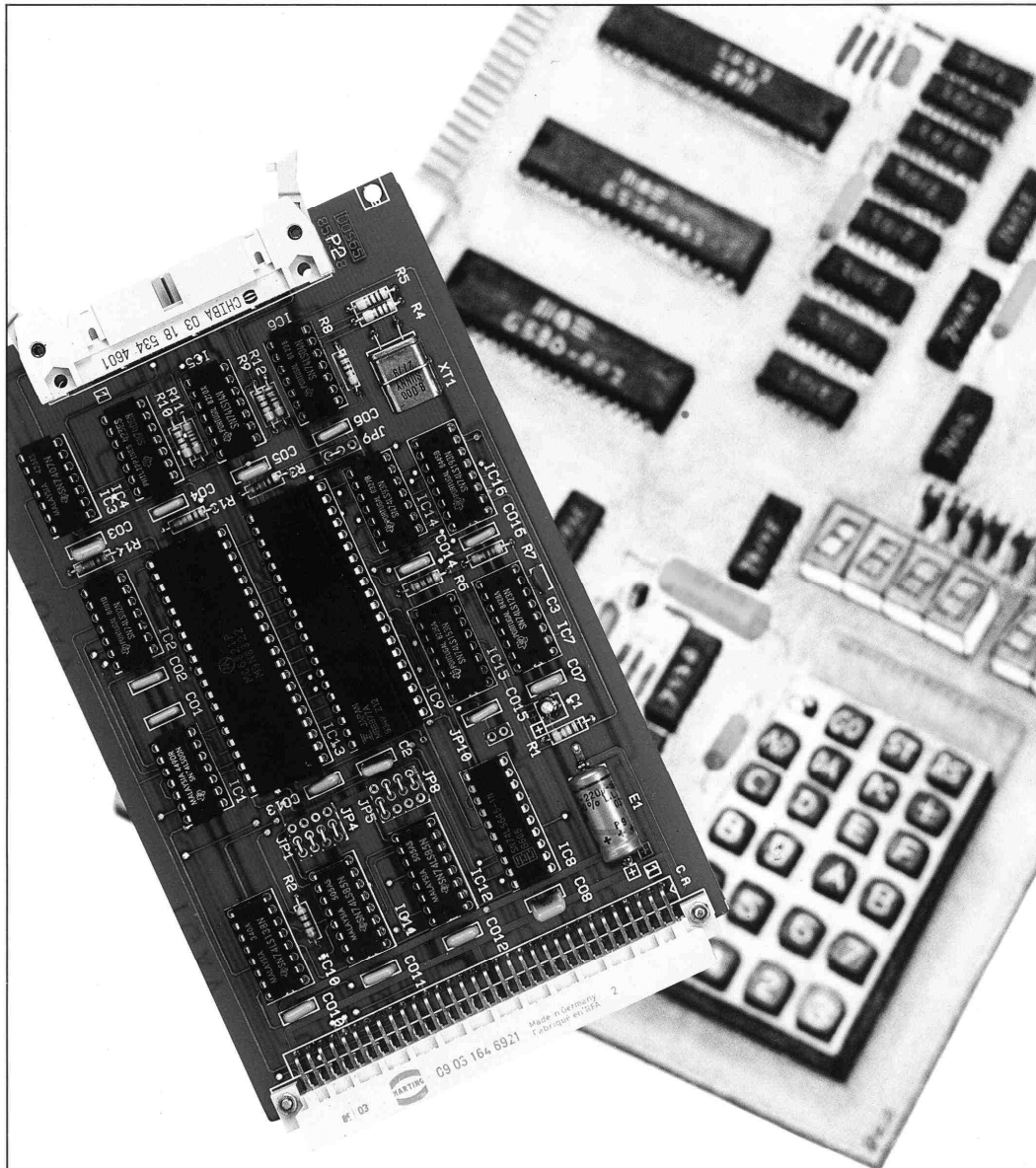




INFORMATIE.

De uP Kenner (De microprocessor Kenner) is een uitgave van de KLM gebruikersclub Nederland. Deze vereniging is volledig onafhankelijk, is statutair opgericht op 22 juni 1978 en ingeschreven bij de Kamer van Koophandel en Fabrieken voor Hollands Noorderkwartier te Alkmaar, onder nummer 634305.

De doelstellingen van de vereniging zijn sinds 1 januari 1989 als volgt geformuleerd:

- Het vergaren en verspreiden van kennis over componenten van microcomputers, de microcomputers zelf en de bijbehorende systeemsoftware.
- Het stimuleren en ondersteunen van het gebruik van microcomputers in de meer technische toepassingen.

Om deze doelstellingen zo goed mogelijk in te vullen, wordt onder andere 6 maal per jaar de uP Kenner uitgegeven. Verder worden er door het bestuur per jaar 5 landelijke bijeenkomsten georganiseerd, beheert het bestuur een Bulletin Board en wordt er een software-bibliotheek en een technisch forum voor de diverse systemen in stand gehouden.

Landelijke bijeenkomsten:

Deze worden gehouden op bij voorkeur de derde zaterdag van de maanden januari, maart, mei, september en november. De exacte plaats en datum worden steeds in de uP Kenner bekend gemaakt in de rubriek Uitnodiging.

Bulletin Board:

Voor het uitwisselen van mededelingen, het stellen en beantwoorden van vragen en de verspreiding van software wordt er door de vereniging een Bulletin Board beschikbaar gesteld.

Het telefoonnummer is: 053-303902.

Software Bibliotheek en Technisch Forum:

Voor het beheer van de Software Bibliotheek en technische ondersteuning streeft het bestuur ernaar zgn. systeemcoördinatoren te benoemen. Van tijd tot tijd zal in de uP Kenner een overzicht gepubliceerd worden. Dit overzicht staat ook op het Bulletin Board.

Het Bestuur:

Het bestuur van de vereniging wordt gevormd door een dagelijks bestuur bestaande uit een voorzitter, een secretaris en een penningmeester en een viertal gewone leden.

Voorzitter:
Rinus Vleesch Dubois
Emiliano Zapataplein 2
2033 CB HAARLEM
Telefoon 023-330993

Secretaris:
Gert Klein
Diedenweg 119
6706 CM WAGENINGEN
Telefoon 08370-23646

Penningmeester:
Jacques H. Banser
Haaksbergerstraat 199
7513 EM ENSCHEDE
Telefoon 053-324137

Leden:

Jan D.J. Derksen
Ed Verkadestraat 9-1
7558 TH HENGEL
Telefoon 074-770970

Adri Hankel
Willem Kloosstraat 32
7606 BB ALMELO
Telefoon 05490-51151

Gert van Opbroek (Redactie uP Kenner)
Bateweg 60
2481 AN WOUBRUGGE
Telefoon 01729-8636

Nico de Vries
Mari Andriessenrade 49
2907 MA CAPELLE A/D IJSSEL
Telefoon 010-4517154

Ereleden:

Naast het bestuur zijn er een aantal ereleden, die zich in het verleden bijzonder verdienstelijk voor de club hebben gemaakt:

Erevoorzitter:

Siep de Vries

Ereleden:

Mevr. H. de Vries van der Winden
Anton Mueller

De uP Kenner:

De uP Kenner is het huis-
orgaan van de KIM Gebrui-
kersclub Nederland en
wordt bij verschijnen gra-
tis toegezonden aan alle
leden van deze club.

Verschijningsdata:

De uP Kenner verschijnt op
de derde zaterdag van de
maanden februari, april,
juni, augustus, oktober
en december.

Kopij:

Kopij voor het blad dient
bij voorkeur van de leden
afkomstig te zijn. Deze
kopij kan op papier of in
machine-leesbare vorm op-
gestuurd worden aan het
redactieadres. De redactie
beslist, op basis van
bruikbaarheid, publicatie-
waarde en actualiteit of
en zo ja, wanneer een
ingezonden artikel ge-
plaatst wordt.

Geplaatste artikelen blij-
ven het geestelijk eigen-
dom van de auteur en mogen
niet zonder diens toe-
stemming door derden gepu-
bliceerd worden.

Helaas kan de redactie
noch het bestuur enige
aansprakelijk aanvaarden
voor de toepassing(en) van
de geplaatste kopij.

Redactie.

De redactie wordt gevormd
door:

Gert van Opbroek

Correspondenten:

Bram de Bruine

Antoine Megens

Nico de Vries

Rinus Vleesch Dubois

Redactieadres:

Gert van Opbroek

Bateweg 60

2481 AN Woubrugge

Druk:

ACI Offsetdrukkerij B.V.

Langsom 10-16

1066 EW Amsterdam

INHOUDSOPGAVE

Vereniging

Informatie	2
Uitnodiging clubbijeenkomst	5

Algemeen

Redactioneel	4
Van de voorzitter	6
Getallen (deel 4)	14
Computers (deel 3)	32

Bulletin Board

Het Bulletin Board	6
--------------------------	---

DOS-65 Corner

DOS-65, 40 track: een kopieertip	6
Programmeren in assembler (deel 2)	7
Nogmaals: CRTC.DOC	13
Basicode3 voor DOS-65	26
Tracer	39

EC-65(k)

Eproms programmeren	28
---------------------------	----

Hardware

Nieuwe prijzen voor EP hardware	31
EP: Errata	31

MS-DOS

De IBM-PC en z'n klonen (deel 3)	47
--	----

Talen/Software

Een Public Domain APL-implementatie	51
---	----

Vraag en aanbod

Te Koop	50
---------------	----

Redactioneel.

Op het moment waarop ik dit schrijf, zit u misschien al uit te kijken naar de uP Kenner. Ik ben dus weer eens wat aan de late kant met het inleveren van het blad bij de drukker en zodoende krijgt u het blad ook iets later dan normaliter de bedoeling is. Over het algemeen komt het blad de derde zaterdag van de even maanden uit. Dat betekent dat het blad in de loop van de week volgend op deze zaterdag (meestal de donderdag) bij u in de bus ligt. Deze keer zal dat niet lukken en zal het blad waarschijnlijk een week te laat bij u in de bus liggen. Ik verontschuldigd mij daarvoor want uw redacteur heeft het in de afgelopen periode zeer druk gehad en als dan ook nog de eerste zaterdag van de maand op 1 april valt, dan wordt het allemaal erg kort dag.

Eerst iet over de vorige uitgave van het blad. Ik was zelf zeer tevreden over de vorm en de inhoud van het blad en met mij een aantal lezers. Ik heb van een aantal mensen te horen gekregen dat het blad er bijzonder goed uitzag en ook de inhoud bijzonder aansprak. Ik ben daar erg blij om, want zelf wil ik, als ik de kans krijg, graag bladen maken die de leden aanspreken. Ik maak de uP Kenner natuurlijk niet alleen. Ik wil de inzenders van kopij dan ook heel hartelijk danken voor hun bijdrage(n) en de complimenten hierbij doorgeven aan hun.

Wat mij overigens opviel, is dat ik steeds meer artikeltjes krijg in de vorm van een verhaaltje. Op deze manier krijgt het blad meer de vorm van een magazine en wordt veel afwisselender. Ik ben daar wel blij mee waarbij ik ook vindt dat er ruimte moet zijn voor programma's. Ik denk dat een middenweg het beste is, enerzijds meer verhalende artikelen, anderzijds ook één of twee programma's die voor anderen nuttig kunnen zijn of als voorbeeld voor meer algemene zaken kunnen dienen. Het nadeel van programma's is echter wel dat ze relatief veel ruimte innemen. Hoe denkt u hier trouwens over? Wat zou u graag in het blad zien? Spreekt het blad u aan of juist niet, laat het mij eens weten, per brief, bulletin board of telefoon. Samen met u kan dan het blad misschien (nog) beter gemaakt worden.

Kopij kan ik natuurlijk altijd gebruiken. De volgende uitgave komt in de zomer en dat is voor computer-hobbyisten vaak een tijd dat ze liever buiten bezig zijn dan

achter een toetsenbord kruipen. Eén van mijn correspondenten heeft geprobeerd buiten in de zon met de computer bezig te zijn, maar dat was ook geen succes; hij kon nauwelijks lezen wat er op zijn beeldscherm stond. Vanwege het feit dat er minder gehobbiet wordt, breekt er voor de redactie vaak de zogenaamde Komkommer-tijd aan met een gebrek aan kopij. Ik wil u dus vragen, mij zoveel kopij te sturen dat ik problemen heb met het uitzoeken wat ik niet in het blad moet plaatsen in plaats van slapeloze nachten te hebben over hoe ik het blad nu weer vol moet krijgen.

Een aantal mensen zullen mij wel op de bijeenkomst in Geldrop gemist hebben. Ik was wel van plan te komen, maar we hadden in ons huis een kleine verbouwing en dat vroeg zo veel energie dat ik de puf niet had naar Geldrop te rijden. Ik vindt dat erg jammer want ten eerste had ik graag de lezing bijgewoond en te tweede mag ik graag even met de aanwezigen keuvelen. Ik beloof u: Op de volgende bijeenkomst ben ik, ijs en weder dienende, wel van de partij, u toch ook? De volgende bijeenkomst vindt plaats in Almelo en ik heb begrepen dat er enkele zeer spectaculaire dingen op het programma staan. Twee van de bestuursleden (Rinus en Nico) houden gezamenlijk een voordracht over de diverse grafische kaarten voor MS-DOS machines. Verder zullen ze met behulp van enkele machines wat grafische mogelijkheden laten zien. Ik heb begrepen dat er hoog-resolutie kleurenbuizen en dergelijke meegesleept zullen worden naar Almelo. 's Middags, tijdens het informele gedeelte, worden er demonstraties gegeven hoe met behulp van een PC printontwerpen gemaakt kunnen worden; kortom hoe bijvoorbeeld de EPROM-Programmer tot stand gekomen is. Al met al zeer interessant dus.

Tenslotte nog iets over de inhoud van het blad. In dit nummer staat de volgende aflevering van de rubrieken getallen, computers, PC's en de assemblercursus. Een verhaal over communicatie staat er dit keer niet in, Bram de Bruine is wel bezig met een volgende aflevering maar heeft iets meer tijd nodig. Verder staat er een tracer-programma voor DOS-65 in en nog wat andere zaken. Ik hoop dat u veel plezier aan het blad beleeft en wens u verder veel genoegen aan uw hobby.

Gert van Opbroek

Uitnodiging voor de clubbijeenkomst

Datum: Zaterdag 20 mei 1989
Locatie: Wijkcentrum 't Veurbook
Jan Tooropstraat 27
7606 Almelo
Tel.: 05490 - 10353

Toegang: fl. 10,--

Routebeschrijving

Van uit het westen en het zuiden (A1/A35):
1. Aan het einde van de snelweg rechtsaf. Bij het eerstvolgende kruispunt MET VERKEERSLICHTEN linksaf, richting Wierden/Zwolle. Bij de eerstvolgende verkeerslichten recht door. Bij de volgende verkeerslichten (links BP tankstation en Opel garage Kamp) gaat u rechtsaf.

2. U rijdt nu op de Windmolenbroeksweg. Doorrijden tot over de brug, dan de eerste straat rechts. Dit is de W. van Konijnenburgstraat. Na plm. 50 meter rechtsaf. Dit is de Tooropstraat. Met de bocht mee naar links. Na plm. 50 meter aan de rechterkant: 't Veurbrook.

Van uit het noorden (via de N 36):

1. Bij de stoplichten rechtsaf, richting streekziekenhuis. U bevindt zich nu op de rondweg om Almelo. Deze weg blijven volgen tot u het BP tankstation ziet bij dit kruispunt linksaf. Zie verder punt 2.

Met openbaar vervoer:

Vanaf NS-station Almelo met de stadsbus naar de wijk Molenbroek. Uitstappen bij de halte Windmolenbroeksweg. Schuin tegenover de bushalte staat een wegwijzer, daarop staat ook 't Veurbrook vermeld.

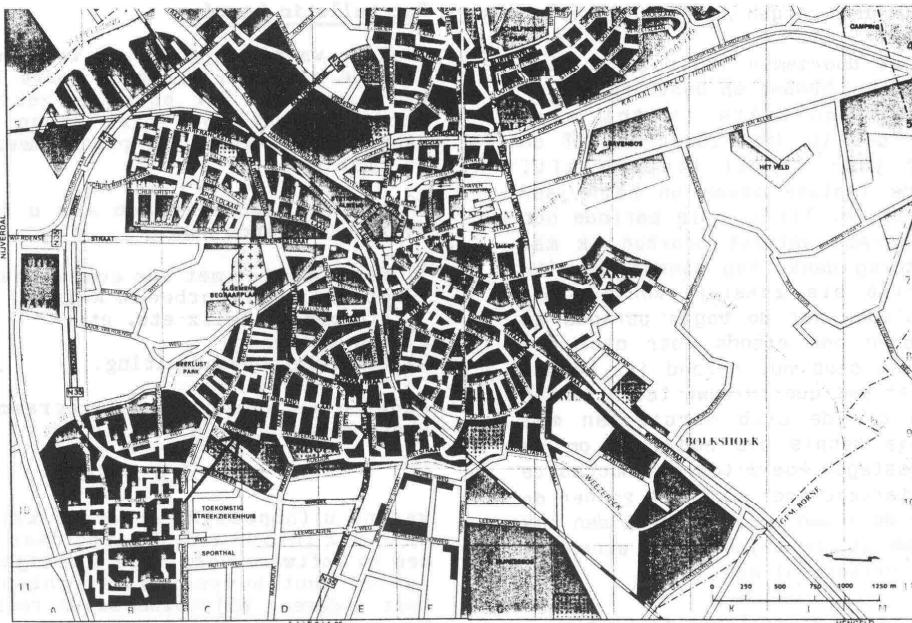
Programma:

9:30 Zaal open met koffie
10:15 Opening
10:30 Lezing door Rinus Vleesch Dubois en Nico de Vries over CGA, Hercules, EGA en VGA; kortom de grafische kaarten voor de PC's en compatibles. Dit met demo's op professionele apparatuur.
11:30 Forum en markt
12:30 Lunch, consumpties tegen betaling

Aansluitend het informele gedeelte met de mogelijkheid om andermans systemen te bewonderen en Public Domain software uit te wisselen. Als extra attractie ditmaal demonstraties printontwerp met behulp van de computer. U en uw systeem zijn van harte welkom.

17:00 Sluiting.

Het bestuur maakt u er nogmaals op attent dat het illegaal verspreiden van software ten strengste verboden is. Doe dat dus niet.



VAN DE VOORZITTER

Helaas heb ik de bijeenkomst te Geldrop door omstandigheden gemist. Ik heb begrepen dat de opkomst mager was, maar de kwaliteit goed. De komende bijeenkomst is zoals vermeld te Almelo. U doet het bestuur een groot genoegen door in groten getale de bijeenkomst te bezoeken opdat het onderwerp van die dag naar mijn mening zeer interessant is.

Het onderwerp van de bijeenkomst is het video gebeuren van de IBM machines en wel zo uitgebreid als mogelijk. Tevens ondersteunen wij dit onderwerp met professionele apparatuur die (onder voorbehoud van ...) ons ter beschikking wordt gesteld door "piep piep piep B.V."

Na de maand mei breekt de vakantie periode los en zal het bestuur tijdelijk op halve kracht functioneren en van een welverdiende vakantie genieten. Ik reken er wel op dat voor het zover is, u onze redactie-man Gert van Opbroek zal overladen met interessante kopij zodat hij zonder al te veel inspanning de mp KENNER kan afleveren bij onze drukker en zelf ook met zijn familie kan genieten van een vakantie zonder zorgen over de kopij.

Tijdens het doornemen van oude notulen die betrekking hebben op bestuursvergaderingen, ontdekte ik tot mijn verbazing dat ik deze zomer twaalf en een half jaar in het bestuur zit, waarvan de laatste zeven en half jaar als voorzitter. Tijdens die periode heb ik vele mensen ontmoet waaraan ik met plezier terug denk. Nog steeds heb ik, zowel privé als zakelijk contact met vele clubleden uit de begin periode en wij verbazen ons steeds weer over het feit dat de club nog gezond is. Hoewel wij nu best wat meer nieuwe leden kunnen gebruiken die de club versterken met vooral die kennis die nodig is om de nieuw ingeslagen koers tot een succes te maken. Uiteraard gaat dat niet zonder de inzet van de leden zelf. Ik stel dan ook voor, neem op de bijeenkomst eens een kennis of vriend(in) mee.

Tot ziens op de bijeenkomst in Almelo.

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 kopiëren 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: blanco disk in drive B: (360k) en de dubbelstep DOS65 disk in drive A: en getypt: COPYIIPC A: B:.

En ja hoor: het werkt vlekkeloos!

Het bulletin board.

Zoals u waarschijnlijk wel weet, wordt er door de club een bulletin board beschikbaar gesteld. Dit board (afgekort BBS) wordt bedient door onze Sysop Jacques Banser die bovendien de penningmeester van de club is.

U kunt het BBS bereiken als u beschikt over de volgende zaken:

- 1) Een computer met een communicatieprogramma; bijvoorbeeld Kermit, Astrid, Procomm of Telix etc. etc.
- 2) Een modem.
- 3) Een telefoonaansluiting.

Nadat u het communicatieprogramma opgestart hebt, zoekt u contact met:

0 5 3 - 3 0 3 9 0 2

waarna u (hopelijk) contact krijgt met het 6502 Info Board. U vindt daar berichten en software voor alle mogelijke systemen en kunt bovendien berichten sturen naar anderen, bijvoorbeeld de redactie of een van de andere auteurs.

Programmeren 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 originele 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, drierwef 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

$$\begin{array}{rcl} 330 = & (1 \times 16^2 = 1 \times 256) & \rightarrow 330 - 256 = 74 \\ & + (4 \times 16^1 = 4 \times 64) & \rightarrow 74 - 64 = 10 \\ & + (10 \times 16^0 = 10 \times 1) & \rightarrow 10 - 10 = 0 \\ \hline & \$14A & \end{array}$$

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 grootste 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.

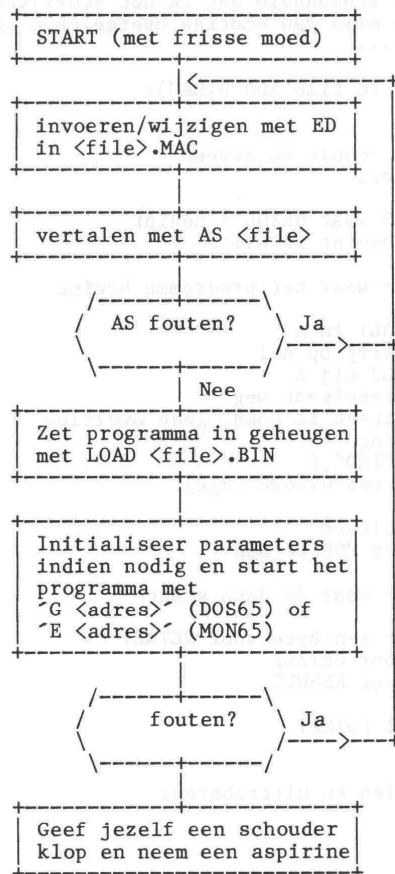
Programma ontwikkeling

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 toetsenbord 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 electronenbuisen 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

MON>

```

; Is dat even luxe? We hoeven niet meer met de hand
; 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
OB06$
```

We hebben GETAL1 gevuld met \$01A0 en GETAL2 met \$0966, het resultaat is \$0B06 en dat klopt. De carry die is ontstaan 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-sigitaal afnemen van pen 12 van IC3 (N22) enz.

Dit moet zijn: Als eerste het VIDEO-sigitaal 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.

Getallen (Deel4).

Door Gert van Opbroek
Bateweg 60
2481 AN Woubrugge

Inleiding.

Na het in de voorgaande aflevering gehad te hebben over het optellen en aftrekken van floating point getallen gaan we in deze aflevering verder met vermenigvuldigen en delen. Ook in deze aflevering staan bijbehorende subroutines voor een 6502-systeem. (De routines zijn m.i.v. vandaag ook beschikbaar op het bulletin board).

Misschien is het wel aardig te vertellen hoe deze serie eigenlijk tot stand gekomen is. Welnu, ik heb, jaren geleden, eens een set procedures voor floating-point getallen overgetikt uit een boekje met programma's voor de 6502 [1]. Dit waren routines voor vier byte getallen (dus net zo als IEEE single precision). Bovendien was ik in die tijd zeer veel bezig met het programmeren in Forth en miste daar in de 79-standaard versie de mogelijkheden van floating point. Ik ben toen bezig geweest in de assembler bij Forth een set routines te maken. Deze routines hanteren een eigen formaat van 6 byte en kennen een zeer grote range en tamelijk grote precisie. (Voor de liefhebbers heb ik deze routines ook op het BBS gezet, evenals de bijbehorende assembler).

Het inlezen en afdrukken van de getallen kan verder in elke basis, dus in decimaal, hex, binair etc. Nadeel is echter dat vooral de afdrukroutine tamelijk traag is. Ik ben met dit project gestopt omdat ik andere interesses (68000) kreeg en omdat het mij toen niet lukte de routines voor de functies zoals sinus, cosinus, exponent etc. te bemachtigen.

Daar ik al het één en ander met floating point gedaan had en omdat ik het ontbreken van floating point in DOS-65 'C' een gemis vindt, heb ik medio vorig jaar besloten me eens bezig te gaan houden met floating point routines. Nadat ik het besluit genomen had was het Duitse tijdschrift MC, waarop ik geabonneerd ben, zo vriendelijk hetzelfde ook te doen zodat ik een groot deel van deze artikelenreeks op de artikelen in MC [2..6] kan baseren. MC heeft ook de routines voor optellen, aftrekken, vermenigvuldigen en delen in assembler voor 68000, 8088 en Z80 gepubliceerd. Daar ik behalve in 6502-assembler ook goed thuis ben in 68000-assembler was het om-

zetten dus relatief eenvoudig. De gepubliceerde routines zijn dan ook vrijwel een één op één conversie van de routines voor de 68000. Het gevolg is echter wel dat de 6502-routines op enkele onderschikte punten wat geoptimaliseerd zouden kunnen worden, dat laat ik echter graag over aan de toekomstige gebruiker.

MC plaatste ook de inlees- en afdruk-routines, echter in 'C' zodat de conversie hiervan naar 6502-assembler mogelijk wat problematischer zal gaan worden. U kunt er echter van uit gaan dat ze zeker in 6502 assembler en mogelijk ook in 68000 assembler gemaakt zullen gaan worden.

Ik denk dat de artikelenreeks als volgt verder gaat: In dit nummer komen de routines voor vermenigvuldigen en delen, in het volgende nummer de inlees- en afdruk-routine waarna in het daarop volgende nummer een rekenmachine gemaakt zal gaan worden. Daarna wil ik eigenlijk de routines voor de diverse functies (SIN, COS, EXP, LOG) behandelen alleen heb ik onvoldoende gegevens over de polynomen met behulp waarvan ze uitgerekend worden. Als er mensen zijn die mij aan deze polynomen kunnen helpen, willen die mij dan deze informatie opsturen?

Een mogelijke referentie zou [7] kunnen zijn wie helpt?

Vermenigvuldigen.

Het vermenigvuldigen (en ook het delen) van floating point getallen lijkt, in tegenstelling tot optellen, zeer veel op het vermenigvuldigen van gehele getallen. Ook in het decimale stelsel is dat zo, kijk maar naar het volgende voorbeeld:

$$1,15E4 * 2,21E3 = 2,5415E7$$

Het rekenschema bestaat uit vier stappen te weten:

- 1) Bereken de nieuwe exponent. Dit is de som (signed) van de exponenten van de operanden.
- 2) Bereken het integer-product van de mantissa's dus in het voorbeeld $115 * 220$. Van het resultaat komen evenveel cijfers achter de komma als de operanden tesamen hadden.
- 3) Normeer het resultaat zodanig dat er in de mantissa weer één cijfer voor de komma staat.

- 4) Bepaal het teken van het resultaat; dit is een '+' als van de operanden de tekens gelijk zijn en een '-' als ze ongelijk zijn (dus een exclusieve OR).

Kijken we nu naar de assembler-listing, dan kunnen we zien dat dit rekenschema ook hier gebruikt wordt. Er zijn echter enkele bijzondere gevallen die behandeld moeten worden. In de eerste plaats is dat het geval dat één van de operanden niet in het genormaliseerde formaat staat maar zo klein is dat ze in het gedenormaliseerde formaat staat. Hiervoor wordt een correctie uitgevoerd door van een operand in het genormaliseerde formaat de exponent te verlagen (regel 404 t/m 409).

Bij de berekening van de nieuwe exponent moeten we na de optelling er rekening mee houden dat de exponenten een bias van \$7F kennen omdat ze in het zogenaamde signed magnitude formaat staan waarbij \$7F betekent dat de exponent 0 is. Bij een recht toe rechtaan optelling is het resultaat dus 127 te groot. Omdat er echter al een klein voorschotje op het normaliseren genomen wordt, wordt er geen 127 doch slechts $127-3 = 124$ afgetrokken. Krijgen we na deze correctie een resultaat kleiner dan -24, dan is zeker dat het resultaat kleiner is dan het kleinste mogelijke getal dat we in 4 byte weer kunnen geven (underflow), en zal het resultaat als 0 (nul) weergegeven worden.

Na deze bewerking wordt de nieuwe mantissa berekend met behulp van een vermenigvuldigings-routine volgens het algoritme uit deel 1.

Alvorens over te gaan tot het normeren, wordt bij het resultaat 1/2 LS bit opgeteld voor het afronden tot nearest. (4/5 in het dagelijks leven). Op dit moment zijn alle resultaten met een exponent 0 genormeerd; dit zijn de gedenormaliseerde getallen. Van de overige getallen wordt de mantissa naar links geschoven en de exponent verhoogd net zo lang totdat we een exponent > 255 (overflow) of een MS-bit 1 hebben. Resultaat 0 en overflow worden afzonderlijk (00 00 00 00 resp. 00 00 00 FF) behandeld.

Als laatste wordt het teken bepaald en wordt het resultaat op de stack geschreven.

Delen.

Als we twee floating point getallen moeten delen, dan moeten we er in de eerste plaats voor zorgen dat beide getallen evenveel cijfers voor de komma hebben of dat beide getallen evenveel nullen tussen het eerste cijfer ongelijk 0 en de komma hebben dus:

12,34 / 0,032

moet omgezet worden naar:

12,34E0 / 32E-3

of naar

0,01234E3 / 0,032E0

of naar

1,234E1 / 3,2E-2

Daarna wordt de exponent van de tweede operand afgetrokken van die van de eerste. Dit is vooralsnog de exponent van het quotient.

Hierna kan men gaan delen. In het resultaat komt de komma meteen na het eerste cijfer (in het voorbeeld een 0) te staan. Eveneens bij vermenigvuldigen is het teken de exclusieve OR van de tekens van de operanden. Het (ongenormeerde) resultaat van het voorbeeld is dus:

0.3843E3

In het binaire geval gaat het niet anders. Na de correctie van de genormaliseerde getallen worden ook hier de exponenten van elkaar afgetrokken. Daar de exponenten met hun bias van 127 van elkaar afgetrokken worden, moet deze er in het resultaat weer bij worden opgeteld. Alles wat hierna een exponent kleiner dan -24 heeft, leidt tot underflow en geeft dus als resultaat 0.

Als de deler gelijk is aan 0, dan is het resultaat oneindig en dit wordt dan ook weergegeven.

Na de berekening van de nieuwe exponent, worden de mantissa's zodanig geschoven dat het MS-bit 0 is. Hierna wordt de deler naar links geschoven totdat het volgende bit 1 is. Voor elke verschuiving wordt hierbij de exponent van het resultaat met 1 verhoogd.

Alvorens de deling te starten, wordt bij de exponent van het resultaat 25 opgeteld. Dit kan beschouwd worden als voorschot op het normeren.

Het algoritme voor het delen is hierna heel eenvoudig. De deler wordt van het deeltal afgetrokken. Lukt dit, dan wordt er in het resultaat van rechts af een bit 1 geschoven, lukt dit niet (resultaat < 0) dan wordt het deeltal weer teruggehaald en wordt er in het resultaat van rechts af een 0 geschoven. Aansluitend wordt het deeltal naar links geschoven waarbij de exponent van het resultaat met 1 verlaagd wordt. Bij deze verschuiving wordt een bit 0 in het deeltal geschoven. Dit proces gaat door totdat er aan de linker kant een 1 uit het resultaat geschoven wordt of totdat de exponent van het resultaat 0 geworden is.

Het afronden, normaliseren en bepalen van het voorteken gebeurt met routines uit de vermenigvuldiging.

Testset.

Naast de gebruikelijke getallen zijn er een groot aantal speciale gevallen. Enkele van deze speciale gevallen zijn ook in de artikelen van mc beschreven. Deze gevallen zijn in de volgende tabel weergegeven.

Voor optellen:

$3f\ 80\ 00\ 00 + 40\ 00\ 00\ 00 = 40\ 40\ 00\ 00$
Dit is $1 + 2$ met als resultaat 3.

$00\ f0\ 00\ 00 + 40\ 00\ 00\ 00 = 00\ 70\ 00\ 00$
Hierbij wordt bij een klein, genormaliseerd getal het kleinst genormaliseerd getal met negatief teken opgeteld. Het resultaat is gedenormaliseerd.

$7f\ 00\ 00\ 00 + 7e\ ff\ ff\ ff = 7f\ 80\ 00\ 00$
Twee grote getallen met als resultaat oneindig.

Voor vermenigvuldigen:

$7f\ 7f\ ff\ ff * 00\ 00\ 00\ 03 = 35\ bf\ ff\ ff$
Het grootst mogelijke getal maal een zeer klein (gedenormaliseerd) getal.

$00\ 80\ 00\ 00 * 3f\ 00\ 00\ 00 = 00\ 40\ 00\ 00$
Het kleinste genormaliseerde getal maal 0.5.

$40\ 00\ 00\ 00 * 7e\ ff\ ff\ ff = 7f\ 7f\ ff\ ff$
Een groot getal wordt met 2

vermenigvuldigd waarbij het resultaat het grootst mogelijke getal is.

Voor delen:

$3f\ 80\ 00\ 00 / 00\ 00\ 00\ 00 = 7f\ 80\ 00\ 00$
1/0 geeft als resultaat oneindig.

$41\ d0\ 00\ 00 / 40\ e0\ 00\ 00 = 40\ 6d\ b6\ db\ 26/7$

$00\ 00\ 00\ 03 / 1f\ ff\ ff\ ff = 15\ 40\ 00\ 01$
Een zeer klein, gedenormaliseerd getal delen door een klein, genormaliseerd getal.

Tenslotte.

In deze aflevering hebben we ons bezig gehouden met de routines voor vermenigvuldigen en delen. In de inleiding heb ik al aangegeven dat de in volgende aflevering (ijs en weder dienende) we ons bezig zullen gaan houden met het inlezen en afdrucken.

De bijgevoegde programmas zijn geschreven op een Junior met PROTON-Senior DOS. Ze zouden zonder al te grote wijzigingen op elk ander 6502-systeem moeten kunnen lopen. Zo niet, dan hoor ik dat graag zodat we hiervoor een oplossing kunnen zoeken. Voor mensen die geen 6502-systeem gebruiken hoop ik dat de listings dermate veel commentaar bevatten dat de algoritmen toch duidelijk naar voren komen. Zoals reeds gezegd, bevatten de referenties [2..6] dezelfde routines voor 68000, 8088 en Z80.

Referenties.

- 1: Findley: 6502 Software Gourmet Guide And Cookbook
- 2: Hagen Völzke: Fliesskomma-Aritmetik und IEEE Specification; mc 10/88 blz. 123
- 3: Hagen Völzke: Fliesskomma-Aritmetik und IEEE Specification; mc 11/88 blz. 78
- 4: Hagen Völzke: Fliesskomma-Aritmetik und IEEE Specification; mc 12/88 blz. 91
- 5: Hagen Völzke: Fliesskomma-Aritmetik und IEEE Specification; mc 1/89 blz. 66
- 6: Hagen Völzke: Fliesskomma-Aritmetik und IEEE Specification; mc 2/89 blz. 65
- 7: H. Kremer: Numerische Mathematik fuer Hochfrequenz- und Elektrotechniker, Bd 2, Muenchen 1978

6502 Floating Point Package PROTON 650X ASSEMBLER V4.4 PAGE: 0001

```

0324 0351      .OPT LIST
0325 0351      ; *****
0326 0351      ;
0327 0351      ; Floating point package for the 6502 microprocessor Part II
0328 0351      ;
0329 0351      ; Written by G. van Opbroek
0330 0351      ; on JUNIOR with Proton Senior DOS
0331 0351      ; (c) Copyright 1989 Kim Gebruikersclub Nederland
0332 0351      ;
0333 0351      ; *****
0334 0351      ;
0335 0351      ; Note: Part II should be included behind line 322 of part I
0336 0351      ;
0337 000C      .EX1
0338 000C      ;
0339 000C      ; Define workspace on page zero
0340 000C      ;
0341 000C      COUNT      *==+1      ; Bit counter
0342 000D      WORK      *==+8
0343 0015      SAVE      *==+4
0344 0019      ;
0345 0351      .EX1
0346 0351      ;
0347 0351      ; Floating point multiplication:
0348 0351      ; based on: Hagen Volzke Fließkomma - Arithmetik und
0349 0351      ; IEEE-Spezifikation
0350 0351      ;
0351 0351      ; mc 11/88 page 78
0352 0351      ;
0353 0351      FMUL      ; Get parameters from stack
0354 0351      68          PLA      ; Get return address and save it
0355 0352      8500        STA RETADD
0356 0354      68          PLA
0357 0355      8501        STA RETADD+1
0358 0357      ;
0359 0357      A900        LDA #0      ; Clear signs
0360 0359      850B        STA SIGN2
0361 035B      8506        STA SIGN1
0362 035D      ;
0363 035D      ; Get second parameter from stack (4 byte)
0364 035D      ;
0365 035D      A207        LDX #<MAN2      ; Zero-page relative address
0366 035F      A004        LDY #4          ; 4 byte
0367 0361      68          FMULP1 PLA      ; Get byte from stack
0368 0362      9500        STA $0000,X    ; Store byte
0369 0364      E8          INX
0370 0365      88          DEY
0371 0366      D0F9        BNE FMULP1
0372 0368      ;
0373 0368      ; Get first parameter from stack (4 byte)
0374 0368      ;
0375 0368      A202        LDX #<MAN1      ; Zero-page relative address
0376 036A      A004        LDY #4          ; 4 byte
0377 036C      68          FMULP2 PLA      ; Get byte from stack
0378 036D      9500        STA $0000,X    ; Store byte
0379 036F      E8          INX
0380 0370      88          DEY
0381 0371      D0F9        BNE FMULP2
0382 0373      ;
0383 0373      ; Rotate operands to get exponents in one byte
0384 0373      ;
0385 0373      18          CLC

```

6502 Floating Point Package PROTON 650X ASSEMBLER V4.4 PAGE: 0002

```

0386 0374 A202          LDX #MAN1          ; Get zero-page relative address of MAN1
0387 0376 A005          LDY #5             ; Rotate 5 byte, result is:
0388 0378 200002        JSR ROTLEF         ; mmmmmmm0 mmmmmmmmm mmmmmmmmm eeeeeeee 0000
0389 037B ;                                     ;                                     0005
0390 037B 18            CLC
0391 037C A505          LDA EXP1           ; Exponent zero?
0392 037E F003          BEQ FMUL1         ; Yes, denormalized, Ror in 0
0393 0380 38            SEC               ; No, Ror in 1
0394 0381 C605          DEC EXP1          ; Compensate exponent
0395 0383 ;
0396 0383 A204 FMUL1     LDX #MAN1+2        ; Get zero-page relative address
0397 0385 A003          LDY #3             ; Rotate mantissa
0398 0387 200702        JSR ROTRIG        ; mmmmmmmmm mmmmmmmmm mmmmmmmmm eeeeeeee 0000
0399 038A ;                                     ;                                     0005
0400 038A 18            CLC
0401 038B A207          LDX #MAN2          ; Get zero-page relative address of MAN2
0402 038D A005          LDY #5             ; Rotate 5 byte, result is:
0403 038F 200002        JSR ROTLEF         ; mmmmmmm0 mmmmmmmmm mmmmmmmmm eeeeeeee 0000
0404 0392 ;                                     ;                                     0005
0405 0392 18            CLC
0406 0393 A50A          LDA EXP2           ; Exponent zero?
0407 0395 F003          BEQ FMUL2         ; Yes, denormalized, Ror in 0
0408 0397 38            SEC               ; No, Ror in 1
0409 0398 C60A          DEC EXP2          ; Compensate exponent
0410 039A ;
0411 039A A209 FMUL2     LDX #MAN2+2        ; Get zero-page relative address
0412 039C A003          LDY #3             ; Rotate mantissa
0413 039E 200702        JSR ROTRIG        ; mmmmmmmmm mmmmmmmmm mmmmmmmmm eeeeeeee 0000
0414 03A1 ;                                     ;                                     0005
0415 03A1 ; Calculate the new exponent
0416 03A1 ;
0417 03A1 18            CLC
0418 03A2 A505          LDA EXP1           ; Get first exponent
0419 03A4 650A          ADC EXP2           ; Add second one
0420 03A6 8513          STA WORK+6         ; Store low byte
0421 03A8 A900          LDA #0            ; Clear accumulator
0422 03AA 6900          ADC #0             ; Get carry
0423 03AC 8514          STA WORK+7         ; Store high byte
0424 03AE ;
0425 03AE 38            SEC               ; Now we have two times the bias
0426 03AF A513          LDA WORK+6         ; correct this
0427 03B1 E97C          SBC #127-3        ; Subtract bias - 3
0428 03B3 8513          STA WORK+6         ;
0429 03B5 A514          LDA WORK+7         ; High byte
0430 03B7 E900          SBC #0             ; Subtract borrow
0431 03B9 8514          STA WORK+7
0432 03BB ;
0433 03BB ; If the (biased) exponent < -25 then we have underflow
0434 03BB ; Result will be zero in this case
0435 03BB ;
0436 03BB 1009          BPL FMUL3          ; High byte >= 0 --> No underflow
0437 03BD A513          LDA WORK+6         ; Get low byte
0438 03BF C9E7          CMP #$E7          ; < -25?
0439 03C1 B003          BCS FMUL3
0440 03C3 4C9B04        JMP MULO          ; Yes, result is zero
0441 03C6 ;
0442 03C6 ; 24 Bits integer multiplication
0443 03C6 ;
0444 03C6 FMUL3
0445 03C6 ;
0446 03C6 ; Clear workspace
0447 03C6 ;

```


6502 Floating Point Package PROTON 650X ASSEMBLER V4.4 PAGE: 0003

```

0448 03C6 A900      LDA #$00
0449 03C8 A206      LDX #$06      ; We use 6 byte in the workspace
0450 03CA 950C FMUL4 STA WORK-1,X
0451 03CC CA        DEX
0452 03CD D0FB      BNE FMUL4
0453 03CF          ;
0454 03CF A218      LDX #24      ; 24 Bits multiplication
0455 03D1 860C      STX COUNT
0456 03D3          ;
0457 03D3 A209 MLOOP LDX #<MAN2+2 ; Rotate right mantissa 2
0458 03D5 A003      LDY #3      ; Rotate 3 byte
0459 03D7 200702    JSR ROTRIG
0460 03DA          ;
0461 03DA 900F      BCC MLOOP2    ; Last bit clear --> continue
0462 03DC          ;
0463 03DC          ; Last bit was set: Add mantissa 1 to result
0464 03DC          ;
0465 03DC 18        CLC          ; Clear carry for addition
0466 03DD          ;
0467 03DD A003      LDY #3      ; Add 3 byte of mantissa 1
0468 03DF A200      LDX #0
0469 03E1 B510 MLOOP1 LDA WORK+3,X ; Start at fourth byte
0470 03E3 7502      ADC MAN1,X
0471 03E5 9510      STA WORK+3,X
0472 03E7 E8        INX          ; Next byte
0473 03E8 88        DEY
0474 03E9 D0F6      BNE MLOOP1
0475 03EB          ;
0476 03EB          ; Rotate the result right; shift in the carry
0477 03EB          ; of the addition
0478 03EB          ;
0479 03EB A212 MLOOP2 LDX #<WORK+5 ; Start at MS byte
0480 03ED A006      LDY #6      ; Rotate 6 bytes
0481 03EF 200702    JSR ROTRIG
0482 03F2          ;
0483 03F2          ; Decrement the counter and continue
0484 03F2          ;
0485 03F2 C60C      DEC COUNT
0486 03F4 D0DD      BNE MLOOP    ; If counter > 0 then continue
0487 03F6          ;
0488 03F6          ; Test mantissa for zero result
0489 03F6          ;
0490 03F6 A206      LDX #6      ; Test 6 bytes
0491 03F8 B50C MLOOP3 LDA WORK-1,X ; Get byte
0492 03FA D006      BNE FMULNZ    ; Not zero
0493 03FC CA        DEX
0494 03FD D0F9      BNE MLOOP3
0495 03FF 4C9B04    JMP MULO
0496 0402          ;
0497 0402          FMULNZ
0498 0402          ;
0499 0402          ; Exponent of result <-24 --> underflow
0500 0402          ; Exponent of result < 0 --> denormalised exponent := 0
0501 0402          ;
0502 0402 A514      LDA WORK+7    ; MS byte of exponent
0503 0404 1017      BPL FMUL6    ; exponent >= 0
0504 0406 A513      LDA WORK+6
0505 0408 C9E8      CMP #$E8    ; < -24 ?
0506 040A B003      BCS FMUL5
0507 040C 4C9B04    JMP MULO    ; Underflow
0508 040F          ;
0509 040F          FMUL5

```

6502 Floating Point Package PROTON 650X ASSEMBLER V4.4 PAGE: 0004

```

0510 040F      ;
0511 040F      ; Rotate right mantissa until exponent = 0
0512 040F      ;
0513 040F 18    MLOOP4 CLC      ; Shift in zero
0514 0410 A212   LDX #<WORK+5    ; Start at MS byte
0515 0412 A006   LDY #6          ; Shift 6 byte
0516 0414 200702 JSR ROTRIG      ; Rotate right
0517 0417 E613   INC WORK+6      ; increment exponent
0518 0419 30F4   BMI MLOOP4
0519 041B      ;
0520 041B E614   INC WORK+7      ; MS byte was $FF; clear it.
0521 041D      ;
0522 041D A514   FMUL6 LDA WORK+7 ; Exponent zero ?
0523 041F D004   BNE FMUL7
0524 0421 A513   LDA WORK+6
0525 0423 F01E   BEQ MROUND      ; Yes round result
0526 0425      ;
0527 0425      ; Rotate result left until MS bit = 1
0528 0425      ; or exponent = zero
0529 0425      ;
0530 0425 A20D   FMUL7 LDX #<WORK ; Start at LS byte
0531 0427 A006   LDY #6          ; Shift 6 byte
0532 0429 200002 JSR ROTLEF
0533 042C B00E   BCS FMUL8      ; Carry set --> normalized
0534 042E      ;
0535 042E      ; Decrement exponent.....
0536 042E      ;
0537 042E A513   LDA WORK+6
0538 0430 E900   SBC #0          ; Carry was clear!
0539 0432 8513   STA WORK+6
0540 0434 A514   LDA WORK+7
0541 0436 E900   SBC #0
0542 0438 8514   STA WORK+7      ; Carry is set
0543 043A      ;
0544 043A B0E1   BCS FMUL6      ; Test exponent and start again
0545 043C      ;
0546 043C A212   FMUL8 LDX #<WORK+5 ; We have rotated one to much
0547 043E A006   LDY #6          ; correct
0548 0440 200702 JSR ROTRIG
0549 0443      ;
0550 0443      ; Round the result by adding $00 $00 $00 $80
0551 0443      ;
0552 0443 A202   MROUND LDX #2    ; Start at the third byte
0553 0445 A004   LDY #4          ; Four bytes
0554 0447 A980   LDA #$80
0555 0449 18    CLC
0556 044A 750D   MLOOP5 ADC WORK,X
0557 044C 950D   STA WORK,X
0558 044E A900   LDA #$0        ; Clear accu
0559 0450 E8     INX            ; Next byte
0560 0451 88     DEY            ; Decrement counter
0561 0452 D0F6   BNE MLOOP5
0562 0454      ;
0563 0454 9014   BCC FMUL9      ; If carry is set then
0564 0456 A212   LDX #<WORK+5    ; Rotate mantissa right
0565 0458 A004   LDY #4          ; four bytes
0566 045A 200702 JSR ROTRIG
0567 045D      ;
0568 045D 18     CLC
0569 045E A513   LDA WORK+6      ; Increment exponent
0570 0460 6901   ADC #1
0571 0462 8513   STA WORK+6

```

6502 Floating Point Package PROTON 650X ASSEMBLER V4.4 PAGE: 0005

```

0572 0464 A514          LDA WORK+7
0573 0466 6900          ADC #0
0574 0468 8514          STA WORK+7
0575 046A          ;
0576 046A A514          FMUL9  LDA WORK+7          ; Exponent > 255 ?
0577 046C F003          BEQ FMUL10          ; No, continue
0578 046E 4CA704        JMP MULOVL          ; Yes, overflow
0579 0471          ;
0580 0471 A513          FMUL10 LDA WORK+6          ; Get exponent
0581 0473 F007          BEQ FMULRE          ; 0 --> denormalised, exit
0582 0475 A210          LDX #<WORK+3          ; Normalised, shift MS bit out
0583 0477 A003          LDY #3          ; Start at fourth byte
0584 0479 200002        JSR ROTLEF
0585 047C          ;
0586 047C A506          FMULRE LDA SIGN1          ; Get sign
0587 047E 450B          EOR SIGN2          ; Calculate sign of result
0588 0480 8514          STA WORK+7          ; Store in result
0589 0482          ;
0590 0482 A214          LDX #<WORK+7          ; Start at eight byte
0591 0484 A005          LDY #5
0592 0486 200702        JSR ROTRIG
0593 0489          ;
0594 0489 A206          LDX #6          ; Start at WORK + 6
0595 048B A004          LDY #4          ; 4 byte
0596 048D B50D          MLOOP6  LDA WORK,X
0597 048F 48          PHA          ; Push result
0598 0490 CA          DEX
0599 0491 88          DEY
0600 0492 D0F9          BNE MLOOP6
0601 0494          ;
0602 0494 A501          LDA RETADD+1
0603 0496 48          PHA
0604 0497 A500          LDA RETADD
0605 0499 48          PHA
0606 049A 60          RTS
0607 049B          ;
0608 049B          ; Zero result --> Exponent zero,
0609 049B          ; mantissa zero
0610 049B          ;
0611 049B A900          MULO    LDA #0
0612 049D A208          LDX #8
0613 049F 950C          MLOOP7  STA WORK-1,X
0614 04A1 CA          DEX
0615 04A2 D0FB          BNE MLOOP7
0616 04A4 4C7C04        JMP FMULRE          ; Save result and exit
0617 04A7          ;
0618 04A7          ; Overflow --> exponent = $FF
0619 04A7          ; mantissa = 0
0620 04A7          ;
0621 04A7 A900          MULOVL  LDA #0          ; Clear workspace
0622 04A9 A208          LDX #8          ; 8 byte
0623 04AB 950C          MLOOP8  STA WORK-1,X
0624 04AD CA          DEX
0625 04AE D0FB          BNE MLOOP8
0626 04B0 A9FF          LDA #$FF          ; Exponent
0627 04B2 8513          STA WORK+6
0628 04B4 4C7C04        JMP FMULRE
0629 04B7          ;
0630 04B7          ; Floating point division:
0631 04B7          ; based on: Hagen Volzke Fließkomma - Arithmetik und
0632 04B7          ; IEEE-Spezifikation
0633 04B7          ;

```

6502 Floating Point Package PROTON 650X ASSEMBLER V4.4 PAGE: 0006

```

0634 04B7      ;          mc 11/88 page 78
0635 04B7      ;
0636 04B7      FDIV          ; Get parameters from stack
0637 04B7 68          PLA          ; Get return address and save it
0638 04B8 8500      STA RETADD
0639 04BA 68          PLA
0640 04BB 8501      STA RETADD+1
0641 04BD      ;
0642 04BD A900      LDA #0          ; Clear signs
0643 04BF 850B      STA SIGN2
0644 04C1 8506      STA SIGN1
0645 04C3      ;
0646 04C3      ; Get second parameter from stack (4 byte)
0647 04C3      ;
0648 04C3 A207      LDX #<MAN2      ; Zero-page relative address
0649 04C5 A004      LDY #4          ; 4 byte
0650 04C7 68          FDIVP1      PLA          ; Get byte from stack
0651 04C8 9500      STA $0000,X    ; Store byte
0652 04CA E8          INX
0653 04CB 88          DEY
0654 04CC D0F9      BNE FDIVP1
0655 04CE      ;
0656 04CE      ; Get first parameter from stack (4 byte)
0657 04CE      ;
0658 04CE A202      LDX #<MAN1      ; Zero-page relative address
0659 04D0 A004      LDY #4          ; 4 byte
0660 04D2 68          FDIVP2      PLA          ; Get byte from stack
0661 04D3 9500      STA $0000,X    ; Store byte
0662 04D5 E8          INX
0663 04D6 88          DEY
0664 04D7 D0F9      BNE FDIVP2
0665 04D9      ;
0666 04D9      ; Rotate operands to get exponents in one byte
0667 04D9      ;
0668 04D9 18          CLC
0669 04DA A202      LDX #<MAN1      ; Get zero-page relative address of MAN1
0670 04DC A005      LDY #5          ; Rotate 5 byte, result is:
0671 04DE 200002    JSR ROTLEF      ; 00000000 00000000 00000000 eeeeeeee 0000
0672 04E1      ;          ; 0005
0673 04E1 18          CLC
0674 04E2 A505      LDA EXP1        ; Exponent zero?
0675 04E4 F003      BEQ FDIV1      ; Yes, denormalized, Ror in 0
0676 04E6 38          SEC          ; No, Ror in 1
0677 04E7 C605      DEC EXP1        ; Compensate exponent
0678 04E9      ;
0679 04E9 A204      FDIV1      LDX #<MAN1+2    ; Get zero-page relative address
0680 04EB A003      LDY #3          ; Rotate mantissa
0681 04ED 200702    JSR ROTRIC      ; 00000000 00000000 00000000 eeeeeeee 0000
0682 04F0      ;          ; 0005
0683 04F0 18          CLC
0684 04F1 A207      LDX #<MAN2      ; Get zero-page relative address of MAN2
0685 04F3 A005      LDY #5          ; Rotate 5 byte, result is:
0686 04F5 200002    JSR ROTLEF      ; 00000000 00000000 00000000 eeeeeeee 0000
0687 04F8      ;          ; 0005
0688 04F8 18          CLC
0689 04F9 A50A      LDA EXP2        ; Exponent zero?
0690 04FB F003      BEQ FDIV2      ; Yes, denormalized, Ror in 0
0691 04FD 38          SEC          ; No, Ror in 1
0692 04FE C60A      DEC EXP2        ; Compensate exponent
0693 0500      ;
0694 0500 A209      FDIV2      LDX #<MAN2+2    ; Get zero-page relative address
0695 0502 A003      LDY #3          ; Rotate mantissa

```

6502 Floating Point Package PROTON 650X ASSEMBLER V4.4 PAGE: 0007

```

0696 0504 200702      JSR ROTRIG      ; mmmmmmmmm mmmmmmmmm mmmmmmmmm eeeeeeee 0000
0697 0507      ;                               0005
0698 0507      ; Calculate the new exponent
0699 0507      ;
0700 0507 38          SEC
0701 0508 A505      LDA EXP1          ; Get first exponent
0702 050A E50A      SBC EXP2          ; Subtract the second one
0703 050C 8513      STA WORK+6       ; Store low byte
0704 050E A900      LDA #0           ; Clear accumulator
0705 0510 E900      SBC #0           ; Subtract the borrow
0706 0512 8514      STA WORK+7       ; Store the high byte
0707 0514      ;
0708 0514 18          CLC              ; We have also subtracted the bias
0709 0515 A513      LDA WORK+6       ; Correct this
0710 0517 697F      ADC #$7F         ; The bias
0711 0519 8513      STA WORK+6       ;
0712 051B A514      LDA WORK+7       ; High byte
0713 051D 6900      ADC #0           ; Get the carry
0714 051F 8514      STA WORK+7
0715 0521      ;
0716 0521      ; If the (biased) exponent < -24 then we have underflow
0717 0521      ; Result will be zero in this case
0718 0521      ;
0719 0521 1009      BPL FDIV3        ; High byte >= 0 --> No underflow
0720 0523 A513      LDA WORK+6       ; Get low byte
0721 0525 C9E8      CMP #$E8         ; < -24?
0722 0527 B003      BCS FDIV3
0723 0529 4C9B04    JMP MULO         ; Yes, result is zero (use part of FMUL)
0724 052C      ;
0725 052C      FDIV3
0726 052C      ;
0727 052C      ;
0728 052C      ; If divisor = 0 then error !
0729 052C      ;
0730 052C A204      LDX #4           ; Test 4 bytes
0731 052E B506      DLOOP1 LDA MAN2-1,X ; Get byte
0732 0530 D006      BNE FDIVN1       ; Not zero
0733 0532 CA        DEX
0734 0533 D0F9      BNE DLOOP1
0735 0535 4CA704    JMP MULOVL       ; Return overflow (use part of FMUL)
0736 0538      ;
0737 0538      FDIVN1
0738 0538      ;
0739 0538      ; If dividend = 0 then result = zero
0740 0538      ;
0741 0538 A204      LDX #4           ; Test 4 bytes
0742 053A B501      DLOOP2 LDA MAN1-1,X ; Get byte
0743 053C D006      BNE FDIVN2       ; Not zero
0744 053E CA        DEX
0745 053F D0F9      BNE DLOOP2
0746 0541 4C9B04    JMP MULO         ; Return zero (use part of FMUL)
0747 0544      ;
0748 0544      FDIVN2
0749 0544      ;
0750 0544      ;
0751 0544      ; 24 bits integer division
0752 0544      ;
0753 0544      ;
0754 0544      ; Clear workspace
0755 0544      ;
0756 0544 A900      LDA #$00
0757 0546 A206      LDX #$06          ; We use 6 byte in the workspace

```

6502 Floating Point Package PROTON 650X ASSEMBLER V4.4 PAGE: 0008

```

0758 0548 950C FDIV4 STA WORK-1,X
0759 054A CA DEX
0760 054B DOFB BNE FDIV4
0761 054D ;
0762 054D A203 LDX #3 ; Shift MAN1 and MAN2 by 1 byte
0763 054F B501 DLOOP3 LDA MAN1-1,X
0764 0551 9502 STA MAN1,X
0765 0553 B506 LDA MAN2-1,X
0766 0555 9507 STA MAN2,X
0767 0557 CA DEX
0768 0558 DOF5 BNE DLOOP3
0769 055A ;
0770 055A A900 LDA #0 ; Clear LS byte
0771 055C 8502 STA MAN1
0772 055E 8507 STA MAN2
0773 0560 A205 LDX #MAN1+3 ; Shift dividend result = 0fff .... fff
0774 0562 A004 LDY #4 ; 4 byte
0775 0564 18 CLC
0776 0565 200702 JSR ROTRIG
0777 0568 ;
0778 0568 A20A LDX #MAN2+3 ; Shift divisor result = 0fff .... fff
0779 056A A004 LDY #4 ; 4 byte
0780 056C 18 CLC
0781 056D 200702 JSR ROTRIG
0782 0570 ;
0783 0570 ; Add 25 to the exponent, result > 0 !
0784 0570 ;
0785 0570 18 CLC
0786 0571 A513 LDA WORK+6
0787 0573 6919 ADC #25
0788 0575 8513 STA WORK+6
0789 0577 A514 LDA WORK+7
0790 0579 6900 ADC #0 ; Add carry
0791 057B 8514 STA WORK+7
0792 057D ;
0793 057D ; Shift left the divisor until bit 30 is set
0794 057D ;
0795 057D 18 CLC ; Shift in zero
0796 057E 240A DLOOP4 BIT MAN2+3 ; Copy bit 30 to V
0797 0580 7013 BVS DLOOP
0798 0582 A207 LDX #MAN2
0799 0584 A004 LDY #4 ; Shift 4 byte
0800 0586 200002 JSR ROTLEF
0801 0589 ;
0802 0589 A513 LDA WORK+6 ; Increment exponent
0803 058B 6901 ADC #1 ; Carry was clear
0804 058D A514 LDA WORK+7
0805 058F 6900 ADC #0 ; Add carry
0806 0591 8514 STA WORK+7 ; Carry is clear .....
0807 0593 ;
0808 0593 90E9 BCC DLOOP4
0809 0595 ;
0810 0595 ; Start division
0811 0595 ;
0812 0595 A204 DLOOP LDX #4 ; Save dividend
0813 0597 B501 DLOOP5 LDA MAN1-1,X
0814 0599 9514 STA SAVE-1,X
0815 059B CA DEX
0816 059C DOF9 BNE DLOOP5
0817 059E ;
0818 059E ; Subtract divisor from dividend
0819 059E ;

```

DE P KENNER

ALGEMEEN

6502 Floating Point Package

PROTON 650X ASSEMBLER V4.4 PAGE: 0009

```
0820 059E A004          LDY #4          ; Four byte
0821 05A0 A200          LDX #0          ; Start at 1s byte
0822 05A2 38           SEC
0823 05A3 B502 DLOOP6  LDA MAN1,X
0824 05A5 F507          SBC MAN2,X
0825 05A7 9502          STA MAN1,X
0826 05A9 E8           INX
0827 05AA 88           DEY
0828 05AB D0F6          BNE DLOOP6
0829 05AD          ;
0830 05AD 08           PHP          ; Save the carry !
0831 05AE          ;
0832 05AE B009          BCS FDIV5      ; If no borrow then the subtraction was
0833 05B0 A204          LDX #4          ; Else restore dividend
0834 05B2 B514 DLOOP7  LDA SAVE-1,X
0835 05B4 9501          STA MAN1-1,X
0836 05B6 CA           DEX
0837 05B7 D0F9          BNE DLOOP7
0838 05B9          ;
0839 05B9 28 FDIV5     PLP          ; Restore carry
0840 05BA A003          LDY #3          ; Shift carry in result (3 byte)
0841 05BC A210          LDX #<WORK+3
0842 05BE 200002        JSR ROTLEF
0843 05C1 08           PHP          ; Save carry for the future
0844 05C2          ;
0845 05C2 18           CLC
0846 05C3 A004          LDY #4          ; Shift the dividend (4 byte)
0847 05C5 A202          LDX #<MAN1
0848 05C7 200002        JSR ROTLEF
0849 05CA          ;
0850 05CA 38           SEC          ; Decrement exponent
0851 05CB A513          LDA WORK+6
0852 05CD E901          SBC #1
0853 05CF 8513          STA WORK+6
0854 05D1 850D          STA WORK      ; Test on zero result
0855 05D3 A514          LDA WORK+7
0856 05D5 E900          SBC #0
0857 05D7 8514          STA WORK+7
0858 05D9 050D          ORA WORK      ; Test on zero result
0859 05DB D00B          BNE FDIV6
0860 05DD 28           PLP          ; Restore carry (MS bit)
0861 05DE A212          LDX #<WORK+5
0862 05E0 A005          LDY #5          ; Shift 5 byte
0863 05E2 200702        JSR ROTRIG    ; Shift in MS-bit
0864 05E5 4C7C04        JMP FMULRE   ; Denormalized, return
0865 05E8          ;
0866 05E8 28 FDIV6     PLP          ; Restore carry
0867 05E9 90AA          BCC DLOOP    ; MS bit clear, continue
0868 05EB A212          LDX #<WORK+5
0869 05ED A005          LDY #5          ; Shift 5 byte
0870 05EF 200702        JSR ROTRIG    ; Shift in MS-bit
0871 05F2 4C4304        JMP MROUND   ; Round result and return (use part of
0872                                .END
```

ERRORS: 0000

<0000>

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 CHARGVID2 (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 geheu-

gen 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 tekstschermbekend TELETXT-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.

NF = 0 t/m 3 hebben geen functie.

De routine op 400 produceert nog geen geluid, hieraan wordt gewerkt en t.z.t. zal er een printje met de geluidchip (6581) beschikbaar komen.

Volgt nu een opsomming van de diverse subroutine's:

- 100 Schakel om naar tekstbedrijf en wis het scherm
- 110 Verplaats de cursor naar positie HO,VE
- 120 Registreer de cursorpositie in HO,VE
- 150 Print op opvallende wijze drie spaties;SR\$;drie spaties
- 200 Geef een eventueel ingedrukte toets in IN\$ en IN
speciale codes:
28 = cursor links
29 = cursor rechts
30 = cursor omlaag
31 = cursor omhoog
127 = wis/delete
verder geldt altijd: 32 <= IN <= 95
- 210 Wacht tot toetsindruk en geef deze in IN\$ en IN
- 220 Geef in IN de code van wat op scherm-positie HO,VE te zien is
N.B.: zie de toevoeging bij subroutine 200
- 250 Geef een piepje als attentiessein
- 260 Geef een randomgetal in RV zodat 0 <= RV < 1
- 270 Doe "garbage collect" en geef in FR het totaal aantal vrije bytes
- 280 Schakel de stoptoets in (FR=0) of uit (FR=1)
- 300 Geef in SR\$ de tekstvorm van SR, zonder spaties
- 310 Geef in SR\$ de tekstvorm van SR, geformatteerd conform CT en CN
- 330 Verander alle kleine letters in SR\$ in hoofdletters
- 350 Stuur SR\$ naar de printer
- 360 Sluit de printerregel af met wagenterug en papieropvoer
- 400 Maak een toon volgens SP, SD en SV
SP tooncode: 0=uiterst laag,
60="Centrale C",
127=uiterst hoog
SD toonduur in stappen van 0,1 seconde
SV volume: 0=stil, 7=normaal, 15=maximum
- 450 Wacht maximaal SD*0,1 seconden op een toetsindruk na afloop: toets in IN en IN\$ conform de toevoeging bij subr. 200

SD verlaagd tot moment van toetsindruk of SD=0

- 500 Open bestand NF\$ volgens code NF
NF=even: invoer,
NF=oneven: uitvoer
NF= 0 of 1 : BASICODE-cassette
NF= 2 of 3 : eigen systeemgeheugen
NF= 4 of 5 : diskette
NF= 6 of 7 : diskette
- 540 Lees in IN\$ uit geopend bestand NF en in IN de status
IN=0: alles OK,
IN=1: einde bestand,
IN=-1: foutcode
- 560 Voer SR\$ uit naar het geopende bestand met code NF
- 580 Sluit het bestand met code NF af
- 600 Schakel om naar grafisch bedrijf en wis het scherm
- 620 Plot een punt op positie HO,VE in kleur CN
- 630 Trek een lijn naar punt HO,VE in kleur CN
N.B.: 0 <= HO < 1 en 0 <= VE < 1
- 650 Print SR\$ als tekst vanaf grafische positie HO,VE
- 950 Beeindig het programma en schakel de machine naar normaal bedrijf

REGENUMMERS IN BASICODE-3

De voorgeschreven indeling voor de regelnommers in BASICODE-3 is:

0 - 999: De standaardroutines

1000: Verplicht van volgende vorm:

1000 A=<stringruimte>:
GOTO 20:REM programma-naam

1010 - 19999: Het hoofdprogramma

20000 - 24999: Eventuele machine-afhankelijke subroutines

25000 - 29999: DATA regels

30000 - 31999: REM-regels met korte programma-beschrijving en eventuele literatuurverwijzingen

32000 - 32767: REM-regels met de naam en het adres van de maker

Oudenbosch 29/03/1989

```

*****
*****      EEEE      CCCC      6      5555      *****
*****      E      C      6      5      *****
*****      E      C      6      5      *****
*****      EEEE      C      6666      5555      *****
*****      E      C      6 6      5 55      *****
*****      E      C      6 6      5 5      *****
*****      EEEE      CCCC      66      5555      *****
*****
*****

```

Eproms programmeren nu ook voor de >>> EC 65 <<<

Eindelijk is het dan zo ver de lang verwachte Eprom Programmer ook voor de Elektuur cq Octopus 65, Samson 65 of ander verwante namen.

Nu dan ook de hardware compleet met print en diverse (E)proms en een pal is uit gebracht ben ik naar aanleiding van een oproep op een bijeenkomst om de software die oorspronkelijk voor de Dos 65 is geschreven om te schrijven naar de Elektuur software onder OHIO OS-65 D V3.3 disk operation system.

Daar het nog vrij gecompliceerd is om deze software om te schrijven is het mij Ton Smits en Paul van Oers gelukt om dit toch compatible met DOS 65 te houden.

De software nl. 2 schijven geheel onafhankelijk van elkaar werkend.

Ten eerste een programma waar mee je de hardware en diverse voedingsspanningen mee kunt controleren testen en afregelen.

Ten tweede een programma waarmee je de eproms mee kan programmeren.

Daar dit VOORLOPIGE VERSIES zijn maar wel werken ben ik nog druk doende bezig om enkele bugs die nog in de beide programma's zitten er uit te halen.

De 2 schijven 40 of 80 tracks 1 en/of 2 MHZ wel op geven bij Uw bestelling zijn alleen te verkrijgen als je de print met (E)proms en pal want zonder deze essensieele onderdelen werkt de Eprommer niet.

De Monitor Eprom hoeft niet perse gewijzigd te worden, maar is wel aan te bevelen om dit te doen, nl dan wordt de EP kaart gelijk geïnitieerd bij het aanzetten van de computer.

Daar er een wild groei onder de Elektuur Monitor Eprom in de jaren is ontstaan is het vrij moeilijk om een goede oplossing te geven namelijk iedereen wijzigde naar eigen inzicht zijn of haar Eprom ten gevolge niemand was of is nog compatible met elkaar, om hier een eind aan te maken heb ik de originele Eprom van ELEKTUUR gedeeltelijk veranderd.

De noodzakelijke wijzigingen zijn als volgt:

Als je nog de originele eprom van Elektuur met bij behorende sources bezit is het heel eenvoudig.

Onder staande gegevens gelden alleen voor de source geassembleerd onder MICRO-ADE deze assembler staat op schijf 5 en 5 a van de OHIO diskettes.

In de source vindt je in het begin van het deel SAMVID de diverse tabellen om verschillende beeldformaten te kiezen.

De wijzigingen zijn als volgt:

In de originele source vindt je een stuk source genaamd :

MOVE THE CRT FILE FROM ROM TO RAM genaamt met label: MOVCRT (F3E1)

Dit doet niets anders dan de data die in Eprom monitor staat te copiereen naar RAM , daar er verschillende tabellen mee te kiezen zijn moet men er enige wijzigingen in aanbrengen zodat er nog maar EEN tabel mee wordt gekozen, namelijk het beelscherm formaat 80 * 24

ER STAAT IN DE SOURCE:

ER KOMT TE STAAN IN DE SOURCE:

LABEL	MNEMONICS	COMMET	LABEL	MNEMONICS	COMMET
MOVCRT	LDAIM REFRAM	START AT HERE	MOVCRT	LDAIM REFRAM	START AT HERE
	LDXIM REFRAM	/256		LDXIM REFRAM	/256
	STA RAMBEG			STA RAMBEG	
	STX RAMBEG	+01		STX RAMBEG	+01
	LDAIM \$00			TAY	
	TAY			LDXIM \$00	GET THE CURRENT
	LDX FORMAT	GET THE CURR.		TAX	FORMAT
	BEQ MCRTB	FORMAT.	MCRTC	LDAX CRTINA	
MCRTA	CLC			STAY TABLE	MOVE THE TABLE
	ADCIM \$12	COMPUTE THE INDEX		INX	
	DEX			INY	
	BNE MCRTA	ALL DONE ?		CPYIM \$12	COMPUTE THE INDEX
MCRTB	TAX			BNE MCRTC	ALL DONE ?
MCRTC	LDAX CRTINA			DEY	
	STAY TABLE	MOVE THE TABLE		LDAY TABLE	SET SCREEN PARA-
	INX				METERS
	INY			STA LPSCR	
	CPYIM \$12	COMPUTE THE INDEX		DEY	
	BNE MCRTC			LDAY TABLE	
	DEY			STA CHAPLN	
	LDAY TABLE	SET SCREEN PARA-		RTS	
	STA LPSCR	METERS			
	DEY				
	LDAY TABLE				
	STA CHAPLN				
	RTS				

 *** CRT TIMING TABLES ***

CRTINA	=\$ 7F 50 60 08 21 06 18 1C	CRTINA	=\$ 7F 50 60 08 21 06 18 1C
	=\$ 00 08 00 09 00 00 00 00		=\$ 00 08 00 09 00 00 00 00
	=\$ 50 18		=\$ 50 18
	=\$ 64 40 52 05 22 00 18 1C	EPROM	LDYIM \$FF GET VALUE DDR REG.
	=\$ 00 08 00 09 00 00 00 00		LDAIM \$00 AND VALUE DDR ACCES.
	=\$ 40 18		STA PCRA SET DDRA ACCES.
			STA PCRB SET DDRB ACCES.
	=\$ 46 30 3A 05 16 0E 0C 12		STY PORa SET PORT A OUTPUT

```

=$ 00 0C 00 09 00 00 00 00
=$ 30 0C

=$ 38 18 26 05 22 00 18 1C
=$ 00 08 00 09 00 00 00 00
=$ 18 18

```

```

STY   PORB SET PORT B OUTPUT
LDAIM $04 VALUE FOR PORT ACC.
STA   PCRA SET PORT A ACC.
STA   PCRB SET PORT B ACC.
INY
STY   PORA SET ALL ADDRES LINES
STY   PORB TO ZERO
TYA   A=0
DEY   Y=$FF
STY   VDDRA SET VIA PORT A
STA   VORANH SET DATA BUS
STY   VDDRB SET VIA PORT B
STA   VORB SET CONTR TO ZERO
JMP   INICEN INIT CENTRONICS
=$    FF FF FF FF FF FF FF FF
=$    FF FF FF FF FF FF FF FF
=$    FF

```

Verder in de bij de pointers and temps :
toevoegen na FILES * KPDOX +01 ; CCP OF ASM/WP BUFFER

```

EPBASE * $E180 SET TO BASE EPROMMER CARD
VIA    * EPBASE
VORB   * VIA VIA OUTPUT REGISTER B
VORA   * VIA +01 VIA OUTPUT REGISTER A
VDDRB  * VIA +02 DATA DIR. REG. PORT B
VDDRA  * VIA +03 DATA DIR. REG. PORT A
VORANH * VIA +0F VIA PORT A. NO HAND SHAKE
PIA    * EPBASE +10 BASE ADDRESS OF PIA
PORA   * PIA PORT A DATA DIR REG A
PCRA   * PIA +01 CONTROL REGISTER A
PORB   * PIA +02 PORT B DATA DIR REG B
PCRB   * PIA +03 CONTROL REGISTER B

```

INICEN * \$F6F1 RETURN TO INITIALIZE CENTRONICS

Als je deze gegevens heb ingevoerd zo als hier boven en je assembleerd dit dan moet dit gaan van \$ F000 tot \$ F45C is totaal 1117 bytes.

Dan het volgende gedeelte van de MONITOR EPROM source genoemd: SAMMON.

In de EQU list external addresses TOEVOEGEN NA

```

KERNEL * $2547 SWAP $0213...$0216 AND GO TO KERNEL
EPROM  * $F41C INITIALIZE EPROMMER VIA AND PIA

```

Verder in de source onder de kop MONITOR met label INITPR (F811)
zoeken naar JSR INICEN INIT. CENTRONICS (F82B)
veranderen in JSR EPROM INITIALIZE EPROM CARD

Daar ik weet dat er nog verschillende bugs in het programma EP EN EPTTEST kunnen zitten oa in het programma EPTTEST met de space balk is niet uit een bepaalde testroutine te komen.
In EP een 27512 is nog niet te programmeren dit gaat bij het eerste bit al

fout verder kunnen er nog wel enkele andere storende fouten in het programma voorkomen, maar dit zal ik in de komende maanden waarschijnlijk tegen de winter wel verholpen hebben.
Graag wil ik bij deze vragen de mensen die de software voor de >>> EC-65 <<< bestellen hun klachten, bevindingen aan het onderstaande adres bekend te maken zodat tegen de winter alle bugs uit het programma zijn.

Gelieve deze te zenden aan :

T. SMITS
DE MEREN 39
4731 WB OUDENBOSCH

De definitieve versies worden tzt in de UP KENNER bekend gemaakt en zijn verkrijgbaar bij bovenstaand adres dit geldt alleen voor degene die de voorlopige versie in het bezit hebben.
In de totale prijs van de hardware zit nl de software inbegrepen zowel voor de versie voor DOS 65 als voor de >>> EC-65 <<< versie.
De totale prijs voor zowel voor de DOS 65 als voor de >>> EC 65 <<< versie include pal + proms + enkelzijdige hoofdprint + hulp print voor de programmeer voet en programatuur bedraagt FL 125,-
Mochten er programmeer problemen zijn dan ben ik wel bereid om de originele MONITOR SAMMON die ik zelf gebruik tegen kostprijs van porto + eventueel opgestuurde eeprom te programmeren.
De beschrijving die bij de EP en EPTTEST geleverd wordt kun je wat betreft de versie voor >>> EC-65 <<< aanhouden mochten er enkele veranderingen zijn ten opzichte van DOS 65 dan zal dit in de UP Kenner bekend gemaakt worden.
De jumpers voor de kaart selectie zijn het zelfde nl: \$ E180
verder is het start adres voor RAM default waarde \$1000 bij DOS 65 bij >>> EC-65 <<< verhoogd naar \$4000 i.v.m kernel op \$2300 ev.
Verder wil ik op deze plaats Nico de Vries bedanken voor het beschikbaar stellen van de software van EPTTEST EN EPTTEST PROGRAMMA.

Nieuwe prijzen voor EP hardware.

Eindelijk is het zover: de printen voor de EPROMprogrammer EP zijn leverbaar. De printen zijn echter wat duurder uitgevallen dan wij oorspronkelijk hebben voorzien. Daarom is in overleg met het bestuur gesloten de prijzen iets te verhogen.
Diegenen die reeds in het vorige jaar besteld hadden hebben uiteraard voor de oude prijs geleverd gekregen. De nieuwe prijzen gelden reeds vanaf 1 maart jongstleden.

De prijzen zijn nu:

Printen, floppy disk en componenten opstellingen: hfl. 85.-

Printen, floppy disk, componenten opstellingen, PROM, EPROM en PAL: hfl. 105.-

Printen, floppy disk, PROM, PAL en EPROM, manuals op papier: hfl. 125.-

Dit alles wordt veroorzaakt door het feit dat vooral de basisprint behoorlijk duurder is uitgevallen dan oorspronkelijk gecalculleerd. Daarentegen vielen de ZIF-print, de PALs, PROMs en EPROMs alsmede de kopieerkosten voor de manuals aanzienlijk voordeliger uit, reden waarom

HARDWARE

vooral de losse printen drastisch in prijs verhoogd moesten worden, en dat de prijsverhoging voor het complete pakket beperkt kon blijven. Het bestuur hoopt dat u begrip voor de verhoging zult kunnen opbrengen.

Namens het bestuur: N. de Vries.

EP: Errata.

Door: Nico de Vries.

Nu de definitieve versie er is, komen er ook wat kleine noodzakelijke wijzigingen boven water. Het blijkt aan te bevelen te zijn de L200 regelaars te voorzien van een kleine koeling. Als u alle regelaars op een klein stripje aluminium monteert, worden ze niet heet. Ongekoeld blijven de regelaars wel heel, maar u kunt uw vingers eraan branden.....
Positief is deze melding: tot op heden heeft nog geen der 12 bestellers met een hardwareprobleem gebeld. Soms gaat er toch nog wel eens iets goed.....

Computers..... (deel 3).

Door Gert van Opbroek
Bateweg 60
2481 AN Woubrugge
01729-8636

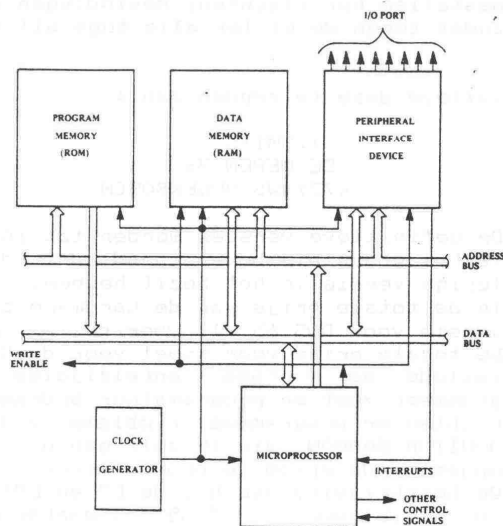
Inleiding.

In het eerste deel van de serie heb ik het gehad over de globale opbouw van de computer. In de tweede aflevering hebben we het geheugen van een computer bekeken. In deze aflevering wil ik het werkpaard, de CPU of processor, wat nader bekijken. Uiteraard bekijken we hoe de processor logisch in elkaar zit en werkt en niet hoe het elektronisch allemaal gaat. In de eerste plaats vindt ik dat niet echt interessant en in de tweede plaats weet ik het niet en ik zou ook niet weten waar ik dat zou kunnen vinden.

Tenslotte nog een opmerking over de voorbeelden die gegeven worden. Specifieke voorbeelden hebben altijd betrekking op een bepaald type processor; daar is natuurlijk niets aan te doen. Wel is het zo dat vrijwel alle microprocessors vergelijkbare mogelijkheden hebben en op een vergelijkbare manier werken. Een uitzondering hierop vormt misschien de ARM (Acorn Risc Machine). Dit is namelijk een echte RISC (Reduced Instruction Set Computer) die nogal afwijkt van de CISC- (Complex Instruction Set Computer) waaronder de andere processors vallen. Voor een vergelijking tussen CISC en RISC wordt verwezen naar [1]. De ARM is het hart van de ACORN ARCHIMEDES.

De Processor.

In figuur 1 is de opbouw van een microcomputer nogmaals weergegeven [2]. We zien de processor weer met de klok, het geheugen en de programmeerbare I/O. Tussen deze onderdelen lopen de adres- data- en control- of stuurbus. In de meeste systemen is de processor een IC (Integrated Circuit of geïntegreerde elektronische schakeling) met bijvoorbeeld 40 of 64 pootjes. Hoe breder de adres- en databus zijn, hoe meer pootjes de processor over het algemeen zal hebben. Het komt echter ook vrij vaak voor dat enkele pootjes voor twee signalen gebruikt worden. Dit kan bijvoorbeeld een adres- en een datasignaal zijn. Een signaal in de stuurbus geeft dan aan of het betreffende signaal een adres is of data. Deze techniek heet multiplexing en men zegt dan wel dat de processor een



Organization of Microcomputer System

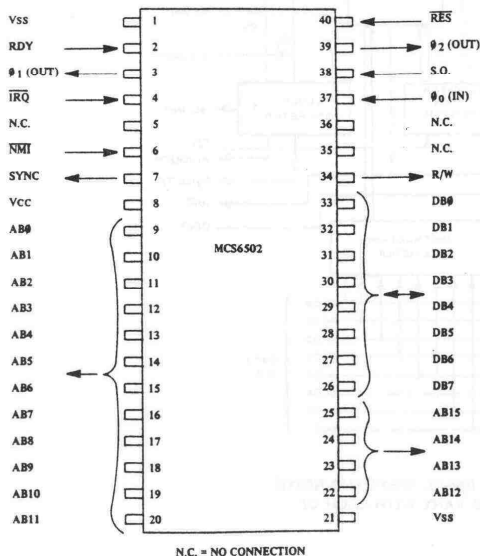
Figuur 1.

gemultiplexte bus heeft.

Een leuke anecdote is misschien wel het ontstaan van gemultiplexte bussen. Processors worden uiteraard door IC-fabrikanten zoals Intel en Motorola gemaakt. Nu is het maken van een gemultiplexte bus extra werk omdat er in het IC meer elektronische schakelingen nodig zijn om het multiplexen voor elkaar te krijgen. Dat doe je dus niet zomaar. In het begin hadden we voorname-lijk te maken met processors met 8 bits databus en 16 bit adresbus. Dat zijn dus al 24 pootjes. Daarbij komt dat de voeding (2 tot 4 pootjes), de aansluiting voor de klok (+/- 3 pootjes) en nog wat andere stuursignalen, kortom 40 pootjes is ruim voldoende. Later ontstond de behoefte aan bredere bussen, bijvoorbeeld 24 adreslijnen en 16 datalijnen. Dat krijg je met 40 pootjes nooit voor elkaar als je elk pootje z'n eigen functie geeft. Dan kun je natuurlijk het IC meer pootjes geven echter..... Meer pootjes aan het IC betekent ook dat bijvoorbeeld de testapparatuur in de fabriek een IC met meer aan-

sluitingen moet kunnen verwerken en die apparatuur was bijna allemaal ontwikkeld voor IC's met maximaal 40 pootjes; kortom de stap van IC's met 40 pootjes naar IC's met 64 pootjes bracht een relatief grote investering met zich mee, zo groot dat men in eerste instantie liever de schakelingen in het IC wat moeilijker maakte en de bussen ging multiplexen. Momenteel gelden deze begrenzings eigenlijk niet meer, er zijn nu IC's met meer dan honderd pootjes. Die worden dan verdeeld over bijvoorbeeld 32 adreslijnen en 32 data lijnen en nog een handvol stuurlijnen.

In figuur 2 [2] is schematisch weergegeven waar de 40 pootjes van een 6502 voor dienen. De belangrijkste zijn de adresbus (AB), de databus (DB), de klok (ϕ) en de voeding (Vcc, Vss). De overige signalen zijn onderdeel van de stuurbus en komen later aan de orde.



MCS6502 Pinout Designation

Figuur 2.

Intern is een processor, net als elk ander IC, opgebouwd uit min of meer normale onderdelen: transistoren, weerstanden en condensatoren. Met deze onderdelen zijn normale elektronische schakelingen gebouwd die echter niet op een printplaat gesoldeerd zijn, maar door middel van diverse diffusie- en opdamptechnieken in een stukje halfgeleidermateriaal (silicium) zijn aangebracht. Dit stukje silicium wordt de chip (het schijfje) genoemd.

Hoe één en ander in zijn werk gaat en hoe de schakeling van de diverse circuits is, valt buiten dit artikel. Wij beschouwen het IC vanaf nu als een black box (ze zijn toch ook bijna altijd zwart) met een aantal functies en een aantal aansluitingen. In de volgende paragrafen wordt dus de logische werking van de processor beschreven.

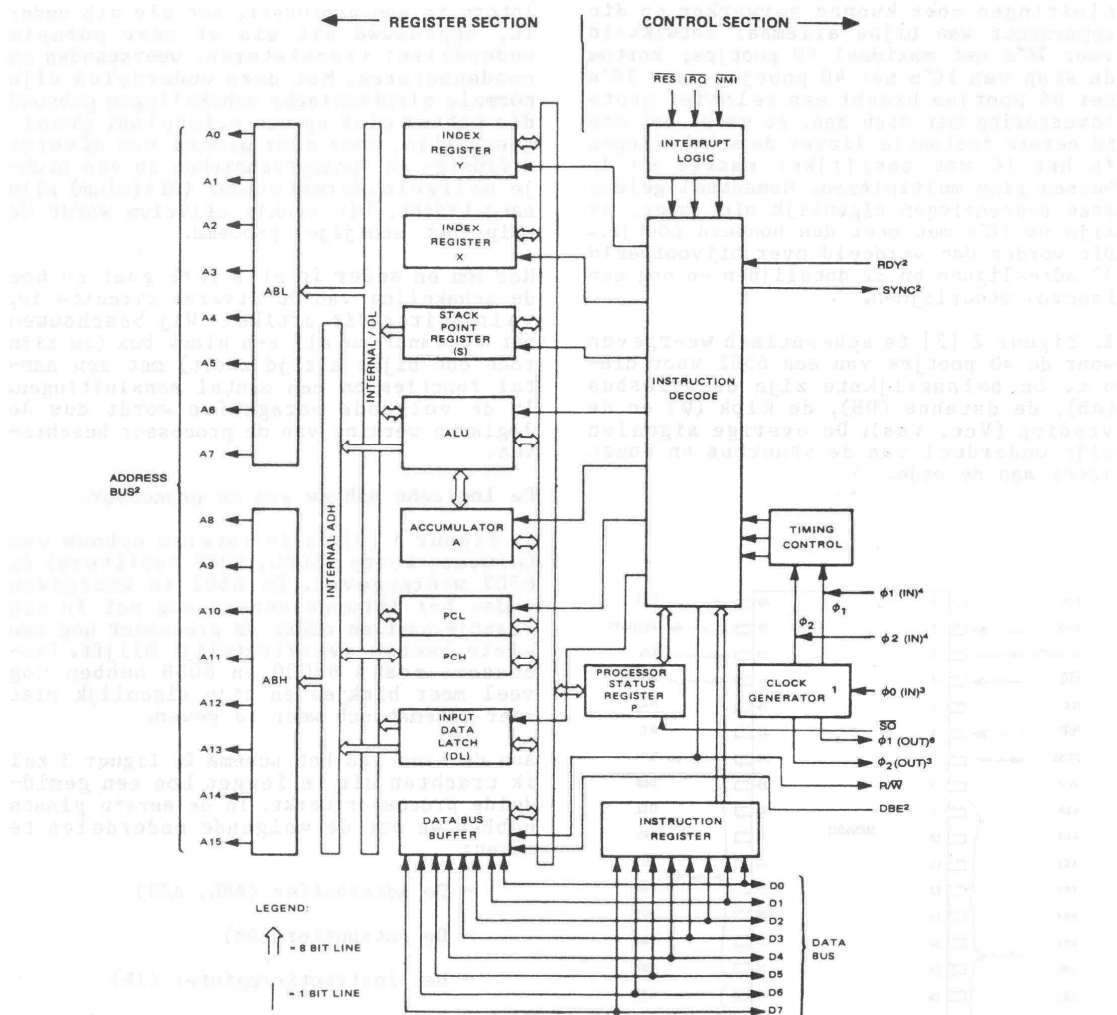
De logische opbouw van de processor.

In figuur 3 [3] is de interne opbouw van (alweer, sorry 68000, 8088 bezitters) de 6502 weergegeven. De 6502 is weergegeven omdat het interne schema nog net in een plaatje past en omdat de processor nog een klein beetje overzichtelijk blijft. Processors zoals 68000 en 8088 hebben nog veel meer blokjes en zijn eigenlijk niet meer schematisch weer te geven.

Aan de hand van het schema in figuur 3 zal ik trachten uit te leggen hoe een gemiddelde processor werkt. In de eerste plaats hebben we met de volgende onderdelen te maken:

- De adresbuffer (ABL, ABH)
- De databuffer (DB)
- Het instructieregister (IR)
- De programmateller (PCL, PCH)
- De instructie decoding

Verder komt in de volgende beschrijving de accumulator of accu naar voren. Dit is een zogenaamd register. In dit register komt het resultaat van een bewerking terecht. In het voorbeeld wordt steeds uitgegaan van de LOAD-instructie waarbij de accu met een bepaald getal geladen moet worden. Als voorbeeld van een instructie die allen intern werkt, is de CLEAR CARRY instructie gekozen. Deze instructie vult een bit (het zogenaamde carry-bit) in het processor status-register met de waarde 0.



- NOTE**
1. CLOCK GENERATOR IS NOT INCLUDED ON R6512, R6513, R6514 AND R6515.
 2. ADDRESSING CAPABILITY AND CONTROL OPTIONS VARY WITH EACH OF THE CPUs.
 3. R6502, R6503, R6504, R6505, R6506 AND R6507.
 4. R6512, R6513, R6514 AND R6515.
 5. R6512 ONLY.
 6. R6502 ONLY.

R650X and R651X Internal Architecture

Figuur 3

Zoals in het vorige deel al is aangegeven, staat het programma dat de processor uit moet voeren in het geheugen van de computer. Om dit programma uit te voeren, moet de processor dus de opdrachten (instructies) uit het geheugen lezen. Dit gaat als volgt:

- 1.0: Zet de inhoud van de programmateller in de adresbuffer
- 1.1: Maak het signaal R/W (in de stuurbus) hoog om aan het geheugen duidelijk te maken dat de processor wil lezen
- 1.2: Na enige tijd (op een niveauverandering van een kloksignaal) staat de inhoud van de geheugenlocatie op de databus en wordt ingelezen in het instructieregister
- 1.3: De inhoud van de programmateller wordt met 1 verhoogd en de instructie wordt door de processor gedecodeerd

Dit is het eerste deel van de bewerkingen die een processor uitvoert om een instructie uit te voeren. Dit eerste deel heet de "opcode fetch" of te wel het ophalen de instructie. Voor elke instructie is er een unieke code (bitpatroon) en aan de hand van deze code weet de processor wat hij moet gaan doen. Deze code heet "Opcode" en bestaat bij de 6502 8 bits (een byte). Er kunnen op een 6502 dus maximaal 256 instructies gecodeerd worden. Bij de 68000 is voor de opcode twee byte beschikbaar zodat deze processor in theorie ruim 65000 verschillende instructies zou kunnen hebben. Hoewel de 68000 zeer veel instructies heeft, zijn niet alle mogelijke opcodes in gebruik. Ook bij de 6502 is er nog enige ruimte in het instructieset die echter bij de 65C02 al weer enigzins is opgevuld.

Na het inlezen en decoderen van de instructie kan het voorkomen dat de processor alle gegevens voor het uitvoeren van de instructie tot zijn beschikking heeft. De processor kan dan zijn taak uitvoeren. (Voorbeeld: Maak de carry-vlag 0; CLC). In dit geval gaat de processor dus verder met:

- 2.0: Voer de instructie uit.

Nu zal het echter zeer vaak voorkomen dat er voor de uitvoering van de instructie meer gegevens nodig zijn. Een voorbeeld is de instructie: Zet het getal 10 in de accumulator (LDA #10). In het programma staat dan eerst de opdracht (bij 6502 \$A9

= LDA #) gevolgd door het getal 10. De processor gaat in dat geval op de volgende manier verder:

- 2.0: Hé het is LDA #, ik moet data ophalen; zet de inhoud van de programmateller in de adresbuffer
- 2.1: Maak R/W hoog
- 2.2: De inhoud van het geheugen komt terecht in de databuffer; kopieer deze naar de accumulator
- 2.3: Verhoog de programmateller met 1

Elke processor heeft een aantal verschillende mogelijk heden waarop hij bij de data kan komen. Dit noemen we de adressermogelijkheden (addressing modes). In het bovenstaande voorbeeld heb ik de directe adressering beschreven. Dit wil zeggen dat de data meteen na de opcode in het programma staat. Elke mij bekende processor heeft deze mogelijkheid.

Een tweede mogelijk die bij elke processor voorkomt is de absolute adressering. Dit wil zeggen dat na de opcode in het programma het complete adres van de data staat. Op een 6502 betekent dit dat na de opcode van 8 bits (1 byte) twee byte met het adres van de data staat. De processor handelt nu als volgt:

- 2.0: Verd.... absolute adressering; zet de inhoud van de programmateller in de adresbuffer
- 2.1: R/W
- 2.2: Het ingelezen byte kopiëren uit de databuffer naar de input data latch
- 2.3: De programmateller weer met 1 verhogen
- 3.0: Inhoud van de programmateller in de adresbuffer
- 3.1: R/W
- 3.2: Het ingelezen byte kopiëren uit de databuffer naar het hoge byte van de adresbuffer; de inhoud van de data latch gaat naar het lage byte van de adresbuffer
- 3.3: De programmateller (alweer) met 1 verhogen
- 4.0: Adresbuffers zijn gevuld

4.1: R/W

4.2: De inhoud van de databuffer kopiëren naar de accumulator

Het bovenstaande schema kan sterk uitgebreid zijn. Het kan bijvoorbeeld zijn dat we na de opcode een adres hebben staan dat wijst naar één of meer locaties waarvan de inhoud het adres zijn waar de data te vinden is (ga naar bosweg 13 en vraag daar waar D. Ata woont). Dit laatste noemen we indirecte adressering. Ook kan het zijn dat bij de berekening van het adres de inhoud van het één of andere register bij het adres opgeteld moet worden. De 6502 heeft speciaal voor dit doel de twee indexregisters X en Y die ook in figuur 3 staan. Geïndexeerde adressering gaat op de volgende manier: Bij het adres wordt de inhoud van het indexregister opgeteld waarna het adres van de data ontstaat dus: D. Ata woont in het derde huis na bosweg 9.

Wat we in de bovenstaande beschrijving meteen kunnen zien, is dat op een 6502 instructies die intern werken twee cycli doormaken en instructies met een absolute adressering 4. Bij de 6502 is een cyclus precies een periode van de klok zodat een instructie met absolute adressering twee keer zo lang duurt als een instructie die intern werkt. Wordt de adresberekening moeilijker, dan duurt het meteen ook langer. Als vuistregel kunt u het volgende schema hanteren:

- In de eerste cyclus wordt de opcode opgehaald (instructie fetch)
- Als de instructie niet intern werkt, dan wordt in de volgende cycli het adres van de data berekend. Om dit te berekenen moet de processor informatie uit het geheugen halen. Hij kan bij elke cyclus informatie ter breedte van de databus ophalen. Heeft een 6502 dus voor de benadering van zijn operand 3 byte nodig dan zijn hiervoor 3 cycli nodig
- Als er voor de benadering van de data berekeningen nodig zijn, dan kost dit een extra cyclus (bijvoorbeeld het optellen van het indexregister bij het adres)
- Het benaderen van de operand kost hierna evenveel cycli als nodig is om de operand over de databus te transporteren (dus bij 68000 een operand van 32 bit over de 16 bit databus: 2 cycli).

- Nadat de operand opgehaald is, moet de instructie nog uitgevoerd worden. Hij moet bijvoorbeeld nog bij de inhoud van de accu opgeteld worden. Vaak kost het uitvoeren van een instructie ook nog een aantal cycli. De 68000 bijvoorbeeld heeft een instructie voor het vermenigvuldigen. Deze instructie heeft, afhankelijk van de waarde van de operanden tot ongeveer 15 cycli (= 60 klokpulsen) voor alleen de uitvoering nodig. Bij de 6502 heeft elke instructie voor de uitvoering slechts één cyclus nodig en meestal wordt dit gedaan tijdens de opcode fetch van de volgende instructie. Zodoende heeft een ADC # (Tel bij de accu het volgende getal op) effectief slechts 2 cycli nodig; in de eerste wordt de opcode opgehaald in de tweede wordt de operand ingelezen en in de derde wordt de optelling uitgevoerd. In deze derde cyclus wordt echter ook al weer de volgende opcode opgehaald en gedecodeerd. Deze techniek wordt "Pipelining" genoemd en werkt perfect zolang de processor weet waar hij de volgende opcode kan vinden, kortom zolang hij een programma van boven naar beneden af kan werken.

In het bovenstaande verhaal is het ophalen van een instructie en de bijbehorende gegevens besproken. Laten we nu eens kijken hoe de instructie uitgevoerd wordt.

Meestal is het zo dat de processor gegevens uit het geheugen haalt en hier intern iets mee doet (bijvoorbeeld haal een getal uit het geheugen en tel dit bij een register op) of gegevens ergens uit de processor in het geheugen schrijft (bijvoorbeeld schrijf de inhoud van een register in het geheugen). Bij het eerste type wordt het geheugen dus benaderd om te lezen en bij het tweede type om te schrijven. Verder zijn er een aantal operaties die alleen intern binnen de processor werken en tenslotte bestaan er ook nog een aantal instructies waarbij de inhoud van een geheugenplaats gelezen wordt, deze gegevens worden gewijzigd en daarna worden de nieuwe gegevens teruggeschreven naar de oude geheugenplaats. Dit zijn de zogenaamde "Read-Modify-Write" operaties zoals bijvoorbeeld: Tel 1 bij geheugenplaats xxx op (INC). Omdat het halen en brengen van gegevens uit resp. naar het geheugen vrij veel klokpulsen vragen, hebben processoren zogenaamde registers. In wezen zijn dit geheugenplaatsen binnen de processor. De 6502 kent één register, bedoeld voor de bewerking van data. Dit register heet de accumulator of accu. De 68000 heeft

16 registers waarbij er acht speciaal voor het bewerken van data en acht speciaal voor het bewerken van adressen bedoeld zijn. Het aantal bits in de registers is ook sterk van de processor afhankelijk. Een 6502 heeft bijvoorbeeld een 8-bits accu en bij een 68000 zijn de registers allemaal 32 bit breed. Het werken met en op registers heeft als voordeel dat het veel sneller gaat; we hoeven namelijk niet iedere keer het (voor de processor) langzame geheugen te benaderen.

In het plaatje van de 6502 zien we naast de accumulator één van de belangrijkste elementen in een processor: de ALU of Arithmetic and Logical Unit. Dit onderdeel is verantwoordelijk voor alle bewerkingen op de interne registers met uitzondering van de programmateller. De ALU doet dus het eigenlijke werk (optellen, aftrekken, schuiven, roteren.....). Een ALU komt in elke processor voor waarbij, afhankelijk van het instructieset, de ALU natuurlijk meer en minder bewerkingen uit kan voeren. Verder kan de ALU uit pure elektronische schakelingen bestaan of zelf ook weer een microprocessor zijn met een eigen programmaatje voor het uitvoeren van de instructies. Dit programmaatje heet dan microprogramma en is geschreven in microcode. Voor dit artikel is dit verder niet van belang.

De indexregisters X en Y uit het plaatje zijn typisch voor de 6502. In bepaalde gevallen wordt de inhoud van één van deze registers opgeteld bij een adres zodat er een nieuw adres ontstaat. De meeste processors kennen wel een vorm van geïndexeerde adressering maar niet elke processor heeft speciale registers voor dit doel.

In de linker kolom hebben we dan nog één blokje niet besproken: Het stack pointer register of de stack pointer. Dit register komt op elke processor voor waarbij de precieze uitvoering van processor tot processor verschilt. Het basisprincipe is als volgt: In de stackpointer staat een adres. Dit adres is de bovenkant van een stapel of stack. Als we nu tijdelijk iets ergens op willen bergen dan zetten we dit op de stack wat als volgt in zijn werk gaat:

Verlaag de inhoud van de stack pointer met 1

Schrijf datgene wat je kwijt wilt weg in de locatie die aangegeven wordt door de stackpointer

Willen we het weer van de stack afhalen, dan gaat dat als volgt:

Haal de inhoud van de locatie waarna de stack pointer wijst op

Verhoog de stack pointer met 1

In het bovenstaande voorbeeld wordt uitgegaan van een stack die van hoog naar laag adres loopt en dat de stack pointer naar de bovenste gevulde locatie wijst. Bij de 6502 loopt de stack ook van hoog naar laag maar wijst de stack pointer naar de eerste vrije locatie. Verder is het natuurlijk zo dat als we op een machine met een databus van 16 bits een 16 bits getal op de stack willen zetten, de stackpointer niet met 1 maar met 2 veranderd wordt. Het basisprincipe blijft echter altijd hetzelfde en wordt wel eens vergeleken met een prikker voor memo-briefjes, een ijzeren pen op een bijvoorbeeld houten voet, waar memobriefjes opgeprikt worden. Het kenmerk van een stack is dat hetgeen er als laatste op terecht gekomen is, er weer als eerste afgehaald wordt. (Last In, First Out).

Rechts onder in het plaatje zien we dan nog het processor status register. Dit register wordt ook wel het vlaggenreger genoemd. De inhoud van dit register geeft aan wat de toestand van de processor na het uitvoeren van een instructie is. Als de laatst uitgevoerde bewerking als resultaat 0 (nul) had, dan wordt dit aangegeven doormiddel van de Zero vlag. Had de laatste bewerking als resultaat een negatief getal (Hoogste bit 1, zie ook [4]) dan wordt dit aangegeven door middel van de Negativ vlag. Verder kennen alle processors minstens een zogenaamde Carry en overflow vlag. Aan de vlaggen kunnen wij en de processor dus zien wat er aan de hand is en of bepaalde stukken programma misschien overgeslagen kunnen worden.

Als laatste vinden we rechts boven in het plaatje een blokje met interrupt logic. Een interrupt is het onderbreken van het normale werk van een processor om hem eerst iets anders te laten doen. Over interrupts valt echter zoveel te vertellen dat ik dat graag in een volgende aflevering van de serie wil doen.

Zo, al met al heb ik nu ruim vier pagina's volgeschreven over de logische opbouw van een processor waarbij ik vooral uitgegaan ben van de 6502. Ik denk dat het nu toch wel zijn plaats is aan te geven wat de verschillen met andere processoren zijn.

Ik denk dat deze verschillen eigenlijk heel klein zijn. Een processor als de 68000 of de 8088 heeft wel meer registers maar in het gebruik maakt dat weinig uit. Bij een 6502 moeten we tussenresultaten terugschrijven naar het geheugen terwijl we bij de 68000 gewoon een ander register nemen maar aan het principe van de werking verandert in wezen niets. Een tweede verschil is natuurlijk het aantal bits in, met name de accumulator. Als er in een register meer bits opgeslagen kunnen worden, worden sommige dingen eenvoudiger te programmeren waardoor de zaken sneller afgehandeld kunnen worden. Ook de adresseermogelijkheden verschillen zeer sterk. Elke processor heeft zijn eigen specifieke set adresseermogelijkheden en daar wil ik in de komende aflevering nog op terugkomen. Verder is het natuurlijk zo dat de ene processor andere instructies heeft dan de andere. Een 68000 kan bijvoorbeeld twee gehele getallen op elkaar delen en de 6502 kan dit niet. Ik denk echter niet dat veel mensen deze mogelijkheid op de 6502 missen want op een 68000 wordt hij vrijwel nooit gebruikt..... In mijn artikel over RISC en CISC wordt uitgelegd dat veel instructies eigenlijk erg slecht is en dat we veel beter een beperkt, goed uitgedacht instructieset kunnen hebben.

Afsluiting.

Ik denk dat de inhoud van dit artikel zeer pittige kost is. Mensen die echt precies willen weten wat er in een computer gebeurt, zouden toch de moeite moeten nemen dit artikel te lezen en te proberen het te begrijpen. Verder vindt ik dat als je echt nauw in contact met de processor wilt komen, dat je toch eens in assembler moet programmeren. Je bent dan gedwongen het instructieset en de adresseermogelijkheden van de processor te bestuderen. In dit kader wil ik voor 6502-ers verwijzen naar de artikelen van Antoine Megens en voor 8088-ers naar de artikelen van Nico de Vries die toegezegd heeft het programmeren in assembler ook te zullen behandelen. Voor 68000-fanaten (waartoe ik ook behoor) zal ik in de nabije toekomst wat meer aan assembler-programmering voor 68000 gaan doen.

In de volgende aflevering van deze serie zullen we ons bezig gaan houden met adresseermethoden en instructiesets waarna daarna waarschijnlijk de tijd rijp geworden is om eens iets aan I/O te gaan doen. U ziet, de excursie door de machine is nog laag niet ten einde, tot de volgende keer

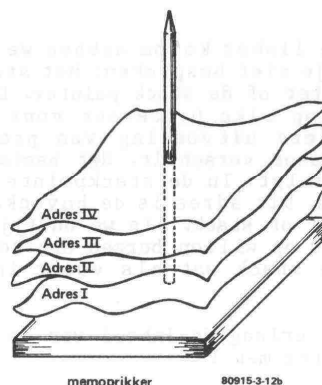
dus.

P.S. Laat me eens weten wat u van deze artikelenreeks vindt. Is het te moeilijk, te gemakkelijk of vind u dat hetgeen u wilt weten niet aan bod komt. Hierbij is mijn eigen doelstelling nog steeds gelijk aan zoals ze in de eerste aflevering geformuleerd is: het inwijden van beginners in de werking van een computer.

Wilt u reageren, dan kunt u mij het beste schriftelijk of via het bulletin board benaderen. Telefonisch is ook mogelijk waarbij er een tamelijk grote kans bestaat dat ik niet thuis ben. U kunt dan het beste aan mijn vrouw vragen of ik u terug wil bellen, wat ik dan ook zeker zal doen.

Referenties.

- 1: G. van Opbroek: CISC en RISC, een inleiding, De 6502 Kenner 57, pag. 28 (augustus 1988)
- 2: Synertek Inc.: SY6500/MCS6500 Microcomputer Hardware Manual (augustus 1976)
- 3: Rockwell: R650X and R651X Microprocessors (CPU) datasheet Rev. 7 (oktober 1984).
- 4: G. van Opbroek: Getallen deel 1, De 6502 Kenner nr. 58 pag. 34 (oktober 1988)



TRACER

Door: Frank Vandekerkhove

Dit programma is enkel en alleen een aanpassing van een programma dat in Elektuur gestaan heeft. De bedoeling was om een tracer te maken zonder hardware aanpassingen/toevoegingen en waarbij het mogelijk zou zijn bepaalde gedeeltes niet te traceren, want dit programma volgt letterlijk alles, ook alle subroutines in bv. de BIOS. Voorts zou een ingebouwde disassembler ook welkom zijn en zo heeft iedereen wel zijn wensen. Ik was al blij dat dit programma werkt en hoop mijn luchtkastelen ooit op de aarde te zetten. Misschien lukt u dat wel beter/snelser.

```
; 6502 tracer
; Elektuur feb. 1984, blz. 66
; door J. Ruppert
;
; aangepast voor DOS65 door Fr. Vandekerkhove
;   nov. 1988
;
; file: Tracer.mac
;
; as Tracer.mac
; lo Tracer.bin
; go 500
;
; Werking:
; Dit programma loopt, instructie per instructie, door een
; ander te testen programma waarvan het startadres wordt
; opgegeven. Programma onderbrekingen zoals BRK, IRQ en NMI
; zijn niet toegelaten.
;
; Dit programma moet in RAM staan:
; - de instructie met zijn operand, uit het test programma,
;   worden in een testveld (label lb619, lb61a en lb61b)
;   geplaatst en gevuld door een geforceerde break. Hierdoor
;   (de BRK) wordt alles gesaved en uitgeprint. Dan wordt
;   de volgende instructie opgehaald, de bewaarde waarden
;   worden teruggezet en deze nieuwe instructie wordt
;   uitgevoerd. Bij het ophalen wordt nagegaan op speciale
;   instructies zoals JSR, RTI, RTS, JMP, JMP-IND. en
;   branches die een speciale behandeling krijgen.
; - alle branch instructies, geplaatst op label "brtst"
;   krijgen een vaste offset van $03. Wordt er aan de
;   voorwaarde voldaan, dan berekent tracer de sprong en
;   voert die uit, ofwel gaat tracer gewoon verder in het
;   programma.
;
; Dit tracer programma begint op het adres $500. Het start-
; adres van het te testen programma wordt in een pointer
; ($00ED en $00EE) gezet. Deze pointer wordt gebruikt om de
; instructies in het testprogramma op te halen en kan als
; een pseude programma counter beschouwd worden.
;
; De volgende RAM-plaatsen zitten midden in het programma:
; lb619: instructie veld
; lb61a: operand
; lb61b: idem
; lb61c: 00, force a break
;
; brtst: test field voor branches
; brffs: vaste offset van $03
;
; De volgende RAM-plaatsen bevatten variabelen:
;
; pnt en pnt+1: de pointer/pseude prog. counter
```



```

;
; lb713 status wordt hier uitgeschoven om uit te printen
; lb714 stack pointer na beh. van een instr.
; lb715 ?
; lb716 lengte v/d instructie minus een (!) die afgeteld wordt bij het
; uitprinten van de instructie na een break
; of save stack-data bij stack printing
;
; lb717 ?
; lb718 instructie veld
; lb719 operand
; lb71a operand
; lb71b accu na beh. van instr.
; lb71c Y-reg
; lb71d X-reg
; lb71e instr. lengte om de programma teller te verhogen
; lb71f ?
; lb720 status na beh. van instructie
; lb721 instr. lengte van pas gevonden instructie
; lb722 ?
;
; P.S.: De namen van deze parameters komen overeen met hun
; origineel adres in het oorspronkelijke programma;
; bv.: "lb718" was in het originele programma een
; RAMplaats ($0718) waar de instructie stond; in
; programma wordt het instructie veld aangeduid
; met "label 718".
;
; *** DOS65 2.01 subroutines ***
;
inecho equ $C026 get and put character
out equ $C023 put 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
aschex equ $C03E a ascii > hex
loupch equ $C041 a lower > upper
;
; *** variables ***
;
org $00ed
;
pnt res 2 pseudo prog. counter/pointer
;
org $0500
;
main lda #$00 reset the pointer
sta pnt
sta pnt+1
;
jsr print
fcc 'Startaddress: $',0
;
1 jsr inecho wait for a input
cmp #$0d cr ?
beq endst end of input startaddress
jsr loupch
jsr aschex
bcs 1.b not a hex
ldx #$04
2 asla hex in low nibble > high nibble
dex
bne 2.b
ldx #$04
3 asla rotate hex in pointer
rol pnt
rol pnt+1
dex
bne 3.b
beq 1.b

```

```

;
endst   jsr      print
        fcc      $0c                      clear ccreen
        fcc      '6502 - tracer\r'
        fcc      'ADR. -INSTR.- :A :Y :X NV BDIZC STACK\r',0
;
lda     #$00                      clear ram from 1b713 untill 1b722
1       ldy      #$0f
        sta      1b713,y
        dey
        bne      1.b
;
lda     #(irqrt&255)-1 overlay software interrupt routine
        sta      $e76e intvec-16
        lda      #irqrt>>8
        sta      $e76f
;
ldx     #$ff                      clear stack
        txs
;
jmp      newinst
;
;
; Na het initialiseren, het ophalen van de eerste instructie
; (en daarna na elke uitgevoerde instructie) volgt een
; geforceerde break, waardoor het programma in dit gedeelte
; komt. (En er nooit meer uitgeraakt, want het tracer
; programma volgt nu het test programma met al zijn
; subroutines ed.) Het tracer programma kan het beste
; onderbroken worden met ^C. (Control-C)
;
; *** irq routine after a break ***
;
; door een brk worden de program counter en de status
; op de stack bewaard.
;
irqrt   sta      1b71b              save accu
        pla      1b720              get status from stack
        pla      1b720              and save it
        pla      1b71c              destroy old return address
        sty      1b71c              save y-reg
        stx      1b71d              save x-reg
        tsx
        stx      1b714              get stack pointer
        cld                          and save it
        set binair mode (could be changed !)
;
; print instructie met operand, dan accu, y-reg en x-reg.
; (het adres, de pointer, wordt bij het ophalen van een
; nieuwe instructie gedrukt.)
;
1       ldy      #$03
1       lda      1b715,y            =1b718...
        jsr      hexout
2       jsr      spa
        iny
        cpy      #$06
        bcs      4.f
        lda      1b716              lengte van instructie minus een
        bne      3.f
        jsr      spa
        jsr      spa
        jmp      2.b
3       dec      1b716
4       cpy      #$09
        bne      1.b
;
lda     1b720                      get saved status
and     #$cf
sta     1b713                      maak brk-flag 0
                                      save it to shift it out
;

```

```

1      ldx      #$08          print 8 bits of the status
      asl      lb713          shift bit in carry
      bcc      2.f
      lda      #'1           carry set = print a 1
      bne      3.f           branch always
2      lda      #'.'         carry clear = print a point
3      jsr      out
      dex
      bne      1.b           loop for 8 bits

;
jsr     spa
;
lda     lb714          get stack pointer
jsr     hexout         and print it
lda     #'-'
jsr     out            print a "-"
tsx
cpx     #$ff          stack pointer in X-reg
bcc     newinst        branch if stack is empty
pla     lb716          get data from stack
sta     lb716          save it
jsr     hexout
cpx     #$fe
bcc     1.f
pla     lb716          get next data from stack
pha     lb716          put it back
jsr     hexout         and print it
lda     lb716          get previous data from the stack back
pha     lb716          and put it back on stack

newinst jsr     crlf

      lda     pnt+1      print pointer (ps.p.counter)
      jsr     hexout     and print it
      lda     pnt
      jsr     hexout     get low byte and print it

      jsr     spa

;
; haal nieuwe instructie op en bepaal de lengte
;
ldy     #$00
      lda     [pnt],y    get new instruction
      sty     lb61a
      sty     lb61b
      sty     0179
      sty     lb71a
      jsr     instl      bepaal de instructie lengte
      sty     lb71e      bewaar instr. 1. om pnt te verhogen
      tya
      sta     lb716
      dec     lb716      lengte minus 1

;
; zet nieuwe instr. met operand op lb619/lb61a/lb61b en lb718/lb719/lb71a
;
1      dey
      lda     [pnt],y    laad instructie/operand
      sta     lb619,y
      sta     lb718,y
      tya
      bne     1.b

;
; *** verhoog pseudo programma teller (pointer) ***
;
1      inc     pnt        verhoog pointer
      bne     2.f
      inc     pnt+1
2      dec     lb71e      verlaag lengte instructie
      bne     1.b

```

```

; *** filter sprongen uit ***
;
lda    lb718          haal instr. uit test veld
and    #$0f           heboud low nibble
bne    tstjmp
lda    lb718          haal opnieuw de instr.
cmp    #$20           jsr ?
beq    2.f            behandel de jsr-instr.
cmp    #$40           rti
beq    3.f            behandel de rti-instr.
cmp    #$60           rts ?
beq    4.f            behandel de rts-instr.
and    #$10           branch instr. ?
bne    5.f            behandel branch-instr.

tstjmp lda    lb718          haal de instr.
cmp    #$4c           jmp ?
beq    6.f            behandel de jmp-instr.
cmp    #$6c           jmp ind. ?
beq    7.f            brhandel de jmp ind.-instr.

;
; *** prepare for execution ***
;
prex   ldx    lb71d          get previous X-reg
ldy    lb71c          get previous Y-reg
lda    lb720          get previous status
pha                    and put it on stack
lda    lb71b          get previous accu
plp                    status has now its previous value

;
; *** execute instruction, closed by a break ***
;
lb619  res    1            instructie veld
lb61a  res    1            operand/00 (brk)
lb61b  res    1            operand/00 (brk)
lb61c  fcc    0            00 (=brk)

;
; *** behandel de jsr-instr. ***
;
2      lda    pnt          zet ps.p.cnt. op stack
pha
lda    pnt+1
pha
jmp    6.f            naar behandel de jmp-instr.

; *** behandel de rti-instr. ***
;
3      pla
sta    lb720

; *** behandel de rts-instr. ***
;
4      pla
sta    pnt+1
pla
sta    pnt
jmp    9.f

; *** behandel de jmp-instr. ***
;
6      lda    lb61a          execute pseudo jmp or jsr
sta    pnt                to the address given by
lda    lb61b              the operand on the test field
sta    pnt+1

```

; *** plaats een brk instr. op het test veld ***

```

9      lda    #$00
      sta    lb619
      jsr    crlf
      ldx    #$05
1      jsr    spa
      dex
      bne    1.b
      jmp    prex      prepare for execution

```

; *** behandel een jmp ind. ***

```

7      lda    lb61a      pointer uit het operand veld
      sta    pnt
      lda    lb61b
      sta    pnt+1
      ldy    #$00
      lda    [pnt],y      laad het ind. address
      tax          bewaar even in x
      iny
      lda    [pnt],y
      sta    pnt+1      ind. address = nieuwe pr. teller
      txa
      sta    pnt
      jmp    9.b

```

; *** behandel een branch instr. ***

```

5      lda    lb720      get status
      pha          to stack
      lda    lb718
      sta    brtst      zet branch test field
      plp          status from stack to status

```

; *** execute branch instr. ***

```

brtst  res    1
brffs  fcc    $03      offset = 3
      jmp    1.f

```

; *** bereken relatieve sprongen ***

; indien een voorwaarde geldig is, wordt door de offset van \$03

; na de te testen branch, steeds naar hier gesprongen.

```

      cli
      cld
      lda    lb61a      get the offset
      bmi    2.f      branch for neg. offset
      clc
      adc    pnt      add real offset to
      sta    pnt      pseude prog. counter
      bcc    1.f
      inc    pnt+1
1      lda    #$00
      sta    lb61a      00 in operand veld
      jmp    tstjmp
2      clc
      adc    pnt      add neg. offset to
      sta    pnt      pseude prog. counter
      bcs    1.b
      dec    pnt+1
      bcc    1.b      branch always

```

```

; *** bepaal instructie lengte ***
;
; instructie blijft onveranderd in accu
; na afloop bevat het Y-reg de instr. lengte
instl    ldy        #$01          lengte van brk, rti en rts is 1
        cmp        #$00          brk ?
        beq        1.f
        cmp        #$40          rti ?
        beq        1.f
        cmp        #$60          rts ?
        beq        1.f
        ldy        #$03          lengte van jmp is 3
        cmp        #$20
        beq        1.f
        and        #$1f          behoud enkel de vijf laagste bits
        cmp        #$19          inst. lengts is 3 ?
        beq        1.f
        and        #$0f          behoud low nibble
        tax        low nibble als index
        ldy        1tbl,x        zoek op in lengte tabel
1        sty        lb721        bewaar de gevonden instr. lengte
        rts
;
1tbl     fcc        $02          instr. lenght = 2
        fcc        $02          instr. length = 2
        fcc        $02          instr. length = 2
        fcc        $01          instr. length = 1
        fcc        $02          instr. length = 2
        fcc        $02          instr. length = 2
        fcc        $02          instr. length = 2
        fcc        $01          instr. length = 1
        fcc        $01          instr. length = 1
        fcc        $01          instr. length = 1
        fcc        $02          instr. length = 2
        fcc        $01          instr. length = 1
        fcc        $01          instr. length = 1
        fcc        $03          instr. length = 3
        fcc        $03          instr. length = 3
        fcc        $03          instr. length = 3
        fcc        $03          instr. length = 3
;
lb713    res        1          to print status
lb714    res        1          stack pointer na beh. van een instr.
lb715    res        1          ?
lb716    res        1          instr. l. minus een (!) to print instr.
lb717    res        1          ?
lb718    res        1          instructie veld
lb719    res        1          operand
lb71a    res        1          operand
lb71b    res        1          accu na beh. van instr.
lb71c    res        1          Y-reg
lb71d    res        1          X-reg
lb71e    res        1          instr. l. to incr. prog. cnt
lb71f    res        1          ?
lb720    res        1          status na beh. van instructie
lb721    res        1          instr. lengte van pas gevonden instructie
lb722    res        1          ?

```

```
;
; Demo van J. Ruppert
```

```
;
org      $0200          7      BEQ      6.f
demo     LDA      #$03    5      BEQ      7.b
          TAY          4      BEQ      8.b
          TAX          6      JSR      subr1
          LDA      #$09    SEC
          STA      $00     NOP
          SED      NOP
          CLC      NOP
          ADC      $00     subr1 JSR      subr2
          DEX      RTS
          BNE      1.b     subr2 RTS
          ROLA     9      JMP      subr3
          RORA     JMP      demo
          SEC
          SBC      $00     ;      org      $02FC
          DEY          3      BCS      4.f
          BNE      2.b     1      BCS      2.f
          SBC      $00     subr3 BCS      1.b
          CLD      2      BCS      3.b
          BEQ      3.f     4      JMP      [vect]
          BEQ      4.f     vect   fcc      $00
          BEQ      5.f     fcc    $02
```

```
;
; Resultaat van de tracer met demo
;
; lo Tracer.bin
; go 500
; Startaddress: $200
;
; 6502 - tracer
; ADR. -INSTR.- :A :Y :X NV BDI ZC STACK
;
; 0200 A9 03      03 00 00 ..... FF-
; 0202 AB        03 03 00 ..... FF-
; 0203 AA        03 03 03 ..... FF-
; 0204 A9 09      09 03 03 ..... FF-
; 0206 B5 00      09 03 03 ..... FF-
; 0208 FB        09 03 03 .....1... FF-
; 0209 18        09 03 03 .....1... FF-
; 020A 65 00      18 03 03 .....1... FF-
; 020C CA        18 03 02 .....1... FF-
; 020D D0 FA      18 03 02 .....1... FF-
; 0209 18        18 03 02 .....1... FF-
; 020A 65 00      27 03 02 .....1... FF-
; 020C CA        27 03 01 .....1... FF-
; 020D D0 FA      27 03 01 .....1... FF-
; 0209 18        27 03 01 .....1... FF-
; 020A 65 00      36 03 01 .....1... FF-
; 020C CA        36 03 00 .....1.1. FF-
; 020D D0 FA      36 03 00 .....1.1. FF-
; 020F 2A        6C 03 00 .....1... FF-
; 0210 6A        36 03 00 .....1... FF-
; 0211 38        36 03 00 .....1.1. FF-
; 0212 E5 00      27 03 00 .....1.1. FF-
; 0214 88        27 02 00 .....1.1. FF-
; 0215 D0 FA      27 02 00 .....1.1. FF-
; 0211 38        27 02 00 .....1.1. FF-
; 0212 E5 00      18 02 00 .....1.1. FF-
;
; 0214 88        18 01 00 .....1..1 FF-
; 0215 D0 FA      18 01 00 .....1..1 FF-
; 0211 38        18 01 00 .....1..1 FF-
; 0212 E5 00      09 01 00 .....1..1 FF-
; 0214 88        09 00 00 .....1.11 FF-
; 0215 D0 FA      09 00 00 .....1.11 FF-
; 0217 E5 00      00 00 00 .....1.11 FF-
; 0219 D8        00 00 00 .....11 FF-
; 021A F0 00      00 00 00 .....11 FF-
; 021C F0 06      00 00 00 .....11 FF-
; 0224 F0 FB      00 00 00 .....11 FF-
; 021E F0 02      00 00 00 .....11 FF-
; 0222 F0 FC      00 00 00 .....11 FF-
; 0220 F0 04      00 00 00 .....11 FF-
; 0226          20 30 02 00 00 00 .....11 FD-0229
; 0230          20 34 02 00 00 00 .....11 FB-0233
; 0234          60          00 00 00 .....11 FD-0229
; 0233          60          00 00 00 .....11 FF-
; 0229 38        00 00 00 .....11 FF-
; 022A EA        00 00 00 .....11 FF-
; 022B          4C 35 02 00 00 00 .....11 FF-
; 0235          4C 00 03 00 00 00 .....11 FF-
; 0300 B0 FC      00 00 00 .....11 FF-
; 02FE B0 02      00 00 00 .....11 FF-
; 0302 B0 FB      00 00 00 .....11 FF-
; 02FC B0 06      00 00 00 .....11 FF-
; 0304          6C 07 03 00 00 00 .....11 FF-
; 0200 A9 03      03 00 00 .....1 FF-
; 0202 AB        03 03 00 .....1 FF-
; 0203 AA
```

```
; ^C
; $(DOS-prompt)
;
end      main
```


De IBM-PC en z'n klonen (Deel 3).

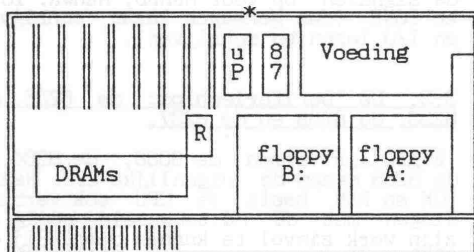
Door: Nico de Vries.

In het vorige deel kon een bescheiden uitleg gevonden worden over het hart van de IBM-PC(/XT): de intel 8088 CPU. We konden hierin lezen dat deze CPU een 8-bit CPU is (ondanks dat de meeste mensen vinden dat de PC(/XT) een 16-bit machine is) en dat de 8088 1 Mbyte geheugen kan adresseren. Daarnaast heeft de CPU de mogelijkheid om maximaal 64k aan I/O te adresseren, buiten de gewone geheugenmap om. Dit laatste is een typisch intel feature: bij de meeste andere populaire processoren van andere merken (w.o. de 6502) bevindt de I/O zich gewoon in de geheugenmap.

Met alleen een CPU heb je geen computer. Er moet van alles om heen: geheugen (liefst veel tegenwoordig) in de vorm van RAM en ROM, I/O onder andere bestaande uit video kaarten, diskcontrollers en RS-232 poorten en nog zo wat losse electronica. Hoe dit bij de PC/XT is georganiseerd komt in dit deel aan de orde.

3.1. Mechanische opbouw.

Als een PC(/XT) openmaakt, dan zie je het volgende:



Helemaal onderin de kast, links, ligt een relatief grote printplaat met een behoorlijke hoeveelheid IC's erop. Dit is het moederbord. Het moederbord neemt ongeveer 60% van het vloeroppervlak van de kast voor zijn rekening. Op het moederbord zit zoals te verwachten de 8088 CPU. In de tekening is plaats waar de CPU zich bevindt bij de meeste klonen aangegeven van de CPU met uP.

Naast de CPU is meestal een leeg 40-pins voetje gemonteerd, dat bedoeld is voor een extra processor: de 8087 floating point coprocessor (hierover later meer).

Midden op de moederbord bevinden zich meestal 2 of 6 voetjes voor ROMs. In echte IBMs zitten hier de ROMs met het BIOS (hierover later VEEEL meer) en de Microsoft BASIC interpreter. Oude moederborden bezitten 6 voetjes waarvan 5 gevuld met 8kx8 ROMs, de nieuwere doen het met 2 voetjes: 1 voor het BIOS (8kx8) en 1 voor de BASIC interpreter (32kx8). De meeste klonen ontberen om copyright redenen de BASIC ROMs, en bezitten slechts een BIOS, die vrijwel altijd in EPROM zit. Wel hebben alle klonen de mogelijkheid om BASIC (EP)ROMs later toe te voegen. Het BIOS ROM bevindt zich bij de meeste mainboards op de plaats gemerkt met R.

Meer naar voren bevindt zich op het moederbord het RAM-geheugen, opgebouwd met dynamische RAMs of DRAMs. Af-

hankelijk van de ouderdom en herkomst kan het RAM zowel in grootte, als in samenstelling behoorlijk variëren. Alle machines bezitten pariteitscontrole op het RAM: reden waarom een PC steeds 9 stuks DRAMs per bank heeft (een AT, die immers echt 16 bits is uiteraard 18).

De eerste IBM PC bezat vier RAM banken, die afhankelijk van de kapitaalcracht van de koper al of niet gevuld waren. De kleinste machines waren 16 kbyte (echt waar!) en hadden 1 bank met 4116 DRAMs. Was het mainboard vol, dat zaten er 4 banken met 4116's in ofwel 64kbyte. Dergelijke PC's duiken soms op op de tweedehands markt. De volgende generatie mainboards was iets beter op dit punt: leverbaar met 64 kbyte (1 bank 4164's), 128 kbyte (2 banken) of 256 kbyte (4 banken, dus vol).

Wil je groter bij IBM, dan komt er een geheugenuitbreidingskaart aan te pas. Zo niet bij de latere generaties klonen: de meeste kunnen de maximale 640 kbyte RAM op het mainboard bergen door 2 banken 41256's en 2 banken 4164's. De laatste generatie XT-klonen heeft 2 banken met 41256's, en de laatste 128k wordt opgebouwd uit 4 stuks 4164 (64kx4) en twee stuks 4164 voor de pariteit.

Aan de achterzijde van het moederbord bevinden zich de uitbreidings-slots. In de oer-PC zaten 5 van deze slots, die zijn uitgevoerd als 62-polige card-edge connectors. De PC/XT en nagenoeg alle klonen hebben 8 slots. In deze slots kunnen allerlei kaarten gestoken worden, waardoor de machine aangepast kan worden aan het gebruiksdoel.

De achterkant van de kast wordt verder gevuld door de voeding, die altijd van het schakelende type is. De karakteristieke rode netschakelaar steekt aan de zijkant door een opening in de kast naar buiten. De voeding heeft in deze meeste gevallen een totaal uitgangsvermogen van 135 Watt. Vroege PC's van IBM hebben soms een 63 of 100 Watt voeding. Sommige moderne klonen bezitten een 150 Watt voeding, en dat terwijl de CMOS opruikt!

De rest van de kast wordt opgevuld door de disk drives. Echte IBM's hebben twee full height floppy drives of 1

floppy drive en een 10 of 20 Mbyte full height winchester. De meeste klonen hebben slim-line drives. Nagenoeg alle kasten hebben plaats voor twee full height, of 4 slim-line drives.

Tot zo ver de mechanische opbouw. We gaan nu eens verder kijken wat er zich nog meer op het moederbord bevindt.

3.2. Supportchips: de 8284 en de 8288.

De CPU kan zijn werk niet alleen doen. Hij heeft niet alleen geheugen nodig, maar ook een aantal besturingschips. Nu is intel op dat punt een rare fabrikant: voordat een 8088 kan werken komen er nog twee zogenaamde supportchips aan te pas: de 8288 bus controller en de 8284 clock generator. Naar onze maatstaven is dit raar: Bij bijna alle processors, dus ook de 6502, zijn deze zaken on-chip aanwezig. Maar goed.

De 8284 is de clock generator. Deze chip bevat een kristal oscillator die een timing generator aandrijft. Verder zit er een aansluiting op voor een externe clock, waardoor je met een TTL-signaal kunt omschakelen van de kristal oscillator naar de externe clock en terug. Dit is het geheim achter de omschakelbare clocksnelheid van de meeste zogenaamde Turbo-klonen. De kristal frequentie wordt in de 8284 door drie gedeeld, waarna de CPU-clock frequentie ontstaat. Originele IBM's lopen op een frequentie van 4.77 MHz, een op het eerste gezicht vreemde waarde. Dit zit zo:

De Amerikaanse kleurentelevisienorm heeft een kleurdraaggolffrequentie van 3.58 MHz. Op deze frequentie in het beeldsignaal wordt de kleurinformatie overgedragen. De PC kan worden uitgerust met een kleurenkaart (CGA kaart, zie later), die ook een signaal kan maken volgens de Amerikaanse kleuren-TV norm (NTSC). Dan heb je dus een frequentie nodig van 3.58 MHz. Om symmetrische golfvormen te verkrijgen, moet je zo'n signaal maken door een hogere frequentie te delen. De IBM-PC deelt door vier, ofwel de kristal-frequentie is 4×3.58 is 14.318 MHz. Zo'n kristal is dan ook op het moederbord te vinden: kijk maar eens in de buurt van de 8284, het enige 18-pins IC op het bord. Zoals gezegd deelt de 8284 de kristalfrequentie door drie zodat de CPU clockrate $14.318 / 3$ is 4.77 MHz wordt. Er wordt nog verder gedeeld in de 8284. Ten behoeve van de I/O chips, die meestal niet zo snel zijn, wordt de CPU clock ook nog eens door 4 gedeeld, zodat er ook een signaal met een frequentie van $4.77 / 4$ is 1.19 MHz ontstaat.

Turbo klonen hebben behalve het standaard kristal van 14.318 MHz ook nog een tweede kristal. Voor een 6.67 MHz turbo PC is dat dus een 20 MHz, voor 8 MHz turbo 24 MHz, en voor een 10 MHz een 30 kristal.

De laatste functie van de 8284 is de afwikkeling van de CPU cycles. De 8284 heeft hiervoor ingangen waarmee gemeld kan worden dat de huidige cycle kan worden afgemaakt. Heeft men langzaam geheugen, dan stelt men dit signaal eenvoudig 1 of meer clockcycli uit, waardoor de CPU gewoon blijft wachten. Deze cycli worden dan ook wachtcycli of wait-states genoemd. De PC(XT) werkt zonder wait states in ROM, 1 wait state in DRAM en 2 wait states als er met I/O gewerkt wordt. Een ROM-cycle duurt dan de minimale 3 clockcycli, een RAM cycle 4 clockcycli en een I/O cyclus 5 clockcycli.

De tweede supportchip is de 8288 bus controller. Deze chip zorgt voor de generatie van de timing op de bus en zorgt ook voor signalen om de gemultiplexte adres- en databus van de CPU te de-multiplexen. Verder wekt de 8288 de signalen op voor MEMRD, MEMWR, IORD en IOWR (dus geheugen lezen, schrijven en I/O lezen en schrijven).

3.3. De periferiechips: de 8255, de 8253, de 8259 en de 8237.

Eigenlijk vormen de 8088, de 8284 en de 8288 samen de eigenlijke CPU. Naast ROM en RAM heeft de CPU ook verbindingen met de buitenwereld nodig om zijn werk zinnig te kunnen doen. Bij de 6502 bedienen we ons van PIA's, VIA's en ACIA's en nog wat van dat spul, bij de 8088 is het nauwelijks anders. Op het moederbord vinden we vier periferiechips uit de intel-stal:

8255 parallelle I/O chip
8253 timer chip
8259 interrupt controller
8237 DMA controller

Hierbij twee relatief onbekende begrippen: DMA controller en interrupt controller.

Een DMA controller is een soort miniatur CPU, die direct het geheugen kan bereiken. Je kun hem gebruiken om bijvoorbeeld vanaf een diskcontroller rechtstreeks het geheugen te laden, zonder tussenkomst van de CPU. Omdat er maar 1 device tegelijkertijd de baas over de bus mag spelen, wordt de 8088 CPU in zo'n geval tijdelijk buitenspel gezet, via de stuurlijnen van de 8284. De DMA controller heeft een set registers aan boord waarin het geheugen adres en de hoeveelheid data wordt ingesteld. Ook zijn er registers voor het bestemmingsadres. De 8237 DMA controller heeft 4 kanalen, die ieder hun eigen set adresregisters hebben. De chip kan drie soorten DMA doen: van geheugen naar geheugen, van I/O adres naar geheugen en van geheugen naar I/O adres. Verder kan de 8237 niet alleen lezen en schrijven maar ook een verify (block compare) uitvoeren. In de

PC(/XT) wordt de DMA controller niet alleen 'normaal' gebruikt voor bijvoorbeeld disk I/O maar ook voor een bijzondere taak: de refresh van de dynamische RAMs. Dit gebeurt als volgt: 1 van de kanalen (kanaal 0) wordt ingesteld op een start source adres van 0, een count van \$FFFF en een bestemmings adres van 0. Dat kanaal wordt vervolgens ingesteld op block move. Iedere keer als er nu een DMA-aanvraag voor dat kanaal binnenkomt, gebeurt er een block move vanaf adres 0 over een block van 64 kbyte. Weinig zinvol zou je zeggen maar dat is niet zo: omdat het volledige 64 kbyte blok wordt afgewerkt, ondergaan alle DRAM adressen een leesen en een schrijfoperatie. Slaat men het DRAM datasheet erop na, dan blijkt dat iedere locatie de gelezen wordt, automatisch gerefreshed is. Niet alle DRAMs in het systeem worden geactiveerd (alleen de eerste 64 kbyte), maar wel is het zo dat alle mogelijke RAS-adressen een keer gegenereerd worden en dat alle DRAMs ook werkelijk een RAS krijgen. Behalve snelheid en eenvoud levert dit nog een voordeel op: in plaats van 3 adressen (rij, kolom en refresh adres) hoeft men nu nog maar twee adressen te multiplexen: rij en kolom. Ook heeft deze methode als voordeel dat eventuele geheugen kaarten op de I/O bus automatisch mee-gerefreshed worden: ook deze kaarten genereren een complete set RAS-adressen.

De tweede voor ons vreemde chip is de 8259 interrupt controller. Net als de 6502 heeft de 8088 maar 1 echte interrupt aansluiting: de INTR-pin. Nu kun je, net als bij de 6502 gebruikelijk, alle interruptbronnen hierop aansluiten en bij een interrupt gaan uitzoeken wie er aan de bel heeft getrokken, maar het kan ook mooier, en daarvoor is de 8259 bedacht.

De 8259 heeft 1 uitgang, die verbonden is met de CPU INTR-pin. Verder zijn er acht ingangen, die IRO tot en met IR7 heten. Op deze ingangen worden alle interruptbronnen aangesloten. Verder is de 8259 voorzien van een CS-pin en een databusaansluiting. Via deze weg kun je de chip programmeren.

Je kunt onder andere bepalen hoe de prioriteit van de acht ingangen is geregeld. In de PC(/XT) wordt dit zodanig ingesteld, dat IRO de hoogste prioriteit heeft, en IR7 de laagste. Wordt meer dan 1 IR-lijn actief, dan zorgt de 8259 ervoor, dat de CPU keurig twee INTR's krijgt aangeboden in de volgorde van de ingestelde prioriteit. De tweede INTR volgt de eerste pas, nadat de 8259 is meegedeeld, dat de eerste interrupt is afgehandeld. Verder is het mogelijk een offset op te geven voor het vectornummer. Ho even... alweer iets nieuws!

Wat is dat: het vectornummer? Als de 8088 een INTR herkent start de processor een normale leescycle op de bus. De

processor leest dan op DO-D7 van de databus het nummer van de interrupt die zojuist gegenereerd werd. Dit nummer is het vectornummer. Op deze manier kunnen we 256 verschillende vectoren onderscheiden. Officieel zijn de vectoren 0 t/m 1F gereserveerd voor de CPU zelf, en rest is voor de gebruiker. In 1 van de volgende delen zullen we zien dat IBM hier een vergissing mee heeft begaan....

De 8259 wordt bij de initialisatie ingesteld op een vectoroffset van 8. Dit betekent, dat wanneer de IRO-pin actief is, de 8088 eerst een INTR krijgt aangeboden waarna de 8259 het getal 8 op de databus zet bij de volgende leescycle. De processor zet dat vervolgens vector nummer 8 in CS:IP gaat de interrupt serven. In de interrupt routine wordt meestal als laatste aan de 8259 meegedeeld, dat de interrupt geserviced is door een End-Of-Interrupt (EOI) commando in het commando register te schrijven. Bij een IR-1 wordt het vectornummer dus 9, enzovoort.

De 8259 houdt intern in registers bij welke IR-pinnen er ge-enabled zijn, en welke interrupts er nog geserviced moeten worden en welke bezig zijn geserviced te worden. Verder is het mogelijk meerdere 8259's in cascade te schakelen, maar dit wordt in de PC(/XT) niet gebruikt (in de PC/AT wel).

Een overzichtje van de 8 hardware interrupts in de PC(/XT):

IRQ0/INT08: Systeem tijd
IRQ1/INT09: Toetsenbord
IRQ2/INT0A: Systeembus
IRQ3/INT0B: RS-232 poort II (COM2:)
IRQ4/INT0C: RS-232 poort I (COM1:)
IRQ5/INT0D: Printer II (LPT2:)
IRQ6/INT0E: Floppy disk controller
IRQ7/INT0F: Printer I (LPT1:)

Nu het bekendere werk: parallel I/O en timers (samen een VIA dus...).

De 8253 timer chip is de eenvoudigste van de periferiechips: hij zit in een 24 pins behuizing. De 8253 bevat drie 16-bit timers, die ongeveer dezelfde mogelijkheden hebben als een timer uit een 6522 VIA. Het enige verschil is dat de clock die bij de VIA naar keuze de CPU-clock (intern) of een externe clock kan zijn, bij de 8253 altijd extern is. In de PC(/XT) is hier het 1.19 MHz signaal uit de 8284 clock generator op aangesloten. Iedere timer heeft zijn eigen clockingang, die ook nog voorzien is van een enable-pin. De enables liggen allemaal aan de plus, waardoor de clock altijd wordt doorgelaten. Verder heeft iedere timer een uitgangspin, die hoog wordt als de timer afloopt.

In de PC(/XT) word alle timers gebruikt. Timer 0 is verantwoordelijk voor de systeemtijd. Hiertoewordt de timer in periodieke 16-bit mode gezet,

met een maximale periodetijd: \$FFFF. Het duurt dus 65536 maal $1/1.19 \text{ us} = 55.07 \text{ ms}$ voordat de timer afloopt. Of iets anders uitgedrukt: $1/55.07$ of 18.2 maal per seconde loopt de timer af. De uitgang van de timer is verbonden met de IRQ0 aansluiting van de 8259 interrupt controller, waardoor het systeem 18.2 maal per seconde een interrupt krijgt van de hoogste prioriteit. In de interrupt service routine wordt een tellertje bijgehouden, dat na 24 uur automatisch op nul wordt gezet.

Timer 1 wordt op soortgelijke wijze gebruikt. De periodetijd van deze timer wordt ingesteld op 3.93 ms. De uitgang van deze timer is aangesloten op DMA kanaal nul, en stuurt de refresh van de dynamische RAMs: eens in de 3.93 ms gebeurt dit dus.

Timer 2 is aangesloten op de luidspreker van het systeem en wordt meestal gebruikt om tonen te maken voor piepjes. De aansluiting is niet rechtstreeks: er zit nog een poort tussen zodat je de luidspreker ook uit kunt schakelen. Hierdoor zijn er 3 mogelijkheden om geluid aan de PC te ontlokken:

- Poort open, via de timer
- Poort aan en uit, timer uit
- Beide tegelijk

Volgt het reeds Meestal wordt de eerste mogelijkheid gebruikt.

Er bestaat ook een versie van 8253 die hogere clock snelheden aankan: de 8254. Deze twee chips zijn over het algemeen gewoon uitwisselbaar omdat er zelden met clockfrequenties groter dan 2 MHz gewerkt wordt.

De laatste I/O chip is de 8255 parallele I/O chip. Hij lijkt sterk op een PIA, zij het dat er niet twee, maar drie poorten op zitten.

Poort A is verbonden met een schuifregister, dat de inkomende data omzet van serieel naar parallel. Zijn er acht bits binnengekomen, dat genereert het schuifregister een IR1 of wel een INT09. U heeft het misschien al geraden: het schuifregister is verbonden met het toetsenbord. Het toetsenbord stuurt zijn informatie namelijk serieel naar de PC(/XT), en wekt zijn eigen clock op waarmee het schuifregister geclockt wordt. Is een compleet byte verzonden, dan volgt een interrupt, waarna poort A van de 8255 kan worden gelezen en het schuifregister gecleared wordt.

Poort B bevat een aantal stuurlijnen die op bitniveau gebruikt worden. Bits 0 en 1 worden gebruikt voor de luidspreker: met bit 0 kan timer 2 aan- en uitgezet worden, terwijl bit 1 de reeds genoemde poort bedient. Bit 2 werd in de oorspronkelijke PC gebruikt om de motor van de cassetterecorder voor dataopslag via een relais aan- en uit te zetten; in de PC/XT wordt het bit niet gebruikt. De meeste Turbo-klonen

gebruiken dit bit om de CPU-clock om te schakelen van gewoon (4.77 MHz) naar Turbo, via de 8284 clock generator.

Bit 3 wordt gebruikt om een set van 4 schakelaartjes uit de set van 8 kiezen die gebruikt worden om de configuratie in te stellen. Met de bits 4 en 5 kunnen de flipflops die onthouden of er een parityfout in het geheugen op het moederbord of op de bus is getreden, worden gereset. De bits 6 en 7 tenslotte zijn verbonden met de keyboard interface, om het schuifregister te kunnen resetten.

Op poort C zitten de schakelaartjes die dienen om de configuratie in te stellen, aangesloten op bits 0 tot 3. Bit 3 van poort B bepaalt welke helft van de acht schakelaartjes wordt uitgelezen. Op bit 4 zit de luidspreker, zodat je ook de toestand van die lijn kunt uitlezen. Bit 5 is verbonden met de uitgang van de DMA controller, die aangeeft of de controller bezig is of niet. Bit 6 en 7 tenslotte zijn aangesloten op de uitgangen van de parityfout-flipflops. Hier kun je dus zien of een parityfout is geweest.

3.4 De volgende keer.....

in het vorige deel beloofde blokschema. Samen met de informatie uit dit deel krijgt u dan een goed beeld wat al die dingen op het moederbord van een PC betekenen. Verder in het volgende deel de indeling van zowel het geheugen als de I/O. Tot de volgende keer.

TE KOOP:

6502 Elektuur CPU kaart
Elektuur floppy disc controller kaart
Elektuur VDU kaart
2 Elektuur statische RAM kaarten (6116/2114)
Eigenbouw geheugenkaart (8 x 6116 wire wrap)
Eigenbouw I/O kaart (wire wrap)
19 inch kaart
Monochrom monitor (Amber)
Wire wrap experimenteerboard

Dit alles met documentatie en schema's.

Prijs: n.o.t.k.
Telefoon: 01899-21481
W. van Asperen

Een Public Domain APL-implementatie.

Door Gert van Opbroek

Normaliter zullen (en kunnen) we in de UP Kenner weinig aankondigingen van nieuwe pakketten voor PC's vermelden maar dit pakket is dermate interessant dat ik dit zeker onder de aandacht van de leden wil brengen.

Het betreft het pakket I-APL, een implementatie van de programmeertaal APL of A Programming Language, speciaal bedoeld voor onderwijs (Instruction) maar dermate krachtig dat het ook voor andere doeleinden geschikt zou kunnen zijn. Dit pakket is op internationaal initiatief ontwikkeld. Met de opzet en het ontwerp van het pakket is als eis gesteld dat I-APL moet kunnen draaien op goedkope machines in standaarduitvoering. Verder is het een volledige APL-implementatie en voldoet het aan de International Standards Organisation (ISO) standaard voor APL. Momenteel is I-APL beschikbaar voor de IBM-PC en daarmee compatible PC's. Verder draaien er testversies op de deense Piccoline, de BCC8, de Archimedes, de Sinclair QL, de Amiga, de Commodore 64 en op Z80 onder CP/M.

I-APL is vrij kopieerbaar en de documentatie wordt tegen kostprijs verspreid. In Nederland wordt I-APL ondersteund door de projectgroep I-APL binnen de werkgroep APL van het Nederlands Genootschap voor Informatica. Deze projectgroep verzorgt ook de verspreiding van I-APL.

Wilt u I-APL voor uw PC ontvangen, dan kunt u deze bestellen door het overmaken van fl. 45,- op postgiro 3446248 van APL werkgroep Nederland te Hengelo. U ontvangt dan een diskette en drie, voorlopig nog engelstalige, boeken: het Tutorial, het I-APL/PC Manual en de Encyclopedia.

Wat is APL?

APL is een programmeertaal net zoals bijvoorbeeld Basic, Pascal, Forth..... Het verschil met een gewone programmeertaal zit hem hierin dat de taal wiskundig geïntegreerd is, je kunt er dus vrij gemakkelijk wiskundige berekeningen mee uitvoeren. APL gebruikt hiervoor ook de standaard wiskundige symbolen, kortom voor APL heb je toetsenborden met allerlei afwijkende symbolen. Verder gebruikt APL voor de vermigvuldiging de 'X' in plaats van de

'*' zoals de meeste andere programmeertalen, dus net zoals we het allemaal op de lagere school geleerd hebben.

Met APL kun je bewerkingen uitvoeren op gehele getallen, drijvende komma getallen en strings. Verder kun je ook één- en meer-dimensionale array's (vectoren en matrices) maken waarmee je op dezelfde manier kunt rekenen. Zo kun je onder APL met de 'X' niet alleen twee getallen, maar ook twee matrices met elkaar vermigvuldigen. Voor de wiskundigen onder ons is verder interessant dat APL in- en uitproduct en inverses van matrices kan berekenen, kortom legio wiskundige mogelijkheden.

APL is een geïnterpreteerde taal wat wil zeggen dat een opdracht meteen uitgevoerd kan worden en er niet eerst een vertaalslag met behulp van een compiler nodig is. Dit houdt wel in, dat in vergelijking met een taal als C de uitvoering van een APL-opdracht langzaam is. De uitvoering is bij I-APL zelfs langzamer als bij Basic. Bij commerciële APL-implementaties is dit niet zo, maar dit is het gevolg van de opzet van I-APL. Nu is de snelheid van uitvoering niet het enige dat telt. Ook de snelheid van ontwikkeling is van belang en eigenlijk moeten we de som van ontwikkelen en uitvoeringstijd bekijken. Op dit punt laat APL bij wiskundige problemen de overige programmeertalen ver achter zich. Het schrijven van een programma voor matrix-inverse in C kostte me toch altijd nog een dag of twee; met APL is het intikken van een opdracht voldoende waarna de PC enkele seconden rekent.

Zelf heb ik het pakket onderhand ontvangen. Ik heb echter nog niet de tijd gehad hier uitgebreid mee te experimenteren. Als ik dit wel gedaan heb, en het me gelukt is de APL-tekens op de printer afgedrukt te krijgen, zal ik eens wat voorbeelden in APL in dit blad opnemen. Mocht u meer informatie willen hebben, dan kunt u uiteraard contact met mij opnemen waarna ik u verder zal proberen te helpen.

Noot: Is het misschien interessant samen met de I-APL projectgroep te onderzoeken of I-APL op DOS-65 en EC-65 kan lopen? Vrijwilligers kunnen zich bij mij opgeven waarna ik dat dan wil coördineren. Ik weet overigens niet of hiervoor mogelijkheden zijn, maar wie weet.