15.1 De architectuur

Het hart van een computer is de centrale verwerkingseenheid (CVE). Ook wordt vaak de Engelse afkorting CPU (Central Processing Unit) gebruikt. De CVE voert de taken uit die door het besturingssysteem worden opgedragen.

Turbo Pascal is, evenals de programmeertalen C en BASIC, een procedure-gerichte taal. Bij procedurele talen bestaat een commando uit een aantal commando's voor de CVE. Tijdens het compileren zet de compiler de commando's van de programmeertaal om in codes die de CVE begrijpt. Als je een programma schrijft in Turbo Pascal, of in een andere procedure-gerichte taal, dan werk je indirect met de CVE. Turbo Pascal biedt een aantal faciliteiten om rechtstreeks met de CVE te werken. Om deze faciliteiten te kunnen gebruiken, moet je een enigszins kennis hebben van de diverse componenten van de machine en hun onderlinge verhoudingen.

In het hoofdstuk over dynamische variabelen is het geheugen al gedeeltelijk aan de orde gekomen. Met name het gegevenssegment, de stack en de heap hebben hier aandacht gekregen. Samen met het codesegment en het videogeheugen vormen deze segmenten het zogenaamde RAM. Het woord RAM is een afkorting van Random Access Memory, ofwel een willekeurig toegankelijk geheugen. In dit geheugen kunnen gegevens worden geschreven en worden uitgelezen. Het RAM is vluchtig van aard. Als de computer uitgezet wordt of als de stroom uitvalt, dan zijn de gegevens die in het RAM stonden in principe verdwenen.

Naast het RAM heeft je computer ook nog het ROM. Dit is een afkorting van Read Only Memory. Dit is een geheugen waaruit alleen gelezen kan worden en waar niet in geschreven kan worden. Het ROM bestaat uit een groot aantal programma's die bestemd zijn om je systeem te laten functioneren. Programma's die in het ROM staan, zijn als het ware in je machine ingebakken. Turbo Pascal heeft faciliteiten om deze programma's direct aan te spreken.

De CVE, het hart van je machine, beheert het geheugen. De maximale hoeveelheid beschikbaar geheugen die je machine heeft, wordt bepaald door het type CVE en het besturingssysteem. De oorspronkelijke IBM PC, die uitgerust was met de Intel 8088 microprocessor, kon 1.048.576 bytes adresseren. Dit is gelijk aan één megabyte, ofwel 1024 * 1024 bytes. De nieuwste Intel-processoren, waarvan de Pentium de laatste is, kunnen maar liefst 4.294.967.296 bytes adresseren.

De nieuwste processoren kunnen echter, vanwege hun compatibiliteit, nog steeds met het geheugenmodel van de eerste PC werken. In dat geval spreken we van werken in de "real mode". Vereisen de werkzaamheden meer dan één megabyte, dan spreken we van werken in de "protected mode".

De Turbo Pascal-compiler maakt programma's in de real mode. Als je programma's wilt maken die in de protected mode werken, dan kun je de Borland Pascal-compiler gebruiken. Voor wat betreft het gebruik van de taal is er nauwelijks verschil tussen Turbo Pascal en Borland Pascal.

De CVE is met het geheugen verbonden door middel van zogenaamde bussen. Een bus is een transportmiddel voor gegevens en heeft aan het begin een ingang en aan het einde een uitgang voor de doorvoer van gegevens. De gegevens in een bus kunnen ook onderweg afgetapt worden. Een bus is vergelijkbaar met een stroomleiding in huis. De wisselstroom van 220 volt komt je huis binnen en kan op zijn weg door de leidingen afgetapt worden door een stekker in het stopcontact te steken. Je computer beschikt over de volgende bussen:

- Adresbus voor het transport van adressen;
- Databus voor het transport van gegevens;
- Besturingsbus voor de besturing van het systeem.

Je kunt de bussen voorstellen als naast elkaar lopende draden waarover bits verstuurd worden. Acht draden naast elkaar kunnen de acht bits van een byte versturen. De adresbus van de oorspronkelijke XT is twintig bits breed. Een adres kan dus nooit groter zijn dan twintig bits. Vandaar dat een XT maximaal één megabyte kan adresseren. Dat is namelijk het maximum dat twintig bits kan bevatten. De nieuwere CVE-typen hebben een bredere adresbus en kunnen dus met meer geheugen werken.

De databus van de oorspronkelijke PC was acht bits breed. Bij de oorspronkelijke PC kon dus telkens maar één byte op de bus gezet worden. Ook hier geldt dat nieuwere processoren beschikken over bredere bussen van zestien en zelfs tweeëndertig bits. Dit betekent weer dat er twee of vier bytes tegelijkertijd verzonden worden in plaats van één. Je kunt je voorstellen dat de breedte van de databus een belangrijke factor is voor de snelheid van je machine.

Terug naar de organisatie van het geheugen. We nemen de geheugenorganisatie in de real mode als uitgangspunt. De adresbus is twintig bits breed en kan maximaal één megabyte adresseren. Dit zijn de adressen 0 tot en met 1.048.576. Hexadecimaal zijn dit de adressen 0 tot en met FFFFF. In hoofdstuk 3 hebben we gezien dat één hexadecimaal cijfer uit vier bits bestaat. Vijf hexadecimale cijfers gebruiken dus twintig bits.

Het geheugen wordt verdeeld in zestien blokken. Dit zijn de blokken hex 0 tot en met hex F. Deze indeling in blokken kun je je als volgt voorstellen:

Bloknummer:	Bestemming geheugen:
0 tot en met 9	640 Kb gebruikersgeheugen

A tot en met B	Videogeheugen
C tot en met D	Adaptors en speciale toepassingen
E tot en met F	ROM-BIOS

De 640 Kb geheugen voor de gebruiker is bij Turbo Pascal verdeeld in een codesegment, een datasegment, een stacksegment en een extra segment (de heap). Het geheugen vanaf blok A wordt "upper area" genoemd.

Gegevens over het waar de segmenten beginnen, houdt de CVE bij in zijn registers. Een register kan zestien bits bevatten. In een register passen dus de waarden hex 0 tot en met hex FFFF. De registers hebben ook namen:

- **CS = Codesegment**
- **DS = Datasegment**
- **SS = Stacksegment**
- **ES = Extra segment**

Als de waarden in de registers gebruikt zouden worden om het geheugen te adresseren, zouden we, om een absoluut adres te krijgen van de zestien bits in een register, een twintig bits adres moeten maken. Dit kunnen we doen door het register met zestien te vermenigvuldigen. Als in CS de segmentwaarde 9A11 staat, zou deze vermenigvuldiging de hexadecimale waarde 9A110 opleveren.

Omdat een dergelijke vermenigvuldiging altijd op 0 uitkomt, zouden we op deze manier steeds zestien bytes tegelijk adresseren. Dat kan echter niet de bedoeling zijn, want we willen iedere byte in het geheugen terug kunnen vinden. Daarom wordt er binnen ons systeem gewerkt met gesegmenteerde adressen. Een gesegmenteerd adres bestaat uit een segmentdeel en een offsetdeel. Beide delen worden door het systeem samengevoegd tot een absoluut adres in het geheugen binnen het bereik hex 0 tot en met hex FFFFF.

Hoe werkt dit nu precies? De segmentwaarde 9A11 uit het voorbeeld wordt vermenigvuldigd met 16. De uitkomst hiervan is hex 9A110. Als we een offsetwaarde van hex 1234 hebben, dan wordt die erbij opgeteld. Zo ontstaat het absolute adres hex 9B344. Het volledige adres wordt geschreven als 9A11:1234. Het deel voor de dubbele punt is het segment, het deel erachter is de offset. Deze manier van rekenen heeft het voordeel dat je 64 kilobyte kunt adresseren zonder het segment te wijzigen.

Voor de uitvoering van een programma zijn de registers heel belangrijk. Deze bevatten namelijk informatie over segmenten en offsets, maar er zijn ook registers die waarden bevatten voor de rekeneenheid. Daarnaast is er een register met statusvlaggen:

Rekenregisters:

Accumulator:	AH:	AX:	AL:
Basis	BH	BX	BL
Teller	CH	CX	CL

Data	DH	DX	DL	

Stackregisters:

Stackpointer:	SP:	
Basepointer	BP	
Source index	SI	
Destination index	DX	

Adressenregisters:

Codesegment:	CS:	
Datasegment	DS	
Stacksegment	SS	
Extra segment	ES	
Instructie-	IP:	
pointer		

Vlaggen:

[ ] [ ] [ ] [ ] [OF] [DF] [IF] [TF] [SF] [ZF] [ ] [AF] [ ] [PF] [ ] [CF]

De eerste groep registers worden ook wel de rekenregisters genoemd. Hier worden getallen in geplaatst die een bewerking moeten ondergaan. De registers bevatten twee bytes. Bij deze registers is het mogelijk om de bytes afzonderlijk te benaderen. AH is de hoge byte van het register AX en AL is de lage byte van het register AX.

Het BP-register bevat de offset van de top van de stack. Het volledige adres is dus
SS:SP. Het BP-register wijst de geldige offset van de stack aan.

De derde groep bevat de 16-bits adressen van het codesegment, het gegevenssegment, het stacksegment en de heap. Als tijdens een programma-uitvoering een instructie uitgevoerd wordt, dan staat in het IP-register het adres van de volgende opdracht.

Ook voor de vlaggen is een 16-bits register beschikbaar. Iedere vlag gebruikt hierin één bit. De vlaggen duiden een situatie aan. Als de uitkomst van een bewerking 0 is, gaat de vlag ZF omhoog. ZF staat voor "Zero Flag".

15.2 Assembler

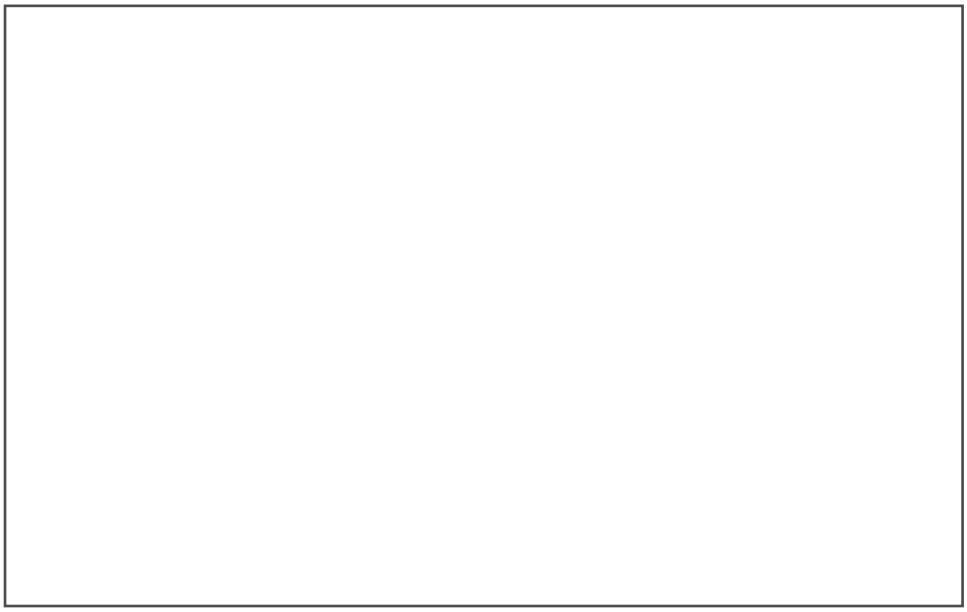
De CVE voert een programma uit door regel voor regel de instructies op te halen uit het codesegment. De CVE haalt de gegevens uit het geheugen, bewerkt ze en zet ze terug. Wat de CVE ermee moet doen, staat in de instructie in de trant van: "Haal twee bytes van dat adres en zet ze in het register. Deel het getal door 2, en zet de 2 bytes terug op het oude adres".

Omdat de CVE uitsluitend met nullen en enen werkt, krijgt hij ook zijn instructies als nullen en enen aangeboden. We kunnen deze instructies met behulp van het programma DEBUG zichtbaar maken in de vorm van hexadecimale getallen. DEBUG wordt meegeleverd bij DOS. Het laat ons tijdens de programma-uitvoering in de registers kijken. Ook kan het een dump van het geheugen maken:

- Start het programma DEBUG vanaf de DOS-prompt. Je krijgt nu een streepje als cursor te zien. Toets nu in:

  n sort_2.exe <R>

- Als je nu de letter "l" intikt en vervolgens op Enter drukt, wordt het programma geladen. Tik vervolgens de letter "d" in. Je krijgt nu een overzicht van een aantal regels machinecode. Als je daarna de letter "r" intikt, kun je de inhoud van de registers zien. Laat deze opdracht volgen door het ingeven van de letter "u" en een druk op Enter. De registers en een soort opdrachtenlijst worden nu getoond:



Afbeelding 14

We zien nu het begin van het programma SORT_2.EXE, dat we in het vorige hoofdstuk gemaakt hebben. De hexadecimale getallen die na het intoetsen van de letter "d" verschenen zijn, zijn de door de compiler gemaakte hexadecimale instructies voor de CVE.

In principe zou je een programma met hexadecimale getallen op deze manier kunnen schrijven. Dit zal echter wel een enorm monnikenwerk worden, waarin heel gemakkelijk fouten kunnen worden gemaakt. Het is mogelijk om dit soort numerieke instructies op te nemen in een Turbo Pascal-programma. Hiervoor wordt de Turbo Pascal-statement Inline gebruikt.

Het is leuk om even naar de inhoud van de registers te kijken. Je ziet dat de inhoud van het register CS samen met de inhoud van IP het adres vormt van de eerste uit te voeren opdracht. Deze opdracht zie je ook onder de uitdraai van de registers staan.

De DEBUG-opdracht "u" levert een lijst van min of meer leesbare instructies. Deze instructies zijn een vertaling van de lijst numerieke instructies die we met het intoetsen van de letter "d" kregen. Instructies als "CALL", "PUSH" en "MOV" kunnen we lezen en een betekenis geven. Met "CALL" wordt een routine aangeroepen, met "PUSH" wordt er iets op de stack gezet en met "MOV" wordt met een waarde geschoven. Deze geschreven opdrachten worden "mnemonics" genoemd. Een mnemonic is een woord dat je gebruikt in plaats van een moeilijk te onthouden code, in dit geval de numerieke instructie voor de CVE. Mnemonics kunnen we gebruiken om te programmeren in assembler, ofwel machinetaal.

Normaal gesproken wordt een programma in assembler-taal verwerkt in een assembler. Dit is een programma dat de mnemonics vertaald naar een numerieke code. Een assembler levert een bestand af met de extensie .OBJ. Dit soort bestanden kunnen dan weer met behulp van een zogenaamde "linker" aan elkaar geplakt worden, met als resultaat een uitvoerbaar programma met de extensie .EXE.

In een Turbo Pascal-programma kunnen bestanden met de extensie .OBJ zondermeer ingelezen worden. Je kunt ook gewoon overschakelen van het gebruik van Turbo Pascal naar het gebruik van assembler.

Assembler wordt vooral gebruikt als je iets wilt doen in je programma, wat niet eenvoudig met de bestaande Turbo Pascal-commando’s kan worden gedaan, of omdat met assembler een sneller resultaat bereikt kan worden omdat het de machine directer aanspreekt.

Hieronder volgt een voorbeeld van de manier waarop assembler in een Turbo Pascal-programma wordt opgenomen. De functie Hoofdletters, die we al eerder gemaakt hebben met behulp van de Turbo Pascal-functie Upcase, wordt nu in assembler weergegeven. Deze functie is, met een paar kleine veranderingen, geleend uit de Turbo Pascal gebruikersgids van Borland:

PROGRAM MACH_1;

```
FUNCTION Hoofdletters(Woord:String):String;
BEGIN
  ASM
     CLD
     LEA SI,Woord
     LES DI,@Result
     SEGSS LODSB
     STOSB
     XOR  AH,AH
     XCHG AX,CX
     JCXZ @EINDE
     @BEGIN_LUS:
     SEGSS LODSB
     CMP AL,'a'
     JB @GEEN_KLEINE_LETTER
     CMP AL,'z'
     JA @GEEN_KLEINE_LETTER
     SUB AL,20h
     @GEEN_KLEINE_LETTER:
     STOSB
     LOOP @BEGIN_LUS
     @EINDE:
  END
END;

BEGIN
  Write(Hoofdletters('test'));
  Readln
END.
```

Regels:Toelichting:

[1]2 Declareer een functie Hoofdletters met een string Woord als parameter, die als resultaat een string afgeeft.

```
   [2]4-24Geef het blok aan waarin machinetaal gebruikt wordt.
   [3]5        Zet de richtingvlag op 0.
   [4]6        Zet de offset van de parameter Woord in SI.
   [5]7   Zet in ES het adres waar het resultaat van  de  functie
          terecht moet komen.
   [6]8   Zet de byte die op het  adres  SS:SI  staat  in  AL  en
          verhoog SI met 1.
   [7]9   Zet de byte die in AL staat op het adres dat aangegeven
          wordt door DI en verhoog DI met 1.
   [8]10       Zet AH op 0.
   [9]11       Verwissel de waarde in AX en CX.
   [10]12      Als CX de waarde 0 heeft, ga dan naar EINDE.
   [11]13      Zet het label BEGIN_LUS.
   [12]14      Zet de byte SS:SI in AL en verhoog SI.
   [13]15-18Als de waarde in AL kleiner is dan de waarde  van  de
          letter "a" of groter dan de letter  "z",  ga  dan  naar
          GEEN_KLEINE_LETTER.
   [14]19      Trek het hexadecimale getal 20 af van de waarde in
AL.
   [15]20 Zet de waarde in AL op het adres dat  aangegeven  wordt
          door DI en verhoog DI met 1.
   [16]22 Verminder de waarde in CX met 1. Als de  waarde  in  CX
          groter is dan 0, ga dan terug naar het label BEGIN_LUS.
  23      Zet het label EINDE.
   [17]27      Zet  de  string  "test"  in  hoofdletters  op  het
scherm.
```

Toelichting:

[1]De declaratie van een functie gebeurt op dezelfde manier als een Turbo Pascal-functie.

[2]Het woord ASM geeft aan dat we vanaf dat punt overschakelen op assembler. Na de volgende END schakelen we weer over op de Turbo Pascal-code.

[3]CLD staat voor "Clear Direction". Dit betekent dat de richtingvlag op 0 gezet wordt. Als de richtingvlag op 0 staat, worden de registers SI en DI automatisch verhoogd. Staat de richtingvlag op 1, dan worden deze registers verlaagd. SI en DI worden gebruikt als indexregisters voor het lezen van de afzonderlijke letters in de parameter Woord en zetten vervolgens het bewerkte resultaat op een andere plek.

[4]LEA staat voor "Load Effective Address" en laadt het offsetdeel van het adres van Woord in SI.

[5]LES staat voor "Load pointer using ES" en laadt het offsetdeel van het adres waarin het resultaat van de functie geplaatst moet worden. Result is een voorgedefinieerde variabele in Turbo Pascal.

[6]LODSB staat voor "LOaD String Byte" en haalt een byte op, op het adres dat aangegeven wordt door SI. De opdracht SEGSS zorgt ervoor dat het complete adres SS:SI is. LODSB verhoogt hierna automatisch het SI-register.

[7]STOSB staat voor "STOre String Byte" en stuurt een byte uit het AL-register naar het geheugen, op het adres dat aangegeven wordt door DI.

[8]De XOR-operator wordt samen met het register AH gebruikt als dubbele operand. Het resultaat van een dergelijke bewerking is altijd 0. Zoals bekend, kunnen de 16- bits rekenregisters gesplitst worden in twee 8-bits registers. Als AH op 0 gezet wordt, bevat AX alleen nog de waarde AL.

[9]XCHG staat voor "eXCHanGe" en verwisselt de inhoud van de opgegeven registers.

[10]JCXZ staat voor "Jump if CX register is Zero". Als het CX-register op 0 staat, wordt er naar het label EINDE gesprongen. Een label in een ASM-blok wordt altijd voorafgegaan door het apestaartje. Om een label te declareren, wordt het apestaartje gebruikt, gevolgd door een getal en/of letter. Bij de declaratie wordt daar een dubbele punt aan toegevoegd. Hier is sprake van een voorwaardelijke sprongopdracht. Als CX op 0 staat, gaan we naar het label EINDE. We hadden de eerste byte van de parameter Woord ingelezen. De eerste byte van een string in Turbo Pascal is de lengtebyte. Deze lengtebyte is eerst ingelezen in AL en daarna op de plaats gezet waar het resultaat van de functie moet staan. De eerste byte verandert natuurlijk nooit, omdat we de string alleen maar bewerken en niet langer of korter maken. Daarna is de waarde in AL in het CX-register gezet.

[11]Als we niet naar EINDE gesprongen zijn, heeft de parameter Woord een lengte die groter is dan 0. We moeten nu de parameter Woord letter voor letter gaan bekijken. Dit doen we in een lus. Het begin van die lus wordt aangegeven door de declaratie van het label BEGIN_LUS.

[12]Op de reeds besproken manier wordt met LODSB de volgende letter opgehaald en in het AL-register geplaatst.

[13]CMP ("CoMPare") vergelijkt de inhoud van AL met de waarde van de letter "a".

JB ("Jump on Below") springt naar het label GEEN_KLEINE_LETTER als de waarde in het AL-register kleiner dan "a" is.

De opdracht "CMP AL,'z'" vergelijkt AL en "z" en springt naar GEEN_KLEINE_LETTER als de waarde in AL groter is dan "z".

[14]Als we niet naar GEEN_KLEINE_LETTER gesprongen zijn, hebben we dus te maken met een letter in het bereik "a" tot en met "z". De hoofdletters van dit bereik hebben een nummer dat 32 decimaal (= hex 20) lager is. Door van de waarde van de kleine letters hex 20 af te trekken, worden

de kleine letters veranderd in hoofdletters.

[15]We passeren nu het label GEEN_KLEINE_LETTER en STOSB zetten zowel de bewerkte als de niet-bewerkte letter op de plaats die aangegeven wordt door DI.

[16]LOOP verlaagt het CX-register met 1. In het CX-register hadden we de lengte van de string gezet. Als CX op 0 staat, hebben we de hele string nagelopen en kunnen we door naar de volgende opdracht. Zolang dat niet het geval is, wordt er gesprongen naar het opgegeven label. In dit geval het label BEGIN_LUS.

[17]In de Write-opdracht is de functie Hoofdletters als parameter gebruikt. Hoofdletters krijgt op zijn beurt een string als parameter mee. Het resultaat is dat de string in hoofdletters op het scherm verschijnt.

15.3 Nog meer assembler

Er zijn verschillende manieren om machinetaal in een Turbo Pascal-programma op te nemen. Hieronder worden ze samengevat:

INLINE

Een deel van het Turbo Pascal-programma wordt geschreven in de numerieke machinecode. Na het beschermde woord INLINE worden er haakjes geopend waartussen de numerieke codes geplaatst worden.

ASM

Bij deze methode geeft het beschermde woord ASM het begin van een blok aan dat geschreven is in assembler-code. Het blok wordt afgesloten met END. De op deze manier in mnemonics geschreven code maakt gebruik van de ingebouwde assembler. Zeer veel codes die bij een echte assembler zijn toegestaan, kunnen ook gebruikt worden in de ingebouwde assembler. Het declareren van variabelen gebeurt buiten het ASM-blok, alsof het gewone Turbo Pascal-variabelen zijn.

ASSEMBLER

Bij deze methode wordt aan een in Turbo Pascal gedeclareerde functie of procedure het beschermde woord "Assembler" toegevoegd. Dus:

PROCEDURE Naam(parameter);Assembler;

Een op deze manier gedeclareerde procedure start niet met BEGIN, maar met ASM en eindigt met een END. Het op deze manier schrijven van procedures en functies optimaliseert de door de compiler te genereren code. Parameters die groter zijn dan 1, 2 of 4 bytes moeten als VAR-parameters behandeld worden. De compiler wijst geen locatie voor @Result aan, behalve als het om een functie gaat die als resultaat een string heeft. Acht-bits waarden worden in het AL-register geretourneerd, zestien- bits in AX, en tweeëndertig-bits in de combinatie DX:AX. Reals worden in de combinatie DX:BX:AX geretourneerd en pointers in DX:AX. Strings worden gezet op de locatie die aangewezen wordt door @Result.

EXTERNAL

Is een code voor een functie of procedure die verwerkt is door een externe assembler en krijgt als extensie .OBJ:

PROCEDURE Naam(Parameters);External;$L FileNaam.OBJ;

Een dergelijke declaratie leest het bedoelde bestand van de schijf en gebruikt de code voor de procedure. Een .OBJ-bestand gemaakt met C of C++ kan hier natuurlijk ook gebruikt worden.

Het schrijven van programma's of delen van programma's in assembler, vereist een aparte studie. De bedoeling van deze kleine kennismaking is dan ook niet dat je na het lezen van deze paragrafen in staat bent om even snel over te schakelen. Zie het meer als een kennismaking die je tegelijk duidelijk maakt hoe de CVE zijn werk doet. Voor het schrijven van zelfstandige programma's levert Borland een uitstekende assembler, maar ook die van Microsoft is zeer goed.

15.4 Interrupts

De ROM BIOS is een grote verzameling programma's, onontbeerlijk voor het functioneren van je programma. Alle programma's die op je computer uitgevoerd worden, maken gebruik van het ROM BIOS.

ROM BIOS betekent "Read Only Memory Basic Input and Output System". De programma's van het BIOS spreken rechtstreeks de verschillende onderdelen van je systeem aan. In je machine zit een BIOS-chip. Deze chip bevat alle programma's van de BIOS en zijn niet zoals een programma in het RAM-geheugen te verwijderen.

Nu zijn er verschillende BIOS-chips op de markt. Die

verschillende chips bevatten programma's die steeds dezelfde functies hebben. Als dat niet zo zou zijn, zouden programma's niet meer op iedere PC kunnen draaien. De adressen van deze programma's verschillen van chip tot chip. Nu ontstaat er een probleem. Want hoe kan een uit te voeren programma het juiste programma in de BIOS bereiken?

Men heeft dit probleem opgelost door de eerste 256 bytes van het RAM te reserveren voor een tabel. Deze tabel bevat de adressen van de programma's in de BIOS. Een programma in de BIOS wordt bereikt door een zogenaamde interrupt, ofwel een onderbreking van het uit te voeren programma, waarbij gesprongen wordt naar het stukje programma in de BIOS.

Voor het vastleggen van een adres hebben we vier bytes nodig. De tabel kan dus in de gereserveerde 256 kilobyte maximaal de adressen van 64 interrupts kwijt. De interrupts hebben allemaal een nummer en kunnen verschillende serviceroutines bevatten. Het hangt af van de waarde in de registers bij het aanroepen van de interrupt, welke routine wordt uitgevoerd. Stel dat we de cursor een bepaalde afmeting op het scherm willen geven. In dat geval roepen we interrupt hex 10 aan. Bij deze aanroep moet het AH-register de waarde 1 bevatten. In CH wordt dan de positie van de bovenste lijn gezet en in CL de positie van de onderste lijn.

Voor alle duidelijkheid gaan we nog even de gang van zaken na. We willen de cursor vergroten:

- In het AH-register wordt de waarde 1 gezet.
- Het register CH krijgt de waarde 16. In CL zetten we de waarde 0.
- We roepen interrupt hex 10 aan.
- Om het juiste adres uit de tabel te vinden, wordt het interruptnummer met vier vermenigvuldigd. De uitkomst is hex 40.
- We springen naar 0000:0040 en vinden daar het adres van de gezochte serviceroutine.
- De routine wordt uitgevoerd en we keren terug naar het programma.

Nu een toepassing in Turbo Pascal. Het programma MACH_2.PAS roept de interrupt hex 11 aan. Deze interrupt retourneert een 16-bits woord in het AX-register. Dit woord bevat informatie over de computerconfiguratie die je gebruikt. De verschillende bits in het geretourneerde woord hebben alle een eigen betekenis:

Bit:	**Betekenis:**

```
0         1/0 = wel/geen diskdrives geïnstalleerd.
1         1/0 = wel/geen rekenkundige coprocessor (AT).

2-3       RAM op systeemkaart (niet op AT):
             00 = 16 Kb
             01 = 32 Kb
             10 = 48 Kb
             11 = 64 Kb

2         1/0 = wel/geen muis of andere aanwijzer (PS2).
3         Niet in gebruik (PS2).

4-5       videomode bij start machine:
             01 = 40 x 25 kleur
             10 = 80 x 25 kleur
             11 = 80 x 25 monochrome

6-7          00 : 1 diskdrive aanwezig
             01 : 2 diskdrives aanwezig
             10 : 3 diskdrives aanwezig
             11 : 4 diskdrives aanwezig

8         Niet in gebruik.
9-11      Aantal geïnstalleerde seriële poorten.
12        Joystick geïnstalleerd (niet voor PS2).
13        Wel/geen interne modem (niet voor XT en PC).
14-15          Aantal printers.
```

Op adres F000:FFFE staat een byte die aangeeft welk type machine in gebruik is:

Waarde(hex): Betekenis:

```
FF               PC
FE               XT
FC               AT, PS2 model 50 en PS2 model 60
FA               PS2 model 25, model 30
F8               PS2 model 80
```

Om de bits makkelijk te kunnen uitlezen, zetten we eerst in de unit DIVERSEN een functie die dit voor ons doet:

```
FUNCTION Bitwaarde(Patroon:Word;Van,Tot:Byte):Word;
BEGIN
  IF (Van>Tot) OR (Van>15) OR (Tot>15) THEN Bitwaarde:=0
  ELSE
  BEGIN
     Patroon   := Patroon SHL (15-Tot);
```

```
    Patroon   := Patroon SHR (15-Tot+Van);
    Bitwaarde := Patroon
  END
END;
```

De parameter Patroon bevat het bitpatroon waar bepaalde bits uitgefilterd moeten worden. Welke bits dit zijn, staat in Van en Tot. Eerst wordt er gecontroleerd of Van en Tot een geldige waarde hebben, dan wordt het patroon naar links geschoven tot aan de hoogste bit die gefilterd moet worden. Daarna wordt het patroon naar rechts geschoven tot het moment dat de eerste bit die gefilterd moet worden op positie 0 staat. Bitwaarde retourneert de aldus ontstane waarde. Als Van en Tot niet geldig zijn, retourneert Bitwaarde een 0. Maak van de functie Bitwaarde een globale functie in de unit DIVERSEN.

In het volgende programma zijn diverse interrupts opgenomen en wordt de functie Bitwaarde uit de unit DIVERSEN gebruikt:

```
PROGRAM MACH_2;
USES CRT, DOS, DIVERSEN;

VAR
   REG: Registers;
   DISKDRIVES: Boolean;
   VIDEOMODE, AANTAL_DRIVES, SER_POORTEN, PRINTERS: Word;

BEGIN
   Intr($11,REG);
   DISKDRIVES    := Bitwaarde(REG.AX,0,0) = 1;
   VIDEOMODE     := Bitwaarde(REG.AX,4,5);
   AANTAL_DRIVES := Bitwaarde(REG.AX,6,7);
   SER_POORTEN   := Bitwaarde(REG.AX,9,11);
   PRINTERS      := Bitwaarde(REG.AX,14,15);
   TextBackground(0);
   Venster(VX1,VY1,VX2,VY2,3,0);
   GotoXY(4,1);
   IF DISKDRIVES THEN
   BEGIN
      Write('Aantal diskdrives geïnstalleerd: ');
      Write(AANTAL_DRIVES+1)
   END
   ELSE
   Write('Geen diskdrives geïnstalleerd.');
   GotoXY(4,3);
   Write('Scherminstelling: ');
   CASE VIDEOMODE of
      1: Write('25 regels van 40 letters in kleur. ');
      2: Write('25 regels van 80 letters in kleur. ');
      3: Write('25 regels van 80 letters monochrome.')
   END;
   GotoXY(4,5);
   Write('Aantal geïnstalleerde seriële poorten: ');
   Write(SER_POORTEN);
   GotoXY(4,7);
    Write('Aantal  met   het   systeem   verbonden   printers:');
Write(PRINTERS);
   GotoXY(4,9);
   Write('Druk op Enter ');
   Readln;
   TextBackground(0);
   Window(1,1,80,25);
   ClrScr
END.
```

Regels:Toelichting:

[1]3-6Declareer een variabele van het type Registers om te gebruiken bij het aanroepen van een interrupt. Declareer daarnaast variabelen voor de opslag van gegevens.

[2]8 Roep de interrupt hex 11 aan.

[3]9-13Leg met behulp van de functie Bitwaarde de in REG.AX geretourneerde waarde vast in de daartoe bestemde variabelen.

[4]14-16 Maak een venster op het scherm.

[4]17-38 Schrijf de betekenis van de diverse variabelen in het venster.

40-43 Beëindig het programma.

Toelichting:

[1]Om de procedure Intr te kunnen gebruiken, moet in de USES-regel de unit DOS opgenomen worden. Deze unit geeft ook de typedefinitie voor Registers:

```
Registers = Record
  CASE Integer OF
  0: (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags: Word);
  1: (AL, AH, BL, BH, CL, CH, DL, DH: Byte);
END;
```

Zoals je ziet, is het type Registers een free union-record. Je kunt de velden AX tot en met DX ook zien als de velden AL tot en met DH. Een variabele van het type Registers moet als parameter meegegeven worden bij de aanroep van de procedure Intr.

[2]Om het nummer van de interrupt aan te geven, krijgt de procedure Intr de variabele Nummer als parameter mee. Als VAR-parameter wordt een record van het type Registers meegegeven. Bij terugkeer uit de BIOS-serviceroutine, staat in REG.AX de informatie over de configuratie.

[3]We gebruiken de nieuwe functie Bitwaarde om de verschillende bits uit te lezen. De betekenis van deze waarden heb ik hierboven reeds beschreven.

[4]Vervolgens roepen we de procedure Venster in de unit DIVERSEN aan en schrijven we de betekenis van de gevonden waarden naar het scherm.

15.5 Hardware-interrupts

Interrupts worden niet alleen door programma's aangeroepen, maar ook door onderdelen van de machine. Dit zijn hardware-interrupts. Op gezette tijden vragen de diverse onderdelen van de computer aandacht van de centrale verwerkingseenheid. Deze zet de uitvoering van het programma dan even stop, en voert de interrupt uit.

Na terugkeer van de aangeroepen serviceroutine wordt de uitvoering van het programma weer voortgezet. Uiteraard gaat dit alles in een zo korte tijd dat je het als gebruiker van het systeem niet merkt. Nu en dan wordt gekeken of er misschien een toets is ingedrukt, of vraagt een diskdrive om aandacht.

Onderstaand programma laat zien dat er hardware-interrupts kunnen optreden. Het laat tegelijkertijd zien hoe je de interrupttabel kunt beïnvloeden. Het programma maakt gebruik van interrupt hex 1C. Deze interrupt wordt aangeroepen door hardware-interrupt hex 8. Interrupt hex 8 wordt elke 18.2 seconde aangeroepen door de systeemklok. Normaal gesproken bevat de serviceroutine waar interrupt hex 1C naar wijst, alleen maar een terugkeerinstructie. Deze routine kan vervangen worden door een andere:

```
PROGRAM MACH_3;
USES CRT, DOS;
VAR
   OUD_ADRES: Pointer;

PROCEDURE Klokje;Interrupt;
VAR
   UUR, MINUUT, SECONDE, HSEC: Word;

BEGIN
   GetTime(UUR,MINUUT,SECONDE,HSEC);
   GotoXY(5,1);
   Write(UUR:2,':',MINUUT:2,':',SECONDE:2)
END;

BEGIN
   ClrScr;
   GetIntVec($1C,OUD_ADRES);
   SetIntVec($1C,@Klokje);
   Readln;
   SetIntVec($1C,OUD_ADRES)
END.
```

Regels:Toelichting:

[1]2 Gebruik de units CRT en DOS.

[2]3-4Declareer een variabele voor het vastleggen van het adres van de oorspronkelijke serviceroutine.

[3]5-12Procedure Klokje;Interrupt.

6-7 Declareer lokale variabelen voor de opslag van de tijd.

10-11 Schrijf de verkregen tijd in de linker bovenhoek van het scherm.

13-19 **Hoofdprogramma.**

[4]15 Sla het adres uit de interrupttabel, dat bij interrupt hex 1C hoort, op in de variabele OUD_ADRES.

[5]16 Zet het adres van de procedure Klokje in de interrupttabel op de plaats voor interrupt hex 1C.

17 Wacht op een druk op de Enter-toets.

[6]18 Herstel de oude situatie voor interrupt hex 1C.

Toelichting:

[1]We moeten de DOS-unit gebruiken om de procedures GetIntVec, SetIntVec en GetTime te kunnen gebruiken.

[2] De bedoeling is dat we een adres in de interrupttabel gaan veranderen. Het oude adres moet bewaard worden en wordt opgeslagen in OUD_ADRES. De procedure GetIntVec uit de unit DOS zet in de parameter OUD_ADRES het adres waar de interrupt hex 1C nu naar wijst.

[3]De procedure Klokje wordt de nieuwe serviceroutine van interrupt hex 1C. Bij een dergelijke procedure moet achter de declaratie het beschermde woord "Interrupt" toegevoegd worden. De procedure SetIntVec uit de unit DOS plaatst het adres van de procedure in de interrupttabel.

De procedure GetTime wordt gebruikt om de tijd van de systeemklok op te vragen. De verkregen tijd wordt in de linker bovenhoek geschreven. GetTime krijgt UUR, MINUUT, SECONDE en HSEC (honderdsten van seconden) als VAR-parameter mee.

[4]GetIntVec krijgt als parameter de waarde van het interruptnummer mee, en als VAR-parameter een pointervariabele. GetIntVec zet het adres dat in de tabel voor interrupt hex 1C staat, in de variabele OUD_ADRES.

[5]SetIntVec krijgt een interruptnummer en het adres van de nieuwe interruptroutine mee. De nieuwe interruptroutine is de procedure Klokje. Het adres van deze procedure wordt dus meegegeven. De klok van de machine genereert iedere 18.2 seconde een interrupt hex 8. Deze routine werkt de klok bij en roept interrupt hex 1C aan. Op de plaats van interrupt hex 1C staat nu het adres van de procedure Klokje. Klokje wordt nu iedere 18.2 seconde aangeroepen om de tijd op het scherm bij te werken.

[6]Als het programma beëindigd wordt, moet de oude situatie natuurlijk weer hersteld worden, want anders zou de interrupttabel naar een adres wijzen waar zich geen procedure meer bevindt. Met SetIntVec wordt het oorspronkelijke adres weer op zijn plaats gezet. Hiermee is de situatie weer zoals bij het begin van het programma.

15.6 De procedure MS-DOS

De interruptnummers tot hex 10 zijn gereserveerd voor hardware-interrupts. De nummers hex 10 tot en met hex 1F worden gebruikt door de ROM BIOS. De nummers vanaf hex 20 wijzen naar serviceroutines van het besturingssysteem MS-DOS. Veel van deze routines zijn opgeslagen onder interrupt hex 21. Turbo Pascal heeft een procedure MS-DOS, als onderdeel van de unit DOS. De procedure MS-DOS krijgt als VAR-parameter een variabele van het type Registers mee. De aanroep:

```
MS-DOS(Registers);
```

heeft hetzelfde effect als de aanroep:

```
Intr($21,Registers);
```

15.7 Tot slot

Zowel het programmeren in assembler als het werken met de interrupts vergt een aparte studie. Wees vooral met de interrupts erg voorzichtig, want een aantal van deze interrupts voert schrijf- en formatteringsopdrachten naar de schijf uit. Ervaring met assembler en interrupts betekent dat je, daar waar Turbo Pascal geen oplossing biedt, zelf een procedure of functie kunt schrijven. Dit zal echter ook weer niet zo snel voorkomen, omdat Turbo Pascal een zeer complete programmeertaal is.

Bij Uitgeverij Pim Oets zal in 1995 het boek AAN DE SLAG MET ASSEMBLER verschijnen. In dit boek zal uitgebreid worden beschreven op welke manier je programma’s in assembler kunt maken. Ook de documentatie over de opdrachtenset en interrupts zal in dit boek worden opgenomen.

259

259