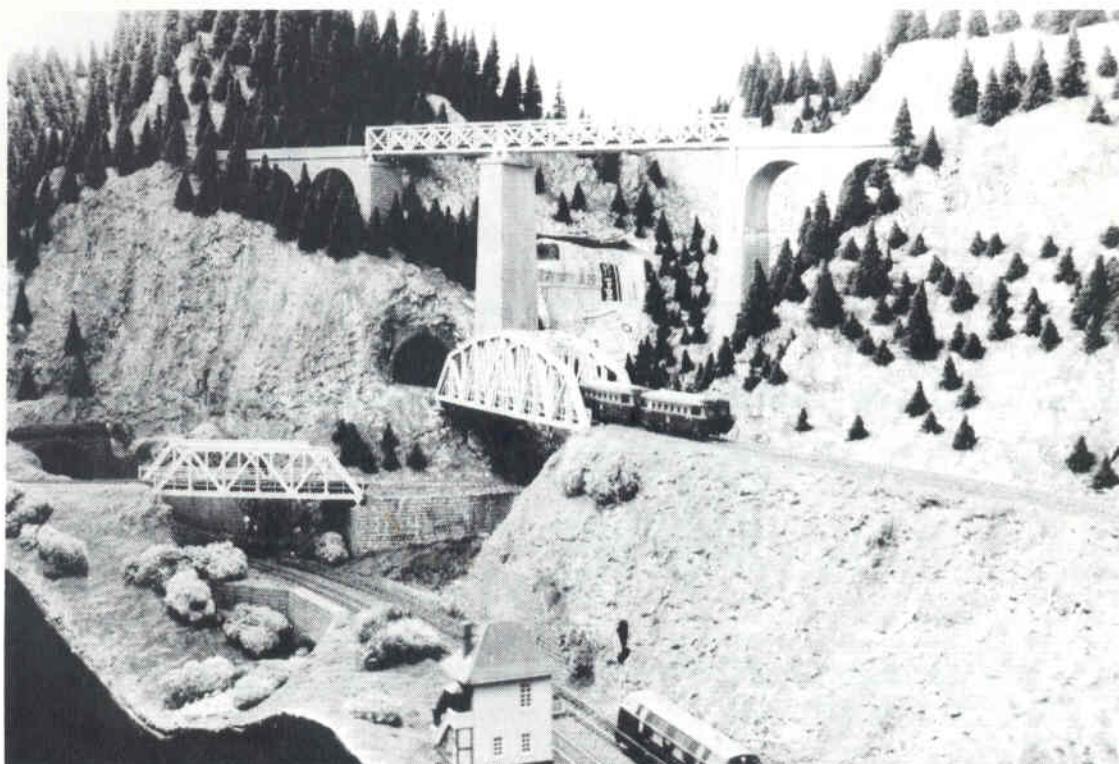


ISSN 0169-9318

PCP

(voor P2000, MSX, PC- en modemgebruikers)

28



Stichting Gebruikersgroep P. Computers

een

uitgave

COLOFON

is het officiële contactorgaan van:
de Stichting GebruikersGroep P Computers i.o.

Uitgever

Stichting GebruikersGroep P. Computers i.o.

Redactie adres Postbus 7268.

2701 AG Zoetermeer

Database TRON-VIEWTEKST

079 - 310.166

(24 uur/dag, 7 dagen/week)

Vidibusnummer 400014759

Hoofdredacteur Albert C. Veldhuis (079 - 316.915)

Hoofdredacteur a.i. Jeroen Wortelboer (079 - 311.864)

Eindredacteur Jo C. Garnier

Lay-out Rob van der Hulst

MSX-zaken Frank van Netten

PC-zaken Paul-Ivo Burgers

Algemene zaken Jannie Aalderink-Bosveld

Druk D.S.W.

Medewerkers aan dit nummer :

Dick Bruggemans, Ad van Eenbergen, Jeroen Hoppenbrouwers, Guido Klemans, Mark Kramer, Michael Nieuwesteeg, Leo Reeuwijk, Barry Sonberg, Robert Vroegop en Karin van Zanten.

Advertentietarief : Op aanvraag**Copyright:**

De inhoud van dit blad mag niet gereproduceerd worden in welke vorm dan ook, zonder voorafgaande schriftelijke toestemming van de uitgever. De aansprakelijkheid uit hoofde van auteursrechten van ingezonden kopij ligt bij de inzender.

Abonnementen:

Deelnemers van de GGPC krijgen het blad gratis toegestuurd.

Losse nummers : f5.85.

De Stichting GebruikersGroep P Computers stelt zich ten doel het gebruik van Philips-computers in de ruimste zin te bevorderen.

Deelname aan de Stichting wordt aangegaan voor tenminste 1 (één) kalenderjaar en geldt tot schriftelijke wederopzegging.

Het deelnemerschap bedraagt f45.-- per jaar, voor individuele personen, bedrijven en instellingen met gratis toezending van 1 (één) nummer van TRON.

Alleen een abonnement op TRON kost f30.-- per 6 (zes) nummers.

Opgave voor het deelnemerschap dienen gericht te worden aan het secretariaat van de GGPC:

Wielingenplein 17
3522 PC Utrecht
Telefoon: 030 - 88.10.87

Betaalwijze:

Binnen 14 dagen na aanmelding, of direct, op Gironummer 240.800 t.n.v.:

Penningmeester Stichting GPC te Utrecht.

Adreswijzigingen

schriftelijk 6 weken van te voren opgeven aan het secretariaat.

Redactioneel**Hoera, te laat!**

Velen van onze lezers houden er inmiddels al rekening mee dat de TRON zo omstreeks de 5de van elke even maand bij hen in de bus moet rollen. Dat bleek overduidelijk toen TRON 27 zo'n dag of 10 te laat was! Van vele kanten werd ons in die tijd gevraagd, wanneer hij nu eindelijk eens zou verschijnen! Wij begrepen hieruit dat u uw TRON interessant vindt (de heer Beerten uit Rotterdam sprak zelfs van "...goede informatie over databanken in TRON 27. In nog geen half uur de letters van papier gegeten (dat was smullen!!)"). En dit is voor ons reden om er steeds weer voor te zorgen dat u dit van ELK nummer blijft zeggen!

Invloed van de tropische temperaturen.

Op het moment dat ik dit schrijf, hebben wij al geruime tijd last van de zomerse temperaturen. Ik bedoel met "wij", al diegenen waarvoor het op enigerlei wijze bezig zijn met het computergebeuren, hobby nummer 1 is. Want menigeen laat zijn computertje toch wel even onaangeroerd als hij in dit kikkerlandje eens de kans krijgt om gulzig van de overvloedige zonnestralen te snoepen! Alle activiteiten worden op een zeer laag pitje gezet, hetgeen ook blijkt uit de kijkdichtheidslijfers van de databanken. En ik vrees dat ook de regiobijeenkomsten minder bezocht zullen zijn, hoewel ik hoop dat de verslagen hiervan op de volgende bladzijden dit logenstraffen.

Maar straks, in het nieuwe seizoen zullen ongetwijfeld de activiteiten weer losbarsten, als iedereen weer vol enthousiasme naar de eerste regiobijeenkomsten in september gaat. Neemt u daar eens een vrien(in) of relatie mee naartoe. U bewijst er hem/haar en uw club wellicht een dienst mee!

ap veldhuis

CORRECTIE

In TRON 27 zijn zodanig storende foutjes geslopen dat 2 artikelen nauwelijks leesbaar zijn, tenzij puzzelen uw tweede grote passie is.

Van het artikel TELESHOPPING zijn de laatste 4 regels van alle kolommen op pagina 12 op bladzijde 13 terechtgekomen. Als u dit blok onder aan pagina 12 "denkt" dan blijkt ineens dat er niets aan het artikel ontbreekt.

Zo ook voor het artikel TREIN-HOBBY-CLUB-ALMELO, op pagina 14. Als u de eerste 9 regels van bladzijde 15 onder aan pagina 14 plakt, wordt alles veel helderder!

Onze excuses voor het ongerief. (Boze tongen spraken reeds van "het groot vakantie knipselblad"!) ap veldhuis

**Uiterste datum voor kopij-ingezending
voor TRON 29 is 26 augustus 1989**

INHOUD

- 5 Liftdemo met P200T
- 6 Teletekst letter, hoe komen we eraan
- 7 Trein-Hobby-Club-Almelo, aflevering 5, software
- 9 Ziekenhuisscholen voorzien van MSX-computers
- 10 Werken met de 8086
- 13 Datacompressie
- 15 Kanttekeningen bij de PC, deel 7
- 15 Lotus 1-2-3, cursus, deel IV
- 16 Gegevensopslag op de HARDE SCHIJF, deel 6
- 17 ... en er was beeld!
- 18 Abstractie: de kern van de zaak
- 21 Karin's column
- 22 Hoe bouw ik een IC
- 23 Boekbesprekingen

LIFT-DEMO.

Voor de Evenementendag in Zoetermeer werd mij gevraagd om iets te maken, dat met een computer kan worden bestuurd.

Daarop heb ik een lift-constructie gebouwd en aangesloten op een P2000-microcomputer. Hoewel het geheel op de Evenementendag nog niet geheel bouwrijp was (er verbrandde een motortje en de voeding begon te roken), kon ik toch enigermate laten zien hoe zo iets werkt. Er was voornamelijk belangstelling van modelbouwers.

Hierna volgt een korte beschrijving van het geheel.

1. De computer. Deze is voorzien van een insteekdoos in slot 2. In deze doos zit een schakeling voor het herkennen van poort-adres &H6e en &H6f. Met beide adressen is input mogelijk, alleen met &H6f is ook output mogelijk. Voor in- en output worden 3 IC's gebruikt van het type 8212. In deze IC's zitten ieder 8 geheugenplaatsen, zodat een output beschikbaar blijft totdat een nieuwe output wordt gegeven. Met BASIC werkt dit als volgt:

```
AE=INP(&H6E) : BE=AE : BE=(BE AND &H80) : IF BE THEN ....  
(dit test het hoogste bitje van inputpoort &H6e).
```

```
IF X7 THEN X=(X OR &H80) ELSE  
X=(X XOR &H80)... enzovoort  
voor x6 t/m x0 met resp. &H40 - 20 - 10 - 08 - 04 - 02 en 01  
, dan OUT &H6F,x  
(zo kunnen alle output-bitjes worden bestuurd met x7 t/m x0).
```

2. Alle input-poorten moeten worden gebufferd om de insteekdoos te beschermen tegen de 12 volt, die in de lift wordt gebruikt.
Ook de output-poorten worden gebufferd, omdat hiermee relais en lampjes worden gestuurd.

3. De liftconstructie. Hierin komt uiteraard een liftmotor voor, maar ook voor alle 4 etages een deurmotor. Op elke etage is een schakelaar om te herkennen of de lift recht achter de deur staat (lift in positie).
Op de begane grond wordt hiervan een extra signaal afgeleid (lift op BG). Als de lift niet op BG is, houdt het programma de positie bij. Dit wordt niet apart getest. Omdat het programma werd ontwikkeld met het idee voor 1 deurmotor i.p.v. 4x, wordt door de étageschakelaars ook de juiste deurmotor gekozen. De computer stuurt steeds "dezelfde" deurmotor. Ook voor elke deur zijn 2 schakelaars nodig, om de standen - geheel open - en - geheel dicht - te herkennen.

Dit levert de 2 signalen -deur open- en -deur dicht-.
Op elke étage wordt de laatste positie van de lift aangegeven met 1 uit 4 lampjes. Bovendien kan men (in de lift) kiezen uit 4 etages en kan men (op een



etage) kiezen uit omhoog of omlaag. Uiteraard op BG alleen omhoog en op de hoogste etage alleen omlaag. Ook is er een alarmtoets aangebracht.

4. Het programma. Dit is het enige "intelligente" deel van het hele knutselwerk. Hierin zijn de volgende hoofdstukken aanwezig:

A) BEGIN

In dit deel worden de lampjes (het display) uitgezet. Daarna wordt getest of de deur dicht is; zo niet, dan wordt de deur gesloten.

Daarna wordt getest of de lift op BG staat; zo niet, dan wordt de liftmotor aangezet -omlaag-. Zodra de lift op BG staat begint het hoofdprogramma.

B) HOOFDPROGRAMMA

Dit is een rondloper, d.w.z. nadat het programma is uitgevoerd wordt opnieuw begonnen. Hier wordt beslist, naar welke etage de lift gaat.

Als de lift bijv. van 1e etage naar 3e gaat, wordt zonodig op 2e etage voor instappers gestopt. Daarbij wordt rekening gehouden met de richting waarin de lift gaat. De deuren worden op het juiste moment geopend en gesloten en de aanduiding op het display wordt steeds bijgewerkt. Na elke output wordt naar alle input gekeken. Als de alarmtoets wordt ingedrukt, maakt de lift de laatste beweging af. Daarna gaat de betreffende etagedeur open en wordt de lift geblokkeerd. Op het display gaan dan ALLE lampjes knipperen.

Voor al deze acties wordt steeds van dezelfde subroutines gebruik gemaakt, nl.

C) INPUT

Hierin wordt de stand van een aantal "vlaggen" bijgewerkt, zodat wordt onthouden dat een bepaalde toets werd ingedrukt, of een signaal van de lift veranderde.

D) OUTPUT

Hierin worden alle bitjes verzameld en als 1 byte naar de lift gestuurd.

E) VERPLAATS LIFT

Ook dit is een voorbereiding voor output.

F) TEST VOLGENDE ETAGE

Hierin wordt getest of na een opdracht "verplaats lift" de étageschakelaar is uitgezet, d.w.z. de lift is onderweg.

G) ETAGE BIJWERKEN

Omdat niet kan worden getest waar de lift is (behalve wanneer de lift op BG is), moet het programma dit bijhouden.

H) DEUR OPEN

Hierin wordt de deur (op de juiste étage) geopend.

Er wordt steeds getest of de deurmotor kan worden uitgeschakeld.

I) DEUR DICHT

Omgekeerd aan "DEUR OPEN". Echter, de deur gaat vanzelf na enige tijd dicht, d.w.z. zonder opdracht uit het hoofdprogramma.

J) BRAND

Als het hoofdprogramma ziet dat op alarmtoets is gedrukt, wordt door deze routine de beweging van de lift voortgezet tot de eerstvolgende etage. Dan gaat de deur open en gaat deze routine rondlopen, d.w.z. er kan niets meer worden bediend.

K) KNIPPEREN

Dit wordt ingeschakeld door de routine "BRAND". Het zet regelmatig alle lampjes aan en na eenzelfde tijd weer uit.

Uit deze (summiere) beschrijving blijkt wel dat een lift een ingewikkelde ding is.

Ik had ook geen idee waaraan ik begon.

Zoals gezegd, door mechanische problemen kwam dit op de Evenementendag niet uit de verf.

De gehele werkwijze is, vooral voor modelbouwers van spoorbanen, interessant genoeg om ook een poging in deze richting te doen.

(En ik heb ook nog nooit een modelboot met computerbesturing gezien)

Veel plezier met de hobby

Leo Reeuwijk
Den Haag
070-257372

Teletekst-letters: hoe komen we eraan?

Toen de P2000 rond 1980 op de aardbol verscheen, had hij weinig concurrentie te duchten. Van PC's had niemand in Nederland nog ooit gehoord en de C-64 en Spectrum-computers waren toch op een andere markt gericht. Dat lag niet zozeer aan de mogelijkheden van de processor of het geheugen, maar vooral aan hun geluids- en beeldcapaciteiten. Vooral de grafische beeldschermen van deze andere machines trokken een groot (spelend) publiek. De P2000 had, wat dat betreft, niets aan boord. Voor spelletjes is dit apparaat dan ook nauwelijks ingezet.

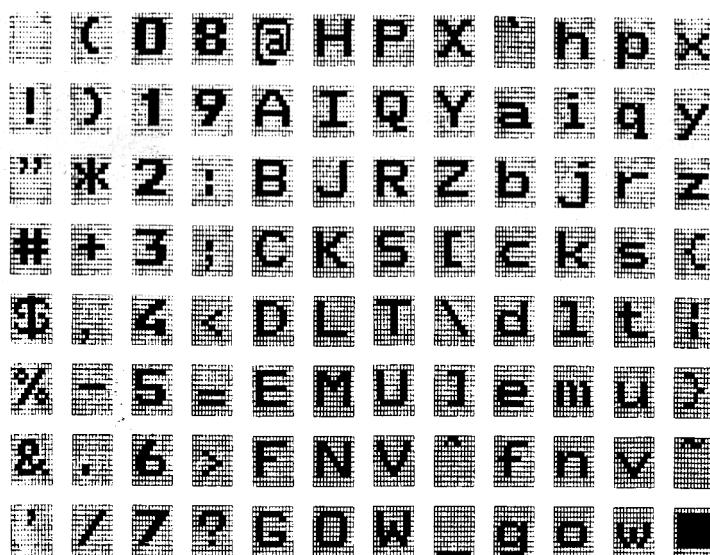
Waarom hebben de P2000-ontwerpers dan hun troetelkind niet ook met zo'n grafische chip uitgerust? Waarom kozen ze voor die simpele niet-grafische Teletekst-chip?

Het antwoord is meteen duidelijk, wanneer je een P2000 naast een C-64, Spectrum of MSX in de TEKST-stand van deze machines, zet. Zelfs met een goede (kleuren)monitor eraan, valt het verschil al meteen op: iedereen zal zeggen dat het P2000-scherm op de één of andere manier rustiger, vriendelijker overkomt. Op demonstraties in den lande is dat ook altijd het tweede wat ik hoor. Eerst komt de opmerking: "Hé, ik wist niet dat die ouwe peetweduizend ook kon ... (vul maar in: muis, floppies, grafisch, MIDI, CP/M etc.)" en daarna werpen ze een blik op het beeldscherm en vragen zich af: "Waarom is dit eigenlijk zo mooi, zo vriendelijk?". Vaak hoor ik ook opmerkingen in de trant van: "Ik stond een eindje verderop naar het P2000-scherm sprong daar meteen uit".

Het wordt nog veel erger, wanneer je niet de directe monitor-aansluiting en een echte monitor gebruikt, maar de gemoduleerde uitgang en een gewone TV. Op alle genoemde kleine computers past ook een TV en in zeker de helft van de gevallen zit (zat) er ook een TV op. In zo'n geval vliegt de beeldkwaliteit achteruit. 80 kolommen haal je eigenlijk niet meer fatsoenlijk (maar geen enkele van deze computers kon standaard 80 kolommen aan, alleen de P2000 na een kleine ombouw). Plaatste je een paar TV-beelden van de machines naast elkaar, dan werd het verschil tussen de P2000 en de rest pas echt duidelijk. Het teletekst-beeld bleef leesbaar, de andere schermen veranderden in een brij van onleesbare letters.

Geschiedenis

De historie van de P2000-chip gaat terug tot 1969. In dat jaar startte op het Instituut voor Perceptie Onderzoek (IPO) in Eindhoven - een samenwerkingsverband tussen de Technische Universiteit en Philips



- een onderzoek naar de leesbaarheid van matrix characters. Dat zijn de bekende - uit puntjes opgebouwde - tekens op goedkope beeldschermen en goedkope printers. Op dure uitvoeringen zijn die puntjes er ook nog wel, maar daar zijn ze zo klein dat ze in elkaar overvloeien en daar gaat het hier niet om.

Teletekst, een techniek om op een gewone kleurentelevisie teksten te produceren via een paar tot dan toe ongebruikte beeldlijnen in het gewone TV-signaal, is een mooi voorbeeld van matrix letters op een goedkope beeldbuis. Een normale TV-buis heeft niet genoeg oplossend vermogen om letters echt scherp weer te geven. Zolang de beelden bewegen of niet zo fijn zijn gedetailleerd, gaat het nog wel. Maar, een heel scherm tekst bleek erg slecht leesbaar. Voor een deel was daar niets aan te doen: je kon niet even alle TV's verbouwen. Maar de gebruikte letters en cijfers konden misschien nog wat worden verbeterd. Tot dan toe werden tekens eigenlijk uit de losse pols, door de ontwerpers van de video-chips, in elkaar gezet. Dat waren wel deskundigen op het gebied van chips bakken, maar niet op het ergonomische vlak. Op het IPO werk(t)en wél ergonomen en bovendien waren er technici genoeg in de buurt. Er werd besloten om een zo goed mogelijke character set te ontwikkelen, die nog binnen de technische grenzen paste.

Tot dan toe werden voor Teletekst letters gebruikt, die uit een matrix van zes bij tien of acht bij acht puntjes waren opgebouwd. Dit was eigenlijk een willekeurige keuze: acht bij acht is computertechnisch lekker makkelijk en zes bij tien lijkt wat meer op een letter-rechthoekje. Binnen deze beperkte matrix is het niet goed mogelijk, duidelijke leesbare letters te maken. Toch gebruiken bijna alle oudere en een groot deel van de nieuwere computers

nog steeds letters van acht bij acht puntjes. Het resultaat is er dan ook naar (IBM-PC: de CGA-kaart!).

De mensen van het IPO sloegen aan het rekenen en bleken op een normale TV letters kwijt te kunnen van 12 bij 10 puntjes. Daarmee konden ze dan 24 regels van ieder 40 kolommen vullen. Nu we gewend zijn aan letters van 9 bij 14 puntjes op 80 kolommen (IBM-PC: monochroom/Hercules/EGA), lijkt dat niet veel, maar voor die tijd was het revolutionair. Twee maal zoveel punten voor dezelfde letter! Volgens de oude manier zouden de chips-bakkers nu in de koffiepauze even op een ruitjesvel de benodigde letters intekenen, maar op het IPO waren ze er al lang achter dat wat onderzoek vooraf wonderen kon doen. In plaats van een tamelijk willekeurige letterverzameling op te stellen, gingen ze als volgt te werk.

De relatief grote matrix bood de mogelijkheid om voor elke kleine letter, vier verschillende ontwerpen te maken. Dat leverde in eerste instantie dus $26 * 4 = 104$ verschillende letters op. Deze 104 letters werden één voor één op een normale TV vertoond. Twaalf proefpersonen kregen al deze letters door elkaar te zien en moesten telkens zeggen welke letter het was. Maar... vanaf een afstand van acht (!) meter. Waarom van zo overdreven ver weg?

Het ging bij dit experiment om erg kleine verschillen in leesbaarheid. Op de normale Teletekst-kijkafstand (van 1.6 meter in het ideale geval) zouden die verschillen tussen de vier letterontwerpen niet duidelijk genoeg naar voren komen. Daarom werd voor iedere proefpersoon de TV zo ver weggezet dat hij of zij nog maar 50% van alle letters juist herkende. Op die manier krijgt een iets betere letter een score van zeg 55 tot 60% en een iets slechter ontwerp een score van 40 tot 45%. Dat zijn grotere verschillen waarmee je kunt

werken.

De verkregen resultaten werden voor alle proefpersonen gemiddeld en toen werd het beste ontwerp van iedere letter gekozen, door het ontwerp te nemen dat het minste met andere letters was verward. De slechtste "a" bijvoorbeeld werd van 36 keer 17 keer als "n" of "o" herkend en maar 4 keer als "a"! De beste versie van de "a" haalde een score van 18 als "a", 6 als "n" en 0 als "o". Een heel stuk beter.

Op dezelfde manier werden later ook de hoofdletters, cijfers en speciale tekens onderzocht. Het resultaat was een tekenverzameling van 196 optimaal leesbare symbolen, die bekend werd als "IPO-Normaal". Er werd ook een vergelijkbare set gemaakt met wat vettere letters, "IPO-Bold". Uit vergelijkende onderzoeken met de bestaande Teletekst-letters bleek IPO-Normaal duidelijk de beste te zijn, op de voet gevolgd door IPO-Bold.

Als bijzonderheidje haal ik nog even het verschil aan tussen de letters en cijfers van IPO-Normaal. Om het onderscheid tussen letters en cijfers

te bevorderen, werd besloten de cijfers in vette uitvoering in IPO-Normaal te zetten. Het resultaat is een redelijk duidelijk onderscheid tussen bijvoorbeeld 0 (letter) en 0 (cijfer), S en 5 en B en 8.

In de internationale wereld werd de superioriteit van IPO-Normaal snel onderkend en tot standaard verheven. Alle Teletekst-chips werden voortaan uitgebracht met de IPO-letters ingebakken.

Toen de Philips-ingenieurs met de P2000 aan het knutselen waren, zagen ze zich voor de moeilijkheid geplaatst om op een normale TV (goedkoop) een goed letterbeeld te zetten. De P2000 werd tenslotte speciaal als tekstverwerker ontworpen en dan mag je op zijn minst een leesbaar beeldscherm hebben, nietwaar? Na wat gerommel op de markt, is toen voor een standaard Teletekst-chip gekozen en het resultaat is bekend.

Het onbedoelde gevolg was natuurlijk ook dat de P2000 gewoon ideaal bleek voor het gebruik als VidiTel-terminal! VidiTel heeft namelijk ook zijn fundamenteen in de gewone televisie

en gebruikt dus eveneens de Teletekst-set. Er zat op de P2000 ook nog een gewone RS-232-connector (uiteindelijk in die dagen) en het succes hing dus alleen nog maar af van een programma. Dat werd door ir. Klaas Robers in zijn vrije tijd op een blocnote uitgepuzzeld (er was toen nog geen assembler!) en het resultaat is bekend. Met de P2000 voorop begon het gebruik van computers als VidiTel-terminal. Telesoftware en prikborden zijn eveneens door VidiTel P2000-pioniers uitgevonden. BASICODE en CD-ROM ook, maar dat is een ander verhaal.

Tot zover de oorsprong van de vreemde P2000-letterverzameling. Leuk voor open dagen waarop er weer gekankerd wordt op die ouwe bak zonder geluid en grafiese mogelijkheden...

Bibliografie

Floris L. van Nes, A New Teletext Character Set with Enhanced Legibility, IEEE Transactions on Electron Devices, Vol. Ed-33, No. 8, august 1986.

Jeroen Hoppenbrouwers

TREIN-HOBBY-CLUB-ALMELO

Aflevering 5: Software bij de THCA

Na de vorige keer het principe van de software te hebben behandeld, nu over naar de praktijk. (zie ook het flowchart in de vorige aflevering).

Programmalussen:

Het programma springt telkens langs die inputkaarten, waar trafoknopen op zijn aangesloten. Via de subroutines voor het inlezen van waarden van die kaarten, wordt gekeken of er actie wordt gevraagd. Elke paar lusjes van het hoofdprogramma wordt er 1 outputpoort van alle kaarten aangestuurd via een aparte subroutine. Wij doen dit per poort, zodat niet alles tegelijk gebeurt, wat vooral bij wissels van belang is. In het groot gaan de wissels van een rijweg ook een voor een.

Eerst de inleesroutine. Vanuit het programma wordt het kaartnummer opgegeven en x geeft de waarde van die kaart terug.

```
5  out &H61,K : x=INP(&H60) : out
   &H61,0 : return
```

Dan de output routine. Het programma geeft de waarde (ou(kk)) en het kaartnummer ko(kk) op. Ko() is de array met hex kaartnummers.

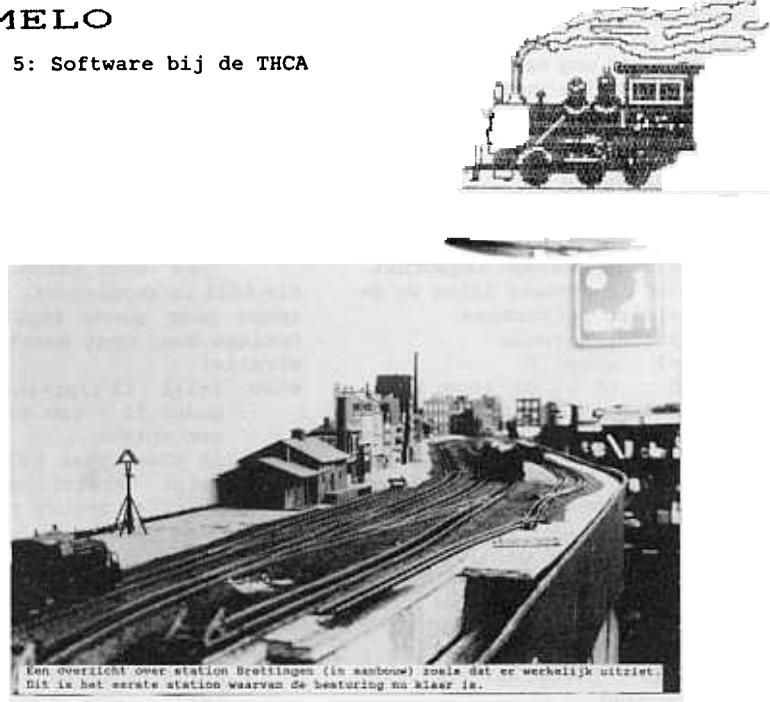
```
10 out &H60(kk) : out &H61,ko(kk)
    : out &H61,0 : return
```

Deze routine wordt zowel rechtstreeks vanuit het programma gebruikt, als wel ook via de nu volgende 'outputlus', die telkens na een bepaalde periode wordt doorlopen.

```
11 pp=pp+1 : if pp>8 then pp=1
12 for kk=1 to 5
13 if p(kk,pp)>1 then
   p(kk,pp)=p(kk,pp)-1
14 if p(kk,pp)=1 then 16
15 ou(kk)=ou(kk) and (not w(pp)) :
   goto 17 : rem uitzetten
16 ou(kk)=ou(kk) or w(pp) : rem
   aanzetten
17 gosub 10 : next kk : return
```

Een klein beetje uitleg is op zijn plaats.
pp is de variabele die van 1 tot en met 8 loopt en daardoor de acht poorten per kaart kiest.

kk geeft het kaartnummer aan, via ko(kk) de hex code van de kaart. p(kk,pp) is de waarde per poort. Is p() gelijk aan nul, dan wordt de poort uitgeschakeld, als p()=1 dan wordt de poort aan geschakeld. w(pp) is een array van acht waarden: 1,2,4,8,16,32,64,128. Deze waarden komen overeen met de waarden die nodig zijn, om de afzonderlijke poorten te sturen. Heeft een output de waarde 8, dan ziet dit er in bits zo uit: 00001000. (de kleinste waarde staat rechts)



Een overzicht over station Breddingen (in aanbouw) zoals dat er werkelijk uitziet. Dit is het eerste station waarvan de besturing nu klaar is.

ou(kk) is de totale waarde voor de hele kaart in het geheugen van de computer. Alle poorten per kaart worden telkens zo ingesteld dat er steeds maar 1 veranderd. Dit gaat d.m.v. de or en and/not routines. Voor 1 bit geldt:

x	y	xory	xandy	not y	xand(noty)
1	1	1	0	0	0
1	0	0	1	1	1
0	1	1	0	0	0
0	0	0	0	1	0

Hierbij is x te vergelijken met 1 bit van ou(kk) en y met w(pp). Voor meerdere bits, dus een hele waarde:

x=00001000 y=00000001
x=x or y dus 00001000 or 00000001 wordt 00001001 is aanzetten.

Daarna: x=x and (not y) dus 00001001 and 11111110 (not y) wordt 00001000. Dit is dus het uitzetten van bit 1 (laatste bit van de rij).

Regel 13 heeft een extra faciliteit. Als p() groter wordt gemaakt dan 1, bv. 3, dan wordt de betreffende poort pas aangezet, nadat delus 2 keer is doorlopen. Dit gebruiken wij bij het schakelen van seinen, die hierdoor als laatste op veilig gaan staan. Op deze manier weten we zeker dat een machinist niet mag vertrekken, voordat alle wissels en railaansluitingen goed staan.

Later hebben wij nog een paar trucjes in onze outputroutine ingebracht, maar voorlopig voert het te ver om dit hier te vermelden.

Het hoofdprogramma.

Hierin kijkt de computer of er traafoknopen of knopen van de verkeersleiding worden ingedrukt. Terwille van de eenvoud laten we de verkeersleiding achterwege.

1000 rem hoofdprogramma

1010 k=&H02 : gosub 5

1020 i=i+1 : if i > pz then i=0 :

gosub 11 : rem outputlus

i is de teller om af en toe een output te doen. pz(pauze) is op het moment 10. als i dus groter is dan tien, wordt er van elke kaart 1 poort bestuurd.

1050 if (x and w(3) and vl(1)>0 then
tr=1 : if ry(tr)>0 then 15000
else kk=1 : ou(kk)=ou(kk) or 8
: ou(kk)=(kk) and (not 16) :
gosub 10 : p(1,6)=1 : p(1,7)=0
: goto 2300

Als de verkeersleiding een spoor vrij geeft (vl>0) en traafoknop 1 wordt ingedrukt (kaart &H02 geeft (ook) waarde w(3)), dan wordt bekeken of het een in gebruik zijnde rijweg is (ry(tr)>0). Is de rijweg in gebruik, dan moet hij worden uitgeschakeld, wat bij regel 15000 en verder gebeurt. Is de rijweg 0, dus niet in gebruik, dan worden de ledjes, die op het paneel de trafo vertegenwoordigen, via een directe output van groen naar rood omgeschakeld. De p()'s houden deze stand vast, zodat de outputlus niet later de instelling op de oude stand terugbrengt. Dan springen we voor dit station naar 2300.

```

2300  rem tx bezet
2310  i=0 : j=0
2320  gosub 6000 : if i>max then
          10000 else gosub 8000 : rem
          knoppen rijwegen/rijwegen
          uitzoeken

```

Regel 6000 loopt de knoppen van het stationspaneel langs om de rijwegopdracht te ontvangen. Daarbij ingebouwd zijn de tellers i en j, die, als ze boven max komen, het programma onderbreken en via 10000 teruggaan naar het hoofdprogramma. Regel 8000 vertaalt de ingestelde knoppen in een rijwegnummer. Als het nummer niet kan, dan blijft ry(tr) nul.

```

2340  if ry(tr)=0 then print "rijweg
          = 0" : goto 10000
2350  gosub 14000 : if ry(tr)=0 then
          10000 : rem rijweg controleren

```

Regel 14000 bekijkt of de rijweg is toegestaan in verband met andere kruisende rijwegen.

```

2360  gosub 13000 : rem rijweg
          instellen
2370  goto 1000

```

Dit was de hoofdlus. Samen met de subroutines draait met deze regels 1 station, de andere stations gaan we hiertussen schrijven met elk eigen subroutines.

Subroutines.

```

6000  rem inlezen knoppen Brettingen
6010  k = &H01 : gosub 5 : kn(1) =
          kk = &H02 : gosub 5 : kn(2)=x
6012  if kn(2)>2 then 6020 : rem
          kn(2)=3 en hoger zijn vl- en
          traafoknopen, dus niet meereke-
          nen
6015  if kn(1)+kn(2)>0 then 6100 : rem
          'van' knop gevonden

```

Als 6015 is gepasseerd, zijn er deze ronde geen goede inputs geweest. Opnieuw dus, maar eerst de administratie:

```

6020  i=1+1 : if i/pz=int(i/pz) then
          gosub 11 : rem dus af en toe
          een output

```

```

6030  if i>max then 6031 else 6040
6031  print "Brettingen duurt te
          lang!" : return rem foutmel-
          ding

```

```

6040  goto 6000 : rem opnieuw probe-
          ren

```

```

6100  rem tweede serie knoppen Bret-
          ting

```

```

6110  k=&H01 : gosub 5 : kn(3)=x-
          kn(1) : k=&H02 : gosub 5 : kn(3)
          dus moet dit worden
          afgetrokken.

```

```

6120  if kn(3)+kn(4)>0 then 6200 : rem
          'naar' knop gevonden

```

```

6130  j=j+1 : if j/pz=int(j/pz) then
          gosub 11 : output

```

```

6140  if j>max then 6141 else 6150
6141  print "Brettingen duurt te lang
          !" : return

```

```

6150  goto 6110 : rem opnieuw probe-
          ren

```

Tot zover zijn dus de 'van' en 'naar' knop bekend. Maar om misverstanden uit te sluiten, hebben we als voorwaarde voor de bediening gesteld dat de tweede knop moet worden ingedrukt, terwijl we nog steeds de eerste vast houden (twee-handen gebruiken).

Daarom regels 6200 en verder om te kijken, of na de kn(3) en kn(4) de eerste knoppen nog zijn ingedrukt.

```

6200  rem sx nog actief ?
6210  k=&H01 : gosub 5 : j=max+1 :
          return : rem j>max is foutmel-
          ding
6230  k=&H02 : gosub 5 : if
          kn(2)=(x-kn(4))then 6250 else
          print "niet twee knoppen" : j
          = max+1 : return : rem idem
6250  return

```

En als er dus twee knoppen zijn ingedrukt, heeft of kn(1) of kn(2) een waarde voor de rijwegen. Hetzelfde geldt voor kn(3) en kn(4). We komen nu terug in regel 2320, waar wordt gecontroleerd, of we boven de max zitten. Als dat goed is, gaan we naar 8000. We geven van deze routine niet de volledige listing, want dat zou neerkomen op ongeveer 20 keer hetzelfde met alleen maar andere waarden. Dit wordt dus een 'bloemlezing'.

```

8000  rem rijwegen zoeken Brettingen
8020  if kn(1)=w(1) and vl(1)=then
          8200 : rem knop inrijspoor
          hoofdbaai. Hierbij is ook de
          instelling van de vl als voor-
          waarde voor een goede rijweg.
8030  if kn(1)=w(2) and tr=2 then
          8300 : rem knop inrijspoor
          voor rangeren. tr=2 is ran-
          geertrafo.
8070  if kn(1)=w(6) then 8700 : rem
          knop spoor 2.
8120  return

```

Hiermee zijn dus drie 'van' knoppen getest. In 8200, 8300 en 8700 worden ze verder uitgerafeld, via de 'naar' knoppen, tot rijwegnummers. Hier geven we alleen de hierboven genoemde knoppen als combinatiemogelijkheid.

```

8200  rem hoofdbaai naar spoor 2 (en
          meer mogelijkheden in echt)
8220  if kn(3)=w(6) and tr=1 then
          ry(tr)=7 : rem tr=1 is hoofd-
          baantrafo
8300  rem inrijspoor rangeren naar
          spoor 2...
8320  if kn(3)=w(6) then ry(tr)=16
8700  rem spoor 2 naar inrijspoor
          resp. hoofdbaai- en rangeer-
          trafo
8720  if (kn(3)=w(1)*(tr=1)*(vl(1)=2)
          then ry(tr)=7 : rem vl(1)=2
          wil zeggen de verkeersleiding
          heeft een uitrijdende route
          toegestaan.
8730  if kn(3)=w(2) and tr=2 then
          ry(tr)=16
9999  return

```

Zo hebben we drie rijwegen gekregen, geldend voor twee richtingen (in en uit). Volgende keer gaan we door met het controleren of de rijweg is toegestaan en het instellen van de poorten.

Copyright THCA en haar leden.
Correspondentieadres: THCA/Ad van Eenbergen en Barry Somberg, Hofkampstraat 1a, 7607 NA Almelo.

ZIEKENHUISSCHOLEN VOORZIEN VAN MSX-COMPUTERS.

De spannendeuren voor een operatie of de lamelidige weken van langdurig herstel kunnen tegenwoordig door veel meer ziekenhuispatiëntjes worden verlevendigd met spelen(d leren) op de computer. De vijftien ziekenhuis-scholen, die Nederland rijk is, zijn door het Input-informatieproject van de spaarbanken voorzien van computers. Voor elke school twee. De dertig apparaten werden namens het landelijk directeurenoverleg van de ziekenhuis-scholen door de heer J.M.Courlander in ontvangst genomen. Dat gebeurde in het Wilhelminakinderziekenhuis te Utrecht. Drs H. Tulner, hoofd van de afdeling Onderwijsbeleid van Instellingen voor (voortgezet) speciaal onderwijs van het Ministerie van Onderwijs en Wetenschappen, verrichtte de symbolische overhandiging.

Courlander, directeur van de Ziekenhuischool Amsterdam, gaf in zijn dankwoord aan hoe belangrijk computeronderwijs voor kinderen is en met name voor kinderen tijdens een verblijf in het ziekenhuis. Hoe aantrekkelijker de leermiddelen, hoe sneller en beter de interesse van het kind wordt gewekt. "Gebruik van computer in het ziekenhuisonderwijs levert een belangrijke meerwaarde. Leermiddelen zullen een beroep moeten doen op de zelfwerkzaamheid en zelfredzaamheid van de leerling, op het nemen van initiatief. (...) Immers ziek zijn, opgenomen zijn in een ziekenhuis, kan gemakkelijk leiden tot grotere passiviteit, tot een sterk, soms een vrijwel uitsluitend gericht zijn op eigen lichaam en ziekte, tot angst, spanningen en onzekerheden".

Omdat, zo bleek uit Courlanders betoog, het NIVO-project van de overheid, ter stimulering van het computeronderwijs, zich in eerste instantie heeft gericht op het regulier voortgezet onderwijs, dreigden de ziekenhuis-scholen wat het computeronderwijs betreft in een achterstandpositie te geraken.

Ziekenhuis-scholen zijn door de overheid ingedeeld bij het speciaal onderwijs.

"Dat betekende voor veel ziekenhuis-scholen dat wanneer een ziek kind, afkomstig uit het voortgezet onderwijs, vroeg of hij iets aan zijn computerlessen kon doen, het antwoord altijd 'nee' was", aldus de heer W.Last, directeur van de Ziekenhuis-school Limburg.

Directeur J. van Loon van Ziekenhuis-school De Piekerbol in Helmond heeft soortgelijke ervaringen. "Wij hebben een heleboel leerlingen uit het voortgezet onderwijs die gewend zijn met computers om te gaan. Die konden wij helemaal niets bieden. In dit opzicht worden binnen het speciaal onderwijs veel te grote lijnen getrokken".

De woorden van Courlander sluiten daarbij aan. Hij zei: "Teneinde aan een van de doelstellingen van het



ziekenhuisonderwijs te kunnen voldoen, namelijk het voorkomen van onnodige leerachterstanden, werd het sinds geruime tijd noodzakelijk computers te introduceren".

AANTREKKELIJK.

De drie genoemde directeuren toonden zich, evenals de overige aanwezige collega's, zeer verguld met het gebaar van de Stichting Input. Courlander maakte daarbij de aantekening, dat de ziekenhuis-scholen door hun vele locaties (Helmond bijvoorbeeld geeft in 13 ziekenhuizen les) met één computer niet echt geholpen zijn, al voegde hij daar onmiddellijk aan toe, dat het aanbod van twee computers per ziekenhuischool een flinke aanzet betekent voor het computeronderwijs aldaar.

Voor de betrokken leerkrachten is de waarde van computers onbetwist. Gebrek aan motivatie bij de ziekenhuisleerling valt gemakkelijker te doorbreken. Van Loon: "Rekenen met een rekenboek is veel minder aantrekkelijk dan wanneer dezelfde rekenstof via een computer wordt aangeboden. En verder speelt nog een heel praktisch voordeel een rol: Dankzij het inschakelen van een computer heb je als leerkracht je handen vrij om naar andere kinderen toe te gaan".

STANDAARD.

Voor de ziekenhuis-scholen gaat het tij, ook wat betreft de overheid, nu toch keren. Al is het lange tijd nadat de computer zijn intrede heeft gedaan in het voortgezet onderwijs, tussen 1990 en 1994 worden alle scholen voor basis- en speciaal onderwijs uitgerust met computers, zij het op bescheiden schaal.

Drs. Tulner (Ministerie van O & W) verwees daarnaar tijdens de bijeenkomst in Utrecht. Hij onderstreepte dat door de minister aanvankelijk

ten aanzien van het basis- en speciaal onderwijs een terughoudend beleid werd gevoerd. "De waarde van de computer daarin moest eerst worden bewezen", verklaarde hij.

"Voordat de minister overgaat tot bevoorrading van de scholen moet een keus worden gemaakt voor één standaard industrie-micro-computer", aldus Tulner. "Het werken met één standaard brengt minder kosten met zich mee. De verschillende programma's hoeven niet steeds voor de verschillende systemen geschikt te worden gemaakt".

SOFTWARE-FONDS.

De keuze voor een standaard, die wellicht niet overeenkomt met de door Input op zoveel basisscholen verstrekte computer, zou een probleem kunnen vormen voor die scholen. Tijdens de Utrechtse bijeenkomst deelde voorzitter drs. B.Hartelust van de Stichting Input echter mee, dat het bestuur van de stichting in principe toestemming heeft gegeven om de door haar ontwikkelde programmatuur en courseware opnieuw aan te passen. "Dat betekent", zei hij "dat de meeste van de programma's bij de start van het basisscholenproject van de overheid in 1990 beschikbaar zullen zijn voor de nieuwste apparatuur".

"Met het ministerie zal", zei hij verder, "als daar belangstelling voor bestaat, overleg worden gepleegd over het gebruik en de verspreiding. Wij hebben gehoord, dat inmiddels bij de overheid belangstelling bestaat voor de zogenoemde 'autonome ontwikkelingen'. Er is ook een constructie denkbaar waarin de programma's worden overgenomen door de overheid. Input zal de mogelijke opbrengst dan storten in een software-fonds, waaruit gezamenlijke ontwikkelingen kunnen worden bekostigd".

HULPMIDDEL.

Hartelust bracht tijdens de computeraanbieding de start van het Input-project in herinnering. Er zou 'slechts' een aantal boekjes worden uitgeven over moderne technologieën. Dat was in 1983. Maar.... "het onderwerp informatica leidde al snel tot de conclusie dat droogzwemmen op dit gebied niet het gewenste resultaat oplevert". Door de inbreng van Philips op het gebied van kundigheden en ervaring kon een project tot stand komen met als doel: het basisonderwijs in staat stellen kennis te maken met informatica. De computer, zuiver als hulpmiddel, als wel om door gebruik van programma's de mogelijkheden en functies van het apparaat te onderzoeken. Met een zekere trots kon Hartelust melden dat er thans zo'n 60 programma's beschikbaar zijn, ten dele vervat in leerpakketten.

"Boekjes, werkbladen, programma's, een experimenteerdoos, posters, een diasierie, een videoband, een gebruikerscursus, noemt u maar op. Wat anders dan dat oorspronkelijk geplande boekje".

Na afloop van het officiële deel werden in een bovenzaal van het ziekenhuis enkele nieuwe MSX-programma's getoond.

Ziekenhuispatiënt Arian Visser (20) uit Roosendaal zorgde voor uitleg en demonstratie aan de geïnteresseerde aanwezigen. "Het Input-project, ooit aangeduid als eendagsvlieg, pure reclame en visieloos genoemd, wordt inmiddels hoog gewaardeerd", betoogde Hartelust.

"Meer dan 1500 scholen maken gebruik van de programma's, direct via de spaarbanken, maar ook via Onderwijsbegeleidingsdiensten en de Onderwijs-

Werkgroep Philips-computers (OWG) worden kopieprogramma's verspreid". Aan het groeiende aantal scholen dat zich met behulp van de Input-apparatuur met computeronderwijs bezighoudt zijn nu dus alle ziekenhuischolen toegevoegd.

Hun leerkrachten meldden zich massaal aan voor de introductiecursus, die naar verwachting één twee personen per school zou opleveren, dus hooguit dertig.

Er was echter een toeloop van 120 belangstellenden.

Uitbreiding tot drie cursusdagen was het gevolg.

Hartelust daarover: "Ik wil best kwijt dat het goed doet op een initiatief zo'n positieve response te ontvangen.

Het sterkt ons in de overtuiging dat de computers op de juiste plaatsen terechtkomen".

Werken met de 8086

deel 3, door Mark Kramer.

Nee, mijn hart doet het nog prima, maar het kan toch geen kwaad het even te luchten: mijn kopij was ik niet vergeten in te leveren. SpaceQuest 3 is al evenzeer afgeraffeld als de editie van een ons niet onbekend computerblad en Borlands Turbo Assembler doet raar. Het eerste zult u even moeten aannemen en Sierra Offline? - het zegt u waarschijnlijk niet veel. Maar, Borlands Turbo Assembler die vreemd doet, daar wil ik toch wel even bij stil staan. Mocht ik mij laatstelijk nog blij verheugen bij een mededeling in: PCM, dat 'de Turbo Assembler volledig compatible' (met: MASM) is, zelf moet ik helaas anders ondervinden. Wat wil nu het geval? Bij de assemblage van de instructie: XOR SI,[DI+BP], genereerde de Turbo Assembler de volgende drie-byte object: 3E3333. Het eerste byte is een data-segment prefix, dus feitelijk staat er: XOR SI,DS:[DI+BP]. Op het eerste gezicht lijkt er niets aan de hand, want een data-segment prefix is default en kan in veel instructies worden vergeten (of er dus worden bijgezet). Helaas ziet MASM de instructie: XOR SI,[DI+BP] als: XOR SI,SS:[DI+BP] (logisch, want de Base Pointer is default aan het stack-segment gerelateerd). De correcte object voor zowel: XOR SI,[DI+BP] als: XOR SI,SS:[DI+BP], is dan ook de twee-byte code: 3333. Zonder te overdrijven, kan ik toch wel stellen dat Borland hier de plank behoorlijk mis slaat - morrelen aan het data-segment als de stack wordt bedoeld, is niet mijn idee van compatibiliteit (Wist u overigens dat IBM ook niet compatible is?). Koud schreef ik deze woorden neer, of ik ontdekte een tweede misser van de eerste orde. In een van mijn door MASM aangemaakte files stond een als: 'define double' aangemerkt BIOS-pointer, te weten: DD 00400015 (leeg). Na het file met TASM te hebben gerassembleerd, bestond TASM het om hier de volgende codes neer te zetten: 8F1A0600 (ziet er

uit als random, maar het was herhaalbaar). MASM compatible, zei u? De directive: .RADIX 16 aan het begin van mijn file werd op een woorddefinitie (DW 1234) wel correct uitgevoerd, maar daar koop ik weinig voor. Toevallig stond ik met: XRAY.COM net in het betreffende stukje BIOS te gluren (en een daar bedoelde vlag werd niet gezet), anders was ik er waarschijnlijk nooit achter gekomen. Maar, het zal daarmee duidelijk zijn, dat er een grens is aan dat wat ik bij toeval kan ontdekken. De lezer oordele zelf... Voor de goede orde: ik werk met MASM 5.1, en gebruik Turbo Assembler 1.00 met de switches: /jmasm51 /jquirks. Ik vrees op deze zaak nog terug te zullen moeten komen, maar laat ons eerst de draad weer oppakken bij het gebruik van labels in MASM. Voordien wil ik echter mijn erkentelijkheid uitspreken jegens de firma: Koning en Hartman uit Delft, voor het welwillend ter beschikking stellen van het: "80286 en 80287 Programmer's Reference Manual".

MOV AX,OFFSET TEST

"OFFSET", is een MASM pseudo-commando en doet in wezen precies hetzelfde als de processor-instructie: LEA (Load Effective Address). We zouden hier dus ook mogen schrijven:

LEA AX,TEST

al is deze instructie een byte langer. Houdt de labels simpel. Een veel gemaakte fout is, het uit de hogere talen geslopen waanidee dat de naam van een label altijd betrekking moet hebben op de routine waar het voor staat. Na een pijnlijke bevalling weet de auteur zich dan namen te ontlokken als: 'isalgei:', 'ong_prm:', 'no_inerr_2:', etc. Oordeel zelf.

1.5.2 Wat u van een label-fout moet weten. Veel label-fouten hebben een

zogenaamde: 'Type Mismatch' als oorsprong. Dat kunt u alleen zelf oplossen. Soms kan het echter heel geniepig fout gaan. Een voorbeeld wil ik u niet onthouden.

Fig. 1.6

DSEG	SEGMENT
	DB 'Your typical data'
DSEG	ENDS
	...
CSEG	SEGMENT
	...
	PUSH CS
	POP DS
	...
OOPS:	MOV BX,DS:[ADR12]
ADR12	DW (?)
	...
CSEG	ENDS

Hier gaat iets gruwelijk mis. Omdat DS nu eenmaal default is, kwam u op het schrandere idee om het code-segment in DS te POPpen, om zo bij het label: OOPS, een byte uit te kunnen sparen (een segment-override vergt immers een extra byte, weet u nog?). Een onaangename verrassing volgt: in plaats dat we in: BX de waarde uit: [ADR12] krijgen - zoals we vroegen - vinden we in: BX de waarde uit: [ADR12+17]. Plus zeventien? Jawel, want laat dat nu precies de lengte van ons data-segment zijn. De bovenstaande vlieger gaat dus enkel op, als het data-segment helemaal leeg is.

Mind you. We kunnen dat op twee manieren oplossen. Door in het begin van ons programma: ASSUME DS:DSEG te wijzigen in: ASSUME DS:CSEG, maar daarmee lopen we een gerede kans dat de rest van het programma de mist in gaat. Neen, beter is het een plaatseijke assume-override toe te passen:

ASSUME DS:CSEG
OOPS: MOV BX,DS:[ADR12]

ASSUME DS:DSEG

1.6 BYTE PTR en WORD PTR.
De 8086 mnemonische instructie-set laat een aantal dubbelzinnigheden toe, die we handmatig moeten omzetten. Dit kunnen we goed illustreren aan de hand van een Z80 voorbeeld.

Fig. 1.7

8086

LD (HL),06 MOV DS:[BX],06

Wat er bij de Z80 staat is duidelijk, maar hetgeen MASM bij de 8086 te verwerken krijgt spreekt niet voor zich en kan hier op twee manieren worden geïnterpreteerd: 'laadt op de geheugenplaats, geadresseerd door [BX], de byte-waarde: 06', als ook: 'laadt [BX] plus [BX+01] met de woord-waarde: 0006'. Voor dergelijke gevallen nemen we onze toevlucht tot de pseudo-commando's: BYTE PTR (Byte Pointer) en: WORD PTR (Word Pointer):

MOV DS:BYTE PTR [BX],06
MOV DS:WORD PTR [BX],06

Dit is niet nodig bij instructies als: MOV DS:[BX],AX. Want het staat buiten kijf dat een enkel-byte adres nu eenmaal geen 16-bits register kan verstouwen (Mocht u daar toch in slagen, dan wil ik graag delen in de winst). Soms zit de knik niet in de operand, maar in de operator zelf. Voorbeeld:

MOV CS:[ADR5],AX

ADR5 DB (?)
ADR6 DB (?)

MASM genereert hier een Warning Error, die neerkomt op een Type Mismatch. ADR5 is als byte gedefinieerd, maar wordt als woord aangesproken. Hier schrijven we dus: MOV CS:WORD PTR [ADR5],AX. We kunnen deze waarschuwing ('Operand type must match') overigens wel in de wind slaan, maar het staat zo slordig.

1.6.1 32-bits operaties met: DWORD PTR.
Ofschoon de 8086 een 16-bits processor is, zijn er twee instructies waarmee 32 bits kunnen worden verwerkt (intern gewoon in tweeën gedeeld, overigens). Het zijn: LDS en: LES, en ze dienen om de segment-registers: DS en: ES te laden in combinatie met een ander register, waarbij in het geheugen de offset voorop staat. Omdat het hier een dubbel-woord betreft, gebruiken we de: DWORD PTR directive, tenzij we verwijzen naar een als: DD (Dubbel Woord) gedefinieerd label (DWORD PTR mogen we ook dan gebruiken, maar een omissie levert dan geen warning errors meer op). Een kleine toelichting kan geen kwaad; houdt de wijze waarop het geheugen wordt uitgelezen, goed in de gaten:

LDS SI,CS:DWORD PTR [ADR12]

ADR12 DW OFFSET label
ADR13 DW SEG DSEG

Na uitvoer van deze instructie bevat SI de waarde: 1234 en DS de inhoud van het volgende woord: 5678. Ik laat het aan uw eigen fantasie over, om te bedenken wat er gebeurt wanneer we LDS vervangen door: LES. Als operand mogen - natuurlijk - ook index-registers dienen. Dan krijg je zo iets:

LES DI,[BX+SI], of:
LDS BP,ES:[SI+BX+560A]

Zit in de instructies geen geheugen-operand, dan mogen we de PTR-aanduidingen weglaten.

1.7 De stack.

Neen, ik ga u niet vervelen met de totaal afgesleten metafoor van de immiddels stukgebruikte bordenstapelaar in een restaurant. U weet allang wat een stack is en hoe die te gebruiken. Een aantal zaken wil ik echter nog wel even over het voetlicht brengen.

1.7.1 De stack in .COM en .EXE files. Een .COM-file kijkt niet langer dan zijn segment lang is, 64K. Afgezien van de enorme verkwisting van die toch al zo dure RAM-IC's, kon er zelfs (of: juist?) geen vaste ruimte voor de stack vanaf. De EXEC-functie van DOS plaatst de stack helemaal bovenin het (segment)geheugen - offset: FFFF. De enige vereiste, die aan een .COM-file wordt gesteld, is dat er voor de stack minimaal twee bytes overblijven. De lengte van de stack is dus heel banaal: dat wat er na het programma van het segment overblijft. 'On entry', staan: SS en: SP reeds op de juiste waarden. Hoeveel te meer "sophisticated" ligt de zaak bij een .EXE-file. Zoals blijkt uit: fig. 1.5 maken we voor de stack een eigen segment aan, met een lengte die we zelf verkiezen. Ook hier worden: SS en: SP door de EXE-loader bij aanvang van het programma op de juiste waarden gezet.

1.7.2 PUSH, POP en de stack in het algemeen.

Over het gebruik van de stapel doen nogal wat misverstanden de ronde, niet in het minst aangewakkerd door de hogere talen en hun gebruikers. Sinds jaar en dag schijnt het mode te zijn, om de stack te misbruiken voor de opslag van data. Inkomende data worden gewoonweg massaal op de stack gedumpt, die dan dikwijls harder groeit dan mijn weerzin ertegen. Zo legt Turbo Pascal een stapel aan met de grootte van een standaard P2000-geheugen (16K), en het zou niet de eerste keer zijn dat ik (ook in TRON) geluiden verneem, van mensen die daar nog prat op gaan ook. Omdat de verstengeling van data en return-adressen voor problemen kan zorgen, bedienen de hogere talen zich van tamelijk omslachtige me-

chanismen om toch alles op z'n POPjes terecht te laten komen. Dat wil (u raadt het al) wel eens mis gaan. Was ik lekker bezig in ProComm, geeft de PTT er zomaar de brui aan. Niets aan de hand natuurlijk, of het moet die: 'Stack Overflow' zijn die, samen met een heleboel andere onzin, plotseling op mijn scherm verscheen. Een koude reset doet wonderen in zo'n geval, maar of ik blij was? Het lijkt me wel duidelijk, waar ik naar toe wil: gebruik de stack enkel voor het doel waar toe hij is ontworpen: de opslag van return-adressen en register-variabelen. De 8086 kent talloze krachtige instructies die de opslag en verplaatsing van data veel efficiënter bewerkstelligen. Sterker nog, vele instructies zijn geënt op het gebruik van een apart data-segment als default. Doorgaans kunt u voor de stack met 256 woorden ruimschoots volstaan, en geloof me: zijn er bij het laatste return-adres nog veertig wachtenden voor hem, dan is er iets grondig mis gegaan.

We hebben ze al genoemd: PUSH en: POP. Oude bekenden, al zijn de variaties bij de 8086 wat groter. Alle registers kunnen op de stapel worden gezet (dus ook de segmenten). Nieuw is echter dat ook alle register-afgeleiden een plaats op de stack kunnen krijgen. Een paar voorbeelden:

```
PUSH AX
PUSH CS
PUSH BX
PUSH [BX]
PUSH [DI+BX]
PUSH [DI+BX+05AF]
```

Helemaal handig is dat ook geheugen-adressen met PUSH kunnen worden 'gedrukt', zelfs met segment-override:

```
PUSH DS:[ADR1]
PUSH CS:[ADR14]
```

Voor: POP geldt dat deze in wezen complementair is aan: PUSH. Niet toegestaan is echter: POP CS (lijkt me duidelijk). POP SS en: POP SP mogen ook, maar zodra u een van deze twee heeft gePOPT, ligt de stack al ergens anders. Een klein gevaar schuilt in een POP van geheugen-operands. Stel, een interrupt-routine begint met het onderhavige stukje code:

```
POP CS:[ADR1]
POP CS:[ADR2]
POP CS:[ADR3]
```

De laatste drie woorden zijn van de stack gehaald en opgeslagen in RAM. Een nadeel van deze methode is dat een op deze wijze geschreven routine niet meer re-entrant is. D.w.z., dat dit stukje machinetaal niet recursief mag worden aangeroepen, omdat dan de oorspronkelijke waarden zijn vernietigd. Heeft u het vermoeden dat een recursie niet ondenkbaar is (onder DOS zelfs schering en inslag), verkies dan een andere methode van 'data-retrieval'. Bij de besprekking van het interrupt-mechanisme komen

we hier nog uitvoerig op terug.
N.B. Voor de AT-bezitter: 286-programmeurs mogen de stack ook immediate gebruiken: PUSH 1234, PUSH OFFSET label. Een 'immediate pop' is dan natuurlijk "out of the question". Bovendien mogen zij met: PUSHA en: POPA 'alle' registers op de stapel zetten, respectievelijk er vanaf halen. Voor de opslag van de segment-registers moet u echter zelf zorgdragen. Bij de 386 verhindert een: POP SS alle interrupts (ook de NMI) tot na uitvoering van de volgende instructie, zodat achtereenvolgens: POP SS, POP SP veilig kunnen worden uitgevoerd. Het gebruik van: LSS SP, geniet - aldus het: '80386 Programmer's Reference Manual' van Intel - echter de voorkeur (overigens kan schrijver dezes zich de zin van: POP SP na een: POP SS maar slecht voorstellen).

1.8 De schijf van vijf: SCAS, LODS, STOS, MOVS, CMPS.
Zoals ik al had aangestipt, beschikt de 8086 over een heel arsenaal instructies voor de behandeling van data. Het is een speciale groep, die we: string-instructies noemen. Ze zijn voor het laden, vergelijken, zoeken en (legaal) kopiëren van data, ontworpen. Sommigen hebben een vergelijkbaar Z80-equivalent, anderen niet. We zullen ze daarom allemaal kort de revue laten passeren. Is er sprake van een bron-bestemming-operatie, dan dient het volgende te worden opgemerkt: de bron default is het registerpaar: DS:SI en: ES:DI is de bestemming (Data-segment + Source-index en: Extra-segment + Destination-index). Daar kunnen we niet omheen (Een NEC V20 kent ook string-bewerkingen met segmentoverride). Betreft het een instructie waar slechts een enkel registerpaar (segment + index) bij is betrokken, dan is: ES:DI de basis, behalve bij: LODS. Dit mag wat verwarring klinken, maar de zin daarvan zal spoedig duidelijk worden. Bovendien zijn opgemerkt dat er van al deze vijf instructies twee variaties bestaan, een byte- en een woordoperatie, respectievelijk aangeduid met het achtervoegsel: 'B', of: 'W'.

1.8.1 SCAS (SCAn String).
Deze instructie vertoont duidelijke overeenkomst met: CPI (CPD), van de Z80. In: AL (AX) zetten we het te zoeken byte, in: ES:DI het basis-adres en in: CX een teller.

Fig. 1.8

Z80	8086
CPI(CPD)	SCASB(SCASW)
A = byte	AL (AX) = byte- (woord)
HL=zoekadres	ES:DI = zoekadres
BC=teller(16)	CX=teller(16)

Wat opvalt, is dat de Z80-instructie: CPI een aparte tegenhanger heeft in: CPD, en de 8086 niet. Dit is juist. De 8086 heeft het increment-decrement

probleem niet opgelost door verschillende instructies te maken, maar de richting laten afhangen van de status van een aparte data-vlag (DF). Is deze vlag gestreken, dan is de richting positief, anders negatief. Voordat we dus kunnen beginnen met een repetente zoek-actie, wissen we eerst de data-vlag met: CLD (CLear Data), of: STD (SeT Data). Dus:

```
CLD  
SCASB
```

zoekt een byte vanaf de lokatie geadresseerd door ES:DI en verhoogt DI na afloop met 1 (na: STD wordt DI dus 1 verlaagd). SCASW doet DI daarentegen met twee verhogen of verlagen. Maak er een goede gewoonte van, bij iedere stringinstructie de data-vlag overeenkomstig de wens te zetten.

1.8.1.1 Repetente string-bewerking met: REPZ, REPNZ.

Een herhaalde string-actie met: REPeat (R bij de Z80), geven de string-instructies pas zin. Z en NZ geven aan dat de actie door moet gaan, totdat aan de voorwaarde is voldaan (Z), respectievelijk tot het moment dat van de voorwaarde wordt afgeweken (NZ). REPZ en REPNZ zijn byte-voorvoegsels, die enkel voor een string-instructie effect sorteren. We geven zowel voor de Z80 als de 8086 een voorbeeld.

Fig. 1.9

Z80	8086
LD A,20	MOV AX,ZOEK
LD HL,ZOEK	MOV ES,AX
LD BC,AANTAL	MOV CX,AANTAL
CPIR	MOV AL,20
	MOV DI,OFFSET ZOEK1
	CLD
	REPNZ SCASB

Merk de: REPNZ op bij de 8086. De opdracht luidde immers: "blijf doorgaan met zoeken zolang het byte niet (NZ) is gevonden". Zowel bij de Z80 als de 8086 is de zoek-actie geslaagd (het byte is gevonden) bij: Z, respectievelijk mislukt bij: NZ. Significant verschil met de Z80 is het gebruik van CX (BC): voor de 8086 geldt, dat CX enkel verandert bij een repeateer-opdracht, daar waar een: CPI (CPD), BC altijd wijzigt. SCASB laat CX ongemoeid, REPNZ SCASB dus niet.

1.8.2 LODS (LOaD String).

LODS laadt een byte(woord) in: AL(AX), geadresseerd door: DS:SI, en vertoont overeenkomst met: LD A,(DE) van de Z80. Na afloop: AL = [SI] en SI = SI + 1, of: AX = [SI] en SI = SI + 2. LODS is een string-instructie, en mag dus ook worden voorafgegaan door de prefix: REP. Veel zin heeft dat natuurlijk niet.

1.8.3 STOS (STOre String).

STOS is in zekere zin complementair

aan: LODS en slaat een byte(woord) uit: AL(AX) op in een lokatie geadresseerd door: ES:DI. Dit komt goed uit, want zo kunt u bv. met: LODSB eerst een byte in AL laden, dat byte vervolgens wijzigen of testen, om het dan met: STOSB weer op te slaan. Een klein voorbeeld:

Fig. 2.0

Z80	8086
LD BC,0800	XOR DI,DI
LD HL,5000	MOV AX,0B800
LD (HL).L	MOV ES,AX
LD DE,5001	CLD
LDIR	XOR AX,AX
	MOV CX,2000D
	REPZ STOSW

Bij de Z80 halen we een truc uit: we 'misbruiken' een bestaande repetente string-opdracht (LDIR) voor een niet bestaande functie (REP STOSB). Dit soort grappen zullen we ook bij de 8086 tegenkomen en ze behoren tot het standaard arsenaal van iedere welopgevoede machinetaal-programmeur.

1.8.4 MOVS (MOVE String).

MOVS is een samenvoeging van: LODS en STOS, maar de data-overdracht gaat direct, zonder tussenkomst van AL(AX). Laat ons dit toelichten:

Fig 2.1

Z80	8086
LD BC,0100	MOV CX,0080
LD DE,DEST	MOV AX,DEST
LD HL,SOURCE	MOV ES,AX
LDIR	MOV DI,OFFSET DEST1
	MOV AX,SOURCE
	MOV DS,AX
	MOV SI,OFFSET SOUR-CE1
	CLD
	REPZ MOVSW

In beide gevallen verschuiven (kopiëren) we 256 bytes. Maar, omdat het hier om een even aantal gaat, verschuiven we bij de 8086 niet 256 bytes, maar 128 (0080) woorden - dat gaat sneller.

1.8.4.1 Alignment en de EVEN directive.

Voor iedere woord-operatie op een oneven adres(offset), heeft de 8086 vier cycli extra nodig. Wie grote lappen data wil verschuiven (of zoeken, of vergelijken, etc), doet er dus verstandig aan, deze data te laten beginnen op een even adres. Dat kan met de MASM directive: EVEN en wordt als volgt gebruikt:

```
EVEN
SOURCE1 DB
'Dit is de bron-tekst uit: fig 2.1'
```

De string-instructie is in dit geval 'in alignment' met de data. Op de 256 bytes uit ons voorbeeld, scheelt

dit toch weer mooi: $128 * 4 = 512$
cycli.

N.B. Bij een (oneven) woord-operatie aan het einde van een offset (altijd: FFFF), vindt bovendien een 'offset wrap-around' plaats: de offset 'loopt over' en begint weer bij nul. Bij de 286 en 386 wordt in zo'n geval interrupt: 13 gegenereerd (ook in de Virtual 8086 Mode). Bij de 8086 blijft dit zonder gevolgen.

1.8.5 CMPS (CoMPare String).

CMPS vergelijkt een string uit: DS:SI met een string uit: ES:DI voor de lengte van: CX. Hierbij worden de vlaggen gezet alsof: ES:DI werd afgetrokken van: DS:SI. Dit gaat aldus:

Fig. 2.2

Z80	8086
LD B,05	MOV CX,0005
LD HL,DEST	MOV AX,DEST
LD DE,SOURCE	MOV ES,AX
P1 LD A,(DE)	MOV AX,SOURCE
CP (HL)	MOV DS,AX
RET NZ	MOV SI,OFFSET SOUR- CE1
INC HL	CLD
INC DE	MOV DI,OFFSET DEST1
DJNZ P1	REPZ CMPSB
RET	RET

De volgende keer besteden we aandacht aan: loops, gateways en de structuur van DOS.

Datacompressie.

Het is het toverwoord van de heden-dagse computer branche. Pctools' backup en de nieuwere generatie modems doen het 'on-line'. Terwijl voor het gewone overzenden van file of het bewaren ervan, er tientallen programma's zijn die het werk doen terwijl u wacht. Het is zelfs noodzaak geworden, zo'n programma te hebben voor als u wel eens met een modem een file uit een databank haalt. Waar u ook kijkt, u komt ze overal tegen!

Datacompressie, het klinkt ingebur-gerd, maar wie je er ook naar vraagt, niemand weet precies hoe nu wel de bekende programma's als ARC, PKXARC, NARC, PKZIP, PAK of LHARC werken. Zelf weet ik het ook niet precies. Dat was een lekker kort artikel, zult u denken! Mis, in de rest van dit artikel zal ik toch proberen, een beetje opheldering te geven over, hoe datacompressie nu eigenlijk in z'n werk gaat.

Het lijkt op het eerste gezicht onmogelijk van een file iets af te halen. Immers, de informatie die de file bevat, is al zo compact. Nemen we een tekst-file, dan valt dat nog wel mee. Als we uitgaan van 26 hoofd en 26 kleine letters plus nog een stuk van vijf leestekens en de cijfers 0 t/m 9, dan hebben we bij elkaar dus 67 verschillende karakters. Als we dan uitgaan van het binair stelsel, dat onze computer gebruikt, komen we uit bij de 7 bits die nodig zijn, om een tekst-file te coderen. 2 tot-de-macht 6 is immers 64 en 2 tot-de-macht 7 127 waardoor we aan 6 bits net te weinig hebben en aan 7 iets te veel. We kennen geen halve bits, dus 7 bits wordt het en het te veel nemen we op de koop toe. Ja, ik weet het, dit is droog maar, bijt even door! De computer slaat een karakter helaas als file van 8 bits op, zodat er per gebruikt karakter eigenlijk een bit teveel op uw diskette wordt weggeschreven. Goh, zult u denken, een flopje is 360Kb groot, dat zijn dus 360 maal 1024 is 368640 karakters die erop kunnen. En ik me druk maken om dat ene beetje!

Helemaal gelijk, ruimte zat: Maar laten we nu eens kijken wat er

gebeurt als u deze diskette, propvol met tekst, over de telefoonlijn gaat versturen. We nemen voor het gemak even de standaard-snelheid van 1200 baud (is ongeveer gelijk aan bits per seconde) Om een karakter over een telefoonlijn te sturen, zijn 10 bits nodig; een startbit, de standaard 8 databits en een stopbit. Het kan ook zonder deze start- en stop-onzin, maar dan hebben we peperdure modems nodig. Met ons telefoonlijntje kunnen we dus precies 120 karakters per seconde verzenden!

Nu even terug naar de 368640 karakters die we te verzenden hebben. Dat gaat dus 368640 gedeeld door 120 is 3072 seconden duren. Gedurende deze ruim 51 minuten dat we de PTT uit de rode cijfers aan het halen zijn, zijn we ook nog steeds die 368640 loze bits aan het meezendende. Deze 307 seconden dat het ons meer kost om die bits mee te sturen, zijn weliswaar niet veel maar toch genoeg om er iets aan te gaan doen!

We kunnen natuurlijk met bits gaan schuiven. Als we nu eens gewoon alle 7 bits van iedere letter achter elkaar leggen. Dan krijgen we een hele lange rij met eenen en nullen. Als we dan vervolgens die rij weer opdelen in groepjes van 8 bits, hebben we de loze bits eruit gehaald!

Schematisch ziet het er dan ongeveer zo uit :

```
B.v. de letters 'ABC' :  
01000001 01000010 01000011  
de 7 bits daarvan zijn :  
.1000001 .1000010 .1000011  
achter elkaar wordt dat :  
.1000001.1000010.1000011  
en zonder de loze bits :  
100000110000101000011  
in groepjes van 8 bits :  
10000011 00001010 00011xxx
```

U ziet het, de onderste rij is al een flink stuk korter dan de rij met de letters. Drie bits om precies te zijn. Zo kunnen we per 8 letters een heel karakter besparen.

Nou, dat viel best mee, hebben we

mooi even 12.5 % bespaart met een hele simpele methode.

Maar tot nu toe zijn we uitgegaan van een tekstfile, waarbij er een beperkt aantal karakters was, zodat we gebruik konden maken van het feit dat er minder karakters waren, dan in totaal mogelijk waren geweest. Maar wat nu, als we te maken hebben met een binaire file, een file dus waarbij alle 256 mogelijke karakters ook worden gebruikt. De truc met het schuiven van een loze bit is er dan niet bij.

Om nu nog iets van de file af te kunnen halen, zullen we wat moeilijker methoden moeten gaan gebruiken. Waar we nu nog wat mee zouden kunnen doen, is kijken naar de verdeling van de verschillende karakters. Sommige karakters zullen immers vaker voorkomen dan andere. Als we die file met karakters nu eens niet zien als een file met karakters, maar als een file met getallen die ieder naar een plaats in een tabel wijzen. Zo zal dan bijvoorbeeld het getal 65 naar de letter A wijzen en het getal 66 naar de letter B enz., enz. We hebben dan een file met getallen en een vertaal-tabel met wat die getallen voorstellen gekregen. Op zich levert dit geen verkorting van de file op, eerder een verlenging. Immers we hebben nu ook nog een keer te maken met een tabel, die we ook nog zullen moeten opslaan, naast de file met getallen. Maar na deze ogenschijnlijk zinloze verlenging van onze file, kunnen we er toch een aantal aardige dingen mee doen. Als we nu het getal dat naar het karakter wijst, wat het meest voorkomt, een bit korter maken en het getal dat naar het karakter wijst, wat het minst voorkomt, een bit langer. Op zich mag dit voor de file niets uitmaken, immers er is ergens een bit weggehaald en ergens anders een bit bijgezet. Er zijn dus nog steeds 256 mogelijkheden te maken. Wat alleen nu wel het geval is, is dat het getal, wat we korter hebben gemaakt, vaker voorkomt dan het getal wat we langer hebben gemaakt. Doen we dit met nog meer getallen en soms wel twee of drie keer met hetzelfde getal, dan

krijgen we een file met heel uiteenlopende bit-lengtes, er kunnen dan getallen met maar 2 bits voorkomen en getallen met wel 12 bits. Door het feit dat die korte waarden het meest voorkomen en die lange het minst, krijgen we - geheel afhankelijk van de file-opbouw natuurlijk - een veel kortere file !

Helaas gaat het niet op deze manier, zoals ik het hier heb beschreven. Je kunt immers niet ongestraft ergens een bit afhalen of bijzetten, zonder een puinhoop te krijgen. Het is echter wel de globale methode die wordt toegepast. Willen we het werkelijk op deze manier doen, dan zullen we er dus eerst voor moeten zorgen dat we ergens ongestraft bits af kunnen halen of bijzetten. Op zich is dat nog wel te doen, maar eenvoudiger is het, om gewoon heel anders te beginnen.

We doen even een sprong terug naar de binaire file, die we hadden en waarin sommige karakters vaker voorkwamen dan anderen. Voor het gemak zullen we een file nemen met maar 5 verschillende karakters erin; A,B,C,D en E, die respectievelijk 3,11,8,7 en 14 keer in de file voorkomen. Helaas zult u zeggen; dat is weer een tekstfile, gaan we nu weer zitten rommelen met die loze bit ? Nee, ik heb voor dit voorbeeld maar 5 letters genomen, omdat een voorbeeld met alle 256 mogelijke karakters, zonder te overdrijven, enkele tronen aan papier zouden kosten. Maar oké, 5 letters en hun frequentie.

A	B	C	D	E
3	11	8	7	14

Als we nu keer op keer de karakters met de laagste frequentie met elkaar verbinden door middel van een streepje, naar een nieuwe positie in de rij, die dan de frequentie krijgt van de som van beide karakters, wordt de rij iedere keer een karakter korter. Voor het overzicht is het soms handig, twee karakters van plaats te wisselen, zodat het een nette 'boom' wordt.

A	D	C	B	E
3	7	8	11	14
\	/			
\./	C	B	E	
10	8	11	14	

Nu hebben we een denkbeeldige letter gemaakt, die ontstaan is uit de letters A en D en die dus bij elkaar 10 keer voorkomt in de file. Laten we dit nog eens een paar keer doen.

A	D	C	B	E
3	7	8	11	14
\	/			
\./	C	B	E	
10	8	11	14	
\	/			
\./	B	E		
18	11	14		
\	\	/		
\./	.	/		
18	25			
\./				
43				

Nu, dat ziet er niet slecht uit. Maar, wat kunnen we ermee. Door deze 'boom' terug te vertalen naar eenen en nullen, kunnen we een positie in de 'boom' aangeven en daarmee ook meteen het karakter zelf. Door nu bijvoorbeeld af te spreken, dat als we een stap naar rechts doen, we een één ophrijven en als we een stap naar links doen een nul, krijgen we voor ieder karakter een rij eenen en nullen, die precies de plaats in de boom aangeeft. Bijvoorbeeld, om bij de letter C te komen, moeten we twee keer naar links en een keer naar rechts. De letter C krijgt dan ook de code 001.

Het lijstje wordt dan :

A	0000
B	10
C	001
D	0001
E	11

En wat zien we? De karakters die vaak voorkomen, zijn korts gecodeerd dan de karakters, die niet zo vaak voorkomen. Deze manier van coderen, de "Hoffman code", heeft ons dus de volgende verkorting opgeleverd : A : 4 bits, B : 2 bits, C : 3 bits, D : 4 bits, E : 2 bits.

De A kwam 3 keer voor, de B 11 keer, de C 8 keer, de D 7 keer en de letter E 14 keer.

Bij elkaar hebben we dus $3 \times 4 + 11 \times 2 + 8 \times 3 + 7 \times 4 + 14 \times 2 = 114$ bits nodig, om de file te coderen. Als we niet hadden gecodeerd, hadden we $3 \times 8 + 11 \times 8 + 8 \times 8 + 7 \times 8 + 14 \times 8 = 344$ bits nodig gehad. We hebben dus nog maar een file, die 33% van de lengte van de oude file is. Niet slecht toch ?

Helaas is ook dit betrekkelijk. Aan alleen deze file hebben we niet veel, want uw computer zal, bij het zien van de code 001, niet meteen denken aan de letter C. Dus zal de hele 'boom' ook nog in de file moeten worden opgeslagen en dat kost heel wat ruimte. De slimmere compressiemethoden slaan dan ook de boom niet in z'n geheel op, maar stukje bij beetje door de hele file heen, zodat - bij het terug vertalen - de computer als het ware de 'boom' leert opbouwen.

De computer leest gewoon bit voor bit in en als die een nieuwe, tot dan toe onbekende code tegenkomt, zijn de 8 daarop volgende bits van het karakter dat die code moet voorstellen.

Oké, we kunnen nu dus een file comprimeren, door voor iedere 8 bits een nieuwe code te vinden, die korts is dan de ASCII code van het karakter. Maar, het zou best wel eens zo kunnen zijn, dat wanneer we naar groepjes van 7 bits gaan kijken in plaats van 8 bits, we nog veel meer van de file af kunnen halen. Zo zou ook een 5- of een 6-bits, of misschien wel een 9- of een 10-bits codering, nog veel meer van de file af kunnen halen. Daarom gebruiken de modernere compressie programma's ook een variabel aantal bits voor het comprimeren. Soms wel 2 tot 13

bits, zoals LHARC. Die ondergrens van 2 bits is natuurlijk, omdat je moeilijk een 1 bits compressie kan toepassen. Als je ieder bit gaat vervangen door een of meer andere bits, wordt je file alleen maar langer en dat was nu net niet de bedoeling. Die bovenlimiet van 13 bits komt, denk ik, door het feit dat de meeste PC's 'maar' 640Kb intern geheugen hebben. Een boom maken voor een 13 bits codering kost ons immers $2^*(2^{13}) - 1$ knopen, en als we voorzichtig in iedere knoop twee 4 bytes Pointers zetten en nog een 2 bytes Word voor de eigenlijke waarde plus nog een byte als vlag, om aan te geven of die knoop een knoop is of een eind-record, komen we uit rond de 200Kb.

Een 14 bits codering zou dit weer verdubbelen en dat zou al krap worden voor de meeste computers.

Ha, zult u denken, als het niet meer in het geheugen gaat, dan maar op schijf. Laten we die 20 Meg harddisk toch even rammelen ! Jammer dat we dan nog niet verder komen dan een 19 bits codering en een enorme wachttijd.

Inmiddels wordt er weer druk verder gedacht over compressie. Zo is nog niet zo lang de MNP5 compressiemethode op de markt gekomen. In de duurdere modems zit die ingebouwd, zodat de snelheid van het modem ermee zou kunnen verdubbelen, volgens de fabrikant.... Een geweldige ontwikkeling, maar uit dit hele artikel heeft u kunnen opmaken dat een compressie eigenlijk maar één keer lukt. (lukt het meer dan eens op dezelfde file, dan is het niet goed gebeurd !) Door de "Hoffman code" neemt de verscheidenheid van de karakters namelijk sterk af. Die MNP5 compressie zal dus aardig werken, als u gewone file'tjes aan het overpadden gaat. Maar, als u gecomprimeerde files gaat overzenden, en dat is meer regel dan uitzondering, komt u weer terug op de oude modemsnelheid. In sommige BBS-en staat een emulatie programma, om op oude modems deze MNP5 techniek te laten lopen. Probeer het eens, maar bedenk dus wel dat een goed van te voren gecomprimeerde file, u nog meer tijdwinst oplevert. Dat er om dit moment echt prima comprimieerprogramma's bestaan, kunt u uit de volgende tabel halen. Als voorbeeld het ik de WP-tekst-file van dit artikel genomen.

Totaal	15835 bytes	100.0 %
LHARC	6914 bytes	43.7 %
PKARC	7919 bytes	50.0 %
PAK	7953 bytes	50.2 %
PKZIP	8026 bytes	50.7 %

Nu blijkt al dat deze proef niet echt representatief is, aangezien PKZIP normaal veel beter is dan PKARC, maar afhankelijk van de file, valt dit dus best tegen.

Tot slot nog even iets over Simpel-tel: als u daar eens wat gratis software uit heeft gehaald, heeft u misschien gezien dat die files, na behandeld te zijn met UNSQZ, alleen nog maar korts worden. De SQZ file zijn dus langer, dan de file die eruit komt. Denk nu niet meteen

'welke zot is daar bezig geweest', want dat heeft wel degelijk een gunstige uitwerking op uw telefoon rekening. Omdat de Telesoftware Standaard, zoals die algemeen hier in Nederland wordt gebruikt, voor alle karakters onder de 32 en voor alle karakters boven de 122 een stuur-karakter zet, wordt het aantal lettertekens dat op de pagina's moet worden neergezet, gemiddeld 180% groter dan de eigenlijke file lang is. Dit is natuurlijk omdat de

Viditel-standaard voorschrijft dat op een pagina geen karakters onder de 32 en boven de 127 mogen staan. Maar dat dit nog geen reden is de software zo banaal te coderen, is duidelijk. De telesoftware in Simpel-tel is daarom voorbewerkt met een programma, zodat de telesoftware steeds 125% langer wordt dan de oude file lengte. Dit is nog altijd beter dan 180%, denkt u niet? Omdat die 180% het uiterste geval is, kan het natuurlijk ook nog voorkomen dat

het onder de 125% komt en in dat geval zal er dus ook nooit een SQZ file van worden gemaakt! Ook daarmee loopt Simpel-tel dus weer voorop in Nederland.

Na zoveel geslijm, zult u wel niet meer verbaasd zijn, als u wat van mijn programma's in Simpel-tel ziet staan.

Jeroen Wortelboer

Kanttekeningen bij de PC deel door Dick Bruggemans

XCOPY

Veel mensen kennen dit handige programmaatje niet, terwijl het vanaf MS/DOS 3.2 standaard in uw software aanwezig is.

Wat doet dit programma? In essentie hetzelfde als het bekende COPY-commando. Indien u echter meer dan één file wilt kopiëren (b.v. COPY *.* A:), zal XCOPY eerst alle te kopiëren files inlezen en ze daarna pas dupliveren (dus in het voorbeeld: op diskette A: zetten). Het verschil is de enorme tijdswinst! XCOPY kent nog meer mogelijkheden die handig kunnen zijn, maar zie hiervoor uw handboek.

ERRORLEVEL

In een batchfile kunt u testen op een z.g. errorlevel dat een programma opzet en zo besturen wat er vervolgens moet gebeuren. Veel programma's zetten een errorlevel, groter dan nul, om daarmee aan te geven dat er iets mis is gegaan.

Voorbeeld: Een programma met de naam HANDIG dient te worden uitgevoerd. Als de verwerking goed is verlopen, dient er "RAAK" op het scherm te verschijnen. Als het echter fout gaat, u raadt het al, "MIS".

De batchfile ziet er als volgt uit

```
ECHO OFF
HANDIG
IF ERRORLEVEL 1 GOTO MIS
ECHO "RAAK"
GOTO EINDE
:MIS
ECHO "MIS"
:EINDE
```

Met GOTO wordt de verwerking vervolgd bij het label (naam) dat erna volgt: in ons geval MIS. Het label begint altijd met een dubbele punt(:), om het van een programmanaam te onderscheiden. Let op! IF ERRORLEVEL 1 betekent: indien het ERRORLEVEL groter of gelijk is aan 1! Dus IF ERRORLEVEL 0 is eigenlijk onzin, omdat dan altijd aan de conditie wordt

voldaan (immers alles is altijd groter of gelijk aan 0, want negatief kan niet).

Indien het programma meer dan één errorlevel kan zetten (dus meerdere waarden van ERRORLEVEL), dan dient u daarom altijd eerst de hoogste waarde te testen en dan vervolgens de waarde die daar onder komt, etc.. etc..

Als voorbeeld:

```
IF ERRORLEVEL 255 GOTO LABEL 255
IF ERRORLEVEL 100 GOTO LABEL 100
IF ERRORLEVEL 010 GOTO ECHO "errorlevel is 10"
GOTO EINDE
:LABEL255
GOTO EINDE
:LABEL100
EINDE
```

Wel dat was het weer voor deze keer

LOTUS 1-2-3 deel IV

Hoofdstuk 4

het maken van deel werkbladen

Voordat we aan hoofdstuk 4 gaan beginnen, volgt hier de oplossing van opdracht 4.

Het getal dat eruit moest komen is 450. Hoe komen we daar nu aan? De @COUNT-functie telt 3 cellen (B3, B4 en B5), de @SUM-functie binnen de @SUM-functie telt de cellen B3 t/m B5 op. De eerste @SUM-functie telt de uitkomst van de @COUNT-functie (3) en de tweede @SUM-functie (225) op. Het totaal is dus 228. Dit getal wordt met 2 vermenigvuldigd (2 * 228 = 456) en van dit laatste getal gaat 6 af. Er blijft dus 450 over. Wanneer u heel goed de opdracht bekijkt, dan ziet u het antwoord al staan. Er staat n.l. OPDRACHT 4(50).

In dit hoofdstuk wordt het maken van een deel-spreadsheet behandeld. Hiervoor wordt de Engelse term Range-zone gebruikt.

Aan de hand van enkele voorbeelden

zal worden toegelicht wat de mogelijkheden van de range-zones zijn. Met een range-zone kunnen diverse manipulaties worden uitgevoerd, zoals het kopiëren, verplaatsen en beschermen tegen wijzigingen.

Vooruitlopend op hoofdstuk 8, vertel ik u, hoe u een spreadsheet van schijf kunt lezen. Type in:

```
/ (slash)
F (file)
R (retrieve)
WINKEELS
```

<R>. Het spreadsheet WINKEELS wordt nu door Lotus 1-2-3 van schijf gehaald en in het geheugen gezet. In 1-2-3-seconden ziet u het op uw scherm staan.

WAT IS EEN RANGE-ZONE?

Een Range-zone is een rechthoek van cellen. Dit kan zelfs 1 cel, een kolom, een rij of een gedeelte van een kolom of rij zijn.

Het opgeven van een range-zone kan op drie verschillende manieren.

- Door de hoekpunten van de celadressen in te geven. Dit is de typing-methode.
- Door de celcursor naar de cellen te verplaatsen, die voor de range-zone in aanmerking komen. Dit is de pointing-methode.
- Door een zone-naam (die van te voren is benoemd) te gebruiken.

Om nu een range-zone te kunnen maken, doet u het volgende:
type in: / (slash)
R (range)

Nu verschijnt het sub-menu van de Range, dat er als volgt uit ziet:

Format	Label	Erase	Name	Justify
Protect	Unprotect	Input	Value	Transpose

U kunt nu uit 9 verschillende mogelijkheden kiezen. Stel, u wilt de labels in het spreadsheet centreren. Hiervoor kiest u dan LABEL. Met deze keuze is het mogelijk om labels Links, Centraal of Rechts in een cel te plaatsen. Laten we het maar eens proberen.
Type in: L (label)

Nu verschijnt er in de statusbalk:
 Left Center Right
 Type in: C (center)
 Nu staat er op de statusbalk de vraag, om een range-zone aan te wijzen. Wij doen dit nu volgens de typing-methode.

Type in : A3..A5 <R>

Op dit moment zult u zien dat het woord Banden eerst begint met een spatie. De woorden Spiegels en Uitlaten zijn even groot als de celbreedte en daarom ziet u hier geen verschil.

OPDRACHT 1.

Ga naar cel A20

Type in: aap <pijl neer>
 noot<pijl neer>
 uw <R>

Centreer A20 t/m A22 op de manier zoals hierboven beschreven.

Nu ziet u een duidelijk verschil. Kijk wat de effecten zijn, als u dit doet met Left en Right.

Om hetzelfde effect te krijgen, maar dan met de pointing-methode, moet u het volgende doen:

druk op <F5> type in:

B20 <R>
 jan <pijl neer>
 Joop <pijl neer>
 ik <R>
 / (slash)
 R (range)
 L (label)
 C (center)
 <pijl op>
 <pijl op>
 <R>

U ziet nu dat het effect gelijk is aan de typing-methode. Op deze manier kunt u alle andere mogelijkheden in het range-menu gebruiken. Probeer er maar eens een paar uit. Wees niet bang om fouten te maken.

De Naming-methode wordt aan de hand van onderstaande voorbeelden besproken. Met deze functie van Lotus 1-2-3- is het mogelijk om aan een cel of een blok cellen een naam te geven.

Wanneer u nu met de cursor naar cel B7 gaat, dan ziet u daar de formule =SUM(B3..B5) staan.

In plaats van deze formule, kan er ook voor deze formule een zinnige naam worden gekozen. De uitdrukking die deze formule geeft, is het totaal van filiaal 1. Deze naam zouden we dus ook aan cel B7 kunnen geven, maar omdat deze nogal lang is, korten we die af tot:

TOTFILL1.

Dit gaat als volgt: Type in:

/ (slash)
 R (range)
 N (name)
 C (create)

TOTFILL1 <R>

In het midden van de statusbalk verschijnt nu:

Enter range:B7..B7

Omdat wij alleen cel B7 de naam TOTFILL1 willen geven, drukken we nu op <R>. Vanaf nu heet cel B7....TOTFILL1.

OPDRACHT 2

Geef de totalen van filiaal2 en filiaal3 de range-naam TOTFIL2 en TOTFIL3, de prijzen van de filialen 1 t/m 3 de namen PRIJSFIL1, PRIJSFIL2 en PRIJSFIL3

Kijk nu eens in de cellen B7, C7 en D7. Ziet u, wat er met de =SUM-functie is gebeurd? Probeer bovenstaande opdracht uit volgens de typing- en pointing-methode.

Nog drie belangrijke mogelijkheden in het range-commando-menu zijn:

Protect, Unprotect en Input.

Met het PROTECT-commando kunnen cellen tegen ongewild wissen, wijzigen of verplaatsen worden beschermd. Het UNPROTECT-commando werkt tegenovergesteld aan het PROTECT-commando. Hiermee kunnen de cellen die worden beschermd, weer worden vrij gegeven. Unprotect werkt het prettigst, wanneer het hele spreadsheet met het commando PROTECT is beschermd. U geeft dan met het commando UNPROTECT aan, welke cel(len) u vrij wilt geven.

VOORBEELD

Zorg dat u in de READY-mode staat

Type in: / (slash)
 W (worksheet)
 G (global)

P (protection)
 E (Enable)

Nu is het hele spreadsheet beschermd tegen wissen, wijzigen enz., enz.. Ga met <F5> naar cel C3
 Type in: / (slash)
 R (range)
 U (unprotect)
 <pijl neer>
 <pijl neer>
 <R>

De cellen C3 t/m C5 zijn nu niet meer beschermd en deze kunnen dus worden gewijzigd, gewist en verplaatst.

Het Input-commando is handig, wanneer u veel gegevens in een bepaalde zone moet invoeren. U kunt een invoerzone als volgt maken:

Ga naar cel C3.

Type in: / (slash)
 R (range)
 I (input)
 <pijl neer>
 <pijl neer>
 <R>

Cel C3 staat nu linksboven in het scherm. U kunt nu door gebruik te maken van de pijltjes-toetsen omhoog en omlaag. Wanneer u nu bij de laatste cel in de zone bent, dan gaat de cursor automatisch weer naar de eerste cel in de zone. In ons geval is dat cel C3. Pas wanneer u op <R> of <ESC> drukt, werkt het INPUT-commando niet meer.

Probeer dit maar eens uit.

OPDRACHT 3.

Probeer alle andere commando's in het Range-menu eens uit.

Bewaar het spreadsheet op schijf onder de naam WINKELS2

Type in: / (slash)
 F (file)
 S (save)
 WINKELS2 <R>

Dit was het voor deze keer

Robert Vroegop.
 Den Haag

tel: 070-29 94 28

(vrijdagavond: 20.00-22.00 uur)

Gegevensopslag op de HARDE SCHIJF, deel 6

Dikwijls zitten de meesten van ons met het probleem dat bij gebruik van een harddisk of hardcard, langzamerhand een chaos gaat ontstaan. Netjes gestart met indelen blijkt later toch niet datgene te zijn, wat we er van hadden verwacht.

Vragen als "Waar zit ook alweer dat programma" en "Op welke manier kom ik bij dat en dat programma" en natuurlijk de vraag "Is het niet mogelijk mijn programma's wat sneller en eenvoudiger te benaderen en te gebruiken".

Voorgaande vragen zullen ongetwijfeld voorkomen, zeker in het begin als er nog niet voldoende ervaring met

de PC is opgebouwd.

Hopelijk kan het volgende artikel enige aanwijzingen geven, zodat u sneller en gemakkelijker van de op de harddisk/card aanwezige programmatuur en files gebruik kunt maken.

Natuurlijk zijn er volop programma's te verkrijgen, waarmee enige structuur in de aanwezige software is aan te brengen.

Denk b.v. maar eens aan de Fixed Disk Organizer van IBM. Dit programma voorziet in een menu-gestuurde "boomstructuur", waarmee u op een gebruikersvriendelijke wijze dat programma bereikt, dat u wilt gebruiken. Stel dat die structuur er zo

zou uitzien:

>>> Hoofdmenu <<<

1. DOS
2. Tekstverwerking
3. Communicatie
4. Kaartenbak

Bij het ingeven van een cijfer of de beginletter van de toepassing, komt u in de volgende laag van de menu's. Stel, u had een 2 ingetikt, dan was u terecht gekomen bij het menu:

>>> Tekstverwerking <<<

1. Tasword
2. PC-Write
3. Volkswriter
4. DW4
5. Wordstar
6. WordPerfect

Met het opnieuw maken van een keuze, belanden we in één van de 6 tekstverwerkingsprogramma's. Stel, dat we 4 hadden gekozen, dan waren we bij het eind van de werkzaamheden uit Wordstar, automatisch weer in het Tekstverwerkings-menu terecht gekomen. Via commando's kunnen we dan weer naar het Hoofdmenu teruggaan.

om naar een ander toepassingsgebied te gaan, of de PC te verlaten. Dit hoofdmenu en de daarbij behorende "sub menu's", kunt u naar eigen inzicht en behoeftie opzetten. Men kan stellen dat de Fixed Disk Organizer een makkelijk te leren programma is en eenvoudig is toe te passen op hard-disk c.q. hardcard. Echter, voor hobbyisten die nogal wat "rommelen" met allerlei software, is dit programma minder goed toepasbaar. Het feit dat men steeds bij aanpassen van directories en software dit programma moet verla-

ten, is hieraan debet. Het is wel mogelijk om e.e.a. vanuit dit programma te doen, maar dat is omslachtig. Helpschermen zelf aanmaken en integreren in dit programma, is zonder meer mogelijk.

Fixed Disk Organizer is een van de vele manieren om structuur aan te brengen op de harddisk/card. Een andere wijze van werken, die erg aantrekkelijk is en die ons door DOS wordt geboden, is het werken met batch-files. De volgende keer zullen we zien op welke wijze dat gaat.

Dick Bruggemans

... en er was beeld!

Naast allerlei kleine verschilletjes is er één groot verschil tussen de computers uit de PC-stal (IBM-achtig) en de rest: de PC's zijn met een aantal verschillende video-kaarten en monitoren te krijgen, terwijl de andere computers gewoon met één standaard-scherm worden geleverd. Deze diversiteit in de PC-wereld is de oorzaak van een hoop problemen en het lijkt dus de moeite waard er een artikelje aan te wijden.

In den beginne was er de Monochrome Display Adaptor. Deze video-kaart voor de PC is in staat om, op een simpele manier, tekst op het beeldscherm te toveren. Het scherm wordt onderverdeeld in 25 regels van 80 kolommen elk en op elke plaats kun je één of andere letter kwijt. Elke letter krijgt nog wat extra spullen mee, zoals onderstreept, "vet" (eigenlijk helderder), knipperend en geverteerd (zwart op wit). Naast de zesentwintig letters van het alfabet in onder- en bovenkast, aangevuld met de normale cijfers en leestekens, is de tekenverzameling voorzien van een aantal gekke, handige en bijzondere tekentjes zoals de kader-letters". En IBM zag dat het goed was.

Met de MDA of monochroomkaart kun je op een uitstekende en goedkope manier met tekst en eenvoudige gestileerde plaatjes werken. Je kunt daarbij volstaan met een eenvoudige (TTL) monitor, waarop de verschillende tekens in een heel duidelijke vorm verschijnen. De MDA maakt namelijk gebruik van een matrix van negen bij veertien puntjes, om de tekens op te bouwen. Op een normale, toch vrij kleine, monitor geeft dit een fraai gedetailleerd beeld van uitstekende kwaliteit.

De MDA had en heeft echter een paar nadelen. Allereerst zijn er natuurlijk alleen maar de standaard-tekenen mee te maken: elk teken is weliswaar opgebouwd uit 9*14 puntjes, maar de computer vertelt alleen maar aan de MDA dat er een "A" op het scherm moet komen, de MDA doet de rest zelfstandig. De losse puntjes kunnen dus niet afzonderlijk worden aangestuurd en echt mooie tekeningen kun je dus

gewoon niet maken. Teletekst-achtige plaatjes vormen het uiterst haalbare. Verder kent de MDA (de naam zegt het al) géén kleuren: alles is in dezelfde monitor-tint.

Om deze leemte op te vangen zette IBM een andere kaart in elkaar: dat werd de Color Graphics Adaptor. Aan de naam is al te zien dat deze kaart precies biedt, wat ervan werd gevraagd: kleuren en grafische mogelijkheden.

De CGA werkt met een scherm van maximaal 320 bij 200 punten in 16 kleuren of 640 bij 200 punten in 4 kleuren. Voor niet al te moeilijke plaatjes is dat best geschikt. Wil je er echter tekst op kwijt, dan kom je in de verdrukking. Tel maar uit: tekens worden door de CGA opgebouwd uit 8 bij 8 puntjes. Vergelijk dat eens met de 9*14 van de MDA! Het resultaat is er dan ook naar: ik duid de CGA altijd aan als de hoofdpijn-kaart. In de tijd dat deze ontwikkelingen plaatsvonden stonden er nog alleen PC's bij bedrijven en op grote kantoren, want de apparatuur was nog maar net te betalen. Op deze grote kostenpost maakte een extra videokaart en een extra monitor niet veel uit. Daarom zag je in die tijd overal waar een CGA en een kleurenmonitor stond óf een MDA met een monochrome monitor. De één voor het grafische- en kleurenwerk, de andere voor "platte" tekst. Een bijna ideale oplossing, die wel veel geld en ruimte kostte.

Toen de PC's echter gemeengoed werden en ook in hobby- en huiskamers verschenen, veranderde er iets. Het werd nu kiezen tussen óf een MDA met monochrome monitor óf een CGA met kleurenmonitor. Beiden werd te duur. Vaak waren zowel de MDA als de CGA alvast in de computer ingebouwd en hoefté je alleen maar de monitor te kiezen. Voor heel veel mensen is de balans naar monochroom doorgeslagen en zij kunnen dus uitstekend en zonder hoofdpijn typen, maar missen alle grafische plaatjes en kleuren. Zowat alle spelletjes vertikken het dus. De kleurenkijkers zijn - wat dat betreft - beter bedeeld, maar gaan wel elke avond met hoofdpijn

naar bed.

Een slimme fabrikant vond hier - althans gedeeltelijk - wat op: hij maakte een aangepaste MDA waarbij je wél ieder puntje afzonderlijk kon aansturen. Op die manier krijs je dus 80 kolommen * 9 puntjes = 720 puntjes horizontaal en 25 regels * 14 puntjes = 350 puntjes verticaal. Dat is heel wat beter dan de CGA, maar natuurlijk niet in kleur. Toch was deze oplossing z fraai dat de hele wereld er op vloog: de Hercules-kaart was geboren. Op nagenoeg elke tegenwoordige MDA zit deze Hercules-stand, zodat je op vrijwel iedere computer heel goed grafisch kunt werken. Alleen de echte IBM's moeten het zonder doen, want IBM zag en ziet niets in Hercules.

Op elke PC zit dus nu een grafische werkstand: óf een monochrome Hercules (720*350) óf een kleuren-CGA (320/640*200). Eigenlijk is elk fatsoenlijk programma wat een grafisch scherm nodig heeft, wel geschikt voor allebei. Persoonlijk geef ik dan altijd de voorkeur aan Hercules, want de slechte resolutie en gemene kleuren van de CGA wegen niet op tegen het rustige en gedetailleerde Hercules-beeld.

Toch bleef er een gat zitten, want het was nog steeds niet mogelijk om op een kleurenbeeldscherm fatsoenlijke letters neer te zetten, en kleuren kunnen - mits goed toegepast - heel veel aan een programma toevoegen. Daarom kropen de ingenieurs van IBM opnieuw achter de tekentafel en het resultaat was de Enhanced Graphics Adaptor. Deze kaart levert een beeld van 640 bij 350 punten, bijna net zoveel als de Hercules-kaart dus, maar dan wel in kleur (16 uit een "palet" van 64). Een simpel rekensommetje leert dan dat de letters en cijfers van deze kaart zijn opgebouwd uit 8*14 puntjes. In de praktijk is het verschil met de MDA en/of Hercules-kaart dan ook te verwaarlozen. De CGA wordt echter compleet uitgelaichen.

We stuiten nu echter op een probleem: de monitor. Het verschil tussen een MDA en een Hercules-kaart is nihil,

want er komen precies evenveel punten en lijnen uit. De monitor zal het dus worst wezen of er een MDA of een Hercules-kaart wordt aangehangen. De CGA-monitor hoeft heel wat minder te presteren, maar die wil een kleursignaal zien en hij is dus toch niet aan een monochroomkaart te hangen. Tot zover geen problemen. De EGA vraagt echter om een kleurenmonitor met een veel hogere punt-dichtheid, zodat er in elk geval een duurdere uitvoering moet worden gekocht. Schakel je over van CGA naar EGA, dan kun je je oude CGA-monitor dus in de kast zetten. Van MDA/Hercules naar EGA gaat natuurlijk al helemaal niet zonder de monitor te vervangen.

Met een EGA kun je eigenlijk alle andere kaarten nadoen. Er zit vaak een MDA/Hercules-compatible stand op, zodat je alle monochrome programmatuur kunt laten lopen, de kaart kan natuurlijk de oude CGA nadoen en je kunt hem in de echte EGA-stand zetten. Op deze manier is vrijwel elk programma aan de gang te krijgen. Je kunt ervan uitgaan dat elk groter pakket Hercules, CGA èn EGA ondersteunt, zodat je met een EGA alle kanten uitkunt.

Ik geloof te mogen stellen, dat EGA op dit ogenblik van de uitgebreidere grafische kleurenkaarten de meest gebruikte is. Bij mijn weten worden hogere resoluties door nog maar weinig software ondersteund en zonder software is zelfs de beste kaart waardeloos. Er zijn echter toch kapers op de kust.

Vooraan in de rij staat VGA te dringen: het Video Graphics Array. Deze kaart lijkt veel op EGA, maar kan weer meer punten en meer kleuren aan: bijvoorbeeld 800*600 en maximaal 256 kleuren. Er is natuurlijk ook weer een (nogal veel) duurder model monitor nodig, terwijl de kaart zelf ook kostbaar is. De prijzen zakken echter flink en ik denk dat binnen een jaar VGA de EGA aardig zal hebben verdrongen (althans bij de nieuwe apparatuur in de duurdere sector).

In de goedkopere (nou nou) modellen van de IBM PS/2-lijn zit geen VGA maar een MCGA, een Multi Color Graphics Array. Dat is een versimpelde VGA met minder kleuren maar een even grote resolutie. Gezien de prijs van PS/2's, niet direct iets om je druk over te maken.

Het post-VGA-tijdperk kan ik nog niet goed overzien. Natuurlijk is ook VGA niet goed genoeg en worden er kaarten op de markt gegooid, die wér meer punten in wér meer kleuren kunnen toveren. Wanneer je dan echter geen duidelijk grotere monitoren gaat gebruiken (20 inch of zo), dan gooij je een hoop geld weg. VGA kan al z veel puntjes op een scherm zetten dat je ze op een kleine 14-inch monitor bijna niet meer uit elkaar kunt houden. Dat levert een heel aangenaam beeld op, maar het verschil tussen dichtgelopen letters en superdichtgelopen letters is nul. Daarvoor betaal ik geen cent. Meer kleuren tegelijk op het scherm is echter wel

leuk. Je kunt dan in plaats van computer-plaatjes, ook fatsoenlijke echte TV-achtige beelden vertonen.

Samenvattend ben ik dus van mening dat voor de huidige aspirant-PC-koper er twee kaart-monitor-combinaties overblijven: een Hercules-kaart met een monochrome monitor (groen, amber of wit) en een EGA met een goede (dure) kleurenmonitor. Het gebied er tussenin - de CGA - kan mij op geen enkele manier overtuigen. Voor een Hercules-kaart en een monochrome monitor ben je ongeveer vierhonderd gulden kwijt, en vaak zit deze prijs al in de standaardprijs voor de computer. Een EGA kost ongeveer vierhonderdvijftig gulden, de bijbehorende monitor zeker duizend. Bij elkaar levert dat vijftienhonderd gulden op, oftewel een meerprijs - bovenop de standaard Hercules-uitvoering - van minstens duizend gulden. Val me niet aan op de exacte prijzen, je kunt vast wel ergens een speciaal adresje vinden. Ik wil hier alleen een schets geven van de kosten die het kiezen voor kleur met zich meebrengt. Weeg dus goed af wát je met de PC wilt gaan doen. Is de kleur die extra duizend gulden waard of niet? Dat hangt helemaal van jouw toepassingen af.

Laat je echter niet verleiden tot de middenweg: die is niet van goud, maar van spaanplaat. Je krijgt er hoofdpijn van.

Jeroen Hoppenbrouwers

Abstractie: de kern van de zaak.

Een tijdje geleden heb ik een vluchtig overzicht gegeven van de meest gebruikte programmeertalen en hun geschiedenis. Ik wilde nu eens wat dieper op de zaak ingaan.

99% van de huidige computers is nog steeds gebaseerd op de ideeën van John von Neumann uit de veertiger jaren. Ze hebben een geheugen en één processor. Deze processor haalt zowel zijn opdrachten - het programma - als zijn gegevens uit het geheugen en plaatst de resultaten van alle bewerkingen ook weer in het geheugen terug. Het gevolg is, dat geheugens (en geheugencellen) een centrale rol spelen bij deze computers. Voorlopig ziet het er niet naar uit dat andere niet-von-Neumann-computers een rol van betekenis gaan spelen in onze directe omgeving. Daarom beperk ik me hier tot de von Neumann-machines.

Zoals machinetaal-freaks wel zullen weten, zien (von Neumann) computers hun geheugen als een lange rij aaneengesloten cellen, waarbij elke cel een uniek nummer - het adres - heeft. Om dus gegevens uit zo'n cel te halen of in zo'n cel te stoppen, moet je het precieze adres weten.

Ook bij sprongen in het programma is het adres uiterst belangrijk: het is de enige manier om exact aan te geven, waar je heen wilt. Bovendien zijn sprongen (zeg maar GOTO's) de enige manier om je programma te sturen.

Voor programma's van een beetje lengte is het werken met vaste adressen voor gegevens (data) en programma (code), al gauw ondoenlijk. Toch moet en zal de processor deze informatie hebben, daar is het ding tenslotte op gemaakt. Dan maar geen grote programma's? Ook niet aanlokkelijk.

In de jaren vijftig bereikte de computertechniek het peil, waarop de programma's voor directe machine-taalprogrammering, te groot werden. Wat nu? De von Neumann-architectuur vaarwel zeggen? Maar dat zou betekenen, dat er een geheel nieuw type computer zou moeten worden gemaakt en dat is de wetenschap nog steeds niet echt gelukt. Een slimme geest kwam toen op het idee om de computer zelf te gebruiken, om de kloof tussen menselijke programmeur en (toen nog net) buizen-processor

te overbruggen. De assembler was uitgevonden.

Een assemblerprogramma biedt een aanzet tot een brug over de kloof. Door een soort schijnbare machine te maken die, in plaats van absolute cijfers, leesbare woorden slikt, kun je een flink stuk verder gaan met het maken van je programma. De adressen verdwijnen namelijk: je vervangt ze door woorden die iets zeggen. Met een assembler kun je op een wat hoger, abstracter niveau gaan werken. Abstract omdat de processor het niet meer begrijpt: je programmeert de computer niet meer, je programmeert een machine die niet bestaat, maar zelf ook geprogrammeerd is. Je maakt gebruik van een virtuele, abstracte machine die een paar stenen van de brug vormt. De rest van de brug moet je nog zelf bouwen. Vandaar dat assemblerprogrammeren (dat is eigenlijk wat wij nu "machinetaal" noemen) moeilijk blijft.

Het is erg belangrijk dat je goed doorhebt, wat ik hier met abstractie bedoel. Abstractie is het gebruikmaken van dingen die niet bestaan omdat

dat makkelijker is, dan het gebruiken van de dingen die wél bestaan. Mensen leven bij gratis van hun vermogen tot abstractie. Een voorbeeld: ons alfabet en onze taal. Aan tamelijk willekeurige tekentjes en klanken hebben we betekenissen toegekend. We werken dan verder met die betekenissen en niet met de tekentjes en klanken! Ik zit nu niet één voor één de letters uit te zoeken, ik weet wat ik wil zeggen en de schrijfautomaat in me zorgt er wel voor dat de juiste woorden en, binnen die woorden, de juiste letters achter elkaar worden gezet. Ik abstraheer van letters via woorden naar betekenissen. Wanneer je dit leest, doe je dat ook: je leest niet de letters, maar hoogstens de woorden en waarschijnlijk de zinnen. En wanneer ik je over een week vraag wat in dit artikel stond, dan gebruik je niet eens de zinnen of de alinea's, maar de informatie die uit de tekst als geheel komt.

Abstractie in de computerwereld wil zeggen: je maakt gebruik van dingen, afspraken, die nergens in de computer zijn terug te vinden, maar die het leven enorm vergemakkelijken.

Eén zo'n abstract object is bijvoorbeeld een variabele. Een variabele is een abstractie van één of meer geheugencellen. Je werkt niet met losse cellen en met losse adressen, je gebruikt een zelfgekozen naam en de abstracte machine doet de rest. BASIC is een prachtig voorbeeld van zo'n virtuele machine: je hebt niets te maken met geheugen of adressen, je zegt gewoon "TELLER wordt 7" en de virtuele machine doet de rest. Variabelen vormen een hoofdkenmerk van de zogenaamde imperatieve talen, de computertalen die rechtsstreeks zijn gemodelleerd naar de von Neumann-architectuur.

Naast deze beperkte data-abstractie kennen programmeertalen meestal ook code-abstractie. Met code-abstractie bedoel ik bijvoorbeeld, het verdoezen van de von Neumann-GOTO-opdrachten. De processor moet deze sprongen nog steeds uitvoeren, maar we gebruiken ze niet meer in de programmeertekst omdat dit veel prettiger werkt. BASIC doet heel weinig aan code-abstractie. GOTO zit er bijvoorbeeld nog gewoon in. Je kunt weliswaar een stuk programma vervangen door één GOSUB-opdracht, maar het mechanisme is z beperkt dat het alleen voordelen biedt in programmeerlengte, namelijk wanneer je hetzelfde stukje programma op meer plaatsen nodig hebt. Dat is geen abstractie! De functiedefinities van BASIC zijn ook geen echte abstracties; het zijn meer macro-substituties van expressies. BASIC is afgeleid van FORTRAN, de eerste programmeertaal ter wereld. FORTRAN was speciaal geschreven voor de IBM 704-computer en vertoont daardoor trekjes die heel sterk aan de hardware van deze computer doen denken. Het is dus niet zo'n fraai voorbeeld van data- en code-abstractie. Daarom

zitten ook in BASIC zo verschrikkelijk weinig mogelijkheden tot abstractie. Echte code-abstractie vindt je pas vanaf ALGOL-60, een paar jaar na FORTRAN. Pascal, een ALGOL-opvolger, kent dus wél code-abstractie in de vorm van procedures, echte functies en bepaalde controlestructuren. Nieuwere BASIC-dialecten zoals Turbo BASIC kennen nu ook code-abstractie, zodat deze talen wat bruikbaarder zijn geworden.

Wanneer mensen praten over gestructureerd programmeren lijkt het wel of er alleen maar aan code-abstractie wordt gedacht. Structuur, dat wil kennelijk zeggen, stikken in de GOSUB's of procedures. Joe, vooruit. Maar de andere helft van het verhaal, de data-abstractie, die wordt gewoon vergeten. Vandaar dat ik altijd kribbig wordt, wanneer iemand een aangepaste BASIC blijft gebruiken omdat die gestructureerd programmeren toelaat: je kunt procedures maken! Hoi hoi! Niks hoi hoi, je bent nu nog maar op de helft, want data-abstractie is heel wat meer dan alleen een variabele met een naam.

Deze grote vergissing is niet echt vreemd, want ook de computerwetenschappers, die dit allemaal uitdenken, hebben er lang overheen gekeken. Data-abstractie boven het niveau van variabelen begint namelijk pas interessant te worden, wanneer programma's wér een stuk groeien boven de drempel van code-abstractie. Anders gezegd: bij het minste of geringste heb je al code-abstractie nodig, maar pas bij zware klussen moet je ook data-abstractie hebben. Wat is nu een zware klus? Let op. Dat is een programma van meer dan een paar pagina's. Ja, ook uw BASIC-programma's zijn dus te groot, om zonder data-abstractie te kunnen. Waar merkt u dat aan? Aan de troep die het wordt, aan de urenlangen foutzoeksessies, aan de brij van variabelen en de slakkengang waarmee u en uw computer voortworstelen.

BASIC kent dus geen code-abstractie én geen data-abstractie van betekenis. Drom is BASIC zo'n verschrikkelijk beroerde taal! Drom lijkt het programmeren in BASIC best wel op programmeren in machinetaal! Drom zijn grote programma's in BASIC zo zeldzaam (afgezien van machinebeperkingen - ook op grote computers draaien geen grote BASIC-programma's). En Turbo/Quick BASIC dan? Ja, die kennen wat code-abstractie. Maar nog steeds helemaal geen data-abstractie! En dat geldt voor alle mij bekende opgepepte BASIC's. Dus alle talen waar BASIC in de naam zit: weg ermee. Ze zijn gewoon niet geschikt voor de huidige normen in programmeren. Je vliegt ook niet meer in een tweedekker naar New York. Deze talen zijn bovendien niet makkelijker te leren dan de andere talen, of beter, en al helemaal niet sneller. Ze zijn gewoon stokoud, onbruikbaar en uit de tijd. Punt.

Welke talen kennen dan wel data-abstractie? Simpel: alle talen behalve FORTRAN en BASIC. Zelfs een "machine"taal als de macro-assembler van de PC (MASM) kent een klont data-abstractie. En dat is dan nog maar machinetaal. Pascal, C, Ada, Modula-2, FORTH, SNOBOL, Oberon, Turing, Green, Euler, APL, PL/I, BCPL, ALGOL, Smalltalk, noem maar op. Allemaal kennen ze data- en code-abstractie. Natuurlijk in verschillende mate. Je kunt stellen: hoe nieuwere een taal is, des te meer abstractie zit er in. PROLOG en LISP zijn zelfs één grote abstracte wereld.

Pascal zal voor de meesten onder ons het dichtste bij zijn, op de voet gevuld door C en Modula-2. Alle drie kennen ze de basis-code-abstractie (procedures en codeblokken) en de basis-data-abstractie (variabelen, types, records, strings, files). Modula-2 gaat nog verder en kent zelfs echte abstracte datatypen. Dat zijn zelfgemaakte variabelen met zelfgemaakte operatoren die erop werken. Alleen met die operatoren kun je eraan komen. Op deze manier is het abstracte datatype gelijkgesteld met de abstracte code, de procedure. Een procedure heeft een naam, een type (namelijk procedure) en parameters. Een abstract datatype heeft ook een naam, een type (namelijk zichzelf) en parameters in de vorm van operatoren.

Een voorbeeld. Niet origineel, wel duidelijk.

We willen een stack maken in ons programma, bijvoorbeeld om wat rekenwerk te vergemakkelijken. Daarvoor zouden we in BASIC zonder nadenken een array STACK(100) declareren en een variabele SP die de top van de stapel bijhoudt. Maar dat heeft als nadeel dat overal in ons programma die "stack" illegaal kan worden gebruikt. Wanneer er een opdracht STACK(8) = 54 staat, protesteert de machine niet en ook SP = 67 kan zonder waarschuwing. Bovendien moet je heel nadrukkelijk onthouden of SP nou naar boven of naar beneden groeit en nooit vergeten om - na het afhalen of ophangen van een getal - SP bij te stellen. Natuurlijk is dit niet zo'n probleem wanneer je in je eentje en alleen binnen dit programma werkt, want dan weet je het wel. Maar wanneer je gebruikt maakt van procedurebibliotheiken of wanneer meerdere mensen aan hetzelfde programma werken, dan gaan deze onvolkomenheden zwaar meetellen.

Het is een teken aan de wand dat zelfs de simpelste processor van zijn stack al een half-abstract datatype gemaakt heeft. Geen zinnig mens zal de opdrachten PUSH en POP zelf gaan maken uit bijvoorbeeld EX SP, HL en DEC SP-opdrachten. Die abstracte stack zorgt daar zelf wel voor, daar maak je gebruik van! In BASIC moet je toch weer alles zelf doen... In bijvoorbeeld Modula-2 kun je zo'n

stack nog veel beter maken. Je maakt eerst een module, waarin je de stack declareert. Dat kan als array zijn, of als linked list (sliert), of als een stapel losse pointers op de heap. Verder maak je een stack pointer. Tot zover geen verschil. Maar, dan maak je een procedure PUSH en een procedure POP om de stack te bewerken. En (nu komt het) alleen deze twee procedures maak je buiten de module bekend. Het gevolg is aller eerst dat niemand meer je stack kan verknoeien, ook al doen ze nog zo hun best, want met alleen PUSH en POP kun je geen gemene dingen doen. Je kunt dus ook niet per ongeluk je eigen programma in de war schoppen. Verder laat deze constructie ook achteraf wijzigingen van de implementatie toe: of je de stack nu maakt van een array of van een sliert, dat maakt voor het gebruikende programma niets uit, want het ziet alleen PUSH en POP. PUSH en POP moeten natuurlijk wel precies weten hoe het zit, maar die zitten bij de stack in een afgesloten module en daar kun je dus aan prutsen zonder dat de rest dat merkt.

Ada, de taal die het moet gaan maken in de komende vijf jaar, gaat nog veel verder dan Modula-2. Met Ada kun je werkelijk alles abstraheren. Maar Ada is zo geweldig groot dat er tot nu toe nog geen compiler is geschreven, die de hele taal aan kan. Modula-2 daarentegen is een kleine taal die toch de hoofdzaken van abstractie bevat. Voor Modula-2 zijn er dan ook al compilers die echt goed zijn. Maar we kijken natuurlijk uit naar Turbo Modula-2, de compiler van Borland, die al een tijdje onderweg is. Die zal vast weer hetzelfde concept bevatten als Turbo Pascal en Turbo C, en dan is mijn keuze al gemaakt.

Een veel gehoorde vraag over abstractie is: "Maar duurt het nou allemaal niet veel langer? Moet die computer niet veel meer nadenken, voordat hij weet wat ik wil? Kan ik dat niet beter zelf doen?". Het antwoord is: het maakt geen klap uit, het wordt er vaak zelfs sneller door. Hoe kan dat?

Als we nu even de standaard-BASIC-interpreter vergeten, dan worden programma's in alle programmeertalen, van assembler tot Ada, voor gebruik even door een vertaalprogramma getrokken. Tijdens dit vertalen (of compileren), bekijkt het vertaalprogramma - de compiler - met argusogen het programma, om zo domme fouten op te sporen. Eén van de dingen waaraan je een moderne programmeertaal herkent, is nu de hoeveelheid domme fouten die zijn compiler kan vinden.

FORTRAN-compilers - de eersten ter wereld dus - hadden en hebben erg weinig mogelijkheden om domme fouten, zoals niet-passende parameters bij subroutines, te ontdekken. Het gaat dan natuurlijk gewoon fout tijdens

het RUNnen van het programma, maar dat ziet de computer niet meer. Het gevolg is dat er erg lastige fouten optreden, die moeilijk zijn te lokaliseren. Een taal als Pascal is juist gemaakt om dergelijke domme fouten op te sporen. Een gegarandeerd verkeerde constructie komt gewoon niet door de compiler heen.

Hoewel Pascal doorgaat voor een "harde" taal die erg strikt is op types en structuren, zijn er toch wel degelijk nog hardere talen. Ada en Modula-2 bijvoorbeeld. Een Ada-compiler is in staat het kleinste onregelmatigheidje feilloos uit het programma te vissen en weigert dan ook prompt verdere dienst. Eerst alle domme fouten eruit, dan pas krijg je de kans het programma te zien lopen. Wat een verschil met BASIC...

Een heel erg groot deel van de hierboven besproken code- en dataabstractie is nu gebouwd rond deze foutencontrole van de taal. Daarom ziet een programma in Pascal, Modula-2 of Ada er ook zo uitgebreid uit: je kunt en moet alles volledig ondubbelzinnig uitspellen, zodat de compiler 100% zeker weet wat je bedoelt. Alle losse blokjes die je maakt, worden uit-den-treure getest op consistentie en aansluitbaarheid. Pas wanneer alle mogelijke controles zijn gepasseerd, wordt er een echte vertaling naar machinetaal geproduceerd. Maar... omdat de vertaler nu weet dat er geen domme fouten meer in het programma zitten, kunnen er heel veel supersnelle oplossingen in machinetaal worden gemaakt. En op dit niveau is de smerigste machinetaaltruc volledig legaal: die truc wordt door een machine in elkaar gezet en is dus foutloos.

Het komt er dus op neer dat tijdens het schrijven van het programma, de programmeur heel erg netjes moet werken om maar zoveel mogelijk informatie op "papier" te zetten. Staan al deze gegevens keurig in de programmeertekst, dan gaat de vertaler er overheen, die werkelijk elk stukje informatie uitpluist en gebruikt. Het allergrootste deel van de abstracties in je programma worden zo, al tijdens de vertaling, omgezet naar reële dingen als geheugenplaatsen en adressen. Doordat de vertaler nu bovendien heel erg veel weet, is hij in staat door de doolhof van mogelijke machinetaalopdrachten, de kortste en snelste weg te zoeken. Het resultaat is de droom van elke programmeur: een fantastisch duidelijk programma waaraan je heel simpel kunt sleutelen en toch een erg compact en supersnel stuk machinetaal als het klaar is.

Helaas, zover zijn we nog niet. Zelfgeschreven machinetaal is altijd nog sneller dan door een machine geschreven code. Maar in 95% van de gevallen loont deze snelheidswinst niet en telt zwaarder mee dat het programma veel sneller geschreven is en veel

betrouwbaarder werkt, omdat alle domme fouten er al uit zijn.

Als laatste stukje van dit artikel wil ik het nog even hebben over de niet-imperatieve talen. Deze talen trekken zich niets aan van de von Neumann-architectuur van de computers en staan veel dichter bij de mens. Zo zijn er logische talen (PROLOG bijvoorbeeld) en functionele talen (LISP). De vertalers voor deze talen staan dus voor een enorme klus, want met deze talen beschrijf je niet HOE een computer iets precies moet doen, maar alleen WAT hij moet doen. Je schrijft geen exacte sorteerprocedure, maar je zegt dat de uitvoerrijz moet zijn geordend, dat overal geldt, dat van twee naast elkaar liggende elementen de kleinste links ligt. Hoe de vertaler daar een von Neumann-programma van bakt is zijn zaak. Het gevolg is, dat deze talen wel degelijk inefficiënt zijn, want zo'n machine is nogal wat dommer dan wijzelf, althans op dit moment nog. Maar er komt een tijd dat de vertalers zo goed en de computers zo snel zijn, dat deze aanpak succes heeft. Vooral de toepassing van parallele processoren lijkt hoopgevend. Met imperatieve talen is parallel programmeren heel erg vervelend, maar met een logische taal als PROLOG gaat het bijna vanzelf. Het vak van programmeur zal dan aanzienlijk veranderen.

Graag zou ik weten, wat u van dit soort artikelen vindt. Het kost nogal wat moeite, om de laatste ontwikkelingen op het gebied van de (theoretische) informatica in een vorm te gieten, die ook voor een groter publiek verterbaar is. Daarom weet ik, bij elkezin die ik schrijf, meteen drie uitzonderingen op te noemen en blijf ik dubben of ik het allemaal niet te simpel voorstel. Daarom zou ik graag wat reacties horen.

Alvast bedankt!

Jeroen Hoppenbrouwers
Wilhelminapark 8
5554 JE VALKENSWAARD
04902 - 1 38 08

OPROEP

Afdeling Apeldoorn 1.0

Wil een ieder uit de wijde omtrek rond Apeldoorn, die geinteresseerd is in regelmatige computerbijeenkomsten, zo spoedig mogelijk contact opnemen met:

ROELAND VAN ZEYST
APELDOORN
Tel: 055 - 21.30.13

Karin's Column.

Was dat schrikken met Tron 27, wat een rommeltje. Je kon wel zien dat ie in de hete dagen van mei was gemaakt. Maar ach het zijn ook maar mensen zullen we maar denken.

PC Software

Goed nieuws voor de PC mensen, JW heeft het niet alleen bij fronsen gelaten, daardoor staat er nu ook PC software in Simp. Het aller leukste is dat hij daardoor Jos heeft laten inzien dat het onzin is, om hele programma's te gaan maken in machinetaal, daar het in Turbo-Pascal veel sneller en makkelijker gaat. Alleen is het nu tussen Dick en Jos net Feyenoord-Ajax, want Dick zweert bij Turbo-Basic en Jos nu dus bij Turbo-Pascal.

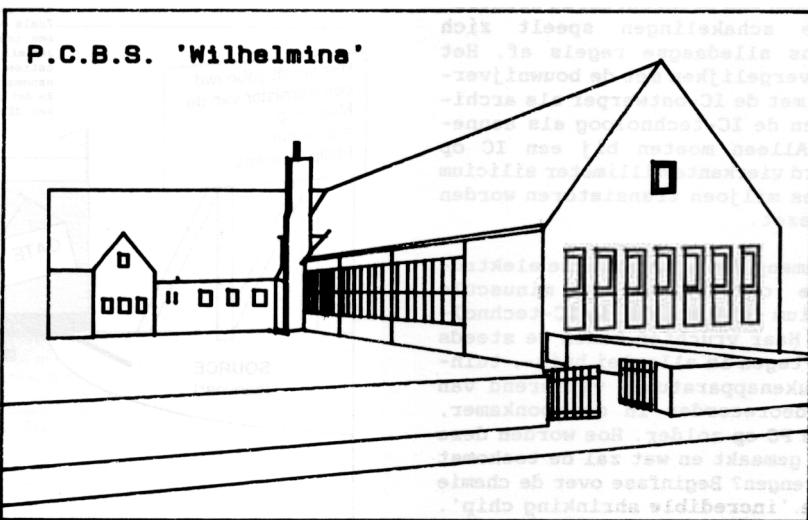
Plotter

Op een mooie zonnige dag deed de plotter zijn intrede in huize Simpel tel. Wat dit inhoudt, snapt u zeker wel. Dit betekent voor mij sloten koffie zetten, want als zoets eenmaal bekend is, wil iedereen komen kijken. Nu is het natuurlijk een mooi gezicht om zo'n apparaat aan het werk te zien, maar om nu helemaal uit je bol te gaan omdat ie 8 print sporen onder een Z80 kan trekken, vind ik toch wel overdreven. Goed met de hand lukt dat nooit en ook plakken zou best een probleem geven. maar daarvoor neem je dan toch ook een plotter?

Neem nou Piet Zeelenberg. Hij vond het wel een duur speeltje, maar toen Jos liet zien dat je er ook hele mooie teksten mee kan maken, zag je hem warm lopen. Piet gaat namelijk na 34 jaar leraar en directeur te zijn geweest, zijn school verlaten door in de VUT te gaan. Nu geeft hij ter afscheid aan zijn leraren en leerlingen een boek. Daar moet dan wel iets ter herinnering in komen. Als Jos dat nu zou kunnen plotten met verschillende kleuren zou dat natuurlijk wel erg mooi zijn. Dat was klaar. Je kunt nu zes stukjes tekst per A3 vel doen en elk stukje een andere naam geven.

Digitaliseren

Toen gingen ze nog wat experimenteren en wat bleek; je kunt er ook mee digitaliseren. Je legt een foto op de plotter, je geeft aan waar de pen Up of Down moet en laat de plotter die gegevens naar de computer sturen. Daarna kan de foto geplot worden. Nu was het hek van de dam. Piet werd nog enthousiaster, want als Jos nu een foto van zijn school zou doen, dan zou deze dus ook op de uitnodiging voor de receptie kunnen worden geplot. Na dagen ploeteren is dit gelukt. Maar je kunt het ook nog verkleinen. Piet vroeg of het ook zo klein zou kunnen dat het op een sticker past. Ik voelde al nattigheid. "Natuurlijk" zegt Jos en laat



het zien. Nu kunnen we dus ook stickers voor Piet gaan plotten, want het is toch leuk als die kinderen ook een sticker van hun eigen school in dat boek hebben? Maar daar hebben we natuurlijk geen plotter voor. Het is een leuke bijkomstigheid, maar hij is bedoeld om printjes te plotten. Nou dat doet ie schitterend. Geen geplak meer, maar hup op de plotter, dan een film er van maken en dan een printje. Ho, dat was een grote misrekening. Omdat het op gecoat papier wordt geplot, is het contrast te weinig (misschien dat het wel kan, als je met rotringpennen werkt, maar die hebben we niet). Jos heeft van alles geprobeerd, meer tijd, 7x over elkaar plotten, maar nee..... geen film. Toen gewoon voor de lol het papier op de print gelegd. Resultaat een schitterende print. Zo dus, dat scheelt weer werk en geld. Je hoeft nu geen film meer te maken of te kopen.

Adventure

Voor de adventures freacks heb ik niet veel. SPQ3 is namelijk erg makkelijk. Er zitten maar 2 moeilijke stukken in. Het eerste is dat je de "grabber" moet gebruiken om dat ding op te takelen en in het schip zetten. Het tweede is een beetje gemeen. Als er iets van je wordt gejat, moet je niet meteen resetten maar het gewoon weer op gaan halen. Als je met het spel begint denk je "ha 738 punten dat wordt een lange". Nou vergeet het maar en het einde is net zo slecht als bij Larry2. Je hoeft niets meer te doen. Jammer hoor. Er zijn natuurlijk best wel andere spellen die ook leuk zijn, zoals Mah-Jong, of Wammy en Sokoban. Dat zijn spellen waar je even lekker voor gaat zitten en op je gemak gaat uitpuzzelen, hoe je dat voor elkaar moet krijgen.

RTTY op de PC

Een telefoontje uit Cuyk: "Zeg weet je dat het RTTY-kastje gewoon op de PC kan?" En daar zaten we nu op te wachten, want de P2000 staat nu in de andere hoek van de kamer en konden we niet meer RTTYen. Maar nu dus wel. We kregen ook een Public Domain programma, maar dat stond iedere keer op tilt zodat Jos er maar meteen een is gaan maken. Wij 's avonds meteen op Beijing (persbureau) afgestemd, om daar de laatste berichten uit China op te vangen. Ben ik toch wel blij dat ik in Nederland woon.

Simpeltel is jarig

Terwijl ik dit zit te schrijven, gaat de bel. Oh mensen wat leuk! Zie ik 2 bloemstukken en een grote gebaksdoos met daarachter de gezichten van Ap, Peter en Jeroen. Want Simp. bestaat vandaag 4 jaar. Wat een verrassing. Loop ik langs Simp. wordt er net een plaatjestartaart op het prikbord gezet door Andor. Dit is dus het einde van mijn verhaal want ik ga nu koffie drinken en taart eten. Tot de volgende keer.

Karin



CHEMIE OP DE VIERKANTE MICROMETER: hoe bouw ik een IC?

Het maken van geïntegreerde elektronische schakelingen speelt zich volgens alledaagse regels af. Het is te vergelijken met de bouwnijverheid, met de IC-ontwerper als architect en de IC-technoloog als aannemer. Alleen moeten bij een IC op honderd vierkante millimeter silicium wel zes miljoen transistoren worden neergezet.

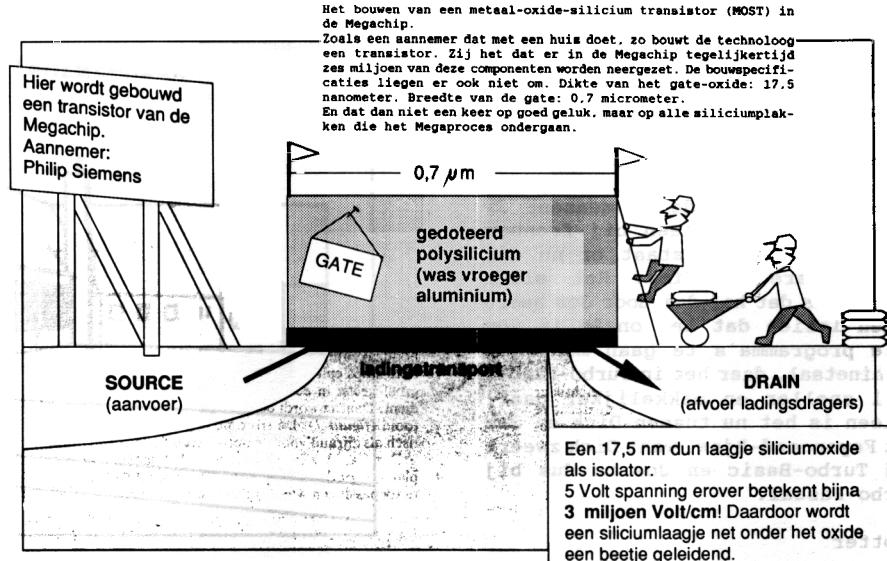
Het samenpakken van diverse elektronische componenten op minuscuile silicium 'chips', dat is IC-technologie. Haar vruchten komen we steeds vaker tegen in allerlei huis-, tuin- en keukenapparatuur, varierend van de videorecorder in de woonkamer, tot de PC op zolder. Hoe worden deze chips gemaakt en wat zal de toekomst ons brengen? Beginfase over de chemie van de 'incredible shrinking chip'.

Het jaar 1959 had 3 belangrijke verrassingen voor mij in petto. Ten eerste zag ik het levenslicht. Ten tweede werd het eerste patent ingediend voor het tegelijkertijd op een siliciumplak aanbrengen van verschillende elektronische componenten, inclusief de bijbehorende bedrading. Tenslotte presenteerden de heren Atalla en Khang 's werelds eerste METAAL-OXIDE-SILICIUM-TRANSISTOR, kortweg MOST genoemd. Anno 1989 verdien ik mijn dagelijks brood op het Philips NatLab en zie met eigen ogen hoe we op 100 vierkante millimeter silicium zes miljoen van deze MOSTen neerzetten. Een miljoen bits informatie kunnen in deze Megachip elektronisch worden opgeslagen en op willekeurige wijze worden uitgelezen.

Voor het hoe en waarom van technologie om (geïntegreerde) elektronische schakelingen (IC's) te maken, kunnen we overigens goed terecht in een wat bekendere branche en wel die van de bouwnijverheid. Dit vanwege het naakte feit, dat ook de 'high-tech' zich in feite volgens alledaagse regels afspeelt.

Als een architect bedenkt de IC-ontwerper een zodanige combinatie van transistoren, weerstanden, capaciteiten en wat dies meer zij, dat de gehele schakeling (op paier tenminste) voldoet.

Een belangrijke bouwsteen is de al eerder genoemde MOST (zie figuur). Hij bestaat uit metaal, siliciumoxide en de halfgeleider silicium zelf. Het metaal is van het silicium gescheiden door het oxide. Al naar gelang de spanning die op de metaalpoort ('gate' genaamd) wordt gezet, kan er onder het oxide een elektrische stroom door het silicium stromen. Aan- en afvoer van de daarvoor benodigde ladingsdragers gebeurt in principe via de zogeheten 'source' en 'drain'. In principe is het nu de taak van de IC-technoloog de gewenste componenten ook werkelijk in het silicium te realiseren - inclusief de benodigde bedrading. (Wat dit betreft zou je hem of haar



dus kunnen vergelijken met de aannemer annex bouwkundige in de woningbouw.) Anno 1989 is dit een behoorlijk zware klus geworden. Niet alleen gaan de ontwikkelingen razendsnel, maar wie ze wil volgen, dient ook steeds beter in het Japans uit de voeten te kunnen.

Ik laat de bouwtechnieken maar eens de revue passeren, ondergebracht in drie categorieën: het afbeelden van patronen, het aanbrengen van dunne lagen en het etsen van (delen van) deze lagen.

De eerste is die van LITHOGRAFIE. Met een masker wordt een patroon op de siliciumplak aangebracht, waarop van tevoren fotogevoelige lak is aangebracht. Dit maakt het mogelijk lokaal lak te verwijderen en dus lokaal op de plak bewerkingen uit te voeren. De maskers (of beter gezegd, de maskerbeschrijvingen) zijn als het ware bouwtekeningen van het circuit, van elke verdieping is er een. Voor een beetje IC zijn toch al gauw vijf verschillende maskers nodig, die na elkaar op dezelfde plaats moeten worden aangebracht. Voor het Megaproces zijn veel af te beelden patronen aanzienlijk kleiner dan 1 micron, vandaar de naam SUBMICRONTECHNOLOGIE. Een andere techniek is die van het DEPONEREN of GROEIEN van dunne lagen, isolerend of juist geleidend. Het technologisch vocabulaire omvat hier sputteren, low-pressure chemical vapour deposition (LPCVD), epitaxie, en oxidatie. Tallos is het aantal processen waarmee bijvoorbeeld het di-elektrische siliciumoxide gemaakt kan worden. Een van de oudste processen is thermische oxidatie door de plak in een oven aan zuurstof bloot te stellen (zeg bij 100 gr. C). Beroemd is het Nat.lab.patent dat het lokaal oxideren van silicium - dankzij een masker uiteraard - beschrijft. Zo kunnen de verschillende componenten gemakkelijk van elkaar geïsoleerd worden.

Door in een gasmengsel van silaan en lachgas (N_2O) een plasma op te

wekken, kun je ook bij 400 gr C een siliciumoxide-achtige substantie op de plak laten neerslaan. Dergelijke lage temperaturen zijn nodig als ergens Aluminium in de structuur is ingebouwd.

Resteert het ETSEN. Vroeger was dit het exclusieve speelterrein van de 'natte' chemicus, maar ook hier rukken de plasma's op.

Het grote voordeel van het etsen in een plasma is, dat je in een specifieke richting kunt etsen. Dat is absoluut nodig als je bijvoorbeeld een reeks gaatjes wilt etsen die niet alleen een micron diep zijn, maar die ook nog op elke diepte in het gat niet meer en niet minder dan 1 micron van elkaar af mogen liggen. De meeste natte etsmiddelen zouden in zo'n geval slechts een groot gat opleveren. Uiteindelijk, na vele honderden van deze processtappen - en veel opnieuw beginnen als er weer iets fout is gegaan - is de siliciumplak klaar. In elke plak zijn meerder, veelal identieke chips gebakken. Die kunnen nu losgezaagd en veilig verpakt worden - maar niet nadat ze op de goede manier aan een frame met metalen pootjes zijn gelijmd. Via die pootjes verloopt namelijk de communicatie met de buitenwereld, bijvoorbeeld de toetsen van een pocket calculator.

Resteert de vraag waarom het toch allemaal zo klein moet. Die is voor een deel simpel te beantwoorden. Niet alleen moet een pocket calculator ook werkelijk in een binnenzak passen, maar ook wil de leverancier geld verdienen. Als je tweemaal zoveel chips op een plak kunt zetten, zal de prijs per stuk lager kunnen; immers in principe blijft het aantal processtappen gelijk. Tel uit je winst. En leg meteen een deel opzij om een aantal slimme chemici in te huren: het kan namelijk vast nog kleiner.

Michael Nieuwesteeg

(Met toestemming overgenomen uit CHEMISCH MAGAZINE, mei 1989)