Pascal uitgediept
Geheugenbenadering

MSX Computer & Club Magazine nummer 67 - mei 1994

Herman Post

Scanned, ocr'ed and converted to PDF by HansO, 2001

In deze aflevering wordt eens bekeken hoe direct het geheugen kan worden benaderd. Ook komt aan de orde wat absolute adressering nu precies is en hoe dit is te gebruiken. De mogelijkheden blijken veel groter als in BASIC.
Het geheugen van de computer is opgebouwd met geheugencellen van elk acht bits groot. Deze informatie is leuk, maar als Pascal gebruiker hebt u er weinig mee te maken. De compiler bepaalt zelf wel wat er op een geheugenplaats moet staan en of er voor bepaalde informatie meer dan één geheugencel nodig is. Wilt u echter een programma schrijven speciaal voor MSX, dan is het vaak handig om dezelfde geheugenplaatsen te gebruiken als het MSX systeem. Om dit te kunnen doen moet je wel over voldoende documentatie beschikken waarin het systeem staat uitgelegd. Als u nog niet over deze documentatie beschikt raad ik u aan om op zoek te gaan naar 'MSX ROM-BIOS handboek' of 'MSX handboek voor gevorderden'. In beide boeken vindt u informatie over het MSX1 systeem, maar ook MSX2 gebruikers kunnen deze gegevens toepassen.

Peek en poke

In BASIC zijn peek en poke de twee commando's om het geheugen direct te benaderen. In Pascal wordt een heel andere aanpak gebruikt. Het volledige geheugen wordt gezien als een voorgede-finieerd array van bytes, waarin elk arrayelement een geheugenplaats voorstelt. Door nu het juiste arrayelement te lezen of te schrijven benadert u die bijbehorende geheugenplaats.

Een voorbeeld:

Pascal	BASIC
a:=MEM[$8000]	A=PEEK(&H8000)
MEM[$8000] :=a	POKE(&H8000,A)

Dit is de eenvoudigste manier om direct toegang te krijgen tot het geheugen. Het is echter een nadeel dat op deze manier het geheugen alleen als een serie bytes is te gebruiken. Verschillende systeem-variabelen nemen namelijk twee bytes in beslag. Wilt u deze in BASIC gebruiken, dan zit u vast aan twee peek opdrachten. In Pascal is een toekenning aan een integer hiervoor de juiste benadering.

Directe adressering

Om twee bytes direct in een integer te kunnen opslaan is het nodig om die integer op een vaste plaats in het geheugen op te slaan. Als voorbeeld zal ik de systeem variabele jiffy nemen. De variabele is in BASIC beter bekend als time. Deze variabele wordt opgehoogd door een interrupt en zal dus vijftig of zestig keer per seconden wijzigen. Om de variabele jiffy als timer te kunnen gebruiken is de volgende declaratie nodig:

```
VAR timer:INTEGER ABSOLUTE $FC9E;
```

Het gevolg is dat de variabele timer nu in het programma gebruikt kan worden op dezelfde manier als iedere andere variabele. Deze declaratie kost u echter geen data. Als u deze declaratie in uw programma opneemt, en hem verder helemaal niet gebruikt zal uw code niet groter worden. U kunt op deze manier een heleboel systeemvariabelen in een include-bestand opnemen, en deze dan alleen gebruiken als u ze nodig hebt. U mag zelfs variabelen dubbel declareren. Wilt u ook nog een timer hebben die loopt van 0-255, dan declareert u:

```
VAR subtimer:BYTE ABSOLUTE $FC9E;
```

Hierdoor krijgt u een byte die 50/60 keer per seconden wijzigt. Nu zult u misschien denken "Ja, leuk hoor maar die timer kende ik al." die stond een aantal afleveringen terug ook al vermeld. Wel, dan heb ik hier goed nieuws. De structuur van Pascal maakt het mogelijk om niet alleen hele bytes te lezen, maar ook bits. Toegegeven, dit is een stuk moeilijker maar het kan. Ik heb als voorbeeld een programma geschreven dat het toetsenbord bekijkt, en op het scherm afdrukt welke toets er is ingedrukt. Het bekijken van het toetsenbord gebeurd door de systeemvariabelen newkey te bekijken. Dit zijn elf bytes, die elk voor een rij van de toetsenbord-matrix staan. Als er een toets wordt ingedrukt, wordt het bijbehorende bit ge-reset, ofwel op nul gezet. Dit gebeurt door de interrupt, dus u heeft er geen omkijken naar. Om nu te kunnen kijken of er een toets is ingedrukt is er een set gedeclareerd over de systeemvariabelen heen. Een set nu is altijd 32 bytes groot, maar omdat dit een set is van een genummerd type en dit genummerde type maar 72 elementen bevat, worden alleen de eerste 9 bytes gebruikt. Merk hierbij op dat de laatste twee bytes niet worden uitgelezen. In deze twee bytes staan de toetsen van het numeriek toetsenbord, maar omdat dit niet op alle MSX machines aanwezig is heb ik deze niet opgenomen. In deze listing is leuk gebruik gemaakt van het genummerde type. Let speciaal op de array-declaratie en de for-lus. De array naam is alleen opgenomen om de waarde van de set ook te kunnen afdrukken. Deze zult u in eigen programma's meestal niet nodig hebben. De namen binnen het genummerde type kunt u natuurlijk zo aanpassen dat u ze zelf gemakkelijk onthouden en binnen uw programma gebruiken kunt op de manier die voor de spatie in de laatste regel is gebruikt. Even een bedankje aan Jacco Kulman, die het idee voor dit programma aandroeg.
De set bevat alle toetsen behalve de
Toets -of toetsen -die is –zijn -ingedrukt. Door te kijken of een toets niet in de set voorkomt weten we of deze toets is ingedrukt. Hier geldt dus een omgekeerde logica, komt de toets niet in de set voor dan is de toets wel ingedrukt. Ik hoop dat er niet teveel

problemen ontstaan met het overtikken van de listing, al die komma's zijn een ramp voor de layout afdeling.

Variabel op variabel

Ook de variabelen, die op de normale manier zijn gedeclareerd, zijn dubbel te benaderen. Stel dat u een array van het type char heeft gedeclareerd.

karakter:ARRAY[0..100] OF CHAR;

Als u aan dit array waarden moet toekennen, die binnen het programma als byte bekend zijn kan dit via de normale Pascal methode:

karakter[x]:=CHAR[waarde];

Moet dit echter vaak gebeuren, dan is het handiger om er een tweede array overheen te declareren van het type byte. U krijgt dan:

karakter:ARRAY[0..100] OF CHAR;
alsbyte :ARRAY[0..100] OF BYTE ABSOLUTE karakter;

U kunt er nu een bytewaarde aan toekennen door alsbyte[x]:"waarde te gebruiken. Omdat de beide arrays op hetzelfde adres staan heeft het arrayele-ment van karakter nu dezelfde waarde als het array element van alsbyte, en dit zonder een type conversie. Een integer is op dezelfde manier te splitsen in twee bytes. Hiervoor declareert u eerst een integer en daarna hier overheen een array van twee bytes of omgekeerd. Het ziet er dan als volgt uit:

VAR hibyte,lobyte : BYTE;
geheel : INTEGER ABSOLUTE lobyte;

Let wel even goed op de volgorde van de bytes. Door aan geheel een waarde toe te kennen is deze direct gesplitst in een highbyte en een lowbyte. In het boek 'Turbo Pascal voor gevorderden' wordt verteld, dat deze methode sneller is dan het gebruik van HI (geheel) en LO (geheel). Dit is op MSX in ieder geval niet waar. Beide methodes zijn even snel. Om echter twee bytes samen te voegen tot een integer is deze methode wel sneller. Ik demonstreer hier de methode in de hoop dat u het principe begrijpt en dit ook toe kunt gaan passen op zelf gedefinieerde typen. Het principe blijft gelijk maar de uitvoering kan wat lastiger worden. Ik geef u hier het basisidee, u werkt het uit, experimenteert en leert al doende de fijne kneepjes van het Pascal vak.

```
PROGRAM Toets;

TYPE letter =
(nul,een,twee,drie,vier,vijf,zes,zeven,
 acht,negen,min,isgelijk,deellinks,rechtopen,rechtsluit,puntkomma,
 acculadeopen,acculadesluit,komma,punt,deelrechts,dodetoets,toetsa,toetsb,
 toetsc,toetsd,toetse,toetsf,toetsg,toetsh,toetsi,toetsj,
 toetsk,toetsl,toetsm,toetsn,toetso,toetsp,toetsq,toetsr,
 toetss,toetst,toetsu,toetsv,toetsw,toetsx,toetsy,toetsz,
 shift,ctrl,graph,caps,code,fun1,fun2,fun3,
 fun4,fun5,esc,tab,stop,backspace,select,return,
 spatie,home,ins,del,cursorl,cursorb,cursoro,cursorr);

CONST naam : ARRAY[letter] OF STRING[10]=
('0'   ,   '1'   ,   '2'   ,   '3'   ,   '4'   ,   '5'   ,   '6'   ,   '7'   ,
 '8'   ,   '9'   ,   '-'   ,   '='   ,   '\'   ,   '['   ,   ']'   ,   ';'   ,
 ''''  ,   '`'   ,   ','   ,   '.'   ,   '/'   ,'DODE TOETS', 'A'   ,   'B'   ,
 'C'   ,   'D'   ,   'E'   ,   'F'   ,   'G'   ,   'H'   ,   'I'   ,   'J'   ,
 'K'   ,   'L'   ,   'M'   ,   'N'   ,   'O'   ,   'P'   ,   'Q'   ,   'R'   ,
 'S'   ,   'T'   ,   'U'   ,   'V'   ,   'W'   ,   'X'   ,   'Y'   ,   'Z'   ,
'SHIFT','CTRL'   ,'GRAPH'  ,'CAPS'   ,'CODE'   ,  'F1'   ,  'F2'   ,  'F3'   ,
'F4'   ,  'F5'   , 'ESC'   , 'TAB'   ,'STOP'   ,'BS'     ,'SELECT' ,'RETURN' ,
'SPATIE','HOME'  ,'INS','DEL','CUR.<','CUR.^','CUR.V','CUR.>T');
VAR toetsen : SET OF letter ABSOLUTE $FBE5;
    telchar : letter;

BEGIN
  REPEAT
    FOR telchar:=nul TO cursorr DO
      IF NOT (telchar IN toetsen) THEN write(naam[telchar]);
  UNTIL NOT (spatie in toetsen);
END.
```