De uP Kenner 61

DOS65, 40 track kopieertip

Programmeren in assembler deel 2, A. Megens

CRTC.DOC bug, E van Kan

Basicode 3 voor DOS65, F. Bens

De uP Kenner 62

DOS65 Basic v2.20 Screeneditor V5.5

SET Video front, F. Vandenkerkhove

Programmeren in assembler deel 3, A. Megens

De uP Kenner 63

Programmeren in assembler deel 4, A. Megens

Linearisering, vevolg Spanningen meten F. Vandenkerkhove

Windows een verzameling subroutines, E. Elderenbosch

De uP Kenner 64

Windows onder DOS65 met een Z80 kaart, E. Elderenbosch

Virtual EPROM disk voor DOS65 Luc de Cock

De uP Kenner 65

Programmeren in assembler deel 5, A. Megens

Een 3.5 inch diskdrive E. Elderenbosch

Een RAM disk voor DOS65 E. Elderenbosch

DOS65 memory map E. Elderenbosch

De uP Kenner 66

Winchester drives en DOS65 E. Elderenbosch

DOS65 geluidsprint F. Bens

De uP Kenner 67

Diskcopy voor DOS65 F. Bens

8-bit microcontrollers van Intel programmeren via DOS65 en AS, E. Elderenbosch

De uP Kenner 68

Een printerbuffer en een toetsenbordbuffer voor DOS65, E. Elderenbosch

De uP Kenner 69

SUFAST t.b.v de realtime klok MC146818, H. Speksnijder

De uP Kenner 70

Een IBM keyboard aan DOS65, F. Bens

DOS65, 40 track: een kopieertip.

Door: Nico de Vries

Laatst moest ik voor een DOS65 gebruiker die alleen 40 track drives in zijn systeem heeft, een floppy disk aanmaken. Mijn systeem heeft echter alleen 80 track drives.

Nu kun je een nieuwe schijf pakken, deze met dubbel-step formatteren en vervolgens de files kopieren en deze schijf dan naar de gegadigde sturen. In theorie werkt dat wel: maar Murphy zegt van niet. De 40 track drive die zo'n schijf gaat lezen, ziet maar een half spoor. Als de andere helft leeg is (dat zijn de oneven tracks op een 80 track drive), dan lukt het wel. Maar o wee als er andere data staat....

Toen kreeg ik een idee: Mijn AT-kloon heeft een (80-track) 1.2 Mbyte drive en een (40 track) 360 kbyte drive. Nu kan een PC geen DOS65 lezen, maar het programma COPYIIPC kopieert een willekeurige 360k disk zonder meer, zeker als het format ruikt naar een format dat door een controller geschreven is, en dat is zo bij DOS65 schijfjes. Dus: blanko disk in drive B: (360k) en de dubbelstep DOS65 disk in drive A: en getypt: COPYIIPC A: B:.

En ja hoor: het werkt vlekkeloos!

Programeren in Assembler (deel 2)

Het tweede stapje

In deel 1 heb ik een optelroutine behandeld die een getal uit geheugenplaats GETAL2 (op $0201) optelde bij het getal uit geheugenplaats GETAL1 (op $0200) en dan het resultaat wegschreef in de geheugenplaats met de orginele naam RESULT (op $0202). Maar die geheugenplaatsen zijn allemaal slechts 8 bits breed dus dat betekent dat GETAL1 en GETAL2 tussen 0 en 255 kunnen liggen en om zulke kleine getallen op te tellen hebben we eigenlijk geen computer nodig. Ook zullen de bollebozen al wel gemerkt hebben dat bij bepaalde getallen het resultaat in RESULT niet meer klopt. Dat komt omdat ook RESULT maar 8 bits breed is en dan kan het volgende gebeuren:

Stel GETAL1 is 160 of $A0 en GETAL 2 is 170 of $AA. Vullen we deze getallen in op de bekende geheugenlocaties en starten we ons sublieme programma uit deel 1 dan zal bij het bekijken van RESULT met de MONITOR het volgende verbijsterende getal zich uit de diepste spelonken van het geheugen van je DOS65 systeem via de beeldbuis van je monitor een weg banen naar je netvlies om aldaar de volgende afdruk te maken (raheuh):

```
MON> @ 0202
0202 4A                 ; $4A = 74 (maar, maar...DAT IS NIET GOED!!)
```

Wordt het tijd om IBM te bellen? Moet de soldeerbout worden opgewarmd om een gigantische hardware fout te herstellen? Is uw processor doorgedraaid? Neen, driewerf neen, Ge kunt weer opgelucht adem halen. Alles laat zich eenvoudig verklaren door het feit dat de geheugenplaats RESULT slechts 8 bits breed is. De optelling GETAL1 + GETAL2 levert het getal 160 + 170 = 330 op, schrijven we dit hexadecimaal uit dan komen we op

```
330 =   ( 1 x 16^2 =  1 x 256) --> 330 - 256 = 74
      + ( 4 x 16^1 =  4 x  64) -->  74 -  64 = 10
      + (10 x 16^0 = 10 x   1) -->  10 -  10 =  0
        ------
          $14A
```

Zoals je ziet gaat dat omrekenen door herhaald aftrekken van de grootst mogelijke macht van 16 totdat het resultaat 0 is. $14A is 9 bits breed en past dus niet in ons 8 bits geheugen, we zien dat in ons programma de 1 is weggevallen. Maar hoe kan onze slaaf nu weten dat deze situatie zich heeft voorgedaan m.a.w. hoe kunnen we ons programma zo maken dat dit overlopen van ons 8 bits teiltje RESULT (Of zoals de Fransen zeggen ´overflow´) wordt gedetecteerd?
Daarvoor dient nu het ´Carry´ bit in het status register P, dit bit geeft bij een optelling aan of er zo´n overflow is geweest en bij aftrekken of er ´geleend´ moest worden (´borrow´ zoals de Fransen zeggen). Door dus na het optellen te testen of het ´C´ bitje ´1´ is kunnen we die ´overflow´ situatie detecteren en eventueel aangeven dat het resultaat niet betrouwbaar is bijv. door een foutmelding op het scherm. De gemakkelijkste manier om op de Carry te testen zijn de ´BCC´ (Branch Carry Clear) en ´BCS´ (Branch Carry Set) instructies. Het woord ´Branch´ betekent tak, vertakking, verder gaan in een andere richting. En dat is precies wat de processor doet bij deze instructies. Eerst wordt het carry-bit getest en afhankelijk van het resultaat wordt er wel of niet een sprong uitgevoerd m.a.w. het programma vertakt zich wel of niet. Het geheel laat zich goed beschrijven met de BASIC regels:

```
IF (CARRY = 0) THEN GOTO ..... (voor BCC)
IF (CARRY = 1) THEN GOTO ..... (voor BCS)
```

Stel nu dat de test waar is dan moet de processor dus een sprong uitvoeren en nu wil ie dat best maar hoe ver en waar naar toe? Nu heeft de chipboer daar een aardige oplossing voor bedacht, die informatie vindt den 6502 op de geheugenlocatie direct achter de instructie voor een branch. Daarbij wordt de grootte van de sprong aangegeven door de onderste 7 bits van het getal en de richting door het 8e bit. Die spronglengte (door de Fransen steevast 'offset' genoemd) is dus met een duur (Frans) woord een 8 bits signed (2's complement) number. Wat is nu de maximale sprong die onze 6502 kan uitvoeren, trouwens dit geldt voor iedere processor waarbij de offset is gegeven door een byte. Voor een positieve sprong geldt dat het 8e bit '0' is, het grootste getal wat we dus kunnen schrijven is:

```
76543210 <-- bitnummer
01111111 = $7F = 127
```

(Je ziet dat ik spreek over het 8e bit toch is het gewoonte om het 1e bit te benoemen met nummer 0 (zeg 'bit nul') en het 8e met nummer 7 (bit zeuven).) De grootste negatieve sprong is dan dus -128 met onze 8 bits offset. Omdat de 6502 werkt met de 2's complement methode (zie ook KENNER 58, deel 1 over getallen binnen de microprocessor van Gert van Opbroek) wordt dat dus:

```
76543210
10000000 = $80 = -128
```

Nu is dat uittellen van zo'n branch of relatieve jump (relatief omdat het berekende adres afhangt van de PC) een vak apart en eigenlijk heel vervelend. Maar gelukkig hoeven wij dat niet meer te doen want al heel snel hebben slimme jongens een programma geschreven dat die vervelende klusjes zoals opcodes opzoeken in de tabel, relatieve jumps uitrekenen e.d. voor ze opknapte. Ook bij de DOS65 is zo'n handig programma het heet AS en dat is zeer kort voor ASSEMBLER (het Franse woord voor samenvoegen, monteren). Het grootte voordeel van werken in assembler is dat je een geheugenplaats een naam kunt geven en de opcodes gewoon in tekst kunt uitschrijven, dus niet meer 8D 02 02 maar gewoon STA RESULT. Een tweede voordeel is dat je je programma's kunt documenteren omdat je gewoon in de EDITOR werkt en commentaar achter de instructies kunt zetten. Kortom je programma's zijn leesbaarder.

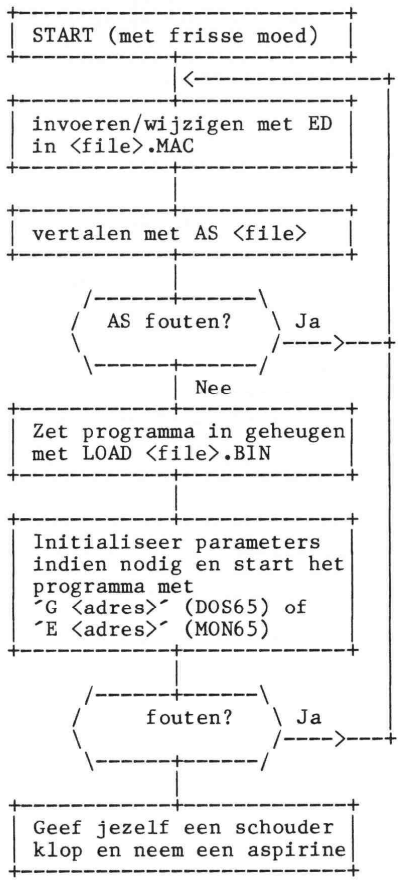
Programma ontwikkeling

We gaan ons programma uit deel 1 uitbreiden met een test op de overflow conditie en zetten daarbij het resultaat op het scherm met wel of geen foutmelding. Daarbij gebruiken we 2 routines uit DOS65 zelf t.w.

```
HEXOUT1 print Accu in hexadecimaal
PRINT1  print string after call until <null>
```

Beide routines zijn zeer eenvoudig te gebruiken en hebben als grote voordeel dat de waarde in de Accu niet wordt aangetast.
In de rest van dit verhaal neem ik aan dat je weet hoe je met de EDITOR moet werken en de syntax van AS (ongeveer) kent. Kijk het anders nog even na in de handleiding van beide utilities.

Het flowdiagram voor programma ontwikkeling met EDITOR en ASSEMBLER ziet er als volgt uit:



Het zal iedereen duidelijk zijn dat vooral die laatste conditie moeilijk te beoordelen is en de reden kan zijn voor lange nachtelijke sessies achter het toestenbord terwijl de radeloze programmeur er achter probeert te komen waarom zijn geesteskind maar niet wil doen wat hij dacht dat het zou moeten doen. Ik geloof trouwens niet dat ik overdrijf als ik beweer dat in de meeste programma's die laatste conditie niet goed is beoordeeld en dat er dus nog fouten in het 'voltooide' product zitten. De Fransen hebben een aardig woord voor dergelijke programmafouten: BUG (insect, luis).
[Er is een aardige geschiedenis aan de oorsprong van dat woord verbonden. In de allereerste computers werden honderden electronen buizen gebruikt en het geheel gaf dan ook aardig wat licht en warmte af. In de zomer zetten de operators dan ook meestal de ramen open omdat de airconditioners van die tijd onvoldoende capaciteit hadden om het vertrek op temperatuur te houden. Nu waren al die lichtjes voldoende om allerlei insecten naar dat wonderlijke schouwspel te lokken met als gevolg dat er nogal eens een kortsluiting optrad in de computer wanneer een insect zich te dicht bij de electronenbuizen had gewaagd. Je ziet dus dat een BUG in eerste instantie een hardware probleem van computers was.]

Het ADD_8 programma is echter BUG-free en getuigt van een genialiteit, sublimiteit en is van een zodanige pure schoonheid dat ik het schrijven van dit artikel even heb moeten onderbreken om mijn van emoties overgelopen gemoed weer enigzins tot rust te laten komen.....

Het programma in AS wordt (invoeren met ED in file ADD_8.MAC):

```
;
; File    : ADD_8.MAC
; Purpose : Add two 8 bit numbers and show result on screen
;           show warning in case of overflow.

HEXOUT1 EQU     $C038           ; vertel AS waar HEXOUT1 begint
PRINT1  EQU     $C03B           ; en hier begint PRINT1

        ORG     $1000           ; vertel AS waar het programma begint

ADD_8   LDA     GETAL1          ; haal GETAL1 in A
        CLC                     ; zet de Carry op nul
        ADC     GETAL2          ; tel GETAL2 bij A
        STA     RESULT          ; zet het resultaat weg
        BCC     OK              ; het resultaat is goed, geen overflow
        JSR     PRINT1          ; foutmelding
        FCC     13,'WARNING, OVERFLOW DETECTED',0
OK      JSR     PRINT1          ; begin op een nieuwe regel
        FCC     13,0
        JSR     HEXOUT1         ; toon resultaat
        RTS                     ; terug naar MON of DOS

        ORG     $0200           ; vertel AS waar de data staat

GETAL1  FCB     0               ; reserveer een byte voor GETAL1
GETAL2  FCB     0               ; en een voor GETAL2
RESULT  FCB     0               ; en een voor RESULT

        END     ADD_8           ; thats all folks!
```

Het ADD_8 programma kun je als volgt vertalen en uitproberen:

```
$ AS ADD_8
Pass 1...
Pass 2...
Last assembled address: 0202
Errors detected: 0
```

Als je andere resultaten krijgt heb je ergens een type fout gemaakt in de ADD_8.MAC file. Is het wel goed dan zie je dat er een file ADD_8.BIN is gemaakt door het AS programma. Deze file kun je in het geheugen laden met:

```
$ LOAD ADD_8.BIN
```

Als je niet precies weet waar het programma start of waar het eigenlijk terecht komt dan kun je het MAP commando gebruiken:

```
$ MAP ADD_8.BIN
1000-1034       ; eerste gedeelte (in dit geval programma)
0200-0202       ; tweede gedeelte (in dit geval data)
1000            ; start address
```

```
$ MON           ; ga in de MONITOR
MON65 2.06
HaViSoft
Rockwell 65C02
MON> D 1000     ; start de DISASSEMBLER
1000   AD 00 02          LDA  $0200
1003   18                CLC
1004   6D 01 02          ADC  $0201
1007   8D 02 02          STA  $0202
100A   90 20             BCC  $102C
100C   20 3B C0          JSR  $C03B
100F   0D 57 41          ORA  $4157
1012   52 4E             EOR  [$4E]
1014   49 4E             EOR  #$4E
1016   47 2C             RMB  #4,$2C
1018   20 4F 56          JSR  $564F
101B   45 52             EOR  $52
101D   46 4C             LSR  $4C
101F   4F 57 20          BBR  #4,$57,$1042
1022   44                ???
1023   45 54             EOR  $54
1025   45 43             EOR  $43
1027   54                ???
1028   45 44             EOR  $44
102A   0D 00 20          ORA  $2000
```

Je herkent in de eerste vier instructies het programma uit deel 1 maar daarna begint het af te wijken. We zien trouwens dat de DISASSEMBLER zich behoorlijk verslikt op het stukje tekst dat we hebben ingevoerd. Een HEXDUMP wil in zo'n geval meestal wel opheldering geven:

```
MON> H 1000,1020

       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
1000 : AD 00 02 18 6D 01 02 8D 02 02 90 20 20 3B C0 0D  ................
1010 : 57 41 52 4E 49 4E 47 2C 20 4F 56 45 52 46 4C 4F  WARNING, OVERFLO
1020 : 57 20 44 45 54 45 43 54 45 44 0D 00 20 3B C0 0D  W DETECTED..... 
```

Nu is snel duidelijk waarom het disassembleren zulke wilde resultaten gaf. Maar goed, we gaan testen. Allereerst weer GETAL1 en GETAL2 vullen:

```
MON> @ 0200
0200 00 34
0201 00 12
0202 00 00
```

En starten maar:

```
MON> E 1000
46                      ; Is dat even luxe? We hoeven niet meer met de hand
MON>                    ; te kijken of het wel klopt. Maar nu met die andere
                        ; getallen en dan toch maar gelijk even kijken of het
                        ; RESULT echt wel goed is.
MON> @ 0200
0200 34 A0
0201 12 AA
0202 46 00

MON> E 1000
WARNING, OVERFLOW DETECTED
4A
MON>                    ; Het (b)lijkt te werken!
```

Allemaal leuk en aardig zul je denken maar wat nu te doen als ik nu wil werken met getallen boven de 255? In dat geval gaan we het getal opslaan in meerdere geheugen locaties. Met twee locaties = 16 bits kun je al rekenen tot 65535, met drie locaties = 24 bits al tot 16777216!! Wil je ook negatieve getallen gebruiken dan is dat voor 16 bits van -32768 tot 32767 en met 24 bits van -8388608 tot 8388607. In de meeste toepassingen in AS is 16 bits meestal ruim voldoende. Daarbij spreken we af dat het HIGH byte van het 16 bits getal altijd op de geheugenplaats na het LOW byte staat. We hanteren dan dezelfde notatie als de 6502 zelf doet bij 16 bits getallen (kijk maar in ons voorbeeld, eerst het LOW en dan pas het HIGH byte). Je HOEFT dat natuurlijk niet te doen maar als je het jezelf vanaf het begin aanleert dan went het vanzelf. Trouwens AS neemt ons dat omkeren al uit handen door gebruik te maken van de FDB instructie.
We gaan nu ons ADD_8 programma uitbreiden voor 16 bits getallen. Gebruik COPY ADD_8.MAC ADD_16.MAC en EDIT de onstane ADD_16.MAC file als volgt:

```
;
; File    : ADD_16.MAC
; Purpose : Add two 16 bit numbers and show result on screen
;           show warning in case of overflow.

HEXOUT1 EQU     $C038           ; vertel AS waar HEXOUT1 begint
PRINT1  EQU     $C03B           ; en hier begint PRINT1

        ORG     $1000           ; vertel AS waar het programma begint

ADD_16  LDA     GETAL1          ; haal LOW part van GETAL1 in A
        CLC                     ; zet de Carry op nul
        ADC     GETAL2          ; tel LOW part van GETAL2 bij A
        STA     RESULT          ; zet het resultaat weg
        LDA     GETAL1+1        ; haal HIGH part van GETAL1
        ADC     GETAL2+1        ; tel HIGH part van GETAL2 plus CARRY bij A
        STA     RESULT+1        ; zet het resultaat weg
        BCC     OK              ; het resultaat is goed, geen overflow
        JSR     PRINT1          ; foutmelding
        FCC     13,'WARNING, OVERFLOW DETECTED',0
OK      JSR     PRINT1          ; begin op een nieuwe regel
        FCC     13,0
        LDA     RESULT+1
        JSR     HEXOUT1         ; toon resultaat HIGH
        LDA     RESULT
        JSR     HEXOUT1         ; toon resultaat LOW
        RTS                     ; terug naar MON of DOS

        ORG     $0200           ; vertel AS waar de data staat

GETAL1  FDB     0               ; reserveer een WOORD=16 bits voor GETAL1
GETAL2  FDB     0               ; en een voor GETAL2
RESULT  FDB     0               ; en een voor RESULT

        END     ADD_16  ; thats all folks!
```

Om het programma te testen blijven we nu voor de verandering eens in DOS en gebruiken daarbij de volgende commando´s:

```
$ AS ADD_16
Pass 1...
Pass 2...
Last assembled address: 0205
Errors detected: 0
$ LOAD ADD_16.BIN
$ MEMFILL 0200,0200,A0
$ LC 0201,0201,01
$ LC 0202,0202,66
$ LC 0203,0203,09
$ G 1000

0B06$
```

We hebben GETAL1 gevuld met $01A0 en GETAL2 met $0966, het resultaat is $0B06 en dat klopt. De carry die is onstaan bij het optellen van $A0 en $66 is inderdaad meegenomen bij het optellen van $01 en $09. We zien dat er nog een klein schoonheidsfoutje in het programma zit, de DOS65 prompt staat direct achter het resultaat. Bij gebruik van MON65 gaat dit wel goed omdat daar de prompt begint met een CRLF. Voor de perfectionisten onder U vermeldt ik daarom de CRLF routine van DOS65 die begint op $C02F. Deze routine kun je voor het verlaten van het ADD_8 en ADD_16 programma uitvoeren om de cursor altijd aan het begin van de regel te krijgen.
Probeer zelf eens een 8 of 16 bits aftrekroutine te schrijven. Gebruik hiervoor de instructie ´SBC´ en bedenk hierbij dat de carry hierbij de functie van ´lenen´ of ´borrow´ heeft en dus vooraf ´1´ moet zijn (gebruik SEC) om correcte resultaten te krijgen. Is er dan daadwerkelijk zo´n ´borrow´ geweest dan is de carry na de SBC instructie ´0´. De test voor ´underflow´ wordt dan gedaan met ´BCS´. Veel succes!

Antoine Megens

Nogmaals: CRTC.DOC.

In de 6502-Kenner nummer 56 beschreef Jaques Banser, op pagina 28, de wijze waarop een monitor met een 9-pens "IBM-aansluiting" op de VDU-kaart van Elektuur kon worden aangesloten.
Hierin ben ik het volgende foutje tegengekomen:
In regel 6 staat: Als eerste het VIDEO-signaal afnemen van pen 12 van IC3 (N22) enz.
Dit moet zijn: Als eerste het VIDEO-signaal afnemen van pen 6 van IC3 (N22) en deze aansluiten op pen 7 van de 9-pens stekker.

Tot slot de volgende opmerking:
Om instabiliteit te vermijden is het verstandig om pen 8 van IC4 buiten de IC-voet te buigen.

Evert van Kan.

Basicode3 voor DOS-65.

Voor DOS-65 bestaat sinds kort, naast een basicode2 programma ook een basicode3 programma. De maker, Frank Bens, beschrijft in het onderstaande verhaal dit vertaalprogramma.

Het programma is beschikbaar op het bulletin board en verkrijgbaar bij de DOS-65 coördinator Jan Derksen.

(Redactie).

Deze file geeft in beknopte vorm weer wat de mogelijkheden zijn van het BASICODE 3 vertaalprogramma versie 1.0 voor DOS65 computers.

Dit BASICODE-3 vertaalprogramma is geschikt voor alle DOS65 computers en is ongeveer 3K byte groot.

Het is geheel in BASIC geschreven, alleen de tekenroutine staat in machinetaal geschreven om de snelheid te verhogen.

Basic V2.00 en 1/065 V2.01 moeten in het systeem aanwezig zijn om de goede werking van het vertaalprogramma te waarborgen.

Om goed met het vertaalprogramma te kunnen werken dienen de volgende file's in het systeem aanwezig of op diskette beschikbaar te zijn:

- BASICODE3.BAS (vertaalprogramma)
- BCODE3V1.DAT (bevat o.m. de plot routine)
- de VDU-EPROM geprogrammeerd met CHARGV1D2 (karaktergenerator)

Tijdens de executie van BASICODE3.BAS wordt de file BCODE3V1.DAT in het geheugen geladen en wordt op het scherm zowel het versienummer van het vertaalprogramma als dat van de .DAT file vertoond, zodat U kunt zien met welke versie gewerkt wordt. Tevens komen er nog wat aanwijzingen, o.a. het gebruik van het MERGE commando, op het scherm.

Het is niet toegestaan om een BASICODE-3 programma incl. het vertaalprogramma op diskette te zetten. Dit is tegen het BASICODE protocol.

In het originele vertaalprogramma staat op regelnummer 1000 en hoger o.a. een commando dat de file BCODE3V1.DAT in het geheugen laadt. Na het runnen van het vertaalprogramma is dit commando verdwenen.

Daarom, wordt er een programma op disk gezet, moet dit vanaf regelnummer 1000 en in ASCII formaat gebeuren, omdat het MERGE commando uitsluitend ASCII file's accepteert.

Houdt er ook rekening mee dat als er een file van cassette wordt geladen welke bij een bestandenprogramma hoort, deze in ASCII formaat met de extensie .VAR op disk wordt gezet.

Bijzonderheden en beperkingen.

Het tekstscherm kent TELETEXT-tekens. In het grafische scherm worden normale ASCII symbolen vertoond.

In de grafische mode staat het display op 9 scanlines i.p.v. 10 ingesteld, waardoor sommige kleine letters niet helemaal netjes vertoont worden.

Om een goed grafisch beeld te krijgen moet het scherm eerst gewist worden, waarna er dan willekeurig lijnen getrokken kunnen worden.

Wordt eerst een karakter geplaatst en daarna een lijnstuk door dit karakter getrokken dan zal dit verminkt worden. In omgekeerde volgorde: dus eerst een lijnstuk en daarna het karakter, dan zal deze niet verminken.

Daar het programma (nog) niet een eigen cassette routine bezit, is het (nog) niet mogelijk volgens het BASICODE-3 protocol een bestand naar cassette te zenden.

Het lezen van bestand files kan gebeuren m.b.v. het programma TAPE. De maximale regellengte is 78 karakters (zowel bij schrijven als bij lezen van bestanden).

Variable OT op regelnummer 21 moet voor een kloksnelheid anders dan 1MHz aangepast worden, daar anders de routine op 450 te snel verlaten wordt.

Cursorbesturing is mogelijk met ^H,^I,^J en ^K resp. links, rechts, omlaag en omhoog. De toets DELete (hex 7F) heeft de wisfunctie.

Afbreken van een programma is mogelijk met ^C, mits deze functie niet door het programma uitgeschakeld is.

DOS-65 BASIC V2.20 SCREENEDITOR V5.5

Auteur: B. de Bruine

Files: Scred55.mac
Basic V2.20
Radixbc3.mac

Literatuur: De 6502 Kenner 41 blz. 15

1. Introductie

Scred is een semi-screen-editor voor DOS-65 Basic V2.20, die het mogelijk maakt om kleine wijzigingen gemakkelijk aan te brengen. Dit programma is niet bedoeld om uitgebreide basiclistings in te voeren. Hiervoor kan men beter ED gebruiken.

2. Principe

Het principe van scred is om de inputroutine van Basic ($C026) te vervangen door een routine die data van het scherm leest. Met de cursortoetsen kan men de plaats van de te bewerken data manipuleren. Men dient er rekening mee te houden dat er tot de cursorpositie gelezen wordt. Wil men dus een hele regel invoeren, dan moet men de cursor vanaf kolom nul tot aan het eind van de regel plaatsen alvorens RETURN te geven. Geeft men b.v. halverwege al een RETURN, dan wordt ook maar een halve regel opgeslagen in het basicgeheugen.

3. Commando-overzicht

```
^H        LEFT           ; Verplaats cursor 1 positie naar links
^I        RIGHT          ; Verplaats cursor 1 positie naar rechts
^K        UP             ; Verplaats cursor 1 positie omhoog
^J        DOWN           ; Verplaats cursor 1 positie omlaag
^L        CHAR.INS       ; Insert een spatie op de plaats waar de cursor staat
^G        CHAR.DEL       ; Verwijder een karakter op de cursorplaats
^Y        END LINE       ; Zet de cursor aan het einde van de regel
^T        BEG LINE       ; Zet de cursor aan het begin van de regel
^\        HOME           ; Zet de cursor in de linker bovenhoek van het scherm
^N        ERA LINE       ; Verwijder het regelgedeelte na de cursorpositie
^O        CLR SCR        ; Veeg het scherm schoon
^P        DUMP SCR       ; Druk de data van het beeldscherm op de printer af
^Z        RESTORE        ; Forceer invoer vanaf het toetsenbord
^E        EXIT           ; Ga terug naar DOS-65
^A        STORE USR$     ; Bewaar string van kol. 1 tot de cursor in een buffer
^U,^V     USR$: RUN      ; Print de string die in het buffer was opgeslagen
ESC :     ERA SCR        ; Veeg het scherm schoon vanaf de huidige cursorplaats
ESC H     HELP           ; Print dit commando-overzicht
ESC O     OLD            ; Haalt na NEW het oude programma weer terug
ESC P     INIT PRINTER   ; Initialiseert de printer met videotexgraphics (BC3)
```

4. Toepassingen en voorbeelden

a. Corrigeren

In een listing staat:

```
10 FOi=1to 10: print i: next
```

Met de cursortoetsen gaan we naar de i in FOi.
Hier geven een ^L en een R. Vervolgens gaan we met ^Y naar het einde van de regel en geven een RETURN. Na een LIST blijkt er het volgende te staan:

```
10 FORi=1to 10: print i: next
```

Noten: Om een regel in te lezen moet men beginnen in Kol. 1 (bv met ^T)
Op dezelfde manier kan men deleten en overschrijven

b. Het opslaan van een string (macro)

Het is mogelijk om een string van maximaal 80 karakters als macro te bewaren. Men tikt de string in of gaat met de cursor naar het laatste karakter van de string toe (gerekend vanaf kolom 1). Men geeft nu een ^A en de string is opgeslagen.

Voorbeeld:

```
LIST 100-200&                (& is de cursor)
^A
LIST 100-&00                 (& is de cursor)
^A
```

In het eerste geval is de string: LIST 100-200, in het tweede geval LIST 100-

^U print de string weer op het scherm.
^V doet hetzelfde, maar geeft er automatisch een RETURN achter, dit kan handig zijn als men bv CONT als string heeft opgeslagen. Na ieder STOP-commando geeft men een ^V om te continueren.

c. Screendump

Met ^P wordt het scherm afgebeeld op papier door een printer. Wil men graphisch printen (basicode 3) dan moet men vooraf eerst nog een ESC P geven om de graphics in de printer te zetten. Dit is NIET noodzakelijk als men alleen asciikarakters wil printen. Indien de printer niet klaar voor gebruik is zal na ca. 10 seconden de melding "Printer not ready" op het scherm verschijnen. Het is mogelijk om een gedeelte van het scherm af te drukken. Dit is in te stellen met de variabele PAG. (Zie ook &6).

d. Noodrem

Indien men met een call&xxxx een routine zelf aanroept, dan kan met name de ^Y-routine de 80-kolom check missen. Het resultaat is dat de computer 'hangt', en alleen een hinderlijk beepje laat horen ten teken dat er sprake is van een regeloverschrijding. Met ^Z forceert men invoer van het toetsenbord, zodat men weer gewoon verder kan werken. Evenals bij de DOS-65 stop redirect, zal men dit commando bij normale toepassingen nooit nodig hebben.

5. Toetsentabel

Desgewenst kan men de toetscodes wijzigen. Karakters als ^C, ^Z, ^S en ^Q zijn verboden.

```
     ;KEYTABLE
     ;========
     ;
000B KEYUP      equ  $0B          ; ^K CURSOR UP
000A KEYDOWN    equ  $0A          ; ^J CURSOR DOWN
0009 KEYRIGHT   equ  $09          ; ^I CURSOR RIGHT
0008 KEYLEFT    equ  $08          ; ^H CURSOR LEFT
000C KEYINSERT  equ  $0C          ; ^L CHARACTER INSERT
0007 KEYDELETE  equ  $07          ; ^G CHARACTER DELETE
0019 KEYEOLN    equ  $19          ; ^Y CURSOR TO EOLN
0014 KEYBOLN    equ  $14          ; ^T CURSOR TO BOLN
001C KEYHOME    equ  $1C          ; ^\ CURSOR HOME
000E KEYLINERA  equ  $0E          ; ^N LINE ERASE FROM CURSOR UNTIL EOLN
000F KEYCLEAR   equ  $0F          ; ^O CLEAR SCREEN AND HOME
001A KEYRES     equ  $1A          ; ^Z RESTORE ALWAYS KEYB VECTOR
0010 KEYPRI     equ  $10          ; ^P SCREENDUMP
```

```
0001 KEYMEMO    equ  $01            ; ^A MEMORIZE
0015 KEYMAC1    equ  $15            ; ^U USER DEFINED STRING
0016 KEYMAC2    equ  $16            ; ^V USER DEFINED STRING + CR
0005 KEYEX      equ  $05            ; ^E EXIT
     ; DECODATIE NA <ESCAPE>$1B
007C KEYSCRERA equ  $7C            ; ^] ERASE SCREEN FROM CURSOR UNTIL EOS
0048 KEYHELP    equ  'H'            ; HELPSCREEN
0050 KEY1PRI    equ  'P'            ; INITIALISE PRINTER
004F KEYOLD     equ  'O'            ; OLD (BASIC COMMAND)

                ; Keys can be redefined by the keytable
                ; Change also the help routine
```

6. Routines

Scred kent een aantal subroutines die ook handig kunnen zijn om tijdens een basicprogramma aan te roepen met call(&xxxx). Zo is READSCR te gebruiken in de basicode-3 subroutine 220 (lees van tekstscherm), en DUMPSCR kan men in een programma aanroepen om een screendump te maken zonder dat men hoeft te breken en een ^P moet geven die meestal voor datavervuiling op het scherm zorgt.
INIMOD geeft het startadres van de initialisatieroutine voor de printer aan. Hier is dat $A000, dwz dat u:g/radixbc3.bin moet beginnen op $A000. De naam van de initialisatiesoftware kan men wijzigen vanaf INILO. Let wel op dat de lengte niet langer mag zijn dan aangegeven.
Indien men een gewone basic zonder SCRED wil hebben, bv om input redirect te versnellen, dan kan men SCRED uitschakelen met call &A41B.

```
A400 4C  6BA4       SCRED     JMP  EDIT
                    ;--jumptable--
                    ;entries can be called from basic with CALL(&XXXX)
A403                ASAVE     res  1              ;Savelocations A,X and Y
A404                XSAVE     res  1
A405                YSAVE     res  1
A406 4C  17A6       READSCR   JMP  VDUCHA         ;Read from screen and place in ASAVE
A409 4C  31A9       INIPR     JMP  INICEN         ;Initialise printer
A40C 4C  58A8       DUMPSCR   JMP  PRSCR          ;Screendump on printer
A40F 4C  EBA8       PRASC     JMP  CENTRO         ;A to printer
A412 4C                       FCB  $4C            ;Forced jump
A413 00A0           INIMOD    FDB  $A000          ;Load address printer init
A415 4C  4FA8       STATOFF   JMP  STATI          ;Statusline off
A418 4C  B5A4       ILOAD     JMP  LOAD           ;Load A,X,Y
A41B 4C  40A9       SCOFF     JMP  NOED           ;Switch scred off
A41E 4C  50A9       SCON      JMP  LOED           ;Load editor
A421 4C  80A4       GRAFON    JMP  NIX            ;Select graphic screen (no funtion at
                                                  ;V5.5)
A424 4C  80A4       TXTON     JMP  NIX            ;Select txt-screen (no function at
                                                  ;V5.5)
A424 4C4F414429     INILO     FCC  'LOAD U:G/RADIXBC3.BIN',0
A427 553A472F52
A431 4144495842
A436 43332E4249
A43B 4E00
A43D 000000                   res  3,0            ;Expansion
A454 18             PAG       res  1,24           ;Screendump: screen of 24 lines.
                                                  ;range [1..25]
A455 00             LINE      res  1,0            ;Linecounter printscreen
A456 00             ABS       res  1,0            ;Autobreak Curright > 80
A457 00             XS        res  1,0            ;Savelocation X printscreen
A458 00             AHOLD2    res  1,0            ;Save A
A459 00             XHOLD2    res  1,0            ;Save X
A45A 00             YHOLD2    res  1,0            ;Save Y
A45B FF             FLGCR     res  1,$FF          ;Auto cr after ^V [0=on, else=off]
```

Voorbeelden:

Dump de eerste helft van het scherm op de printer als keuze=3:

```
10  PRINT" 1 = uitleg"
20  PRINT" 2 = stoppen"
30  PRINT" 3 = screendump"
40  GET I: IF I=3 THEN POKE &a454,12:call&a40c:poke&a454,24
```

Lees het karakter onder de cursor in B$:

```
100 call&a406:B$=chr$(peek&a403))
```

7. Geheugenindeling

```
Scred5.5        $A400-$A9FF          ;LET OP! mag $AA00 niet overschrijden!
Printerini      $A000-$A3ff
```

8. Grafisch printen

Maak een routine die de grafische karakterset in de printer zet. Dit moet als subroutine gebeuren met startadres op INIMOD. Een terugkeer van deze routine met C=0 is succesvol. Indien C=1 is er iets mis, bv de printer is not ready. (Zie radixbc3.mac)

Errata artikel De 6502 Kenner 41 SCRED 3.0

```
blz. 16 2e regel:   Dit zijn commando´s die betrekking hebben op het HELE SCHERM
        onder B2:   Voorlaatste commando moet zijn: (CNTRL+N) spatiebalk
                    Laatste commando (CNTRL+X), hier ontbreekt: was voorheen ´@´
blz. 17 voor Opmerking: Dient te staan (ESC) spatiebalk en (CNTRL+N) spatiebalk
blz. 25 regelnr 3910:   HOOFDletters worden kleine letters.
```

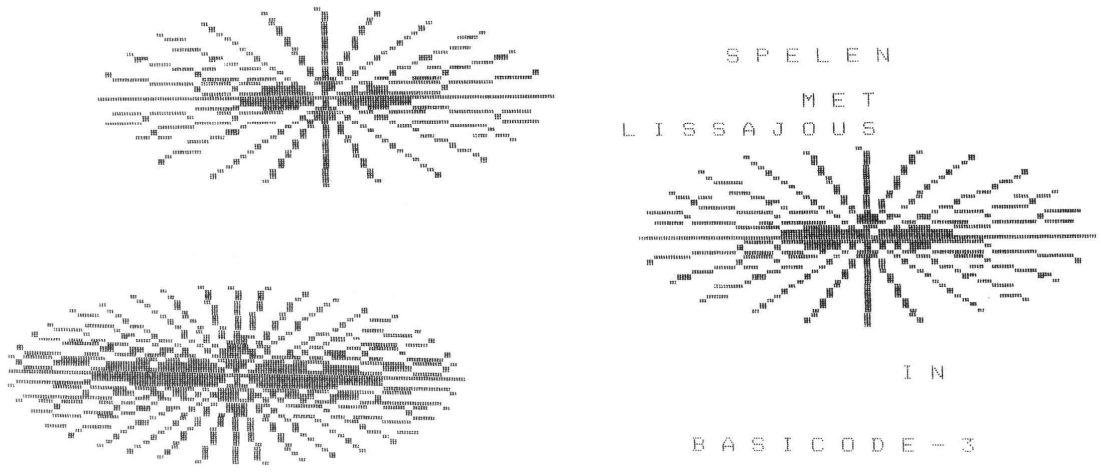
Zelf SCRED toevoegen aan Basic V2.20

```
AS SCRED55
APPEND -A SCRED55.BIN BASIC
```

SCRED en Basicode-3

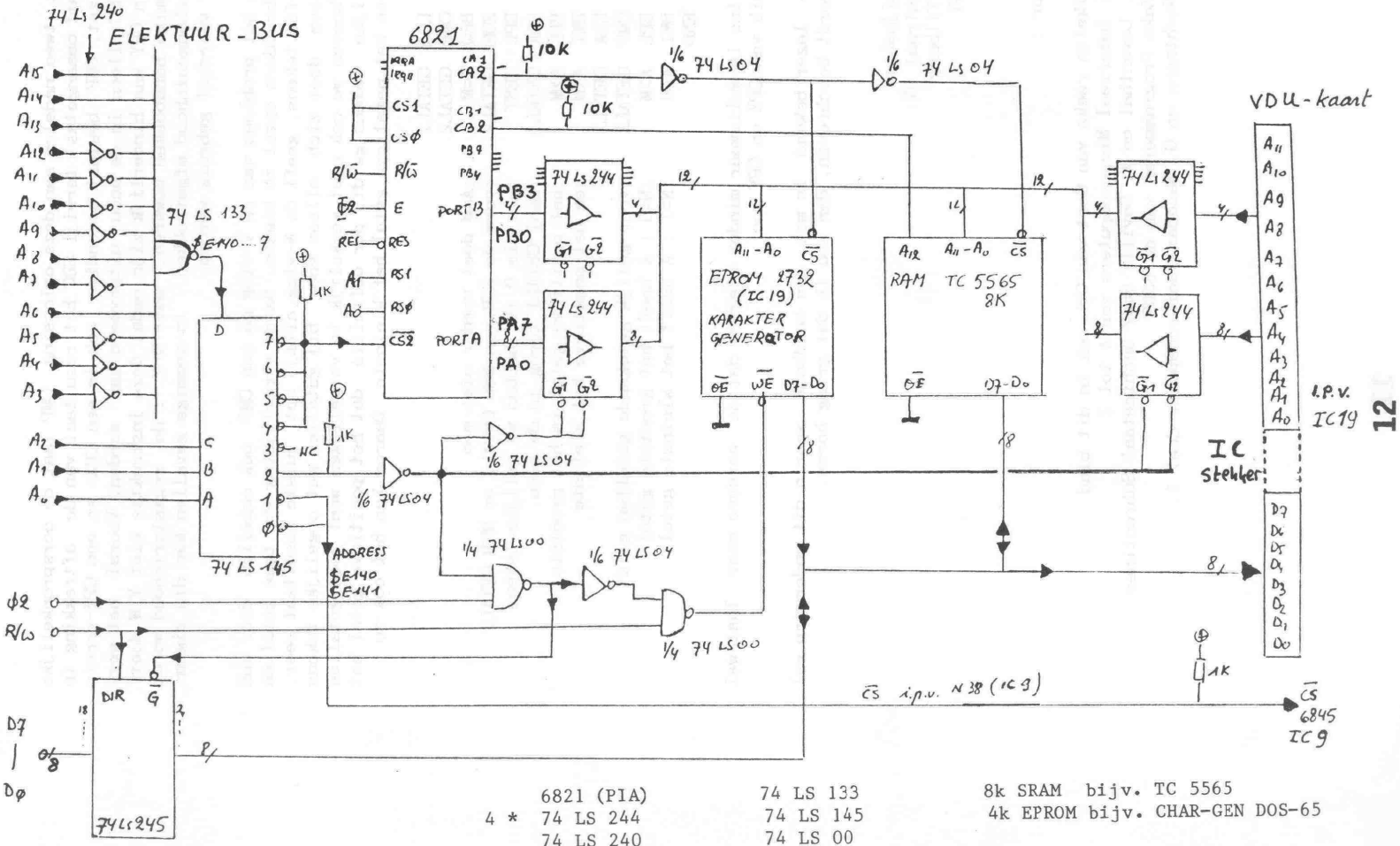
Van Frank Bens ontving ik een basicode-3 vertaalprogramma. Aansluitend hierop heb ik ESC P en ^P toegevoegd. ESC P is printerafhankelijk, en kan analoog aan de viditelroutines aangepast worden voor diverse printers.

Hieronder een demo van een basicode-3 plot.



VIDEO FRONT

Deze schakeling wordt verbonden met de VDU-Kaart met behulp van een flat-cable en een IC-voet steker naar de plaats van IC 19 en een draadje als nieuwe CS voor de 6845 via IC9.



	6821 (PIA)	74 LS 133	8k SRAM bijv. TC 5565
4 *	74 LS 244	74 LS 145	4k EPROM bijv. CHAR-GEN DOS-65
	74 LS 240	74 LS 00	
	74 LS 245	74 LS 04	IC 9 en IC 19 vervallen op de VDU-kaart.

SET Video FRONT

Door: Frank Vandekerkhove.

De CRTC controller 6845 wordt in vele computers gebruikt, alleen wordt hij op de VDU-kaart op een "eenvoudige" wijze gebruikt. Daar ik de VDU-kaart niet wens te vervangen, heb ik een kaart gemaakt die naast de VDU-kaart komt. De voornaamste delen zijn een PIA, een multiplexer en RAM.

Beide kaarten worden verbonden d.m.v. een draadje voor de nieuwe chip select naar de 6845 (IC 9 wordt verwijderd, dus geen viervoudige adressering meer!) en een flatcable met een IC voet connector. Deze flatcable gaat naar de plaats van IC 19, de karakter-generator, die ook verwijderd wordt en waardoor de data- en adreslijnen naar de nieuwe print gebracht worden.

In rust schakelt de multiplexer de adreslijnen van de VDU-kaart door, wordt er echter gelezen of geschreven naar de kaart, dan krijgen de RAM en de EPROM hun adres aangeboden door de poorten van de PIA (een pseudo adres).

Na een reset zijn CA2 en CB2 altijd als ingang geset; door de pull-up weerstand en de inverter wordt daarom altijd de EPROM geselecteerd. (Bij een power-on bevat de RAM nog geen geschikte data.)

Met het onderstaande programma kan de RAM geladen worden met een willekeurige karakterset die bewaard wordt op schijf, bv.: "SETVF cgDOS.rom". Ook is het mogelijk een karakterset, die op dat moment gebruikt wordt, op de schijf weg te schrijven. We kunnen ook eerst een set laden, elke bit (of enkele bits) wijzigen, en dan bewaren met zijn eigen willekeurige naam. Voorts zijn er nog enkele opties waarmee we de huidige set kunnen laten zien (-d) of kunnen omschakelen tussen de karaktersets (-e, -r en -t), waarbij twee verschillende karaktersets in RAM bewaard kunnen worden. (Bij het laden wordt steeds omgeschakeld naar het andere gedeelte.)

De -n optie zal, na het laden, het tweede gedeelte van een karakterset (van $80 t/m $FF) veranderen tot het tegengestelde van het eerste gedeelte waardoor, zoals in de DOS-65 karakterset, de inverse karakters ontstaan.

Natuurlijk is het mogelijk om dit programma op te nemen in de "LOGIN.COM", waardoor bij een start steeds de juiste karakterset aanwezig is voor bv. EC-65, VIDITEL, ... of gewoon DOS-65.

```
        ; File:         SETVFRONT.mac
        ; Prog:         SETVFRONT
        ; Function:     set video character generator
        ; Usage:              SETVF options file
        ; Date:         november/december 1988
        ; By:           Frank Vandekerkhove
        ;                      Sint-Michielsstraat 4
        ;                      B-2789 Verrebroek (ANTW)
        ;
        ; Zero page locations
        0080            org     $80
0080            optmask res     1
0081            pointer res     2
0083            filein  res     1
0084            ascii   res     1
0085            temp    res     1
0086            curx    res     1
0087            cury    res     1
0088            refx    res     1
0089            refy    res     1
008A            bitcnt  res     1
008B            lincnt  res     1
                ;
                ; DOS-65 variables & entries
        AA00    buffer  equ     $aa00
        C006    command equ     $c006           execute command
        C020    in      equ     $c020           get a char
        C023    out     equ     $c023           put a char
        C026    inecho  equ     $c026           get and put char
        C029    bufin   equ     $c029           input in buffer
        C02F    crlf    equ     $c02f           print <cr> and <lf>
        C032    spa     equ     $c032           print a space
        C035    hnout   equ     $c035           low nibble as a ASCII
        C038    hexout  equ     $c038           hex as two ASCII
        C03B    print   equ     $c03b           print string after call
        C03E    aschex  equ     $c03e           convert ascii to hex
        C041    loupch  equ     $c041           convert lower to upper
```

```
C056    redrin  equ     $c056           redirect input
C068    sopt    equ     $c068           scan options
C06B    spar    equ     $c06b           scan parameters
D003    sread   equ     $d003           single read
D00C    swrite  equ     $d00c           sigle write
D039    create  equ     $d039           open/create a file
D04B    close   equ     $d04b           close file
D0B7    ermes   equ     $d0b7           error message
        ;
F024    posit   equ     $f024
        ;
        ; Mask for options
0080    bitch   equ     %10000000       change bits
0040    displ   equ     %01000000       display
0020    eprom   equ     %00100000       set eprom
0010    negat   equ     %00010000       set invers (negative)
0008    printer equ     %00001000       NOT READY
0004    ram     equ     %00000100       set RAM
0002    schars  equ     %00000010       save char. set
0001    toggle  equ     %00000001       high/low RAM
        ;
        ; *** Data ***
        ; At this address you can read or write into the
        ; RAM/EPROM. The address is then given by portA
        ; and portB of the PIA.
E142    data    equ     $e142
        ;
        ; *** PIA ***
        ; Normally the VDU board generates the address
        ; to read into the RAM/EPROM character generator.
E144    portA   equ     $E144           low address
E145    contrA  equ     $e145           if CA2=1, then EPROM is selected
E146    portB   equ     $e146           b0, b1, b2 & b3 = high address
E147    contrB  equ     $e147           if CB2=1, then high 4K selected
        ;
        yi      macro   src
                ldy     #src&255
                lda     #src>>8
                endm
        ;
        ;
1000            org     $1000
        ;
1000 4C C013    setvf   jmp     main
1003 C8C5CCD0           fcc     $c8,$c5,$cc,$d0
1007 20 3BC0    phelp   jsr     print
100A 0C                 fcc     $0c             clear screen
100B 5574696C69         fcc     'Utility to set the VDU character generator which'
103B 0D69732070         fcc     '\ris placed on a new board with RAM facility.'
1067 0D466F7220         fcc     '\rFor more details: De 6502 kenners nr. xx p. xx'
1096 0D53796E74         fcc     '\rSyntax: \Ei SETVFRONT [-CDENPST] file \En'
10BE 0D4F707469         fcc     '\rOptions: without a file:'
10D7 0D20202020         fcc     '\r             none: print this help info'
10FE 0D20202020         fcc     '\r              -D : print actual character set'
112B 0D20202020         fcc     '\r              -E : set EPROM character set'
1155 0D20202020         fcc     '\r              -R : set RAM character set'
117D 0D20202020         fcc     '\r              -S : save actual character set'
11A9 0D20202020         fcc     '\r              -T : toggle between two char. sets in
RAM'
11E0 0D20202020         fcc     '\r          with a file name:'
11FB 0D20202020         fcc     '\r              -C : change bit pattern'
1220 0D20202020         fcc     '\r              -D : print  actual character  set and
load file'
125B 0D20202020         fcc     '\r                   -N : set invers in second part of
char. set'
1294 0D20202020         fcc     '\r              -P : set also printer front'
12BD 0D41626272         fcc     '\rAbbreviation: \Ei SETVF \En'
12D7 0D4578616D         fcc     '\rExample: SETVF -C xyz.rom'
12F1 0D49662074         fcc     '\rIf there  is not  a file  name, you  will see this
help'
1325 0D696E666F         fcc     '\rinfo again.  If the  filename xyz.rom is found, it
will'
```

```
135B 0D6265206C          fcc     '\rbe loaded into RAM  and then  the program  ask you
which'
1391 0D63686172          fcc     '\rcharacter you desire to change (-C option).'
13BD 0D00                fcc     '\r',0
13BF 60                  rts                     back to caller
                 ;
13C0 20 68C0     main    jsr     sopt            get options
13C3 4344454E50          fcc     'CDENPRST',0
13CC 90 03               bcc     1.f
13CE 20 B7D0             jsr     ermes
13D1 86 80       1       stx     optmask         save the option
13D3 84 81               sty     pointer         save file pointer address
13D5 85 82               sta     pointer+1
13D7 20 A914             jsr     tstpia          test if board is present
                 ; *** Check options without a filename ***
                 ; Display the actual characterset

13DA A5 80       1       lda     optmask         get option mask
13DC 29 40               and     #displ          mask D-option
13DE F0 06               beq     1.f
13E0 20 B215             jsr     dchar           display actual char.
13E3 4C 0A14             jmp     tsfile          get file
                 ; Switch to char. set in EPROM
13E6 A5 80       1       lda     optmask         get option mask
13E8 29 20               and     #eprom          mask E-option
13EA F0 03               beq     1.f
13EC 4C 1815             jmp     setepr          set eprom
                 ; Switch to RAM
13EF A5 80       1       lda     optmask         get option mask
13F1 29 04               and     #ram            mask R-option
13F3 F0 03               beq     1.f
13F5 4C 2115             jmp     setram          set ram
                 ; Switch between low/high char. set in RAM
13F8 A5 80       1       lda     optmask         get option mask
13FA 29 01               and     #toggle         mask T-option
13FC F0 03               beq     1.f
13FE 4C 2A15             jmp     tggl            set high/low ram
                 ; Save actual character set
1401 A5 80       1       lda     optmask         get oprion mask
1403 29 02               and     #schars         mask S-option
1405 F0 03               beq     tsfile
1407 4C 5F15             jmp     svch            save char. set
                 ; Test for file name
140A A4 81       tsfile  ldy     pointer         get file pointer back
140C A5 82               lda     pointer+1
140E A2 80               ldx     #$80            read mode
1410 20 56C0             jsr     redrin
1413 90 11               bcc     3.f             branch if no error
1415 C9 12               cmp     #$12            no filename ?
1417 D0 09               bne     2.f             branch on other error
1419 A5 80               lda     optmask         no filename, check option
141B 29 40               and     #displ          mask D-option
141D D0 06               bne     4.f             branch on D-option
141F 4C 0710             jmp     phelp           print help info
1422 20 B7D0     2       jsr     ermes
1425 60          4       rts
1426 86 83       3       stx     filein          save file number
1428 20 2115             jsr     setram          set first ram
142B 20 2A15             jsr     tggl            and take oldest part
                 ; Read file into RAM
142E A6 83       rfile   ldx     filein          get file number back
1430 20 03D0             jsr     sread           read one byte
1433 B0 0D               bcs     1.f             end ?
1435 8D 42E1             sta     data            no, store byte in ram
1438 EE 44E1             inc     portA           increase pseudo address
143B D0 F1               bne     rfile
143D EE 46E1             inc     portB
1440 D0 EC               bne     rfile           branch always
                 ; Change bit pattern
1442 A5 80       1       lda     optmask         get option mask
1444 29 80               and     #bitch          mask C-option
1446 F0 03               beq     1.f
```

```
1448 4C 6816            jmp     cbits           change bits
                ; Get file and display it
144B A5 80      1       lda     optmask         get option mask
144D 29 10              and     #negat          mask N-option
144F F0 32              beq     1.f
1451 A9 00              lda     #$00            reset pseudo address
1453 8D 44E1            sta     portA
1456 8D 46E1            sta     portB
1459 AD 46E1    2       lda     portB           get first part char. set
145C 29 07              and     #%00000111      by making b3 low
145E 8D 46E1            sta     portB
1461 A9 FF              lda     #%11111111      prepare for invers
1463 4D 42E1            eor     data            invers data
1466 48                 pha                     save it on stack
1467 AD 46E1            lda     portB           set high part char. set
146A 09 08              ora     #%00001000      by making b3 high
146C 8D 46E1            sta     portB
146F 68                 pla                     get invers data back
1470 8D 42E1            sta     data            store in ram
1473 EE 44E1            inc     portA           increase pseudo address
1476 D0 E1              bne     2.b
1478 EE 46E1            inc     portB
147B AD 46E1            lda     portB
147E C9 10              cmp     #$10            check for end
1480 D0 D7              bne     2.b
1482 60                 rts                     back to caller
1483 A5 80      1       lda     optmask         get option mask
1485 29 08              and     #printer        mask P-option
1487 F0 1F              beq     1.f
1489 20 3BC0            jsr     print
148C 0D1B692050         fcc     '\r\Ei P-option not ready \En\r\r',0
14A8 60         1       rts                     no more options, back to caller
                ; *** Subroutines ***
                ; Test if board is present
14A9 AD 45E1    tstpia  lda     contrA          check if PIA is already init.
14AC 29 34              and     #%00110100
14AE C9 34              cmp     #%00110100
14B0 F0 30              beq     1.f
14B2 A2 FF      init    ldx     #$ff            initialisation of the PIA
14B4 8E 44E1            stx     portA           portA as output
14B7 8E 46E1            stx     portB           portB as output
14BA AD 44E1            lda     portA           get data of dir-reg. A
14BD C9 FF              cmp     #$ff            port as output?
14BF D0 2A              bne     2.f             branch on error
14C1 AD 46E1            lda     portB
14C4 C9 FF              cmp     #$ff
14C6 D0 23              bne     2.f
14C8 A9 3C              lda     #%00111100
14CA 8D 45E1            sta     contrA          CA2 & CB2 as output (high)
14CD 8D 47E1            sta     contrB          and select i/o registers
14D0 AD 45E1            lda     contrA          get data of i/o reg.A
14D3 29 3C              and     #%00111100      mask i/o reg.
14D5 C9 3C              cmp     #%00111100      and verify
14D7 D0 12              bne     2.f
14D9 AD 47E1            lda     contrB
14DC 29 3C              and     #%00111100
14DE C9 3C              cmp     #%00111100
14E0 D0 09              bne     2.f
14E2 A9 00      1       lda     #$00            reset pseudo address
14E4 8D 44E1            sta     portA           portA = low address
14E7 8D 46E1            sta     portB           portB = high address
14EA 60                 rts
14EB 20 3BC0    2       jsr     print
14EE 1B20537065         fcc     '\E Special 8k RAM board not present \En\r',0
1515 68                 pla                     distroy return address
1516 68                 pla
1517 60                 rts                     back to main caller
                ;
1518 AD 45E1    setepr  lda     contrA          get status of control reg. A
151B 09 08              ora     #%00001000      make CA2 high
151D 8D 45E1            sta     contrA          select EPROM
1520 60                 rts
```

```
                        ;
1521 AD 45E1    setram  lda     contrA          get status of control reg. A
1524 29 F7              and     #%11110111      make CA2 low
1526 8D 45E1            sta     contrA          select RAM
1529 60                 rts
                        ;
152A AD 47E1    tggl    lda     contrB          get status of control reg. B
152D 29 08              and     #%00001000      mask CB2 status
152F F0 0A              beq     seth            branch if CB2 is high
1531 AD 47E1            lda     contrB          get status of control reg. B
1534 29 F7              and     #%11110111      make b3 low
1536 8D 47E1            sta     contrB          select low 4K in RAM
1539 D0 08              bne     tggle           branch always to end
153B AD 47E1    seth    lda     contrB          get status of control reg. B
153E 09 08              ora     #%00001000      set b3
1540 8D 47E1            sta     contrB          select high 4K in RAM
1543 60         tggle   rts                     back to caller
                ; print a byte as 8 bits using . and X
1544 85 85      prbits  sta     temp            save byte
1546 A9 08              lda     #$08            prepare bitcounter
1548 85 8A              sta     bitcnt
154A 26 85      1       rol     temp            get bits
154C B0 07              bcs     2.f             branch if bit is high
154E A9 2E              lda     #'.             it was a low set bit
1550 20 23C0            jsr     out             print . for a low bit
1553 D0 05              bne     3.f             branch always
1555 A9 58      2       lda     #'X             it was a high set bit
1557 20 23C0            jsr     out             print X for a high bit
155A C6 8A      3       dec     bitcnt          dec counter
155C D0 EC              bne     1.b             loop for 8 bits
155E 60                 rts                     back to caller
                        ;
155F 20 3BC0    svch    jsr     print
1562 0D456E7465         fcc     '\rEnter new file name: ',0
1579 20 29C0            jsr     bufin           get file name in buffer
157C A9 AA              lda     #buffer>>8      buffer is file name pointer
157E A0 00              ldy     #buffer&255
1580 A2 E0              ldx     #%11100000      rwd mode
1582 20 39D0            jsr     create          open/create file
1585 90 03              bcc     1.f             branch or
1587 4C B7D0            jmp     ermes           print error
158A A9 00      1       lda     #$00            reset pseudo address
158C 8D 44E1            sta     portA
158F 8D 46E1            sta     portB
1592 AD 42E1    2       lda     data            get ram data
1595 20 0CD0            jsr     swrite          and save it on disk
1598 90 06              bcc     1.f             branch or
159A 20 B7D0            jsr     ermes           print error message
159D 4C 4BD0            jmp     close           and close the file
15A0 EE 44E1    1       inc     portA           no error, so increase
15A3 D0 ED              bne     2.b             the pseudo address
15A5 EE 46E1            inc     portB
15A8 AD 46E1            lda     portB
15AB C9 10              cmp     #$10            last byte ?
15AD D0 E3              bne     2.b             loop for 4K
15AF 4C 4BD0            jmp     close           end, close file and return
                        ;
15B2 20 3BC0    dchar   jsr     print
15B5 0C                 fcc     $0c             clear screen
15B6 1B6920436F         fcc     '\Ei Contents of the actual VDU character set \En'
15E4 0D0D0D             fcc     '\r\r\r'
15E7 2020202030         fcc     '    0  1  2  3  4  5  6  7'
1601 2020382020         fcc     '  8  9  a  b  c  d  e  f\r',0
161B A9 00              lda     #$00            reset accu
161D AA                 tax                     accu is saved in the X-reg.
161E 20 2FC0    nline   jsr     crlf
1621 20 35C0            jsr     hnout           print low nibble of accu
1624 20 32C0            jsr     spa
1627 20 32C0    nchar   jsr     spa             always two spaces between char.
162A 20 32C0            jsr     spa
162D 8A                 txa                     get the value and check it
162E C9 20              cmp     #$20            smaller than $20 ?
```

```
1630 90 05              bcc     grafic          yes
1632 20 23C0            jsr     out             no, it was a normal ASCII
1635 D0 1A              bne     inch            branch always
1637 A9 1B      grafic  lda     #$1b            print a control character
1639 20 23C0            jsr     out             smaller than $20
163C A9 46              lda     #'F             start escape sequence
163E 20 23C0            jsr     out             for grafic mode
1641 8A                 txa                     get the character
1642 09 40              ora     #%01000000      set b6 (capital ASCII)
1644 20 23C0            jsr     out             print this pseudo code
1647 A9 1B              lda     #$1b            escape sequence to leave
1649 20 23C0            jsr     out             the grafic mode
164C A9 47              lda     #'G
164E 20 23C0            jsr     out
1651 8A         inch    txa                     get actual value
1652 18                 clc                     prepare for ADD
1653 69 10              adc     #$10            add $10 for the next colom
1655 B0 03              bcs     incl            if last one, next line
1657 AA                 tax                     save this new value
1658 D0 CD              bne     nchar           print next value (character)
165A 69 00      incl    adc     #$00            add $01 for next row (C=1)
165C AA                 tax                     save this new value
165D 29 0F              and     #$0f            last line ?
165F D0 BD              bne     nline           no, start at next line
1661 20 2FC0            jsr     crlf            yes
1664 20 2FC0            jsr     crlf
1667 60                 rts                     back to caller
                ;
1668 20 3BC0    cbits   jsr     print
166B 0C                 fcc     $0c
166C 1B69204368         fcc     '\Ei Change bit pattern \En'
1684 0D0D4B6579         fcc     '\r\rKeys:   Q : exit'
1696 0D20202020         fcc     '\r         + : next character'
16B1 0D20202020         fcc     '\r         - : previous character'
16D0 0D20202020         fcc     '\r        LF : next line'
16E6 0D20202020         fcc     '\r        VT : previous line'
1700 0D20202020         fcc     '\r         . : enter a 0'
1716 0D20202020         fcc     '\r         X : enter a 1'
172C 0D20202020         fcc     '\r         ? : print this help info again'
1753 0D0D456E74         fcc     '\r\rEnter ASCII value in hex: $',0
1771 20 20C0    1       jsr     in              get input
1774 85 85              sta     temp            save it to print it
1776 20 41C0            jsr     loupch          convert to upper character
1779 20 3EC0            jsr     aschex          convert to hex nibble
177C B0 F3              bcs     1.b             try again
177E 85 84      2       sta     ascii           save this first nibble
1780 A5 85              lda     temp            get input back
1782 20 23C0            jsr     out             and print it
1785 20 20C0    1       jsr     in              get low nibble
1788 20 41C0            jsr     loupch          convert to upper character
178B 20 3EC0            jsr     aschex          convert ascii to hex nibble
178E B0 F5              bcs     1.b
1790 0A         2       asla                    shift low to high nibble
1791 0A                 asla
1792 0A                 asla
1793 0A                 asla
1794 A2 04              ldx     #$04            prepare loop
1796 0A         1       asla                    shift low nibble
1797 26 84              rol     ascii           into ascii (make byte)
1799 CA                 dex
179A D0 FA              bne     1.b             4 times
179C 20 3BC0    prchars jsr     print
179F 0C                 fcc     $0c
17A0 1B69204368         fcc     '\Ei Change bit pattern \En'
17B8 0D0D415343         fcc     '\r\rASCII : ',0
17C3 A5 84              lda     ascii           get byte
17C5 20 23C0            jsr     out             print as character
17C8 20 3BC0            jsr     print
17CB 0D56616C75         fcc     '\rValue : $',0
17D6 A5 84              lda     ascii           get byte
17D8 20 3BC0            jsr     hexout          and print hex
17DB A9 00              lda     #$00            reset pointer+1
```

```
17DD 85 82              sta     pointer+1
17DF A5 84              lda     ascii           ascii value in pointer
17E1 85 81              sta     pointer
17E3 A2 04              ldx     #$04            prepare for loop
17E5 06 81      1       asl     pointer         shift ascii value 4 times
17E7 26 82              rol     pointer+1       (low nibble = 16 char. lines)
17E9 CA                 dex
17EA D0 F9              bne     1.b             loop
17EC A5 81              lda     pointer         pointer = pseudo address
17EE 8D 44E1            sta     portA
17F1 A5 82              lda     pointer+1
17F3 8D 46E1            sta     portB
17F6 A2 01              ldx     #$01            prepare cursor
17F8 A0 07              ldy     #$07
17FA 20 24F0            jsr     posit           place the cursor
17FD 20 3BC0            jsr     print
1800 2020202020         fcc     '       EPROM           RAM             NEW\r'
1829 2020202020         fcc     '     76543210        76543210        76543210\r\r',0
1857 A9 0A              lda     #$0a
1859 85 8B              sta     lincnt          print 10 lines
185B AD 44E1    1       lda     portA           get low address
185E 29 0F              and     #$0f            mask low nibble
1860 20 35C0            jsr     hnout           print line number
1863 20 1815            jsr     setepr          set eprom
1866 A0 04              ldy     #$04            loop to print
1868 20 32C0    2       jsr     spa             4 spaces
186B 88                 dey
186C D0 FA              bne     2.b
186E AD 42E1            lda     data            print data from EPROM
1871 20 4415            jsr     prbits
1874 20 2115            jsr     setram          set ram
1877 A0 07              ldy     #$07            loop to print
1879 20 32C0    3       jsr     spa             7 spaces
187C 88                 dey
187D D0 FA              bne     3.b
187F AD 42E1            lda     data            print data from ram
1882 20 4415            jsr     prbits
1885 A0 07              ldy     #$07            loop to print
1887 20 32C0    3       jsr     spa             7 spaces
188A 88                 dey
188B D0 FA              bne     3.b
188D AD 42E1            lda     data            print data from ram again
1890 20 4415            jsr     prbits
1893 20 2FC0            jsr     crlf            next line
1896 EE 44E1            inc     portA           increase pseudo address
1899 C6 8B              dec     lincnt          adjust line counter
189B D0 BE              bne     1.b             loop for 10 lines
189D A2 24              ldx     #$24
189F A0 0A              ldy     #$0a
18A1 86 88              stx     refx
18A3 84 89              sty     refy            cursor reference in pattern
18A5 86 86              stx     curx
18A7 84 87              sty     cury            cursor in reference
18A9 AD 44E1            lda     portA           pseudo address to first
18AC 29 F0              and     #$f0            caracter line
18AE 8D 44E1            sta     portA
18B1 A9 00              lda     #$00            reset bit- and line counter
18B3 85 8A              sta     bitcnt
18B5 85 8B              sta     lincnt
18B7 20 C619    1       jsr     prlin           print first character line
18BA 20 20C0    getin   jsr     in              get keys:
18BD 20 41C0            jsr     loupch
18C0 C9 51              cmp     #'Q             exit ?
18C2 F0 3E              beq     nxtch
18C4 C9 0A              cmp     #$0a            <LF> ?
18C6 D0 06              bne     1.f
18C8 20 D719            jsr     inclin
18CB 4C BA18            jmp     getin
18CE C9 0B      1       cmp     #$0b            <VT> ?
18D0 D0 06              bne     1.f
18D2 20 F819            jsr     declin
18D5 4C BA18            jmp     getin
```

```
18D8 C9 3F      1       cmp     #'?             help ?
18DA D0 03              bne     1.f
18DC 4C 6816            jmp     cbits
18DF C9 2D      1       cmp     #'-             previous character ?
18E1 D0 05              bne     1.f
18E3 C6 84              dec     ascii
18E5 4C 9C17            jmp     prchars
18E8 C9 2B      1       cmp     #'+             next character ?
18EA D0 05              bne     1.f
18EC E6 84              inc     ascii
18EE 4C 9C17            jmp     prchars
18F1 C9 2E      1       cmp     #'.             enter 0 ?
18F3 D0 03              bne     1.f
18F5 4C 6919            jmp     cbit
18F8 C9 58      1       cmp     #'X             enter X ?
18FA D0 03              bne     1.f
18FC 4C 6919            jmp     cbit
18FF 4C BA18    1       jmp     getin
                ;
1902 A2 01      nxtch   ldx     #$01
1904 A0 15              ldy     #$15
1906 20 24F0            jsr     posit
1909 20 3BC0            jsr     print
190C 4E65787420         fcc     'Next character ? (Y*/N) ',0
1925 20 26C0            jsr     inecho
1928 20 41C0            jsr     loupch
192B C9 4E              cmp     #'N
192D F0 03              beq     1.f
192F 4C 6816            jmp     cbits
1932 20 3BC0    1       jsr     print
1935 0D53617665         fcc     '\rSave this character set ? (Y*/N) ',0
1958 20 26C0            jsr     inecho
195B 20 41C0            jsr     loupch
195E C9 4E              cmp     #'N             exit only on "N"
1960 F0 03              beq     1.f
1962 4C 5F15            jmp     svch            save actual character set
1965 20 2FC0    1       jsr     crlf            give always a new line
1968 60                 rts                     back to caller
                ;
1969 48         cbit    pha
196A A9 08              lda     #$08
196C 85 8A              sta     bitcnt
196E 68                 pla
196F C9 51      1       cmp     #'Q             exit ?
1971 D0 03              bne     8.f
1973 4C 0219            jmp     nxtch
1976 C9 3F      8       cmp     #'?             help ?
1978 D0 03              bne     8.f
197A 4C 6816            jmp     cbits
197D C9 2E      8       cmp     #'.             enter 0 ?
197F F0 06              beq     3.f
1981 C9 58              cmp     #'X             enter 1 ?
1983 F0 0D              beq     4.f
1985 D0 2E              bne     6.f             branch always
1987 A9 2E      3       lda     #'.             print "."
1989 20 23C0            jsr     out
198C 18                 clc                     clear carry to
198D 26 85              rol     temp            shift a 0 in
198F 4C 9A19            jmp     5.f
1992 A9 58      4       lda     #'X             print "X"
1994 20 23C0            jsr     out
1997 38                 sec                     set carry to
1998 26 85              rol     temp            shift a 1 in
199A C6 8A      5       dec     bitcnt
199C D0 17              bne     6.f             loop for 8 bits
199E A5 85              lda     temp            transfer new byte
19A0 8D 42E1            sta     data            in charracter generator
19A3 A2 09              ldx     #$09
19A5 A0 03              ldy     #$03
19A7 20 24F0            jsr     posit
19AA A5 84              lda     ascii           get charecter
19AC 20 23C0            jsr     out
```

```
.9AF 20 BE19            jsr     curpos          place cursor back
.9B2 4C BA18            jmp     getin           wait for input
.9B5 20 20C0    6       jsr     in              input routine
.9B8 20 41C0            jsr     loupch          during the 8 bit loop
.9BB 4C 6F19            jmp     1.b
                ;
.9BE A6 86      curpos  ldx     curx            adjust the cursor
.9C0 A4 87              ldy     cury
.9C2 20 24F0            jsr     posit
.9C5 60                 rts
                ;
.9C6 A4 87      prlin   ldy     cury            print actual line
.9C8 A6 88              ldx     refx
.9CA 20 24F0            jsr     posit
.9CD AD 42E1            lda     data            get original data
.9D0 20 4415            jsr     prbits
.9D3 20 BE19            jsr     curpos          cursor back
.9D6 60                 rts
                ;
19D7 EE 44E1    inclin  inc     portA           increase pseudo address
19DA E6 87              inc     cury            and cursor
19DC E6 8B              inc     lincnt          increase line counter
19DE A5 8B              lda     lincnt          and check for max.
19E0 C9 0A              cmp     #$0a
19E2 D0 10              bne     1.f
19E4 AD 44E1            lda     portA           restore pseudo adddress
19E7 29 F0              and     #$f0            counter
19E9 8D 44E1            sta     portA
19EC A9 00              lda     #$00            restore line counter
19EE 85 8B              sta     lincnt
19F0 A5 89              lda     refy            adjust cursor
19F2 85 87              sta     cury
19F4 20 C619    1       jsr     prlin           print this new line
19F7 60                 rts
                ;
19F8 A5 8B      declin  lda     lincnt          first line ?
19FA D0 18              bne     1.f
19FC A9 09              lda     #$09            yes, set line counter to max
19FE 85 8B              sta     lincnt
1A00 AD 44E1            lda     portA           set pseudo address to max
1A03 29 F0              and     #$f0
1A05 18                 clc
1A06 69 09              adc     #$09
1A08 8D 44E1            sta     portA
1A0B 18                 clc                     adjust cursor
1A0C A5 89              lda     refy
1A0E 69 09              adc     #$09
1A10 85 87              sta     cury
1A12 D0 07              bne     2.f
1A14 CE 44E1    1       dec     portA           no, decrease pseudo address
1A17 C6 87              dec     cury            cursor
1A19 C6 8B              dec     lincnt          and line counter
1A1B 20 C619    2       jsr     prlin           print this new line
1A1E 60                 rts
                ;
          1000          end     setvf

Errors detected: 0
```

Programeren in Assembler (deel 3)

Decimaal rekenen

In deel 1 en 2 zijn optelroutines behandeld die twee hexadecimale getallen optellen en het resultaat ook weer in hexadecimaal wegzetten. Nu is hexadecimaal een erg mooi talstelsel voor gebruik in computers, zie ook deel 1, een byte is precies met twee hexadecimale karakters 0...F te beschrijven. Voor 'gewone' stervelingen is hexadecimaal echter abracadabra, men is gewend zijn getallenbrij met 0...9 voorgeschoteld te krijgen. En ziet, ook voor dit probleem is een oplossing bedacht; de BCD notatie, in goed Frans 'Binary Coded Decimal'. Decimale getallen weergegeven in binair. Dat betekent dus dat van de 16 mogelijke combinaties van 4 bits er slechts 10 gebruikt worden:

```
0 = 0000    1 = 0001    2 = 0010    3 = 0011
4 = 0100    5 = 0101    6 = 0110    7 = 0111
8 = 1000    9 = 1001
```

De codes A..F zijn in BCD dus niet toegestaan. Alles goed en wel zul je zeggen, hele leuke code, maar eh, wat gebeurt er nu als je twee van die BCD gecodeerde getallen gaat optellen? Voorbeeldje: 08 + 03 --> BCD 1000 + 0011 = 1011 --> $0B Het resultaat klopt niet want in BCD zijn de codes A..F VERBOTEN, STRICTLY PROHIBITED, NJET TOVARITSJ, AF FIKKIE! Met andere woorden, zodra het resultaat van een optelling groter wordt dan 9, moeten we er 6 bijtellen om die illegale codes te overbruggen: $0B + 06 = 1011 + 0110 = 0001 0001 = 11 BCD Nu klopt het antwoord 08 + 03 = 11 (BCD). Errug aardig hoor ik je denken, maar moet ik nou in al mijn programma's eerst checken of mijn resultaat wel in BCD blijft en zo niet hoeveel moet ik er dan bijtellen om het kloppend te krijgen? En hoe zit dan dan met BCD aftrekken? En hoe vertel ik het mijn moeder? Voorwaar problemen van niet geringe omvang!! Ook de chipboeren hadden dat in de gaten en bakten op hun chipjes ofwel een instructie mee met de flag dekkende benaming DAA, (Frans voor Decimal Adjust Accumulator), of zoals in het geval van de 6502, en daar mogen we de ontwerpers van Rockwell nog immer dankbaar voor zijn, een decimal mode op de processor itself. Die DAA instructie dient uitgevoerd te worden na iedere optelling c.q. aftrekking om ervoor te zorgen dat het resultaat in BCD notering blijft. Deze oplossing wordt o.a. toegepast in chips van Motorola. Good ol' 6502 heeft echter een hele luxe decimal mode die we met twee instructies aan en uit kunnen zetten. Die instructies zijn:

```
SED  SEt   Decimal mode
CLD  CLear Decimal mode
```

Deze instructies beinvloeden het D-bit van het statusregister P van de 6502. Het D-bit geeft aan of de ALU (Arithmetic Logic Unit) moet rekenen in binair of in decimaal. Is het zo simpel? En ik antwoord met een even simpel Ja..... (een gepaste stilte lijkt me nu wel op zijn plaats dacht ik zo).

Probeer het volgende maar eens:

Copier ADD_8.MAC uit deel 2 naar ADD_8_BCD.MAC en lees die copie in in ED (lastig lezen met twee keer in na elkaar). Voeg de SED instructie toe op de eerste regel in het ADD_8_BCD programma en de CLD instructie tussen de STA RESULT en de BCC OK instructie. Assembleer het geheel, laad het programma in memory met LOAD en voer twee BCD gecodeerde getallen in op de bekende geheugenlocaties.

N.B. getallen dus die alleen de codes 0...9 mogen bevatten!!

V.B.

```
MON> @ 0200
0200 00 83
0201 00 09
MON> E 1000
92
MON>
```

En dat klopt!! Wederom voel ik een niet te bedwingen emotie zich meester maken van mijn strottehoofd zodat mij het schrijven voor een ogenblik onmogelijk wordt gemaakt..... Gelukkig had ik nog zo'n pilletje van die aardige dokter waar ik weer helemaal kalm van wordt, zodat we verder kunnen met het verhaal:

De 6502 heeft inderdaad in decimaal gerekend en we zien dat het resultaat netjes in BCD notatie blijft. Met het ADD_8 programma had er immers $8C als antwoord gestaan. De overflow waarschuwing werkt nog steeds maar nu bij een resultaat groter dan 99 i.p.v. getallen groter dan $FF zoals bij ADD_8. Probeer maar eens. Het is ZEER belangrijk dat je de decimale mode van de 6502 weer direct uitzet als je klaar bent met de berekening. Laten we de CLD instructie achterwegen of zetten we hem pas vlak voor de RTS dan zal het systeem 'hangen' zodra je het programma probeert uit te voeren of de resultaten zijn onbetrouwbaar. De HEXOUT1 en PRINT routine verwachten namelijk de binaire mode van de 6502. Ook de rest van de I/O65 en DOS65 routines zijn nogal kieskeurig op dit punt en dus: ALTIJD UIT DIE DECIMAL MODE ZODRA JE KLAAR BENT MET DE BEWERKING.

Voor getallen tot 9999 kun je het ADD_16 programma op soortgelijke wijze aanpassen. De 8 en 16 bits aftrek routine is gemakkelijk af te leiden van ADD_8 en ADD_16 door de ADC instructies te vervangen met SBC, de CLC instructie wordt SEC en de BCC OK wordt BCS OK. Ook van deze routines is dan een decimale versie te maken door toevoeging van SED en CLD instructies op de aangegeven plaatsen. Ons pakketje reken routines wordt op diemanier al aardig uitgebreid. Maar we zijn er nog niet want zoals de meesten van jullie weten is er ook nog zoiets als vermenigvuldigen en delen, vroeger toen je nog klein was heb je dat immers allemaal op school geleerd. Nu kan de 6502 NIET vermenigvuldigen en ook NIET delen. In bijna alle processoren van de huidige generatie zit meestal wel een MUL en DIV instructie. Gelukkig is met wat hogere wiskunde eenvoudig te bewijzen dat vermenigvuldigen niets anders is dan herhaald optellen. Delen is herhaald aftrekken totdat het resultaat te klein wordt en dan bijhouden hoe vaak dat ging, als er nog iets overblijft van het getal dan is dat 'de rest'. We werken hier immers met gehele (of integer) getallen, dus geen cijfers achter de komma. Om jeugdige lezers niet al te zeer te verwarren zal ik het wiskundige bewijs van deze stellingen achterwege laten. Hoe vermenigvuldigt nu een doodnormaal mensch zoals U en ik (hoewel, doodnormaal?) een tweetal getallen:

```
  123
  156 x
 -----
  738        = 6 x 123
 6150        = 0 opschrijven, 5 x 123=615
12300 +      = 00 opschrijven, 1 x 123=123
-------
18888
```

Wat doet men nu eigenlijk? Eerst 6 x 123 = 738 gewoon opschrijven in de optelkolom, maar dan gebeurt er iets raars, je schrijft eerst een 0 op en berekent vervolgens 5 x 123 = 615 en schrijft dat voor de 0. In feite bereken je dus 5 x 1230 = 6150. Bij de laatste berekening geldt hetzelfde alleen hier met twee nullen dus in feite 1 x 12300. Als we dit nu eens door de computer laten proberen, alleen werkt die zoals we al weten met slechts twee getallen 0 en 1, de tafels die we dus uit het hoofd moeten leren zijn de binaire tafel van 0 en de binaire tafel van 1. Volgende week overhoring! Die tafels zien

er zo uit:

```
tafel van 0:    0 x 0 = 0
                0 x 1 = 0

tafel van 1:    1 x 0 = 0
                1 x 1 = 1
```

Binair optellen gaat zo:

```
0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 0 (1 onthouden --> carry)
```

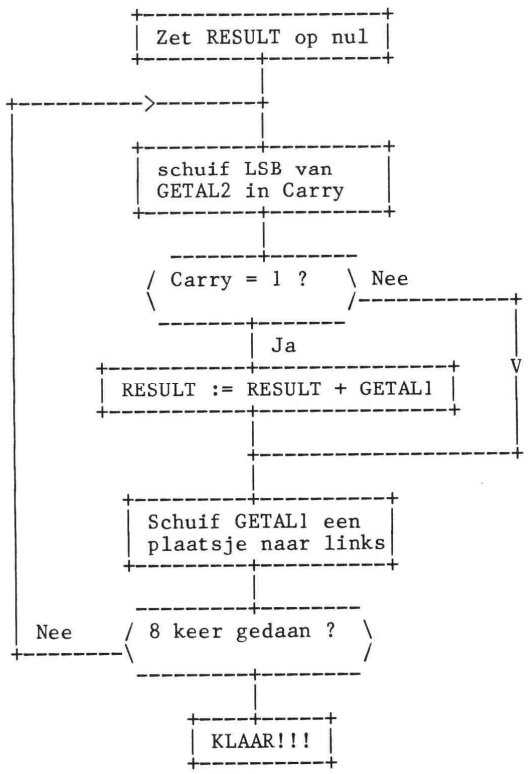
We gaan nu vermenigvuldigen in het binaire stelsel aan de hand van onze ervaringen in het decimale stelsel en met behulp van bovenstaande tafels die iedereen voor volgende week uit het hoofdmoet leren. Stel we nemen de getallen 13 en 5:

```
13(dec) = 1101(bin)
 5(dec) = 0101(bin)


 1101    GETAL1
 0101 x  GETAL2
 ------
 1101 = 1 x    1101 []
00000 = 0 x   1101 [0]
110100 = 1 x  1101 [00]
0000000 = 0 x 1101 [000]
-----------
1000001 = 0100 0001 =
          4     1   = 65 = 13 x 5
```

Zoals je ziet schuift GETAL1 a.h.w. iedere keer een bit naar links, daarna wordt het vermenigvuldigt met het overeenkomstige bit van GETAL2 en het tussenresultaat wordt opgeschreven voor de optelling. We kunnen dit nog wat vereenvoudigen door alleen iets op te tellen als het resultaat van de tussenberekening ongelijk is aan nul, dus:

```
 1101    GETAL1
 0101 x  GETAL2
 ------
 1101
110100 +
---------
1000001    RESULT
```

Hoe gieten we dit nu in een programma? Het beste gaat dit door eerst een stroomschema of in goed Frans 'flowdiagram' van het te schrijven programma te maken:

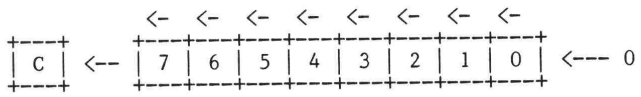


We zien in het flowdiagram twee keer het woordje 'schuif' staan. Nu heeft de 6502 een grote hoeveelheid aan schuif operaties in het instructie arsenaal:

ASL = Arithmetic Shift Left
LSR = Logic Shift Right
ROL = ROtate Left
ROR = ROtate Right

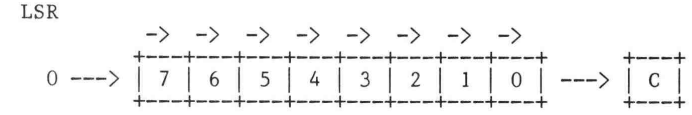
Omdat een plaatje meer zegt dan 1000 woorden zal ik deze schuif operaties verduidelijken aan de hand van de volgende diagrammen:

ASL



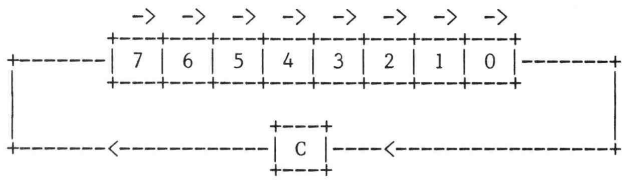
Alle bits schuiven een plaats op naar links bij bit 0 komt een '0' binnenschuiven en het hoogste bit 7 schuift in de Carry. Omdat ieder bit precies 2x zoveel waard is geworden mag je ook zeggen dat het resultaat met 2 is vermenigvuldigt (vandaar de term Arithmetic = rekenkundig).

LSR



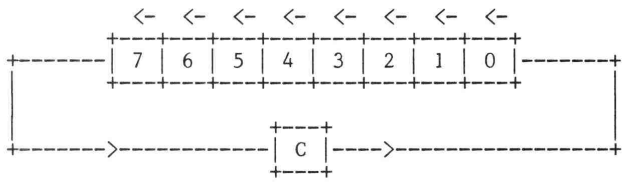
Precies de andere kant op gaat het met deze operatie, alle bits schuiven een plaatsje op naar rechts. Het zevende bit (MSB) wordt 0 en het laagste bit (LSB) schuift in de Carry. Deze bewerkingheet 'Logical Shift Right' omdat bij 'Arithmetic Shift Right' het tekenbit (MSB) niet wordt aangetast (dat wordt dus naar zichzelf gecopieerd). Nu heeft onze 6502 helemaal geen ASR dus waar heb ik het eigenlijk over?

ROR



Alle bit schuiven een plaatsje naar rechts, daarbij schuift het LSB in de Carry en het MSB krijgt de oude waarde van de Carry. Voor de oplettende lezertjes zal de volgende operatie zeer zeker een verassing zijn want bij

ROL



schuiven alle bits naar links!! Het MSB schuift daarbij in de Carry en het LSB krijgt de oude waarde van de Carry. Maar genoeg geschoven, we gaan nu het flowdiagram omzetten in een werkend programma:

```
; File    : MULT_8x8.MAC
; Purpose : Multiply two 8 bit numbers

HEXOUT1 EQU     $C038           ; print A in hexadecimaal
PRINT1  EQU     $C03B           ; print tekst tot aan <NULL>
CRLF    EQU     $C02F           ; print <CR><LF>

; Data section

        ORG     $0200

GETAL1  FCC     0               ; 8 bits operand
GETAL2  FCC     0               ; 8 bits operand
RESULT  FDB     0               ; 16 bits result
TMP     FCC     0               ; tijdelijk hulp byte
;
; Code section
;
        ORG     $1000
```

```
MUL_8x8 LDA     #0              ; zet RESULT op nul
        STA     RESULT
        STA     RESULT+1        ; 8 x 8 levert 16 bits resultaat
        STA     TMP             ; nodig voor GETAL1 schuif
        LDX     #8              ; gebruik X als teller
LOOP    LSR     GETAL2          ; schuif LSB van GETAL2 in de Carry
        BCC     NO_ADD          ; het bit is nul, dus we hoeven
                                ; niets op te tellen, het tussen-
                                ; resultaat is immers toch nul
        LDA     RESULT
        CLC
        ADC     GETAL1          ; RESULT := RESULT + GETAL1
        STA     RESULT
        LDA     RESULT+1
        ADC     TMP
        STA     RESULT+1
NO_ADD  ASL     GETAL1          ; schuif GETAL1 een plaatsje naar
                                ; links met een 0 op bit 0
        ROL     TMP             ; schuif het MSB (bit 7) van GETAL1
                                ; via de Carry in TMP
        DEX                     ; teller := teller - 1
        BNE     LOOP            ; herhaal totdat teller = 0
        JSR     CRLF
        LDA     RESULT+1        ; laat resultaat zien op scherm
        JSR     HEXOUT1
        LDA     RESULT
        JSR     HEXOUT1
        JSR     CRLF
        RTS

        END     MUL_8x8         ; that's all folks!!
```

Voer het programma in met ED en save het in de file MUL_8x8.MAC, Assembleer dit programma op de gebruikelijke manier en laad het in het geheugen:

```
$ as mul_8x8
Pass 1...
Pass 2...
Last assembled address: 1040
Errors detected: 0
$ load mul_8x8.bin
```

Ga vervolgens in de MONITOR:

```
MON> @ 200
0200 00 05      --> $05 = (0 x 16) + (5 x 1) =  5
0201 00 50      --> $50 = (5 x 16) + (0 x 1) = 80

MON>e 1000
0190     ---> = (1 x 256) + (9 x 16) + (0 x 1) = 256 + 144 = 400
```

Het antwoord klopt!! Probeer zelf maar wat amdere getallen. Je merkt dan meteen dat zowel GETAL1 als GETAL2 door het MUL_8x8 programma worden aangetast d.w.z. de waarde die je erin zet is na uitvoering verdwenen. Wil je die getallen bewaren dan zul je ze dus eerst in twee andere geheugen locaties moeten bewaren. We hebben in dit programma voor het eerst gebruik gemaakt van het index register X en wel als teller. We moeten immers bijhouden hoeveel bits we al gedaan hebben. Nu is register X daar heel geschikt voor want de 6502 heeft een hele leuke instuctie DEX (in goed Frans 'DEcrement index X') die

de waarde van het X register met precies 1 vermindert. Dus niet omslachtig:

```
LDA TELLER
SEC
SBC #1
STA TELLER
```

maar slecht een enkele instructie die bovendien slechts een byte lang is tegen minimaal 7 bytes zoals in bovenstaand subprogrammaatje. Bovendien kun je nadat de instructie is uitgevoerd direct testen of X misschien al nul is geworden doordat de DEX instructie de Z-flag van het status register P aantast. Afhankelijk van de waarde van X wordt de Z-flag gezet (X is dan nul) of gewist (X is niet nul). De Z-flag staat voor ZERO flag (zie ook deel 1) en dat is eigenlijk een naam die de lading volledig dekt want met deze flag kun je testen of iets nul is of niet. Zoals we ook al in deel 2 hebben gezien kun je zo'n flag het gemakkelijkste testen met een branch-instructie. In deel 2 was dat de BCC en de BCS en nu komt daarbij de:

```
BNE  Branch if Not Equal (to zero)
BEQ  Branch if EQual (to zero)
```

in BASIC:

```
IF (Z=1) THEN GOTO ......  (BEQ)
IF (Z=0) THEN GOTO ......  (BNE)
```

Dus als we in ons programma direct na de DEX instructie een BNE opnemen dan zal de branch precies 8x worden uitgevoerd want daarna is X=0 dus de test is niet langer waar.

Als we naar het programma kijken dan is het eigenlijk een recht toe, recht aan oplossing van ons probleem. Het programma werkt wel maar is niet erg optimaal. Nu is dat voor een enkele vermenigvuldiging niet erg, maar stel dat we deze routine willen gebruiken in een real-time procesbesturing waarbij duizenden vermenigvuldigen per seconde gedaan moeten worden. In dat geval is het erg belangrijk om het programa zo optimaal mogelijk te maken qua snelheid. Ook proberen we de lengte van het programma in te korten, bij kortere algorithmes voor hetzelfde probleem blijft er meer geheugenruimte over voor data of die ene routine die het geheel nog flitsender maakt dan het al is.

Laten we maar eens kijken waar we op kunnen besparen:

We gebruiken steeds geheugenlocaties op bladzijde 2 van ons RAM geheugen. Nu heb ik het in deel 1 als eens gehad over het zero page geheugen. Dat zijn dus alle adressen van $0000...$00FF, de oplettende lezer ziet direct dat bij deze adressen de bovenste 8 bits altijd 0 zijn m.a.w. we kunnen deze adressen ook met een 8 bits getal 'aanspreken' i.p.v. met de gebruikelijke 16 bits. Nu hadden de chipbakkers van Rockwell dat ook in de gaten en daarom bedachten ze een adresseermethode waarbij het adres van de geheugenlocatie bepaald wordt door een enkel byte, orgineel als die jongens zijn noemden ze dat de ZERO PAGE adressering. Wat is nu het voordeel zul je vragen? Wel dat is eenvoudig te beantwoorden, omdat de 6502 nu slechts een byte hoeft in te lezen om te weten waar ie de data vandaan moet halen, kan de instructie veel sneller worden uitgevoerd. Die instructie tijden worden uitgedrukt in het aantal clock cycles die nodig zijn om de gehele instructie uit te voeren. Instructies die betrekking hebben op interne registers zoals de Accumulator kunnen veel sneller worden uitgevoerd dan instructies die via het RAM werken. Zetten we bijvoorbeeld de instructie tijden van de ASL instructie naast elkaar:

Accumulator	Absoluut adres	Zero page
1	3	2

dan zien we dit duidelijk geillustreerd. Draait de 6502 bijvoorbeeld op een clock frequentie van 1 Mhz dan duurt de instructie ASL A dus 1 uS, de instructie ASL $0200 duurt 3 uS en ASL $30 duurt 2 uS. Dat lijkt niet veel uit te maken totdat je deze

Programeren in Assembler (deel 4)

PANIEK

Sinds lange tijd weer eens ingelogd op ons aller BBS (053-303902) en wat van de laatste berichten doorgekeken. Plotseling valt mijn nietsvermoedende oog op een bericht van onze enige echte redacteur, editor, secretaresse en een-mans-redactie, kortom Gert van Opbroek (applaus!!). Sluitingsdatum voor uP Kenner #63 is 4 Augustus staat er!! En het is nu 2 Augustus!!!! En ik heb mijn deel 4 nog niet af!!!!!!!! PANIEK!!!!!!!!!!!!!!!!!

Dankzij de moderne pharmaceutische industrie weer helemaal kalm geworden, toch maar weer de draad opgepakt in een poging het 4e deel op tijd af te krijgen. Excuses indien de kwaliteit van het gebodene wellicht ietwat onder het goede (lees geniale) niveau ligt zoals U dat van mij gewend bent.

In het 3e deel heb ik een aantal schuif functies behandeld die de 6502 aan boord heeft. Deze schuif functies bleken weer heel erg goed toepasbaar voor een vermenigvuldig routine. Na opzetten van een eerste ruwe versie aan de hand van een flowdiagram, is deze routine daarna op een aantal punten verbeterd. O.a. door gebruik te maken van zero-page adressering (hogere snelheid, korter programma) en een betere keuze van registers. Ook werd voor het eerst het index register toegepast, welliswaar nog als eenvoudige teller, maar geduld, het betere werk komt nog. Ik zal in deze aflevering de laatste rekenroutine behandelen t.w. een deelroutine met een 8 bits deler op een 16-bits deeltal, het resultaat is dan 8 bits.

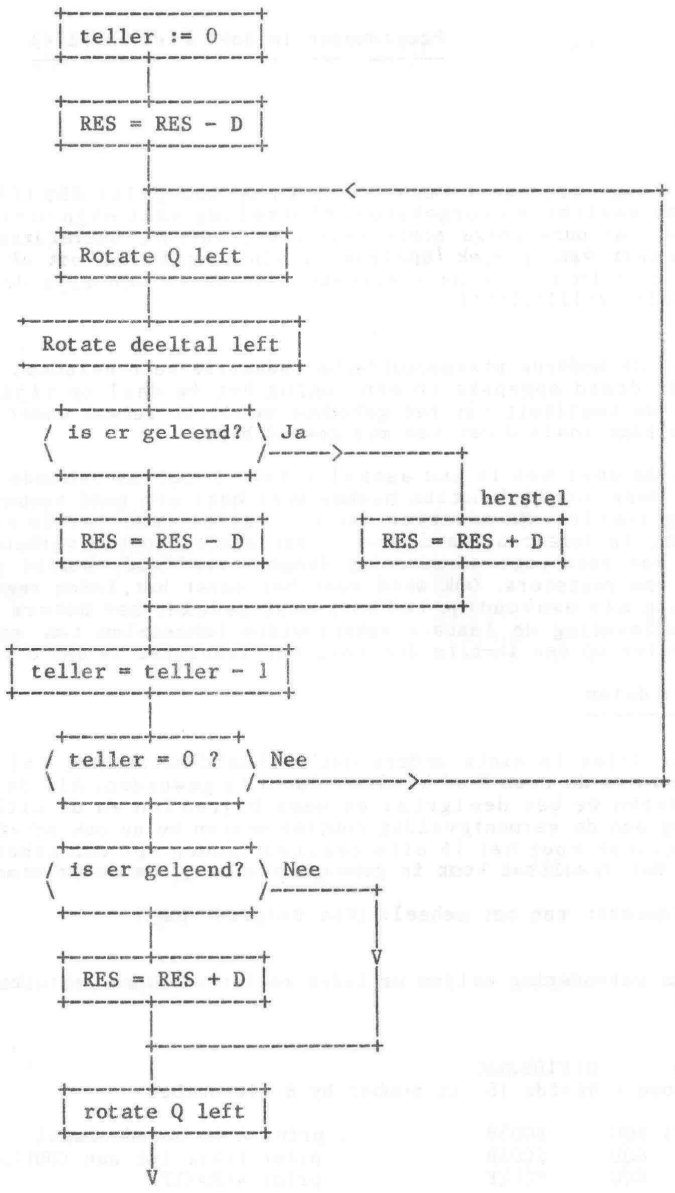
Integer delen

Integer delen is niets anders dan herhaald aftrekken, tellen hoe vaak dat gaat en stoppen als de rest 0 of kleiner dan 0 is geworden. Als de rest kleiner dan nul is dan moeten we het deelgetal er weer bijtellen en de uitkomst met een verlagen. Analoog aan de vermenigvuldig routine moeten we nu ook schuiven, dus de keuze van de accumulator voor het 16 bits deeltal (samen met een geheugenplaats B) is weer de beste. Het resultaat komt in geheugenplaats Q, de deler staat in locatie D.

Een flowchart van het geheel: (Zie volgende pag.)

Voor de verandering zullen we index register Y eens gebruiken als teller.

```
;
; File    : DIVIDE.MAC
; Purpose : Divide 16 -it number by 8 bit number

HEXOUT1 EQU     $C038           ; print A in hexadecimaal
PRINT1  EQU     $C03B           ; print tekst tot aan <NULL>
CRLF    EQU     $C02F           ; print <CR><LF>
;
; Data section
;
        ORG     $20             ; vertel AS waar de data staat

GETAL   FDB     0              ; 16-bits deeltal
D       FCC     0              ;  8 bits deler
Q       FCC     0              ;  8 bits quotient
B       EQU     GETAL          ;  hulp accu B (= lo part GETAL)
```



klaar (Q = uitkomst, A = rest)

```
;
; Code section
;
        ORG     $1000           ; vertel AS waar het programma begint

DIVIDE  LDA     GETAL+1         ; haal high part deeltal in A
        LDY     #8              ; gebruik Y als teller
        SEC                     ; we gaan aftrekken, dus CARRY aan!
        SBC     D               ; deeltal(high) - D
LOOP    PHP                     ; bewaar CARRY status eventjes op stack
        ROL     Q               ; schuif carry in Q
        ASL     B               ; en schuif deeltal (A <- B <- 0)
        ROLA
        PLP                     ; haal status register P weer van stack
        BCC     ADD             ; geen borrow?
        SBC     D               ; nee, doe volgende subtract
        JMP     NEXT            ; en kijk of we klaar zijn
ADD     ADC     D               ; te ver gegaan...
NEXT    DEY                     ; verlaag bit teller
        BNE     LOOP            ; en ga door!!
        BCS     LAST            ; laatste SBC gaf borrow?
        ADC     D               ; ja, te ver gegaan...
        CLC
LAST    ROL     Q               ; schuif laatste carry in Q
        STA     B               ; en bewaar rest in hulp accu B
        JSR     CRLF
        LDA     Q               ; laat resultaat zien op scherm
        JSR     HEXOUT1
        JSR     PRINT1
        FCC     '  rest: ',0
        LDA     B
        JSR     HEXOUT1
        JSR     CRLF
        RTS                     ; terug naar MON of DOS

        END     DIVIDE          ; thats all folks!
```

(Let wel, deze prachtige routine heb ik niet zelf bedacht, deze heb ik gepikt uit Rodnay Zaks (uitstekende) boek, 'Programming the 6502' dat voor zover ik weet ook in het Nederlands te krijgen is).

Deze routine 'verknalt' zoals je ziet het <low> gedeelte van het 16-bits deeltal, na 8x schuiven is dat byte helemaal met 0-en vol geschoven (ASL). Daarom kunnen we het net zo goed gebruiken om de rest van de deling (in A) op te slaan.

Assembleren en laden die hap:

```
$ AS DIVIDE.MAC
Pass 1...
Pass 2...
Last assembled address: 103F
Errors detected: 0
$ LO DIVIDE.BIN
```

Nu even proberen m.b.v. van de monitor:

```
MON> @ 0020
0020 00 00    --> GETAL = $1200  = 4608
0021 00 12
0022 00 40    -->     D =   $40  =   64

MON> e 1000
48 rest: 00   -->     Q =   $48  =   72 en dat klopt
```

Probeer zelf maar wat andere getallen, je zult zo nu en dan een rest krijgen. Bij floating point rekenen zouden dat de getallen achter de komma worden.

Zo, nu hebben we alle rekenkundige handelingen gehad. Maar de ALU, (Arithmetic & Logic Unit zoals de Fransen zeggen) kan, zoals de naam al aangeeft, niet alleen rekenkundig maar ook voor logische functies gebruikt worden.

Die logische functies zijn AND, OR en exclusive OR (EOR). Bij deze groep zou je ook de schuif operaties uit deel 3 kunnen indelen. Een buitenbeentje is de functie CMP (CoMPare) van de ALU. Deze functie kan gebruikt worden om twee getallen te vergelijken. Wat er in feite gebeurt is een SBC van die twee getallen zonder dat het resultaat ergens bewaard wordt, maar wat wel wordt bijgehouden is het gevolg van die SBC op het flag-register P. Dus na een CMP kun je aan de hand van de Z-flag kijken of de twee getallen gelijk zijn, of met de C-flag of de ene kleiner is dan de ander etc.etc.

```
TEST          LDA GETAL1      ; haal getal
              CMP GETAL2      ; en vergelijk
              BEQ GELIJK      ; ze zijn precies gelijk!
              ..
              JMP END
GELIJK  ..
              ..
END           RTS
```

De logische functies OR, AND en EOR zijn een van de meest gebruikte functies om een link te leggen met de hardware van je systeem. Veel van de hardware is immers gecontrolleerd d.m.v. van allerlei control-registers, status-registers en noem maar op. Al die registers stikken van de bitjes die allemaal weer hun specifieke betekenis hebben. Die betekenis kun je uit een databoek/datasheet van dat betreffende IC halen. Wil je nu een specifiek bitje uit een status register testen dan zul je dat ene bitje dus moeten isoleren van de andere bits. Welnu daarvoor zijn de logische functies ideaal. Voor diegenen die een beetje roestig zijn waar het logische of booleaanse functies betreft volgen hieronder de waarheids (en niets dan de waarheid) tabellen:

bit1	bit2	OR	AND	EOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

De slimste rakkers hebben het misschien al gezien, de AND functie is ideaal voor het isoleren van een bit zodat hij ter observatie kan worden opgenomen. Kijk maar:

```
bit1  bit2     AND
 0     x   -->  0
 1     x   -->  x
```

Waarbij x zowel '0' als '1' kan zijn. Zoals je ziet is de uitgang van de AND functie

altijd ´0´ als bit1 dat ook is. Is bit1 echter ´1´ dan wordt de uitgang volledig bepaald door ons onbekende bitje. Duidelijk?
Wellicht zal een voorbeeld in ASM U de schellen van den ogen doen vallen:

```
        ...
        LDA     STATUS                  ; lees status register
        AND     #%00100000              ; maskeer alles behalve bit 5
        BNE     BIT SET                 ; het bitje is ´1´
        JMP     CONT                    ; geen actie, bit is ´0´
BIT SET JSR     ALARM                   ; ALARM!!!!!!
CONT    ...
```

Zo zou een subroutine in de procesbesturing van een kerncentrale kunnen detecteren hoe de toestand van de kern op een bepaald moment is. Die toestand wordt bijgehouden in een STATUS register waarbij bit 5 bijvoorbeeld aangeeft dat er stront aan de knikker is, (MELTDOWN IN PROCESS, 10 SECONDS TO GO AND COUNTING 9,8,7.....) afijn, dat verhaal kent U.

Heel algemeen kun je stellen dat de AND functie gebruikt kan worden voor het ´resetten´ van bits, de OR functie voor het ´setten´ van bits. Maar die EOR functie, wat is dat voor rare druif? Welnu die is heel geschikt om een bitje als een blad aan de boom te doen omklappen (met een Frans woord ´toggle´). Kijk maar:

bit1	bit2		EOR
1	0	-->	1
1	1	-->	0

Zoals je ziet is de uitgang van de EOR functie steeds precies het tegenovergestelde van bit2 mits we bit1 op ´1´ houden. Maar wat te doen als we bepaalde bits niet willen beinvloeden? Welnu, die houden we gewoon ´naast´ een ´0´, kijk maar:

bit1	bit2		EOR
0	0	-->	0
0	1	-->	1

want dan wordt dat bit gewoon aan de uitgang doorgegeven. Deze functie kun je o.a. gebruiken om een puls op te wekken op een I/O poort zonder de andere lijnen van die poort te beinvloeden. Om dat te demonstreren heeft mijn nimmer rustende brein de volgende, simpele edoch geniale, creatie geproduceerd:

```
; File: EOR_DEMO.MAC
; Doel: demonstratie EOR functie d.m.v. knipperende letter op scherm

OUTCHAR EQU     $C023           ; DOS65 routine, output char. in A
CLS     EQU     12              ; <FormFeed>
VIDEO   EQU     $E800           ; adres in video RAM

        ORG     $1000

DEMO    LDA     #CLS            ; wissen die hap
        JSR     OUTCHAR
        LDA     #´A             ; we zetten nu een ´A´ d.m.v. een directe
        STA     VIDEO           ; write-actie in video RAM op het scherm
FLASH   LDA     #%10000000      ; toggle normal/inverse bit
        EOR     VIDEO
        STA     VIDEO           ; en schrijf dat terug
        JSR     DELAY           ; effe wachten, anders gaat het te snel..
        JMP     FLASH           ; en toggelen maar weer

DELAY   LDX     #0              ; we gaan gewoon 65535x niets doen!
DLOOP1  LDY     #0
DLOOP2  DEY
```

```
        BNE     DLOOP           ; deze gaat 256x keer rond
        DEX
        BNE     DLOOP           ; en deze oo !
        RTS

        END     DEMO
```

Dit programma demonstreert dat de EOR functie alleen die bits aantast waarvan het overeenkomende broertje in de Accu op '1' is gezet. In dit geval wordt gebruik gemaakt van de eigenschap dat de karaktergenerator van het DOS65 systeem het karakter 'inverse' op het beeldscherm weergeeft zodra het 8e bit 'geset' is. Welnu door daarvoor de EOR functie te gebruiken hoeven we verder niet te weten die andere bits allemaal zijn, het laat ons volkomen koud, het zal ons kortom worst zijn wat die andere bitjes allemaal in hun vrije tijd plegen te doen. Probeer maar eens door met MON de LDA #'A' (A9 41) instructie te wijzigen in bijv. LDA #'B' (A9 42) en je zult zien dat er dan ineens een knipperende 'B' op het scherm staat.

```
$ AS EOR_DEMO
Pass 1...
Pass 2...
Last assembled address: 1022
Errors detected: 0
$ LO EOR_DEMO.BIN
$ G 1000
^C
$ MON
(MON> D 1000 (eventueel))
MON> @ 1006
1006 41 42

MON> E 1000
```

Let wel, omdat het DOS65 scherm niet het gehele RAM geheugen weergeeft is het best mogelijk dat de geheugenlocatie die in dit programma gebruikt wordt net even niet zichtbaar is. Dat is eenvoudig op te lossen door het scherm een paar regels te laten 'scrollen' en het dan nog eens te proberen. Zoals je ziet, zit het programma bovendien in een oneindige lus (JMP FLASH), dus afbreken kan alleen met <CTRL-C>. Probeer zelf het programma zo te veranderen dat de letter precies 10x knippert en dan terugkeert naar DOS of MON. Gebruik daarvoor bijvoorbeeld het index X register, maar let wel op want dat wordt ook al gebruikt in de DELAY routine!! Probeer ook eens wat het effect is van een ander getal in de EOR functie (een leuke bij letters is bijv. $20). Enfin, zoals de Engelsen zeggen, tot het volgende deel maar weer!

Antoine.

HELP HELP HELP HELP HELP HELP HELP HELP

Enige tijd geleden werd ik benaderd door iemand met de vraag of de club hem aan (een kopie van) de originele beschrijving van KB9 Basic kon helpen. Ik heb toen binnen het bestuur geïnformeerd of er iemand was die deze beschrijving had en kreeg een bevestigend antwoord. Daarna heb ik de vraagsteller eerst lid gemaakt met de belofte de beschrijving op te sturen. Echter, wat bleek..... Degene die mij een beschrijving had toegezegd was hem kwijt.

Dus: **Wie heeft voor mij een beschrijving van KB9 Basic, zodat ik mijn belofte na kan komen?**

Graag contact opnemen met:

Gert van Opbroek, 01729-8636. (Kosten worden uiteraard vergoed).

LINEARISERING

Door Frank Vandekerkhove

Dit artikel kan gezien worden als een vervolg op het artikel "Spanningen meten met Dos65, uP Kenner nr. 60 pag. 33 van dezelfde auteur (red.).

Het meten met NTC's moet gepaard gaan met een correctie (zie "Temperatuur meten met NTC's" in Elektuur, februari 1988, blz. 52). De kromme weerstandskurve moet recht gemaakt worden. Dat geldt ook voor een Platina-Rhodium thermo-koppel: de afgegeven spanning is niet-lineair met de temperatuur.

Het onderstaande programma komt tussen de gemeten spanning (A/D converter) en de verwerking.

Op basis van de hoogste bits van de meting, die als een pointer gebruikt worden, wordt in de tabel gekeken wat de waarde van deze meting had moeten zijn. De vier laagste bits worden door linearisering gevormd op basis van het verschil tussen de vervangen waarde en de eerst volgende.

De tabel bevat voor een aantal metingen, waarvoor de laagste vier bits nul zijn, de vervangwaarde volgens de gegevens van de fabrikant.

Om het programma te demonstreren, wordt geen data van een A/D converter genomen, maar de waarde van een teller die steeds verhoogd wordt.

```
;
; File:         Ptlocnv.mac
; Program:      Ptlocnv
; Function:     Convert input data
;
; Frank Vandekerkhove
; Sint-Michielsstraat 4, B-2789 Verrebroek
;
; *** 1/065 2.02 subroutines ***
;
conver    equ  $FF02          convert 2 bytes hex into 3 bytes dec.
pridec    equ  $FF41          print dec numbers (deci 1, 2 & 3)
;
; *** io_var ***
;
deci1     equ  $E73A          variables for calculate decimal
deci2     equ  $E73B
deci3     equ  $E73C
;
; *** Dos 65 2.01 subroutines ***
;
out       equ  $C023          put a character
crlf      equ  $C02F          print <CR> and <LF>
spa       equ  $C032          print a space
hexout    equ  $C038          a hexout
print     equ  $C03B          print string after call

          org  $1000
;
conv      lda  #$00           reset counter
          sta  cnt
          sta  cnt+1
;
          jsr  crlf
;
```

```
1       ldx  cnt          print counter decimal
        lda  cnt+1
        jsr  conver
        lda  deci2        high dec out
        jsr  hexout
        lda  #'.          decimal point
        jsr  out
        lda  decl
        jsr  hexout       low dec out
        jsr  spa
        jsr  print
        fcc  'mV',0       print "mV"
;
        jsr  mspa         print 4 x <SP>
;
        lda  cnt          counter into input
        ldx  cnt+1
        sta  input
        stx  input+1
;
        jsr  Pt10cnv      convert input into PtRh/Pt 10% value
;
        ldx  input        print result of Pt10cnv
        lda  input+1
        jsr  conver       convert hex into dec
        lda  deci2
        jsr  hexout       print high byte
        lda  deci1
        jsr  hexout       print low byte
        jsr  spa
        jsr  print
        fcc  '°C',0       print "Celsius"
;
        jsr  crlf         print <CR> and <LF>
        inc  cnt          inc counter
        bne  2.f
        inc  cnt+1
2       lda  cnt+1        check for maximum
        cmp  #$05         counter < $0500
        bcc  1.b          repeat
        lda  cnt
        cmp  #$9b         counter < $059b (max. = 1400 C)
        bcc  1.b
        rts               back to Dos65
;
mspa    ldx  #$04
1       jsr  spa          print 4 x <SP>
        dex
        bne  1.b
;
; *** Pt10cnv: change input (b0-b10) into PtRh-Pt 10% value ***
;
Pt10cnv ldx  #$04         prepare loop
1       asl  input        xxxx xddd dddd dddd becomes
        rol  input+1      xddd dddd dddd 0000
        dex
        bne  1.b          shift 4 times
;
        asl  input+1      xddd dddd becomes dddd ddd0
        ldy  input+1      make index of it
        lda  conTAB,y     get high parameter
        sta  para+1
```

```
          iny                   increase index
          lda  conTAB,y         get low
          sta  para
          iny                   increase index
          lda  conTAB,y         get next parameter
          sta  parb+1
          iny
          lda  conTAB.y
          sta  parb
;
; para contains the new input according to b4-b10
;
; unused lowest nibble of the input residents as high nibble in input:
; therefore add, for b7, the diff. between para and parb /2 to para
;                for b6, the diff. between para and parb /4 to para
;                for b5, the diff. between para and parb /8 to para
;                for b4, the diff. between para and parb /16 to para
; get difference by subtract parb minus para
;
          sec
          lda  parb             subtract para from parb
          sbc  para
          sta  result
          lda  parb+1
          sbc  para+1
          sta  result+1
;
; *** lineair regression ***
;
          lda  #$80             first b7, then b6, b5 and b4
          sta  mask
          ldx  #$04             preset the loop
1         lsr  result+1         divide result by 2
          ror  result
          lda  mask             get mask for b7, b6, b5 or b4
          and  input
          beq  2.f
          clc                   add divided result to para
          lda  result
          adc  para
          sta  para
          lda  result+1
          adc  para+1
          sta  para+1
2         lsr  mask             make next mask for b6, b5 and b4
          dex
          bne  1.b
;
          lda  input
          and  #$f0
          beq  3.f
          clc                   add $001 to lineair regression
          lda  #$01
          adc  para
          sta  para
          lda  #$00
          adc  para+1
          sta  para+1
;
3         ldx  para             low in x, high in accu
          lda  para+1
          stx  input
```

```
          sta  input+1
          rts
;
conTAB    FCC  $00,$00,$00,$1C,$00,$36,$00,$4E
          FCC  $00,$64,$00,$7A,$00,$8E,$00,$A2
          FCC  $00,$B6,$00,$C8,$00,$DC,$00,$E6
          FCC  $01,$00,$01,$12,$01,$24,$01,$36
          FCC  $01,$46,$01,$58,$01,$69,$01,$7A
          FCC  $01,$8B,$01,$9C,$01,$AC,$01,$BD
          FCC  $01,$CE,$01,$DE,$01,$EE,$01,$FE
          FCC  $02,$0E,$02,$1E,$02,$2E,$02,$3E
          FCC  $02,$4E,$02,$5E,$02,$6D,$02,$7C
          FCC  $02,$8C,$02,$9C,$02,$AB,$02,$BA
          FCC  $02,$C9,$02,$D8,$02,$E7,$02,$F6
          FCC  $03,$05,$03,$0A,$03,$23,$03,$32
          FCC  $03,$40,$03,$4F,$03,$5E,$03,$6C
          FCC  $03,$7A,$03,$88,$03,$97,$03,$A4
          FCC  $03,$B2,$03,$C1,$03,$CF,$03,$DD
          FCC  $03,$EA,$03,$F8,$04,$06,$04,$14
          FCC  $04,$22,$04,$30,$04,$3D,$04,$4A
          FCC  $04,$58,$04,$65,$04,$72,$04,$80
          FCC  $04,$8E,$04,$9B,$04,$A8,$04,$B6
          FCC  $04,$C3,$04,$D0,$04,$DD,$04,$EA
          FCC  $04,$F8,$05,$05,$05,$12,$05,$20
          FCC  $05,$2D,$05,$3A,$05,$48,$05,$55
          FCC  $03,$62,$05,$70,$05,$7D
;
cnt       res  2
input     res  2
power     res  2
para      res  2
parb      res  2
result    res  2
mask      res  1
;
          end  conv
;
```

==========

<u>Te Koop</u>

- Uitgebreide Junior 6502 computer met:
- Elektuur's grote buskaart
- Elektuur VDU kaart
- Elektuur dynamische 64k RAM kaart
- Elektuur statische 64k RAM kaart (bevat 16 k)
- Zelfbouw I/O kaart (2 * 6522, pieper, etc.)
- Elektuur voeding met cassette-deck
- Elektuur toetsenbord
- DOS-65 disk controller kaart
- 2 floppy drives met ingebouwde voeding
- DOS-65 software en documentatie
- DATAPOINT terminal, te gebruiken als monitor

Alles eventueel met Junior boeken en 6502 bladen.

Prijs : Nog overeen te komen
Inlichtingen : Jeroen Oort
Telefoon : 020 - 866740 ('s avonds)

==========

Windows, een verzameling subroutines door Ernst Elderenbosch.

Op bijna elke moderne computer vind je wel een of andere vorm van windows (vensters). Op de Apple computers, Atari, Commodore, IBM pc en Acorn Archimedes zijn programma's in overvloed die er zonder windows heel anders uit zouden zien. Het is zo gemakkelijk, even vanuit een programma een hulpschermpje openen en later weer spoorloos laten verdwijnen. Het lijkt erop alsof hier heel grote programma's aan het werk zijn met tientallen zoniet honderden kilobytes aan geheugen en ingewikkelde routines maar het hoeft helemaal niet zo ingewikkeld te zijn. Sterker nog, het is al mogelijk om aardige resultaten te krijgen met een buffer van 2k en een stel subroutines die gemakkelijk in 512 bytes ram passen. Die subroutines zijn dan eenvoudig op te roepen vanuit een programma dat je zelf maakt, zodat je zonder vee moeite in je eigen assembler programma's windows kunt gebruiken. Drie listings zijn bijgevoegd:

windows.mac - de subroutines die bijvoorbeeld tijdens het booten worden geladen op $e400-e5ff en daar blijven.
windowlib.mac - een libraryfile die aan je programma te koppelen is en waar alle routines en variabelen in staan.
demo.mac - een demonstratieprogramma waarin je kunt zien hoe je de routines gemakkelijk kunt gebruiken vanuit een zelfgemaakt assembler programma.

Even een korte uitleg van de subroutines. Er zijn er vier: savewi, makewi, fillwi, remowi.

savewi — Maakt een kopie van het huidige scherm naar geheugengebied $a000-a7ff. Dit geheugengebied wordt veel gebruikt voor tijdelijke klussen zoals allerlei commando's die je vanuit het toetsenbord ingeeft. Deze kopie van het scherm kan later gebruikt worden om de oude scherminhoud weer te herstellen.

remowi — Dit herstellen van de oude scherminhoud gebeurt met de routine remowi (remove window). Deze routine maakt weer een kopie van $a000-a7ff en overschrijft daarmee de scherminhoud. Zolang niet opnieuw savewi wordt gebruikt en $a000-a7ff met rust wordt gelaten kun je altijd met remowi het oude scherm terugkrijgen.

makewi — Nadat je vanuit je programma de koordinaten van het te maken window hebt bekend gemaakt (door ze te zetten in $e40c-e40f), zorgt deze subroutine ervoor dat er een keurig kader wordt getekend en dat dit leeg wordt gemaakt.

fillwi — Om tekst in het window te krijgen, is het genoeg om het beginadres van de tekst in lokatie $e410-e411 te zetten en dan subroutine fillwi uit te voeren. Deze kopieert de tekst uit het geheugen naar het window, breekt de tekst af bij de rand van het window en gaat net zolang door tot er een $00 voorkomt in de tekst. Je moet er zelf voor zorgen dat de tekst niet te lang wordt anders loopt die onderuit het window.

Probeer eens hoe het werkt, demo.mac is een aardig voorbeeld van hoe je de routines kunt gebruiken en er zijn vast ook andere toepassingen te bedenken. Je hoeft niet perse een window te maken. Als je even je scherm wilt bewaren om een compleet ander scherm te voorschijn te

toveren zijn savewi en remowi vast ook heel nuttig. Mogelijkheden te over. De volgende keer een listing van de windowroutines die gebruik kunnen maken van de Z80 kaart om het scherm in op te bergen.

Veel succes - Ernst Elderenbosch.

```
                        ; file          windows.mac
                        ;
                        ; purpose       allow windowing on dos65 machines
                        ;               dos65 standard version
                        ;
                        ; author        E.R.Elderenbosch
                        ;
                        ; date          10 sept 1988
                        ;
                        ;
                        ;
          0000  pointr  equ     $0000           ; memory address pointer [2]
          0002  points  equ     $0002           ; memory address pointer save [2]
          0004  txtadr  equ     $0004           ; memory address text pointer [2]
                        ;
          C03B  prtext  equ     $c03b           ; print string till $0
                        ;
          A000  scratc  equ     $a000           ; scratch memory for screen copy
                        ;
          E400          org     $e400           ; free memory (2 pages)
                        ;
E400                    init
                        ;
E400 4C 19E4    savewi  jmp     copy1           ; copy screen area to scratch memory
E403 4C 3BE4    makewi  jmp     border          ; make window border & clear text area
E406 4C A6E4    fillwi  jmp     instxt          ; copy text to text area
E409 4C E4E4    remowi  jmp     copy2           ; restore original text from scratch m
ry
                        ;
E40C            crdxy1  res     2               ; x,y coordinates top left border
E40E            crdxy2  res     2               ; x,y coordinates bottom right border
E410            txtvec  res     2               ; memory address text to be placed in
dow
E412            adrxy1  res     2               ; screen memory address top left
E414            adrxy2  res     2               ; screen memory address bottom right
E416            linlen  res     2               ; hor & vert length of window
E418            lincnt  res     1               ; line count, temp variable
                        ;
E419 A9 E8      copy1   lda     #$e8
E41B 85 01              sta     pointr+1
E41D A9 A0              lda     #$a0
E41F 85 03              sta     points+1
E421 A9 00              lda     #0
E423 85 00              sta     pointr
E425 85 02              sta     points
E427 A0 00              ldy     #0
E429 B1 00      1       lda     [pointr],y
E42B 91 02              sta     [points],y
E42D C8                 iny
E42E D0 F9              bne     1.b
E430 E6 01              inc     pointr+1
E432 E6 03              inc     points+1
E434 A5 01              lda     pointr+1
E436 C9 F0              cmp     #$f0
E438 D0 EF              bne     1.b
E43A 60                 rts
```

```
                        ;
E43B 20 06E5    border  jsr     gtmadr          ; get memory address of coordinates
E43E 20 50E5            jsr     intadr          ; initialise address pointers
                        ;
E441 A0 00      top     ldy     #0              ; subroutine does check screen wrap.
E443 A2 00              ldx     #0
E445 A9 10              lda     #$10
E447 91 00              sta     [pointr],y
E449 20 5FE5    1       jsr     updptx          ; increment pointer
E44C E8                 inx
E44D EC 16E4            cpx     linlen
E450 F0 07              beq     2.f
E452 A9 18              lda     #$18
E454 91 00              sta     [pointr],y
E456 4C 49E4            jmp     1.b
E459 A9 16      2       lda     #$16
E45B 91 00              sta     [pointr],y
                        ;
E45D AD 17E4    sides   lda     linlen+1
E460 8D 18E4            sta     lincnt
E463 CE 18E4    1       dec     lincnt
E466 20 72E5            jsr     updpty
E469 A0 00              ldy     #0
E46B A2 00              ldx     #0
E46D AD 18E4            lda     lincnt
E470 F0 1B              beq     bottom
                        ;
E472 A9 19      2       lda     #$19
E474 91 00              sta     [pointr],y
E476 20 5FE5    3       jsr     updptx
E479 E8                 inx
E47A EC 16E4            cpx     linlen
E47D F0 07              beq     4.f
E47F A9 20              lda     #$20
E481 91 00              sta     [pointr],y
E483 4C 76E4            jmp     3.b
E486 A9 19      4       lda     #$19
E488 91 00              sta     [pointr],y
E48A 4C 63E4            jmp     1.b
                        ;
E48D A9 12      bottom  lda     #$12
E48F 91 00              sta     [pointr],y
E491 20 5FE5    1       jsr     updptx
E494 E8                 inx
E495 EC 16E4            cpx     linlen
E498 F0 07              beq     2.f
E49A A9 18              lda     #$18
E49C 91 00              sta     [pointr],y
E49E 4C 91E4            jmp     1.b
E4A1 A9 14      2       lda     #$14
E4A3 91 00              sta     [pointr],y
E4A5 60                 rts
                        ;
E4A6 20 06E5    instxt  jsr     gtmadr          ; get memory address of coordinates
E4A9 20 50E5            jsr     intadr          ; initialise address pointers
E4AC 20 72E5            jsr     updpty          ; first blank line
E4AF 20 5FE5            jsr     updptx          ; first char pos
E4B2 AD 10E4            lda     txtvec          ; get txt address
                ;       lda     #text&255
E4B5 85 04              sta     txtadr
E4B7 AD 11E4            lda     txtvec+1
                ;       lda     #text/256
```

```
E4BA 85 05              sta     txtadr+1
E4BC A2 01              ldx     #1
E4BE A0 00      1       ldy     #0
E4C0 B1 04              lda     [txtadr],y
E4C2 48                 pha
E4C3 20 8AE5            jsr     inctxt
E4C6 68                 pla
E4C7 F0 1A              beq     9.f             ; 0 = end of window text
E4C9 C9 0D              cmp     #$d
E4CB F0 0B              beq     2.f             ; next line within window
E4CD 91 00              sta     [pointr],y
E4CF 20 5FE5            jsr     updptx
E4D2 E8                 inx
E4D3 EC 16E4            cpx     linlen
E4D6 D0 E6              bne     1.b
E4D8 20 72E5    2       jsr     updpty
E4DB 20 5FE5            jsr     updptx
E4DE A2 01              ldx     #1
E4E0 4C BEE4            jmp     1.b
E4E3 60         9       rts
                ;
E4E4 A9 A0      copy2   lda     #$a0
E4E6 85 01              sta     pointr+1
E4E8 A9 E8              lda     #$e8
E4EA 85 03              sta     points+1
E4EC A9 00              lda     #0
E4EE 85 00              sta     pointr
E4F0 85 02              sta     points
E4F2 A0 00              ldy     #0
E4F4 B1 00      1       lda     [pointr],y
E4F6 91 02              sta     [points],y
E4F8 C8                 iny
E4F9 D0 F9              bne     1.b
E4FB E6 01              inc     pointr+1
E4FD E6 03              inc     points+1
E4FF A5 03              lda     points+1
E501 C9 F0              cmp     #$f0
E503 D0 EF              bne     1.b
E505 60                 rts
                ;
E506 38         gtmadr  sec
E507 AD 0EE4            lda     crdxy2          ; calculate hor length
E50A ED 0CE4            sbc     crdxy1          ; if carry cleared, => error x2 < x1
E50D 8D 16E4            sta     linlen
E510 EE 16E4            inc     linlen          ; take care of border
E513 AD 0FE4            lda     crdxy2+1
E516 ED 0DE4            sbc     crdxy1+1
E519 8D 17E4            sta     linlen+1
E51C EE 17E4            inc     linlen+1
                ;
E51F AE 0EE4    1       ldx     crdxy2          ; get end x and y
E522 AC 0FE4            ldy     crdxy2+1
E525 20 24F0            jsr     $f024           ; position cursor
E528 AD 24E7            lda     $e724           ; get memory address current position
E52B 8D 14E4            sta     adrxy2          ;  and save it
E52E AD 25E7            lda     $e724+1
E531 8D 15E4            sta     adrxy2+1
E534 20 27F0            jsr     $f027           ; restore cursor position
                ;
E537 AE 0CE4    2       ldx     crdxy1          ; get start x and y
E53A AC 0DE4            ldy     crdxy1+1
E53D 20 24F0            jsr     $f024           ; position cursor
```

```
E540 AD 24E7            lda     $e724           ; get memory address current position
E543 8D 12E4            sta     adrxy1          ;  and save it
E546 AD 25E7            lda     $e724+1
E549 8D 13E4            sta     adrxy1+1
E54C 20 27F0            jsr     $f027           ; restore cursor position ?
E54F 60                 rts
                ;
E550 AD 12E4    intadr  lda     adrxy1          ; initialise pointers
E553 85 00              sta     pointr
E555 85 02              sta     points
E557 AD 13E4            lda     adrxy1+1
E55A 85 01              sta     pointr+1
E55C 85 03              sta     points+1
E55E 60                 rts
                ;
E55F E6 00      updptx  inc     pointr          ; update pointer (increment by one and c
eck
E561 A5 00              lda     pointr          ;       for screen wrap)
E563 D0 0C              bne     1.f
E565 E6 01              inc     pointr+1
E567 A5 01              lda     pointr+1
E569 C9 F0              cmp     #$f0
E56B D0 04              bne     1.f
E56D A9 E8              lda     #$e8
E56F 85 01              sta     pointr+1
E571 60         1       rts
                ;
E572 18         updpty  clc                     ; set pointr to next line and check
E573 A5 02              lda     points          ;       for screen wrap
E575 69 50              adc     #80
E577 85 00              sta     pointr
E579 85 02              sta     points
E57B A5 03              lda     points+1
E57D 69 00              adc     #0
E57F C9 F0              cmp     #$f0
E581 D0 02              bne     1.f
E583 A9 E8              lda     #$e8
E585 85 01      1       sta     pointr+1
E587 85 03              sta     points+1
E589 60                 rts
                ;
E58A E6 04      inctxt  inc     txtadr
E58C D0 02              bne     9.f
E58E E6 05              inc     txtadr+1
E590 60         9       rts
                ;
          E400          end     init
                        label table

adrxy1    E412  adrxy2    E414  border    E43B  bottom    E48D  copy1     E419
copy2     E4E4  crdxy1    E40C  crdxy2    E40E  fillwi    E406  gtmadr    E506
inctxt    E58A  init      E400  instxt    E4A6  intadr    E550  lincnt    E418
linlen    E416  makewi    E403  pointr    0000  points    0002  prtext    C03B
remowi    E409  savewi    E400  scratc    A000  sides     E45D  top       E441
txtadr    0004  txtvec    E410  updptx    E55F  updpty    E572

Errors detected: 0
```

```
; file          windowlib.mac
;
; author        E.R.Elderenbosch
;
; date          sept 10, 1988
;
;
; reserved addresses in page zero
;
pointr  equ     $0000   ; memory address pointer [2]
points  equ     $0002   ; memory address pointer save [2]
txtadr  equ     $0004   ; memory address text pointer [2]
;
; addresses of resident window routines (used by user program)
;
savewi  equ     $e400   ; copy screen area to scratch memory
makewi  equ     $e403   ; make window border & clear text area
fillwi  equ     $e406   ; copy text to text area
remowi  equ     $e409   ; restore original text from scratch memory
;
; window variable block (must be filled by user program)
;
crdxy1  equ     $e40c   ; [2] x,y coordinates top left border
crdxy2  equ     $e40e   ; [2] x,y coordinates bottom right border
txtvec  equ     $e410   ; [2] memory address text to be placed in window
;
; internal variables window routines
;
adrxy1  equ     $e412   ; [2] screen memory address top left
adrxy2  equ     $e414   ; [2] screen memory address bottom right
linlen  equ     $e416   ; [2] hor & vert length of window
;
; file          demo.mac
;
; purpose       demonstration program for window routines
;
; author        E.R.Elderenbosch
;
; date          08 sept 1988
;
;
        lib     windowlib
;
getcha  equ     $c020
;
; one way to use the window routines is to use a macro like this one
; everytime you want to open a window. The parameters are:
; x1    leftmost x coordinate
; y1    upper y coordinate
; x2    rightmost x coordinate
; y2    lower y coordinate
; txt   2 byte address of text string
; Be sure to check boundaries and text length because the window
; routines assume that all parameters are checked
;
setvec  macro   x1,y1,x2,y2,txt
        lda     #x1
        sta     crdxy1
        lda     #y1
        sta     crdxy1+1
        lda     #x2
```

```
        sta     crdxy2
        lda     #y2
        sta     crdxy2+1
        lda     #txt&255
        sta     txtvec
        lda     #txt>>8
        sta     txtvec+1
        endm
;
; A second way to open a window is to give a starting point (top-left)
; and the width and heigth of your window and of course the text address
; In this case, use a macro like the setvec1 macro below
; x1    window width
; y1    window height
;
setvec1 macro   x1,y1,x1,y1,txt
        lda     #x1
        sta     crdxy1
        lda     #y1
        sta     crdxy1+1
        lda     #x1+x1
        sta     crdxy2
        lda     #y1+y1
        sta     crdxy2+1
        lda     #txt&255
        sta     txtvec
        lda     #txt>>8
        sta     txtvec+1
        endm
;
        org     $2000           ; start demo program
;
start
;
        jsr     savewi          ; saves the current screen
;
        setvec  45,10,70,22,text        ; use absolute values this time
;
        jsr     makewi          ; make a window and clear it
        jsr     fillwi          ; copy text to window
        jsr     getcha          ; any key to continue
        jsr     remowi          ; restore old screen
;
        setvec  5,4,32,14,text2
;
        jsr     makewi
        jsr     fillwi
        jsr     getcha
        jsr     remowi
;
        setvec1 17,14,53,6,text3        ; use width and height now
;
        jsr     makewi
        jsr     fillwi
        jsr     getcha
;                                       don't clear the window
        setvec1 22,12,32,4,text4
;                                       ; overlay a second window
        jsr     makewi
        jsr     fillwi
        jsr     getcha
        jsr     remowi                  ; restore original screen
```

```
;
        setvec1 17,14,53,6,text3        ; restore last window
;
        jsr     makewi
        jsr     fillwi
        jsr     getcha
        jsr     remowi
;
;
        setvec1 70,2.2,18,0
        jsr     makewi
;
        setvec1 40,3.33,5,text5
        jsr     makewi
        jsr     fillwi
        jsr     getcha
        jsr     remowi
;
        setvec1 3,3,69,16,0
        jsr     makewi
        setvec1 5,4,65,14,text6
        jsr     makewi
        jsr     fillwi
        jsr     getcha
        jsr     remowi
;
        rts
;
; The text must consist of normal printable characters. It is copied
; directly into screen ram. Use $d (cr) to go to the next line within
; the window and $0 (null) as end of text.
;
text    fcc     ' This is a window demo',$d
        fcc     ' ====================',$d,$d
        fcc     ' could be used as help',$d
        fcc     ' window in a program',$d
        fcc     ' or as a menu',0
;
text2   fcc     ' A window can be opened',$d
        fcc     ' by using routines savewi',$d
        fcc     ' and makewi, text can be',$d
        fcc     ' inserted with fillwi, the',$d
        fcc     ' window can be cleared by',$d
        fcc     ' makewi and removed by the',$d
        fcc     ' remowi routine.',$d
        fcc     ' window coordinates and',$d
        fcc     ' text address must be set',$d
        fcc     ' in table $e40c-$e411',0
;
text3   fcc     ' The routines are loaded in a memory area at $e400',$d
        fcc     ' by the login.com file and use 512 bytes of memory',$d
        fcc     ' Take care not to wipe them by using a program like',$d
        fcc     ' exetime that will be loaded at the same location',$d
        fcc     ' The original screen contents are stored in scratch',$d
        fcc     ' memory at $a000-a7ff',0
;
text4   fcc     ' Text windows can also be',$d
        fcc     ' overlayed. The underlaying',$d
        fcc     ' window will not be deleted',$d
        fcc     ' if you do not want it deleted',0
;
```

Windows onder DOS65 met een Z80 kaart

In het vorige artikel heb ik de window subroutines beschreven voor een standaard dos65 machine. Ik beloofde toen een set subroutines die gebruik kunnen maken van de Z80 kaart van elektuur om een scherm in op te bergen. Deze subroutines maken gebruik van exact dezelfde adressen en parameters als de routines van de vorige keer, dus je kunt eenvoudigweg deze routines laden in $e400-$e5ff ipv. de andere routines zonder wijzigingen aan te moeten brengen in je programma's. Er moet een kleine kanttekening bij geplaatst worden: De elektuur Z80 kaart 'zit' normaal op

$e300-$e303. Dit vond ik niet praktisch want het waren de enige adressen in pagina $exx die gebruikt werden voor I/O. Daarom heb ik in mijn systeem de Z80 kaart geplaatst op $e158-$1e5b, vier adressen die goed aansluiten bij de andere I/O adressen. Als je de kaart nu op een ander adres hebt ingesteld hoef je alleen de waarde 'z80bas' (nu $e258) aan te passen in de source. Er zijn twee listings, de source 'windowz.mac' (met de z van Z80), en de file graphlib.mac, waarin alle graphics characters vermeld staan met hun resp. hexcodes. Zo kun je de routines gemakkelijker aanpassen als je een andere character generator gebruikt.

succes,

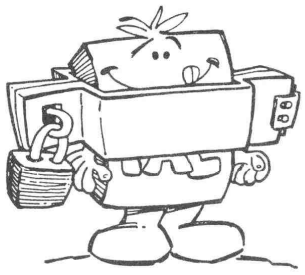
Ernst Elderenbosch

```
; file                    graphlib.mac
;
; purpose                 graphic character definitions
; author                  Ernst Elderenbosch
; system                  dos65
; date                              18 september 1988
;
; io65 standard graphics
;
vb          equ         $18         ; vertical bar
hb          equ         $19         ; horizontal bar
tl          equ         $10         ; top left corner
tr          equ         $16         ; top right corner
bl          equ         $12         ; bottom left corner
br          equ         $14         ; bottom right corner
tm          equ         $17         ; top bar with middle connection
lm          equ         $11         ; left bar      "
bm          equ         $13         ; bottom bar    "
rm          equ         $15         ; right bar     "
vh          equ         $1a         ; crossed vertical and horizontal bar
;
; file          windowz.mac
;
; purpose       allow windowing on dos65 machines
;               using a z80 second processor card at $e158-e15b
; author        E.R.Elderenbosch
; date          08 sept 1988
;
;
              lib         graphlib
; file                     graphlib.mac
;
; purpose       graphic character definitions
; author        Ernst Elderenbosch
; system        dos65
; date          18 sept 1988
;
```

```
; io65 standard graphics
;
vb          equ         $18             ; vertical bar
hb          equ         $19             ; horizontal bar
tl          equ         $10             ; top left corner
tr          equ         $16             ; top right corner
bl          equ         $12             ; bottom left corner
br          equ         $14             ; bottom right corner
tm          equ         $17             ; top bar with middle connection
lm          equ         $11             ; left bar  „
bm          equ         $13             ; bottom bar „
rm          equ         $15             ; right bar  „
vh          equ         $1a             ; crossed vertical and horizontal bar
;
pointr      equ         $0000           ; memory address pointer [2]
points      equ         $0002           ; memory address pointer save [2]
txtadr      equ         $0004           ; memory address text pointer [2]
;
z80bas      equ         $e158           ; z80 base address
;
prtext      equ         $c03b           ; print string till $0
;
            org         $e400           ; free memory (2 pages)
;
                        init
;
savewi      jmp         copy1           ; copy screen area to scratch memory
makewi      jmp         border          ; make window border & clear text area
fillwi      jmp         instxt          ; copy text to text area
remowi      jmp         copy2           ; restore original text from scratch memory
;
crdxy1      res         2               ; x,y coordinates top left border
crdxy2      res         2               ; x,y coordinates bottom right border
txtvec      res         2               ; memory address text to be placed in window
adrxy1      res         2               ; screen memory address top left
adrxy2      res         2               ; screen memory address bottom right
linlen      res         2               ; hor & vert length of window
lincnt      res         1               ; line count, temp variable
;
copy1       jsr         iniz80          ; init z80 interface pia
            lda         #1              ; copy to z80 command
            jsr         outz80
            lda         #$10            ; dest address $1000
            jsr         outz80
            lda         #0
            jsr         outz80
            lda         #8              ; copy 2k bytes = $0800
            jsr         outz80
            lda         #0
            jsr         outz80
;
1           lda         #0
            sta         pointr
            lda         #$c8
            sta         pointr + 1
            lda         #$f0            ; end of screen memory area
            sta         points + 1
2           ldy         #0
            lda         [pointr],y
```

```
                jsr     outz80
                inc     pointr
                bne     2.b
                inc     pointr + 1
                lda     pointr + 1
                cmp     points + 1
                bne     2.b
                rts
;
border          jsr     gtmadr          ; get memory address of coordinates
                jsr     intadr          ; initialise address pointers
;
top             ldy     #0              ; subroutine does check screen wrap.
                ldx     #0
                lda     #tl
                sta     [pointr],y
1               jsr     updptx          ; increment pointer
                inx
                cpx     linlen
                beq     2.f
                lda     #hb
                sta     [pointr],y
                bra     1.b
2               lda     #tr
                sta     [pointr],y
;
sides           lda     linlen + 1
                sta     lincnt
1               dec     lincnt
                jsr     updpty
                ldy     #0
                ldx     #0
                lda     lincnt
                beq     bottom
;
2               lda     #vb
                sta     [pointr],y
3               jsr     updptx
                inx
                cpx     linlen
                beq     4.f
                lda     #$20
                sta     [pointr],y
                bra     3.b
4               lda     #vb
                sta     [pointr],y
                bra     1.b
;
bottom          lda     #bl
                sta     [pointr],y
1               jsr     updptx
                inx
                cpx     linlen
                beq     2.f
                lda     #hb
                sta     [pointr],y
                bra     1.b
2               lda     #br
                sta     [pointr],y
```

Secure Data

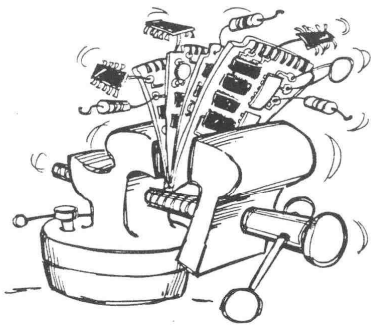


```
        rts
;
instxt  jsr     gtmadr          ; get memory address of coordinates
        jsr     intadr          ; initialise address pointers
        jsr     updpty          ; first blank line
        jsr     updptx          ; first char pos
        lda     txtvec          ; get txt address
        sta     txtadr
        lda     txtvec + 1
        sta     txtadr + 1
        ldx     #1
1       ldy     #0
        lda     [txtadr],y
        pha
        jsr     inctxt
        pla
        beq     9.f             ; 0 = end of window text
        cmp     #$d
        beq     2.f             ; next line within window
        sta     [pointr],y
        jsr     updptx
        inx
        cpx     linlen
        bne     1.b
2       jsr     updpty
        jsr     updptx
        ldx     #1
        bra     1.b
9       rts
;
copy2   lda     #3              ; copy from z80 command
        jsr     outz80
        lda     #$10            ; org address $1000
        jsr     outz80
        lda     #0
        jsr     outz80
        lda     #8              ; copy 2k bytes = $0800
        jsr     outz80
        lda     #0
        jsr     outz80
;
1       lda     #0
        sta     pointr
        lda     #$e8
        sta     pointr + 1
        lda     #$f0            ; end of screen memory area
        sta     points + 1
2       jsr     getz80
        ldy     #0
        sta     [pointr],y
        inc     pointr
        lda     pointr
        bne     2.b
        inc     pointr + 1
        lda     pointr + 1
        cmp     points + 1
        bne     2.b
        lda     #$ff
        jsr     outz80
```

```
        rts
;
gtmadr  sec
        lda     crdxy2          ; calculate hor length
        sbc     crdxy1          ; if carry cleared, = error x2 x1
        sta     linlen
        inc     linlen          ; take care of border
        lda     crdxy2+1
        sbc     crdxy1+1
        sta     linlen+1
        inc     linlen+1
;
1       ldx     crdxy2          ; get end x and y
        ldy     crdxy2+1
        jsr     $f024           ; position cursor
        lda     $e724           ; get memory address current position
        sta     adrxy2          ;  and save it
        lda     $e724+1
        sta     adrxy2+1
        jsr     $f027           ; restore cursor position
;
2       ldx     crdxy1          ; get start x and y
        ldy     crdxy1+1
        jsr     $f024           ; position cursor
        lda     $e724           ; get memory address current position
        sta     adrxy1          ;  and save it
        lda     $e724+1
        sta     adrxy1+1
        jsr     $f027           ; restore cursor position ?
        rts
;
intadr  lda     adrxy1          ; initialise pointers
        sta     pointr
        sta     points
        lda     adrxy1+1
        sta     pointr+1
        sta     points+1
        rts
;
updptx  inc     pointr          ; update pointer (increment by one and check
        lda     pointr          ;                 for screen wrap)
        bne     1.f
        inc     pointr+1
        lda     pointr+1
        cmp     #$f0
        bne     1.f
        lda     #$e8
        sta     pointr+1
1       rts
;
updpty  clc                     ; set pointr to next line and check
        lda     points          ;                 for screen wrap
        adc     #80
        sta     pointr
        sta     points
        lda     points+1
        adc     #0
        cmp     #$f0
        bne     1.f
```

```
                lda     #$e8
1               sta     pointr+1
                sta     points+1
                rts
;
inctxt          inc     txtadr
                bne     9.f
                inc     txtadr+1
9               rts
;
iniz80          lda     #40
                ldy     #44
                ldx     #0              ; set port directions
                sta     z80bas+1
                stx     z80bas
                sty     z80bas+1
                dex
                sta     z80bas+3
                stx     z80bas+2
                sty     z80bas+3
                ldx     #20
1               lda     z80bas          ; purge z80 fifo buffer
                dex
                bne     1.b
                rts
;
outz80          pha
1               lda     z80bas+3
                bpl     1.b             ; wait until last byte accepted
                lda     z80bas+2        ; reset pbc flag
                pla
                sta     z80bas+2        ; send byte to z80
                rts
;
getz80
1               lda     z80bas+1
                bpl     1.b
                lda     z80bas          ; get data byte
                rts
;
                end     init
```

Space Efficiency



Virtual Eprom Disk voor DOS65

Waarom na de virtual disk nu nog een epromdisk zullen velen zeggen. Toen ik met DOS65 startte, beschikte ik slechts over 1 diskdrive. Om een beetje opti-maler met DOS65 te kunnen werken, bouwde ik een virtual disk van 512kB (niet volledig identiek aan het schema van de club). Bij het opstarten werden dan de meest gebruikte utilities in een subdirectory hiervan gecopieerd, die dan als systeemdisk fungeerde. Een andere subdirectory werd gereserveerd als workdrive en de root als userdrive. Toen onstond het idee om de informatie van de systeemdisk in EPROMs te pro-grammeren en deze dan ergens in de beschikbare ruimte, voorzien voor de virtual disk, te adresseren. De systeemdisk zou dan direkt beschikbaar zijn na het booten. De beschikbare ruimte van 1MB zou ik toch nooit volledig gebruiken als ramdisk (reeds met een virtual userdrive van 256k kan men al heel wat uit-richten). Uiteindelijk heb ik gekozen voor virtual epromdisk (v.e.d.) van 256kB (4 * 27512), hierin kan men heel wat utilities en eventuele programmeer-talen plaatsen. Hieronder een voorbeeld wat de inhoud van een v.e.d. zou kunnen zijn. Deze v.e.d. is voorzien voor 4 EPROMs, er is echter nog iets meer dan 1 EPROM leeg (ongeveer 68k).

Een kleine toelichting bij het schema. De ICs 1,2 en 9 zijn de buffers voor de adres- en datalijnen. De poorten IC10C,10B,10F en IC11 dekoderen het adres FFXD (enkel aktief bij het schrijven) voor de latch IC7, die de hoogste adreslijnen A12'-A19' aanmaakt. De geheugenruimte DXXX, waarin de virtual disk aktief is, wordt gedekodeerd door de poorten IC10A en IC13B. De v.e.d. wordt door poort IC12A in het hoogste blok van 256k geadresseerd. Een beveiliging tegen het schrijven naar de eproms, wordt gerealiseerd met de poorten IC10E en IC12B. Hieronder de kleine software-aanpassing van DOS65, die noodzakelijk is om de v.e.d. door DOS65 te laten herkennen. Er wordt gebruik gemaakt van bepaalde zaken die reeds in DOS65 voorzien waren voor een winchester disk.

In principe is de v.e.d. enkel te gebruiken als er een virtual disk in het systeem zit. De RAM in het gebied D000..DFFF moet namelijk uitgeschakeld worden op het moment dat de v.e.d. wordt geadresseerd. Door middel van een eenvoudige hardware modificatie, kan de v.e.d. toch werken zonder virtual disk in het systeem. Deze wijziging houdt wel in dat er een verbinding loopt tussen de v.e.d. en de statische RAM-kaart. Van zodra er iets uit de v.e.d. ge-lezen wordt, gaat pin 6 van IC13A naar laag. Dit signaal wordt doorverbonden met pin 11 van N9 op de statische RAM-kaart, waardoor deze volledig geblok-keerd wordt. Op de statische RAM-kaart moet natuurlijk de draadbrug P verwijderd worden. Ter voorkoming van buskonflikten, als de verbinding tussen de twee kaarten ontbreekt, wordt pin 11 van N9 op de RAMkaart best met een weerstand van 2k2 ohm met de massa verbonden. Ontbreekt de verbinding, dan werkt heel het systeem niet, maar beter dit dan enkele gesneuvelde databusbuf-fers. Voor de dynamische RAM-kaart bestaat er een ongeveer gelijkaardige oplossing. Het aanmaken van de EPROMs voor de v.e.d. kan eigenlijk het eenvoudigst als men beschikt over een virtual disk en een epromprogrammer. De virtual disk wordt geladen met alle programma's, die men in de v.e.d. wenst. Het verkegen bitpatroon van de virtual disk wordt in binary file's geladen, waarmee de EPROMs voor de v.e.d. geprogrammeerd kunnen worden. Indien men zelf geen EPROMprogrammer heeft, kan men met behulp van de volgende commandfile mbin &1 de binary files aanmaken. De diskette met de binary files kan je dan

```
SDIR 3:
28-May-89 21:42 Disk: Virtual Eprom
>D          ECHO        PAUSE       SETUP       csys.mac    APPEND
EDDHLP.EDD  PLIST       TIME        cuser.mac   AS          EDITOR
PRINT       TYPE        cvar.mac    ASN         EDTAB       RENAME
UNEXPAND    delC        BOOTLINK    EP          RS232       VFORMATX
direct      BOOTved     EXPAND      RS232.DAT   asdc        instalC
CAT         FORMAT      SDIR        asmc        lmacro      CLEAR
HELP        SEROFF      cam         login.com   COPY        LABEL
SERON       cc          mccom       CREATE      LIST        SETBEGIN
cc.bin      smacro      DDOCTOR     MAP         SETCURS     ccas
DDOCTOR.DFT MEMFILL     SETDRIVES   cmac.mac    DELETE      MEMMOVE
SETMODE     cmat.mac    DPTIME      MONITOR     SETPROMPT   cstr.mac
DUMP        OLOAD       SETRTC      csub.mac    sprint
71 files, 268 blocks free.
```

Fig. 1: Directory van EPROM- (virtual) disk

aan iemand bezorgen, die de EPROMs voor je wil programmeren.

```
; filename    :        mbin &1
; author      :        De Cock Luc
; date        :        6 january 1989
CL
ECHO This commandfile makes and saves the
        binary files needed
ECHO to program the eproms for the virtual
        eprom disk.
ECHO
ECHO Warning! Files in the virtual disk will be
        erased.
ECHO The virtual disk will be wipped clean (filled
        with $FF) and formatted.
ECHO
ECHO Everytime there is waited on your keystroke
        you can leave
ECHO this commandfile with Ctrl-C (all made
        binary files will be complete).
ECHO PAUSE Strike any key to continue
;FILLFF
VFORMAT
ECHO
nmtrck&1
copysort
mcbin1
PAUSE Make .bin for eprom 2? Yes : Strike a key
mcbin2
PAUSE Make .bin for eprom 3? Yes : Strike a key
mcbin3
PAUSE Make .bin for eprom 4? Yes : Strike a key
mcbin4
ECHO This completes mbin.
```

Zorg ervoor dat de programma's mbin, mcbin1-4, nmtrck1-4, restvirt, NAME en PAUSE op je systeemdisk staan. Het argument &1 van mbin is een getal ($1 \leq m \leq 4$) dat aangeeft hoeveel EPROMs er op de v.e.d-kaart zullen komen te staan. Dit argument bepaalt welke nmtrck* commandfile uitgevoerd zal worden en aldus hoeveel tracks en vrije blocks de virtual disk nog zal bevatten na wijziging van zijn bitmap. PAUSE is een variatie op ECHO met een ingebouwde lus, die wacht op een toetsaanslag. Het programma FILLFF is voor de werking van de v.e.d. niet nodig. Enkel wanneer de v.e.d. bekeken wordt met DDOCTOR, geeft het een netter uitzicht (in de lege sectors staat overal $FF i.p.v. willekeurige data). Alle gewenste programma's worden door de commandfile copysort naar de virtual disk gecopieerd.

```
; file : nmtrk1
; function : places name, number of tracks and ad-
justs bitmap ;
of drive 2: for 1 EPROM 27512.
;select track 0 at 2XXX
MEMFILL FFF2,FFF2,01
;place name
LOAD NAME
;number of tracks : $10
MEMFILL 2021,2021,10
;adjust bitmap
MEMFILL 2070,20FF,00
;reselect original memory
MEMFILL FFF2,FFF2,0D

; file : nmtrk2
; for 2 EPROMs
MEMF FFF2,FFF2,01
LOAD NAME
LC 2021,2021,20
LC 2080,20FF,00
LC FFF2,FFF2,0D

; file nmtrk3
; for 3 EPROMs
MEMF FFF2,FFF2,01
LOAD NAME
LC 2021,2021,30
LC 2090,20FF,00
LC FFF2,FFF2,0D

; file : nmtrk4
; for 4 EPROMs
MEMF FFF2,FFF2,01
LOAD NAME
LC 2021,2021,40
LC 20A0,20FF,00
LC FFF2,FFF2,0D

; file : name.mac
; places v.e.d. name in the drive 2:
;
        org     $2040
;
name  fcc     'Virtual Eprom'      ;place name
;
        end     name
```

een voorbeeld van de file copysort :

```
; file : copysort
COPY S:APPEND 2:
LC S:AS 2:
LC S:ASN 2:
LC S:CAT 2:
LC S:COPY 2:
LC S:CREATE 2:
LC S:DDOCTOR 2:
;enzovoort ...
```

De commandfiles mcbin* zorgen ervoor dat telkens 8 opeenvolgende tracks van de virtual disk worden geadresseerd in het geheugengebied 1000 8FFF, waarna ze op diskette worden gesaved. Vooraleer de commandfile te verlaten, wordt opnieuw het normale geheugen geselecteerd door de commandfile restvirt.

```
; file:       mcbin1
; function:   commandfile to make the the binary
              files for EPROM 1 of the v.e.d.
ECHO
ECHO EPROM 1
MEMFILL FFF1,FFF1,01    ;track 0 at 1000 1FFF
LC FFF2,FFF2,00         ;track 1 at 2XXX
LC FFF3,FFF3,1F         ;track 2 at 3XXX
LC FFF4,FFF4,1E
LC FFF5,FFF5,1D
LC FFF6,FFF6,1C
LC FFF7,FFF7,1B
LC FFF8,FFF8,1A
ECHO SAVE -b ep1a.bin 1000,8FFF
SA -b ep1a.bin 1000,8FFF
MEMFILL FFF1,FFF1,19
LC FFF2,FFF2,18
LC FFF3,FFF3,17
LC FFF4,FFF4,16
LC FFF5,FFF5,15
LC FFF6,FFF6,14
LC FFF7,FFF7,13
LC FFF8,FFF8,12
ECHO SAVE -b ep1b.bin 1000,8FFF
SA -b ep1b.bin 1000,8FFF
restvirt
```

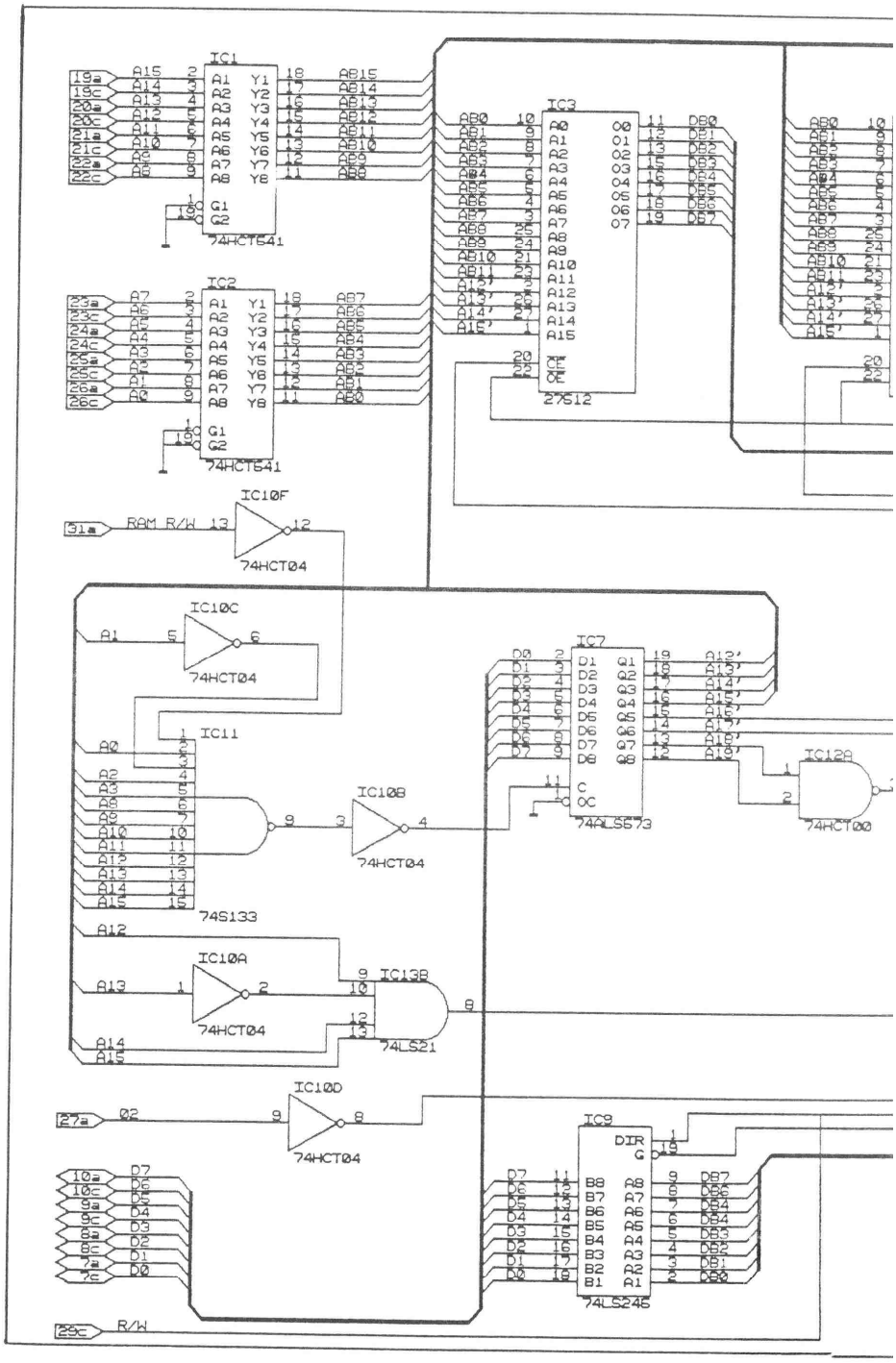
De commandfiles mcbin2, 3 en 4 voor de andere EPROMs zijn identiek dan opzet. Enkel de waarden, die op de adressen FFFX worden geplaatst, verschillen alsook de namen van de verschillende binary files.

```
; file:       restvirt
; function:   reselect the original systemmemory
;             1000-8FFF
MEMFILL FFF1,FFF1,0E
MEMFILL FFF2,FFF2,0D
MEMFILL FFF3,FFF3,0C
MEMFILL FFF4,FFF4,0B
MEMFILL FFF5,FFF5,0A
MEMFILL FFF6,FFF6,09
MEMFILL FFF7,FFF7,08
MEMFILL FFF8,FFF8,07
```

Heeft men een EPROMprogrammer, dan hoeven de bin.files niet op diskette geplaatst te worden. In de commandfile mcbin* komt dan telkens een aanroep van EP in de plaats van SAVE ... . Let op dat je telkens bij de 2de binary file van elke EPROM het EPROMstartadres en EPROMeindadres in EP wijzigd in respectievelijk 8000 en FFFF. Hetgeen deze command files doen, zou men makelijk door 1 programma kunnen laten uitvoeren, maar deze oplossing vroeg toen minder ontwerptijd en het werkt ook.

```
AS DRIVERSVE
makecom DRIVERSVE
LOAD DRIVERSVE
ASN S=3:
```

Indien de nodige hardware en geprogrammeerde EPROMs aanwezig zijn, staat de epromdisk ter uwer beschiking. Opgelet: doordat bepaalde adressen en routines rechtstreeks in DOS65 worden aangesproken, werkt deze aanpassing enkel bij de originele versie 2.01! Om geen nieuwe versie van BOOT in omloop te brengen, is het aan te bevelen LOAD DRIVERSVE en 3:ASN S=3: in LOGIN.COM op te nemen. Deze v.e.d. heeft natuurlijk een groot nadeel dat men telkens de EPROMs opnieuw moet programmeren, wanneer men een nieuwe utilitie in de systeemdisk wil bijplaatsen. Vandaar mischien de vraag: "Waarom geen ram disk, uitgevoerd met statische RAMs en voorzien van battery backup. Akkoord dat is waarschijnlijk de mooiste oplossing, maar EPROMs bieden nog steeds meer bytes op dezelfde ruimte tegen een gunstigere prijs en zodikwijls zal dat toch niet gebeuren, dat men iets moet wijzigen aan de systeemdisk. Iemand bezorgde mij het idee om als de foutmelding "3:FILE not found" komt, eventueel te gaan kijken of deze FILE niet op drive 0: staat. Maar voorlopig ben ik er nog niet in geslaagd om met deze tip iets te doen. Zowaar zou ik nog het voornaamste argument voor het bouwen van een virtual epromdisk vergeten, namelijk het booten vanuit de de virtual epromdisk. Hoe dat kan, leest U mischien in een volgend artikel van mij.


IC1
74HCT641
IC2
74HCT641
IC3
27512
IC10F
74HCT04
IC10C
74HCT04
IC11
74S133
IC10B
74HCT04
IC7
74ALS573
IC12B
74HCT00
IC10A
74HCT04
IC13B
74LS21
IC10D
74HCT04
IC9
74LS245
RAM R/W
R/W

Programeren in Assembler voor DOS-65 (deel 5)

Na een korte periode van afwezigheid (mijn fans zullen tevergeefs gezocht hebben naar deel 5 in 'de μPKenner' nr. 64) heeft mijn psychiater weer het groene licht gegeven voor wederom een literair hoogstandje, kortom lieve lezers en vooral lezeressen, deel 5! Maar eerst...

Voordat U kunt gaan genotteren, even een korte terugblik op deel 4. Bij mijn exemplaar van nr.63 was in de tekst op pagina 12, de assembler text niet volledig. Er stond tweemaal BNE DLOOP, voor de goede orde dit moet zijn:

```
        BNE  DLOOP2
        DEX
        BNE  DLOOP1
        ...
```

Ook had ik gevraagd om het programma zo te wijzigen dat de letter precies 10x zou knipperen en dan terugkeert naar MON of DOS. Welnu, het zal een ieder duidelijk zijn dat we voor deze truuk een teller nodig hebben. Verder zal die JMP FLASH eruit moeten, want deze, zoals de Fransen zeggen, 'unconditional jump' springt immers ALTIJD. Wat wij nodig hebben is een 'conditional jump', kortom een spring-in-het-veld die verder kijkt dan z'n binaire neus lang is. Maar 'quelle malheur', zoals de Engelsen zeggen, zo'n conditonele jump heeft de 6502 niet aan boord. Gelukkig heeft ie wel conditionele branches, een aantal hebben we al gezien (BCC, BCS, BNE en BEQ). Die branches hebben als nadeel (zie deel 2) dat ze een beperkt bereik hebben van -128 tot +127 geheugenlocaties. Maar voor het EOR_DEMO programma is dat ruimschoots voldoende. Zo'n conditionele branch verlengen tot een conditionele jump is overigens een piece of cake, zoals de Fransen zo treffend weten te zeggen, kijk maar:

```
        ...
        BNE  NOGO
        JMP  GO
NOGO    ...
```

En zie daar, een conditionele jump en wel van het type JEQ (Jump if EQual). Maar goed terug naar ons FLASH probleem, wat we moeten maken is een conditionele jump die alleen springt als de loop teller nog niet klaar is met zijn 10 rondjes knipperen. Zoals ik al had gewaarschuwd kunnen we de X of Y registers niet gebruiken omdat die al in gebruik zijn in de DELAY subroutine. Natuurlijk kun je X of Y eerst bewaren in een geheugenlocatie:

```
        ...             ; begin van EOR_
                            DEMO (zie deel 4)
        LDX  #20        ; 10x knipperen =
                                20 x EOR!!
FLASH   LDA  ...        ; zie deel 4
        ...
        STX  XSAVE      ; bewaar loop teller X
        JSR  DELAY      ; effe wachten, anders
                            gaat het te snel..
        LDX  XSAVE      ; loop teller ophalen
        DEX             ; teller = teller - 1
        BNE  FLASH      ; als NIET nul, doorgaan!
        RTS             ; anders terug naar MON
                            of DOS
XSAVE   FCC 0           ; geheugenlocatie om X
                            tijdelijk te bewaren
```

Natuurlijk zijn er legio andere mogelijkheden, bijvoorbeeld een geheugen (b.v. op de zero-page) locatie gebruiken als TELLER, dit heeft als voordeel dat we X niet hoeven te 'saven'. Gebruik dan LDA #20, STA TELLER als initia- lisatie en DEC TELLER voor de conditionele branch. Ook kun je de teller laten optellen vanaf 0 en dan CMP, CPX of CPY (resp. bij gebruik van een geheugen locatie, index register X of index register Y als teller) gebruiken om te kijken of je al klaar bent. Dit laatste heeft als nadeel dat het programma twee bytes langer wordt, door aftellen te gebruiken maak je namelijk handig gebruik van het feit dat de Z-flag automatisch gezet wordt als je teller de 0 bereikt. Gebruik je optellen, dan heb je de CMP instructie nodig om de Z-flag te zetten. Vraagje, waarom is het (dus) onzin om CMP #0 te gebruiken?

De adresseer mogelijkheden

In dit deel gaan we (eindelijk) eens kijken naar de adresseer mogelijkheden van ons wonderdoosje. Vooral in het begin duizelt het de machinetaal programmeur van de vele mogelijkheden op dat gebied en welke kun je nu het beste gebruiken en waarom? Allereerst maar een opsommimg, de 6502 heeft totaal 13 adresseer- mogelijkheden:

1. Implied
2. Accumulator
3. Absolute
4. Zero Page
5. Immediate
6. Absolute,X
7. Absolute,Y
8. Zero Page,X
9. Zero Page,Y
10. Relative
11. Indirect
12. Indexed Indirect
13. Indirect Indexed

1. Implied

Bij instructies van dit type ligt het adres van de data besloten in de opcode zelf of het zijn instructies die

verder geen data nodig hebben (bijv. NOP). Bij de 6502 zijn dat de instructies BRK, CLC, CLD, CLI, CLV, DEX, DEY, INX, INY, NOP, PHA, PHP, PLA, PLP, RTS, RTI, SEC, SED, SEI, TAX, TAY, TSX, TXA, TXS en TYA (hijg, hijg). Een aantal van deze instructies hebben we al gebruikt in de voorbeelden, een fijnere onderverdeling zou kunnen zijn:

a. instructies voor controle van status register bitjes:
CLC, CLD, CLI, CLV, SEC, SED, SEI

b. instructies voor data transfer tussen registers onderling:
TAX, TAY, TSX, TXA, TXS, TYA

c. instructies voor stack operaties:
PHA, PHP, PLA, PLP, RTS, RTI

d. instructies voor de index registers X en Y:
DEX, DEY, INX, INY

e. overig:
BRK, NOP

Alle implied opcodes zijn slechts een byte lang. De AS syntax is simpel, eenvoudig de 3 karakters voor de opcode intypen en klaar is Clara!

Voorbeeld:

Instructie	Memory
NOP	EA

2. Accumulator

Deze adresseer mogelijkheid is eigenlijk ook Implied, namelijk instructies waarbij in de opcode zelf al vastligt dat ze betrekking hebben op de accu. Bij de 6502 zijn dat er maar vier:

a. ALU bewerkingen:
ASLA, LSRA, ROLA en RORA.

Allemaal schuifoperaties (zie deel 3) die werken met de inhoud van de accu en daar ook het resultaat weer in terugzetten. Ook deze instructies zijn een byte lang. De AS syntax is uitgebreid met een A achter de instructie. Sommige assemblers gebruiken ook wel ASL-A of ASL A. Het onderscheid tussen ASL A en ASL $A is dan echter wat moeilijk (voor de lezer, NIET voor de assembler).

Voorbeeld:

Instructie	Memory
ASLA	0A

3. Absolute

Bij deze methode wordt het adres volledig beschreven in twee bytes die volgen op de opcode. De postbode van de processor weet dus precies waar ie moet wezen want zoals we al zagen in deel 1, de 6502 heeft een adres bereik van 64K en met 16 bits kun je een enkel adres precies beschrijven. De absolute methode wordt gebruikt bij:

a. ALU bewerkingen:
ADC, AND, ASL, BIT, CMP, CPX, CPY, EOR, INC, DEC, LSR, ORA, ROL, ROR, en SBC

b. Registers laden en schrijven:
LDA, LDX, LDY, STA, STX en STY

c. Programma flow instructies:
JMP en JSR

De absolute instructies zijn altijd 3 bytes lang, t.w. de opcode en 2 bytes voor het adres. Zoals we al weten staat het LSB (Least Significant Byte) van het adres in de eerste geheugenlocatie, en het MSB (Most Significant Byte) in de daaropvolgende (zie ook deel 1). De syntax voor de AS assembler is eerst de instructie en vervolgens een 16 bits getal of een label.

Voorbeeld:

instructie		Memory
LDA	$1234	AD 34 12

4. Zero Page

Bij deze methode 'weet' de processor (door de opcode) dat het adres op de eerste pagina van het geheugen staat, zoals we al gezien hebben zijn pagina's precies 256 bytes groot en dus heeft de postbode aan een enkel byte genoeg. Je zou kunnen zeggen dat Zero Page adresering een soort wijkpost is terwijl de Absolute adresering voor de hele stad met 65535 inwoners te gebruiken is. De Zero Page methode is bij precies dezelfde instructies als hierboven bij de Abslotute methode is beschreven. Met uitzondering van de JSR en de JMP instructie. Het voordeel van de Zero Page methode is dat het programma sneller wordt, je kunt met je boerenverstand aanvoelen dat de processor minder werkt heeft aan het ophalen van de instructie (een byte korter). Nadeel is dat bij bijna alle 6502 systemen een gedeelte van de Zero Page al in gebruik is bij de systeem software zodat niet alle locaties in jouw flitsende programma gebruikt mogen worden. Bij het DOS65 systeem valt dat gelukkig allemaal mee, omdat hier de systeem software immers voor een groot gedeelte zelf in RAM staat zijn er heel weinig locaties van de Zero Page in gebruik. Voor zover ik weet zijn dat adres 0..3 (PTA en PTB) die (tijdelijk) in gebruik zijn in IO65 (de data in die locaties wordt overigens wel netjes opgeslagen en later terug gezet, zodat je bij normaal gebruik deze locaties best kunt gebruiken, bij programma's die veel interrupts gebruiken is het oppassen geblazen). Verder worden de adressen $FB, $FC, $FE en $FF (tijdelijk) gebruikt tijdens het 'booten' van het systeem. (klopt dat Erwin?). De Zero Page instructies zijn alle 2 bytes lang. De syntax voor AS is net als bij Absolute alleen moet het adres maar 8 bits zijn of het label moet verwijzen naar de zero page.

Voorbeeld:

Instructie	Memory
STA $12	85 12

5. Immediate

Dit is de meest directe vorm om iets aan de processor door te geven, de te gebruiken data staat namelijk direct op de volgende geheugenlocatie na de opcode. Er hoeft dus geen postbode door weer en wind gestuurd te worden om ergens in de binnenlanden van Borne Bokkie een pakketje data op te gaan halen. Je zou verwachten dat deze methode ook toegepast zou kunnen worden bij alle ALU bewerkingen, maar een aantal van die bewerkingen zijn zogenaamde Read-Modify-Write instructies. Dat is Frans voor: data lezen, doe er iets mee en zet vervolgens het resultaat weer terug waar je het gevonden hebt. Het is duidelijk dat deze operaties niet echt bruikbaar zijn bij de immediate instructie, immers iedere keer dat je programma zou starten zou er een ander getal achter de opcode (kunnen) staan. Kortom: de immediate methode wordt alleen gebruikt waarbij een getal gelezen wordt, nooit voor schrijfoperaties. Over blijven dus die ALU bewerkingen waarbij alleen data gelezen wordt en het resultaat alleen in de Accu komt te staan.

a. ALU bewerkingen:
ADC, AND, CMP, CPX, CPY, EOR, ORA en SBC
b. Register laden:
LDA, LDX en LDY

De immediate instructies zijn altijd 2 bytes lang. De syntax voor AS is eerst de instructie, vervolgens een # karakter en dan de 8 bits data of een EQU (ja, ook van 8 bits ja!).

Voorbeeld:

Instructie	Memory
AND #$AA	29 AA

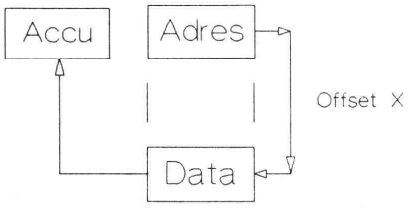


Fig. 1: LDA,X

6. Absolute,X

Deze methode heet ook wel 'indexed adressing', het adres wordt namelijk berekend door bij het 16 bits absolute adres het index register X op te tellen. De postbode krijgt dus een adres door, fietst ermee naar vriend X, krijgt daar zijn offset, zoals de Fransen zeggen, gebruikt vervolgens z'n zakjapanner en weet dan pas waar ie naar toe moet met zijn door hogerhand verstrekte dienstvoertuig. Deze methode is heel handig als je de postbode een tabel wilt laten doorfietsen, in de instructie staat dan het beginadres van de tabel en het index register X doet de rest. Je ziet dat X register kan heel wat meer dan een simpel tellertje. In figuur 1 is dit schematisch weergegeven.

Deze methode kan gebruikt worden bij bijna alle instructies die ook bij de standaard Absolute methode genoemd worden, behalve die instructies die betrekking hebben op het index register X zelf CPX, LDX en STX (kan ik nog begrijpen), de JMP en JSR (kan ik ook nog begrijpen) maar ook op de instructies BIT, CPY en STY (en daar begrijp ik geen hout van, waarom wel LDY en geen CPY of STY meneer Rockwell?). Over blijven dus:

a. ALU instructies:
ADC, AND, ASL, CMP, DEC, EOR, INC, LSR, ORA, ROL, ROR en SBC.
b. Registers laden en schrijven:
LDA, LDY en STA

De instructie lengte is altijd 3 bytes. De syntax voor AS is precies hetzelfde als bij de Absolute methode alleen wordt de operand nu gevolgd door een komma en een X (of x).

Voorbeeld:

Instructie	Memory
STA TABEL,X	9D 00 20 (dus TABEL begint op $2000)

7. Absolute,Y

Nou zul je zeggen, die kennen we al. I.p.v. naar X te fietsen, fietst onze postbode nu eerst langs Y alvorens zijn zakjapanner te voorschijn te halen. Inderdaad, beste lezer (en lieve lezeres) U heeft volkomen gelijk! In figuur 2 is deze adresseermethode schematisch weergegeven.

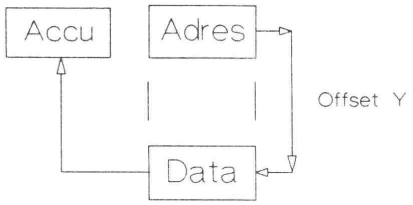


Fig. 2: LDA,Y

Dus alle instructies die bij Absolute,X gebruikt worden kan ik ook bij Absolute,Y gebruiken? Neen, beste lezer (en lieve lezeres) nu heeft U het mis. Meneer Rockwell heeft in zijn oneindige wijsheid beslist dat het niet nodig is om alle instructies zowel in Absolute,X als in Absolute,Y uit te voeren. Wat er over blijft is:

a. ALU instructies:
   ADC, AND, CMP, EOR, ORA en SBC
b. Registers:
   LDA, LDX en STA

De instructielengte en de syntax voor AS is precies hetzelfde als bij de voor gaande methode alleen vervangen we de X (of x) door een Y (of y).

Voorbeeld:

Instructie	Memory
LDX OFFSET,Y	BE 34 12 (OFFSET tabel start op $1234)

8. Zero Page,X

Welnu, even de grijze massa in beweging en je weet gelijk hoe deze instructie groep te werk gaat. Ze geven de postbode een adres in de wijk Zero Page, die fietst (alweer) naar vriend X, gebruikt z'n zakjapanner en weet het huisadres van de gevraagde sjampetter. Voordeel van deze methode is dat de instructie een byte korter is, nadeel is dat de tabel alleen in Zero Page kan staan. Alle instructies die opgesomd zijn bij Absolute,X kunnen ook gebruikt worden met deze adresseermethode en nu wel met STY (how come, mr. Rockwell?) Schematisch is dit verhaal hetzelfde als bij Absolute,X mits je in gedachten houdt dat ADRES nu ergens tussen $0000 en $00FF moet liggen. De instructielengte is 2 bytes en de AS syntax is precies hetzelfde als bij Absolute,X alleen moet de operand nu 8 bits zijn.

Voorbeeld:

Instructie	Memory
STY 50,X	94 50

9. Zero Page,Y

Deze hoef ik niet meer uit te leggen, dacht ik zo. Ook hier heeft meneer Rockwell weer drastisch bezuinigd want bij deze adresseermethode zijn nog slechts twee instructies overgebleven, namelijk LDX en STX. En dat is best wel vervelend soms, je kunt de Accu in de Zero Page dus alleen via X op de verkorte manier (2 bytes) laden e.d. via Y heb je dan gelijk Absolute,Y dus 3 bytes nodig. De instructielengte is 2 bytes, de AS syntax is weer idem als bij Zero Page,X door de X te vervangen met een Y.

Voorbeeld:

Instructie	Memory
STX DATA,Y	96 40 (DATA begint bij adres $0040)

10. Relative

Bij de 6502 wordt deze adresseermethode alleen gebruikt voor de offset van de branches. De data achter de opcode wordt gebruikt om de programmateller ten opzichte van de huidige waarde van de programmateller of PC (zie ook deel 1) te gaan verplaatsen. Het komt er dus op neer dat er een sprong wordt uitgevoerd van -128 tot +127 t.o.v. de huidige locatie. De 6502 kent totaal 8 verschillende branches die alle aan de hand van een bit van het statusregister P wel of niet werken. De conditionele branch dus. Dat zijn:

a. Branches die de C-flag testen:
   BCC, BCS
b. Branches die de Z-flag testen:
   BEQ, BNE
c. Branches die de N-flag testen:
   BMI, BPL
d. Branches die de V-flag testen:
   BVC, BVS

Het zelf berekenen van zo'n relatieve branch gebeurt vanaf de locatie waar de offset moet komen te staan. Voorwaartse referenties zijn eenvoudig uit te tellen. Moeten we bijvoorbeeld over een JMP instructie heen 'branchen' dan is de offset 3 (immers een JMP is 3 bytes lang) (zie figuur 3).

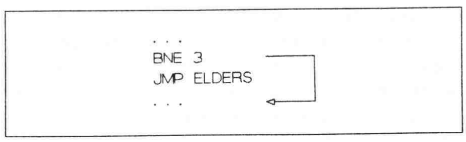


Fig. 3

Een terugwaartste referentie is wat lastiger omdat we nu het two's complement moeten berekenen, bovendien moet de lengte van de branch zelf t.w. 2 bytes ook worden meegeteld. Programeren we nu de kortste oneindige programmaloop die mogelijk is dan moeten we dus als offset -2 opgeven. In 2's complement wordt dat:

```
2 =    00000010
       11111101      (1's complement, alle bits
                                  inverteren)
       00000001 +    (+1)
       11111110      = $FE
```

Ons superkorte nutteloze processortijd verspillende programma wordt dus zoals in figuur 4 afgedrukt is.

```
      CLC        ; Initialisatie
  ┌─▷ BCC $FE    ; and start
  │              ; loop ad infinitum
  └──────┘
```

Fig. 4

Het effectieve bereik van een branch is gegeven door de 8 bits signed offset dus -128 tot +127. Omdat de instructie zelf 2 bytes lang is wordt dat dus -128 + 2 = -126 (terug) tot 127 + 2 = 129 (vooruit). De instructie lengte is 2 bytes, de AS syntax is de instructie gevolgd door de relatieve offset als je zelf graag relatieve jumps uitrekend, anders een verwijzing naar een label binnen het bereik van de branch.

Voorbeeld:

Instructie	Memory
BNE 3	D0 03

11. Indirect

Dit is een buitenbeentje (of stiefkind) bij de 6502. De methode werkt als volgt, de postbode krijgt een adres door van de processor, hij fietst erheen en krijgt op dat adres te horen waar het pakketje naar toe moet (of moet worden opgehaald). Algemeen: de operand wijst naar adres A1 dat op zijn beurt wijst naar adres A2 waar de data staat. Bij de 6502 wordt deze methode alleen gebruikt voor de JMP instructie. In figuur 5 is de indirecte methode schematisch weergegeven.

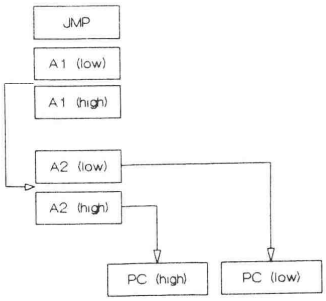


Fig. 5: JMP [A1] ; Jump indirect

Deze instructie is machtig mooi voor gebruik in een menu programma, het programma moet dan immers aan de hand van de invoer steeds een subprogrammaafwerken en dan weer terugkomen in de hoofdlus. Dat kan natuurlijk als volgt:

```
        ...              ; keuze zit in A (0...X)
        CMP  #0          ; was het keuze 0?
        BNE  +6          ; nee, ga door
        JSR  KEUZE0      ; ja, doe subroutine
        JMP  LOOP
        CMP  #'1         ; keuze 1?
        BNE  +6
        JSR  KEUZE1
        JMP  LOOP
        ...
```

Heel vervelend programeren is dat, bovendien heeft het als groot nadeel dat het niet eenvoudig is uit te breiden. Het programma wordt immers bij iedere keuze 10 bytes langer. Het is eigenlijk alleen bruikbaar bij relatief weinig mogelijkheden. En last but not least, iedereen kan dit zo schrijven. De Echte Programmeur doet het met tabellen, en wel als volgt:

```
        ...              ; keuze zit in A (0...X)
        ASLA             ; A = A * 2
        TAX              ; zet A in X
        LDA  KEUZE,0     ; haal subroutine adres
                                          (low)
        STA  ADRES
        LDA  KEUZE+1,X   ; en high part
        STA  ADRES+1
        JSR  DO_IT       ; doe geselecteerde
                                        functie
        JMP  LOOP        ; en voor de volgende keer
DO_IT   JMP  [ADRES]     ; ga subroutine uitvoeren

KEUZE   FDB  KEUZE0,KEUZE1,KEUZE2,....
ADRES   FDB 0000
```

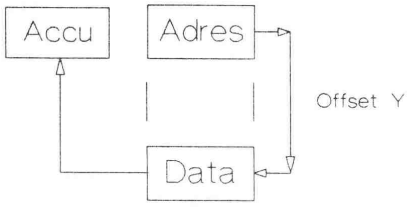


Fig. 2: LDA,Y

Dus alle instructies die bij Absolute,X gebruikt worden kan ik ook bij Absolute,Y gebruiken? Neen, beste lezer (en lieve lezeres) nu heeft U het mis. Meneer Rockwell heeft in zijn oneindige wijsheid beslist dat het niet nodig is om alle instructies zowel in Absolute,X als in Absolute,Y uit te voeren. Wat er over blijft is:

a. ALU instructies:
ADC, AND, CMP, EOR, ORA en SBC
b. Registers:
LDA, LDX en STA

De instructielengte en de syntax voor AS is precies hetzelfde als bij de voor gaande methode alleen vervangen we de X (of x) door een Y (of y).

Voorbeeld:

Instructie	Memory
LDX OFFSET,Y	BE 34 12 (OFFSET tabel start op $1234)

8. Zero Page,X

Welnu, even de grijze massa in beweging en je weet gelijk hoe deze instructie groep te werk gaat. Ze geven de postbode een adres in de wijk Zero Page, die fietst (alweer) naar vriend X, gebruikt z'n zakjapanner en weet het huisadres van de gevraagde sjampetter. Voordeel van deze methode is dat de instructie een byte korter is, nadeel is dat de tabel alleen in Zero Page kan staan. Alle instructies die opgesomd zijn bij Absolute,X kunnen ook gebruikt worden met deze adresseermethode en nu wel met STY (how come, mr. Rockwell?) Schematisch is dit verhaal hetzelfde als bij Absolute,X mits je in gedachten houdt dat ADRES nu ergens tussen $0000 en $00FF moet liggen. De instructielengte is 2 bytes en de AS syntax is precies hetzelfde als bij Absolute,X alleen moet de operand nu 8 bits zijn.

Voorbeeld:

Instructie	Memory
STY 50,X	94 50

9. Zero Page,Y

Deze hoef ik niet meer uit te leggen, dacht ik zo. Ook hier heeft meneer Rockwell weer drastisch bezuinigd want bij deze adresseermethode zijn nog slechts twee instructies overgebleven, namelijk LDX en STX. En dat is best wel vervelend soms, je kunt de Accu in de Zero Page dus alleen via X op de verkorte manier (2 bytes) laden e.d. via Y heb je dan gelijk Absolute,Y dus 3 bytes nodig. De instructielengte is 2 bytes, de AS syntax is weer idem als bij Zero Page,X door de X te vervangen met een Y.

Voorbeeld:

Instructie	Memory
STX DATA,Y	96 40 (DATA begint bij adres $0040)

10. Relative

Bij de 6502 wordt deze adresseermethode alleen gebruikt voor de offset van de branches. De data achter de opcode wordt gebruikt om de programmateller ten opzichte van de huidige waarde van de programmateller of PC (zie ook deel 1) te gaan verplaatsen. Het komt er dus op neer dat er een sprong wordt uitgevoerd van -128 tot +127 t.o.v. de huidige locatie. De 6502 kent totaal 8 verschillende branches die alle aan de hand van een bit van het statusregister P wel of niet werken. De conditionele branch dus. Dat zijn:

a. Branches die de C-flag testen:
BCC, BCS
b. Branches die de Z-flag testen:
BEQ, BNE
c. Branches die de N-flag testen:
BMI, BPL
d. Branches die de V-flag testen:
BVC, BVS

Het zelf berekenen van zo'n relatieve branch gebeurt vanaf de locatie waar de offset moet komen te staan. Voorwaartse referenties zijn eenvoudig uit te tellen. Moeten we bijvoorbeeld over een JMP instructie heen 'branchen' dan is de offset 3 (immers een JMP is 3 bytes lang) (zie figuur 3).

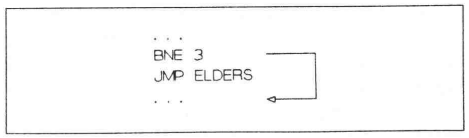


Fig. 3

Een terugwaartste referentie is wat lastiger omdat we nu het two's complement moeten berekenen, bovendien moet de lengte van de branch zelf t.w. 2 bytes ook worden meegeteld. Programeren we nu de kortste oneindige programmaloop die mogelijk is dan moeten we dus als offset -2 opgeven. In 2's complement wordt dat:

```
2 =    00000010
       11111101      (1's complement, alle bits
                                  inverteren)
       00000001 +    (+1)
       11111110      = $FE
```

Ons superkorte nutteloze processortijd verspillende programma wordt dus zoals in figuur 4 afgedrukt is.

```
      CLC          ; Initialisatie
  ┌─▷ BCC $FE      ; and start
  │                ; loop ad infinitum
  └──┘
```

Fig. 4

Het effectieve bereik van een branch is gegeven door de 8 bits signed offset dus -128 tot +127. Omdat de instructie zelf 2 bytes lang is wordt dat dus -128 + 2 = -126 (terug) tot 127 + 2 = 129 (vooruit). De instructie lengte is 2 bytes, de AS syntax is de instructie gevolgd door de relatieve offset als je zelf graag relatieve jumps uitrekend, anders een verwijzing naar een label binnen het bereik van de branch.

Voorbeeld:

Instructie	Memory
BNE 3	D0 03

11. Indirect

Dit is een buitenbeentje (of stiefkind) bij de 6502. De methode werkt als volgt, de postbode krijgt een adres door van de processor, hij fietst erheen en krijgt op dat adres te horen waar het pakketje naar toe moet (of moet worden opgehaald). Algemeen: de operand wijst naar adres A1 dat op zijn beurt wijst naar adres A2 waar de data staat. Bij de 6502 wordt deze methode alleen gebruikt voor de JMP instructie. In figuur 5 is de indirecte methode schematisch weergegeven.

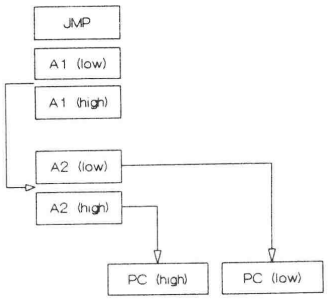


Fig. 5: JMP [A1] ; Jump indirect

Deze instructie is machtig mooi voor gebruik in een menu programma, het programma moet dan immers aan de hand van de invoer steeds een subprogrammaafwerken en dan weer terugkomen in de hoofdlus. Dat kan natuurlijk als volgt:

```
        ...              ; keuze zit in A (0...X)
        CMP  #0          ; was het keuze 0?
        BNE  +6          ; nee, ga door
        JSR  KEUZE0      ; ja, doe subroutine
        JMP  LOOP
        CMP  #'1         ; keuze 1?
        BNE  +6
        JSR  KEUZE1
        JMP  LOOP
        ...
```

Heel vervelend programeren is dat, bovendien heeft het als groot nadeel dat het niet eenvoudig is uit te breiden. Het programma wordt immers bij iedere keuze 10 bytes langer. Het is eigenlijk alleen bruikbaar bij relatief weinig mogelijkheden. En last but not least, iedereen kan dit zo schrijven. De Echte Programmeur doet het met tabellen, en wel als volgt:

```
        ...              ; keuze zit in A (0...X)
        ASLA             ; A = A * 2
        TAX              ; zet A in X
        LDA  KEUZE,0     ; haal subroutine adres
                                        (low)
        STA  ADRES
        LDA  KEUZE+1,X   ; en high part
        STA ADRES+1
        JSR  DO_IT       ; doe geselecteerde
                                        functie
        JMP  LOOP        ; en voor de volgende keer
DO_IT   JMP  [ADRES]     ; ga subroutine uitvoeren

KEUZE   FDB  KEUZE0,KEUZE1,KEUZE2,....
ADRES   FDB 0000
```

De tabel KEUZE bevat nu de subroutine adressen. De ASLA is nodig omdat de tabel 16 bits adressen bevat, en de offset dus 2 bytes per entry moet overslaan. Het grote voordeel van deze methode is evident, mijn waarde Watson. Het is leuk programeren, het is eenvoudig uit te breiden door alleen de tabel aan te passen, het programma wordt per entry slechts 2 bytes langer, en het getuigt van genialiteit. De instructie lengte is 3 bytes, de AS syntax is de JMP instructie gevolgt door de operand tussen blokhaken. Bij een aantal andere assemblers wordt ook wel gebruik gemaakt van de JMI (Jump Indirect) instructie, of staat de operand data tussen (normale) ronde haken.

Voorbeeld:

Instructie	Memory
JMP [SUBR]	6C 00 33 (SUBR staat op adr. $3300)

12. Indexed indirect

Dit is een hele rare druif. De postbode krijgt een adres door binnen de wijk Zero Page, hij fietst naar X en krijgt daar een offset. Na gebruik van zijn zakjapanner fietst ie naar de aangegeven locatie en krijgt daar van de bewoner en zijn buurman het adres door van de gezochte knakker. De arme postbode fietst dus heel wat af bij deze methode. En dat is dan ook goed te merken aan de executietijd van deze groep instructies, want hoewel ze alle slechts 2 bytes lang zijn heeft de processor er toch 6 cycles voor nodig. Deze methode is in figuur 6 schematisch weergegeven.

Deze methode gebruikt dus een tabel van 16 bits pointers en het index register is steeds een veelvoud van 2. Eerlijk gezegd heb ik zelf deze methode heel weinig gebruikt, simpel omdat ik er geen toepassing voor wist. Een groot nadeel is ook dat de tabel in Zero Page moet staan. De methode is geimplementeerd voor de volgende instructies:

a. ALU operaties:
ADC, AND, CMP, EOR, ORA en SBC
b. Register operaties:
LDA en STA

De instructie is 2 bytes lang, de AS syntax is eerst de instructie gevolgd door een 8 bits operand met ,X en dat geheel tussen blokhaken (zie voorbeeld).

Voorbeeld:

Instructie	Memory
ADC [PTR,X]	61 20 (PTR staat op $0020)

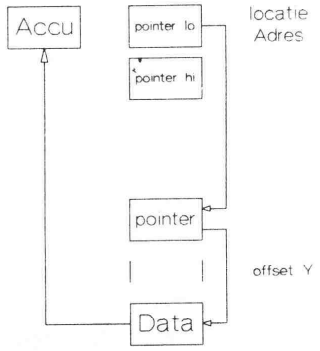


Fig. 7: LDA [ADRES],Y

13. Indirect indexed

Dit is een beauty. De postbode krijgt een adres in de wijk Zero Page, daar krijgt ie van de bewoner en diens buurman een adres door. Vervolgens telt ie met z'n zakjapanner de inhoud van Y bij dat adres en fietst daar vervolgens naar toe. Waarom is dit nu zo'n mooie? Welnu op deze manier heeft de 6502 eigenlijk een 16 bits index register met bovendien daarbovenop nog een 8 bits offset (het Y register). Omdat er in de Zero Page 256 locaties zijn heeft de 6502 dus 128 16 bits index registers!! In figuur 7 is deze methode schematisch weergegeven.

Deze methode werkt voor de volgende instructies:

a. ALU operaties:
ADC, AND, CMP, EOR, ORA en SBC
b. Register operaties:
LDA en STA

De instructie lengte is 2 bytes, de AS syntax is de instructie gevolgd door een 8 bits adres (in Zero Page dus) tussen blok haken en dit weer gevolgd door ,Y (zie voorbeeld).

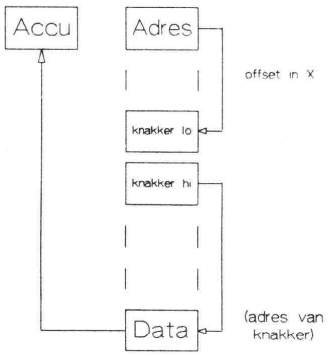


Fig. 6: LDA [ADRES,X]

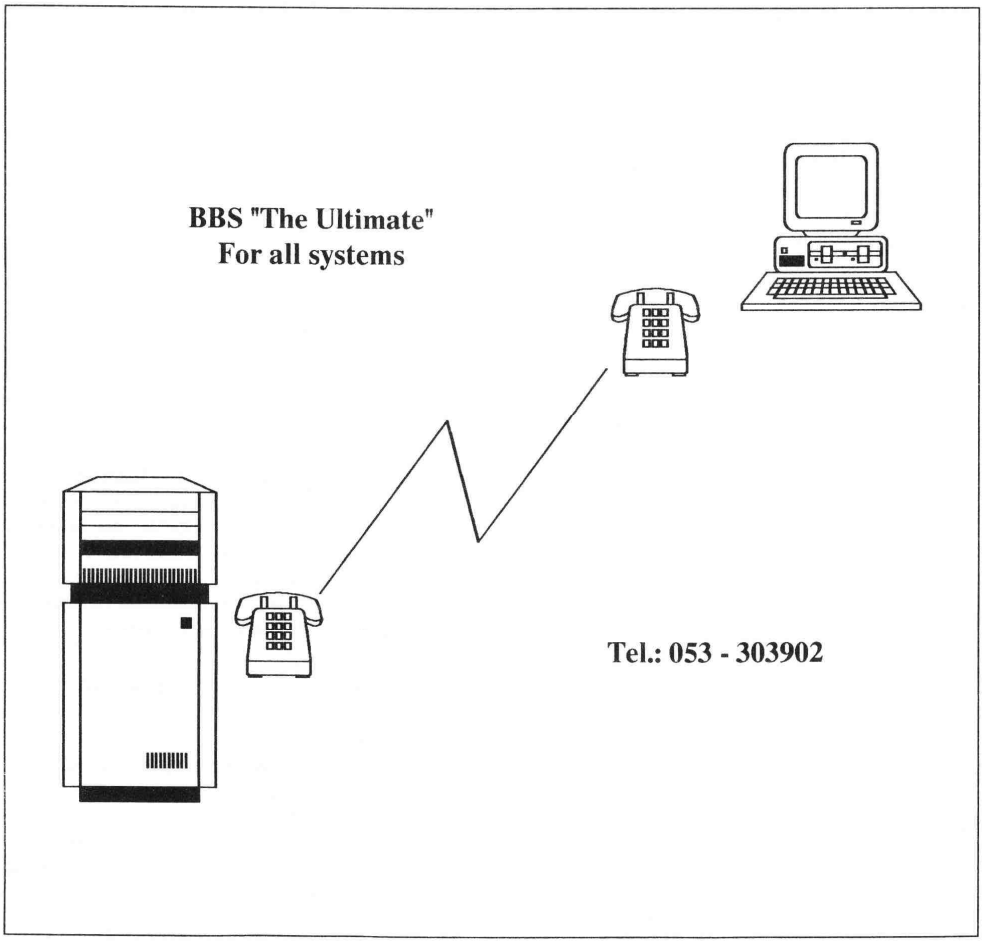
Voorbeeld:

Instructie	Memory
LDA [PTR],Y	B1 20
	(PTR staat op $0020)

Tenslotte

Zo, dat zijn ze. Hiermee zul je het moeten doen, meer heeft de 6502 niet aan boord. Nog een kort woord over de te kiezen methode. Eigenlijk is dat helemaal vrij. Dat is juist het leuke van programeren. Je kunt er een eigen stijl in ontwikkelen, maar pas op voor broddelwerk. Ik hoop dat dit verhaal een beetje duidelijk is geworden, helaas dit keer geen voorbeeld programma van de hand van uw nederige dienaar. Voor de volgende keer heb ik eigenlijk nog geen onderwerp, dus laat eens wat van je horen. Wat wil je graag behandeld hebben? Subroutines? Interrupts? Vectoren? I/O technieken? Graag wat reacties dus, of zit ik toch in een zwart gat te schrijven, is dit inderdaad alleen wat bezigheidstherapie zoals mijn psychiater me voorschreef om van de spanningen af te komen? Reacties liefst via ons BBS (053-303902), via de redactie Gert van Opbroek, maar je kunt mij ook direct bellen ('s-avonds 038-537073).

Antoine Megens



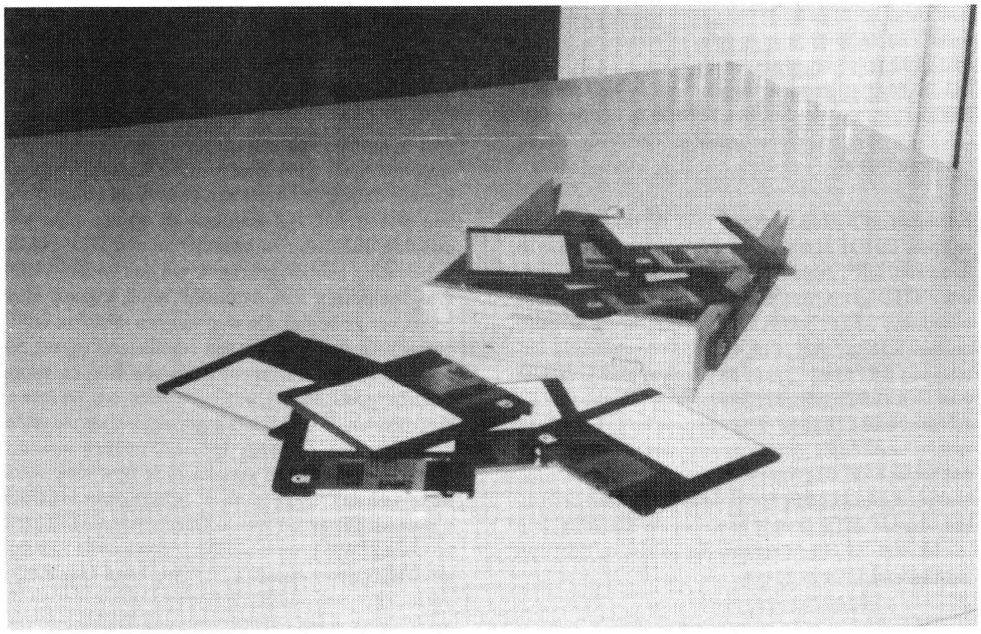
Een 3.5 inch diskettedrive

De diskette drives worden steeds goedkoper. Zo kon je een paar jaar geleden een 5.25 inch diskette drive kopen voor zo'n zeshonderd gulden, nu is de prijs gehalveerd en op het moment zie je zelfs aanbiedingen van 3.5 inch drives voor onder de tweehonderd gulden (80 track / dual sided, bijv. van Timtronix). Deze prijzen gelden voor fabrieksnieuwe drives, tweedehands drives zijn uiteraard nog goedkoper. Kortom, het loont de moeite om te zien wat de mogelijkheden zijn op de diverse computersystemen. Het antwoord is eenvoudig, de drives zijn zo goed als uitwisselbaar met bestaande 5.25 inch drives. Er zijn echter een paar verschillen.

- De connectoren zijn verschillend. De voedingsconnector van de 5.25 inch drive is een moeilijk verkrijgbaar geval met vier holle pennen die resp. zorgen voor +12V, gnd, gnd, +5V. Bij de 3.5 inch drives is deze connector vervangen door een type waarop zelfs de standaard flatcable connectors passen. De aan te bieden spanningen zijn identiek aan die van de 5.25 inch drive, het stroomverbruik kan flink lager zijn.

- Voor de interface-connector geldt hetzelfde, de 5.25 inch connector is een zogenaamde 'edge' type, een connector die op de rand van een printplaat geschoven moet worden. Bij de 3.5 inch drive is deze connector vervangen door een standaard flatcable connector van een paar gulden (34 pin).

- De interface-signalen komen bijna overeen. Alle oneven pennen liggen aan massa. Het enige probleem dat we tegenkwamen was het ready signaal (pin 34). Dit signaal geeft aan dat er een diskette in de drive aanwezig is en dat de drive "op toeren" is. Onder DOS-65 en op de MC68000 verwacht de controller een ready signaal en deze werd niet door een 3.5 inch drive afgegeven. Op de TEAC fd35fgn blijken een aantal strap-mogelijkheden aanwezig en bij één van die mogelijkheden staat 'RY'. Door hier een kortsluitstripje op te steken, wordt het probleem verholpen en geeft de drive het Ready-signaal af. Op een ander type drive, de ALPS S/N1A zijn geen strapmogelijkheden gevonden. Deze drives zijn ingebouwd in de MC68000 van Gert van Opbroek. Door in dit systeem op de controller een verbinding te maken tussen gnd en pen 34, is hier het probleem verholpen. De drive geeft nu altijd Ready en dat werkt.



3.5" diskdrive, ook in de PC

In het volgende overzicht zijn de diverse signalen weergegeven.

De interface signalen:

2		reserved
4	in	in use / head load
6	in	drive select 3
8	out	index / sector
10	in	drive select 0
12	in	drive select 1
14	in	drive select 2
16	in	motor on
18	in	direction (track steppermotor)
20	in	step
22	in	write data
24	in	write gate
26	out	track 00
28	out	write protect
30	out	read data
32	in	side select
34	out	ready

Alle oneven pennen liggen aan massa.
In = signaal van controller naar drive.

De voedingsplug:

1	+ 12 Volt
2	massa
3	massa
4	+ 5 Volt

Voor mensen die op een eenvoudige manier hun 5.25 inch drives uit willen wisselen voor een 3.5 inch drive, zijn er bij de meeste merken zogenaamde inbouwframes te koop. Een 3.5 inch drive in een dergelijk frame heeft dezelfde afmetingen als een 5.25 inch drive. Verder verzorgt het frame ook de overgang van de connectoren voor een 5.25 drive naar die van een 3.5 inch drive.

Een 3.5 inch drive heeft verschillende voordelen. Ze is kleiner dus gemakkelijker (verticaal!) in de kast te passen, heeft minder stroomopname en is vooral geruislozer. Bovendien zijn de floppy's wat handzamer.

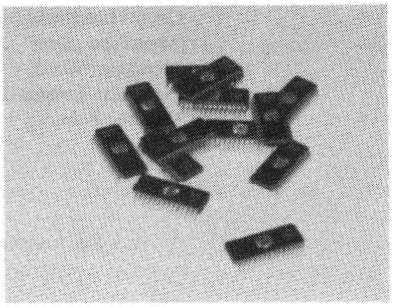
Een MS-DOS machine heeft meestal de beschikking over 360 kB of over 1.2 MB 5.25 inch floppy drives. Wisselt men een 360 kB drive in voor een 3.5 inch drive, dan ruilt men meestal een 40 tracks drive in voor een 80 tracks drive. Bij de XT-compatible systemen, kan men dan 720 kB in plaats van 360 kB op een floppy kwijt. Voor de 1.44 MB die bij een AT-compatible systeem op een 3.5 inch floppy geschreven kan worden heeft men een speciale drive nodig. Bovendien wordt dit formaat niet door de XT ondersteund. Wil men een 3.5 inch floppy drive in een XT-compatible systeem gebruiken, dan moet men (bij mijn weten) een MS-DOS versie 3.0 of hoger gebruiken. Bovendien moet in de CONFIG.SYS-file deze drive gespecificeerd worden. Voor een echte kloon is er echter ook een andere mogelijkheid. Het door Nico de Vries ontwikkelde BIOS is in staat een eventuele 3.5 inch drive te herkennen en definieert deze dan ook als een drive van 720 kB. Dit BIOS is via de penningmeester te bestellen (zie elders in dit nummer).

Ernst Elderenbosch

(Advertentie)

Een 3.5" drive in uw PC/XT?
Slowstart door "power on selftest"?
Gebrek aan seriële poorten?

Dan wordt het de hoogste tijd voor een nieuw BIOS! Het speciale KIM-club BIOS bijvoorbeeld. Alleen verkrijgbaar voor leden...



Bestellen:

Maak *f* 25,- over aan de KIM-Gebruikers Club Nederland in Enschede, giro 3757649 o.v.v. "KIM XT-BIOS".
U krijgt de EPROM dan zo snel mogelijk thuis gestuurd.

Een ramdisk voor DOS-65

Het DOS65 systeem heeft heel veel ingebouwde mogelijkheden. Zo kun je standaard 4 diskdrives aansluiten, floppy, winchester of virtual disk. De drivers die nodig zijn om floppy drives aan te sturen zijn al aanwezig en ook voor de virtual disk, die het gewone geheugen van de 6502 gebruikt (weliswaar alleen het gedeelte boven 56k), is de complete software ondersteuning al 'ingebakken' in dos65. Alleen voor de winchester disk zijn de drivers nog niet standaard aanwezig. Als het dos voor winchester geassembleerd wordt, moeten hierin twee adressen vastgelegd worden van de twee driver routines. Deze twee routines zorgen resp. voor het schrijven en voor het lezen van een sector van harde schijf. Verder wordt het aan de user overgelaten hoe die routines er uit moeten zien.

Ik heb voor mijn systeem gekozen voor een leesroutine die begint op $e200 en een schrijfroutine met een startpunt op $e203. Op het moment dat dos65 een sector wil lezen of schrijven wordt het tracknr doorgegeven in het x register (max. 256 tracks) en het sectornr in het y register (max. 128 sectors). De meeste drives maken gebruik van 32 sectors per track (x 256 bytes).

De vrijheid die dos65 biedt om zelf diskdrivers te schrijven maken het systeem zeer flexibel. Zo is het helemaal niet zo moeilijk om een driver te maken die een harddisk simuleert terwijl gebruik wordt gemaakt van een heel ander medium, zoals een z80 kaart of zelfs een ander computersysteem. Ook een harddisk driver is niet moeilijk te maken, bijvoorbeeld voor een SASI drive of SCSI drive. In een volgend artikel wil ik graag de listing geven van een driver voor een SASI harddisk.

De z80 kaart heeft in de eenvoudigste uitvoering maar 64k. Hiervan kunnen we ruim 4k niet gebruiken want dit wordt ingenomen door het operating system van de z80. Er blijft dus krap 60 k over. Dit lijkt heel weinig, maar blijkt ruim voldoende voor een redelijk grote assembler source plus de vertaalde versie. De source van deze driver, de vertaalde versie, de listing en de assembler zelf vonden gemakkelijk een plaatsje in de z80 'harddisk'.

We zijn gewend om een floppy of harddisk eenmaal te formatteren en daarna nooit meer. Op de z80 kaart wordt gebruik gemaakt van dynamische rams en deze moeten telkens weer opnieuw geladen worden met zinnige informatie nadat het systeem uit geweest is. Daarom bevat het eerste deel van de software (zdriver.mac) een initialisatie routine die niet alleen de 6821 registers de juiste startwaarde geeft maar ook de eerste sectors van de z80 disk vult met een lege directory en systeem sector. Er zijn maximaal 239 blocks beschikbaar, inclusief de system en directory sectors. De z80 schijf is volledig als schijf aan te spreken, bijv. met discdoctor, dir, sdir, cat etc. Voor het initialisatie deel gebruik ik $0200-$0470. Dit kan op elke willekeurige plaats in het geheugen gelegd worden. Normaal wordt dit door login.com geladen en uitgevoerd, waarna het kan worden overschreven. De driver routines worden gelijktijdig door login.com geladen (het is gewoon een ander deel van het zelfde programma) maar deze moeten altijd beschikbaar blijven. Ze moeten dan ook geladen worden op een lokatie die niet door andere programma's overschreven wordt. Als je hiervoor bijv. $e400 kiest, dan hoef je verder niets anders te doen dan dit adres in de source file te zetten, het adres van de z80 kaart in de source file te zetten en een versie van dos65 te gebruiken waarin staat dat de harddisk drivers op $e400 beginnen. Deze versie is eenvoudig te maken als je de source files hebt (vraag het de dos65 distributeur of mij). Hierna werkt de z80 kaart zonder verdere hardware wijzigingen als ramdisk. Het systeem ziet 'm alleen als harddisk maar daar merk je verder niets van. Bij mij draait de z80 als drive 2.

Als je net als ik vindt dat elektuur wel wat slordig met de I/O adressen is omgesprongen dan stel ik een kleine wijziging voor. De tapekaart zit bij elektuur op $e280-$e283, de z80 kaart op $e300-$e303 en verder zijn deze twee pagina's leeg. Dat vond ik nogal slordig en daarom heb ik de tape naar $e150 gezet en de z80 kaart naar $e158-$e15b.

De vrijgekomen ruimte op $e200-$e3ff heb ik benut door een stukje ram van de cpu kaart op deze plaats te zetten (deze werd maar voor de helft gebruikt). Hiervoor heb ik een 74ls02 gebruikt op de volgende wijze: A9 naar pin 8, A10 naar pin 9, pin 10 naar pin 11 en naar pin 12, pin 13 naar ingang (pin 11) van n60 (74ls30). Deze ingang moet eerst worden losgemaakt van de 5 Volt. De 74ls02 heb ik met pin 7 en pin 14 bovenop n60 gesoldeerd en de rest van de pennen ingekort en opzij laten steken. De adreslijnen zijn dan vlakbij beschikbaar op de cpu print. Hiermee wordt de 6116 dan voor driekwart gebruikt ipv. voor de helft, maar belangrijker is dat er 512 bytes beschikbaar zijn die door geen enkel programma worden gebruikt. Dit is de ideale plaats voor bijvoorbeeld winchester drivers (of z80 kaart drivers).

De z80 kaart is nog eenvoudiger te modificeren (die kan nu niet op pagina $e3xx blijven zitten). Om deze op $e158 te krijgen kun je het volgende doen: Snij de verbindingen door naar ic23 (74ls688) pin 15, pin 6 en leg deze pennen aan +5Volt (pin 20). Snij de

verbinding door naar pen 18 en verbind pin 18 met 22a van de busconnector. Snij de verbinding door met pin 13 van ic.. en leg daarna pin 13 van ic.. aan 23c van de busconnector. De eerste twee onderbrekingen heb ik aan de soldeer zijde gemaakt en de laatste twee aan de componenten zijde van de print (eps 86238).

Na deze wijzigingen kun je de drivers laden op $e200 ipv. $e400 en de z80 kaart zit nu op $e158. Kijk wel even uit met een elektuur tape kaart. Deze moet nu ook gemodificeerd worden voor je 'm in de bus steekt, anders krijg je dubbele adressering.

Na het laden van zdriver.bin en het uitvoeren van de initialisering (met GO 0200) is de 'silicon disk' gereed voor gebruik. Enkele gemeten tijden om een indruk te geven van de snelheidsverschillen:

AS	zdriver.mac	zdriver.bin	tijd ca.(sec.)
floppy 1	floppy 2	floppy 2	13.25
floppy 1	z80 disk	z80 disk	5.70
z80 disk	z80 disk	z80 disk	3.80
LC (ipv.AS)	z80 disk	z80 disk	2.85

Hierbij werd de source van zdriver geassembleerd met de dos65 assembler op mijn 2 MHz 65C02 systeem naar een .bin file zonder listing. In het laatste geval werd gewoon LC gegeven, waarbij de assembler niet opnieuw geladen wordt, maar slechts opnieuw wordt gestart. De tijden varieren in de praktijk nogal, hier geef ik de snelste tijden weer. Omdat de driver bij mij in het geheugengebied $e200-$e2ff geladen wordt, is het mogelijk om te meten m.b.v. exetime, dat op $e400-$e5ff draait. Het is mogelijk dat ik nog eens uitvis hoe de driver er uit moet zien om optimaal gebruik te kunnen maken van een 256k z80 kaart. Je zou dan ca. 240k 'silicon disk' tot je beschikking hebben. Wie weet. Voor nu in ieder geval: succes!

Ernst Elderenbosch

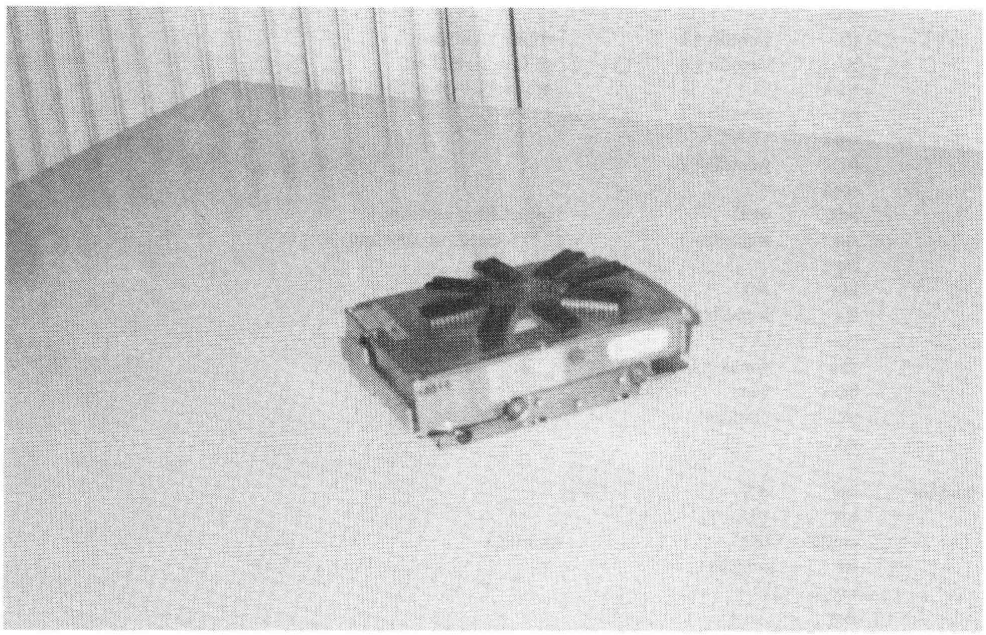
```
; file                    zdriver.mac
;
; purpose       use z80 pcb as harddisk
; system        DOS65 with Z80 card (64k or more)
;
; use                     copy zdriver.bin to disk 0:
;                         include in your login.com:
;
;                         LO S:ZDRIVER.BIN
;                         GO 0200
;
; size                    239 blocks x 256 bytes
;
; date                    21-jan-90               60k silicon disk (z80 card)
;
; author        Ernst R. Elderenbosch
;=================================================================

;WARNING !!
;
;               The following two addresses are system dependant:
;               worg is a free memory area of 256 bytes. In a normal
;               dos65 system, you could use $e400-$e4ff. In that case,
;               do not use EXETIME or my window routines at $e400.
;   -           z80bas is normally $e300, the address of the z80 card.
;
worg            equ     $e200                   ; drivers memory $e200-e3ff
z80bas          equ     $e158                   ; z80 base address
;=================================================================
```

```
zofs        equ     $0100           ; base address in z80 memory to be used
zsiz        equ     $ef00           ; silicon disk size
memp        equ     $e8             ; dos65 r/w memory pointer
prtbyt      equ     $d038           ; dos65 print accu hex routine
prtext      equ     $d03b           ; dos65 print text routine
; z80 commands
cread       equ     $03             read sector
cwrite      equ     $01             write sector
            org     $0200

silicon     jmp     iniz80          ; initialize z80 card
;
;           z80 card initialisation
;
iniz80      lda     #40
            ldy     #44
            ldx     #0              ; set port directions
            sta     z80bas+1
            stx     z80bas
            sty     z80bas+1
            dex
            sta     z80bas+3
            stx     z80bas+2
            sty     z80bas+3
            ldx     #20
1           lda     z80bas          ; purge z80 fifo buffer
            dex
            bne     1.b
            jsr     dirsec
            rts
;
dirsec      lda     memp
            pha
            lda     memp+1
            pha
            ldy     #1
            ldx     #0
            lda     #buffer1&255
            sta     memp
            lda     #buffer1>8
            sta     memp+1
            jsr     winwrite
            ldy     #2
            ldx     #0
            lda     #buffer2&255
            sta     memp
            lda     #buffer2>8
            sta     memp+1
            jsr     winwrite
            ldy     #3
            ldx     #0
            lda     #buffer2&255
            sta     memp
            lda     #buffer2>8
            sta     memp+1
            jsr     winwrite
            ldy     #4
            ldx     #0
```

```
                lda     #buffer2&255
                sta     memp
                lda     #buffer2>8
                sta     memp+1
                jsr     winwrite
                pla
                sta     memp+1
                pla
                sta     memp
                rts
;
buffer1
                fcc     $01,$02,$03,$04,$05,$06,$07,$08 ; 00
                fcc     $09,$0a,$0b,$0c,$0d,$0e,$0f,$10
                fcc     $11,$12,$13,$14,$15,$16,$17,$18 ; 10
                fcc     $19,$1a,$1b,$1c,$1d,$1e,$1f,$20
                fcc     $20,$08,$20,$20,$ff,$ff,$ff,$ff ; 20
                fcc     $01,$01,$00,$c0,$01,$01,$00,$00
                fcc     $00,$00,$00,$00,$00,$00,$00,$00 ; 30
                fcc     $00,$00,$00,$00,$00,$00,$00,$00
                fcc     $53,$69,$6c,$69,$63,$6f,$6e,$20 ; 40
                fcc     $64,$69,$73,$6b,$20,$7a,$38,$30
                fcc     $00,$00,$00,$00,$00,$00,$00,$00 ; 50
                fcc     $09,$08,$e9,$d8,$12,$34,$12,$34
                fcc     $7f,$ff,$ff,$ff,$ff,$ff,$ff,$ff ; bitmap 239 sectors
                fcc     $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff ;  = z80 mem 0100-efff
                fcc     $ff,$ff,$ff,$ff,$ff,$ff,$ff,$ff ;
                fcc     $ff,$ff,$ff,$ff,$fe ;
                res     bufend-*,0              ;
;
                org     buffer1+256
bufend
;
buffer2         res     256,0                   ; directory
;
bufend2
;
;               z80 card driver (read write sector)
                org     worg
winread         jmp     zread
winwrite jmp zwrite
woff            fdb     zofs                    offset address
wmax            fdb     zofs+zsiz               max address

wcmdb           fcc     0                       command buffer, 1=write, 3=read
                fcc     0,0                     z80 address default $100 (=zofs)
                fcc     1,0                     sector length 256 bytes
ytemp           fcb     0
;
;read sector from silicon drive
zread           lda     #cread                  ; get read sector command
                jsr     wrwsec                  ; start read sector procedure
                bcs     9.f                     ; carry set = error
                ldy     #0
read1           sty     ytemp
                jsr     getz80
                sta     [memp],y
                ldy     ytemp
                iny
```

```
            bne     read1
9           rts
;
;
;write sector to silicon drive
;
zwrite      lda     #cwrite         ; get write sector command
            jsr     wrwsec          ; start write sector procedure
            bcs     9.f             ; carry set = error
            ldy     #0
write1      sty     ytemp
            lda     [memp],y
            jsr     outz80
            ldy     ytemp
            iny
            bne     write1
9           rts
;
;
wrwsec      sty     ytemp
            sta     wcmdb           ; command (r/w)
            dey                     ; calculate address
            cpy     #32
            bcs     werr
            tya
            asla                    ; 32 sectors/track
            asla                    ; shift 3 bits left
            asla
            stx     wcmdb+1         ; rotate right and
            lsr     wcmdb+1         ; shift in right 3 bits
            rora                    ; of track nr (max ff)
            lsr     wcmdb+1
            rora
            lsr     wcmdb+1
            rora
            adc     woff            ; offset (this partition)
            sta     wcmdb+1         ; **** sector nr absoluut
            tay
            lda     #0
            sta     wcmdb+2
            tya
            sbc     wmax+1
            bcs     werr
            jsr     cmdout
            rts
                    ;
cmdout      ldy     #5
            ldx     #0
            cmd1    lda             wcmdb,x
            jsr     outz80
            inx
            dey
            bne     cmd1
            rts
;
werr        sec
            rts
;
getz80
```

```
1       lda     z80bas+1
        bpl     1.b
        lda     z80bas          ; get data byte
        rts
;
outz80  pha
1       lda     z80bas+3
        bpl     1.b             ; wait until last byte accepted
        lda     z80bas+2        ; reset pbc flag
        pla
        sta     z80bas+2        ; send byte to z80
        rts
;
res     fill-*,$ff
        org     $e2ff
fill
;
        end     silicon
```



Winchester drives en de DOS65 computer

Het dos65 operating system houdt rekening met een totaal van vier drives. Er worden vier ram pagina's (256 bytes) gereserveerd voor de system sectors van de vier drives ($ad00-b0ff). Deze drives kunnen diskettedrives zijn, de dos65 diskette controller kaart kan er drie aansturen, maar ook een winchester drive of een ramdisk behoren tot de mogelijkheden. De exacte configuratie moet opgegeven worden bij het assembleren van het operating system (de file 'BOOT'). De dos65 versie zoals die standaard verspreid is, is geassembleerd voor max. 2 diskettedrives (5.25 of 3.5 inch) en een ramdisk. Bij versie 2.02 kan deze ramdisk zelfs 1 Mbyte groot zijn. Voor de ramdisk geldt dat de benodigde drivers al in het operating system ingebakken zitten. Ook wordt er een programma vformat meegeleverd, hiermee kan de gehele ramdisk geformatteerd worden in hetzelfde formaat als een floppy. Als er gebruik gemaakt wordt van dynamische rams moet dit elke keer gebeuren als de computer wordt aangezet, bij statische rams met battery backup hoeft dit maar eenmaal gedaan te worden, net als bij een echte floppy. Een ramdisk heeft natuurlijk veel voordelen, zoals snel laden van commando's, snel assembleren etc. maar daarnaast is een ramdisk ook veel kwetsbaarder. Een kleine spanningsstoring kan veel informatie onbruikbaar maken. Bovendien zijn ramchips nog zo duur dat bijv. een 1 Mbyte ramdisk nog aardig in de papieren loopt, zeker als er statische chips gebruikt worden.

Een alternatief is een winchester drive oftewel harddisk. Ook niet echt goedkoop maar minder kwetsbaar. Verder komen er steeds meer harddisk drives beschikbaar die niet meer voldoen aan de huidige standaard van 20 Mbytes en hoger die in de PC wereld geldt. De kleinere drives zijn echter bij uitstek geschikt voor een computer als de dos65 computer omdat de programma's vaak zo klein zijn dat er heel wat moeite nodig is om bijvoorbeeld 2.5 Mbyte vol te maken.

Drive types

Er zijn een paar standaards voor harddisk interfaces. Een van die standaards is de st506 interface. Deze door Shugart technology (nu Seagate) op de wereld gebrachte norm gaat uit van een 34 pin cable voor de control signalen en een 20 pin cable voor de data. Er zijn drives met zgn. 'edge' connectors, maar ook met standaard flatcable connectors. De power aansluiting is identiek aan die van 5.25 inch floppy drives, deze kan dus ook gewoon gebruikt worden, mits de max. stroom voldoende hoog is (sommige diskdrives gebruiken minder dan een floppydrive). Met de st506 kon niet snel gezocht worden. Om bijv. van track nul naar track 250 te gaan moesten er 250 pulsen van 3msec. gegeven worden. Elke puls veroorzaakte een stap naar een volgend spoor. Een nieuwere versie was de st412 interface standaard. Bij deze standaard wordt er van uit gegaan dat de 'seek' pulsen heel kort zijn (typisch tussen 12 en 200 usec). De pulsen worden geteld en er wordt door de drive in een keer gesprongen naar het juiste spoor. Verder zijn de st506 en st412 uitwisselbaar, de st412 kan ook gebruik maken van de 3msec seek pulsen.

SASI

Een st506/412 drive heeft een nogal ingewikkelde controller nodig. De data die serieel van de drive komt moet gedecodeerd worden, er wordt meestal gebruik gemaakt van MFM (net als bij dual density floppy's). Hiervoor heeft Western Digital bijv. een chip set op de markt gebracht (WD1010 etc.). Elke computer die wil communiceren met een harddisk met st506/412 interface kan hiertoe gebruik maken van die chipset. Elke fabrikant moet dan een controller board ontwerpen. De firma Xebec bracht een eenvoudiger oplossing. Waarom niet een controller die op de harddisk gemonteerd kan worden, met een zeer eenvoudige interface naar de computer toe, iets wat lijkt op een centronics interface? Deze SASI interface (SASI oorspronkelijk van Shugart) is eenvoudig met de computer te verbinden door middel

2	i/o	data0	tijdens een select (sel low)
4	i/o	data1	moet op de datalijnen het
6	i/o	data2	controller address staan.
8	i/o	data3	data 01 = address 1,
10	i/o	data4	data 02 = address 2
12	i/o	data5	data 04 = address 3
14	i/o	data6	data 08 = address 4 etc.
16	i/o	data7	
18-24		spare	----------
26		reserved	
28-34		spare	
36	in	-busy	SASI response to sel and address
38	out	-ack	computer made another byte available
40	out	-rst	reset the SASI controller
42	in	-msg	low if command completed
44	out	-sel	computer selects controller (address)
46	in	-c/d	SASI command (low) or data
48	in	-req	SASI requests another byte
50	in	-i/o	low is data from controller

Alle oneven nummers aan massa. Voedingsplug van Xebec 1410a identiek aan 5.25 inch floppy voedingsplug.

Tabel 1: SASI Cable layout (50-pin flatcable)

van een paar parallel poorten en een 50 polige flatcable.

Er kunnen maximaal 8 Xebec 1410A controllers worden aangesloten op dezelfde flatcable en maximaal twee (dezelfde) drives per controller. Ook zijn er drives die een ingebouwde SASI interface hebben zoals bijv. de 'OWL' drive (zie elektuur computing).

SCSI

De Amerikaanse 'ANSI', het instituut dat de officiële standaards maakt, doopte de SASI interface om naar SCSI en voegde er het een en ander aan toe. Zo kunnen er bijvoorbeeld ook tapedrives gebruik maken van de SCSI interface en werd de snelheid opgevoerd.

Als SASI cable maar met de volgende wijzigingen:

18	dbp data parity bit
26	+5 Volt
32	atn attention
25	reserved

Tabel 2: SCSI Cable layout (50-pin flatcable)

Er worden door diverse fabrikanten SCSI (spreek uit 'skoezie') drives op de markt gebracht. Een voordeel voor de interface ontwikkelaars, bijna al het werk gebeurt in de SCSI drive.

Oudere drives

Voor de hobbyist zijn de goedkoopste drives meestal het meest interessant. Deze drives hebben soms wat andere onhebbelijkheden die nog niet genoemd zijn. Zo is bijvoorbeeld de schrijf dichtheid aan de binnenzijde van de schijf veel hoger dan aan de buitenzijde omdat op de binnenste kleinere tracks net zoveel informatie moet staan als op de veel langere buitenste tracks. Omdat dit moeilijkheden gaf werken de oudere drives met een gereduceerde schrijfstroom vanaf een bepaald spoornummer. Ook is het mogelijk om de timing te wijzigen vanaf een bepaald track, de zgn. write precompensation. Om een voorbeeld te geven, de seagate st225, niet eens zo oud, heeft geen reduced write current nodig (die zetten we dus op 65535, zoveel tracks heeft de drive niet, dus wordt die optie ook niet gebruikt), maar wel write precompensation vanaf track 300 (die moet dus op 300 gezet worden , = 012C hex). Een ouder type, de seagate st412, vraagt om reduced write current vanaf track 128 en write precompensation vanaf track 128, dit wordt hex 0080-0080.

Embedded servo

Dan zijn er nog verschillen in de manier waarop de drive ziet hoe de head gepositioneerd staat. Sommige drives hebben een apart schijfoppervlak met eigen head, alleen maar voor de zgn. servo informatie. Deze informatie is nodig om bij een 'seek' de servo motor te kunnen sturen en daarmee de head op exact de juiste plaats terecht te laten komen. Veel drives hebben deze informatie 'verborgen' zitten tussen de data sectoren (zgn. embedded servo). Dit geeft aanmerkelijk meer ruimte voor data opslag omdat er een heel oppervlak vrijkomt. Voor de Xebec controller kan met een bit gekozen worden voor een van beide opties. Als vuistregel geldt dat een drive met een oneven aantal heads een aparte servohead heeft (daarom blijft er een oneven aantal heads over) en een drive met een even aantal heads embedded servo gebruikt.

ECC burst

Diskdrives maken soms fouten, er kan een bitje 'omgevallen' zijn bijvoorbeeld. Ook is het mogelijk dat een bit eenmaal fout gelezen wordt en de volgende keer weer correct naar buiten komt. Om datafouten te voorkomen zijn er verschillende methoden in gebruik. Ten eerste bevat de datasector een checksum, de data wordt opgeteld via een eenvoudig algoritme en de som wordt achter de data weggeschreven. Als tijdens het lezen blijkt dat de som niet klopt, dan is de data niet juist gelezen of geschreven. Er bevindt zich zowel een checksum achter de data als achter de sector informatie. Bij elke sector staat het sector en track nummer op de schijf geschreven. Als hier wat fout mee gaat, kun je meldingen verwachten als 'sector not found at ...'. Om te voorkomen dat er bij elk fout bit paniek in de elektronische tent is wordt eerst een aantal maal de leesoperatie herhaald (retry). Als de informatie dan nog steeds niet correct tevoorschijn komt, neemt de controller zijn toevlucht tot de zgn. ECC, een error correcting methode. Als de ECC burst lengte op 11 gezet wordt (normale waarde voor Xebec 1410A) dan betekent dit dat een grote fout waarbij 11 bits achterelkaar niet correct gelezen kunnen worden, nog verbeterd kan worden. Dit wordt bereikt door bij het schrijven extra informatie toe te voegen aan de bestaande data.

Verdere opties

Er is een trend om opslagcapaciteit zoveel mogelijk te vergroten zonder daarvoor meer platen in een drive te hoeven monteren. Een voorbeeld hiervan is de RLL encoding. Op de oude manier, met MFM encoding, is een Seagate st225 in staat om ca. 20 Mbyte op te slaan, met RLL kan een st238, praktisch dezelfde drive, tot ca. 32 Mbyte gevuld worden. Dit wordt bereikt door de plaats van de pulsjes op het spoor wat nauwkeuriger te definiëren. Hierdoor kan

de schrijfdichtheid weer wat groter worden waardoor je meer informatie kwijt kunt. Op een RLL controller kun je vaak nog wel een st225 gebruiken, er zijn echter ook al A(dvanced)RLL en E(xtended)RLL controllers die tot 1.9 x zoveel informatie op een drive persen. Dit stelt hogere eisen aan de drive en daarom zijn alleen speciaal daarvoor ontworpen drives geschikt. Ook is er een steeds doorgaande vraag naar hogere snelheden. Ook hier zijn mogelijkheden, zoals de ESDI interface. Drives, werkend met MFM, maken meestal gebruik van 16 of 17 sectoren per track (512 bytes) of 32 sectoren per track (256 bytes). Voor RLL combinaties geldt een aantal van 26 sectoren van 512 bytes en voor ESDI combinaties ben ik 34 sectoren tegengekomen. De Xebec 1410A controller werkt met 17 sectoren van 512 bytes of 32 sectoren van 256 bytes en altijd MFM.

Dos 65

In het operating system wordt rekening gehouden met een harddisk, maar er moet nog wel wat gebeuren voor deze drive ook werkelijk gebruikt kan worden. Ten eerste moet er een verbinding gemaakt worden met de harddisk. Voor een SASI drive is dat eigenlijk geen probleem. In de duitse 'elektuur computing special 6' werd een printplaat beschreven met een SASI interface (en een floppy interface), verkrijgbaar onder nummer EPS 86354-1. Of deze kaart nog verkrijgbaar is weet ik niet maar de interface is eenvoudig en daarom ook best zelf te maken. Een tweede oplossing is de SASI interface van Ad Brouwer, een eenvoudige schakeling met een 6821. Dan de software. Voor de interface van Ad Brouwer zijn de drivers beschikbaar. Deze drivers nemen twee pagina's in beslag (2 x 256 bytes) en moeten bijvoorbeeld door login.com geladen worden. Voor de elektuur interface vind je deze drivers in dit artikel. Door in mijn systeem de I/O een beetje op te schonen (zie artikel Z80 ramdisk) heb ik van $e200 t/m $e3ff precies twee pagina's ram beschikbaar gekregen die door geen enkel programma gebruikt worden. Hier worden mijn drivers geladen. De drivers blijken zelfs in een pagina te passen, bv. van $e500-e5ff. Verder heb je een dos65 versie nodig waarin rekening wordt gehouden met de winchester drivers. De drivers zien er als volgt uit: een schrijfroutine die een sector naar schijf stuurt en een leesroutine die een sector vanaf schijf haalt. Verder moet de SASI controller weten om wat voor drive het gaat (heads/tracks/precomp). Hiervoor is een kleine initialisatie routine nodig die alleen tijdens het opstarten gebruikt wordt. De eerste keer dat een harddisk gebruikt wordt zal deze geformatteerd moeten worden. Voor de elektuur interface is hiervoor door mij een 'wformat' programma geschreven dat met behulp van een configuratie file heel eenvoudig is aan te passen voor een ander disk drive type. Verder kun je met dit programma de drive testen met diverse bitpatronen en een dos65 systeemsector maken. De drive wordt dan geschikt gemaakt voor 2.5 Mbytes. Dit zijn ca 9999 blocks van 256 bytes. Om hoger te gaan moeten eerst wat grenzen verlegd worden in de diverse dos65 programma's. Een volgende keer meer over dit formatteer programma.

Ernst Elderenbosch.

```
; file                    st225.mac
;
; program       winches
;
; function      load winchester disk drivers
;
; usage                   winches
;
; by                      ernst elderenbosch
;
; original by   ad brouwer
;
; date                    04-jun-89       ernst elderenbosch system
;=============================================================

cpuclock equ    2                       ; 2 MHz 6502

worg            equ     $e200           ; drivers memory $e200-e3ff
```

```
wofs        equ     $0000       ; offset DOS65 directory (partition?)
wsiz        equ     160*128     ; size directory

memp        equ     $e8         ; dos65 r/w memory pointer

prtbyt      equ     $d038       ; dos65 print accu hex routine
prtext      equ     $d03b       ; dos65 print text routine

udc         equ     $e080       ; universal disk controller
rwdsasi     equ     udc         ; read write data from/to hd
rssasi      equ     udc+$10     ; read status from sasi
selsasi     equ     udc+$20     ; select sasi controller
resasi      equ     udc+$30     ; reset sasi controller
;
mask        equ     $d9         ; mask for sasi input
wcd         equ     $80
wreq        equ     $40
wmsg        equ     $10
wbsy        equ     $08
wio         equ     $01

werror      equ     $02         error mask

; winchester controller commands
ctstrdy     equ     $00         test drive ready
ctrck00     equ     $01         recalibrate
csense      equ     $03         request sense status
cread       equ     $08         read sector
cwrite      equ     $0a         write sector
cinitdsk equ $0c                initialize drive characteristics

            org     $200

winches     jmp     winit

wcmdbf      fcb     0,0,0,0,1,$07   command buffer (fast step)
;
; Seagate 225 specification (must be after wcmdbf)
wspec       fdb     615         cylinders
            fcb     4           heads
            fdb     $ff,$ff     reduced write current
            fdb     300         write precompensation
            fcb     11          maximum ECC data burst length
;
;
winit       sta     resasi      ; send reset pulse to controller
resdel      lda     #6          ; 450 ms delay
resdel1     jsr     del75       ;  75 ms delay routine
            sec                 ; decrement nr of delay times
            sbc     #1
            bne     resdel1     ; total delay not yet reached
;
testrdy     jsr     selcntr     ; select the controller
            lda     #ctstrdy    ; test drive ready command ($0)
            jsr     taskout     ; send command to controller
            jsr     getstat     ; get the controller status
```

```
                beq     inidisk
                jmp     error           ; error, drive not ready
;
inidisk         jsr     selcntr         ; select the controller
                lda     #cinitdsk       ; initialize drive characteristics cmd ($c)
                jsr     taskout         ; send command to the controller
                ldx     #0              ; init. charact. byte counter
inidsk1         jsr     reqwait         ; wait for cntlr request
                bit     rssasi
                bmi     inidsk2         ; if command request, xfer completed
                lda     wspec,x         ; get data byte
                sta     rwdsasi         ; send it to the controller
                inx                     ; bump the buffer pointer
                bra     inidsk1         ; more to do
inidsk2         jsr     getstat         ; get completion status
                beq     recal           ; test for error completion
                jmp     error           ; error
;
recal           jsr     selcntr         ; select the controller
                lda     #ctrck00        ; recalibrate command code ($1)
                jsr     taskout         ; send command to controller
                jsr     getstat         ; get completion status
                bne     1.f             ; branch on error
                rts
1               jmp     error
;
taskout         cmp     #2              ; check for forbidden code
                beq     9.f
                sta     wcmdb           ; store command in table
                jmp     cmdout          ; standard routine
9               jmp     error           ; error
;
del75                                   ; 75 msec delay routine
                if      cpuclock = = 4
                ldx     #$a5
                ldy     #$ea
                elseif  cpuclock = = 2
                ldx     #$52
                ldy     #$75
                elseif  cpuclock = = 1
                ldx     #$a9
                ldy     #$3a
                endif
del751          dex
                bne     del751          ; lsb zero? = = decrement msb
                dey
                bne     del751          ; msb zero? = = ready
                rts
;
error           jsr     prtext
                fcc     '\rHarddisk error ',0
                jsr     sense
                fcc     '\r\r',0
                rts
;
sense           jsr     selcntr         ; select the controller
```

```
                lda     #csense         ; request sense status command ($3)
                jsr     taskout
                jsr     del75
                lda     rwdsasi         ; get sense byte
                sta     sense0
                and     #$3f            ; extract error code
                jsr     prtbyt
                jsr     prtext
                fcc     ' sense bytes: ',0
                lda     rwdsasi         ; get sense byte
                sta     sense0+1
                jsr     prtbyt
                lda     rwdsasi         ; get sense byte
                sta     sense0+2
                jsr     prtbyt
                lda     rwdsasi         ; get sense byte
                sta     sense0+3
                jsr     prtbyt
;
sense0          res     4,0             ; buffer for sense info
;
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;               winchester driver (read write sector)
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

                org     worg

winread         jmp     wread
winwrite jmp    wwrite
woff            fdb     wofs            offset address
wmax            fdb     wofs+wsiz       max address

wcmdb           fcb     0,0,0,0,1,$18   command buffer (fast step)

ytemp           fcb     0

;read sector from winchester drive

wread           lda     #cread          ; get read sector command
                jsr     wrwsec          ; start read sector procedure
                bcs     9.f             ; carry set = error
                ldy     #0
read1           jsr     reqwait
                bit     rssasi
                bmi     read2
                lda     rwdsasi
                sta     [memp],y
                iny
                bra     read1
read2           jsr     getstat
                bne     4.f
                rts
;
;write sector to winchester drive
;
wwrite          lda     #cwrite         ; get write sector command
```

```
                jsr     wrwsec      ; start write sector procedure
                bcs     9.f         ; carry set = error
                ldy     #0
write1          jsr     reqwait
                bit     rssasi
                bmi     write2
                lda     [memp],y
                sta     rwdsasi
                iny
                bra     write1
write2          jsr     getstat
                bne     4.f
                rts
;
4               jsr     wrqs
                ldy     rwdsasi     result
                jsr     wrqs
                lda     rwdsasi
                tya                 error?
                beq     9.f
;
werr            sec                 set carry
9               ldy     ytemp
                rts
;
getstat         jsr     reqwait     ; wait for request
                lda     rwdsasi     ; get status
                jsr     reqwait     ; wait for second byte
                ldx     rwdsasi
                and     #2          ; isolate the error bit (%0000 0010)
                rts
;
reqwait         bit     rssasi      ; check request bit (%0100 0000)
                bvc     reqwait     ; loop until request bit set
                rts
;
selcntr         jsr     waitrdy     ; wait for ready
                lda     #1          ; controller select code
                sta     rwdsasi     ; send to transparent latch
                sta     selsasi     ; generate a select strobe
waitbsy         lda     rssasi      ; read sasi status port
                and     #wbsy       ; mask busy status bit (%0000 1000)
                beq     waitbsy     ; loop if ready
                rts
waitrdy         lda     rssasi      ; read sasi status port
                and     #wbsy       ; mask busy status bit (%0000 1000)
                bne     waitrdy     ; loop if busy
                rts
;
;
wrwsec          sty     ytemp
                sta     wcmdb       ; command (r/w)
                dey                 ; calculate address
                cpy     #128        ;  128 sectors/track
                bcs     werr        ;   1 logical track
                tya                 ; equals 4 physical tracks
```

```
                asla
                stx         wcmdb+2         ; rotate right and
                lsr         wcmdb+2         ; shift in right 3 bits
                rora                        ; of track nr (max ff)
                adc         woff            ; offset (this partition)
                sta         wcmdb+3
                tay
                lda         wcmdb+2
                adc         woff+1
                sta         wcmdb+2
                bcs         werr
                cpy         wmax            check maximum address
                sbc         wmax+1
                bcs         werr

; start command

                jsr         selcntr         ; select controller
                jsr         cmdout          ; command to controller
                rts
;
cmdout          ldy         #6
                jsr         reqwait
                ldx         #0
cmd1            lda         wcmdb,x
                sta         rwdsasi
                inx
                dey
                bne         cmd1
                rts
;
wrqs            bit         rssasi          ; check request bit (%0100 0000)
                bvc         wrqs            ; loop until set
                rts
;
                res         fill-*,$ff
;
                org         $e3ff
;
fill
;
                end         winches
```

DOS65 Geluidsprint

De computersystemen die heden ten dage door hobbyisten worden gebruikt zijn voorzien van een mogelijkheid tot het maken van muziek. Voor de DOS65 is nu een geluidsprint ontwikkeld om een "af" systeem te krijgen. Deze geluidsprint kan zonder meer aangesloten worden op de uitbreidingsbus van de DOS65 computer. Het enige dat verder nog nodig is, is een stuk software dat de geluidsprint ondersteunt, b.v. in de vorm van het BASICODE-3 vertaalprogramma.

Het geheel bestaat uit een geluids-IC, de Sound Interface Device (SID) 6581 (wel bekend bij Commodore gebruikers), een tweetal TTL poorten, een Opamp en nog wat componenten.

Even een opsomming van wat de sound controller 6581 in huis heeft:

- 3 programmeerbare, onafhankelijke oscillatoren.
- 4 mengbare golfsoorten per stem.
- 3 mengbare filters (hoog-, laag-, en bandfilter).
- Per stem een envelope generator.
- 2 te koppelen ringmodulatoren.
- Vervormingsmogelijkheid voor externe signaalbronnen.
- 2 8-bits analoog-digitaal omzetters.

M.b.v. de Opamp is het mogelijk om het signaal dat de SID produceert te versterken om het vervolgens aan een luidspreker aan te bieden. Indien de monitor een versterker heeft, is het niet nodig de schakeling rond deze Opamp te bouwen.

De twee TTL poorten zijn nodig voor adresdecodering t.b.v. de SID. Op de NOR poort worden de adreslijnen A12, A11, A10 en A8 aangeboden welke allen laag moeten zijn om een hoog aan de uitgang te krijgen en op de NAND poort worden de adreslijnen A15, A14, A13, A9, A7, A6 en A5 aangeboden welke hoog moeten zijn, waardoor uiteindelijk op de adressen $E2E0-$E22FF het CS signaal actief kan zijn. Via de adreslijnen A0-A4 worden de registers in de SID geadresseerd, mits de SID met CS geselecteerd is.

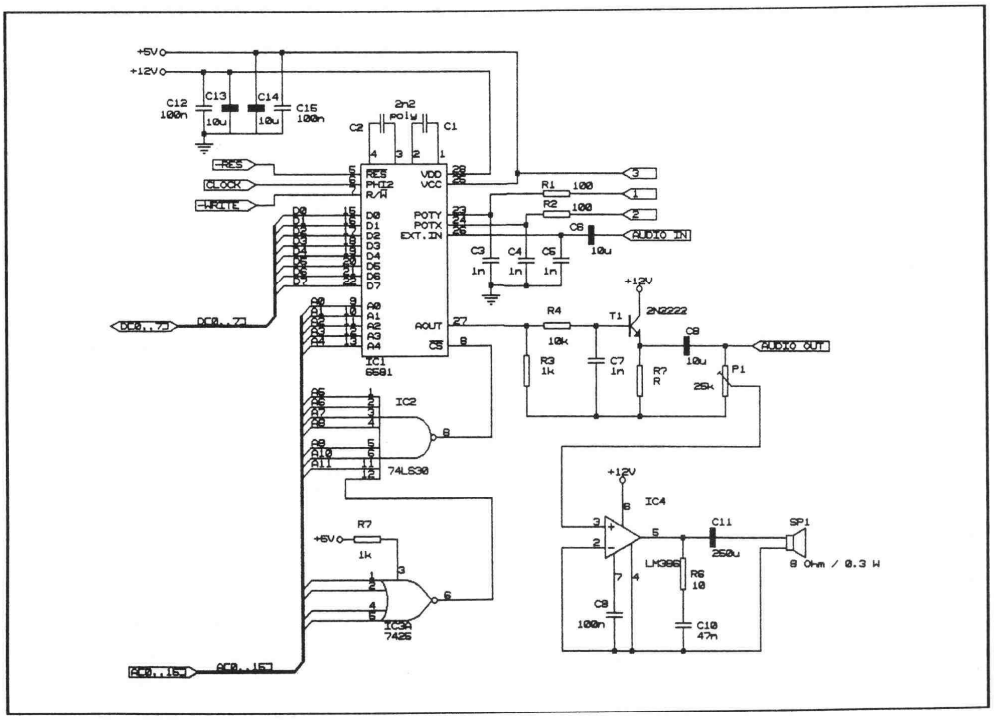


Fig. 1: Schakeling rond de SID-6581 t.b.v. DOS-65

Diskcopy voor DOS-65

Met behulp van het onderstaande programma is het mogelijk om een complete schijf, inclusief de directory-structuur naar een andere schijf te kopieren.

Frank Bens

```
; File :        DCOPY.MAC
;
;             Written by Frank Bens for DOS65
;
;             Creating date: 14-03-90
;
              opt     nogen

waitfl        equ     $d9             ; wait flag
track         equ     $da             ; track number
trkcopy       equ     $db             ; copy of track number
trkmax        equ     $dc             ; max number of tracks
sector        equ     $dd             ; sector number
seccopy       equ     $de             ; copy of sector number
secmax        equ     $df             ; max number of sectors
rwpoin        equ     $e8             ; read/write pointer
srcdrv        equ     $f0             ; source drive number
tardrv        equ     $f2             ; target drive number

datablk       equ     $0200           ; start address data block
sisblk        equ     $9f00           ; System Info Sector (256 bytes)

input         equ     $c020           ; get a character
prchar        equ     $c023           ; print a character
inecho        equ     $c026           ; get and print a character
crlf          equ     $c02f           ; print a cr & lf
hnout         equ     $c035           ; print a hex-nibble
hexout        equ     $c038           ; print a hex-byte
prtext        equ     $c03b           ; print a string until $00
sopt          equ     $c068           ; scan options
spar          equ     $c06b           ; scan parameters
ermes         equ     $d0b7           ; print error message
readsec       equ     $d0c0           ; read a sector
wrchsec       equ     $d0c9           ; write & verify a sector
posit         equ     $f024           ; position the cursor


;**** Macro definitions ****

poscur        macro   xpos,ypos
              ldx     #xpos
              ldy     #ypos
              jsr     posit
              endm
```

```
print       macro   $text
            jsr     prtext
            fcc     $text,0
            endm

            org     $a000

            fcc     $c8,$c5,$cc,$d0,'xxx'
            fcc     'Function : Copying an Entire Disc.\r'
            fcc     'Syntax  : DCOPY [-W] sourcedrive,targetdrive\r'
            fcc     'Options  : -W : Wait to allow change of discettes.\r'
            fcc     'Abbreviation  : None.\r'
            fcc     'Example  : DCOPY 1,0  copy disc from drive 1: to drive 0:\r'
            fcc     'Note : The source and target discs must'
            fcc     ' be the same size discs.',0

er          jmp     error               ; error exit

dcopy       print   '\f\t\t\t\t*** DISKCOPY ***'

            jsr     sopt                ; check option
            fcc     'W',0               ; option is W
            stx     waitfl              ; save option
            bcs     er                  ; branch on option error
            ldx     #-1                 ; preset parameters
            stx     srcdrv
            stx     tardrv
            ldx     #%10100000
            jsr     spar                ; get parameters
            fcc     srcdrv,tardrv,0     ; 2 parms
            bcs     er                  ; branch on error
            lda     srcdrv              ; get source drive number
            ora     tardrv              ; 'or' with target drive number
            bpl     waityn
            poscur  1,3                 ; position the cursor
            print   'Which drive contain the source disc? : '
            jsr     inecho              ; wait for input
            and     #3                  ; mask only 2 LSB's
            sta     srcdrv              ; and save it
            print   '\rWhich drive contain the target disc? : '
            jsr     inecho
            and     #3
            sta     tardrv

waityn      poscur  1,6                 ; position the cursor
            ldx     #-1
            lda     tardrv              ; get target drive number
            cmp     srcdrv              ; check target with source drive
            beq     setwait             ; branch if equal
            lda     waitfl              ; get wait flag
            beq     cstart              ; no wait statement ?
            print   'Hit a key when ready...'
            jsr     input               ; else wait...
            inx                         ; dont wait any more
setwait     stx     waitfl
```

```
cstart      print   '\r\rCopy started...\r'
            poscur  9,14
            print   'track: 00  sector: 01'
            ldy     #0              ; track 0 / sector 1
            ldx     #0
            stx     track


;***** read track/sector ******


readsrc     stx     trkcopy
            iny
            sty     sector          ; save sector
            sty     seccopy
            poscur  1,12
            ldx     waitfl          ; wait ??
            beq     prread          ; No
            print   'Insert \Eisource\En disc and hit a key when ready...'
            jsr     input

prread      print   '\r\rReading'
            jsr     filpoin         ; set pointer of sector in memory
            ldy     sector          ; restore sector

datain      jsr     prtrsec         ; print track & sector number
            lda     srcdrv          ; get drive number
            jsr     readsec         ; read data sector
            bcc     getbuf          ; branch if no read error
            jmp     error           ; error exit

getbuf      lda     track
            bne     1.f
            lda     sector
            cmp     #1
            bne     1.f             ; only in track 0 / sector 1
            lda     datablk+$21     ; get max number of tracks
            sta     trkmax          ; and save it
            lda     datablk+$22     ; get max number of sectors
            sta     secmax          ; and save it

1           inc     rwpoin+1        ; next address in data block
            lda     rwpoin+1        ; get current address
            cmp     #$a0            ; end of data block
            beq     coptar
            ldy     sector          ; get sector number
            cpy     secmax          ; all sectors scanned
            bne     2.f             ; branch if not
            ldy     #0              ; set first sector
            inc     track           ; next track

2           iny
            lda     track           ; get track number
            cmp     trkmax          ; all tracks/sectors copied ?
            bne     datain          ; branch if not completed
```

```
;****** write and check sector *******

coptar      lda     trkcopy
            ldy     seccopy
            sta     track
            sty     sector
            poscur  1,12
            ldx     waitfl              ; wait ??
            beq     rdsisbl             ; No
            print   'Insert \Eitarget\En disc and hit a key when ready...'
            jsr     input

rdsisbl     lda     track
            bne     prwrite             ; branch if track =/0
            lda     #sisblk&255         ; set read pointer to system block
            ldy     #sisblk>8
            sta     rwpoin
            sty     rwpoin+1
            lda     tardrv              ; get target drive number
            ldx     #0                  ; set track number
            ldy     #1                  ; set sector number
            jsr     readsec             ; read system sector
            bcc     gettrsc             ; branch if no read error
            jmp     error               ; error exit

notequ      poscur  1,10
            print   'The target disc must be the same size as the source disc.'
            jmp     exit

gettrsc     lda     sisblk+$21          ; get number of tracks
            cmp     trkmax              ; check with max of track numbers
            bne     notequ              ; branch if not equal
            lda     sisblk+$22          ; get number of sectors
            cmp     secmax              ; check with max of sector numbers
            bne     notequ              ; branch if not equal

prwrite     print   '\r\rWriting'
            jsr     filpoin             ; set pointer of memory in sector
            ldy     sector              ; restore sector

dataout     jsr     prtrsec             ; print track & sector number
            lda     tardrv              ; get drive number
            jsr     wrchsec             ; write & check sector
            bcs     error               ; branch on error
            inc     rwpoin+1            ; next address in data block
            lda     rwpoin+1            ; get current address
            cmp     #$a0                ; end of data block
            beq     2.f
            cpy     secmax              ; all sectors scanned
            bne     1.f                 ; branch if not
            ldy     #0                  ; set first sector
            inx                         ; next track
            stx     track
            cpx     trkmax              ; all tracks/sectors copied ?
            beq     progend             ; branch if so
```

```
1               iny
                bne     dataout         ; always

2               cpy     secmax          ; all sectors scanned
                bne     3.f
                ldy     #0              ; set first sector
                inx                     ; next track
                stx     track

3               jmp     readsrc         ; continue

progend         poscur  1,10
                print   'Copy completed.'
exit            poscur  1,16
                rts                     ; program exit

error           poscur  1,15
                sec
                jmp     ermes           ; print an error

;*************************
;                       *
;   Subroutines         *
;                       *
;*************************
prtrsec         sty     sector          ; save sector number
                tya
                poscur  28,14           ; posit cursor for sector number
                jsr     hexout          ; and print it
                poscur  16,14           ; posit cursor for track number
                ldy     sector          ; get sector number
                ldx     track           ; get track number
                txa
                jmp     hexout          ; and print it

filpoin         lda     #datablk&255    ; set read/write pointer
                sta     rwpoin
                lda     #datablk>8
                sta     rwpoin+1
                rts

                end     dcopy
```

8 bits microcontrollers van intel programmeren via DOS65 en AS

Je realiseert je het niet zo snel in deze tijd waarin een 7400 al een computer heet in de ogen van de reclamemensen maar er worden werkelijk 'overal' computers ingebouwd. Neem nou die hele oude Philips videorecorders (2020 ed.) of veel moderne en zelfs iets minder moderne toetsenborden, diastuurapparaten, huishoudelijke apparatuur enz. enz. Vaak gaat het dan niet om een 80486 met 16 Mbytes maar dat is voor dergelijke besturingsfuncties ook niet nodig. In deze situaties is een compacte microprocessor op zijn plaats, liefst met wat ram en ROM en parallel I/O aan boord, met een directe aansluiting voor een kristal als clock. Aan al deze voorwaarden voldoen de microcontrollers van intel. Er zijn overigens niet alleen 8 bit versies maar dit zijn nog wel de meest voorkomende, zeker in de dump. Intel begon met de 8041, 1k ROM en 64 bytes ram aan boord en 18 I/O lijnen. Hierna kwam de 8048 en dit is een werkelijk veel voorkomend IC. Er bestaat een versie van met eprom (8748), meestal met een venster zodat de eprom gewist kan worden maar soms ook in een plastic uitvoering zonder venster, eenmalig te programmeren door de gebruiker. Dan is er ook een versie zonder ROM, te gebruiken met een externe eprom (de 8035). Iets later kwam een iets uitgebreidere versie op de markt (8049/8749/8039) met weer iets meer ROM en ram aan boord maar dezelfde instructieset. De 8048 werkt standaard met een Xtal van 6MHz, de 8049 kan zelfs met een Xtal van 11MHz aan de slag. Wat kan zo'n ding? Heel veel. Eigenlijk is er niet zoveel verschil met een echte microprocessor. Alleen is een microcontroller meer gericht op het bekijken van signalen, testen en dan bijvoorbeeld conditioneel springen in plaats van de brute rekenkracht die je in de grotere broers vindt. Als je dus iets wilt ontwerpen waarbij het aankomt op testen en controleren, dan loont het de moeite om even in de dumpbak te kijken. Ik haalde pas nog een 8748 uit een gesloopte 5Mb winchester drive, je weet wel, zo'n hoge. Een duits elektronica bedrijf adverteerde enige tijd terug met pakketjes van 5 x 8049 voor drie mark. Daar zit dan interne (geprogrammeerde) ROM in maar daar heeft intel het volgende op gevonden. Er bevindt zich op die controllers met ingebouwde ROM een input EA (external access). Als je deze ingang hoog maakt dan schakel je de interne ROM uit en zoekt de controller z'n instructies buitenshuis dwz. uit een eprom. De controllers hebben meerdere I/O lijnen (27 voor de 8048/49) waarvan een gedeelte als databus en adresbus te gebruiken is. Zo kun je externe chips aansluiten. Er is ook een speciale I/O chip voor op de markt gebracht, de intel 8043. Zo kun je dan het aantal I/O lijnen eenvoudig uitbreiden, maar dat kan natuurlijk ook op andere manieren. Om het programmeren van de 8048 familie wat eenvoudiger te maken heb ik een macro file gemaakt voor de assembler van DOS65, waarin je alle instructies kunt vinden die in de instructieset van de 8048 voorkomen. Als je hiermee een programma schrijft en deze macro file in je programma oproept mbv. 'lib i8048', dan wordt het geheel keurig netjes in 8048 machine code vertaald. De 8049 heeft exact dezelfde instructieset dus ook hiervoor kun je dezelfde file gebruiken. Wie met de modernere 8051 of 8052 aan de gang wil kan gerust uitgaan van deze macro listing maar zal er het een en ander aan moeten toevoegen. Per slot van rekening zijn deze controllers weer een stukje uitgebreider (RAM, ROM, I/O, timers, en zelfs seriële in- en output). Ik hoop dat deze macrolisting van nut kan zijn. Het demonstreert evenals de door mij ooit eens gepubliceerde 8080 macro's en de door iemand anders in de club geschreven 65C816 macro's dat de DOS65 macroassembler een zeer veelzijdige assembler is en zeker voor 'thuis' projecten een waardevol stuk gereedschap. Dit artikel werd trouwens geschreven op een DOS65 computer met daarin een.....8039 met 6MHz Xtal en een 2716.

Alleen is een microcontroller meer gericht op het bekijken van signalen, testen en dan bijvoorbeeld conditioneel springen in plaats van de brute rekenkracht die je in de grotere broers vindt

Ernst Elderenbosch

```
;filei8048.mac
                    opt     nolis
;
;                   macro file for 8048/35/49/39/8748/49 code assembly
;
;                   all devices with internal rom can be forced to use
;                   external rom by applying +5v to ea (pin 7). this way
;                   you can use an old microcontroller from a keyboard
;                   or a videorecorder or whatever for a new project!
;
;                   intel 8 bit micro controller family
;
;                   rom     eprom           extern          romrami/otimerclck
;
;                   8041    8741            1k              641816MHz
;                   8048    8748            8035            1k642716MHz
;                   8049    8749            8039            2k12827111MHz
;                   8051    8751            8031            4k12832212MHz
;                   8052    8752            8032            8k25632312MHz
;
;
;
;                   intel 8048/8748/8035 code macro's
;
;
;                   registers 0-7 have values 0-7
;
;
;                   accumulator instructions
;
iadd                macro   byt             add a, #data
                    fcc     $03,byt
                    endm
;
iadr                macro   reg             add a, reg
                    fcc     $68|reg
                    endm
;
iadri               macro   reg             add a, [reg 0/1]
                    fcc     $60|reg
                    endm
;
iaddc               macro   byt             addc a, #data
                    fcc     $13,byt
                    endm
;
iadcr               macro   reg             addc a, reg
                    fcc     $78|reg
                    endm
;
iadcri              macro   reg             addc a, [reg 0/1]
                    fcc     $70|reg
                    endm
;
```

```
ianl    macro   byt         anl a, #data
        fcc     $53,byt
        endm
;
ianr    macro   reg         anl a, reg
        fcc     $58|reg
        endm
;
ianri   macro   reg         anl a, [reg 0/1]
        fcc     $50|reg
        endm
;
icpl    macro   cpl a
        fcc     $37
        endm
;
iclr    macro   clr a
        fcc     $27
        endm
;
idaa    macro   da a
        fcc     $57
        endm
;
idec    macro   dec a
        fcc     $07
        endm
;
iinc    macro   inc a
        fcc     $17
        endm
;
iorl    macro   byt         orl a, #data
        fcc     $43,byt
        endm
;
iorr    macro   reg         orl a, reg
        fcc     $48|reg
        endm
;
iorri   macro   reg         orl a, [reg 0/1]
        fcc     $40|reg
        endm
;
irl     macro   rl a
        fcc     $e7
        endm
;
irlc    macro   rlc a
        fcc     $f7
        endm
;
irr     macro   rr a
        fcc     $77
        endm
;
irrc    macro   rrc a
```

```
                fcc     $67
                endm
;
iswap           macro   swap a
                fcc     $47
                endm
;
ixrl            macro   byt             xrl a, #data
                fcc     $d3,byt
                endm
;
ixrr            macro   reg             xrl a, reg
                fcc     $d8|reg
                endm
;
ixrri           macro   reg             xrl a, [reg 0/1]
                fcc     $d0|reg
                endm
;
;               branch instructions
;
idjnz           macro   reg,addr        dnjz reg,addr
                fcc     $e8|reg,addr
                endm
;
ijbb            macro   bit,addr        jbb addr
                fcc     $12|bit<5,addr
                endm
;
ijc             macro   addr            jc addr
                fcc     $f6,addr
                endm
;
ijf0            macro   addr            jf0 addr
                fcc     $b6,addr
                endm
;
ijf1            macro   addr            jf1 addr
                fcc     $76,addr
                endm
;
ijmp            macro   addrh,addrl     jmp addr (2k)
                fcc     $04|addrh<5,addrl
                endm
;
ijmpp           macro   jmp [a] (within current page)
                fcc     $b3
                endm
;
ijnc            macro   addr            jnc addr
                fcc     $e6,addr
                endm
;
ijni            macro   addr            jni addr
                fcc     $86,addr
                endm
```

```
;
ijnt0       macro   addr            jnt0 addr
            fcc     $26,addr
            endm
;
ijnt1       macro   addr            jnt1 addr
            fcc     $46,addr
            endm
;
ijnz        macro   addr            jnz addr
            fcc     $96,addr
            endm
;
ijtf        macro   addr            jtf addr
            fcc     $16,addr
            endm
;
ijt0        macro   addr            jt0 addr
            fcc     $36,addr
            endm
;
ijt1        macro   addr            jt1 addr
            fcc     $56,addr
            endm
;
ijz         macro   addr            jz addr
            fcc     $c6,addr
            endm
;
;           control instructions
;
ieni        macro   eni
            fcc     $05
            endm
;
idisi       macro   disi
            fcc     $15
            endm
;
ient0c      macro   ent0clk
            fcc     $75
            endm
;
ismb0       macro   selmb0 (select first 2k prog mem)
            fcc     $e5
            endm
;
ismb1       macro   selmb1 (select sec. 2k prog mem)
            fcc     $f5
            endm
;
isrb0       macro   selrb0 (select reg at data addr 0-7)
            fcc     $c5
            endm
;
isrb1       macro   selrb1 (select reg at data addr 24-31)
            fcc     $d5
```

```
                endm
;
;               the following instruction is for 80c48 only
;               = = = = = = = = = = = = =
;
ihalt           macro   halt
                fcc     $01
                endm
;
;               data move instructions
;
ima             macro   data            mov a, #data
                fcc     $23,data
                endm
;
imar            macro   reg             mov a, reg
                fcc     $f8,reg
                endm
;
imari           macro   reg             mov a, [reg 0/1]
                fcc     $f0|reg
                endm
;
imap            macro   mov a, psw
                fcc     $c7
                endm
;
imr             macro   reg,data        mov reg, #data
                fcc     $b8|reg,data
                endm
;
imra            macro   reg             mov reg, a
                fcc     $a8|reg
                endm
;
imria           macro   reg             mov [reg 0/1], a
                fcc     $a0|reg
                endm
;
imri            macro   reg,data        mov [reg 0/1], data
                fcc     $b0|reg,data
                endm
;
impa            macro   mov psw, a
                fcc     $b7
                endm
;
imaia           macro   mov a, [a] (current page)
                fcc     $a3
                endm
;
imai3a          macro   mov a, [$300 + a] (page 3)
                fcc     $e3
                endm
;
imxari          macro   reg             mov a, ext[reg 0/1]
                fcc     $80|reg
```

```
                endm
;
imxria          macro   reg             mov ext[reg 0/1], a
                fcc     $90|reg
                endm
;
ixar            macro   reg             xch a, reg
                fcc     $28|reg
                endm
;
ixari           macro   reg             xch a, [reg 0/1]
                fcc     $20|reg
                endm
;
ixdari          macro   reg             xch a, [reg 0/1] (lower nibble only)
                fcc     $30|reg
                endm
;
;               flag instructions
;
icpc            macro   cpl c
                fcc     $a7
                endm
;
icpf0           macro   cpl f0
                fcc     $95
                endm
;
icpf1           macro   cpl f1
                fcc     $b5
                endm
;
iclc            macro   clr c
                fcc     $97
                endm
;
iclf0           macro   clr f0
                fcc     $85
                endm
;
iclf1           macro   clr f1
                fcc     $a5
                endm
;
;               input / ouput instructions
;
ianb            macro   data            anl bus, #data
                fcc     $98,data
                endm
;
ianp            macro   port,data       anl port 1-2, #data
                fcc     $98|port,data
                endm
;
iand            macro   port            anld port (bit 4-7), a (bit 0-3)
                fcc     $9c|port
                endm
```

```
;
iinp        macro   port              in a, port 1-2
            fcc     $08|port
            endm
;
iinsb       macro   ins a, bus
            fcc     $08
            endm
;
imdap       macro   port              movd a (bit 0-3), port (bit 4-7)
            fcc     $0c|port
            endm
;
imdpa       macro   port              movd port (bit 4-7), a (bit 0-3)
            fcc     $3c|port
            endm
;
iorb        macro   data              orl bus, #data
            fcc     $88,data
            endm
;
iordp       macro   port              orld port (bit 4-7), a (bit 0-3)
            fcc     $8c|port
            endm
;
iorp        macro   port,data         orl port 1-2, #data
            fcc     $88|port,data
            endm
;
ioutb       macro   out bus, a
            fcc     $02
            endm
;
ioutp       macro   port              out port 1-2, a
            fcc     $38|port
            endm
;
;           register instructions
;
idecr       macro   reg               dec reg
            fcc     $c8|reg
            endm
;
iincr       macro   reg               inc reg
            fcc     $18|reg
            endm
;
iincri      macro   reg               inc [reg 0-1]
            fcc     $10|reg
            endm
;
;           subroutine instructions
;
icall       macro   addrh,addrl       call addr (2k)
            fcc     $14|addrh<5,addrl
            endm
```

Een printerbuffer en een toetsenbordbuffer voor DOS65

Al tijden lang maak ik gebruik van mijn goeie ouwe printer, een Diablo hitype II met eigengemaakte interface die ik vlak na het uitkomen van de Junior computer gekocht heb. Dat ding leek onverwoestbaar maar een paar maanden terug was het opeens zover, er kwamen nog wat tekens uit maar het leek in het geheel niet meer op de tekst die er in ging. Van een kompleet vel A4 bleef ongeveer een halve regel haakjes sluiten over en verder moest ik het maar zelf invullen. Als je je printer tamelijk vaak gebruikt, bijvoorbeeld ook als typemachine, is dat een regelrechte ramp. Ik organiseerde een andere printer (te leen voor een paar dagen) en ging vrolijk weer aan het werk. Hier kwam echter helemaal niets op papier, wat ik ook probeerde. Na wat zoekwerk bleek dat de Centronics printer de parallel signalen zo sterk belastte dat de 6522 in de DOS65 (= octopus) gewoon geen fatsoenlijk signaal meer afgaf. Een oplossing bleek niet zo moeilijk en is eventueel best in te bouwen in de computer: twee IC's 74LS367 tussen de uitgangen van de 6522 (ook de strobe) en de printer ingangen. Met als resultaat dat elke parallel printer voortaan te gebruiken is. In plaats van de 74LS367 kan natuurlijk ook een andere driver gebruikt worden, zoals de 74LS244.

Een tweede probleem doet zich af en toe voor als ik enthousiast verder tik terwijl mijn computer aan het assembleren is of iets anders doet dat veel interrupt afhandeling vergt. Voor elke toets die ik indruk moet de 6502 een interrupt afhandelen. Als de disk nogal intensief gebruikt wordt gaan deze interrupts voor (zo hoort het ook) maar tijdens het afhandelen van een diskinterrupt kan ik best twee toetsen ingetikt hebben (daar hoef je ook weer niet zo snel voor te tikken). De eerste toets wordt vergeten want voordat de keyboard interrupt wordt afgehandeld staat de tweede toets alweer klaar. Nu zijn er verschillende oplossingen mogelijk. Ten eerste zou ik niet meer zo ongeduldig kunnen zijn en rustig kunnen wachten tot de computer er weer klaar voor is, of in ieder geval zo langzaam kunnen typen dat alle interrupts er door komen. Dit begon me mateloos te irriteren. Een tweede mogelijkheid is om er wat aan te doen. Dat leek me wel wat. De mogelijkheid om dan maar op een andere computer verder te gaan heb ik niet serieus overwogen omdat ik toch nog steeds zeer tevreden ben met DOS65. Nu zijn er van die mooie kleine geheugen IC's die FIFO's genoemd worden (AMD 2841 of 3341). Deze 2841 bestaat uit 64 x 4 bits, voor het verwerken van bytes heb je er dus twee nodig. Het leuke van deze mini geheugenchips is dat je vier bits klaarzet en als je dan een strobepuls geeft dan begint er een hele operatie in de fifo waardoor die vier bits door het hele geheugen heenvallen en op de laatste positie terecht komen, bij de uitgang. Verder geeft de uitgang een signaal af om te kennen te geven dat er informatie klaar staat. Als je nu weer data aanvoert en een strobe puls geeft, zal dit tweede byte doorstomen naar de voorlaatste plaats en zo voort, tot alle 64 plaatsen gevuld zijn. Als je een acknowledge geeft aan de uitgang om aan te geven dat je een 'nibble' hebt opgepakt, dan schuift de hele zaak een plaats verder en zal er weer een strobe signaal komen, mits er tenminste nog een nibble aanwezig is. Dit IC leek me ideaal om mijn probleem op te lossen, temeer omdat er helemaal geen clock of zo nodig is, het IC regelt alles zelf. Het enige probleem was nog dat DOS65 geen acknowledge stuurt naar het toetsenbord, maar dit bleek helemaal geen probleem te zijn. Door het veranderen van twee bytes in de eprom (I/O65) wordt de 6522 gevraagd om een automatische handshake te doen. Als je die handshake niet gebruikt heeft I/O65 er ook geen last van dus je kunt altijd terug zonder wisselen van eproms. De volgende lokaties in eprom zijn bij mij aangepast (I/O65 2.02 of 2.01):
F0DFwas 0E, wordt 0A en F0E1was FE, wordt FA.

De schakeling is zo ontworpen dat je het toetsenbord kunt lostrekken en via de schakeling weer op dezelfde manier kunt aansluiten. Je zou het geheel in een klein kastje tussen toetsenbord en computer kunnen hangen hoewel het mooier is om het in de computer in te bouwen. Mochten er problemen zijn met de verkrijgbaarheid van de AMD 2841 laat het dan even weten. Succes,

Ernst Elderenbosch.

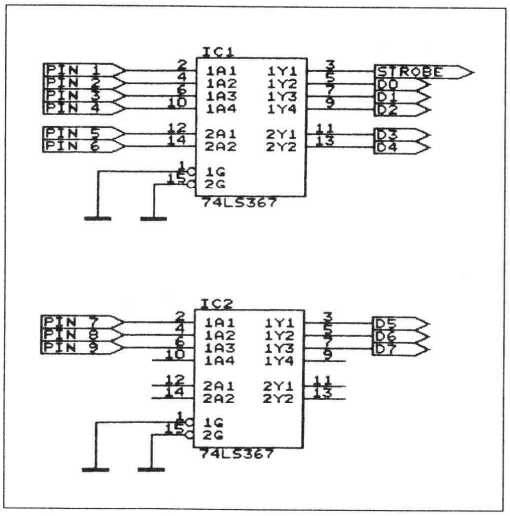


Fig. 1: printerbuffer

SUFAST t.b.v. de real time clock MC146818

Dit programma dient om de real time clock te gebruiken in Dos65, net als de bestaande utility SETUP.

SETUP is geschreven door HaViSoft 1985/86.

Het verbaasde mij dat Dos65 bij het opstarten nogal lang bezig was met "iets". Na het geheel leeg maken van login.com bleek dat e.e.a. aanzienlijk sneller ging. Het meest tijdrovend bleek te zijn: SETUP. Een nadere bestudering daarvan leert dat er in SETUP nogal veel moet gebeuren. Het meeste is wel nuttig maar het hoeft niet elke keer als het systeem opgestart wordt. Dus SETUP blijft bestaan en er is een utility SUFAST gemaakt voor allerdaags gebruik.

Een overzicht van wat SUFAST doet in vergelijking met SETUP:

- Er wordt door SETUP bepaald welke dos versie u heeft. Daartoe wordt door het gehele geheugen gezocht naar de text Dos65. SUFAST doet dit niet, als u het toch wilt weten gebruik dan SETUP. (Nb: dit is de belangrijkste oorzaak dat SETUP zo traag is. Beide programma's SETUP en SUFAST controleren of er echt een rtc is.)
- De rtc wordt zodanig ingesteld dat er elke 500 ms een interrupt is. Waarom is mij niet bekend. De tijd verandert toch pas na 1 seconde. Daarom maakt SUFAST gebruik van een interrupt na iedere seconde. Die arme 6502 heeft het toch al zo druk, dit scheelt mooi een irq in elke seconde.
- Er wordt bijgehouden hoelang het systeem in totaal aan heeft gestaan. Dit doet SUFAST ook om te zorgen dat hier geen rare dingen gebeuren bijv. als meestal SUFAST gebruikt wordt, en soms eens SETUP. Dan zal SETUP dus toch de juiste tijd weergeven.
- De totale systeem gebruikstijd wordt door SETUP op het display gezet. Dit doet SUFAST niet want hoe kleiner het programma hoe sneller het in te lezen is van disk. Het scheelt natuurlijk ook in rekentijd.
- SETUP kontroleert of de rtc zonder stroom heeft gestaan. Kan gebeuren als deze bijvoorbeeld nieuw in het systeem zit. In dit geval zorgt SETUP voor het correct instellen van de rtc. SUFAST controleert dit wel de rtc en geeft dan een waarschuwing, maar kan in zo'n geval niet de rtc instellen. Gebruik daarvoor SETUP.
- SETUP zet een welkomst text op het display met enige versiering. Daarbij staat ook wanneer het systeem voor het laatst gebruikt is. Om tijd en ruimte te besparen is dat weggelaten in SUFAST.

Henk Speksnijder

```
;file sufast.mac                        1989 04 30

;H.Speksnijder
;A.Cuypstraat 43
;2902 GA Capelle a/d IJssel

                ;system
clock           equ     $E120           adres van rtc

                ;io65
datupd          equ     $E77E           variabele voor datum
day1            equ     $E77F           variabele voor dag
month1          equ     day1+1          variabele voor maand
year1           equ     day1+2          variabele voor jaar
hour1           equ     day1+3          variabele voor uur
minut1          equ     day1+4          variabele voor minuut
second1         equ     day1+5          variabele voor seconde
intvec          equ     $E7AF           unreg.interrupt vector

                ;dos65
out             equ     $C023           print char
```

```
crlf        equ     $C02F       print CR + LF
spa         equ     $C032       print spatie
hnout1      equ     $C035       print A hex of nibble
hexout      equ     $C038       print accu als byte
text        equ     $C03B       print string tot nul
hexdec      equ     $C044       convert hex naar decimaal

;interrupt voor aanpassen elke seconde:
areg        equ     $20         32 kHz zonder periodieke interrupt
breg        equ     $17         UIE, DM, 24/12, DSE enable
creg        equ     $10         vlag: einde update
                                ;opmerking: UIE = update interrupt enable

;variabelen in het rtc-ram (beschermde gedeelte)
;elke seconde aangepast:
;           $10     updating year
;           $11     updating month
;           $12     updating day
;           $13     updating hour
;           $14     updating minutes
;           $15     updating seconds
;gebruikt om systeem gebruiks tijd te berekenen:
;           $16     start uur v.h. gebruik
;           $17     start minuut  v.h. gebruik
;           $18     (2 bytes)totaal aantal uren systeem gebruik
;           $1A     total aantal minuten systeem gebruik
;voor status v.d. rtc. ( om te bepalen of deze geheel nieuw is )
;           $1B,$1C programmeer status
;           $1D     rtc             aangesloten status


xstore      macro
            stx     clock
            sta     clock + 1
            endm

ystore      macro
            sty     clock
            sta     clock + 1
            endm

getrtc      macro
            stx     clock
            lda     clock + 1
            endm

            org     $A000
helptxt     jmp     sufast
            fcc     $C8,$C5,$CC,$D0
            fcc     $4C,$4c,$4c                         dummy
            fcc     '\rsupport real time clock MC146818'
            fcc     '\rsyntax: SUFAST'
            fcc     '\rno options, no abbreviation'
            fcc     '\rnote: if there is no rtc then an error message'
            fcc     '\rwill be displayed and the program aborted.\r'
```

```
                fcc     '\rnote: SETμP is needed to clear the rtc the first time'
                fcc     0
;variabelen voor tijdelijk gebruik in werkgeheugen
year            fcc     1               jaar
month           fcc     1               maand
day             fcc     1               dag
hour            fcc     1               uur
minut           fcc     1               minuut
second          fcc     1               seconde
sthour          fcc     1               start uur
stminut         fcc     1               start minuut
uhourl          fcc     1               ( 2 bytes) uren syseem tijd
uhourh          fcc     1
uminut          fcc     1               minuten systeem tijd
statusl         fcc     1               status
statush         fcc     1               status

;_____________________MAIN PROGRAM_____________________________

sufast          lda     #$C0            test of er een rtc is
                ldx     #$1D
                xstore                  schrijf $C0 in de rtc
                getrtc                  lees het terug van de rtc
                cmp     #$C0
                bne     1.f             indien verschillend dan fout
                lda     #$0C
                xstore                  schrijf $0C in de rtc
                getrtc
                cmp     #$0C
                beq     main1           ok en ga door met de rest
1               jsr     text
                fcc     '\rNo real time clock'
                fcc     '\rsetupfast aborted',0
                rts
;kopieer jaar, maand, dag etc van rtc in werkgeheugen
main1           ldy     #$00
                ldx     #$10
main2           getrtc
                sta     year,y
                iny
                inx
                cpx     #$1D
                bne     main2           herhaal tot alles gedaan is
;kontroleer status v.d. rtc
;statusl moet $AA zijn  en statush moet $55 zijn
;als dat niet zo is: gebruik dan eerst SETμP voor initialisatie v.d. rtc
                lda     statusl
                cmp     #$AA
                bne     1.f
                lda     statush
                cmp     #$55
                beq     cusetime        indien ok dan doorgaan
1               jsr     text
                fcc     '\rtry SETUP',0
                rts
```

```
;bereken nieuwe tijd in gebruik
;   usetime: = endtime - starttime
;controleer of een minuut of uur gepasseert wordt
cusetime lda    minut
         cmp    stminut
         bcs    10.f
         lda    minut
         adc    #60
         sta    minut
         dec    hour
         cmp    #$FF
         bne    10.f
         lda    #23
         sta    hour
10       lda    hour
         cmp    sthour
         bcs    20.f
         lda    hour
         adc    #24
         sta    hour
;compute new usetime
20       sec
         lda    minut
         sbc    stminut
         clc
         adc    uminut
         sta    uminut
         cmp    #60
         bcc    40.f
         lda    uminut
         sbc    #60
         sta    uminut
         inc    uhourl
40       sec
         lda    hour
         sbc    sthour
         clc
         adc    uhourl
         sta    uhourl
         cmp    #100
         bcc    50.f
         lda    uhourl
         sbc    #100
         sta    uhourl
         inc    uhourh
;kopieer nieuwe gebruikstijd van werkgeheugen naar rtc-ram
50       ldx    #$18
         ldy    #$00
1        lda    uhourl,y
         xstore
         inx
         iny
         cpy    #$03
         bne    1.b
;haal de actuele tijd uit de rtc en zet dat in het rtc-ram
```

```
                ldx     #$04
                ldy     #$16
                getrtc
                ystore
                dex
                dex
                iny
                getrtc
                ystore
;de interrupt routine omleggen
                sei
                lda     intvec
                cmp     #interr&255
                bne     1.f
                lda     intvec+1
                cmp     #interr>>8
                beq     2.f
1               lda     intvec          kopie oude vector naar extra interrupt routine
                sta     lexint
                lda     intvec+1
                sta     hexint
                ;
                lda     #interr&255     zet nieuwe vector naar extra interrupt routine
                sta     intvec
                lda     #interr>>8
                sta     intvec+1
                ;
2               ldx     #$0A
                lda     #areg
                xstore                  reg. A in rtc = interrupt every update
                ldx     #$0B
                lda     #breg
                xstore                  reg. B in rtc = UIE, DM, 24/12, DSE enable
                cli
                rts
;aanpassen tijd en datum van de rtc
                org     $E600           nieuwe interrupt routine
interr          ldx     #$0C
                getrtc                  haal register C van rtc
                and     #creg
                bne     rtcint          als interrupt van rtc dan ga naar rtcint
                pla                     anders herstel registers
                tay
                pla
                tax
                pla
                fcc     $40             code voor jmp....
lexint          fcc     1               naar oude interrupt routine
hexint          fcc     1
rtcint          ldx     #$07
                getrtc                  haal dag
                cmp     day1
                beq     10.f
                pha
                lda     #$FF            zet vlag voor aangepaste datum
```

Een IBM keyboard aan DOS65

Het is nu mogelijk om, op een simpele manier, een IBM PC/XT keyboard aan de DOS65 te koppelen. Deze sessie is zowel hard- als software ondersteund. Zowel de hard- als de software zijn volledig zelfsupporting, d.w.z. er hoeft niets in de DOS65 omgesoldeerd, danwel opnieuw geprogrammeerd te worden. Dit geeft gelijk het zeer grote voordeel aan deze oplossing t.o.v. eerder gepubliceerde artikelen.

De software

Hiervoor wordt het programma "IBM.MAC" en "ASCII.MAC" van Wim Schimmel te voorschijn gehaald. Hier en daar wat wijzigen en het is helemaal geschikt voor dit project. (zie de 6502 kenners no.53, blz 7 e.v.). Tevens is er een mogelijkheid voor "printscreen" ingebouwd. Iedere keer als de toets SHIFT "*/PrtSc" ingedrukt wordt, wordt eerst controleert of het eigenlijke printscreen programma in het geheugen staat ($A000-einde). Is dit niet het geval dan wordt het programma van disk gehaald en gerunned, staat het programma wel in het geheugen dan wordt het direct gerunned.

IBM_keyb.mac is het toets afvraagprogramma. Het staat in figuur 2.
IBM_table.mac bevat de ASCII toets-waarden en staat in figuur 3. Figuur 4 tenslotte bevat scrdmp.mac, een simpele printer-driver.

De hardware

De hardware bestaat uit een 3-tal IC's, 3 weerstanden, 2 diodes en 2 condensatoren. De schakeling wordt enerzijds d.m.v. een 5 polige DIN-plug met de toetsenbordconnector verbonden en anderzijds door middel van bijvoorbeeld een flatcable aan de toetsenbordingang van de computer verbonden. De computer levert de benodigde voeding.

De werking

Als er een toets ingedrukt wordt genereert het toetsenbord een reeks klokpulsen en een 8 bits toetscode, welke in serie aangeboden wordt, beginnend met D0.

De klokpulsen gaan via N1 naar een pulsvertragend netwerk (R1-R2/C1) om dan als klokpuls aan de 2 SIPO registers aangeboden te worden. Parallel aan deze klokpulsen wordt de datalijn rechtstreeks aan deze SIPO aangesloten. Het pulsvertragend netwerk is aanwezig om straks het KS signaal op het juiste moment te laten komen, zodat er niet tijdens een flankverandering geclocked kan worden. Na 8 vertraagde klokpulsen wordt poort N4 vrijgegeven, zodat de 9e klokpuls de KeyStrobe genereert. Deze strobe blijft de "on" periode aanwezig. Het toetsenbord levert 10 klokpulsen, waardoor aan het eind van de omzetting de schakeling zichzelf reset m.b.v. N3/D1. De combinatie R3/C2-D2 zorgt dat tijdens het inschakelen van de systeem de 2 SIPO's gereset worden.

Opmerking:
Nu zullen enkelen onder ons zich afvragen: "Leuk, maar hoe boot ik nu het systeem op, daar er nog geen keyboard-driver in mijn computer zit !?!?" Heel eenvoudig; door het drukken op toets F8, want deze heeft de toetscode $42 = B(oot) en hoppekee de systeem drive gaat lopen en het computer plezier kan aanvangen, nu met een "echt" toetsenbord.
Uiteraard wel even de keyboard-driver vanuit de LOGIN.COM installeren.

Veel succes,

Frank Bens

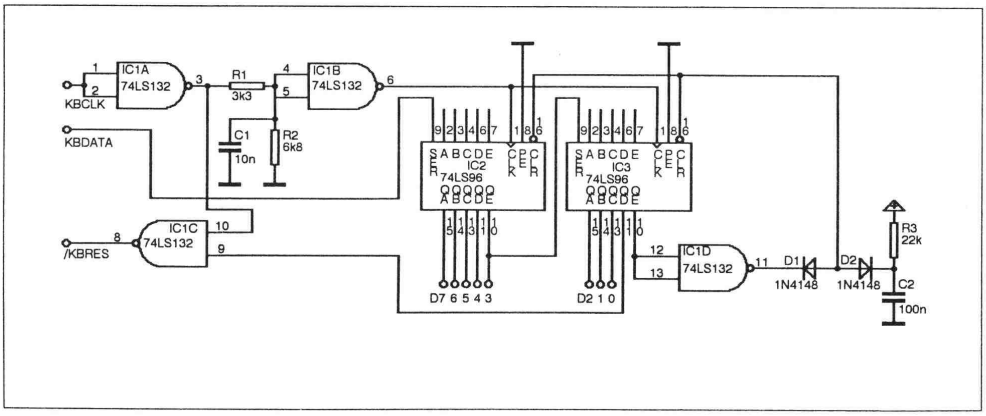


Fig. 1: toetsenbordinterface schakeling

```
; File  : ibm_keyb.mac
;
;               IBM toetsenbord vertaler voor DOS-65 computer
;               Een idee van: Wim Schimmel (6502 kenner no.53)
;               Datum : 23-OKT-86
;               Update/aangepast : Frank Bens dd. 24-DEC-90
;
                opt     nogen

;*
;*
;*  Variables & subroutines
;*
;*
workmem         equ     $a000
initorg         equ     $a000
introrg         equ     $e4af

jsr             equ     $20             ; OPcode
bit             equ     $24

command         equ     $c006           ; DOScommand

viaa            equ     $e100           ; VIA 1 (o.a keyboard)
vapad           equ     viaa+1          ; Poort A Data

conint          equ     $cb2e
keypnt          equ     $e7b7           ; Pointer in keyboardbuffer
tstnd           equ     $e7e0           ; Test speciale toets adres

                org     initorg
ibm             php
                sei                     ; Disable interrupts
                lda     #0
                sta     keypnt          ; Reset keyboardbuffer pointer
                sta     tstnd           ; Initieele waarde is 0 (alles uit)
                lda     #jsr            ; JSR INTER
                sta     conint
                lda     #inter&255
                sta     conint+1
                lda     #inter>>8
                sta     conint+2
                lda     #bit            ; Skip RTS instruktie
                sta     conint+3
                plp                     ; Enable interrupts
                rts                     ; Einde initialisatie routine

                org     introrg

inter           php                     ; Bewaar status register
                and     #%01111111      ; Verwijder het MSB
                tay     ; Toetswaarde in Y
                ldx     tascii-1,y      ; Haal ASCII waarde uit de tabel
                txa
```

Fig. 2: sourcetekst van IBM_keyb.mac

```
                plp                     ; Herstel status register
                bpl     maak            ; Spring als toets is ingedrukt
                cmp     #numlck
                bcc     brkend          ; Spring indien geen control toets
                cmp     #caplck+1
                bcc     lockkey         ; Spring indien CAPS- of NUM-lock toets
                and     tstnd           ; Reset het bijbehorende bit
spclend         sta     tstnd
brkend          pla                     ; Geen geldige ASCII waarde,
                pla                     ; Verwijder terugkeeradres
                rts
                ;
lockkey         lda     #~ctrl
                bit     tstnd           ; Test Ctrl bit
                bne     brkend          ; Spring als Ctrl ingedrukt is
                txa
                eor     #$ff            ; CAPS- of NUM-lock toets
                eor     tstnd           ; Inverteer het 'toggle' bit
                bcc     spclend         ; Altijd
                ;
maak            cmp     #numlck
                bcc     no_ctrl         ; Spring indien geen control toets
                cmp     #caplck+1
                bcc     brkend          ; Spring indien CAPS- of NUM-lock toets
                eor     #$ff            ; Ctrl, Alt of Shift toets
                ora     tstnd           ; Zet het bijbehorende bit aan
                bcs     spclend         ; Altijd
                ;
no_ctrl         cmp     #'a'
                bcc     no_lttr
                cmp     #'z'+1
                bcc     no_mrk1
no_lttr         cpy     #$47            ; Numeriek pad toets?
                bcc     no_mrk          ; Spring indien niet numeriek
                lda     tstnd
                and     #~numlck|~shift
                beq     mkend           ; Spring als NUM-lock en Shift uit zijn
                cmp     #~numlck|~shift
                beq     mkend           ; Spring als NUM-lock en Shift aan zijn
                ldx     tnumlck-$47,y   ; Haal numeriek pad toets
                bpl     mkend           ; Altijd
                ;
no_mrk1         lda     #~caplck|~shift
                fcc     $2c             ; Dummy BIT instruktie
                ;
no_mrk          lda     #~shift
                bit     tstnd           ; Test Shift bit
                beq     mkend           ; Spring als Shift niet ingedrukt
                ldx     tshift-1,y      ; Haal shift toets
                cpx     #prtscr         ; Print scherm ?
                beq     prscr
                ;
mkend           lda     #~ctrl
                bit     tstnd           ; Test Ctrl bit
                beq     einde           ; Spring als Ctrl niet ingedrukt
                txa
```

```
                cmp     #ins
                bcs     curctrl         ; Als ASCII waarde > = cursor-control
                and     #%00011111      ; Control code ($00 - $1F)
                fcc     $2c             ; Dummy BIT instruktie
                ;
curctrl         adc     #15             ; Cursor control code
                tax
                ;
einde           txa                     ; Karakter in A-register
                rts                     ; Geldige ASCII waarde
                ;
prscr           ldx     #6              ; Controleer of file
check           dex     ; reeds in geheugen
                bmi     exec            ; aanwezig is
                lda     scrdmp,x
                cmp     workmem+3,x
                beq     check
loadfil         lda     #getdmp>>8      ; Zoniet, lees file
                ldy     #getdmp&255
                jsr     command
                lda     #'\r'
                bvs     exit            ; Spring indien file niet bestaat
exec            jsr     workmem         ; Executeer programma
                pla                     ; Verwijder terugkeeradres
                pla
exit            rts
```

```
; File  : ibm_tabl.mac
;
;               ASCII converter voor IBM (/XT) toetsenbord.
;               Naar een idee van: Wim Schimmel (6502 kenner no.53)
;               Datum : 22-OKT-86
;
;*
;*
;*              Variables
;*
;*
break           equ     $03             ; shift scroll lock
scrlck          equ     $13             ; scroll lock (DC3)
esc             equ     $1b             ; escape
spcbar          equ     $20             ; spacebar
del             equ     $7f             ; delete

                ; funktie toetsen

f1              equ     $80
f2              equ     $81
f3              equ     $82
f4              equ     $83
```

Fig. 3: toetsentabel voor IBM toetsenbord

```
f5          equ     $84
f6          equ     $85
f7          equ     $86
f8          equ     $87
f9          equ     $88
f10         equ     $89

tab         equ     $8a             ; tab rechts

            ; Shift funktie toesten

sf1         equ     $8b
sf2         equ     $8c
sf3         equ     $8d
sf4         equ     $8e
sf5         equ     $8f
sf6         equ     $90
sf7         equ     $91
sf8         equ     $92
sf9         equ     $93
sf10        equ     $94

stab        equ     $95             ; tab links

prtscr      equ     $96             ; print screen

            ; numeriek pad toetsen

ins         equ     $a0
end         equ     ins+1
down        equ     ins+2
pgdn        equ     ins+3           ; page down
left        equ     ins+4
centre      equ     ins+5
right       equ     ins+6
home        equ     ins+7
up          equ     ins+8
pgup        equ     ins+9           ; page up
delkp       equ     ins+10          ; delete (keypad)

            ; Speciale control karakters

numlck      equ     %11101111
caplck      equ     %11110111
alt         equ     %11111011
shift       equ     %11111101
ctrl        equ     %11111110

            ; Tabellen

tascii      fcc     esc,'1234567890-=',del
            fcc     tab,'qwertyuiop[]\r'
            fcc     ctrl,'asdfghjkl;\'"
            fcc     shift,'\\zxcvbnm,./',shift,'*'
            fcc     alt,spcbar,caplck
            fcc     f1,f2,f3,f4,f5,f6,f7,f8,f9,f10
```

```
                fcc     numlck,scrlck
                fcc     home,up,pgup,'-'
                fcc     left,centre,right,'+'
                fcc     end,down,pgdn
                fcc     ins,delkp

tshift          fcc     esc,'!@#$%^&*()_+',del
                fcc     stab,'QWERTYUIOP{}\r'
                fcc     0,'ASDFGHJKL:"~'
                fcc     0,'|ZXCVBNM<>?',0,prtscr
                fcc     0,spcbar,0
                fcc     sf1,sf2,sf3,sf4,sf5,sf6,sf7,sf8,sf9,sf10
                fcc     0,break
tnumlck         fcc     '789-'
                fcc     '456+'
                fcc     '123'
                fcc     '0.'

                end     ibm
```

```
;
; File : SCRDMP.MAC
;
; Door          : Frank Bens
;
;                Screendump routine
;
; Datum : 24-DEC-90
;
                opt     nogen

;*
;*
;*              Variables & subroutines
;*
;*
workmem         equ     $a000

cold            equ     $c000
statpri equ     cold+$a15               ; Status printer
putpri          equ     cold+$a18       ; Print always (ignore status)
userbrk         equ     cold+$a1e       ; Check user break

start           equ     $e7ad           ; Top current videoscreen (2 bytes)

vidram          equ     $e800           ; Start of videoram

jmp2byt         equ     $2c             ; OPcode BIT xxxx

hight           equ     24              ; Rows per screen
width           equ     80              ; Character per row
```

Fig. 4: eenvoudige Print_Screen driver

```
                org     workmem

stdump          jmp     scrdmp

file_id         fcc     'scrdmp'        ; File identifier (always 6 bytes long)
version         fcc     ' V1.0'         ; Version number
linecnt         res     1               ; Linecounter
;*
;*
;*              Screendumproutine
;*
;*
scrdmp          cli                     ; Printing possible
                lda     start           ; Print screen
                ldx     start+1
                sta     getscr+1
                stx     getscr+2
                ldx     #hight          ; Set line-counter
                stx     linecnt
prloop          ldx     #width          ; Set colomn-counter
getscr          lda     getscr          ; Get screen data
                and     #%01111111      ; Strip hi bit/deselect inverse video
                cmp     #' '
                bcs     2.f             ; Branch when > = $20
                lda     #'\'            ; Print special character
2               jsr     centro          ; To centronic device
                bcs     prerr           ; Branch on error
                inc     getscr+1        ; Incerease currentaddress videoram
                bne     chkcol
                inc     getscr+2
                lda     getscr+2        ; Check videoramborders
                and     #vidram+$0700>>8                ; Max. $EFFF
                ora     #8              ; Min. $E800
                sta     getscr+2
chkcol          dex
                bne     getscr          ; End of line ?
                lda     #'\r'           ; Start on new line
                jsr     centro          ; To centronic device
                lda     #'\n'
                jsr     centro
                dec     linecnt         ; Linecounter -1
                bne     prloop          ; Entire screen printed ?
prerr           rts                     ; End of print screen routine

centro          tay                     ; Save accu
wait            jsr     userbrk         ; Test breakkey
                bcs     90.f            ; Breakkey pressed, yes then exit
                jsr     statpri         ; Get printer status
                bcs     wait            ; ACK received, no then wait
                tya                     ; Restore accu
                jsr     putpri          ; Print always (ignore status)
90              rts                     ; Exit

                end
```