Hoofdstuk 11

DEBUG ontsluieren

In dit hoofdstuk

- De vele mogelijkheden van DEBUG
- De bewerkmogelijkheden van DEBUG
- De opdrachten van DEBUG
- Programma's bewerken met DEBUG
- Handige utility's met DEBUG maken

DEBUG en videorecorders

Laat één ding duidelijk zijn: DEBUG is een handig programma.

Volgens mij hebben videorecorders en DEBUG een hoop dingen gemeen. Op het eerste gezicht zien beide er ingewikkeld en onhandig uit, maar nadat u de basisprincipes doorhebt, kunt u er makkelijk mee overweg. Net zoals een videorecorder is DEBUG rijk aan geavanceerde mogelijkheden waarvan u de meeste waarschijnlijk nooit zult gebruiken. Dat moet u er echter niet van weerhouden de handige opties wel te benutten.

DEBUG is ontworpen om programmeurs in staat te stellen programmeerfouten uit programma's te halen. Tegenwoordig wordt DEBUG daar nauwelijks voor gebruikt, omdat er veel betere alternatieven voor zijn. Maar DEBUG kan veel meer dan programmafouten opzoeken.

Met DEBUG kunt u kleine veranderingen en verbeteringen aanbrengen in programma's, een werkwijze die vaak *patching* wordt genoemd. Het standaard bestandsmasker *.TXT in het DOS-programma EDIT kan bijvoorbeeld worden veranderd in *.BAT, en u kunt de standaard prompttekens in CHOICE wijzigen.

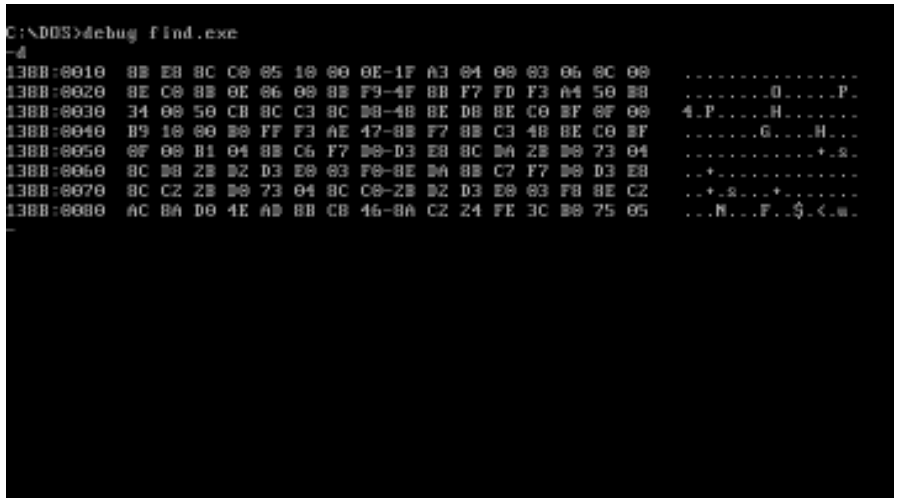
Het is zelfs mogelijk met DEBUG complete programma's te schrijven, maar denk hierbij niet gelijk aan een driedimensionaal spreadsheet. U kunt echter handige en compacte utility's maken, bijvoorbeeld een utility voor het herstarten van de computer, om te controleren of SHIFT wordt ingedrukt gehouden, de dag van de week te bepalen en om de cursor uit te zetten.

Om even terug te komen op de videorecorder: hoe programmeert u dat de videorecorder televisieprogramma's van twee verschillende zenders opneemt?

De bewerkmogelijkheden van DEBUG

EDIT wordt gebruikt om ASCII-bestanden te bewerken, zoals batchbestanden en INI-bestanden. DEBUG is ook een bestandsbewerker, maar dan een bewerker voor binaire programmabestanden, dat wil zeggen bestanden met een extensie COM of EXE. Binaire bestanden zijn niet zoals ASCII-bestanden opgebouwd uit regels; een binair bestand is een homogene klomp bytes. In DEBUG wordt er dus ook niet van de regel uitgegaan, maar de inhoud van een bestand wordt in bytesblokken weergegeven.

Afbeelding 11-1 geeft een voorbeeld van een bestand dat is opgeroepen met de opdracht D (dump) van DEBUG. Het gegevensblok wordt onderverdeeld in drie blokken. In het linker blok staat het gegevensadres (in de notatie segment:offset) van de eerste byte, het middelste



Figuur 11-1

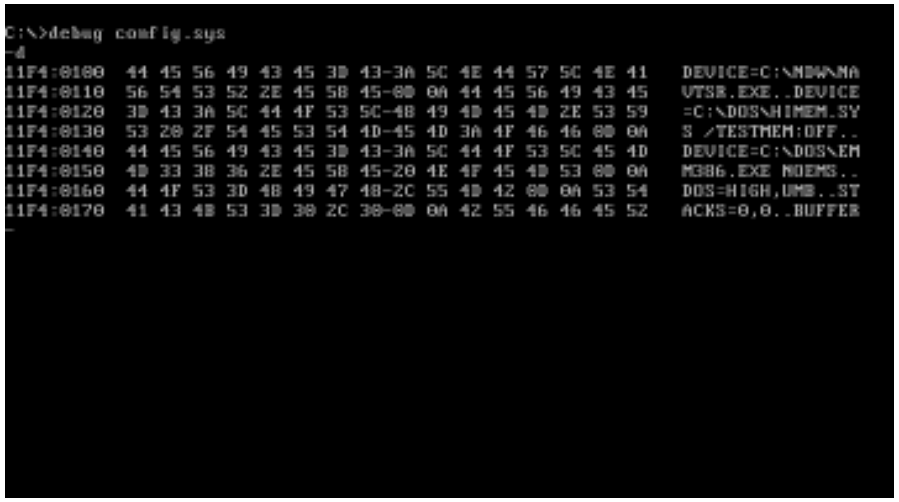
gedeelte laat 16 bytes van het bestand in hexadecimaal formaat zien en in het rechter blok staan diezelfde 16 bytes in ASCII-formaat. In dit laatste blok worden de alfanumerieke tekens weergegeven, dat wil zeggen de ASCII-tekens tussen 32 en 126; andere, niet weer te geven tekens worden voorgesteld door een punt.

Met DEBUG kan elk bestand worden bewerkt, dus niet alleen binaire bestanden. Bekijk het volgende, magere bestand CONFIG.SYS eens:

```
DEVICE=C:\DOS\HIMEM.SYS
DEVICE=C:\DOS\EMM386.EXE RAM I=B000-B7FF NOEMS
```

```
BUFFERS=15,0
FILES=40
DOS=HIGH,UMB
```

Afbeelding 11-2 is een DEBUG-dump van dit bestand. Aan het einde van de eerste tekstregel (te zien in het midden van de tweede regel van de dump) staan de tekens *0D* en *0A*. Deze tekens zijn respectievelijk de harde return en de line feed en worden achter de schermen in ASCII-bestanden gebruikt om het einde van een nieuwe tekstregel aan te geven.



Figuur 11-2

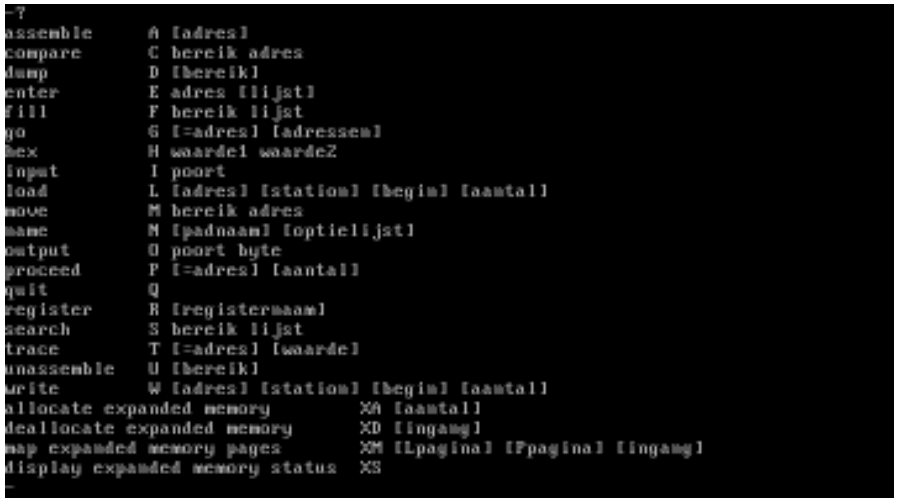
De opdrachten van DEBUG

Wanneer u DEBUG start, wordt er een enkel teken afgebeeld: het minteken (-). Dit is de fantasieloze prompt van DEBUG, die aangeeft dat er op invoer van de gebruiker wordt gewacht. Ik kan me een vriendelijker begroeting voorstellen.

Overzicht DEBUG-opdrachten

U dient om te beginnen twee opdrachten te onthouden. Allereerst de opdracht Q waarmee het programma wordt afgesloten. Ten tweede de opdracht ?, de help-opdracht, waarmee u een lijst oproept met een korte beschrijving van alle opdrachten in DEBUG (zie afbeelding 11-3).

Zoals u in afbeelding 11-3 kunt zien, zijn er maar liefst 23 opdrachten in DEBUG. Deze worden in de volgende paragrafen uitgelegd.



Figuur 11-3

Het commando Help (?)

Help laat een lijst zien met een overzicht van de opdrachten. De syntaxis is eenvoudig:

```
?
```

Het commando A

De opdracht A is een assembleeropdracht. De opdrachten (die in assembleertaal worden ingevoerd) worden vertaald naar machinecode. De syntaxis is als volgt:

```
A [adres]
```

Hiermee kunt u instructies invoeren in de notatie van de machinetaal:

```
MOV AH,2A
INT 21
MOV AH,4C
INT 21
```

Zo hoeft u de bytes van de machinecode niet in het volgende formaat in te voeren:

```
e 0100 B4 2A CD 21 B4 4C CD 21
```

Gewoonlijk zult u met de instructie A een nieuw programma assembleren voordat u dit naar schijf schrijft. DEBUG begint vanaf het begin van het bestand te assembleren of vanaf het punt waar de vorige assemblage is gestopt. Een andere mogelijkheid is, dat u het adres van het bereik opgeeft dat moet worden geassembleerd. Een voorbeeld:

```
-a0100:0108
```

Het commando C

Dit is de vergelijkingsopdracht. De gegevens in twee geheugenlokaties worden vergeleken en voor elke lokatie worden de bytewaarden naast elkaar weergegeven. De syntaxis is:

```
c bereik adres
```

De parameter *bereik* is het begin- en eindadres van de eerste geheugenlokatie, gescheiden door een komma. DEBUG laat elk adres zien waar de waarde van de byte verschilt. Stel dat tijdens het bewerken van het voorbeeldbestand CONFIG.SYS (zie afbeelding 11-2) de volgende opdracht wordt uitgevoerd:

```
-c 0100,010f 0119
```

Dan ziet u als resultaat:

```
11AA:010E  48  45  11AA:0127
11AA:010F  49  4D  11AA:0128
```

Deze vergelijkingsopdracht instrueert DEBUG om de gegevens in de geheugenplaatsen 0100 tot en met 010f te vergelijken met de gegevens die beginnen op geheugenplaats 0119. Beide adressen beginnen met de string DEVICE=C:\DOS\, maar de resterende twee bytes zijn verschillend.

Het commando D

De vergelijkingsopdracht laat DEBUG de gegevens dumpen (dat wil zeggen afbeelden) die op een bepaalde gegevenslokatie liggen opgeslagen. De syntaxis van de opdracht is:

```
d [bereik]
```

Met de dump-opdracht wordt de inhoud van het bestand uit de afbeeldingen 11-1 en 11-2 weergegeven. De optionele bereikparameter geeft de begin- en eindadressen aan of het beginadres en de lengte van het geheugengebied dat u wilt bekijken. Wanneer u geen bereik opgeeft, dumpt DEBUG 128 bytes gegevens, te beginnen bij de byte die ligt na de gegevens die worden weergegeven met de laatste dump-opdracht, of bij het begin van het bestand. Door telkens op D te drukken bladert u in blokken van 128 bytes door een bestand.

Het commando E

De opdracht E (Enter) voert gegevens bij een bepaalde gegevenslokatie in. De syntaxis van de opdracht is:

```
e adres [lijst]
```

De parameter *adres* is het beginadres waar de gegevens zullen worden ingevoerd; de optionele parameter *lijst* is de lijst met de bytewaarden waarmee de gegevens worden bepaald die worden geïnstalleerd. Dit is een voorbeeld:

```
e 0100 B4 2A CD 21 B4 4C CD 21
```

Wanneer u geen gegevens specificeert, schakelt DEBUG over naar de bewerkmodus en vraagt om invoer, byte voor byte. DEBUG laat de bestaande waarde zien (gevolgd door een punt) en wacht totdat de vervangende waarde wordt ingevoerd. De opdracht E is geschikt voor het patchen van bestanden, maar het invoeren van nieuwe programma's kan beter worden gedaan met de assembleertaal, met behulp van het commando A.

Het commando F

De opdracht F is een opdracht om een geheugengebied een aantal malen met een byte of een serie bytes te vullen. De syntaxis voor deze opdracht is als volgt:

```
f bereik lijst
```

De variabele *bereik* geeft het begin- en eindadres van het gebied aan dat wordt gevuld. *Lijst* is de gegevens waarmee het gebied wordt gevuld.

Het commando G

De opdracht G (go) start het programma. De syntaxis hiervoor is:

```
g [=adres] [breekpunten]
```

Deze opdracht is een echt debugmiddel. Het programma wordt gestart op een bepaald adres en u kunt daarna maximaal tien breekpunten instellen door elk breekpunt van het geheugenadres afzonderlijk op te geven. Zoals ik de tracking optie op mijn videorecorder nooit gebruik, zo gebruik ik deze opdracht ook nooit.

Het commando H

H is de hex-opdracht. U kunt er twee hexadecimale getallen mee optellen en aftrekken en het antwoord weergeven. De juiste syntaxis hiervoor is:

```
h waarde1 waarde2
```

Het volgende batchbestand maakt met H een eenvoudige maar effectieve hexadecimale rekenmachine:

```
@echo off
echo Hexadecimale rekenmachine
echo Plus  Min
echo h %1 %2 >$$$$TEMP.SCR
echo q >> $$$$TEMP.SCR
debug < $$$$TEMP.SCR | find "-" /v
del $$$$TEMP.SCR
```

Het commando I

De opdracht I (input) leest gegevens van één byte van een bepaalde poort en geeft deze gegevens weer. De syntaxis voor deze opdracht is:

```
i poort
```

De variabele *poort* verwijst naar het poortadres, bijvoorbeeld 03f8 voor COM1.

Het commando L

De opdracht L (load) laadt een bestand van schijf in een bepaald adres in het geheugen. De syntaxis voor deze opdracht is:

```
l [adres]
```

Standaard wordt het bestand geladen op adres cs:100 (100 bytes in het codesegment). Gewoonlijk laadt u een bestand door een bestandsnaam op de opdrachtregel in te voeren als u DEBUG start. Bij een bestand met de extensie EXE wordt het laadadres genegeerd. In EXE-bestanden staat een laadlokatie in de koptekst van EXE; DEBUG maakt gebruik van dit adres. Wanneer u een EXE-bestand en ook de bytes in de koptekst van EXE wilt bewerken, zult u het bestand moeten herbenoemen, zodat het een andere extensie heeft.

Met DEBUG kunt u ook bepaalde schijfsectoren laden met de opdracht L. Maar omdat hierdoor gegevens kunnen worden veranderd, is deze mogelijkheid niet aan te bevelen.

Het commando M

De opdracht M (move) kopieert de inhoud van een geheugenblok naar een ander geheugenblok. De syntaxis is:

```
m bereik adres
```

Als bron- en doeladres elkaar overlappen, worden de gegevens overschreven.

Het commando N

De opdracht N (name) specificeert de naam van een bestand dat wordt geladen of opgeslagen. Normaal gesproken werkt u met deze opdracht om kleine bestandsutility's (COM) te maken. Wijs met N de bestandsnaam toe voordat met de opdracht W het bestand naar schijf wordt geschreven. De syntaxis is:

```
n [station:][pad]bestandsnaam.ext
```

Het commando O

De opdracht O (output) kopieert een bepaalde byte naar een poort met de volgende syntaxis:

```
o poort byte-waarde
```

Net zoals bij de opdracht I is de *poort* het poortadres, bijvoorbeeld 03f8 voor COM1.

Het commando P

De opdracht P (proceed) voert een programma-instructie uit. De opdracht is voor het opsporen en corrigeren van programmafouten, dus zult u er zelden of nooit mee werken. Voor het geval dat u deze opdracht wel nodig hebt, is hier de juiste syntaxis:

```
p [=adres] [aantal]
```

Het commando Q

De opdracht Q (quit) wordt door iedereen gebruikt. DEBUG wordt hiermee afgesloten en u wordt teruggebracht naar de DOS-prompt. Eventuele wijzigingen worden niet automatisch op schijf opgeslagen. De syntaxis is:

```
q
```

Het commando R

De opdracht R (register) laat een bepaald register zien en stelt u in staat dat register bij te werken. De syntaxis van deze opdracht is:

```
r [register]
```

De volgende standaardregisters worden ondersteund: AX, BX, CX, DX, SP, BP, SI, DI, ES, SS, CS, IP, PC en F (voor flags). De registers AX tot en met DX kunnen ook ter grootte van een byte worden gewijzigd met de hoge en lage registercomponenten. Om bijvoorbeeld AX in twee secties te benaderen moet u AH en AL opgeven. Wanneer u R zonder registernaam start, wordt de status van alle registers afgebeeld. Als u een nieuw COM-bestand maakt, moeten de registers BX:CX worden bijgewerkt met de bestandsgrootte voordat u de opdracht W kunt oproepen. W bepaalt met deze registers hoeveel bytes er kunnen worden geschreven.

Het commando S

De opdracht S (search) zoekt in een geheugengebied een bepaalde byte, een aantal bytes of een reeks. De syntaxis hiervoor is:

```
s bereik lijst
```

Het *bereik* geeft het begin- en eindadres van het geheugen aan dat wordt gezocht. De *lijst* geeft de bytes weer (of series bytes gescheiden door spaties) die u wilt opzoeken. Wanneer u een enkele byte of een reeks specificeert, laat DEBUG alle geheugenlokaties zien waar een overeenkomst is gevonden. Als u een reeks bytes opgeeft, wordt het adres van de eerste overeenkomst weergegeven. U kunt met de schakeloptie D de gegevens op dat adres weergegeven.

Het commando T

De opdracht T (trace) voert een enkele instructie uit en laat vervolgens de inhoud van de registers zien. Dit is ook een debugoptie die u zelden zult gebruiken. Voor het geval u deze optie wel nodig hebt, is de juiste syntaxis:

```
t [=adres] aantal
```

Het commando U

De opdracht U (unassemble) laat een deel van het programma zien. Dit wordt *disassembleren* genoemd. De syntaxis hiervoor is:

```
u [bereik]
```

De opdracht U is een erg handig hulpmiddel voor het inspecteren van kleine programma's. U moet echter de assembleertaal kennen. Wanneer u geen *bereik* opgeeft, wordt een code ter grootte van 32 bytes weergegeven, te beginnen bij de geheugenlokatie die onmiddellijk volgt op de laatste niet-geassembleerde code.

Het commando W

De opdracht W (write) schrijft een bestand naar schijf, met andere woorden slaat een bestand op. Hiervoor is de volgende syntaxis:

```
w [adres]
```

Als u geen adres opgeeft, wordt het volledige bestand weggeschreven naar bepaalde schijfsectoren. Om een nieuw bestand naar schijf te schrijven moet u, voordat u de opdracht W start, met de schakeloptie N het bestand een naam geven en de grootte van het bestand in de registers BX:CX opslaan.

U kunt ook met de opdracht W gegevens naar bepaalde schijfsectoren schrijven. Omdat deze techniek DOS overslaat en waardevolle gegevens kan vernietigen, is het niet aan te raden deze mogelijkheid te gebruiken.

Het commando XA

De opdracht XA (expanded geheugen toewijzen) wijst expanded geheugenpagina's toe en laat het handlernummer zien als het is gelukt. Tenzij u met XMS wilt spelen, zult u weinig met deze en de volgende opdrachten werken. Voor de geïnteresseerden is de syntaxis:

```
xa [aantal]
```

Het commando XD

De opdracht XD (expanded geheugen) maakt de toewijzing van expanded geheugen ongedaan en maakt de pagina's beschikbaar voor andere programma's. De syntaxis voor deze opdracht is:

```
xd [ingang]
```

Het commando XM

De opdracht XM (expanded geheugenpagina's koppelen) koppelt een logische pagina aan een fysieke pagina. Gebruik de volgende syntaxis:

```
xm [lpagina] [fpagina] [ingang]
```

Het commando XS

De opdracht XS (status expanded geheugen) geeft de huidige status van het expanded geheugen. Van alle debugschakelopties van XMS is dit de interessantste optie omdat deze een overzicht geeft van het gebruik van het XMS. De syntaxis hiervoor is eenvoudig:

```
xs
```

Het helpsysteem van DOS 6 geeft een uitgebreid overzicht van alle debug-opdrachten. Voer `help debug` in om uitleg op te vragen over deze opdrachten.

De opdrachten van DEBUG invoeren

DEBUG maakt zich niet echt druk om spaties in opdrachten. Over het algemeen hoeft u tussen de opdracht en de parameters geen spaties te typen. Voor extra duidelijkheid kunt u de parameters wel van elkaar scheiden door een spatie of komma. De volgende drie opdrachten betekenen allemaal hetzelfde:

```
-nutil.com
-n util.com
-n,util.com
```

U moet bij veel opdrachten van DEBUG het bereik of adres van het geheugen opgeven. Voer de adressen in het formaat *segment:offset* en in de hexadecimale notatie in. (Zie voor segmenten, offsets en hexadecimale getallen ook hoofdstuk 5.) Alle DOS-programma's hebben twee speciale segmentadressen: het code- en het gegevenssegment. Deze segmenten geven het beginadres van de code

van een programma en de gegevens aan. DEBUG ondersteunt de segmentaliassen CS en DS. Een geheugenadres dat bijvoorbeeld 200 bytes in een codesegment ligt, is CS:0200.

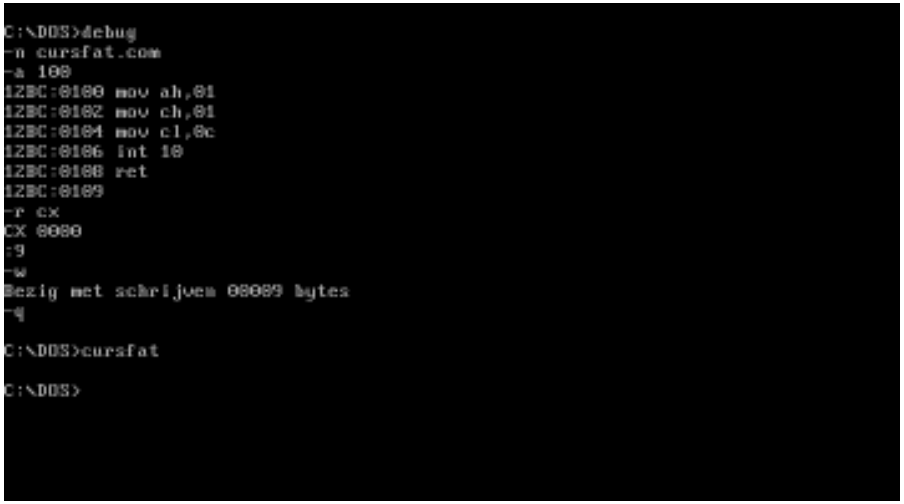
Wanneer u een enkel adres (zonder dubbele punt) opgeeft, gaat DEBUG ervan uit dat het een geheugen-offset is en voegt een standaardsegment toe. Het standaardsegment is het codesegment bij de opdrachten A, G, L, T, U en W; de standaard is het gegevenssegment voor alle andere opdrachten.

Een proefrit

Na de theorie is het nu tijd voor de praktijk. Zoals ook bij de meeste programmeertalen het geval is, leert u DEBUG het beste door ermee te werken.

Standaard is de DOS-cursor een dun, knipperend streepje onder aan de tekst. Met DEBUG kunt u een klein programmaatje maken dat de cursor verandert in een vierkant blokje. Zie afbeelding 11-4 voor dit programma, CURSFAT.COM genaamd.

Zie tabel 11-1 voor de instructies die u moet invoeren voor dit programma.



Figuur 11-4

Wanneer u dit programma start, verandert de cursor in een blokje. Het is wel prettig als u de oorspronkelijke cursor weer kunt terughalen. Met het programma CURSNORM.COM krijgt u de oorspronkelijke cursor weer. Ondertussen kunt u de normale cursor terughalen met de opdracht mode co80 (of bw80).

De sleutel tot het schrijven van kleine programma's met DEBUG is te weten welke mogelijkheden van DOS kunnen worden gebruikt. In CURSFAT.COM wordt met interrupt 10h (de

Tabel 11-1 Het programma CURSFAT.COM

Stap	Invoer	Omschrijving
Stap 1	debug	Start met de opdracht DEBUG achter de prompt een sessie met DEBUG.
Stap 2	n cursfat.com	Noem het bestand CURSFAT.COM
Stap 3	a 100	Deze opdracht begint met de assembleercode op adres CS:0100 dat het beginadres is voor programma's met de extensie COM. Zodra u deze opdracht invoert, schakelt DEBUG over naar Assemble Mode en wacht op invoer van assembleertaal.
Stap 4	mov ah,01	Stel eerst de waarde van AH in op 01 om de cursor te veranderen.
Stap 5	mov ch,01	Deze stap stelt het register CH in op de eerste cursorregel, dat wil zeggen de positie van de scanregel van de bovenkant van de cursor.
Stap 6	mov c1,0c	Deze stap stelt het register CL in als de laatste scanregel van de cursor. *Opmerking*: bij een kleurensysteem wordt gebruik gemaakt van 14 scanregels per tekenblok, terwijl een monochroom systeem slechts 8 scanregels gebruikt. De scanregels zijn respectievelijk van 0 tot C (13 decimaal) en van 0 tot 7 genummerd. Voer `mov c1,07` in als u dit programma maakt voor een monochroom systeem.
Stap 7	int 10	Roept DOS-interrupt 10h aan. Wanneer dit gebeurt, wordt het register AH geïnspecteerd om vast te stellen welke taak er moet worden uitgevoerd. In dit geval wordt de waarde 01 gevonden, die de code is voor het veranderen van de vorm van de cursor.
Stap 8	ret	Deze stap draagt de besturing over aan DOS.
Stap 9	Druk op Enter	Met Enter stopt u de sessie van DEBUG en keert u terug naar de prompt.

Stap 10	r cx	Voer deze opdracht in om aan DEBUG te laten weten dat de waarde van het register CX moet worden veranderd. DEBUG laat de huidige registerwaarde zien (0000 in dit geval) en wacht totdat u een nieuwe waarde invoert.
Stap 11	9	Geeft de grootte van het programma aan. Dit programma bestaat uit slechts 9 bytes, omdat elke instructie die wordt ingevoerd in stap 4 tot en met 7 uit twee bytes bestaat. De instructie RET in stap 8 gebruikt 1 byte.
Stap 12	w	Schrijft het programma (alle 9 bytes) naar schijf.
Stap 13	q	Sluit DEBUG af en keert terug naar de prompt.

service-interrupt van BIOS), subfunctie 01h de vorm van de cursor veranderd. De afmetingen van de cursor liggen opgeslagen in de registers CH en CL voordat de interrupt werd aangeroepen. U zult dergelijke bijzonderheden niet kennen, tenzij u een ervaren programmeur in DOS bent. Wanneer u uw eigen utility's wilt gaan schrijven, hebt u een boek over programmeren in DOS nodig waarin alle DOS-interrupts staan. Een goed boek hiervoor is *PC Interrupts* van Ralph Kyle en Jim Brown.

ASCII-scriptbestanden maken

Het interactief schrijven van programma's met DEBUG (zoals u hebt gedaan met CURSFAT.COM) is praktisch alleen geschikt voor erg kleine programma's. Bij grotere programma's kan het invoeren van een verkeerde instructie zeer frusterend zijn. U kunt correcties aanbrengen met de schakeloptie /E, maar echt probleemloos werkt dit niet. Vaak wordt het zo lastig dat u ermee stopt, opnieuw begint en alle instructies van begin af aan invoert.

Een andere mogelijkheid is EDIT of een andere tekstverwerker voor het maken van ASCII-bestanden en per regel één opdracht in te voeren. Nadat u de opdrachten hebt ingevoerd en gecontroleerd, kunt u het resultaat omleiden naar DEBUG en het programma maken.

ASCII-bestanden waarin opdrachten van DEBUG staan, worden vaak *scriptbestanden* genoemd.

In het volgende scriptbestand, CURSNORM.SCR, staan alle instructies voor het programma CURSNORM.COM, waarmee de oorspronkelijke vorm van de cursor wordt teruggehaald:

```
n cursnorm.com
a 100
mov ah,01
mov ch,0c; gebruik ch,06 bij monochroom
mov cl,0d; gebruik cl,07 bij monochroom
int 10
ret

r cx
9
w
q
```

Tekens achter een puntkomma worden genegeerd als DEBUG in *assemble mode* staat. Hierdoor kunt u makkelijk verklarende tekst opnemen.

Wanneer u een scriptbestand voor DEBUG maakt, dient u de opdrachten goed te controleren om te voorkomen dat er fouten in staan. Door een onjuiste opdracht kan het systeem hangen, en in uitzonderlijke gevallen kunnen de FAT-bestanden worden beschadigd. De twee meest gemaakte fouten zijn het vergeten van een lege regel achter RET (of laatste INT 21) en de opdracht Q. Controleer vooral op deze twee opdrachten.

Als u het scriptbestand hebt gemaakt, kunt u dit met het teken < naar DEBUG omleiden. Met de volgende opdracht maakt u het bestand CURSNORM.COM:

```
debug < cursnorm.scr
```

Programma's met DEBUG wijzigen

DEBUG wordt vaak gebruikt voor het bewerken of veranderen van een bestaand programma. U kunt een fout (bug) uit een programma halen, foutberichten veranderen of de kleuren van het programma wijzigen. Voordat u een programma gaat veranderen, moet u eerst een reservekopie maken voor het geval het resultaat niet is wat u ervan verwachtte.

Om een programma te wijzigen moet u eerst natuurlijk de gegevens opzoeken die u wilt veranderen. In een ASCII-bestand zult u waarschijnlijk met PgUp of PgDn of met een zoekopdracht het gewenste tekstgedeelte opzoeken. Dezelfde procedure kunt u volgen bij een programmabestand; de corresponderende opdrachten in DEBUG zijn D (dump) en S (search).

Nadat u het te wijzigen gedeelte hebt opgezocht, wijzigt u met E (edit) de gegevens en schrijft u met W (write) de gegevens naar schijf. Omdat het bestand is geladen met DEBUG, kunt u W gebruiken zonder de bestandsgrootte eerst in de registers BX:CX op te slaan. DEBUG gaat ervan uit dat het bestand even groot is als toen het werd geladen.

Ter illustratie hiervan kunt u twee populaire opdrachten van DOS veranderen.

De opdracht CHOICE veranderen

Standaard wordt u door de opdracht CHOICE gevraagd Y of N te kiezen. Met DEBUG zijn deze standaardtoetsen makkelijk te veranderen.

Kopieer eerst het programma CHOICE.COM naar CHOICE12.COM en laad vervolgens het nieuwe programma in DEBUG door de volgende opdracht in te voeren:

```
debug choice12.com
```

Typ achter de prompt van DEBUG (-) de opdracht D om de eerste 128 bytes te dumpen. U zult het volgende zien:

```
7738:0100  E9 32 04 59 4E 00 00 00-00 00 00 00 00 00 00 00   .2.YN...........
7738:0110  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
7738:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
7738:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
7738:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
7738:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
7738:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
7738:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Het segmentadres van het geheugen (in de eerste kolom) kan in uw geval verschillen.

U ziet de tekens Y en N op de eerste regel. Deze zijn de standaardtekens die worden gebruikt als u CHOICE zonder de schakeloptie /C start. Om deze waarden te veranderen in *1* en *2* moet u de volgende opdracht invoeren:

```
-e 0103 31 32
```

Door deze opdracht worden de gegevens te beginnen bij offset 0103 bewerkt, en worden de eerste twee bytes veranderd in de waarden 31 en 32. (31 en 32 zijn de hexadecimale ASCII-waarden voor de tekens 1 en 2.) Met de volgende opdracht kunt u de wijzigingen controleren:

```
d 0100
```

Als laatste slaat u de wijzigingen op met W; vervolgens sluit u DEBUG af met Q.

Met een paar eenvoudige instructies hebt u de standaardwaarde Y en N van CHOICE in 1 en 2 veranderd.

Om helemaal volledig te zijn zou u ook de helptekst moeten aanpassen. Start DEBUG met de volgende opdracht:

```
debug choice12.com
```

Zoek met een zoekopdracht (S) alle verwijzingen naar YN op. Voer bij de prompt van DEBUG de volgende opdracht in met YN in hoofdletters (de zoekopdracht maakt onderscheid tussen hoofdletters en kleine letters):

```
-s 0100 FFFF "YN"
```

Deze opdracht laat DEBUG door het gehele codesegment naar alle YN's zoeken. Als antwoord worden alle adressen gegeven waar de tekst staat, bijvoorbeeld het volgende adres:

```
11CC:0228
```

Het codesegment (in dit voorbeeld 11CC) kan bij u anders zijn.

Voer de opdracht D (dump) in om de gegevens in dit gebied weer te geven:

```
-d 0220
```

De gegevens worden afgebeeld en u zult zien dat "YN" inderdaad in de helptekst staat. Met de

volgende opdracht EDIT kunt u de helptekst veranderen in "12":

```
-e 0228 31 32
```

Tot slot slaat u met W de veranderingen op naar schijf en sluit u het programma af met Q. U kunt uw werk bekijken met de volgende opdracht:

```
choice12 /?
```

Het zal u misschien zijn opgevallen dat er veel lege ruimte aan het begin van het programma CHOICE.COM staat. Mocht u zich afvragen of deze 00's kunnen worden veranderd door extra standaardtoetsen, dan kan het antwoord hierop ja zijn. Er kunnen zelfs speciale toetsen worden toegevoegd die niet direct op de opdrachtregel worden ondersteund door CHOICE, zoals de spatiebalk en de Enter-toets. Het is dus mogelijk CHOICE zo te veranderen dat invoer van de spatiebalk en de Enter-toets wordt geaccepteerd.

Het volgende scriptbestand, CHOICEAZ.SCR, verandert de standaardtoetsen van CHOICE in de spatiebalk, Enter, 0-9 en A-Z:

```
n choiceaz.com
l
d
e 0103 20 d 30 31 32 33 34 35 36 37 38 39
e 0110 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D
4E 4F 50
e 0120 51 52 53 54 55 56 57 58 59 5A
d 0100
w
q
```

Kopieer CHOICE.COM naar CHOICEAZ.COM en start daarna de volgende opdracht om het script naar DEBUG om te leiden en CHOICEAZ.COM bij te werken:

```
debug < choiceaz.scr
```

De opdracht CHOICEAZ is een ideale optie als u met batchbestanden werkt waarin u wilt, dat er pas wordt verder gegaan als op een toets wordt gedrukt, of na een bepaalde tijd als er geen toets wordt ingedrukt. Met de volgende opdracht kunt u een batchbestand tien seconden laten wachten of laten wachten op een toetsaanslag, waaronder de spatiebalk of Enter (het bestand reageert op wat er het eerst gebeurt):

```
choiceaz /n /T:A,10
```

Het standaard bestandsmasker van EDIT wijzigen

Ik gebruik EDIT hoofdzakelijk voor het schrijven en veranderen van batchbestanden. Een irritant ding vind ik het standaard bestandsmasker *.TXT in het dialoogvenster File Open. Met een beetje hulp van DEBUG is dit masker te veranderen in *.BAT.

EDIT.COM is niet een tekstverwerker met uitgebreide mogelijkheden. Wanneer u de opdracht EDIT uitvoert, wordt door EDIT het programma QBASIC met de schakeloptie /EDITOR gestart. QBASIC.EXE levert alle bewerkmogelijkheden en ergens diep in QBASIC ligt het standaard bestandsmasker *.TXT.

DEBUG ondersteunt niet het bewerken van bestanden met de extensie EXE. Een EXE-bestand wordt op een bepaalde manier geladen, zodat de geladen gegevens niet een exacte kopie van het bestand zijn. U kunt dit omzeilen door QBASIC.EXE te kopiëren naar een bestand met een andere extensie, bijvoorbeeld DBG.

Hierna kunt u een DEBUG-sessie met de volgende opdracht starten:

```
debug qbasic.dbg
```

In het voorbeeld dat in de vorige paragraaf is besproken, zoekt u met de zoekopdracht van DEBUG de tekstreeks "YN" op. U gaat er misschien van uit dat eenzelfde benadering in dit voorbeeld zou werken. Om dit uit te proberen moet u met de volgende opdracht *.TXT (hoofdletters) zoeken:

```
-s 0100 FFFF "*.TXT"
```

DEBUG vindt geen reeksen "*.TXT" en dus worden er geen geheugenadressen weergegeven. Waarom? De zoekopdracht zoekt alleen in de eerste 64K van het geheugen en QBASIC is bijna 200 bytes groot. DEBUG vindt het bestandsmasker niet omdat dit niet in de eerste 64K ligt opgeslagen. Om de eerste 64K te laten doorzoeken moet u enkele hexadecimale berekeningen uitvoeren.

De eerste stap is om het speciale adres van het codesegment te bepalen. Dump met de opdracht D de eerste 128 bytes van QBASIC.EXE. Wat

hieronder staat, zal lijken op wat u op het scherm zult zien.

```
1324:0100  4D 5A 05 01 7C 01 01 00-08 00 94 15 FF FF 71 2F   MZ..|.........q/
1324:0110  00 02 00 00 00 01 F0 FF-52 00 00 00 0F 21 50 4B   ........R....!PK
1324:0120  4C 49 54 45 20 43 6F 70-72 2E 20 31 39 39 30 2D   LITE Copr. 1990-
1324:0130  39 32 20 50 4B 57 41 52-45 20 49 6E 63 2E 20 41   92 PKWARE Inc. A
1324:0140  6C 6C 20 52 69 67 68 74-73 20 52 65 73 65 72 76   ll Rights Reserv
1324:0150  65 64 07 00 00 00 20 01-F8 01 00 00 20 00 2C 06   ed.... ..... .,.
1324:0160  FF FF D8 40 80 00 00 00-10 00 D9 3D 1E 00 00 00   ...@.......=....
1324:0170  01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

De resterende stappen hebben betrekking op het werken met segmentadressen. Het adres kan anders zijn dan op uw systeem, dus werk met de adressen die op uw monitor worden weergegeven en niet met de adressen uit deze voorbeelden.

Met de hexadecimale rekenmachine van DEBUG kunt u het segment voor de volgende 64K opzoeken. Voeg 0FFFh toe aan het codesegment; elk segmentadres wordt vermenigvuldigd met 16 (decimaal) om een lineair adres te krijgen. Specificeer met de opdracht H het segmentadres en 0FFF. Als voorbeeld het volgende:

```
-h 1324 0fff
```

DEBUG laat als antwoord twee waarden zien. De eerste waarde is de som van de twee getallen; de tweede waarde is het verschil tussen de twee getallen. Dit wordt door het volgende voorbeeld geïllustreerd:

```
2323 01CD
```

De eerste waarde vertegenwoordigt het beginsegment van de volgende 64K van QBASIC.DBG. Gebruik als volgt deze waarde in een andere zoekopdracht:

```
-s 2323:0000 FFFF "*.TXT"
```

Ook nu wordt er geen overeenkomst gevonden. Bereken met de opdracht H het beginsegment voor de volgende 64K. Een voorbeeld:

```
-h 2323 0FFF
```

In dit geval wordt de waarde 3322 geretourneerd. Probeer nog eens te zoeken in de derde 64K.

```
-s 3322:0000 FFFF "*.TXT"
```

Bingo! DEBUG antwoordt met een enkel adres waarmee wordt aangegeven dat de tekst is gevonden:

```
3322:DD90
```

De reeks "*.TXT" moest worden gevonden. Met EDIT kan deze reeks nu makkelijk worden veranderd. Gebruik het specifieke adres dat wordt weergegeven door de zoekopdracht (bijvoorbeeld 3322:DD90) en verander als volgt de eerste vijf tekens in *.BAT:

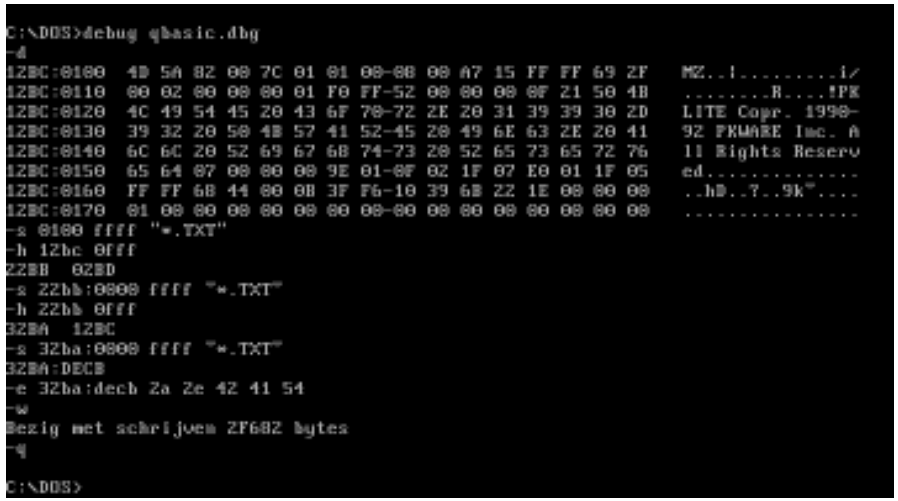
```
-e 3322:DD90 2A 2E 42 41 54
```

Sla het bestand op met W en sluit DEBUG af met Q.

Voordat u QBASIC.DBG herbenoemt als QBASIC.EXE, moet u QBASIC. EXE eerst herbenoemen als QBASIC.BAK, voor het geval u niet de juiste wijzigingen hebt aangebracht. Nadat u de bestanden hebt herbenoemd, zult u zien dat het standaard bestandsmasker nu *.BAT is. Zie afbeelding 11-5 voor de instructies om het bestandsmasker te wijzigen.

Handige utility's met DEBUG maken

DEBUG is een ideaal programma voor het maken van compacte, krachtige utility's. In de volgende paragrafen worden programma's beschreven voor



Figuur 11-5

het retourneren van de systeemdatum en interactie met het toetsenbord. Bovendien leest u hoe utility's voor diskettestations zijn te maken.

Terug naar DOS en ERRORLEVEL instellen

Veel utility's die zijn gemaakt met DEBUG, zijn ontworpen voor batchbestanden. Omdat de opdracht IF de ERRORLEVEL van een programma kan testen, wordt een ERRORLEVEL door veel utility's ingesteld voor het doorgeven van gegevens aan het batchbestand.

Met interrupt 21h, subfunctie 4Ch (liever dan RET) kan de besturing terug worden gegeven aan DOS en kan een afsluitcode (ERRORLEVEL) worden ingesteld. De volgende twee instructies van de assembleertaal geven een voorbeeld van deze basistechniek:

```
mov ah,4c
int 21
```

De ERRORLEVEL die wordt geretourneerd door een programma dat op deze manier wordt beëindigd, komt overeen met de waarde in het register AL. Deze techniek wordt in de volgende voorbeelden toegepast.

Data

Interrupt 21h, subfunctie 2Ah geeft informatie over de systeemdatum. Wanneer deze techniek wordt aangeroepen, plaatst DOS de dag van de week in het register AL, de dag van de maand in het register DL, de maand in het register DH en het jaar (vier cijfers) in het register CX.

Gewapend met deze kennis kunt u makkelijk met DEBUG enkele nuttige datumroutines maken.

Het volgende scriptbestand, WEEKDAY.SCR, geeft de dag van de week als een ERRORLEVEL:

```
n weekday.com
a 100
mov ah,2a
int 21      ;weekdag staat nu in AL
mov ah,4c
int 21

r cx
8
w
q
```

Met de volgende opdracht maakt u het programma WEEKDAY.COM:

```
debug < weekday.scr
```

WEEKDAY.COM stelt een ERRORLEVEL in waarmee de dag van de week wordt aangegeven. Zondag wordt weergegeven door ERRORLEVEL 0, maandag door ERRORLEVEL 1, enzovoorts. De ERRORLEVEL voor zaterdag is 6.

Het volgende scriptbestand, WHATDAY.SCR, maakt het programma WHATDAY.COM,

waarmee de dag van de maand wordt weergegeven en retourneert een ERRORLEVEL van 1 tot 31:

```
n whatday.com
a 100
mov ah,2a
int 21
mov a1,d1  ;retourneer dag van de maand
mov ah,4c
int 21

r cx
A
w
q
```

Het scriptbestand is gelijk aan WEEKDAY.SCR, maar de dag van de maand wordt geretourneerd in het register DL, en dit moet vervolgens worden overgebracht naar register AL, zodat het kan worden geretourneerd als ERRORLEVEL. De aanvullende instructie verandert de bestandsgrootte in A bytes (10 decimaal).

Het volgende scriptbestand, WHATMON.SCR, maakt het programma WHATMON.COM, dat de maand bepaalt en een ERRORLEVEL retourneert van 1 tot 12:

```
n whatmon.com
a 100
mov ah,2a
int 21
mov a1,dh  ;retourneert maand
mov ah,dh
int 21

r cx
A
w
q
```

Dit scriptbestand is bijna gelijk aan WHATDAY.COM. Het enige verschil is dat de waarde van de maand wordt geretourneerd in het register DH.

Het laatste scriptbestand van dit kwartet datumprogramma's is WHATYEAR.SCR, dat het jaar als een ERRORLEVEL retourneert. DOS geeft het jaar in vier cijfers in het register CX. ERRORLEVELs worden alleen tussen 0 en 255 ondersteund. Door het jaar te verplaatsen naar het register AX en 1980 ervan af te trekken (7CB in

hexadecimaal), wordt de ERRORLEVEL ingesteld als het jaar min 1980. Zo wordt 1993 geretourneerd als ERRORLEVEL 13. Het bestand WHATYEAR.SCR ziet er als volgt uit:

```
n whatyear.com
a 100
mov ah,2a
int 21
mov ax,cx     ; plaats jaar in AX
sub ax,07bc   ; min 1980
mov ah,4c     ; restant nu in AL
int 21

r cx
D
w
q
```

Utility's voor toetsenbordbeheer schrijven

De programmeertaal van batchbestanden heeft maar weinig manieren voor gegevensinvoer of controle van de status van het toetsenbord. De utility's in dit gedeelte vullen aan waar de batchtaal tekort schiet.

De status van bepaalde toetsen bekijken en wijzigen

Diep weggedoken in BIOS, op adres 0040:0017, ligt een zeer nuttige byte met informatie. Deze byte geeft de status aan van de speciale toetsen Left Shift, Right Shift, Ctrl, Alt, Scroll Lock, Num Lock, Caps Lock en Ins. Elke bit van de acht bits is ingesteld op *aan* of *uit*. Hiermee wordt aangegeven of één van deze toetsen wordt ingedrukt gehouden of, in geval van de toetsen die kunnen worden vastgezet, een dergelijke toets is vastgezet. Door deze byte als ERRORLEVEL te retourneren kan er met een kleine utility aan een batchbestand de status van deze speciale toetsen worden meegedeeld.

Met het volgende scriptbestand, KEYSTATE.SCR, kunt u KEYSTATE.COM maken:

```
n keystate.com
a 100
mov ax,0040
mov ds,ax
move al,[17]   ;retourneert
toetsenstatus in errorlevel
mov ah,4c
int 21
```

```
r cx
C
w
q
```

Wanneer vierkante haakjes ([]) worden gebruikt in een opdracht MOV, wordt de waarde als een geheugenlokatie opgevat en worden de gegevens op die bepaalde geheugenlokatie in een register geplaatst. In dit voorbeeld wordt de zeventiende byte in het gegevenssegment (dat verwijst naar segment 0400) naar het register AL verplaatst.

De ERRORLEVELs die zijn ingesteld door elke toets, staan in tabel 11-2. KEYSTATE opent de mogelijkheid 'sluiproutes' aan batchbestanden toe te voegen. Als voorbeeld wordt het bestand BYPASS.BAT gegeven:

```
1 @echo off
2 keystate
3 if errorlevel 16 if not errorlevel 17 goto
jump
4 choice "Wilt u echt alle bestanden *.BAK
verwijderen"
5 if errorlevel 2 goto quit
6 :JUMP
7 del *.bak
8: QUIT
```

Tabel 11-2 ERRORLEVELs ingesteld door KEYSTATE.COM	
ERRORLEVEL	Status toets
0	Er zijn geen toetsen actief
1	Right Shift wordt ingedrukt gehouden
2	Left Shift wordt ingedrukt gehouden
4	Ctrl wordt ingedrukt gehouden
8	Alt wordt ingedrukt gehouden
16	Scroll Lock is actief
32	Num Lock is actief
64	Caps Lock is actief
128	Ins wordt ingedrukt gehouden

Op regel 2 wordt gecontroleerd of de toets Scroll Lock is geactiveerd (omdat KEYSTATE een ERRORLEVEL van 16 instelt als dat zo is), en als dat zo is, wordt de vraag overgeslagen en worden de bestanden verwijderd.

Als er meerdere toetsen zijn geactiveerd of worden ingedrukt gehouden, is de ERRORLEVEL die wordt geretourneerd door KEYSTATE, de som van de afzonderlijke ERRORLEVELs. Wanneer bijvoorbeeld Caps Lock is geactiveerd en de Alt-toets wordt ingedrukt gehouden, wordt de ERRORLEVEL ingesteld op 72.

Door de waarde van de byte op adres 0040:0017 te veranderen kunt u de status van Caps Lock, Num Lock en Scroll Lock veranderen. Het volgende scriptbestand maakt CAPSOFF.COM, dat de toets Caps Lock uitzet:

```
n capsoff.com
a 100
mov ax,40
mov ds,ax
mov al,[17]
and al,BF   ; zet bit 1 uit
mov [17],al
ret

r cx
E
q
w
```

Het volgende scriptbestand maakt CAPSON.COM, waarmee Caps Lock wordt geactiveerd:

```
n capson.com
a 100
mov ax,40
mov ds,ax
mov al,[17]
or al,40    ; zet bit 1 uit
mov [17],al
ret

r cx
E
w
q
```

De opdrachten AND en OR in deze twee scriptbestanden zetten respectievelijk Caps Lock aan of uit. Met dezelfde techniek kunt u utility's maken waarmee Num Lock en Scroll Lock kunnen worden aan- of uitgezet:

`and al,df` Zet Num Lock uit

`or al,20` Zet Num Lock aan

`and al,ef` Zet Scroll Lock uit

`or al,10` Zet Scroll lock aan

Herstarten

Met het scriptbestand REBOOT.SCR kan de utility REBOOT.COM worden gemaakt:

```
n reboot.com
a 100
mov ah,0d       : disk cache legen
int 21          : interrupt aanroepen
mov ax,0040
mov ds,ax
mov ax,1234
mov [72],ax
jmp FFFF:0000

r cx
14
w
q
```

Wanneer u REBOOT.COM uitvoert, wordt het systeem herstart.

Door de eerste aanroepen van dit kleine programma wordt AH gelijk aan 0 gemaakt en wordt interrupt 21 aangeroepen om ervoor te zorgen dat eventuele gegevens in een schrijfvertraagde diskcache worden opgeslagen op schijf. (Zie hoofdstuk 4 voor een schrijfvertraagde cache.) Vervolgens wordt de waarde 1234h naar het adres 0040:0072 verplaatst, waarna de code op adres FFFF:0000 wordt uitgevoerd. Dit is de BIOS-code power-on self test (POST) die gebruikt wordt bij het starten van het systeem. Wanneer de instructies van POST worden uitgevoerd, wordt de waarde op adres 0040:0072 geïnspecteerd. Als de waarde daarvan 1234h is, wordt het systeem 'warm' herstart. Wanneer deze geheugenlokatie een ander adres heeft, wordt het systeem 'koud' herstart. U kunt REBOOT.SCR zo aanpassen dat het systeem 'koud' wordt gestart. Vervang de opdracht MOV AX,1234 door MOV AX,ABCD.

Wanneer REBOOT.COM wordt uitgevoerd in een DOS-sessie die vanuit Windows is gestart, wordt de poging om te herstarten niet door Windows onderschept zoals gebeurt als u op Ctrl-Alt-Del drukt; het systeem herstart gewoon.

Op een functietoets wachten

Het kan misschien ouderwets lijken, maar ik werk graag met functietoetsen. CHOICE ondersteunt geen functietoetsen. Het volgende scriptbestand

GETFUNCY.SCR maakt het programma GETFUNCY.COM:

```
n getfuncy.com
a 100
jmp 0108    ; toontje overslaan
mov dl,7    ; piepje
mov ah,2    ; subfunctie 2
int 21      ; piep
mov ah,00   ; subfunctie 0
int 16      ; wacht op toetsaanslag
cmp al,00   ; functietoets?
ja 0102     ; nee
cmp ah,3b   ; vgl. 59 (F1)
jl 0102     ; te laag - nog een toets
cmp ah,44   ; vgl 68 (F10)
ja 0102     ; te hoog - nog een toets
sub ah,3a   ; min 58
mov al,ah   ; zet als errorlevel
mov ah,4c
int 21

r cx
23
w
q
```

GETFUNCY wacht totdat de gebruiker een functietoets indrukt (F1 tot en met F10). Wanneer er een andere toets wordt ingedrukt, produceert de computer een geluidssignaal en wacht totdat er een andere toets wordt ingedrukt. Er wordt een ERRORLEVEL geretourneerd waarmee wordt aangegeven welke toets er is ingedrukt, van ERRORLEVEL 1 voor F1 tot en met ERRORLEVEL 10 voor F10.

Het volgende batchbestand laat zien hoe GETFUNCY functioneert:

```
@echo off
echo Druk op een functietoets
getfuncy
if errorlevel 10 echo U drukte op F10
if errorlevel 9 if not errorlevel 10 echo U drukte op F9
if errorlevel 8 if not errorlevel 9 echo U drukte op F8
if errorlevel 7 if not errorlevel 8 echo U drukte op F7
if errorlevel 6 if not errorlevel 7 echo U drukte op F6
if errorlevel 5 if not errorlevel 6 echo U drukte op F5
if errorlevel 4 if not errorlevel 5 echo U drukte op F4
if errorlevel 3 if not errorlevel 4 echo U drukte op F3
if errorlevel 2 if not errorlevel 3 echo U drukte op F2
if errorlevel 1 if not errorlevel 2 echo U drukte op F1
```

Utility's voor schijf en station

In de volgende paragrafen worden twee utility's belicht voor ondersteuning bij het beheren van schijven.

Afbreken, Herhalen, Overslaan? Niet!

Er zijn weinig berichten die nog lelijker zijn dan `Abort, Retry, Fail?`, dat verschijnt wanneer er wordt geprobeerd een leeg station te benaderen.

Het volgende scriptbestand, DRIVEAOK.SCR, maakt een programma, DRIVEAOK.COM, waarmee wordt getest of er in station A een diskette is geplaatst:

```
n driveaok.com
a 100
mov ax,2524     ; interrupt handler 24
veranderen
mov dx,0113     ; nieuwe handler op adres 0113
int 21          ; roep interrupt
mov ah,36       ; vrije ruimte op schijf
opvragen
mov dl,01       ; station A
int 21          ; roep interrupt
mov ax,4c00     ; OK: errorlevel op 0 zetten
int 21          ; klaar
mov ax,4c01     ; mis: errorlevel op 1 zetten
int 21          ; Call it a day

r cx
18
w
q
```


Er zit meer in dit bestand dan u zo op het eerste gezicht zou vermoeden. Wanneer er een leeg station wordt benaderd, wordt interrupt 24 aangeroepen, de kritieke fouthandler. De eerste taak is de kritieke fouthandler te veranderen in een eigen code en het bericht `Abort, Retry, Fail?` te omzeilen. Interrupt 21h, subfunctie 25 laat DOS de interrupthandler vervangen door de interrupt die wordt gespecificeerd in register AL. De nieuwe interrupthandler wordt geïdentificeerd in register DX. In deze situatie wijst DX naar byte 113 van het gegevenssegment, dat gewoon de code is waarmee een ERRORLEVEL van 1 wordt ingesteld en het programma wordt beëindigd.

De volgende stap is te proberen het station te benaderen, waarbij een kritieke fout wordt aangeroepen als er geen diskette in het station zit. Interrupt 21h, subfunctie 36h krijgt de vrije ruimte op het station gespecificeerd in register DL. Als de informatie over de vrije ruimte wordt geretourneerd, stopt het programma met een ERRORLEVEL van 0. Met andere woorden, er doet zich een kritieke fout voor en de eigen fouthandler wordt aangeroepen.

Dit programma stelt een ERRORLEVEL van 1 in als het station is geopend en een 0 als er een diskette in zit. Het volgende gedeelte van een batchbestand laat zien hoe DRIVEOK in een batchbestand kan worden toegepast en hoe het standaardbericht van DOS kan worden omzeild:

```
@echo off
:START
driveaok
if errorlevel 1 goto message
rem voer hier de commando's in
echo A is OK
goto quit
:MESSAGE
echo plaats geformatterde schijf in A
pause
goto start
:QUIT
```

Met het volgende scriptbestand, DRIVEBOK.SCR, kan het programma DRIVEBOK.COM worden gemaakt dat controleert of er een diskette in station B is geplaatst:

```
n drivebok.com
a 100
mov ax,2524     ; interrupt handler 24
veranderen
mov dx,0113     ; nieuwe handler op adres 0113
int 21          ; roep interrupt
mov ah,36       ; vrije ruimte op schijf
opvragen
mov dl,02       ; station B
int 21          ; roep interrupt
mov ax,4c00     ; OK: errorlevel op 0 zetten
int 21          ; klaar
mov ax,4c01     ; mis: errorlevel op 1 zetten
int 21          ; klaar

r cx
18
w
q
```

Een script voor vaststellen startstation

Wanneer u een batchbestand maakt waarmee de startbestanden worden gekopieerd, aangepast of opgezocht, moet u het startstation aangeven. Ga er niet vanuit dat station C altijd het startstation is.

In het volgende scriptbestand, BOOTDRV.SCR, wordt met interrupt 21h, subfunctie 33h het startstation bepaald en het bestand BOOTDRV.COM gemaakt:

```
n bootdrv.com
a 100
mov ax,3305     ; vraag station op
```

```
int 21          ; roep interrupt
mov al,dl       ; schijf naar AL
mov ah,4c       ; klaar
int 21

r cx
B
w
q
```

Register DL wordt bijgewerkt met het startstation dat vervolgens als een ERRORLEVEL wordt geretourneerd. Station A is ERRORLEVEL 1, station B is ERRORLEVEL 2, enzovoorts.

Het volgende batchbestand, TESTBOOT.BAT, laat BOOTDRV.COM in actie zien:

```
@echo off
bootdrv
if errorlevel 4 if not errorlevel 5 echo
Gestart van D
if errorlevel 3 if not errorlevel 4 echo
Gestart van C
if errorlevel 2 if not errorlevel 3 echo
Gestart van B
if errorlevel 1 if not errorlevel 2 echo
Gestart van A
```

Samenvatting

Dit hoofdstuk laat zien hoe DEBUG voor allerlei doeleinden kan worden gebruikt en niet alleen voor het opsporen van bugs in programma's. Hierbij zijn de volgende zaken aan de orde gekomen:

- Met de opdrachten D (dump) en S (search) kunt u door programmabestanden en ASCII-bestanden bladeren. DEBUG kan elke byte in een bestand laten zien, ook de harde returns en de line feeds.
- U kunt met de opdracht E (edit) programma's bewerken zoals QBASIC.EXE en CHOICE.COM, zodat zij zich anders gaan gedragen. EXE-bestanden moeten echter eerst worden herbenoemd met een andere extensie.
- DEBUG is een ideaal programma voor het maken van kleine utility's waarmee batchbestanden kunnen worden uitgebreid.
- Programma's kunnen interactief of door het omleiden van scriptbestanden naar DEBUG worden gemaakt.
- Met de opdracht A (assemble) kunt u met behulp van de assembleertaal opdrachten invoeren.

In deel 4 van dit boek worden alle opdrachten en stuurprogramma's uitgebreid besproken.