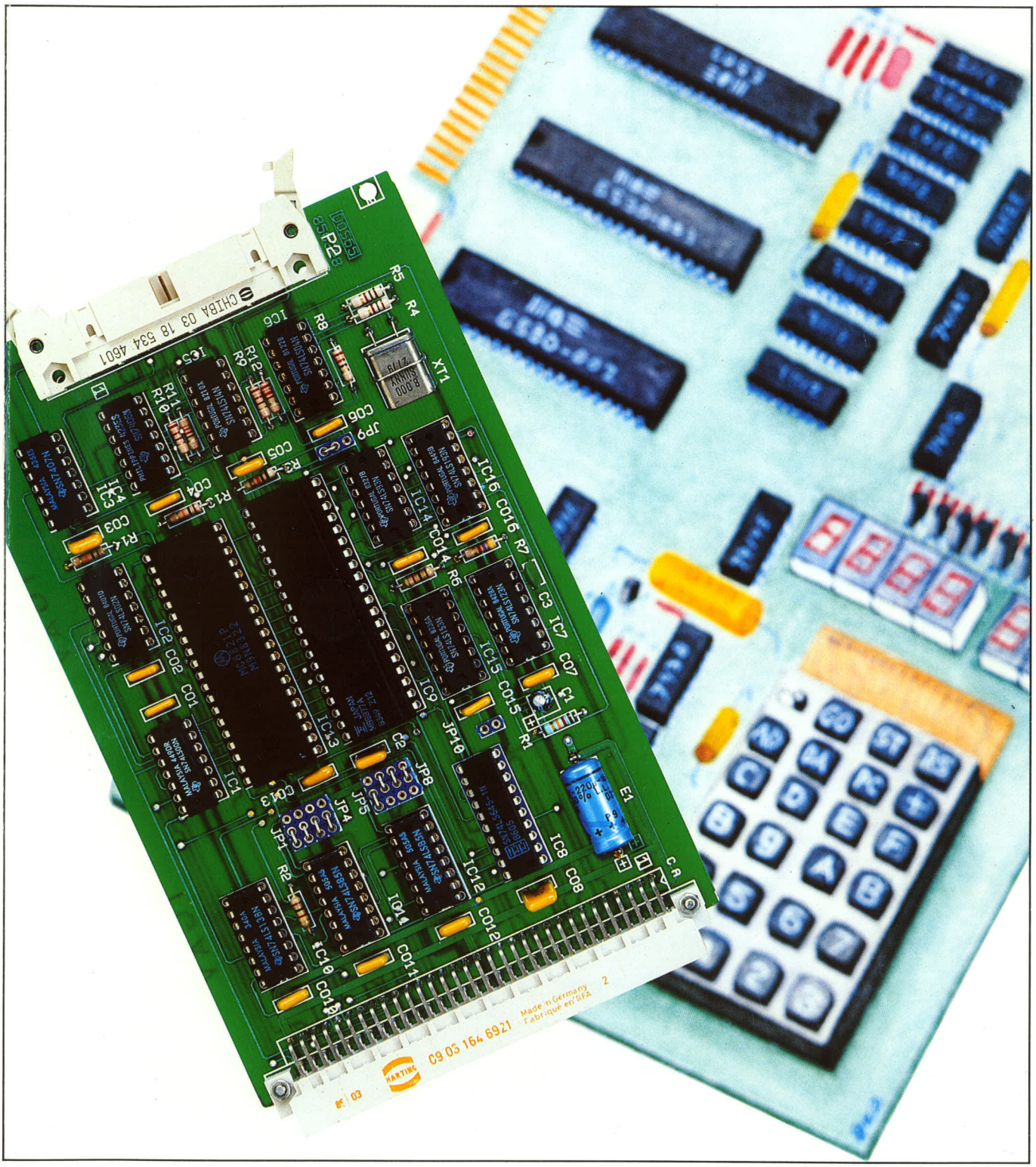
DE µP KENNER

63

Dertiende jaargang nr. 4.



INFORMATIE.

De uP Kenner (De microprocessor Kenner) is een uitgave van de KIM 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. systeemcoördinators 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 HENGELO
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 63

DERTIENDE JAARGANG NR. 4 OKTOBER 1989

De uP Kenner:
De uP Kenner is het huisorgaan van de KIM Gebruikersclub Nederland en wordt bij verschijnen gratis 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 opgestuurd worden aan het redactieadres. De redactie beslist, op basis van bruikbaarheid, publicatiewaarde en actualiteit of en zo ja, wanneer een ingezonden artikel geplaatst wordt.
Geplaatste artikelen blijven het geestelijk eigendom van de auteur en mogen niet zonders diens toestemming door derden gepubliceerd worden.
Helaas kan de redactie noch het bestuur enige aansprakelijkheid aanvaarden voor de toepassing(en) van de geplaatste kopij.

Redactie.
De redactie wordt gevormd door:
Gert van Opbroek

Correspondenten:
Bram de Bruine
Ernst Elderenbosch
Antoine Megens
Frank Vandekerkhove
Nico de Vries

Redactieadres:
Gert van Opbroek
Bateweg 60
2481 AN Woubrugge

==========

INHOUDSOPGAVE

Vereniging

Informatie 2
Uitnoding clubbijeenkomst en agenda ledenvergadering 3
Verslag van de buitengewone ledenvergadering d.d. 7 oktober 1989 5

Algemeen

Redactioneel 4
Programmeren in assembler (deel 4) 7
Help, wie heeft een beschrijving van KB9 Basic 12
Te koop 16
Getallen (deel 5) 21

DOS-65 Corner

Linearisering 13
Windows, een verzameling subroutines 42

MS-DOS

De IBM-PC en zijn klonen (deel 5) 21

Talen/Software

Zeef van Eratosthenes 50

INFORMATIE.

De uP Kenner (De microprocessor Kenner) is een uitgave van de KIM 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 5 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. systeemcoördinators 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 geformd 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 HENGELO
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

KIM GEBRUIKERS CLUB NEDERLAND

Enschede, 10 Nov 1989

CONCEPT BEGROTING 1990

BATEN :

Contributie	120 * Fl 50,-	Fl	6000,-
Losse uP Kenners		-	100,-
Bijeenkomsten		-	400,-
Reclame		-	500,-
Dos 65		-	500,-
	Totaal	Fl	7500,-

LASTEN :

uP Kenner drukken/verzenden	5 * Fl 650,-	Fl	3250,-
Afschijving inventaris		-	2000,-
Sysop		-	1000,-
Bestuurs kosten		-	750,-
Reclame - ledenwervingsactie		-	500,-
	Totaal	Fl	7500,-

De begroting geeft een, misschien, te rooskleurig plaatje.
- De uP Kenner zal in ieder geval geval een stuk goedkoper gemaakt worden. Er is besloten om in ieder het blad niet meer te laten drukken.
Dit neemt al een aanzienlijke kostenbesparing met zich mee.
- De inkomsten die voortkwamen uit de reclame die zoals gebruikelijk achter op het blad stond komt tot nu toe helaas te vervallen, de adverteerder zag hiervan het nut niet meer in.
Het is dus zoeken naar een of meerdere nieuwe adverteerders.
- De losse verkoop van uP Kenners ligt zogoed als plat.
Het komt nog sporadisch voor dat men een los nummer koopt.
Dit gebeurd dan meestal op aanvraag via het BBS, hoewel dit niet van betekenis is...
- Dos 65 is ook aan de lage kant gehouden. Hopende nog de resterende Eprom programmers te kunnen verkopen zit er nog een kleine inkomst uit die hoek bij.
- Het leden aantal zal waarschijnlijk een 'kleine' daling ondergaan..... waardoor er dus minder geld binnenkomt.

J.B.

Uitnodiging voor de clubbijeenkomst

Datum: Zaterdag 11 november 1989
Locatie: Wijkcentrum 't Veurbook
Jan Tooropstraat 27
7606 Almelo
Tel.: 05490 - 10253

Toegang: fl. 10,--

Byz.: Jaarvergadering.

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 rechtdoor. 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 Jaarvergadering KIM Gebruikersclub Nederland: (Toegang gratis)
Agenda:
1) Opening ledenvergadering.
2) Notulen buitengewone ledenvergadering d.d. 7-10-89.
3) Concept begroting 1990.
4) Verkiezing kas-contrôle-commissie 1990.
5) Verkiezing bestuursleden:
Aftredend en niet herkiesbaar zijn:
- Gert Klein
- Rinus Vleesch Dubois
- Adri Hankel
Kandidaten voor het bestuur kunnen zich uiterlijk bij aanvang van de vergadering melden bij de voorzitter of diens plaatsvervanger.
6) Rondvraag.
7) Sluiting.

12:30 Lunch, consumpties tegen betaling.

13:30 Forum en markt.

Aansluitend het informele gedeelte met de mogelijkheid om andermans systemen te bewonderen en Public Domain software uit te wisselen.

17:00 Sluiting.

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

==========

NB Ivm de hcc-dagen de 2e zaterdag i.p.v de derde !!

Van de redactie.

In de eerste plaats heb ik goed nieuws:

De KIM Gebruikersclub Nederland blijft toch bestaan.

Wat is er de afgelopen maanden allemaal gebeurd? Wel, een heleboel. Het begon allemaal op een bestuursvergadering in juli. Het bleek toen dat we onvoldoende geldmiddelen over hadden om in het lopende jaar nog drie uP Kenners uit te geven op de manier die we gewend waren. Verder begon ook het gebrek aan kopij ons parten te spelen zodat we als bestuur toen besloten hebben de procedure die zou leiden tot liquidatie van de club op te starten. Het was toen nog wel de bedoeling het augustusnummer uit te geven met daarin de uitnodigingen voor de twee ledenvergaderingen die voor liquidatie noodzakelijk zijn. Toen wreekte zich echter het gebrek aan kopij. Om een kenner op tijd bij de leden te krijgen, moet hij ongeveer drie weken voor verschijnen samengesteld worden. Op dat moment had ik slechts een tiental pagina's. Zelf was ik wel aan het schrijven maar dat kreeg ik ook niet op tijd klaar, ik ben momenteel namelijk door de week niet thuis en moet dus in het weekend alle redactiewerk doen, samen met mijn andere huiselijke verplichtingen. Kortom het bleek niet mogelijk op tijd een Kenner klaar te krijgen. Omdat de ledenvergaderingen door moesten gaan, hebben we toen besloten tot het versturen van een mailing, die u allemaal eind augustus gehad hebt.

Goed, toen kwam de vergadering in Haarlem. De daar aanwezige leden vonden het unaniem zeer jammer dat de club op zou houden, de een vanwege DOS-65, de ander vanwege de bijeenkomsten. We hebben toen met een paar mensen gekeken of er toch mogelijkheden zijn door te gaan. Dat bleek wel mogelijk als we de volgende maatregelen zouden nemen:

1) Een goedkoper clubblad uitgeven.
2) Meer zekerheid over kopij.
3) Enkele nieuwe bestuursleden.

Het goedkopere clubblad is noodzakelijk omdat het tot nu toe zo was dat elk blad ons ongeveer fl 12,-- kostte en dus teerden we per lid per jaar ongeveer twee tientjes in op de reserves. Daar hebben we een oplossing voor gevonden. We hebben iemand gevonden die het blad (spot-) goedkoop voor ons wil drukken en enkele actieve leden zorgen zelf voor het binden en verzenden.

Het tweede punt, de kopij kan ook opgelost worden. Enkele clubleden hebben me toegezegd geregeld (elke twee maand) een artikel te schrijven. Bovendien heb ik naar aanleiding van ons voorstel tot liquidatie ook al wat extra kopij ontvangen.

Het derde punt, bestuursleden, is bij mijn weten ook opgelost. Van het huidige bestuur blijven vier leden zitten en voor de overige drie functies hebben zich kandidaten aangemeld.

Op de vergadering in Almelo (waar ik helaas zelf niet aanwezig kon zijn) is daarna terecht unaniem het voorstel van het bestuur tot liquidatie afgestemd.

Blijft natuurlijk de vraag of onze club ook na het volgende jaar toekomst heeft. Ik kan daar geen zinnig antwoord op geven. Ik denk wel dat het belangrijk is dat we ook wat meer buiten het wereldje van de 6502, dat van oudsher onze specialiteit is, gaan doen. Verder is ook een flinke stroom kopij van het grootste belang want voor de meeste leden is toch het blad het belangrijkste communicatiemiddel. Ik hoop dan ook dat u allen wilt meewerken aan het op interessante manier vullen van het blad. Verder zullen er ook een aantal actieve (bestuurs-) leden noodzakelijk zijn. Om de kosten te drukken moeten nu namelijk veel meer werkzaamheden binnen de club uitgevoerd worden.

Tenslotte toch iets over dit nummer van de uP Kenner. Hierin staat van de serie getallen de inlees- en afdrukroutine. Deze routines zijn vanuit C omgezet naar 6502 assembler en veel groter geworden dan ik verwacht had. Verder staan er enkele leuke programa's voor DOS-65 in en gaan Nico en Antoine verder met resp. MS-DOS en het programmeren in assembler. Waar ik echt gebrek aan heb, is kopij die ik kan plaatsen in de rubriek Talen /Software. Ik zou het erg leuk vinden als ik wat korte of middellange (maximaal 10 pagina's = 600 regels) programma's in Forth, Basic, Pascal, C of iets dergelijks zou krijgen. Niet iedereen programmeert namelijk in 6502 assembler.

Uw redacteur:
Gert van Opbroek.

Verslag van de buitengewone ledenvergadering gehouden op 7 oktober 1989.

1. Opening

De voorzitter opent de vergadering.

2. Voorstel tot liquidatie van de vereniging.

Zoals bekend heeft het bestuur een buitengewone ledenvergadering uitgeschreven die op 16 september 1989 is gehouden. Deze vergadering was uitgeschreven omdat het bestuur van mening was dat het gezien de financiële situatie van de club niet langer verantwoord was om door te gaan. Derhalve zou op deze vergadering een voorstel tot liquidatie van de vereniging behandeld worden. De statuten bepalen dat een dergelijk voorstel alleen behandeld kan worden indien 2/3 van het aantal leden van de vereniging aanwezig is. Op de vergadering van 16 september was dit niet het geval en kon het voorstel derhalve niet behandeld worden. De statuten schrijven in een dergelijk geval voor dat binnen vier weken een nieuwe ledenvergadering wordt uitgeschreven. In deze vergadering wordt dan ongeacht het aantal aanwezigen met meerderheid van stemmen besloten. Dit is de reden dat deze vergadering thans wordt gehouden.

Het bestuur heeft besloten het voorstel tot liquidatie in te dienen om een aantal redenen:

1. Het aantal leden van de vereniging bedraagt momenteel 180. De begroting is echter gebaseerd op een aantal van 300. Als gevolg hiervan is de financiële situatie danig verslechterd. De liquide middelen van de vereniging zijn niet toereikend om de 6502 Kenner dit jaar zes maal te laten verschijnen. Er zal noodgedwongen een editie moeten vervallen. Te verwachten valt dat volgend jaar het aantal leden opnieuw minder zal zijn dan dit jaar, waardoor de situatie nog nijpender wordt.

2. Er komt steeds minder copy binnen bij de redactie. Het gevolg hiervan is dat de redacteur steeds meer tijd moet steken in het zelf volschrijven van het blad.

3. De belangstelling voor de bijeenkomsten loopt langzaam maar zeker terug. Er is een harde kern van leden die elke bijeenkomst bezoekt, maar er komen nauwelijks nog nieuwe leden bij.

De afgelopen weken zijn er gelukkig een aantal ontwikkelingen geweest die het bestuur van mening hebben doen veranderen. Een groot aantal leden heeft het bestuur laten weten dat zij het buitengewoon zouden betreuren indien de vereniging zou stoppen. Een aantal van hen heeft aangeboden zich in te zetten voor het behoud van de vereniging. Daarnaast lijkt de financiële situatie op korte termijn toch minder slecht dan in eerste instantie gedacht was. De vereniging beschikt nog over een financiële reserve van ruim f8000,-. Tenslotte blijkt het mogelijk te zijn de 6502 Kenner veel goedkoper te laten drukken, mits een aantal werkzaamheden zoals samenvoegen, verzenden enz. zelf worden gedaan. Het bovenstaande heeft het bestuur van mening doen veranderen ten aanzien van de overlevingskansen van de vereniging.

Voorwaarde is echter wel dat het bestuur versterkt moet worden met een aantal nieuwe en aktieve bestuursleden. Drie bestuursleden hebben te kennen gegeven dat zij niet meer de tijd en de energie kunnen opbrengen voor een bestuursfunctie. Dit zijn: Rinus Vleesch Dubois (voorzitter), Gert Klein (secretaris) en Adri Hankel (lid). Deze drie bestuurders zullen derhalve vervangen moeten worden. Vier bestuursleden zijn bereid gevonden om zich voor tenminste een jaar te blijven inspannen in een bestuursfunctie. Het zijn: Jaques Banser (penningmeester), Jan Derksen (lid), Gert van Opbroek (redacteur) en Nico de Vries (lid). Er is een

kandidaat gevonden voor de functie van voorzitter in de persoon van Mick Agterberg. Twee aanwezige leden, E. Elderenbosch en Ton Smits stellen zich tevens kandidaat voor een bestuursfunctie.

De ledenvergadering stelt voor om nu geen beslissing te nemen over de benoeming van nieuwe bestuursleden, maar dit uit te tellen tot de reguliere jaarvergadering van november. Aldus wordt besloten.

Resumerend is de situatie dus als volgt:

1. Het bestuur trekt het voorstel tot liquidatie van de vereniging in.

2. Er wordt een reguliere jaarvergadering uitgeschreven op 11 november 1989.

3. In deze jaarvergadering moeten drie nieuwe bestuursleden, waaronder een voorzitter en een secretaris gekozen worden.

4. Het nieuwe bestuur zal een beleidsplan moeten maken voor de toekomst.

5. Er zal zo spoedig mogelijk een 6502 Kenner uitgebracht worden waarin dit verslag gepubliceerd zal worden. In deze editie zal tevens een tussenstand van de financiën gepubliceerd worden.

6. Er zal een concept begroting gemaakt worden voor 1990 op basis van een verwacht ledental van 100.

7. In de 6052 kenner zal tevens een oproep geplaatst worden voor kandidaat bestuursleden.

8. Om zicht te krijgen op het aantal leden voor het volgend jaar zal bij deze editie van de 6502 Kenner reeds de acceptgiro voor de contributie van 1990 bijgevoegd worden.

3. Rondvraag.

De vergadering verzoekt het bestuur om in het verslag van deze vergadering uitdrukkelijk te vermelden dat bovengenoemde besluiten in goede harmonie tussen leden en bestuur, en tussen bestuursleden onderling zijn genomen. Er is geen sprake van slechte onderlinge verhoudingen of wanbestuur. De voorzitter dankt de leden voor deze uitspraak en sluit de vergadering.

Wageningen, 8 oktober 1989

Gert Klein
secretaris

Programeren in Assembler (deel 4)

PANIEK

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

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

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

Integer delen

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

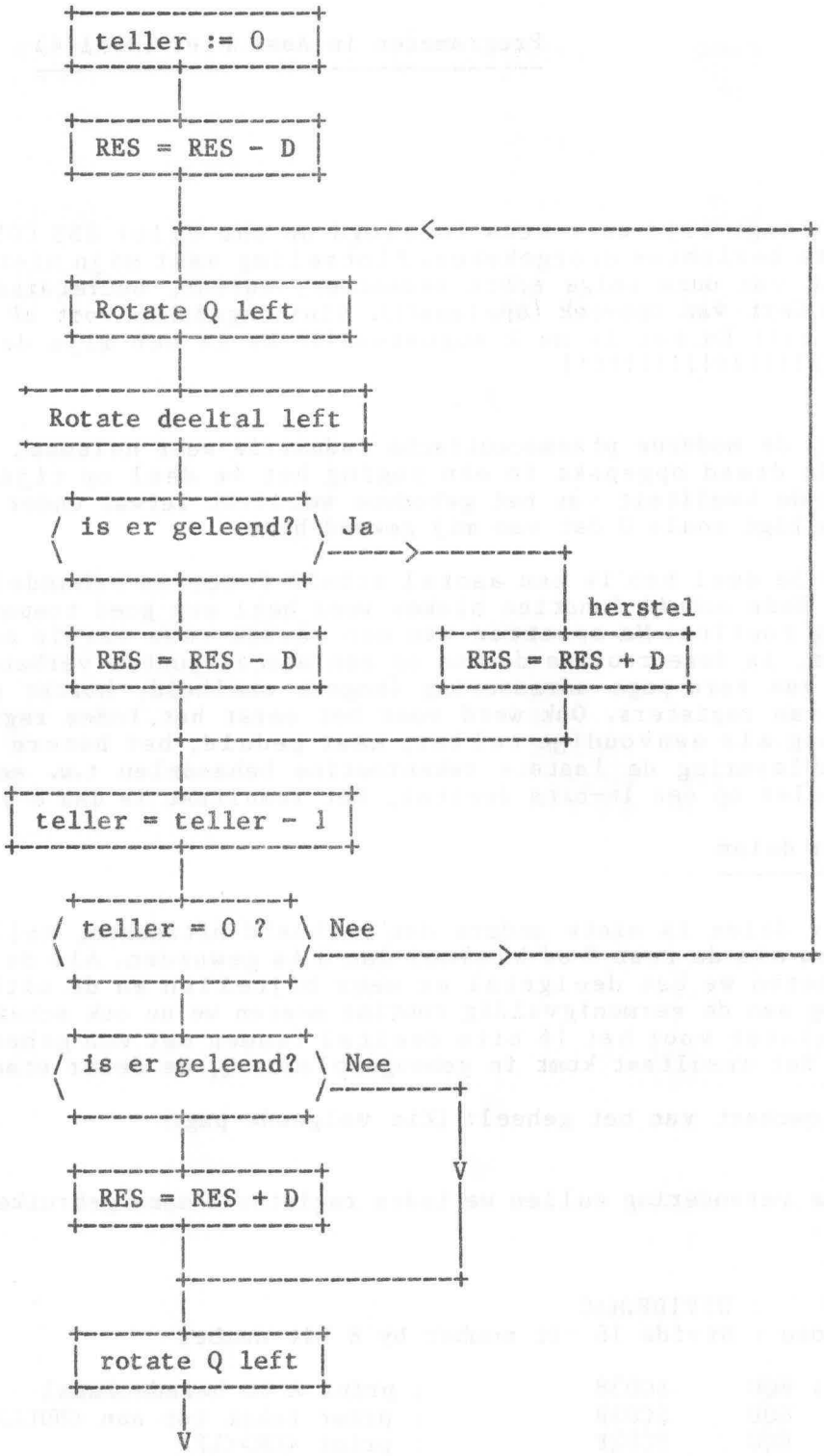
Een flowchart van het geheel: (Zie volgende pag.)

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

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

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

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



klaar (Q = uitkomst, A = rest)

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

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

        END     DIVIDE          ; thats all folks!
```

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

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

Assembleren en laden die hap:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        ORG     $1000

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

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

```
        BNE     DLOOP1          ; deze gaat 256x keer rond
        DEX
        BNE     DLOOP2          ; en deze oo !
        RTS

        END     DEMO
```

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

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

MON> E 1000
```

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

Antoine.

HELP HELP HELP HELP HELP HELP HELP HELP

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

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

Graag contact opnemen met:

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

LINEARISERING

Door Frank Vandekerkhove

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

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

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

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

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

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

```
;
; File:         Pt10cnv.mac
; Program:      Pt10cnv
; Function:     Convert input data
;
; Frank Vandekerkhove
; Sint-Michielsstraat 4, B-2789 Verrebroek
;
; *** I/065 2.02 subroutines ***
;
conver    equ  $FF02          convert 2 bytes hex into 3 bytes dec.
pridec    equ  $FF41          print dec numbers (deci 1, 2 & 3)
;
; *** io_var ***
;
deci1     equ  $E73A          variables for calculate decimal
deci2     equ  $E73B
deci3     equ  $E73C
;
; *** Dos 65 2.01 subroutines ***
;
out       equ  $C023          put a character
crlf      equ  $C02F          print <CR> and <LF>
spa       equ  $C032          print a space
hexout    equ  $C038          a hexout
print     equ  $C03B          print string after call

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

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

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

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

==========

Te Koop

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

Alles eventueel met Junior boeken en 6502 bladen.

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

==========

De IBM-PC en z'n klonen (Deel 5).

Door. Nico de Vries.

Aan alleen een processor en een kwak geheugen en wat primitieve I/O, of kortweg de inhoud van het moederbord in een PC heb je niet zoveel. Je kunt geen resultaten zien, en je kunt geen gegevens invoeren. Daarom verlaten we in dit deel het moederbord, en gaan we de diverse mogelijkheden voor video output naar een monitor bekijken. IBM heeft voor de PC en de PC/XT in principe twee display opties, beide bestaande uit een monitor en een adapterkaart, bedoeld om in 1 van de slots op het moederbord gestoken te worden.

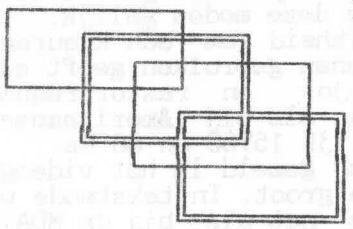
De eerste mogelijkheid is speciaal toegesneden op alle toepassingen waar met teksten gewerkt wordt: het kantoor en de serieuze hobby toepassingen. Hiervoor heeft IBM de monochrome adapter met behorende monitor. Deze adapter wordt meestal aangeduid met MDA (van Monochrome Display Adapter).

De tweede optie lijkt mooier, maar is het in de praktijk meestal niet: de Color Graphics Adapter of kortweg CGA, compleet met kleurenmonitor. Tijd voor meer detail.

5.1. Algemene opbouw van de adapters.

Beide displaykaarten lijken in opbouw op elkaar. Ze zijn beide gebaseerd op de ons wel bekende 6845 CRT-controller chip. De 6845 verzorgt het opwekken van de lijn- en rastersynchronisatiesignalen en het leveren van de videogeheugen-adressen voor het laten zien van de informatie op het scherm.

Ook hebben beide kaarten een karakter-generator in ROM, die de complete IBM karakterset bevat, dus 256 verschillende tekens. De tekens die bij de codes $20 tot en met $7F horen, komen overeen met de gebruikelijke ASCII tekenset. Tekens boven $80 zijn onder andere: speciale nationale tekens zoals die met accenten of umlauten, de nederlandse gulden (ƒ) en de ij met puntjes (ÿ), verder zijn er tekens om kadertjes en dergelijke te maken, de zogenaamde line-draw karakters. Dit levert onder ondere de volgende mogelijkheden op:



Integraalteken: ∫

De rest van de tekenset wordt gebruikt voor deel van het Griekse alfabet en wat wiskundige tekens, waaronder het complete integraalteken, bestaande uit twee karakters (zie boven).

Om de scherminhoud te bewaren, heb je een videogeheugen nodig. Bij de MDA is dit statisch RAM, en wel 4kbyte, opgebouwd met 2 2kx8 SRAMs. De CGA-kaart heeft 16 kbyte dynamisch RAM, opgebouwd met 2 stuks 16kx4 DRAMs. De refresh gebeurt door het periodiek uitlezen van het RAM t.b.v. het display. Tot zover de overeenkomstige zaken. Nu de beide kaarten in detail.

5.2. De Monochrome Display Adapter.

Zoals reeds opgemerkt is de monochrome display adapter bedoeld voor teksttoepassingen. De kaart kan dan ook alleen maar tekst laten zien, net als de CRTC-kaart in de DOS65 en Octopus systemen. Het schermformaat is ook standaard: 25 regels van ieder 80 tekens. De kaart is echter iets uitgebreider dan de 6502 CRTC kaart: de karakters kunnen op meerdere manieren getoond worden.

Om dit mogelijk te maken wordt van ieder karakter niet alleen de ASCII-waarde in het videogeheugen bewaard, maar ook nog een tweede byte, dat aangeeft hoe het karakter op het scherm moet verschijnen. Ieder karakter beslaat dus 2 bytes in het videogeheugen. Dit is overigens bij alle IBM displayadapters het geval. Het tweede byte wordt het attribuut-byte genoemd. In de MDA zijn de volgende attributen mogelijk:

Bit	7	6	5	4	3	2	1	0	Attribuut
	B	0	0	0	I	0	0	0	Niet zichbaar
	B	0	0	0	I	0	0	1	Onderstreept
	B	0	0	0	I	1	1	1	Normaal
	B	1	1	1	I	0	0	0	Invers

Hierbij geldt verder:
I=0: normale karakterhelderheid
I=1: verhoogde karakterhelderheid (Intensified)

Het B-bit kan twee functies hebben, afhankelijk van het blink-enable bit in 1 van de statusregisters:
Blink-enable=0
B=0: achtergrond is zwart
B=1: achtergrond is helder

Blink-enable=1
B=0: continue weergave
B=1: karakter knippert

In het algemeen stellen de bits 0-2 van het attribuutbyte de voorgrond helderheid van het karakter voor, terwijl de bits 4-6 de helderheid van de achtergrond bepalen. De helderheid van de voorgrond kan nog worden vergroot door het I-bit 1 te maken. Deze opbouw lijkt op de opbouw van het attribuutbyte in de CGA kaart (zie verderop), zij het dat niet alle mogelijkheden worden gebruikt.
De goede weergave van de karakters met deze kaart vindt zijn oorsprong in het feit dat een karaktercel 14 bij 9 pixels groot is, waarvan er 14 bij 8 gevuld worden door de karaktergenerator. Door een truc in de displayhardware sluiten grafische tekens toch aan: de laatste pixelkolom wordt in die gevallen tweemaal getoond. Ook de grote videobandbreedte in de monitor gecombineerd met een fosfor met hoge scherpte draagt bij tot de werkelijk uitstekende beeldkwaliteit. Het beeld maakt verder een rustige indruk: dit wordt veroorzaakt doordat IBM gebruik gemaakt heeft van een nalichtende fosfor. Deze fosfor is er de oorzaak van, dat het scherm na het scrollen enigszins nagloeit, en de neiging tot vegen vertoont.
De monitor wordt met de MDA kaart verbonden via een 9-polige D-connector, die de volgende signalen voert:

Pin	
1	GND
2	GND
3	Niet aangesloten
4	Niet aangesloten
5	Niet aangesloten
6	Intensify
7	Video
8	Horizontal drive
9	Vertical drive

Alle signalen zijn TTL-niveau: dus nul of een. Dit houdt in, dat de karakters op het scherm wit (groen) of zwart kunnen zijn. Hetzelfde geldt voor de achtergrond, die wit (groen) of zwart is. In de monitor is een circuit aanwezig dat de voorgrond helderheid verdubbelt als het Intensify signaal hoog is. De kontrastregelaar op de monitor regelt dan ook de helderheidsverhouding tussen de normale en de intensified karakters.
Omdat het scherm altijd 25 regels van 80 tekens bevat en de MDA maar 4 kbyte RAM heeft, kan de kaart maar 1 scherm bewaren, inclusief de attribuutbytes.
Als enige displaykaart in de IBM reeks is de MDA voorzien van een aansluiting voor een parallel printer.
De 6845 CTRC bevindt zich op I/O adressen 03B4 en 03B5. Op 03B8 bevindt zich het displaymode register, en op 03BA zit het statusregister van de displaykaart. De 6845 wordt zodanig geprogrammeerd, dat de lijnfrequentie 18.7 kHz en de rasterfrequentie 52.5 Hz bedragen.

De printerpoort bevindt zich op de I/O adressen 03BC, 03BD en 03BE, en komt geheel overeen met de printerpoorten op andere kaarten. Als de MDA in het systeem zit, is de printerpoort ervan altijd LPT1:.
Het videogeheugen bevindt zich op de absolute adressen B0000-B0FFF, meestal aangeduid met B000:0000 tot en met B000:0FFF.

5.3. De Color Graphics Adapter.

Als tegenhanger van de MDA kaart bood IBM ook een kleurenkaart aan: de CGA. Op grond van de beeldkwaliteit is de CGA kaart minder geschikt voor tekst-toepassingen: een karaktercel beslaat slechts 8 bij 8 pixels. Daarentegen heeft de kaart een groot scala aan mogelijkheden meegekregen.
Zoals de naam al aangeeft, kan de CGA kaart in grafische mode bedreven worden, waarbij ieder pixel op het scherm aan of uit kan zijn. Daarnaast heeft de kaart ook een aantal tekstmodes. Een overzichtje:

Mode	Beschrijving
0	Tekst, 25 regels van 40 karakters
1	Als mode 0, doch kleur
2	Tekst, 25 regels van 80 karakters
3	Als mode 2, doch kleur
4	Graphics, 320x200 pixels, kleur
5	Graphics, 320x200 pixels, ZW
6	Graphics, 640x200 pixels, ZW

Het verschil tussen de zwart/wit en kleur modes is alleen te zien, wanneer een Amerikaanse NTSC kleurentelevisie gebruikt wordt met een modulator. In de zwart/wit modes ontbreekt namelijk het burstsignaal. Op een gewone kleurenmonitor zijn deze modes gelijk.
De mogelijkheid om een kleurentelevisie te kunnen gebruiken geeft al aan, dat de lijn- en rasterfrequenties dezelfde zijn als bij (Amerikaanse) TV: respectievelijk 15750 en 60 Hz.
Zoals reeds gemeld is het videogeheugen 16 kbyte groot. In tekstmode worden de karakters net als bij de MDA kaart als twee bytes opgeslagen: de ASCII-waarde, gevolgd door het attribuutbyte. De bits in het attribuutbyte hebben de volgende betekenis:

Bit	Betekenis
0	Voorgrond, Blauw
1	Voorgrond, Groen
2	Voorgrond, Rood
3	Intensify voorgrond
4	Achtergrond, Blauw
5	Achtergrond, Groen
6	Achtergrond, Rood
7	Blink bit

Net als bij de monochrome kaart bepaalt een bit in het statusregister het gedrag van het blink-bit: naar keuze wordt de achtergrond geintensiveerd of knipperen de karakters met dit bit

gezet Merk op dat de CGA niet kan onderstrepen als attribuut

Samen met het intensify-bit zijn er in totaal 16 voorgrondkleuren mogelijk, terwijl er keus is uit 8 achtergrond kleuren De schermrand kan nog apart een kleur krijgen via het kleurregister: eveneens 16 kleuren (RGB + intensify)

Een 40x25 scherm beslaat 1000 karakters, of 2000 bytes VRAM. Hierdoor is het mogelijk in totaal 8 van dergelijke schermen tegelijk te bewaren Deze mogelijkheid wordt dan ook door het systeem BIOS ondersteund: door het veranderen van het display startadres in de 6845 kan de gewenste pagina op het scherm getoond worden Ook in 80x25 mode werkt dit, zij het dat er dan uiteraard maar vier pagina's VRAM ter beschikking staan

In grafisch bedrijf wordt het allemaal heel anders. In alle grafische modes wordt het gehele videoRAM gebruikt voor 1 enkel scherm. In de twee 320x200 pixel modes zijn er dan 2 bits per pixel beschikbaar, waardoor een pixel 4 kleuren kan hebben. Deze kleuren zijn niet vrij te kiezen in het kleurenregister is een bit dat bepaalt welke van de twee beschikbare paletten gebruikt wordt.

Bits	Palet 1	Palet 2
0 0	Achtergrond	Achtergrond
0 1	Cyaan (G+B)	Groen
1 0	Magenta (R+B)	Rood
1 1	Wit (R+G+B)	Bruin (R+G)

De achtergrondkleur in grafisch bedrijf is dezelfde als de schermrandkleur in de tekstmodes.

In 640x200 pixel mode is er voor ieder pixel slechts 1 bit beschikbaar, dat per definitie als wit of achtergrond wordt weergegeven.

Van de grafische modes is nog iets bijzonders te melden: als je als gebruiker normale karakters wilt afdrukken op een grafisch scherm, dan hoef je niet zelf de diverse pixels uit te knobbelen: het systeem BIOS doet dat automatisch mits er van de call 'write character' gebruik gemaakt wordt Het gaat zelfs nog een stap verder Het systeem BIOS heeft ook een call die de gebruiker het karakter op de cursorpositie geeft (read character). Ook deze call werkt normaal in grafisch bedrijf. Beide calls werken natuurlijk wel veel langzamer dan in tekstmode, dat wel

De 6845 van de CGA kaart bevindt zich op de I/O adressen 03D4 en 03D5. Op 03D8 zit het mode register, dat onder andere de mode (tekst of grafisch) van de kaart bepaalt. Adres 03D9 bevat het kleurenregister waarmee de rand/achtergrondkleur en de functie van het blink-bit bepaald wordt.

Het videogeheugen bevindt zich op de absolute adressen B8000 t/m BBFFF, of normaal geschreven: B800:0000 t/m B800:3FFF. In de tekstmodes wordt het geheugen 'normaal' gebruikt: de karakters staan in dezelfde volgorde in het geheugen als dat zij op het scherm verschijnen. In de grafische modes treedt er echter iets geks op: de even lijnen staan in de eerste helft van het geheugen (B8000-B9FFF), en de oneven lijnen in de tweede helft (BA000-BBFFF). Het is even wennen.....

In tegenstelling tot de MDA kaart heeft de CGA geen eigen clockgenerator, maar wordt de door 4 gedeelde systeemclock gebruikt. Omdat de CGA een NTSC signaal kan opwekken met daarin een kleurdraagggolfrequentie van 3.5795 MHz, is de systeemclock daardoor bepaald op de nogal vreemde waarde van 14.318 MHz. (Omdat de 8284 clockgenerator door drie deelt, is de CPU_clock daardoor 4.77 MHz).

Net als bij de MDA kaart zijn alle signalen naar de monitor TTL niveau, dus 1 of nul. De connector naar de monitor is weer een 9-polige D-connector, met als pinbezetting:

Pin	Signaal
1	GND
2	GND
3	Rood
4	Groen
5	Blauw
6	Intensify
7	Niet aangesloten
8	Horizontal drive
9	Vertical drive

Naast de D-connector is er ook een tulp-aansluiting, waarop het zogenoemde composiet video signaal aanwezig is. De diverse kleuren worden hierin voorgesteld door verschillende grijswaarden. Deze aansluiting kan gebruikt worden om een analoge monochroom monitor of een monitor met NTSC decoder aan te sturen.

Verder heeft de kaart nog aansluitingen voor een modulator en een lichtpen. Deze zijn uitgevoerd als rijtjes printpinnen.

Omdat de CGA graphics kan weergeven, en omdat er kleur geproduceerd wordt, is de kaart uitgegroeid tot het standaard display voor spelletjes. In de professionele omgeving gaf men meestal de voorkeur aan de MDA kaart.

5.4. Meer dan 1 display.

Dat kan. Als men de adresbezetting van de kaarten bekijkt, dan blijkt er geen conflict te bestaan, zodat het geen bezwaar is om zowel een MDA als een CGA kaart in 1 PC(/XT) te hebben. Ook het systeem BIOS ondersteunt beide kaarten. De schakelaartjes op het moederbord bepalen op welke kaart het video verschijnt na inschakelen. Met het DOS MODE commando kan in zo'n geval van display gewisseld worden. Er zijn een paar pakketten in de handel die een dergelijke twee-scherm configuratie

ondersteunen: het grafische (kleuren)-plaatje op de CGA kaart en de foutmeldingen en andere tekstmededelingen op de MDA kaart. Dit heeft een snellere werking tot gevolg: mem hoeft dan niet het grafisch scherm te saven, de kaart in tekst mode zetten, een mededeling printen, wachten op een toets en vervolgens weer het grafische scherm op te bouwen.

5.5. De Hercules kaart.

Het ontbreken van een grafische mode op de MDA kaart was voor de Amerikaanse firma Hercules aanleiding om een kaart te ontwerpen die het IBM monochrome scherm kon aansturen in een grafische mode. De resolutie werd 720x350 pixels. Omdat dit in totaal 252000 pixels zijn, ofwel 31500 bytes, was een aanzienlijk groter video geheugen dan bij de MDA kaart nodig. De Hercules kaart heeft dan ook 64 kbyte videogeheugen, opgebouwd met 64kx1 DRAM chips. Ook hier wordt de refresh verzorgd door het periodiek uitlezen van het geheugen. Een bit in het mode register bepaalt welke van de twee mogelijke schermen getoond wordt.

De kracht van de Hercules kaart ligt in het feit dat de kaart in tekstmode 1:1 compatibel in met IBM's MDA kaart. Ook het videogeheugen levert in grafisch bedrijf (HALF mode) geen conflict op met de CGA kaart: het loopt van B0000 t/m B7FFF (B000:0000 - B000:7FFF).

Wil men ook de tweede helft gebruiken dan is er wel een conflict met de CGA kaart: het geheugen loopt dan van B0000 t/m BFFFF. De laatste mode wordt FULL mode genoemd, en wordt slechts zelden gebruikt. Na inschakelen staat de kaart altijd in de MDA-compatibele tekstmode.

De kaart is verder voorzien van dezelfde pinterpoort als de oorspronkelijke MDA kaart.

De Hercules kaart is zo populair geworden, dat het de standaard voor monochroom displays op PC's is, ondanks het feit dat IBM geen aandacht aan de kaart besteedt, en zelfs in latere producten geen Hercules emulatie aanbiedt.

5.6. De EGA kaart.

Eigenlijk valt deze kaart een beetje buiten het bestek van dit verhaal, omdat deze kaart tegelijk met de AT is geintroduceerd. Ook wordt de EGA kaart weinig gebruikt in PC(/XT)'s vanwege de beperkte prestaties van deze machines. Daarom een kort overzicht.

EGA staat voor Enhanced Graphics Adapter, hetgeen aangeeft dat de kaart mag worden gezien als een sterk verbeterde en uitgebreide CGA kaart. Om te beginnen overlapt de EGA zowel MDA als CGA in mogelijkheden: alles wat deze kaarten kunnen, kan de EGA kaart ook, op dezelfde monitor (dus kleur in CGA modes en monochrome in de MDA mode). Zelfs de adressering van de CRTC en het videogeheugen verandert afhankelijk van de mode naar dezelfde adressen als bij CGA respectievelijk MDA. Maar er is meer.

Naast de reeds bestaande modes zijn er ook speciale EGA modes aanwezig. In tekst kan de EGA kaart ook 80x25 in kleur weergeven met een kwaliteit die vergelijkbaar is met de MDA mode. Hiervoor is wel een speciale EGA monitor nodig die de nieuwe lijnfrequntie van 21.5 kHz ondersteunt. En dergelijke monitor wordt van oudsher zodanig uitgevoerd, dat ook de CGA lijnfrequentie van 15.75 KHz geen problemen oplevert, zodat ook de CGA modes bekeken kunnen worden.

In graphics heeft de EGA een hogere resolutie dan CGA: 640x350 pixels. Ook het geheugen is groter in EGA mode: standaard 64 kbyte, maar tegenwoordig bijna altijd de maximale 256 kbyte, bankswitched. Met de laatste hoeveelheid VRAM kan de EGA kaart ook bij de hoogste resolutie nog 16 kleuren weergeven.

De monitor wordt nog steeds in TTL aangestuurd, maar heeft nu twee stel RGB lijnen: primary en secondary RGB. Met zes lijnen kun je 64 combinaties maken: in tekstmode zijn er inderdaad 64 kleuren mogelijk. In graphics mode is het palet in tegenstelling tot de CGA kaart nu wel programmeerbaar zodat 16 van in totaal 64 kleuren vrij gekozen kunnen worden.

Verder heeft de EGA kaart een grafische mode voor de monochrome monitor met een resolutie van 640x350 pixels. De standaard IBM EGA ondersteunt de Hercules mode (uiteraard) niet, de meeste klonen van de EGA kaart doen dit echter wel.

Zoals reeds opgemerkt is de adressering in CGA en MDA modes gelijk aan die van de oorspronkelijke kaarten. In de EGA modes zit de CRTC op de adressen 03C4 en 03C5 (dus precies tussen de MDA en CGA adressen in), en bevindt het videogeheugen zich op absoluut adres A0000 t/m AFFFF. Is er meer dan 64 kbyte videogeheugen, dan is het bankswitched.

Het gedrag na power-up wordt niet bepaald door schakelaars op het moederbord, maar door een viertal schakelaartjes op de EGA kaart zelf.

De EGA kaart heeft zijn eigen video ROM BIOS dat nagenoeg alle kunstjes die de kaart beheerst ondersteunt. Omdat het geheugen in EGA mode zo groot is, is het ook mogelijk het tekstveld groter te maken: zo kan een EGA kaart 132x43 tekst op een EGA of monochrome monitor laten zien.

De CGA en MDA zijn eigenlijk alleen maar 1 op 1 gekopieerd door het Verre Oosten. De EGA kaart daarentegen is uitgebreid onderwerp geweest van uitbreidingen, die zich mestal in de richting van hogere resoluties bewogen: zo zijn er een groot aantal kaarten op

de markt die resoluties van 800x600 of zelfs 1024x600 bieden. Het probleem is dat de toepassingssoftware dit meestal niet kan gebruiken, of pas na de installatie van een speciale driver, terwijl de meeste EGA monitoren moeite beginnen te krijgen met resoluties boven 640 pixels horizontaal.

5.6 De volgende keer.....

Gaan we zoals beloofd de andere zijde van de gebruikersinterface bekijken: het toetsenbord. Tot dan

053-303902
THE ULTIMATE
het Bulletin Board
van de KGN

Getallen (Deel 5).

Door Gert van Opbroek
Bateweg 60
2481 AN Woubrugge

Inleiding

In de voorgaande afleveringen is een pakket subroutines afgedrukt voor de vier basisfuncties voor het rekenen met floating point getallen. In deze aflevering worden een aantal conversie-routines behandeld.

Conversie van een integer naar real.

De eerste routine die afgedrukt is, is een conversie van een geheel getal (integer) met een lengte van 4 byte naar een IEEE single precision floating point getal. Hierbij wordt de integer op de top van de stack vervangen door een real.

Deze routine is de eenvoudigste van de conversie-routines. In eerste instantie wordt het teken van het gehele getal bepaald aan de hand van het voorste (MS) bit. Is het teken negatief, dan wordt dit onthouden waarna het getal positief gemaakt wordt. Door het (negatieve) getal van nul af te trekken.

De tweede stap gaat als volgt: Het grootste (positieve) gehele getal is 2 tot de macht 31 - 1. Dit is het getal:

01111111 11111111 11111111 11111111

Door nu de integer in een 4 byte mantissa te stoppen en in de exponent 32 (+ de bias) te schrijven, krijgen we een niet genormeerd floating point getal met dezelfde waarde als de absolute waarde van de integer. Het getal wordt nu genormeerd door de mantissa zo vaak te schuiven dat er aan de voorkant een 1 uit schuift. Bij elke schuifoperatie wordt de exponent met 1 verlaagd.

De laatste stap bestaat uit het toevoegen van het teken en het in het juiste formaat zetten van de mantissa en exponent.

Conversie van een real naar een integer.

Bij de conversie-routine van een real naar een integer wordt de real op de top van de stack vervangen door een integer.

In deze conversie-routine wordt eerst getest of er een integer ontstaat die in 4 byte opgeslagen kan worden. Dit betekent dat de exponent tussen 0 en 31 moet liggen omdat een genormeerd floating point getal altijd tussen 0 en 1 ligt. Mocht dit het geval zijn, dan wordt er verder gerekend.

Hierna wordt de omgekeerde bewerking van de vorige routine uitgevoerd, de mantissa wordt zolang geschoven en de exponent verhoogt totdat de exponent gelijk is aan 31. Hierna wordt, bij een negatief getal, het complement genomen.

10 to de macht een getal.

Deze routine is eigenlijk alleen bedoeld voor intern gebruik. Hij kent namelijk geen enkele beveiliging tegen overflow. Het is zelfs niet gedefinieerd wat de

routine doet als het resultaat van de berekening te groot wordt.

Deze routine kan het floating point getal berekenen dat de tienmacht van de ingevoerde integer is (dus 100 als resultaat van de parameter 2). Voor deze routine wordt gebruik gemaakt van de constanten 10 (10^1), 100 (10^2), 10000 (10^4), 100000000 (10^8) etc. Afhankelijk van de bits die geset zijn in de integer worden deze constanten al dan niet met elkaar vermenigvuldigd. Dus:

10^13 = 10 ^ 10001001 = 10^1 * 10^4 * 10^8

(^ = tot de macht).

De 10-logarithme van een getal.

Deze routine is de inverse van de voorgaande. Hierbij wordt bepaald wat de grootste macht van 10 is die in het (positieve) getal voorkomt. Bovendien wordt de rest (fractie) bepaald.

Deze routine werkt voor zowel getallen groter dan 1 (positieve machten van 10) als voor getallen kleiner dan 1 (negatieve machten van 10). De rest (fractie) wordt op de stack achtergelaten, de macht van 10 in een variabele op pagina 0.

Voor getallen groter dan 1 wordt door middel van proberen bepaald door welke 10 machten het getal deelbaar is. Dit betekent dat het resultaat van de deling groter dan of gelijk aan 1 moet blijven. Bij getallen tussen 0 en 1 wordt door middel van vermenigvuldigen gekeken welke machten van 10 bij vermenigvuldigen een resultaat kleiner dan 10 opleveren. Door in de variabele op pagina 0 deze machten bij elkaar op te tellen en eventueel het complement te nemen, krijgen we het resultaat van de logarithme. Datgene wat we aan het eind van de bewerking over houden is de fractie.

Conversie van een real naar ascii.

De bovenstaande routines zijn hulproutines voor de eigenlijke conversieroutines. De eerste routine die behandeld wordt is de afdrukroutine. Deze routine drukt niet zelf het getal af, maar geeft in een buffer een zogenaamde characterstring die eventueel aan een printroutine doorgegeven kan worden. Het startadres van deze string kan men in een variabele op pagina 0 schrijven. Het resultaat van de conversie wordt dan in dit buffer geschreven. Dit buffer dient 14 byte groot te zijn, waarbij in het 14e byte altijd een eind-teken (in dit geval ascii 0) geschreven wordt. Verder is het resultaat altijd in het volgende formaat weergegeven:

1) Een '-' teken of een spatie
2) Een cijfer
3) De decimale punt
4) Zes cijfers
5) De letter E als aanduiding van de exponent
6) Het teken van de exponent
7) Twee cijfers voor de exponent
8) Een afsluitteken

Het floating point getal staat op de stack en gaat bij de conversie verloren.

In de routine worden eerst enkele bijzondere gevallen onderzocht. Dit zijn de gevallen positieve en negatieve overflow (+ en - oneindig), positieve en negatieve underflow, het getal 0 en de aanduiding voor Not A Number.

Nadat deze byzondere gevallen afgesplitst zijn en het teken ingevuld is in het uitvoerbuffer, wordt de 10-logarithme bepaald. Dit levert in de eerste plaats de 10-exponent van het getal op. Bovendien is het resterende getal (de fractie) een getal tussen 1 en 10 (1 meegerekend, 10 niet). Vervolgens wordt een kopie van deze fractie geconverteerd naar een integer. Dit levert een van de cijfers 1 t/m 9 op waarvan het bijbehorende ascii-teken in het buffer geschreven wordt. Vervolgens wordt dit getal (1 t/m 9 dus) teruggeconverteerd naar een floating point getal en van de fractie afgetrokken. Daarna wordt het resultaat hiervan met 10 vermenigvuldigd. Op deze manier krijgen we een getal tussen 0 en 10 waarvan een kopie weer omgezet wordt naar een integer etc. Dit wordt een aantal keren herhaald.

De decimale punt staat na het eerste cijfer.

Als we voldoende cijfers op deze manier berekend hebben, gaan we afronden. We berekenen altijd een cijfer meer als het aantal dat we af willen drukken. Is dit cijfer een 5 of hoger, dan wordt het laatste cijfer met 1 verhoogd. Was dit voorlaatste cijfer een 9, dan wordt het een 0 en wordt het voor-voorlaatste

cijfer met 1 verhoogd. Dit kan zo doorgaan tot het voorste cijfer. Als dit een 9 is en het moet ook verhoogd worden, dan moet het hele getal een positie opgeschoven worden en moet de (10-) exponent met 1 verhoogd worden.

Na de afronding wordt de exponent (positief of negatief) omgezet naar twee cijfers en, met het teken, vastgelegd in het uitvoerbuffer.

Als laatste wordt het uitvoerbuffer afgesloten met een ascii-nul.

Conversie van ascii naar real.

Als laatste routine in deze aflevering is de uitvoerroutine opgenomen. Deze routine leest niet zelf een getal in, maar verwerkt een ascii-string in een invoerbuffer. Op pagina 0 wordt het startadres van dit buffer ingevuld en het ingelezen getal wordt over de stack teruggegeven. Het getal in het invoerbuffer moet afgesloten zijn met bijvoorbeeld een ascii 0.

Een floating point getal moet aan bepaalde formaatsregels voldoen. Deze regels staan in het commentaar van het programma opgesomd. Mocht het getal niet aan deze regels voldoen, dan wordt de constante NAN (Not A Number) teruggegeven.

De conversie zelf is vrij rechttoe rechtaan. De lettertekens die het getal vormen worden een voor een ingelezen en verwerkt. Dit kan in de eerste plaats een plus- of minteken zijn. Wordt er geen teken opgegeven, dan is het getal positief. Meer dan een teken is niet toegestaan. Hierna wordt de mantissa ingelezen. Deze mantissa bestaat uit cijfers en hooguit een decimale punt. De ingelezen cijfers worden stuk voor stuk omgezet naar een integer tussen 0 en 9 (grenzen inclusief) en opgeteld bij wat we al verzameld hadden maal 10. Op deze manier krijgen we dus een integer die de waarde van de mantissa aangeeft. Voor elk cijfer in de mantissa dat achter de decimale punt staat, wordt de (10-) exponent met 1 verlaagd.

De 10-exponent wordt ingelezen, uitgerekend en gecorrigeerd voor de cijfers achter de decimale punt.

Met behulp van de conversie-routine van integer naar floating point getal, wordt de 10-mantissa omgezet naar een floating point getal. De exponent wordt omgezet met behulp van 10 tot de macht X. Was de 10-exponent positief, dan worden de twee floating point getallen met elkaar vermenigvuldigd, was ze negatief, dan wordt de mantissa gedeeld door de exponent. Tenslotte wordt de stack in orde gemaakt.

Diverse opmerkingen.

De tot nu toe gepubliceerde routines vormen samen een basisset voor floating point operaties. Het is waarschijnlijk zinvol deze routines nog eens uit te breiden met routines voor machtsverheffen, trigoniometrische functies etc. Momenteel heb ik nog niet de rekenschema's (algorithmen) voor deze functies. Wel heb ik een nieuwe referentie hiervoor en weet ik een bibliotheek in Nederland waar ik het betreffende boek kan lenen. Als ik nog eens tijd en zin heb, ga ik ze misschien ook nog wel eens een keer maken.

Een tweede opmerking betreft de gepubliceerde programmatuur. Ik heb me bij het schrijven van de programma's geconcentreerd op de werking van de routines. Ik heb me niet druk gemaakt over optimalisaties met betrekking tot snelheid en geheugengebruik. Ik denk dat er nog wel wat optimalisaties mogelijk zijn en met name het intensieve gebruik van de processorstack is misschien iets dat veranderd moet worden. De 6502 heeft namelijk slechts een stack van 256 byte en als we parameters via de stack door gaan geven, kan dat wel eens schaarste gaan geven op de stack. Bij berekeningen met haakjes komt er bijvoorbeeld voor elk niveau een getal van 4 byte op de stack te staan, samen met een return-adres. Verder hebben de routines zelf soms ook nog wel enkele bytes stack-ruimte nodig.

Verder zijn er enkele zaken die specifiek op mijn systeem slaan. In de eerste aflevering heb ik de routines geassembleerd naar adres $0200. Toen het systeem echter groter werd, bleek ik daar onvoldoende ruimte te hebben. Op mijn systeem zijn daar alleen de pagina's 2 t/m 4 vrij; 5 t/m 7 zijn voor het operating systeem en vanaf pagina 8 zitten de Junior eproms. Kortom, zoals de routines afgedrukt zijn, kunnen ze dus niet op mijn systeem draaien. Ik heb daarom ook steeds voor het testen een ander stuk geheugen gebruikt maar heb, om de serie

artikelen uniform te houden de versie die afgedrukt zou gaan worden geassembleerd naar $0200. Hoe het op andere systemen zit weet ik niet, iedereen zal dat zelf moeten bekijken.

Een tweede punt is het gebruik van pagina 0. Op mijn systeem is pagina 0 bijna helemaal ongebruikt. Ik weet dat op andere systemen pagina 0 wel eens wat vol is. Ook op dit punt moet het programma mogelijk aangepast worden voordat ze op een ander systeem kan draaien. Het is volgens mij zonder meer mogelijk een aanzienlijk deel van de klad-variabelen elders in het geheugen te leggen. Alleen die variabelen die bij een indirecte addressering gebruikt worden, moeten in pagina 0 blijven liggen.

Tenslotte, wat kunnen we met dit floating point pakket?

Wel, een heleboel. In de eerste plaats zou er een programma gemaakt kunnen worden waarmee de computer als rekenmachine gebruikt kan gaan worden. (Heeft u wel eens geprobeerd op een computer een simpel rekensommetje uit te rekenen?). Verder denk ik dat het mogelijk moet zijn, het floating point pakket in de taal Forth te integreren. Ik heb dat al eens een keer gedaan en hoop dit ook voor dit pakket nog eens te gaan doen. Daarmee wordt naar mijn mening de taal Forth die op zich al krachtig is, nog een stuk krachtiger.

Tenslotte nog een opmerking over de verspreiding van het programma. Het programma is ontwikkeld op mijn Senior, een machine die maar weinig clubgenoten zullen hebben. Ik zal de sources echter op het bulletin board ter beschikking stellen en mogelijk zijn er dan bezitters van andere systemen die de moeite nemen de routines te converteren naar andere systemen. Als u geen modem hebt en toch graag een kopie van de routines wilt hebben, neem dan even contact met me op, dan zoeken we naar een oplossing want het overtikken van +/- 1600 regels source is in natuurlijk echt niet nodig.

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

0873  05F5                .OPT LIST
0874  05F5          ; ***********************************************************
0875  05F5          ;
0876  05F5          ; Floating point package for the 6502 microprocessor Part III
0877  05F5          ;
0878  05F5          ; Written by G. vn Opbroek
0879  05F5          ; on JUNIOR with Proton Senior DOS
0880  05F5          ; (c) Copyright 1989 Kim Gebruikersclub Nederland
0881  05F5          ;
0882  05F5          ; ***********************************************************
0883  05F5          ;
0884  05F5          ; Conversion of a 4 byte LONG INTEGER to a single
0885  05F5          ; precision IEEE floating point number
0886  05F5          ; Based on: Hagen Volzke Fliesskomma - Arithmetik und
0887  05F5          ;           IEEE-Spezifikation
0888  05F5          ;
0889  05F5          ;           mc 1/89 page 67
0890  05F5          ;
0891  05F5  BA    FLTOF   TSX                 ; Get the stack pointer
0892  05F6  A000          LDY #0              ; Byte counter
0893  05F8  BD0301 TOFL1  LDA $0100+3,X
0894  05FB  990700        STA MAN2,Y          ; Store integer in MAN2,EXP2
0895  05FE  E8            INX
0896  05FF  C8            INY
0897  0600  C004          CPY #4              ; Get four bytes from the stack
0898  0602  D0F4          BNE TOFL1
0899  0604          ;
0900  0604          ; Clear result (MAN1, EXP1, SIGN1)
0901  0604          ;
```

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

0604  A900              LDA #00
0606  A004              LDY #4              ; 5 byte
0608  990200 TOFL2      STA MAN1,Y
060B  88                DEY
060C  10FA              BPL TOFL2
060E            ;
060E            ; Test INTEGER for zero; when yes,
060E            ; result is zero too
060E            ;
060E  A003              LDY #3              ; Test four byte
0610  B90700 TOFL3      LDA MAN2,Y
0613  D005              BNE TOF1            ; <> 0 --> Continue
0615  88                DEY
0616  10F8              BPL TOFL3
0618  3035              BMI TOFRET          ; Integer is zero --> Result is zero
061A            ;
061A  A99F   TOF1       LDA #127+32         ; Exponent bias + 32 (bit)
061C  8505              STA EXP1
061E            ;
061E  A50A              LDA MAN2+3          ; MS-byte of the integer
0620  1016              BPL TOF2            ; Integer is positive
0622  A901              LDA #1              ; Negative integer; save sign
0624  8506              STA SIGN1
0626            ;
0626            ; Integer is negative, get 2's complement
0626            ; by subtracting the integer from zero
0626            ;
0626  38                SEC
0627  A000              LDY #0
0629  A900   TOFL4      LDA #0
062B  F90700            SBC MAN2,Y
062E  990700            STA MAN2,Y
0631  C8                INY
0632  C004              CPY #4
0634  D0F3              BNE TOFL4
0636            ;
0636  900C              BCC TOF3            ; Carry is clear --> number is
                                            ; already normalized
0638            ;
0638            ; Normalize number by shifting until carry is set
0638            ;
0638  18     TOF2       CLC
0639  C605   TOFL5      DEC EXP1            ; Decrement exponent
063B  A004              LDY #4              ; Rotate 4 byte
063D  A207              LDX #<MAN2
063F  200002            JSR ROTLEF
0642  90F5              BCC TOFL5           ; IF Carry clear, continue shifting
0644            ;
0644            ; Copy result to MAN1 (3 byte)
0644            ;
0644  A002   TOF3       LDY #2              ; 3 byte
0646  B90800 TOFL6      LDA MAN2+1,Y        ; MS-byte is in EXP2
0649  990200            STA MAN1,Y
064C  88                DEY
064D  10F7              BPL TOFL6
064F            ;
064F  4606   TOFRET     LSR SIGN1           ; Get sign bit
0651  A205              LDX #<MAN1+3        ; Shift in sign bit
0653  A004              LDY #4              ; Rotate 4 byte
0655  200702            JSR ROTRIG
```

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

0658            ;
0658            ; Save result on stack
0658            ;
0658  BA                TSX Get stack pointer
0659  A003               LDY #3              ; Save 4 byte
065B  B90200 TOFL7       LDA MAN1,Y          ; Get byte
065E  9D0601             STA $0100+6,X
0661  CA                 DEX
0662  88                 DEY
0663  10F6               BPL TOFL7
0665  60                 RTS
0666            ;
0666            ;
0666            ; Conversion of a single precision IEEE floating
0666            ; number to a 4 byte LONG INTEGER
0666            ; Based on: Hagen Volzke Fliesskomma - Arithmetik und
0666            ;           IEEE-Spezifikation
0666            ;
0666            ;           mc 1/89 page 67
0666            ;
0666  BA      FFTOL      TSX                  ; Get the stack pointer
0667  A000               LDY #0              ; Byte counter
0669  BD0301 TOLL1       LDA $0100+3,X
066C  990700             STA MAN2,Y          ; Store float in MAN2,EXP2
066F  E8                 INX
0670  C8                 INY
0671  C004               CPY #4              ; Get four bytes from the stack
0673  D0F4               BNE TOLL1
0675            ;
0675            ; Rotate floating point number to get result in one byte
0675                                         ;
0675  A900               LDA #0
0677  850B               STA SIGN2           ; Clear sign byte 2
0679  18                 CLC
067A  A207               LDX #<MAN2          ; Zero page relative address
067C  A005               LDY #5              ; 5 byte
067E  200002             JSR ROTLEF
0681            ;
0681  38                 SEC                 ; Subtract bias from the exponent
0682  A50A               LDA <EXP2
0684  E97F               SBC #$7F
0686  850A               STA EXP2
0688            ;
0688  3045               BMI TOLZER          ; Negative exponent --> result is zero
068A            ;
068A  C91F               CMP #31             ; Exponent >= 31
068C  104D               BPL TOLOVL          ; Yes, --> overflow
068E            ;
068E  38                 SEC                 ; A normalized number, ROR in a one
068F  A204               LDX #4              ; 4 byte
0691  A209               LDX #<MAN2+2
0693  A003               LDY #3              ; 3 byte
0695  200702             JSR ROTRIG          ; Rotate mantissa
0698            ;
0698  A900               LDA #0              ; LS byte becomes zero
069A  8506               STA SIGN1           ; Use signl for the LS-byte
069C            ;
069C            ; Rotate mantissa right until exponent = 31
069C            ;
069C  18      TOLL2      CLC                 ; Shift in zero
```

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

069D  A209                LDX #<MAN2+2
069F  A004                LDY #4              ; Rotate 4 byte
06A1  200702              JSR ROTRIG
06A4  E60A                INC EXP2            ; Increment exponent
06A6  A50A                LDA EXP2            ; Get exponent for test
06A8  C91F                CMP #31             ; Exponent = 31 ?
06AA  D0F0                BNE TOLL2           ; No --> continue shifting
06AC            ;
06AC  A50B      TOLRET    LDA SIGN2           ; Get sign
06AE  F011                BEQ TOLRE1          ; Positive
06B0            ;
06B0            ; Construct 2's complement
06B0            ;
06B0  38                  SEC                 ; Set carry to add 1 to result
06B1  A004                LDY #4              ; negate 4 bytes
06B3  A206                LDX #<MAN2-1        ; Start at LS-byte
06B5  B500      TOLL3     LDA $00,X           ; Get byte
06B7  49FF                EOR #$FF            ; Negate byte
06B9  6900                ADC #$00            ; Add 1 to result
06BB  9500                STA $00,X           ; Store complement
06BD  E8                  INX
06BE  88                  DEY
06BF  D0F4                BNE TOLL3
06C1            ;
06C1            ; Save result on stack
06C1            ;
06C1  BA        TOLRE1    TSX                 ; Get stack pointer
06C2  A003                LDY #3              ; Save 4 byte
06C4  B90600    TOLL4     LDA MAN2-1,Y        ; Get byte
06C7  9D0601              STA $0100+6,X
06CA  CA                  DEX
06CB  88                  DEY
06CC  10F6                BPL TOLL4
06CE  60                  RTS
06CF            ;
06CF            ; Zero result
06CF            ;
06CF  A900      TOLZER    LDA #0
06D1  A003                LDY #3              ; Clear 4 byte
06D3  990600    TOLL5     STA MAN2-1,Y
06D6  88                  DEY
06D7  10FA                BPL TOLL5
06D9  30E6                BMI TOLRE1
06DB            ;
06DB            ; Overflow (positive or negative)
06DB            ;
06DB  A9FF      TOLOVL    LDA #$FF            ; Maxint = $7F $FF $FF $FF
06DD  8506                STA MAN2-1
06DF  8507                STA MAN2
06E1  8508                STA MAN2+1
06E3  A980                LDA #$80
06E5  8509                STA MAN2+2
06E7            ;
06E7  A50B                LDA SIGN2           ; Result > 0 ?
06E9  D0D6                BNE TOLRE1          ; Yes, return
06EB            ;
06EB  A003                LDY #3              ; Negate 4 bytes --> MININT
06ED  B90600    TOLL6     LDA MAN2-1,Y
06F0  49FF                EOR #$FF            ; Negate --> 1's complement
06F2  990600              STA MAN2-1,Y
```

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

06F5  88                DEY
06F6  10F5              BPL TOLL6
06F8  30C7              BMI TOLRE1
06FA            ;
06FA            ; Define some constants
06FA            ;
06FA          P10T                        ; Table with powers of 10
06FA  00      C1E0      .BYTE $00,$00,$80,$3F; 1
06FB  00
06FC  80
06FD  3F
06FE  00      C1E1      .BYTE $00,$00,$20,$41; 10
06FF  00
0700  20
0701  41
0702  00      C1E2      .BYTE $00,$00,$C8,$42; 100
0703  00
0704  C8
0705  42
0706  00      C1E4      .BYTE $00,$40,$1C,$46; 10000
0707  40
0708  1C
0709  46
070A  20      C1E8      .BYTE $20,$BC,$BE,$4C; 100000000
070B  BC
070C  BE
070D  4C
070E  CA      C1E16     .BYTE $CA,$1B,$0E,$5A; 10000000000000000
070F  1B
0710  0E
0711  5A
0712  AE      C1E32     .BYTE $AE,$C5,$9D,$74; 100000000000000000000000000000
0713  C5
0714  9D
0715  74
0716  00      COVL      .BYTE $00,$00,$80,$7F; Positive overflow
0717  00
0718  80
0719  7F
071A  FF      CNAN      .BYTE $FF,$FF,$FF,$7F; Not A Number
071B  FF
071C  FF
071D  7F
071E            ;
0019                    .EXI
0019          EXSAVE    *=*+1               ; 10-exponent
001A          POINTI    *=*+2               ; Pointer to a floating point number
001C          FOUTPU    *=*+2               ; Pointer to output buffer
001E          FINPUT    *=*+2               ; Pointer to input buffer
0020            ;
0020          EXP10     *=*+1               ; Power of 10
0021          COUN10    *=*+1               ; Bit counter
0022          COUNCV    *=*+1               ; Digit counter
0023          TWOPX     *=*+1
0024          CFLAGS    *=*+1               ; Flags for conversion
0025            ;
0025          NMASK=%00000001
0025          ENMASK=%00000010
0025          DMASK=%00000100
0025          AMASK=%00001000               ; Power of 10 is negative
```

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

071E                    .EX1
071E            ;
071E            ; F10PX: 10 to the power X --> Result = float
071E            ; X is a 4 byte integer small enough to get a correct result
071E            ; Based on: Hagen Volzke Fliesskomma - Arithmetik und
071E            ;           IEEE-Spezifikation
071E            ;
071E            ;           mc 2/89 page 67
071E            ;
071E            ; Initialize on 1E0
071E            ;
071E            ;
071E  A003   F10PX     LDY #3              ; 4 BYTE
0720  A9FA             LDA #<C1E0          ; Get address of 1e0
0722  851A             STA POINT1          ; Initialize pointer
0724  A906             LDA #>C1E0
0726  851B             STA POINT1+1
0728            ;
0728  B9FA06 F10PX1    LDA C1E0,Y          ; Get byte
072B  48               PHA                 ; Save on stack
072C  88               DEY
072D  10F9             BPL F10PX1          ; Next byte
072F            ;
072F            ; Get LS-byte of the 10-exponent
072F            ;
072F  BA               TSX
0730  BD0701           LDA $100+7,X        ;
0733  8519             STA EXSAVE          ; Save the exponent on page zero
0735  F01F   F10PX2    BEQ F10PX3          ; WHILE exponent > 0 DO
0737  18               CLC                 ; Increase pointer
0738  A51A             LDA POINT1
073A  6904             ADC #$04            ; Length of float = 4
073C  851A             STA POINT1
073E  A51B             LDA POINT1+1
0740  6900             ADC #$00
0742  851B             STA POINT1+1
0744            ;
0744            ; Get next bit of exponent
0744            ;
0744  4619             LSR EXSAVE
0746  90ED             BCC F10PX2          ; Bit is clear --> next bit
0748            ;
0748            ; Copy power of 10 on stack
0748            ;
0748  A003             LDY #03 , 4 byte
074A  B11A   F10PX4    LDA (POINT1),Y
074C  48               PHA
074D  88               DEY                 ; Next byte
074E  10FA             BPL F10PX4
0750            ;
0750            ; Multiply (partial) result by power of 10
0750            ;
0750  205103           JSR FMUL            ; Yes, multiply
0753  4C3507           JMP F10PX2
0756            ;
0756  A003   F10PX3    LDY #3              ; Move 4 byte from the top of the
                                           ; stack
0758  68     F10PX5    PLA                 ; Get byte
0759  BA               TSX                 ; Get top of stack
075A  9D0601           STA $100+6,X
```

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

075D  88                 DEY
075E  10F8               BPL F10PX5          ; Next byte
0760  60                 RTS
0761              ;
0761              ; F10LG: Calculate the 10's log of a positive FP-number
0761              ; The result is stored in EXP10 (integer part) and in the
0761              ; parameter (fraction).
0761              ; Based on: Hagen Volzke Fliesskomma - Arithmetik und
0761              ;           IEEE-Spezifikation
0761              ;
0761              ;
0761              ;           mc 2/89 page 67
0761              ;
0761  A003   F10LG       LDY #3              ; Copy 4 bytes from parameter to top
                                             ; of stack
0763  BA     F10L1       TSX                 ; Copy stack pointer
0764  BD0601             LDA $100+6,X        ; Get byte
0767  48                 PHA
0768  88                 DEY
0769  10F8               BPL F10L1           ; Next byte
076B              ;
076B  A905               LDA #5              ; Initialize bit counter
076D  8521               STA COUN10
076F  A920               LDA #32             ; 2 power 5
0771  8523               STA TWOPX
0773  A900               LDA #0              ; Clear result
0775  8520               STA EXP10
0777  A912               LDA #<C1E32         ; initialize pointer on 10e32
0779  851A               STA POINT1
077B  A907               LDA #>C1E32
077D  851B               STA POINT1+1
077F              ;
077F  A003               LDY #3              ; Copy 4 byte of 1E0 on stack
0781  B9FA06 F10L2       LDA C1E0,Y          ; Get byte
0784  48                 PHA                 ; Save on stack
0785  88                 DEY
0786  10F9               BPL F10L2           ; Next byte
0788              ;
0788  203103             JSR FSUB            ; Subtract bytes to test on < 1E0
078B              ;
078B  68                 PLA                 ; Drop result
078C  68                 PLA
078D  68                 PLA
078E  68                 PLA
078F              ;
078F  305D               BMI F10LN4          ; FP-number < 1
0791              ;
0791  A003   F10LP4      LDY #3              ; Copy 4 bytes
0793  BA     F10LP5      TSX                 ; Get stack pointer
0794  BD0601             LDA $0100+6,X       ; Get byte
0797  48                 PHA
0798  88                 DEY
0799  10F8               BPL F10LP5          ; Next byte
079B              ;
079B  A003               LDY #3              ; Copy 4 bytes
079D  B11A   F10LP6      LDA (POINT1),Y
079F  48                 PHA                 ; Push byte
07A0  88                 DEY
07A1  10FA               BPL F10LP6          ; Next byte
07A3              ;
```

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

07A3  20B704          JSR FDIV          ; Divide actual mantissa by actual
                                        ; power of 10
07A6            ;
07A6            ; Copy result for compare with 1
07A6            ;
07A6  A003            LDY #3            ; Copy 4 byte
07A8  BA      F10PLA  TSX               ; Get stack pointer
07A9  BD0401          LDA $0100+4,X     ; Get byte
07AC  48              PHA
07AD  88              DEY
07AE  10F8            BPL F10PLA
07B0            ;
07B0  A003            LDY #3            ; Copy 4 byte
07B2  B9FA06  F10LP7  LDA C1E0,Y
07B5  48              PHA
07B6  88              DEY
07B7  10F9            BPL F10LP7
07B9            ;
07B9  203103          JSR FSUB          ; Result > 1 ?
07BC  68              PLA               ; Drop result of subtraction
07BD  68              PLA
07BE  68              PLA
07BF  68              PLA
07C0  3014            BMI F10LP8
07C2                                    ;
07C2  A523            LDA TWOPX
07C4  18              CLC               ; Add actual power of 2 to result
07C5  6520            ADC EXP10
07C7  8520            STA EXP10
07C9            ;
07C9  A003            LDY #3            ; Copy actual mantissa to result
07CB  BA              TSX               ; Get stack pointer
07CC  BD0101  F10LP9  LDA $0100+1,X     ; Get byte
07CF  9D0701          STA $0100+7,X     ; Copy byte
07D2  E8              INX
07D3  88              DEY
07D4  10F6            BPL F10LP9
07D6            ;
07D6  68      F10LP8  PLA               ; Remove top fp-number
07D7  68              PLA
07D8  68              PLA
07D9  68              PLA
07DA  4623            LSR TWOPX         ; Decrease power of two
07DC  38              SEC               ; Decrease actual power of 10
07DD  A51A            LDA POINT1
07DF  E904            SBC #4            ; By decreasing the address
07E1  851A            STA POINT1
07E3  A51B            LDA POINT1+1
07E5  E900            SBC #0
07E7  851B            STA POINT1+1
07E9            ;
07E9  C621            DEC COUN10        ; 6 times ?
07EB  10A4            BPL F10LP4
07ED  60              RTS
07EE            ;
07EE  A003    F10LN4  LDY #3            ; Copy 4 bytes
07F0  BA      F10LN5  TSX               ; Get stack pointer
07F1  BD0601          LDA $0100+6,X     ; Get byte
07F4  48              PHA
07F5  88              DEY
```

```
6502 Floating Point Package    PROTON 650X ASSEMBLER  V4.4    PAGE: 0009

07F6  10F8            BPL F10LN5        ; Next byte
07F8            ;
07F8  A003            LDY #3            ; Copy 4 bytes
07FA  B11A    F10LN6  LDA (POINT1),Y
07FC  48              PHA               ; Push byte
07FD  88              DEY
07FE  10FA            BPL F10LN6        ; Next byte
0800            ;
0800  205103          JSR FMUL          ; Multiply actual mantissa by actual
                                        ; power of 10
0803            ;
0803            ; Copy result for compare with 10
0803            ;
0803  A003            LDY #3            ; Copy 4 byte
0805  BA      F10NLA  TSX               ; Get stack pointer
0806  BD0401          LDA $0100+4,X     ; Get byte
0809  48              PHA
080A  88              DEY
080B  10F8            BPL F10NLA
080D            ;
080D  A003            LDY #3            ; Copy 4 byte
080F  B9FE06  F10LN7  LDA C1E1,Y
0812  48              PHA
0813  88              DEY
0814  10F9            BPL F10LN7
0816            ;
0816  203103          JSR FSUB          ; Result >= 10 ?
0819  68              PLA               ; Drop result of subtraction
081A  68              PLA
081B  68              PLA
081C  68              PLA
081D  1014            BPL F10LN8
081F                                    ;
081F  A523            LDA TWOPX
0821  18              CLC               ; Add actual power of 2 to result
0822  6520            ADC EXP10
0824  8520            STA EXP10
0826            ;
0826  A003            LDY #3            ; Copy actual mantissa to result
0828  BA              TSX               ; Get stack pointer
0829  BD0101  F10LN9  LDA $0100+1,X     ; Get byte
082C  9D0701          STA $0100+7,X     ; Copy byte
082F  E8              INX
0830  88              DEY
0831  10F6            BPL F10LN9
0833            ;
0833  4623    F10LN8  LSR TWOPX         ; Decrease power of two
0835  38              SEC               ; Decrease actual power of 10
0836  A51A            LDA POINT1
0838  E904            SBC #4            ; By decreasing the address
083A  851A            STA POINT1
083C  A51B            LDA POINT1+1
083E  E900            SBC #0
0840  851B            STA POINT1+1
0842            ;
0842  68              PLA               ; Remove scratch
0843  68              PLA
0844  68              PLA
0845  68              PLA
0846            ;
```

```
6502 Floating Point Package    PROTON 650X ASSEMBLER  V4.4   PAGE: 0010

0846  C621            DEC COUN10        ; 6 times ?
0848  10A4            BPL F10LN4
084A            ;
084A  38              SEC               ; Negate result
084B  A900            LDA #0
084D  E520            SBC EXP10
084F  8520            STA EXP10
0851  60              RTS
0852            ;
0852            ;
0852            ; Conversion of a single precision IEEE floating
0852            ; number to an ASCII string. Format: -D.DDDDDDE+DD
0852            ;
0852            ; The floating point number is on the stack; a pointer to
0852            ; the output buffer is passed in the (Zero Page) variable
0852            ; FOUTPU.
0852            ;
0852            ; Based on: Hagen Volzke Fliesskomma - Arithmetik und
0852            ;           IEEE-Spezifikation
0852            ;
0852            ;           mc 2/89 page 67
0852            ;
0852  A00C    FFTOA   LDY #12           ; Clear 13 byte in the output buffer
0854  A920            LDA #' '
0856  911C    FTOA1   STA (FOUTPU),Y    ; Clear byte
0858  88              DEY
0859  10FB            BPL FTOA1
085B            ;
085B            ; Rotate the top of the stack 2 times over 4 byte
085B            ; to get the FP-number on top
085B            ;
085B  BA              TSX               ; Get stack pointer
085C  BD0101          LDA $0101,X       ; Get return address
085F  8500            STA RETADD
0861  BD0201          LDA $0102,X
0864  8501            STA RETADD+1
0866            ;
0866  A003            LDY #3            ; Move 4 byte
0868  BD0301  FTOAL   LDA $103,X        ; Get byte
086B  9D0101          STA $101,X
086E  E8              INX
086F  88              DEY
0870  10F6            BPL FTOAL
0872  A500            LDA RETADD        ; Restore return address
0874  9D0101          STA $101,X
0877  E8              INX
0878  A501            LDA RETADD+1
087A  9D0101          STA $101,X
087D            ;
087D            ; Get the sign bit (in the MS Word)
087D            ;
087D  BA              TSX               ; Get stack pointer
087E  BD0401          LDA $100+4,X      ; Get MS byte of FP-number
0881  100B            BPL FTOA2         ; Positive number
0883  297F            AND #$7F          ; Clear sign bit
0885  9D0401          STA $100+4,X      ; Restore FP-number
0888  A92D            LDA #$'-'         ; Write sign in buffer
088A  A000            LDY #0
088C  911C            STA (FOUTPU),Y
088E            ;
```

```
6502 Floating Point Package    PROTON 650X ASSEMBLER  V4.4   PAGE: 0011

088E                ;
088E                ; Some special cases: EEEEEEEEEE Positive/Negative overflow
088E                ;                     ??????????? Not A Number
088E                ;                               0 Zero
088E                ;
088E  BA      FTOA2     TSX                  ; Get stack pointer
088F  A003              LDY #3               ; Compare 4 byte
0891  BD0401  FTOA3     LDA $100+4,X         ; Compare FP-number with overflow
0894  D91607            CMP COVL,Y
0897  D010              BNE FTOA4            ; Not equal, continue
0899  CA                DEX
089A  88                DEY
089B  10F4              BPL FTOA3            ; Next byte
089D                ;
089D                ; It is overflow ........
089D                ;
089D  A00C              LDY #12
089F  A945              LDA #'E'
08A1  911C    FTOA5     STA (FOUTPU),Y
08A3  88                DEY
08A4  D0FB              BNE FTOA5            ; FOUTPU[0] contains sign
08A6  4C9C09            JMP FTOAE            ; Exit
08A9                ;
08A9  BA      FTOA4     TSX                  ; Get stBck pointer
08AA  A003              LDY #3               ; Compare 4 byte
08AC  BD0401  FTOA6     LDA $100+4,X         ; Compare FP-number with NAN
08AF  D91A07            CMP CNAN,Y
08B2  D010              BNE FTOAZ            ; Not equal, continue
08B4  CA                DEX
08B5  88                DEY
08B6  10F4              BPL FTOA6            ; Next byte
08B8                ;
08B8                ; It is not a number ........
08B8                ;
08B8  A00C              LDY #12
08BA  A93F              LDA #'?'
08BC  911C    FTOA7     STA (FOUTPU),Y
08BE  88                DEY
08BF  D0FB              BNE FTOA7            ; FOUTPU[0] contains sign
08C1  4C9C09            JMP FTOAE            ; Exit
08C4                ;
08C4  BA      FTOAZ     TSX                  ; Get stack pointer
08C5  A003              LDY #3               ; Compare 4 byte
08C7  BD0401  FTOAZ1    LDA $100+4,X         ; Compare FP-number with zero
08CA  C900              CMP #0
08CC  D00D              BNE FTOA8            ; Not equal, continue
08CE  CA                DEX
08CF  88                DEY
08D0  10F5              BPL FTOAZ1           ; Next byte
08D2                ;
08D2                ; It is zero
08D2                ;
08D2  A00C              LDY #12
08D4  A930              LDA #'0'
08D6  911C              STA (FOUTPU),Y
08D8  4C9C09            JMP FTOAE            ; Exit
08DB                ;
08DB  206107  FTOA8     JSR F10LG            ; 0 <= Result < 10, exponent in EXP10
08DE  A901              LDA #1               ; Calculate 8 digits; start at digit 1
08E0  8522              STA COUNCV
```

```
6502 Floating Point Package   PROTON 650X ASSEMBLER V4.4   PAGE: 0012

08E2            ;
08E2  A003   FTOAR    LDY #3              ; Copy 4 byte
08E4  BA     FTOA9    TSX                 ; Get Stack pointer
08E5  BD0401          LDA $104,X          ; Get byte
08E8  48              PHA
08E9  88              DEY
08EA  10F8            BPL FTOA9
08EC            ;
08EC  206606          JSR FFTOL           ; Get integer part
08EF  68              PLA                 ; Get 0 <= Integer <= 9
08F0  48              PHA
08F1  18              CLC
08F2  6930            ADC #'0'
08F4  A422            LDY COUNCV
08F6  911C            STA (FOUTPU),Y      ; Store digit
08F8            ;
08F8  E622            INC COUNCV
08FA  A522            LDA COUNCV
08FC  C909            CMP #9              ; 8th digit?
08FE  F015            BEQ FTOA11          ; Yes, exit
0900            ;
0900  20F505          JSR FLTOF           ; Convert digit back to float
0903  203103          JSR FSUB            ; Subtract digit
0906  A003            LDY #3              ; Copy 4 byte
0908  B9FE06 FTOA10   LDA C1E1,Y          ; Copy 10 to stack
090B  48              PHA
090C  88              DEY
090D  10F9            BPL FTOA10
090F            ;
090F  205103          JSR FMUL            ; Next digit......
0912  4CE208          JMP FTOAR
0915            ;
0915  68     FTOA11   PLA                 ; Remove last integer
0916  68              PLA
0917  68              PLA
0918  68              PLA
0919            ;
0919            ; Round result ......
0919            ;
0919  A008            LDY #8
091B  B11C   FTOA12   LDA (FOUTPU),Y
091D  C935            CMP #'5'
091F  9028            BCC FTOA20
0921  88     FTOA15   DEY                 ; Previous digit
0922  B11C            LDA (FOUTPU),Y
0924  C939            CMP #'9'
0926  F008            BEQ FTOA13
0928  18              CLC
0929  6901            ADC #1              ; Add 1 to digit
092B  911C            STA (FOUTPU),Y
092D  4C4909          JMP FTOA20
0930  A930   FTOA13   LDA #'0'            ; Digit = zero
0932  911C            STA (FOUTPU),Y
0934  D0EB            BNE FTOA15          ; Continue with next digit
0936            ;
0936            ; Overflow on last digit --> increase exponent
0936            ; and shift result
0936            ;
0936  E620            INC EXP10
0938  A008            LDY #8
```

```
6502 Floating Point Package    PROTON 650X ASSEMBLER  V4.4    PAGE: 0013

093A  B11C   FTOA14   LDA (FOUTPU),Y
093C  C8              INY
093D  911C            STA (FOUTPU),Y     ; Copy byte
093F  88              DEY
0940  88              DEY
0941  D0F7            BNE FTOA14         ; Index 0 = sign
0943  A931            LDA #'1'           ; MS Digit
0945  A001            LDY #1
0947  911C            STA (FOUTPU),Y
0949          ;
0949          ; Shift result and include '.' and 'E'
0949          ;
0949  A009   FTOA20   LDY #9
094B  A945            LDA #'E'           ; Exponent
094D  911C            STA (FOUTPU),Y
094F  88              DEY
0950  88              DEY
0951  B11C   FTOA21   LDA (FOUTPU),Y     ; Get digit
0953  C8              INY
0954  911C            STA (FOUTPU),Y     ; Store digit
0956  88              DEY
0957  88              DEY
0958  D0F7            BNE FTOA21         ; Index > 0 --> Continue
095A  A002            LDY #2
095C  A92E            LDA #'.'
095E  911C            STA (FOUTPU),Y     ; Store decimal point
0960          ;
0960          ; Format exponent
0960          ;
0960  A00A            LDY #10            ; Index 10 contains exp-sign
0962  A92B            LDA #'+'
0964  911C            STA (FOUTPU),Y     ; Intialize on +
0966  A520            LDA EXP10          ; Get exponent
0968  100A            BPL FTOA22         ; Sign is correct
096A  49FF            EOR #$FF
096C  8520            STA EXP10
096E  E620            INC EXP10          ; Store 2's complement
0970  A92D            LDA #'-'           ; Minus
0972  911C            STA (FOUTPU),Y
0974          ;
0974         FTOA22
0974  A00B            LDY #11            ; Index 11 is first digit of exponent
0976  A930            LDA #'0'           ; Initialize on zero
0978  911C            STA (FOUTPU),Y
097A  18              CLC
097B  A900            LDA #0
097D  690A   FTOA23   ADC #10            ; Increment A by 10
097F  C520            CMP EXP10          ; exponent > A ?
0981  100A            BPL FTOA24
0983  48              PHA                ; Save accu
0984  B11C            LDA (FOUTPU),Y     ; Get MS digit of exponent
0986  6901            ADC #1             ; Increment digit
0988  911C            STA (FOUTPU),Y
098A  68              PLA                ; Restore accu
098B  90F0            BCC FTOA23
098D          ;
098D         FTOA24
098D  49FF            EOR #$FF           ; Negate accu
098F  18              CLC
0990  690B            ADC #1+10          ; 2's complement of last digit*10
```

```
0992  18                CLC
0993  6520              ADC EXP10          ; Add exponent to get ls digit
0995  18                CLC
0996  6930              ADC #'0'           ; Ascii
0998  A00C              LDY #12
099A  911C              STA (FOUTPU),Y     ; Store digit
099C            ;
099C  68        FTCAE   PLA                ; Remove scratch
099D  68                PLA
099E  68                PLA
099F  68                PLA
09A0  A900              LDA #0             ; Terminate output buffer
09A2  A00D              LDY #13
09A4  911C              STA (FOUTPU),Y
09A6  60                RTS
09A7            ;
09A7            ;
09A7            ; Conversion of an ASCII string to a single precision IEEE
09A7            ; stack, the ASCII string is passed by the pointer FINPUT on
09A7            ; page zero.
09A7            ;
09A7            ; Leading spaces and tabs are ignored and the ASCII string
09A7            ; should be terminated by a non-digit.
09A7            ;
09A7            ; The number may be proceeded by an optional sign. Before the
09A7            ; decimal point and behind the E or e and an optional sign of
09A7            ; the exponent should at least be one digit.
09A7            ;
09A7            ; Based on: Hagen Volzke Fliesskomma - Arithmetik und
09A7            ;           IEEE-Spezifikation
09A7            ;
09A7            ;           mc 2/89 page 67
09A7            ;
09A7  A900      FATOF   LDA #0             ; Clear flags
09A9  8524              STA CFLAGS
09AB  A8                TAY                ; Clear index on string
09AC  8520              STA EXP10          ; Clear exponent
09AE  A003              LDY #3             ; Clear 4 byte on page zero
09B0  991500    ATOF22  STA SAVE,Y
09B3  88                DEY
09B4  10FA              BPL ATOF22
09B6            ;
09B6            ; Skip leading spaces and tabs
09B6            ;
09B6  C8        ATOF1   INY                ; Y was $FF the first time
09B7  B11E              LDA (FINPUT),Y     ; Get next character
09B9  C920              CMP #' '
09BB  F0F9              BEQ ATOF1          ; Skip Character
09BD  C909              CMP #$09
09BF  F0F5              BEQ ATOF1          ; Skip Character
09C1            ;
09C1            ; Do we have a sign?
09C1            ;
09C1  C92B              CMP #'+'           ; Plus ?
09C3  F00E              BEQ ATOF2          ; Yes, continue
09C5  C92D              CMP #'-'           ; Minus
09C7  F004              BEQ ATOF3
09C9  88                DEY                ; No sign, decrement pointer
09CA  4CD309            JMP ATOF2
09CD  A901      ATOF3   LDA #NMASK         ; Negative
```

```
6502 Floating Point Package    PROTON 650X ASSEMBLER  V4.4   PAGE: 0015

09CF  0524              ORA CFLAGS
09D1  8524              STA CFLAGS          ; Store negative flag
09D3            ;
09D3  C8      ATOF2     INY                 ; Get next character, this should be
                                            ; a digit
09D4  B11E              LDA (FINPUT),Y
09D6  C930              CMP #'0'            ; < '0'
09D8  9006              BCC ATOF40          ; Yes, Not A Number
09DA  C93A              CMP #':'            ; > 9 ?
09DC  B002              BCS ATOF40          ; Yes, Not A Number
09DE  9003              BCC ATOF4
09E0  4C280B  ATOF40    JMP ATOFNA          ; Branch is out of range
09E3            ;
09E3  B11E    ATOF4     LDA (FINPUT),Y      ; Get character
09E5  AA                TAX                 ; Save character
09E6  A524              LDA CFLAGS          ; Get the flags
09E8  2904              AND #DMASK          ; Already a decimal point detected ?
09EA  D00E              BNE ATOF5           ; Yes, skip the next test
09EC  E02E              CPX #'.'            ; Test for decimal point
09EE  D00A              BNE ATOF5
09F0  A904              LDA #DMASK          ; Flag decimal point
09F2  0524              ORA CFLAGS
09F4  8524              STA CFLAGS
09F6  C8                INY                 ; Increment index
09F7  4CE309            JMP ATOF4           ; Get the next character
09FA            ;
09FA  8A      ATOF5     TXA                 ; Restore character
09FB  C930              CMP #'0'            ; < '0'
09FD  9068              BCC ATOF6           ; Yes, end of mantissa
09FF  C93A              CMP #':'            ; > '9'
0A01  B064              BCS ATOF6
0A03            ;
0A03  98                TYA                 ; Save Y on the stack
0A04  48                PHA
0A05  A003              LDY #3              ; Copy 4 byte
0A07  B91500  ATOF10    LDA SAVE,Y
0A0A  990D00            STA WORK,Y
0A0D  88                DEY
0A0E  10F7              BPL ATOF10
0A10            ;
0A10            ; Multiply the contains of the working space by 10
0A10            ; and add the next digit.
0A10            ;
0A10  18                CLC                 ; Multiply by 10: Shift 2 bit, add,
0A11  A004              LDY #4              ; 4 byte                shift 1 bit
0A13  A20D              LDX #<WORK
0A15  200002            JSR ROTLEF
0A18  18                CLC
0A19  A004              LDY #4
0A1B  A20D              LDX #<WORK
0A1D  200002            JSR ROTLEF
0A20            ;
0A20  18                CLC
0A21  A000              LDY #0              ; Add WORK to SAVE
0A23  08                PHP                 ; Save carry
0A24  B90D00  ATOF11    LDA WORK,Y
0A27  28                PLP                 ; Restore carry
0A28  791500            ADC SAVE,Y
0A2B  991500            STA SAVE,Y
0A2E  08                PHP                 ; Save carry bit
```

```
6502 Floating Point Package     PROTON 650X ASSEMBLER  V4.4    PAGE: 0016

0A2F  C8                INY
0A30  C004              CPY #4              ; End ?
0A32  D0F0              BNE ATOF11
0A34  28                PLP
0A35            ;
0A35  A004              LDY #4              ; Rotate 4 byte
0A37  A215              LDX #SAVE
0A39  18                CLC
0A3A  200002            JSR ROTLEF
0A3D            ;
0A3D  68                PLA                 ; Restore Y
0A3E  A8                TAY
0A3F  38                SEC
0A40  B11E              LDA (FINPUT),Y      ; Get character
0A42  38                SEC
0A43  E930              SBC #'0'            ; Get digit value
0A45  18                CLC
0A46  6515              ADC SAVE            ; Add to LS byte
0A48  8515              STA SAVE
0A4A  A201              LDX #1
0A4C  08                PHP                 ; Save carry
0A4D  A900      ATOF13  LDA #0              ; Add carry to next byte
0A4F  28                PLP                 ; Get saved carry
0A50  7515              ADC SAVE,X
0A52  9515              STA SAVE,X
0A54  08                PHP                 ; Save carry
0A55  E8                INX
0A56  E004              CPX #4              ; MS-byte ?
0A58  D0F3              BNE ATOF13
0A5A  28                PLP
0A5B            ;
0A5B            ; Decrement the exponent for all digits behind the point
0A5B            ;
0A5B  A524              LDA CFLAGS
0A5D  2904              AND #DMASK          ; Select flag
0A5F  F002              BEQ ATOF41          ; Process next character
0A61  C620              DEC EXP10           ; Decrement exponent
0A63  C8        ATOF41  INY                 ; Next index
0A64  4CE309            JMP ATOF4           ; Process next character
0A67            ;
0A67            ; The optional exponent.......
0A67            ;
0A67  A900      ATOF6   LDA #0              ; Clear exponent
0A69  850D              STA WORK
0A6B            ;
0A6B  B11E              LDA (FINPUT),Y      ; Get character
0A6D  295F              AND #%01011111      ; To upper....
0A6F  C945              CMP #'E'            ; 'E' or 'e' ?
0A71  D044              BNE ATOF7           ; No exponential part
0A73  C8                INY                 ; Next index
0A74  B11E              LDA (FINPUT),Y      ; Get next character
0A76            ;
0A76  C92B              CMP #'+'            ; Optional sign of the exponent
0A78  F00A              BEQ ATOF8
0A7A  C92D              CMP #'-'
0A7C  D009              BNE ATOF9
0A7E  A902              LDA #ENMASK         ; Set negative exponent bit
0A80  0524              ORA CFLAGS
0A82  8524              STA CFLAGS
0A84  C8        ATOF8   INY                 ; Next index
```

```
6502 Floating Point Package     PROTON 650X ASSEMBLER  V4.4    PAGE: 0017

0A85  B11E                LDA (FINPUT),Y
0A87              ;
0A87  C930      ATOF9     CMP #'0'          ; This has to be a digit
0A89  900C                BCC ATOF42        ; Not A Number
0A8B  C93A                CMP #':'
0A8D  B008                BCS ATOF42
0A8F  38                  SEC               ; Subtract '0' to get the value of
0A90  E930                SBC #'0'          ;                  the digit
0A92  850D                STA WORK          ; Save result
0A94  4C9A0A              JMP ATOF12
0A97  4C280B    ATOF42    JMP ATOFNA
0A9A              ;
0A9A  C8        ATOF12    INY               ; Next digit
0A9B  B11E                LDA (FINPUT),Y
0A9D  C930                CMP #'0'          ; Is it a digit ?
0A9F  9016                BCC ATOF7         ; No, end of conversion
0AA1  C93A                CMP #':'
0AA3  B012                BCS ATOF7
0AA5  A50D                LDA WORK          ; Multiply saved exponent by 10
0AA7  18                  CLC
0AA8  0A                  ASL A
0AA9  0A                  ASL A
0AAA  650D                ADC WORK
0AAC  0A                  ASL A
0AAD  711E                ADC (FINPUT),Y    ; Add digit
0AAF  38                  SEC               ; Subtract '0'
0AB0  E930                SBC #'0'
0AB2  850D                STA WORK
0AB4              ;
0AB4  4C9A0A              JMP ATOF12
0AB7              ;
0AB7  A524      ATOF7     LDA CFLAGS        ; Negative exponent?
0AB9  2902                AND #ENMASK       ; Select flag
0ABB  D00A                BNE ATOF14
0ABD              ;
0ABD              ; Positve exponent
0ABD              ;
0ABD  18                  CLC
0ABE  A520                LDA EXP10         ; Digits behind the decimal point
0AC0  650D                ADC WORK
0AC2  8520                STA EXP10
0AC4  4CCE0A              JMP ATOF15
0AC7              ;
0AC7  38        ATOF14    SEC
0AC8  A520                LDA EXP10         ; Digits behind the decimal point
0ACA  E50D                SBC WORK
0ACC  8520                STA EXP10
0ACE              ;
0ACE  A003      ATOF15    LDY #3            ; Copy mantissa
0AD0  B91500    ATOF18    LDA SAVE,Y
0AD3  48                  PHA
0AD4  88                  DEY
0AD5  10F9                BPL ATOF18
0AD7  20F505              JSR FLTOF         ; Convert mantissa to float
0ADA  A524                LDA CFLAGS        ; Test for negative number
0ADC  2901                AND #NMASK
0ADE  F009                BEQ ATOF60
0AE0  BA                  TSX               ; Negative, get stack pointer
0AE1  A980                LDA #%10000000    ; Sign bit
0AE3  5D0401              EOR $0100+4,X     ; MS-byte
```

```
6502 Floating Point Package     PROTON 650X ASSEMBLER  V4.4    PAGE: 0018

0AE6  9D0401          STA $0100+4,X
0AE9  A900   ATOF60   LDA #0              ; Positive exponent, push 4 byte
0AEB  48              PHA                 ; Push 4 byte        pos. integer
0AEC  48              PHA
0AED  48              PHA
0AEE  A520            LDA EXP10
0AF0  100F            BPL ATOF17          ; Negative exponent,
0AF2  49FF            EOR #$FF            ; 2's complement
0AF4  18              CLC
0AF5  6901            ADC #1
0AF7  48              PHA                 ; Push complemented LS byte
0AF8  201E07          JSR F10PX           ; Calculate FP-number for complemented
0AFB  20B704          JSR FDIV            ; Divide mantissa by exponent     exp.
0AFE  4C080B          JMP ATOF19
0B01             ;
0B01  48     ATOF17   PHA                 ; Push LS byte
0B02  201E07          JSR F10PX           ; Calculate FP-number for exponent
0B05  205103          JSR FMUL            ; Multiply mantissa and exponent
0B08             ;
0B08             ; Reorganise the stack
0B08             ;
0B08  BA     ATOF19   TSX                 ; Get stack pointer
0B09  BD0501          LDA $0100+5,X       ; Get return addres
0B0C  8500            STA RETADD
0B0E  BD0601          LDA $0100+6,X
0B11  8501            STA RETADD+1
0B13             ;
0B13  A003            LDY #3              ; Move 4 byte
0B15  BD0401 ATOF30   LDA $0100+4,X
0B18  9D0601          STA $0100+6,X
0B1B  CA              DEX
0B1C  88              DEY
0B1D  10F6            BPL ATOF30
0B1F  68              PLA                 ; Remove two bytes on top
0B20  68              PLA
0B21             ;
0B21  A501            LDA RETADD+1        ; Restore return address
0B23  48              PHA
0B24  A500            LDA RETADD
0B26  48              PHA
0B27  60              RTS
0B28             ;
0B28  68     ATOFNA   PLA                 ; Get return address
0B29  8500            STA RETADD          ; Save it
0B2B  68              PLA
0B2C  8501            STA RETADD+1
0B2E  A003            LDY #3              ; Transfer 4 byte
0B30  B91A07 ATOFN1   LDA CNAN,Y          ; Get byte
0B33  48              PHA                 ; Store it
0B34  88              DEY
0B35  10F9            BPL ATOFN1
0B37  A501            LDA RETADD+1        ; Restore return address
0B39  48              PHA
0B3A  A500            LDA RETADD
0B3C  48              PHA
0B3D  60              RTS
0B3E                  .END FLOAT1
```

Windows, een verzameling subroutines door Ernst Elderenbosch.

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

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

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

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

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

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

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

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

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

Veel succes - Ernst Elderenbosch.

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

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

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

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

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

Errors detected: 0
```

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

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

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

```
text5   fcc     '  c      ===          A N W B',$d
        fcc     '   /-----/\ ',$d
        fcc     '  / \ o /--\      26.3  Haarlem',$d
        fcc     ' (o)  =   (o)     30.7  Amsterdam',$d
        fcc     '                  29.5  Delft',0
;
text6   fcc     $d,$d,$d
        fcc     '    GGG      OOO      OOO     DDDD     BBBB     Y   Y    EEEEE',$d
        fcc     '   G        O   O    O   O    D   D    B   B     Y Y     E',$d
        fcc     '   G  GG    O   O    O   O    D   D    BBBB       Y      EEEE',$d
        fcc     '   G   G    O   O    O   O    D   D    B   B      Y      E',$d
        fcc     '    GGGG     OOO      OOO     DDDD     BBBB       Y      EEEEE'
        fcc     $d,$d,$d,$d,$d,'                                   Ernst Elderenbosch'
        fcc     0
;
        end     start
```

TALEN/SOFTWARE

Zeef van Eratosthenes.

```
/*
 * Dit is een van de vele zogenaamde
 * Bechmark-programma's; speciaal
 * bedooeld om de snelheid van de
 * processor, in combinatie met het
 * geheugen te bepalen.
 *
 * Dit programma berekent priemge-
 * tallen (dat zijn getallen die
 * alleen deelbaar zijn door 1 en
 * door zichzelf) volgens de zeef
 * van Eratosthenes.
 *
 * Het rekenschema is als volgt:
 * 1) Doe alle oneven getallen
 *    tussen 3 en n in een zeef.
 * 2) Haal het kleinste getal
 *    uit de zeef; dit is een
 *    priemgetal.
 * 3) Haal alle veelvouden van
 *    het getal uit 2 uit de
 *    zeef.
 * 4) Herhaal stap 2 t/m 4 totdat
 *    de zeef leeg is.
 */
```

```
#include <stdio.h>

#define size 8190

main()
{
char flags[size];
int i, prime, k, count, iter;

printf("\nStart iterations");
for(iter = 1; iter <= 10; iter++){
    count = 0;
    for(i = 0; i <= size; i++)
        flags[i] = (char) 1;
    for(i = 0; i <= size; i++){
        if (flags[i]){
            prime = i + i + 3;
            k = i + prime;
            while (k <= size){
                flags[k] = (char) 0;
                k += prime;
                }
            count++;
            }
        }
    printf("\n%d primes",count);
    }
}
```