Pascal uitgediept
Hybride programmeren

MSX Computer & Club Magazine nummer 69 - augustus 1994
Herman Post

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

Programmeren in twee of meer talen is zelden noodzakelijk, maar biedt vaak interessante mogelijkheden. Dit artikel gaat in op de combinatie van snel te programmeren Pascal met snel werkende machinetaal.

Letterlijk betekent hybride bastaardachtig ofwel kruising uit verschillend ras of soort. In BASIC wordt het begrip hybride gebruikt, als er in het programma een machinetaalroutine is ingebouwd. In meer algemene zin wil het zeggen dat er een stuk code wordt gebruikt, dat door een andere programeertaal is verkregen. In de meeste gevallen zal het gaan om routines in machinetaal, omdat daarin de snelste code is te schrijven en de programmeur deze techniek gebruikt om in het programma snelheidswinst te behalen.

In Pascal zijn er twee mogelijkheden om machinetaal te gebruiken. De eerste methode werkt via het statement INLINE en de tweede via EXTERNAL. Aan de kant van de Pascal-programmeur is het verschil dat een inline vaak een onderdeel van een procedure of functie is, en dat EXTERNAL een volledige procedure of functie declareert. Als we de definitie van hybride letterlijk zouden nemen, valt het INLINE systeem buiten deze bespreking, maar omdat INLINE erg krachtig is en makkelijk te gebruiken, zal ik dit ook bespreken.

External

De methode die gebruik maakt van het statement EXTERNAL, wordt vooral gebruikt voor grote stukken machinetaal. Het nadeel van deze methode is dat de code op een vast adres in het geheugen moet staan. De programmeur moet er zelf voor zorgen dat het gebruikte geheugen niet door andere zaken wordt overschreven en het gebruiken van globale variabelen is alleen met erg veel trucs mogelijk. Wel is het mogelijk om parameters aan een routine door te geven. Dit gebeurt op dezelfde manier als Pascal parameters aan een procedure of functie meegeeft. Om dit uit te leggen, moet ik helaas veronderstellen dat u iets weet van machinetaal en opslagmethodes, omdat het te ver zou voeren er in dit artikel op in te gaan.
De aanroep van een external is precies hetzelfde als de aanroep van een gewone pascal procedure. Het verschil zit hem echter in de procedure- of functiedeclaratie. Na de identifier- en de parametersectie volgt het gereserveerde woord EXTERNAL, dat weer wordt gevolgd door het absolute adres van de ml-routine. Vanuit Pascal kan dan de procedure of functie worden gebruikt als een gewone Pascal routine.

Stack

Het doorgeven van parameters gebeurt via de stack. Bovenaan de stack staat het returnadres en lager in de stack— hoger in het geheugen—staan de parameters. De ml-routine moet dus eerst het returnadres en daarna de variabelen ophalen en tot slot het returnadres weer terugzetten.
Als het gaat om variabele parameters— gedeclareerd met VAR in de procedure heading—dan staat er eenvoudig een word van twee bytes op de stack met daarin het eerste absolute adres van de gebruikte variabele.
Gaat het om een waardeparameter— dus zonder VAR gedeclareerd—dan is het afhankelijk van het type variabele dat is gebruikt. Bij INTEGERS, BOOLEANS, CHARS en gedeclareerde scalaire types, wordt er een word op de stack gezet met de gewenste waarde.
Gaat het hierbij om een variabele, die slechts één byte groot is, dan zal het meest significante byte nul zijn.
Een REAL komt als zes bytes op de stack. Als we deze ophalen via de in-structie's POP HL, POP DE, POP BC, bevat L de exponent en H, E, D, C, B de mantisse. H bevat hierbij de minst significante en B de meest significante byte.
Een STRING staat letter voor letter op de stack, voorafgegaan door een lengtebyte. U haalt dus eerst de lengte op en dan alle letters. Een SET staat als 32 bytes op de stack. Hiervan is het byte op het laagste adres het minst significant.
Voor een POINTER staat er een word op de stack waarin het absolute adres van de dynamisch variabele is opgeslagen. Als het opgeslagen word de waarde nul heeft, verwijst deze naar de lege dynamische variabele NIL.
ARRAY's en RECORD's worden anders behandeld dan alle voorgaande type. Ook al gaat het hierbij om waardeparameters, dan worden deze toch niet op de stack geplaatst. Alleen het adres van de eerste byte van deze variabele komt op de stack terecht en het is dan de verantwoordelijkheid van de ml-routine dat deze de blokverplaats-routine uitvoert. Dit is gedaan om de benodigde stack niet te groot te laten worden.

Functies

Tot nu toe was het niet nodig om resultaten terug te geven. Als de ml-routine echter een functie is, moeten de resultaten wel worden teruggegeven. Dit gebeurt gedeeltelijk via de stack en gedeeltelijk via de registers. Als het gaat om een scalair type van een byte, dan moet de waarde staan in register L, gaat het om een scalair type van twee bytes dan moet de waarde staan in register-paar HL. Een REAL moet op dezelfde manier in de registers staan als dat ik ze hierboven heb uitgelezen, dus L moet de exponent bevatten, en BCDEH de mantisse, waarvan B het meest significant is. Strings en sets moeten via de stack worden teruggegeven, op dezelfde manier als dat ze worden aangeboden wanneer u ze ophaalt. ARRAY's en RECORD's kunnen niet als waarde van een functie worden teruggegeven. Dit zal dus altijd via een variabele parameter moeten gebeuren.

Inline

Voor kleine ml-routines is een inline vaak veel handiger in het gebruik dan de external-methode. Met de inline kunnen de instructies in machinecode direct worden opgenomen in de source. Als dan ook nog de mnemonics in een commentaarregel

worden geplaatst, blijft de code begrijpelijk voor mensen die iets van assernbly language weten. Het nadeel van INLINE is dat de mnemonics met de hand moeten worden vertaald naar de juiste opcodes en dat ook de 'jump'-afstanden uitgerekend dienen te worden.
Een inline statement bestaat uit het gereserveerde woord INLINE, gevolgd door een of meer code elementen gescheiden door slashes (/) en tussen ronde haken. De code-elementen worden opgebouwd uit één of meer dataele-menten, gescheiden door een plus- of rninteken. De code elementen worden vertaald naar een byte of een word en zo in het programma opgenomen.
Ik ben me er van bewust dat dit erg formeel is, maar het bevat wel de volledige beschrijving van inline. Bovendien wordt erin benadrukt, dat alle elementen, die gescheiden worden door slashes, maximaal twee bytes bevatten. Als binnen een inline een identifier wordt gebruikt—een identifier is de naam van een variabele, procedure of functie— zal op die plaats het adres van die identifier worden opgenomen. Het aanroepen van een bestaande procedure of functie wordt dan:

```
PROCEDURE DoetNiets;
  BEGIN
  END;

PROCEDURE InlineVoorbeeld;
  BEGIN
     INLINE($CD/Doetniets); (
     CALL doetniets     )
  END;
```

Dit is natuurlijk een erg zinloos voorbeeldje, omdat ook alleen de naam van de procedure, zonder inline, hetzelfde effect zou hebben. Dit voorbeeld gene-
reert wel drie bytes ml-code. Nu rijst de vraag wanneer er één byte en wanneer er twee bytes worden gegenereerd. Als de waarde van een codeëlement binnen het bereik van 0.. .255 valt, zal de compiler één byte genereren. Dit lijkt op het eerste gezicht erg gemakkelijk, omdat de compiler zelf kijkt of een byte groot genoeg is om de waarde in op te slaan en anders een word ervoor kiest. Maar als bijvoorbeeld een adres wordt opgegeven, zal dit altijd een word moeten zijn. Zou de berekende waarde dan toevallig binnen het bereik vallen, dan gaat er van alles mis, omdat er maar één byte wordt gegenereerd. Daarom bevat het inline statement twee extra opties, die deze automatische keuze uitschakelen. Zet u voor een codeëlement het 'kleiner dan'-teken (<), dan wordt een byte gegenereerd en zet u er een 'groter dan'-teken (>) voor, dan wordt er een word gegenereerd. Bij de regel:

```
INLINE(<$1234/<$5678);
```

wordt de code: $34,$78. Bij de regel:

```
INLINE(>$88/>$23);
```

wordt de code : $88,$00,$23,$00.

Met wat er tot nu toe over inline is gezegd, is het alleen mogelijk om relatieve code toe te voegen. Omdat we niet weten op welk adres de inline begint, zou alle code

relatief moeten zijn, zodat alle sprongen zeker op het juiste adres aankomen. Om dit te omzeilen, is er de mogelijkheid om met de asterisk de program counter als waarde op te nemen. We kunnen dan uit het huidige adres berekenen naar welk adres een sprong moet worden uitgevoerd, of op welk adres een variabele staat.

UpperCase

Als voorbeeld staat een routine afgedrukt die alle letters in een string omzet naar hoofdletters. De informatie uit dit artikel en dit voorbeeld zijn afkomstig uit de reference manual van turbo pascal versie 3.0. Ik geef toe dat ik niet de juiste persoon ben om uit te leggen wat u in de external en de inline allemaal kunt maken, omdat machinetaal niet mijn sterkste kant is. Vandaar dat ik mij in dit artikel heb beperkt tot het implanteren en vanuit Pascal aanroepen van ml-routines. Het bespreken van de mogelijkheden die u in ml kunt gebruiken, laat ik graag over aan andere specialisten.

```
{ TYPE str=STRING[255] }

PROCEDURE UpperCase(VAR strg: str);
BEGIN
  INLINE($2A/strg/    {       LD  HL,(strg) }
         $46/         {       LD  B,(HL)    }
         $04/         {       INC B         }
         $05/         { L1:   DEC B         }
         $CA/*+20/    {       JP  Z,L2      }
         $23/         {       INC HL        }
         $7E/         {       LD  A,(HL)    }
         $FE/$61/     {       CP  'a'       }
         $DA/*-9/     {       JP  C,L1      }
         $FE/$7B/     {       CP  'z'+1     }
         $D2/*-14/    {       JP  NC,L1     }
         $D6/$20/     {       SUB 20H       }
         $77/         {       LD  (HL),A    }
         $C3/*-20);   {       JP  L1        }
END;                  { L2:   EQU $         }
```