel een Teletype (een geregistreerd handelsmerk van Teletype Corporation in Skokie, Illinois) of TTY genoemd.

Terugkeer (vanuit een subroutine). Draagt de controle weer over aan het programma dat de subroutine oorspronkelijk aanriep en hervat de uitvoering ervan.

Threaded code. Zie *Geregen programma*.

Tiencomplement. Het resultaat dat wordt verkregen als een decimaal getal van nul wordt afgetrokken (waarbij het negatieve teken genegeerd wordt), het negencomplement plus een.

Time-out. Zie *Blokkeertijd*.

Trace. Een hulpmiddel om fouten op te sporen. Geeft informatie over een programma, terwijl het programma wordt uitgevoerd. De trace print gewoonlijk alle of een deel van de tussentijdse resultaten.

Trailing Edge (van een binaire puls). De kant die het einde van een puls aangeeft.

Trap. Zie *Zekering*.

Tweecomplement. Een binair getal dat, wanneer het binair bij het oorspronkelijke getal wordt opgeteld, nul als resultaat geeft. Een tweecomplement van een getal kan worden verkregen door het getal van nul af te trekken of door 1 bij het eencomplement op te tellen.

Tweecomplement-overloop. Een situatie waarin een getekende rekenkundige handeling een resultaat oplevert dat niet corerct kan worden weergegeven. Of wel, de grootte loopt over in het teken-bit.

U

UART (Universal Asynchronous Receiver/Transmitter). Een LSI-apparaat dat als interface dient tussen systemen die data in parallelle vorm en apparaten die data in asynchrone serie~le vorm verwerken.

Uitschakelen. Verbiedt de doorgang van een activiteit of de herkenning van een signaal (zoals een interrupt).

Underflow of *onderloop* (van een stack). De poging meer data uit de stack te halen dan er in gestopt is.

Utility. Zie *Functie-programma.*

V

Valid data. Een signaal dat actief is wanneer nieuwe data beschikbaar zijn voor de ontvanger.

Vectored interrupt. Zie *Gerichte interrupt.*

Verplaatsbaar. Kan zonder wijzigingen overal in het geheugen geplaatst worden. Dit betekent dat een programma elke willekeurige groep opeenvolgende geheugenlocaties kan bezetten.

Versatile Interface Adapter (VIA). De naam die veel aan het 6522 parallelle interface apparaat wordt gegeven. Het bestaat uit twee 8-bits bidirectionele I/O-poorten, vier status- en controlelijnen, twee 16-bits timers en een schuifregister.

Vertaal-instructie. Een instructie die de operand vertaalt in het corresponderende gegeven in een tabel.

VIA. Zie *Versatile Interface Adapter.*

Vlag (of conditie-code of status-bit). Een enkel bit die een toestand binnen de computer aangeeft en dat veel wordt gebruikt om tussen alternatieve instructie-reeksen te kiezen.

Vlag (software). Een indicator die of aan (1) of uit (0) is en kan worden gebruikt om tussen twee alternatieve handelwijzen te kiezen. Booleaanse variabele en semafoor zijn andere benamingen met dezelfde betekenis.

Vlagregister. Zie *Processor statusregister.*

Vluchtig geheugen. Een geheugen dat de inhoud verliest wanneer de stroom wordt uitgezet.

Vooraf-verhogen. Verhoogt een adresregister voordat het wordt gebruikt.

Vooraf-verlagen. Verlaagt een adresregister voordat het wordt gebruikt.

Voorafindexering. Zie *Geindexeerd indirecte adressering.*

Vrije-loop. Een werkstand van een timer waarbij deze het einde van een interval aangeeft en vervolgens een van dezelfde lengte begint. Wordt ook wel doorlopende modus genoemd.

Vullen. Het plaatsen van waarden in geheugenlocaties die nog niet eerder waren gebruikt, het initialiseren van geheugen of opslag.

W

Wachtrij. Zie *Queue.*

Walking bit-test. Zie *Lopende bit-test.*

Wijzer. Zie *Pointer.*

Wissen. Op nul zetten.

Woord. De basis-groepering van bits die de computer in een keer verwerken. Bij microprocessoren refereert deze term vaak aan een 16-bits data-eenheid.

Woord-grens. Een grens tussen 16-bits opslag-eenheden die twee bytes informatie bevatten. Als informatie in eenheden ter grootte van een woord is opgeslagen, bevatten alleen de bytes die conform (aangepast aan) woordgrenzen zijn geldige
informatie. Niet-aangepaste byte-paren bevatten een byte van het ene en een byte van het andere woord.

Woord-lengte. Een lengte van 16 bits per voorwerp.

Wrap-around. Zie *Omwikkeling.*

Write-only register. Een register dat de CPU kan wijzigen maar niet kan lezen. Als het programma de inhoud van een dergelijk register moet vaststellen, moet het een kopie van de daar gezette data bewaren.

Z

Zekering (of software-interrupt). Een instructie die een sprong naar een bepaald (CPU-afhankelijk) adres forceert, vaak gebruikt om breakpoints (onderbrekingspunten) te produceren of om hardware- of software-fouten aan te geven.

Zelfherstellende code. Een code die de ontvanger kan gebruiken om fouten in berichten op te sporen. De code zelf bevat gaan extra bericht.

Zoned decimal. Zie *Ongepakte decimaal.*

Index

A

A register. zie Accumulator
Aanhalingstekens rond ASCII strings, 10
Aanpassinginstructies, 130
Absolute (directe) adressering, 17-18
    instructies, 17
    volgorde adresbytes, 14
Absolute geindexeerde adressering, 18-19, 20, 21
    beperking (tot 256-byte arrays), 148
    instructies, 17
    volgorde adresbytes, 14
Absolute waarde (16-bit), 92-93, 176-177, 244-245
Accumulator (register A), 11, 12, 15
    beslissing reeksen, 35
    decimale operaties, 79-88
    functies, 15
    instructies, 16
    testen, 100-101
    verhoog met 1, 13, 84-85
    verlaag met 1, 13, 86
    verwisseling met top stack, 102-103
Actieve transitie in een 6522 VIA, 63-64, 66
ADC, 11-12, 24, 25, 26, 142, 143
    decimale mode, 12
    overdrachtvlag,
    resultaat, 142
    uitsluiting van,
    verhoog met 1, 12-13
    vlaggen, 12, 142
Adres arrays, 41-42, 43-45
Adres indeling in geheugen (achterstevoren), 14, 148
Adresseer modes,
    absoluut (direct), 19-20, 23, 148
    absoluut geindexeerd, 20-21, 22, 24, 154
    auto indexering, 135-138
    direct, 16, 17, 19, 21, 22-23, 154
    geindexeerd, 18, 20-21, 22, 133-135
    geindexeerd indirect (pre indexering), 11-12, 18
    indirect, 11-12
    indirect geindexeerd (post indexering), 11-12, 18
    onmiddelijk,
    post indexering, 11-12, 18
    pre indexering, 11-12, 18
    overzicht,
    pagina 0 (direct), 16, 20, 23
    pagina 0 geindexeerd, onmiddelijk, 17, 20-21
    systeem gekozen waarde (absoluut direct), 9, 17
    6502 terminologie, 11
    Afkortingen, herkenning van, 253, 358, 359
Aftrekinstructies,
    omgekeerd, 83-84
    met lenen, 84-85

zonder lenen, 82-83
Aftrekking, 81-83, 287-290
BCD, 12
binaire, 11-12, 81-83
8-bit, 11-12, 25, 81-83
16-bit, 81-83, 234-236
decimaal, 12, 81-83
eerst overdracht zetten, 11-12, 25
geinverteerde leen, 11-12, 33-34, 142-143
meervoudige precisie, 47-48, 259-261
omgekeerde, 83-84
overdrachtvlag, 11-12, 142-143
AND,
bits testen, 30-31
bits wissen, 26-27
invoer instructie, 58
maskeren, 61-62, 343-344, 353-354
Apostrofe ASCII karakter aanduidend, 10
Apparaatnummers, 60-61, 453
Arrays, 38-43, 135-137, 195-198, 391-394
adressen, 40-41, 44-47, 427-429
grote (meer dan 256 bytes), 40-42
manipulatie, 38-43
variabel basisadres, 39-44
initialisatie, 195-198
ASCII naar EBCDIC conversie, 189-191
ASCII, 523
assembler notatie, 9-10
conversies, 170-194
tabel, 523
ASL, 31, 42, 58
Assembler,
fout herkenning, 149-150
formaat, 9-10
pseudo operaties, 525
systeem gekozen waarde, 150-151,157
Asynchronous Communications Interface
Adapter (ACIA), 62-63,473-474,479-487,497-507
Auto indexering, 135-138
Automatisch achteraf verhogen, 136
Automatisch achteraf verlagen, 137
Automatisch vooraf verhogen, 135
Automatisch vooraf verlagen, 136

B

B (BREAK) vlag, 9
B (binair getal aanwijzend), 10
BCC, 33-34,35,36
BCD (decimale) rekenkunde, 12,79-86,151-152,283-286
BCD naar binair conversie, 168-169
BCS, 33-35,36,37
BEQ, 33,34,143-144
BIT,33,142,147-148
adressen, 12-13,28-29,131-132
invoer instructie, 58,158-159

vlaggen, 12-13,144
.BLOCK pseudo operand, 9
BMI, 12-13,33-34,146
BNE, 12-13,30,32,146-147
BPL, 12-13,31,146-147
BRK, 526
BSC protocol, 447-452
BVC, 12-13,130
BVS, 12-13,147
.BYTE pesudo operand, 9
Basisadres van een array of tabel, 20,21,22,26-27
Baudrates, veel gebruikte, 539
Bepaal origine (.ORG of *=) pseudo-operatie, 10
Beslissingreeksen, 35-36
Bidirectionele poorten, 158,424,425
Binair naar BCD conversie, 165,167
Binaire zoekmethode, 407-413
Binary-coded-decimal (BCD), 12,150
Bit manipulatie,26-29,94-98,310-311
Bit veld extractie, 310-322
Bit veld toevoeging, 323-327
Blok vergelijking, 9,353-357
Blok verplaatsing, 199-204
Boole algebra, 26-27
Bovenkant van stack, 16
Break (B) vlag, 9
Bubble sort, 414-418

C

Case opdrachten, 45
Cirkelverschuiving (rotatie), 27,30,99-100,343-347
CLC, 11-12,47-48
CLD, 12,76,80 zie ook Decimale mode vlag
CLI, 14,31
CLV, 130
CMP, 143-144
gebruik van, 31-33
invoer instructie, 58
Nul vlag, 31-32
Overdracht vlag, 12-13,31-32,142-143
Overloop vlag (geen effect), 146
SBC, verschil met, 24-25
Code conversie, 46-47
Combo chips, 62
Commando register, 158-159 zie ook Controle register
Commentaar, 9
Communicatie tussen hoofdprogramma en interrupt afhandelingsroutines, 159-479-480,488-489,497-498
Complementeren (inverteren) van bits, 26,28,96-97
Complementeren (logische NIET) instructies, 96-97
Complementeren Overdracht vlag, 97
Complementeren van accumulator (EOR #$FF), 26,96
Comprimeren van strings, 407
Concatenatie van strings, 180-181,358-363
Conditie code, zie Vlaggen, Status Register,

Conditionele oproep instructies, 126-127
Conditionele terugkeer instructies, 127
Conditionele vertakkingsinstructies, 36-37,109-125
uitvoeringstijd (variabel), 523,524
paginagrens, 523,524
Controle lijnen op 6522 VIA, 165-170
Controle register, 6522 VIA, 62,159-160
Controle signaal,61-61
Controlegetal, 96 zie ook Pariteit
Copieren substring, 370-376
CPX, 36,77,142-143
CPY, 36,77,142-143
CRC (cyclic redundancy check),447-452

D

D (decimale mode) vlag, 12,74,527
.DBYTE pseudo operand, 9
Data Directie register (DDR), 63,65
6520 PIA, 469-470
6522 VIA, 63,470
Data transfer instructies, 101-105
Debugging, 141-160
interrupt afhandeling routines, 159-160
Invoer/Uitvoer besturingsroutines, 158-160
DEC,
beslissingsreeksen, 35-36
complementeren bit 0, 27,96-97
decimale mode, 13
Overdracht vlag (geen effect), 144
uitvoer instructie, 58
wissen bit 0, 27
Decimale (BCD) rekenkunde, 283-286
aftrekking, 287-290
binaire conversies, 165-167
8-bit, 79-86
deling, 298-306
multibyte, 283-307
optelling, 283-286
vergelijkingen, 307
verhoog met 1, 85-86
verlaag met 1, 86-87
vermenigvuldiging, 291-297
vlaggen, 12-13
waarmerking, 130
Decimale mode (D) vlag, 12,74,527
betekenis, 12
CLD, 12,74,80
gebruik, 12-12
initialisatie, 12-13,152
interrupt afhandeling routines, 76-77,152,159-160
positie in status register, 9,527
reset (geen effect), 12-13
redden en herstellen, 12-13,80-81
SED, 76,151
systeemgekozen waarde in meeste computers, 12,152

testen, 112,114
Decrement instructies, 86-87
accumulator, 12-13
16-bit getal, 38,86-87,144
Deling, 89-90
16-bit, 241-249
door 2, 89-90
door 4, 89-90
door 10, 166-167
door 100, 166-167
decimale, 298-306
eenvoudige voorbeelden, 89-90
meervoudige precisie binair, 270-277
Directe adressering,
absolute versie, 19-20,148
gebruik van -, 19-20
onmiddelijke adressering, verschil met, 148
pagina 0 versie, 16
6502 terminologie, 20
Disassembly van numerieke operatie codes, 524
Documentatie van programma's, 32,45
Dollarteken voor hexadecimale getallen, 9,148
Doorgeven parameters, 52-56,161-162
geheugen, 52-56
registers, 52-53
stack, 54
Drempel controle, 30,32-33
Dubbele punt (optioneel scheidingsteken na label), 9
Dynamisch toewijzen van geheugen, 54-55,75-76

E

EBCDIC naar ASCII conversie, 192-194
Een complement, 97-98 zie ook EOR
Een-dimensionele arrays, 205-207
Eindeloze lus instructie, 130-131
.END pseudo operand, 9
EOR, 96-97
bits inverteren, 97
complementeren accumulator (EOR #$FF), 25,97
vergelijking (bit-voor-bit), 96
logische som, 97
.EQU pseudo operand, 9
Even pariteit, 441-443
Exclusief-OR functie, 25 zie ook EOR,

F

F (vlag) register, zie ook Vlaggen, Status register
FIFO buffer (wachtrij), 51-52,497
Formaat fouten, 149-151
Formaat voor opslag van 16-bit adressen, 14
Fout afhandeling, 162
Fout-corrigerende codes, zie CRC,
Fout-detecterende codes, zie Pariteit,

Fouten in programma's, 141-160
Fysiek Invoer/Uitvoer apparaat, 60-61

G

Gebufferde interrupts, 497-507
Geheugen gebruik, beperking van, 77-78
Geheugen test, 419-426
Geheugen vullen, 105,195-198
Geindexeerd indirect adressering (preindexing), 21,41,60-61,138,148
    fouten, 61,148
    gebruik, 41
    uitlijnen woorden, 148
    instructies, 19
    restricties, 21
    uitsluitend even indices, 21
    uitbreiding, 138
    wraparound op pagina 0, 61,138
Geindexeerde adressering,
    absolute versie, 20-22,23
    16-bit index, 43-44
    fouten in gebruik, 142
    gebruik van, 38-39
    geindexeerd indirect (preindexed) versie, 21,41,60-61,138
    indirect geindexeerd (postinexed) versie,21,41,138
    offset van 1 in basisadres, 39
    opzoektabellen, 43
    pagina 0 versie, 18,20-22
    subroutine oproepen, 44-46,427-430
Geindexeerde sprong, 44-46,109,427-430
Geinverteerde beslissingslogica, 141,143,144-145
Geinverteerde leen in aftrekking, 11,33
Gelijke waardes, vergelijking, 33-34,144
Geneste lussen, 37-38
Getal (4-bit) verschuiving, 89,304
Getallen met tekens, 34

H

H (hexadecimaal getal aangevend), 10,150
Haakjes rond adressen (indirectie aangevend), 9
Handmatige uitvoermode van 6522 VIA, 66-70
Handshake, 65-70
Herstartbaarheid, 53,55-56,75-76
Hexadecimaal ASCII naar binair conversie, 171-181
Hexadecimaal naar ASCII conversie, 173-175

I

I vlag, zie Interrupt Disable vlag
INC,
    bit 0 zetten, 27
    16 bit increment, 85,86

complementeer bit 0, 27
decimale versie, 86
overdracht vlag (geen effect), 144-145
uitvoer instructie, 58
Implementatie fout, (indirecte sprong op pagina grens), 157
Impliciete effecten van instructies, 154-155
Increment instructies, 85-86
accumulator, 85-86
16-bit getal, 83-86,144-145
Index registers,
beslissingsreeksen, 36
CPX,CPY, 36,77,144
gebruik van, 15-16
instructies, 16-17
LDX,LDY, 19-20
lengte, 13-14
opzoeken in tabel, 43-45
speciale eigenschappen, 15-16
STX,STY, 22-23
testen, 100
transfers, 104
van stack laden, 22
verschillen tussen X en Y, 15,19
verwisselen, 106
Indexering bij arrays, 38-43,48-49,205,228
byte arrays, 205-207,211-215
een-dimensioneel byte array, 205-207
een-dimensioneel word arrays, 208-210
multidimensionele arrays, 48-49,211-215
twee-dimensioneel byte array, 216-220
twee-dimensioneel word array, 221-229
word arrays, 208-210,221-229
Indirect geindexeerde adressering (postindexing), 12,14,40-43,50-52,148-149
instructies, 19
lange arrays, 41-42
restricties, 21
uitbreiden, 138
variabel basisadres, 43-45,50-52
Indirecte adressering, 50,102,108-109,131
absolute versie (uitsluitend JMP), 12,148-149
geindexeeerd indirect (preindexing), 21-22, 41,60-61,138
indirect geindexeerd (postindexing), 21-22,41,138
JMP, 12,149
simulatie met 0 in een index register, 12,102,131-133
subroutine oproepen, 44-45,108-109,126-127
Indirecte oproep, 126-127
Indirecte sprong, 44-46,109,126-127,459,460
fout op pagina grens, 157
Initialisatie fouten, 155
Initialisatie,
6850 ACIA, 474,484-485
arrays, 195-198
decimale mode vlag, 12,155,160
indirecte adressen, 24,103
interrupt systeem, 470,484-485,488-489,493-494
Invoer/Uitvoer apparatuur, 469-474
RAM, 23-24,195-198
status register, 102-103

stack wijzer, 102-103
6522 VIA, 63-71
wijzer op pagina 0, 24,103
Instructie uitvoeringstijden, 523-524
Instructieset,
alfabetische lijst, 523-524
numerieke lijst, 524
Interpolatie in tabellen, 78
Interrupt Disable (I) vlag,
betekenis, 14-15
CLI, 14-15,131
interrupt accepteren, 73
positie in status register, 74-75
redden en herstellen, 65,131
RTI, 74,131
SEI, 14-15,75,131
toestand in stack, 74,75
testen, 112,114
verandering in de stack,74
Interrupt afhandelingsroutines, 479-487,488-496,497-507
6850 ACIA, 479-487,497-507
fouten, 159-160
hoofdprogramma, communicatie met, 160,479-480,488-489,497-498
programmeerrichtlijnen, 73-74
real-time klok, 508-522
6522 VIA, 488-496
voorbeelden, 479-521
Interrupt enable register (in 6522 VIA), 71-72,493-494,534
Interrupt responsie, 73-74,526
Interrupt status,
6502 CPU, 73-74,131
redden en herstellen, 75,131
verandering in stack, 74-75
6522 VIA, 71-733,494,534
Interrupt vlag registers (in 6522 VIA), 67,68,71-74,494,534
Interrupts opnieuw toestaan, 75-76,131
Interrupts. zie ook Toestaan en verbieden van interrupts,
accepteren, 73-75,526
6850 ACIA, 479-487,497-507
afhandelingsroutines, 479-522
gebufferde, 497-507
handshake, 479-487
opnieuw toestaan, 75-76,131
programmeerrichtlijnen, 74-76
real-time klok, 508-522
response, 73-74
verlopen tijd, 508-522
6522 VIA, 71-73,493,534
vlaggen (6522 VIA), 71-73,493,534
volgorde in stack, 74
Inverteren van bits, 26-27,97
Invoer/Uitvoer apparaat tabel, 60-61,453-468
Invoer/Uitvoer controle blok (IOCB), 453-468
Invoer/Uitvoer,
6850 ACIA, 479-487,497-507
apparaat onafhankelijk, 453-474
apparaat tabel, 60-61,453-468
controle blok (IOCB), 453-468

fouten, 158-159
fysieke apparaten, 60-61
initialisatie, 469-474
instructies, 58-60
interrupt gestuurd, 479-487
lees-poorten, 58-60
logische apparaten, 60-61
peripheral chips, 62-63
status en controle, 61-62
terminal afhandelaar, 431-438
uitvoer,algemeen, 431-438
verschillen tussen invoer en uitvoer,438-440
6522 VIA, 63-73,488-496

J

JMP, 12,15,148
absolute adressering, betekenis van, 148
adresseringsmodes, 148
indirecte adressering, 44-45
paginagrens, fout bij (indirect), 157
JSR, 13
adresseringsmodes, betekenis van, 148
offset van 1 in terugkeeradres, 53-54
operatie, 526
terugkeeradres, 12
variabele adressen, 427-430

K

Kalender, 508-522
Karakter manipulatie, 45 zie ook String manipulatie
Kettinglijst, 49-52,454,455,460-461
Kladblok (pagina 0), 16

L

LDA, 13,20,21,31
LDX (LDY), 19,20
LSR, 28,29,58
Laad instructies, 101-103
adresseer beperkingen, 20
vlaggen, 12,31
Lange arrays (meer dan 256 bytes), 13,41-43,153
volle pagina's apart, 195,197
Leen, 11-12,32-33
Leespoorten, 58-60
Lijst verwerking, 49-51,460-461
Limiet controle, 30,32-34
Logische Invoer/Uitvoer apparaten, 60-61,453,454
Logische instructies, 94-101
Logische som, 95
Logische verschuiving, 27,28-29,58,98-99,333-342

Lussen, 37-38
reoganisatie voor tijdsbesparing, 76-77

M

Maskeren van bits, 61-62
Maximum, 399-402
Meervoudige precisie rekenkunde, 47,231-307
Meervoudige precisie verschuivingen, 328-352
getal (4-bit), verschuiving links, 304
logisch links, 334-337
logisch rechts, 338-342
rekenkundig rechts, 328-332
roteer links, 348-357
roteer rechts, 343-347
Meervoudige vertakkingen (sprongtabel), 43-44,427-429
6800 microprocessor, verschillen met 6502, 15,143
6809 microprocessor, verschillen met 6502, 15,95,96,143
8080/8085 microprocessors, verschillen met 6502, 15,143
Milliseconde vertragingsprogramma, 475-477
Minimum byte lengte element, 403-406
Missende instructies, 14,79-131
Multibit verschuivingen, 27-28
Multibyte plaatsingen in arrays of tabellen, 40,42-45
Multidimensionele arrays, 221-229
6530 Multifunction Device (RRIOT), 473
6532 Multifunction Device (RIOT), 473

N

N vlag zie Negatief vlag,
NOP, vullen met, 198
NOT instructies, 97-98
Negatief vlag,
BIT, 14,32,145
decimale mode, 12
instructies, effect van, 523-524
laad instructies, 14
opslag instructies (geen effect), 14
positie in status register, 8,527
SBC, 141
vergelijkingen, 144-145
vertakkingen, 33-36
Negatief, calculatie van, 92-93,245
Negatieve logica, 92-93,245
Negen complement, 93
Nibble (4 bits), 167-169
Normalisatie, 99-101
Nul vlag,
betekenis, 143
CMP, 31-33
decimale mode, 14
gebruik van, 30-31,35,36
INC, 38,144
inversie bij maskeren, 30,94-95

laad instructies, 14,31
maskeren, 30-31
positie in status register, 8,527
vertakkingen, 35-36
Numerieke vergelijkingen, 32-36
Nummerteken (geeft directe adressering aan), 10

O

ORA, 26,27,95-96,310,326
ORG (*=) pseudo operand, 9
Onafhankelijke mode van 6522 VIA controle lijnen, 67-68,71,72
Onderkast ASCII letters, 187-188
Onderloop van stack, 52,91
Oneven pariteit, 444
Onmiddelijke adressering, 9
assembler notatie, 10
directe adressering, verschil met, 148
gebruik van, 20
(gebrek aan) opslag instructies, 22
Ontvangen van interrupt, 73-74,526
Operatie (op) codes, 523-524
alfabetische volgorde, 523-524
numerieke volgorde, 524
Oproep instructies, 126-127 zie ook JSR
Opslag instructie,
effect op vlaggen (geen), 13,144
Optelinstructies, 80-82
met overdracht, 80-81
zonder overdracht, 81-82
Optelling,
BCD, 12,13,80-82,86-87,283-286
binair,12,24-26,47-48,80-81,255-262
8-bit, 12,24-26,80-82,85,86-87,284-286
16-bit, 81,82,86,231,233
decimaal, 12,24-26,47-48,283-286
meervoudige precisie, 47-48
Opzoek tabellen,
Opzoektabel,
Overdracht (C) vlag,
aftrekken van accumulator, 82-83
aftrekking, 12
beinvloedende instructies, 146
betekenis, 12
complementeren, 97
CLC, 12,47-48
decimale rekenkunde, 12-13
geinverteerde leen, 12,142
meervoudige precisie rekenkunde, 47
optellen bij accumulator, 80,81
positie in status register, 8,527
rekenkundige toepassingen, 12,47-48
SBC, 12
SEC, 12,80
vergelijkingsinstructies, 12,31-33
verlaag instructies (geen effect), 144

verhoog instructies (geen effect), 144
verschuivingen, 27-28
vertakking, 35-36
Overloop (V) vlag,14,32,147
beinvloedende instructies, 14,145
BIT, 14,31-32,147
CLI, 130
gebruik van, 32,34-35
Kies Overloop invoer, 130
positie in status register, 8,527
vertakkingen, 36
Overloop van de stack, 52,113-114,165
Overloop,twee complement, 34,118-120,144-145,146

P

P (processor status) register, 527
zie ook Vlaggen, Statusregister
Pagina 0,adresseer modes,
direct, 17,20
geindexeerd, 18
instructies, 17
Pagina 0,speciale eigenschappen, 16
Pagina grens passeren, 13,41-42
fout in indirecte sprong, 157
voorbeeld, 152-153
Parallel/serieel conversie, 27,58,59
Parameters doorgeven, 53-54,162-162
Pariteit, 441-446
controle, 441-443
even, 441-444
genereren, 444-446
oneven, 444
PC register zie ook Programma Teller 527
Percentage teken (geeft binair getal aan), 10,149
Peripheral Interface Adapter (6520 PIA), 62,63,159,472-473
Peripheral Ready signaal, 66-70
PHA, 22,54,55,56,103,128
PHP, 75,104,130,131
PIA (6520 Peripheral Interface Adapter), 62,159,472-473
PLA, 22,53-54,56,74,104,129
PLP, 22,75,103
Pop instructies, 129
Positie van substring, 364-369
Postindexing (indirect geindexeerd adresseren), 12,14,18,21,41-43,138,148
Preindexing (geindexeerd indirect adresseren), 18,21,60-61,138,148
Programma teller, 5,27
JSR, 12,148,526
RTS,12,45-46,526
Programmeerbare randapparatuur, 62-63
6850 ACIA,479-487,497-507
initialisatie, 469-474
operatiemodes, 62
6522 VIA, 63-73,488-496
voordelen van, 62
Programmeermodel voor 6502 microprocessor, 527

Pseudo-operaties, 9,525
puntkomma, geeft commentaar aan, 9
Push instructies, 128-129

R

RAM,
    data wegschrijven, 22-23
    initialisatie, 23-24
    testen, 419-426
    vullen, 195-198
RIOT, 62
ROL, 27,28,29,58
ROM,(lees-geheugen), 58,149
ROR, 27,28,29,58
RTI, 74,526
RTS, 13,108
    geindexeerde sprong, 45
    operatie, 526
    1 optellen bij opgeslagen adres, 12,45
Ready vlag (gebruik bij interrupts), 479,488
Real-time klok, 508-522
Redden en herstellen van D vlag, 75,131
Redden en herstellen van interrupt status, 74,131
Redden en herstellen van registers, 74,129-130
Register transfers, 19,104,106
    vlaggen, 12
Registers, 8-9,15-23,527
    doorgeven parameters, 8
    functies, 15
    instructies, 16
    lengte, 8-9,15-23,527
    programmeermodel, 527
    redden en herstellen, 128-130
    speciale eigenschappen, 15,19
    transfers, 19
    volgorde in stack, 73-74,128
Rekenkunde, 231-307
    BCD, 12,283-307
    binair, 12,24-26,47-48,231-282
    8-bit, 12,24-26
    16-bit, 231-253
    decimaal, 12,283-307
    meervoudige precisie, 47-48
Rekenkundige schuifbewerking, 29,83-84,98,328-332
Reset,
    decimale mode vlag (geen effect), 12
    6522 VIA, 65
Richting van groei op de stack, 14,22-23,526
Rotatie (cirkel-verschuiving), 27-28,29,99-100,343-352

S

S register zie Stack wijzer,
SBC, 12,25,143

CMP, verschil met, 25
decimale mode, 12,87
verlaag accumulator met 1, 12,87
operatie, 12,43
Overdrachtsvlag, 12,143
SEC, 12,82
SEI, 15,75,131
SP register, zie Stack wijzer,
Schrijf-poorten, 58-62,158,159,160
Serieel/Parallel omzetting, 26,72
Seriele invoer/uitvoer, 27,62,479-489,497-507
Set Overflow invoer, 130
Software stack, 52
Software vertraging, 475-477
Sommeren,
binair, 39,391-398
8-bit, 391-394
16-bit, 395-398
Sorteren, 414-418
Speciale eigenschappen van de 6502, overzicht van, 11-16
Sprongtabel, 43-46,158,427-430
implementeringen, 149
Staart van een wachtrij,497-498
Stack transfers, 14,22
Stack wijzer,
automatische verandering door gebruik, 14,22
definitie, 14
dynamische toewijzing van geheugen, 55-56
grootte van wijziging, 154
inhoud, 14
laden, 19,102
opslag, 19,102
pagina nummer (1), 12
reductie, 55-56
transfers, 102
vergelijking, 90-91
verhogen, 85-86
verlagen, 87
volgende beschikbare adres, 14
Stack,
beperking tot 256 bytes, 12,13,14,22-23
bereik door indexering, 55
data transfers, 14,22-23
doorgeven parameters, 55-57
groei omlaag, 45
onderloop, 52
overloop, 52
pagina 1, locatie op, 12,22-23
PHA, 22-23,55-56,74,103,128
PHP, 74,103-104,130-131
PLA, 22-24,53,54,56,74,104,129
PLP, 22-23,75,103
redden registers, 22
sofware, 52
veranderen van waarden, 74-75
wijzer, 14
onderloop,
Start van een wachtrij, 51-52,497-498

Status bit, zie vlaggen Status register
Status register,
definitie, 8-9,527
laden, 15-16,102-103
ongebruikt bit, 8-9
opslag, 15-16,103-104
organisatie, 8-9,527
transfers naar of van accumulator, 103-104
verandering in stack, 102-103
wijziging, 74-75
Status signalen, 61-61
String operatie, 45,353-390
afkorting, herkenning van, 354,364,365
comprimeren, 406
concateneren, 358-363
copieren van een substring, 370-376
positie substring, 364-369
tussenvoegen, 383-390
vergelijken, 353-357
verwijdering, 377-382
zoeken, 45-46
Strobe van 6522 VIA, 66-67,69
Subroutine oproep, 12,125-126 zie ook JSR,
variabele adressen, 125-126
Subroutine verbindingen, 12,525
Subscript, grootte van, 16,62,212,217,221
Systeem gekozen waardes voor assembler, 149-150,157
Systeen programma's, conflict met, 142

T

Tabel, 43-44,77-78,189-191
Teken extensie, 30,90-93,94,328-332
Teken functie,93-94
Teken vertakkingen, 117-119
Teken vlag, zie Negatief vlag
Terminal Invoer/Uitvoer, 431-437
Terugkeer instructies, 127-128 zie ook RTS
Terugkeer met overslaan van instructies, 127
Testen, 100-101
16-bit getal, 6,51-52,96,100-101
bits, 26,30-32,35-36,100-101
bytes, 30-36,99-100
meervoudige precisie getal, 273-274,302
TEXT pseudo operatie, 9
Threaded code, 93
Tien complement,
Tijdsbepaling voor instructies, 523,524
Tijdschriften gespecialiseerd in de 6502 microprocessor, 78
Tijdsoverschrijding, 475-477
Toestaan en verbieden van interrupts,
accepteren van interrupts, 73-76
CLI, 14,131
Interrupt status,redden en herstellen, 75,131
interrupt status, testen, 111,113
RTI, 74,526

SEI, 14,75,131
stack, 75-76
6522 VIA, 71-73
wanneer nodig, 75
Transfer instructies, effect op vlaggen, 12-13,30
Triviale problemen, 161
TSX, 19,30,52-53,104
Tussenvoeging in een string, 383-390
Twee complement overloop, 32-33,146-147
Twee complement, 92-93
Twee-byte plaatsen, 39-44,131
Twee-dimensionele arrays, 47-48,211-215
TXS, 19,102
vlaggen, effect op (geen), 12-13,30

U

UART zie 6551 ACIA;6850 ACIA,
Uitbreidings instructies, 93
Uitvoer regel routine, 438-440
Uitvoeringstijd, reductie van, 76-77
Uitvoeringstijden van instructies, 523,524

V

V (Overloop) vlag, 30-31,33-34,36,130-131,143-146
Variabel basisadres, 38-39
Veelvoorkomende programmeerfouten, 141-160
interrupt afhandelingsroutines, 159-160
Invoer/Uitvoer besturingsroutines, 157-159
Verdubbeling elementnummer, 40,41-42
Vergelijkings instructies, 90-91
bit-voor-bit (logische Exclusief-OR), 96
Overdracht vlag, 11,32-33,142-143
decimaal, 12,307
meervoudige precisie, 278-282
operatie, 25
16-bit, 250-254
string, 353-357
Nul vlag, 32-33
Verhoog achteraf, 136
Verhoog vooraf, 136-137
Verlaag achteraf, 137
stack wijzer, 14,22
Verlaag voortaf, 137-138
Vermenigvuldiging, 48-49,88-89
met een kleine integer, 48,88-89
met 10, 168-169,184-185
decimaal, 291-297
meervoudige precisie, 263-269,291-297
16-bit, 237-240
Verschuivingsinstructies, 26-30,90
diagrammen, 27-28
Invoer/Uitvoer, 58-60
multibit-, 26-30

multibyte, 328-332
Vertaal instructies, 131
Vertakkingsinstructies, 35-36, 108-125
beslissingsreeksen, 35,36
conditionele vertakkingen, 109-125
geindexeerde vertakkingen, 108-109
niet-conditionele vertakking, 108-109,155
niet-teken vertakkingen, 119-125
teken vertakkingen, 117-119
Vertragingsprogramma, 475-477
Verwijdering substring, 377-382
Verwissel instructies, 107
Verwisseling van elementen, 40,107,421
Verwisseling wijzers, 275-276
6522 Versatile Interface Adapter (VIA), 62-73
actieve overgang in, 62-66
adressering, 62-64
automatische modes, 66-70
auxiliary control register, 66,68-69,529
blok diagram, 529
controle lijnen, 66-70
controle registers, 63-65
data direction registers, 63-66,531
handmatige mode, 66-70
initialisatie voorbeelden, 65-71,469-474
interne adressering, 66,67,529
interrupt enable registers, 73-74
interrupt vlag registers, 70,75
interrupts, 73-75
invoer controle lijnen, 65-68
invoer/uitvoer controle lijnen, 65-69
Invoer/Uitvoer poorten, 530
onafhankelijke mode, 63-64,66,67
operatie modes (overzicht), 68
uitvoer registers, 530
verschillen tussen poort A en B, 66
Vlag registers, zie Status register
Vlaggen,
decimale mode, 12-13
effect van instructies, 523
laden, 104
organisatie van status register, 9,527
opslag, 105

W

Wacht instructie, 129-130
Wachtrij, 51-52,497-498
Wijzer, 11-12,14,24,49-50
laden, 103
verwisselen, 274-275,303
Wisinstructies, 106-107
Wissen status peripherie, 64,66-67,159-160,479-480,497-498
Wissen van bits, 26-27,106-107,333-337
Wissen van een array, 41-42,198
Wissen van vlaggen, 94-95

Woord grens, 148-149
.WORD pseudo operand, 91
Wraparound op pagina 0, 61,138

X

X register. zie Index registers

Y

Y register. zie Index registers

Z

Z vlag zie 0 vlag
Z-80 microprocessor, verschillen met 6502, 11,14,142
16-bit operaties, 12,50
    aftrekking, 92,93
    deling, 241-249
    indexering, 42-44
    optelling, 81-82,231-233
    pop, 129
    push, 129
    registers,gebrek aan, 12,49-50
    teller, 13
    test op 0, 52, 101,246
    vergelijking, 90-91,250-253
    verhoog met 1, 13,38,86-87,144
    verlaag met 1, 11, 13
    vermenigvuldiging, 237-240
    verschuivingen, 98-100
Zet (vlag) instructies, 107
Zetten van bits op 1, 26-27,28,95-96,309-311
Zetten van vlaggen, 96
Zoeken, 46,407-413