Pascal uitgediept Snel, sneller, snelst

Herman Post
MSX Computer & Club Magazine nummer 65 - februari 1994

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

Ondanks de snelle code die Pascal aflevert, is het wel mogelijk om de snelheid nog wat op te voeren door snelle routines te schrijven. Hoe? dat leest u hier.

In de meeste programma's is het niet nodig om een enorme snelheid in de code te verkrijgen. Maar in enkele gevallen is juist die snelheid van essentieel belang. Ook komt het voor dat een programma niet snel hoeft te zijn maar juist enkele routines wel zo snel mogelijk moeten worden uitgevoerd. Niemand is erg blij als hij bijvoorbeeld lang moet wachten op de uitdraai van een prijslijst uit de printer terwijl tijdens het invoeren van de gegevens er weinig mensen de snelheid van de computer bij kunnen houden met typen. In zo'n geval moet dus de printer routine erg snel zijn en kan tijdens de invoer routine de tijd worden genomen voor bijvoorbeeld foutcontrole.
Nu is het al lang bekend dat er vele wegen naar Rome leiden, en dat die wegen soms goed en soms slecht zijn. Zo kunt u dus voor een probleem meerdere routines of oplosmethodes schrijven en er dan de snelste of de meest complete uitzoeken. Maar wat is de snelste routine. Zeker als het gaat om korte routines, is het vaak moeilijk te zien wat het snelste is. Als die korte routinein het programma dan regelmatig wordt gebruikt kan hij toch een behoorlijke vertraging tot gevolg hebben.

Een klein voorbeeldje. Het komt erg vaak voor dat een variabele met één moet worden opgehoogd. De constructie is natuurlijk eenvoudig: a:=a+1; Dit ziet er duidelijk uit en iedereen begrijpt wat er gebeurd. De constructie a:=SUCC(a); doet echter precies hetzelfde, maar is wel sneller en bovendien levert het nog een kortere code op ook. Toegegeven, het is minder duidelijk, maar bij snelle routines vind ik dat minder belangrijk en kan er altijd nog in een regeltje met wat commentaar bij worden gezet wat er precies gebeurd. De hierboven gemaakte constructie kan natuurlijk ook nog worden uitgebreid. a:=a+2; komt overeen met a:=SUCC(SUCC(a)); Ook hier is de tweede methode, ondanks dat de functie SUCC twee keer wordt aangeroepen, sneller als de eerste methode. Dit zogenaamde 'nesten' van deze functie blijft sneller tot en met a:=a+6;. De versie met de geneste SUCC functies die hetzelfde doet wordt er echter volledig onleesbaar door. Telt u mee:

a:=SUCC(SUCC(SUCC(SUCC(SUCC(SU CC(a))))));

Zoals u ziet volledig onleesbaar, maar nog steeds sneller. Bij a:=a+7; ligt het omslagpunt. Dit is met de SUCC constructie langzamer.

Dit voorbeeld staat natuurlijk niet alleen. Er zijn zo talloze routines te verzinnen die op snelheid kunnen worden getest. Om te kijken welke routine het snelste is heb ik een

programma opgenomen die dit voor ons doet. Voor elke routine die u uitprobeert moet u dit programma aanpassen. Het is erg veel werk om op deze manier een heel programma te versnellen, maar voor een losse routine loont het de moeite.

```
PROGRAM TestSnelheid;

CONST maxcount = 30000;

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

PROCEDURE een;

VAR a:BYTE;

BEGIN
  a:=0; a:-a+1
END;

PROCEDURE twee;

VAR a:BYTE;

BEGIN
  a:=0; a:=SUCC(a)
END; '

BEGIN
  WRITELN('Testprogramma procedures');
  WRITELN('de routine wordt ',maxcount,' keer
uitgevoerd.');
  WRITELN('de eerste test loopt....');
  INLINE($FB/$76); {wacht op interrupt}
  timer:=0;
  FOR teller:=0 TO maxcount DO     een;
  WRITELN('de test heeft '.timer.' interrupts geduurd.');
  WRITELN('de tweede test loopt....');
  INLINE($FB/$76); {wacht op interrupt!}
  timer:=0;
  FOR teller:=0 TO maxcount DO twee;
  WRITELN('de test heeft '.timer.' interrupts geduurd.')
END.
```

Werking

Hoe werkt het programma nu precies. In eerste instantie ziet alles er nogal simpel uit. Een

paar writein statements en twee lussen met een procedure aanroep. Dat is eigenlijk alles. Alleen de INTEGER ABSOLUTE en de twee INLI-NE regeltjes vragen om wat extra aandacht. Bij het schrijven van een ben-chmark (zo heten snelheids vergelij-kings programma's) moeten we rekening houden met de tijd die de computer zelf opslokt. Deze tijd wordt gebruikt voor de interrupt routine. De interrupt is een signaal aan de processor dat er eerst een systeem routine moet worden uitgevoerd. Een interrupt wordt gegeven door een randapparaat en kan dan ook vanaf veel verschillende delen van de computer afkomstig zijn. Bijvoorbeeld uit een cartridge. In een standaard msx computer kan echter alleen de videoprocessor een interrupt afgeven. Deze doet dan ook 50 of 60 keer per seconden. (Dit is ook de instelling die vaak leidt tot het 'lopen' van het beeld op een televisie) Op het moment dat er een interrupt optreedt zal de processor ophouden waarmee hij bezig was, en eerst de interrupt routine uitvoeren. Dit houd onder andere in het lezen van het toetsenbord. Nu is het mogelijk om die interrupt uit te zetten en dan gewoon met het programma door te gaan. (Om precies te zijn wordt niet de interrupt uitgezet maar krijgt de processor opdracht om niet meer op een interrupt te reageren.) Deze mogelijkheid is echter de minst goede voor een benchmark omdat de tijd die nodig is voor de interrupt dan vervalt terwijl deze normaal wel aanwezig zal zijn.
Bovendien heb ik de interrupt nodig. Tijdens de interrupt wordt er namelijk ook een variabele in het werkgebied opgehoogd. Deze variabele (ASCII naam=jiffy) heb ik timer genoemd, en deze wordt dus 50 keer per seconden opgehoogd. Deze gebruik ik als een stopwatch om de tijd bij te houden. Ik kijk na het uitvoeren van de routine hoe vaak deze timer is opgehoogd, en geef dit op het scherm weer. De gebruikte inline zorgt er voor dat de routine altijd direct na een interrupt gestart wordt. De reden hiervoor staat getekend in het schema rechtsboven. In beide gevallen wordt gedurende een klok-minuut om de 8 seconden een signaal afgegeven. Afhankelijk van het startmo-ment zijn dit 7 of 8 interrupts in die ene minuut.

Het verschil is maximaal 1

Als de interrupt op ieder moment kan optreden, kan bij het uitvoeren van dezelfde lus een verschillend aantal keren de interrupt optreden. Om dit te ondervangen wordt aan het begin van de lus gewacht tot er een interrupt optreedt. Dit gebeurt met de INLINE. De lus zal hierdoor dus gesynchroniseerd worden met de interrupt, en de afwijking van één interrupt is dan opgeheven.

Ik heb in de procedure de toekenning a:=0; moeten opnemen omdat anders de variabele a een niet gedefinieerde waarde heeft, en dus boven de 255 zal komen. Omdat ik deze toekenning in beide procedures heb gezet heeft hij geen invloed op het tijdsverschil dat wordt gemeten. Het heeft natuurlijk wel invloed op de totaaltijd die gemeten wordt. Bij het testen van eigen routines moet u hier ook op letten.

Nu nog een opmerking over maxcount, de constante die boven in het programma is opgenomen. Ik heb deze nu op 30000 staan en dit is voor deze routines een goede waarde. Als u nog geen idee hebt hoe lang u routine duurt kunt u deze het beste op een lage waarde instellen omdat het met dit getal erg goed mogelijk is om een testprogramma te schrijven dat erg lang bezig is. Dit zou niet zo'n probleem zijn, maar de timer kan

maximaal opgehoogd worden tot $FFFF. Bovendien krijgt u een negatieve waarde terug als het getal boven de $7 FFF komt. Dit gebeurd na ongeveer 10 minuten. Het is echter niet zinvol om zo lang op een testroutine te wachten.

Lengte van de code

Tot zover heb ik het alleen nog maar gehad over de snelheid van het programma. In het begin van het verhaal heb ik ook gezegd dat het korter in uiteindelijke programma code is. Dit bekijk ik echter op een heel andere manier. Ik neem beide routines onder dezelfde naam op in mijn programma, maar ik maak één van de twee inactief door er remark tekens omheen te plaatsen {} of (**). Dan compileer ik het programma naar COM-file en schrijf op wat het vrije geheugen is. Nu maak ik in de code de andere routine actief en zet de eerste tussen remark tekens en compileer opnieuw naar een COM-file. Het verschil tussen de vrije ruimte die er nu over is en de vrije ruimte die er bij de eerste routine is, geeft het verschil aan in programma code.

7 MHz

Ook kunt u met dit programma de werking van uw 7 MHz bestuderen. U zult zien dat de routines meer dan twee keer zo snel worden. Hoe kan dit als de computer's frequentie precies twee keer zo hoog wordt? Het antwoord is eenvoudig. De interrupt wordt ondanks de 7 MHz toch 50 keer per seconden uitgevoerd. Omdat de interrupt routine op zich ook twee keer zo snel wordt uitgevoerd blijft er dus per 50e seconden meer tijd over voor ons eigen programma. Het gevolg is dat ons programma meer dan twee keer zo snel is geworden.

50/60 Hz

Een tweede misvatting die ik hier recht wil zetten is het idee dat de computer sneller zou worden als er wordt overgeschakeld naar 60 Hz. Het overschakelen naar 60 Hz zorgt ervoor dat het beeldscherm niet 50 maar 60 keer per seconden wordt opgebouwd. Het programma wordt er niet sneller door. integendeel zelfs. De videoprocessor genereert namelijk zijn interrupt bij een bepaalde beeldlijn. Omdat het scherm nu 60 keer per seconden wordt opgebouwd zal dus ook die beeldlijn 60 keer worden geschreven, en zal er dus ook 60 keer per seconden een interrupt worden gegeven. Die interrupt zorgt ervoor dat de interrupt routine wordt aangeroepen, en die wordt dus ook 60 keer uitgevoerd. De computer is dus per seconden 10 keer de tijd van de interrupt routine extra kwijt, en die gaan ten koste van de snelheid van uw eigen programma.

Tot zover de informatie van deze keer.
Het is erg leerzaam om met dit programma u eigen routines te versnellen en te optimaliseren. .Bovendien is het erg leuk om op deze manier met een taal bezig te zijn. Ik hoop dan ook dat u er gebruik van maakt.