MSX

MSX CLUB BASIC COMPILER

SOFTWARE

MSX
CLUB
BASIC
COMPILER

HANDLEIDING VOOR HET GEBRUIK

Bedankt Vivian, zonder jouw gedogen was deze handleiding nooit klaar gekomen.

Inhoudsopgave

Inleiding

Van harte

Gefeliciteerd met uw aanschaf. Met MCBC zult u uw programma's nog beter (= sneller) kunnen laten werken. Leer snel met MCBC om te gaan en het programmeren ermee zal u dan spoedig veel genoegen verschaffen.

Gemakkelijk machinetaalsnelheid

De door u in MSX-BASIC geschreven programma's zullen door MCBC vele malen sneller gaan. Ook hoeft u niet meer allerlei zaken over de interne opbouw van de MSX-computer te kennen, die wel noodzakelijk zijn als u zelf geheel of gedeeltelijk in machinecode wilt gaan programmeren. Als u direct in Z80-code programmeert zult u de MSX en zijn opbouw goed moeten kennen.

Maar ook met behulp van assembly language zult u goed op de hoogte moeten zijn van vele MSX-zaken. U zult echter met MCBC als gevorderd basic-programmeur vrij eenvoudig gebruik kunnen maken van de memorymapper. Hierdoor is het mogelijk in basic een programma te schrijven dat nooit in zijn geheel in het basic deel van het geheugen van uw MSX past.

Ergert u zich ook altijd zo aan de mededeling na opstarten van zo'n slordige 23 duizend bytes vrij als u een machine met 128 Kbyte of zelfs 256 Kbyte heeft ? Dit hoeft nu niet meer, hak het programma in geschikte stukken en compileer de delen afzonderlijk.

Zet de machinetaalcode in de verschillende delen van de memorymapper en verbindt ze met elkaar door middel van een eenvoudig basicprogramma en draaien maar.

Over de handleiding

In deze handleiding zijn er een aantal verschillende onderdelen te onderscheiden.

Eerst wordt in het kort uitgelegd wat een compiler is en wat u er mee kunt doen. En wat veel belangrijker is wat u er niet mee kunt doen. Vervolgens zullen we wat dieper ingaan op de diverse (on)mogelijkheden van de MCBC om zo tot het hart van deze handleiding in de vorm van de gebruiksaanwijzing te komen.

Dan een toelichting op de diverse demonstratie programma's. Tot slot volgt dan nog een paragraaf met technische gegevens en een lijst met problemen en mogelijke oplossing.

Back up maken

We raden iedereen aan om zo spoedig mogelijk een backup te maken van de geleverde schijf. U kunt MCBC zo vaak als u wilt voor persoonlijk gebruik copiëren. Doorgeven van MCBC aan derden is echter niet toegestaan. Programma's die met behulp van MCBC gemaakt werden mogen echter wel op niet commerciële basis aan derden worden verspreid. Heeft u commerciële plannen met het door u met behulp van MCBC gemaakte produkt neem dan contact met ons op voor een licentie.

Gebruikte termen

Mochten er voor u onbekende termen gebruikt worden kijk dan achterin de handleiding in de lijst met gebruikte termen.

In deze lijst worden alle termen in het kort verklaard, in de handleiding zelf wordt uitgebreider op de begrippen ingegaan.

Medewerking gezocht

Wij, van de MSX-Club, hopen dat u de resultaten niet alleen voor uzelf zult houden maar, als deze resultaten ook voor anderen geschikt zijn, ze aan ons zult opsturen. Is het inderdaad voor ons bruikbaar materiaal dan zullen wij het zeker helpen verspreiden en..... u kunt ook er ook iets voor terug verwachten. Naast de waardering van uw mede MSX'ers kunt u ook het aanschafbedrag voor MCBC terug 'verdienen'. Als u liever iets anders wilt voor uw moeite en de club kan u daar aan helpen zal dat zeker geregeld kunnen worden.

Als redactie van het MSX Club Magazine zouden wij ook graag artikelen over het gebruik van MCBC en tips voor het gebruik willen publiceren. Bent u van plan in die geest iets te schrijven neem dan even contact op met de redactie om doublures en de daaruit volgende teleurstelling te voorkomen.

Succes met het werken met MCBC.

Compiler algemeen

Wat is een compiler ?

Een compiler is een programma dat een programma, geschreven in een hogere programmeertaal, omzet in een programma in machinetaal, dat dan op een later tijdstip kan worden uitgevoerd.

Dat weet ik al

Weet u al wat een compiler is en kan dan kunt u de volgende bladzijden overslaan. Weet u dat niet of weet u niet goed wat u kunt verwachten, lees dit stuk dan door omdat het u veel ergernis zal besparen bij pogingen (principieel) onmogelijke dingen van de compiler te verlangen.

Een in een hogere programmeertaal geschreven programma is in principe onafhankelijk van de computer waar het programma op moet worden uitgevoerd. Sterker zelfs; het kenmerk van een hogere programmeertaal is juist dat onafhankelijk van de processor zijn. Een compiler echter wordt juist geschreven om deze hogere programmeertaal om te zetten in een code voor een bepaalde processor. Een compiler zal derhalve geschreven worden voor gebruik op een specifieke processor en in de praktijk meestal op een specifieke computer. In de rest van dit deel van het verhaal zal ik BASIC zeggen maar bijna alles zal ook opgaan voor andere hogere programmeertalen.

Processor onderdeel computer

Ik gebruikte hiervoor zowel het woord processor als het woord computer. Dit zijn echter verschillende begrippen al zijn beide in dit geval toepasselijk. Een processor is als het ware het hart of voor de meer pragmatischen onder u, de motor van het hele systeem. Tegenwoordig is de processor bijna altijd als één chip uitgevoerd die we de processor noemen. We spreken over een processing unit als hij uit meerdere componenten bestaat. Een processor zoals de Z80 die in de MSX-computer zit schrijft voor hoe de code eruit moet zien om voor hem begrijpelijk te zijn. Maar voor de processor begrijpelijk betekent nog niet automatisch voor de computer uitvoerbaar. Ook de andere componenten moeten 'meewerken' al was het maar door aanwezig te zijn.

In de computer zit meer dan alleen een processor, het aanwezige geheugen speelt bijvoorbeeld een belangrijke rol. Een compiler die alleen maar BASIC omzet naar Z80-code zal dan ook lang niet alle faciliteiten die een computer biedt benutten.

Niet alle faciliteiten

In de praktijk van de homecomputers blijkt het meestal een illusie te menen dat een compiler alle mogelijkheden kan gebruiken die op de computer aanwezig zijn. En denk nu niet dat compilers voor andere computers dan homecomputers dit bezwaar niet kennen. Alleen ligt het daar wat anders. Op de mini's en mainframes zijn er vaak geen leuke extra's en de compiler kan dan misschien wel alle mogelijkheden benutten, maar niet doordat de compiler zo goed is, maar doordat de algemene taal in deze omgeving zo beperkt is. Zo zal ondersteuning van grafische mogelijkheden bijna altijd volledig ontbreken. De compilers die op de markt zijn gaan meestal uit van een algemeen BASIC en een algemene Z80. Een compiler kan dan snel en betrouwbaar voor een bepaalde computer vervaardigd worden.

Voor de gebruikers van zo'n compiler betekent dat echter dat juist al de leuke extra's die uw computer heeft niet gebruikt kunnen worden. En juist die extra's die uw computer onderscheidt van andere, in uw ogen mindere, machines was destijds de reden van aanschaf. Juist doordat een fabrikant zijn machine zoveel extra's heeft meegegeven, kan alleen een zeer kaal en simpel programma gecompileerd worden.

Geheugentekort

Voordat iemand het volgende verkeerd begrijpt, het is in het algemeen zo maar geldt **juist niet voor MCBC !**

Voor een volledige ondersteuning naar een code voor algemeen gebruik zijn r meestal voor homecomputers vrij veel geheugenproblemen. Een vrij simpel en klein programma dat in BASIC zo ongeveer 1 Kbyte inneemt zal in gecompileerde machinetaalcode al snel vele tientallen Kbyte groot zijn. Dit komt door een enorme overhead aan routines. De meeste compilers plaatsen bij 'onze code' voor de zekerheid alle (sub-)routines of die nu nodig zijn of niet. Dit komt voor veel computers neer op bijna de volledige inhoud van de BIOS-ROM's. Tevens is de opbouw van de machine in kwestie voor bijvoorbeeld het beeldscherm zodanig afwijkend dat een algemene code op processorniveau haast bij voorbaat al is uitgesloten. Vandaar dat compilers meer en meer geschreven worden niet alleen voor gebruik van het resultaat op processor maar op computer. Dit levert zoals besproken een aantal voordelen :

We kunnen als het geheugen gebruik dit toestaat de routines die in de BIOS-ROM's staan gebruiken zodat de gecompileerde versie veel kleiner kan zijn.

Omdat voor een speciale computer werd geschreven kan gebruikt gemaakt worden van alle hardware en software mogelijkheden van de computer.

Zeker bij homecomputers waar immers vrij veel extra's voor geluid en grafische mogelijkheden zijn opgenomen kan dit de gecompileerde code veel meer de volledige machine laten gebruiken.

Compiler is géén snelle interpreter

Als u van een compiler in het algemeen onmogelijke zaken verlangt zal het resultaat een lange reeks foutmeldingen zijn en geen of een niet bruikbare code opleveren. Elke compiler eist dat u een syntactisch juiste source (broncode) levert die dan gecompileerd kan gaan worden. Zitten er in de source al fouten kan er vanzelf nooit een correcte machinecode gegenereerd worden.

Pas op sommige sourcecodes lijken goed omdat zij door een interpreter wel gebruikt kunnen worden. Dit is echter geen garantie voor correct zijn van de source. Het omgekeerde is normaal wel waar : als een source niet of niet juist werkt zal een de gecompileerde versie dat ook niet doen. Het grootste gevaar schuilt echter in verschillende uitkomsten van de beide versies, de source bij de interpreter en de gecompileerde versie daarvan, de zogenaamde object code. Beide werken op het eerste gezicht correct, maar blijken wel verschillende resultaten te geven. Wees altijd verdacht op zulke mogelijkheden. Is dit overigens zo bij een van uw programma's vertrouw dan als regel de uitslag van de object code. De syntax-controle is in dat geval veel strenger. Pas uw programmeren indien nodig aan op overzichtelijk en modulair programmeren.

Modulair programmeren

Veel van de fouten bij programmeren met een interpreter komen vanzelf aan het licht als we streng modulair programmeren. Ik geef alvast wat voorbeelden van fouten die bij gebruik van een interpreter niet aan het licht komen en bij een compiler wel.

Ik kom er bij het hoofdstuk Vragen en antwoorden op terug.

- Bij elke programmalus mag er altijd maar een beginpunt en een eindpunt zijn.

- Een subroutine mag eigenlijk maar op één plaats verlaten worden en ook bij voorkeur maar op één plaats binnengegaan worden.
- De verschillende onderdelen van het programma moeten in een logische opbouw op elkaar volgen.

Voor veel hogere programmeertalen geldt dat de programmeur door de regels van de taal gedwongen wordt zo te programmeren. Bij BASIC is dit echter niet zo. Dit heeft voor fantasierijke programmeurs voordelen maar voor de zwakkere programmeurs betekent het vaak het produceren van spaghetticode.

Slierten zonder duidelijk begin en eind en dat lust een compiler niet.

Fouten bij compileren

Bij het compileren zult u vaak foutmeldingen krijgen. Er blijken twee soorten te zijn : de eerste is vrij onschuldig en u zult ze vaak tegenkomen; u maakte een typefout, vergat een haakje, zette er een teveel of iets dergelijks. Elke compiler zal u op deze fout wijzen en na de verbetering van de fout is het probleem weg. De andere fout is lastiger. Er kan niet gecompileerd worden omdat de compiler de structuur van uw programma niet begrijpt. Zoek de fout dan het eerst bij het niet voldoende modulair opzetten van het programma.

Fouten na compilatie

Ook na compilatie blijken sommige programma nog steeds niet correct te werken. Als de normale versie voor compilatie nog wel werkte zal vrijwel altijd de fout gevonden kunnen worden in de structuur van het programma. Op het opsporen van deze fouten kom ik later nog terug.

De compiler MCBC

Bij het schrijven van MCBC is er gewerkt met de volgende uitgangspunten in gedachten.

Als allerbelangrijkste :

- De bedoeling is het om programma's te schrijven in MSX-BASIC en dan met vrijwel het tempo van machinetaal op de MSX te laten draaien.

en zoals van een echte compiler mag worden verwacht

- de runtime versie moet zonder compiler of andere software hulpmiddelen kunnen werken.

Als hoofduitgangspunten :

- Snelheid, snelheid en snelheid. Waar een keuze gemaakt moest worden is altijd gekozen voor het snelste eindprodukt ook al beperkt dit gebruikersgemak en toepassingsmogelijkheden.
- De compiler zelf mag geen geheugenruimte in beslag nemen die door BASIC kan worden gebruikt.
- De object code die na compilatie verkregen wordt mag geen geheugenruimte in beslag nemen die door BASIC kan worden gebruikt.

Als direct gevolg van voorafgaande punten :

- De compiler zelf moet in een memorypage passen en zal dientengevolge maximaal 16 Kbyte groot mogen zijn.

Verder hield Adriaan rekening met :

- De compiler zelf moet in elke MSX-2 met memorymapper gebruikt kunnen worden.
- De object code dient naast snel vooral ook klein te zijn.
- Om een hoge snelheid te halen wordt geen runtime error-handling ingebouwd. MCBC gaat er van uit dat de te compileren source correct is. Goede (of luie) programmeurs kunnen hier soms slim gebruik van maken.
- Wordt in de source een fout gevonden zal de fout gemeld worden en de compilering wordt gestaakt.
- Vooral de grafische kant van de MSX-computer, vooral de MSX–2, moest ondersteund worden.
- In verband met de keuze voor snelheid en grafische zaken is gekozen voor alleen integers als getallen.

En tot slot moest gelden :

- De normale goede BASIC-programmeur moet zonder veel rompslomp gebruik kunnen maken van MCBC.

Opzet MCBC

In het geval van MCBC is de hogere programmeertaal een deel van de MSX-BASIC en de gegenereerde machinetaalcode een code voor de Z80 processor die toegang moet hebben tot de zogenaamde BIOS-routines in de ROM's van de MSX computer. Dit laatste lijkt een behoorlijke beperking, maar is in feite niets anders dan een slim gebruik maken van de inhoud van de ROM's in de MSX-computer. Wilt u de door MCBC gegenereerde object code op een andere computer met Z80-processor laten werken dient u de inhoud van de BIOS voor een deel mee te nemen. Na deze waarschuwing hierover zal ik er niet verder op ingaan omdat het niet in de opzet van de compiler ligt om algemene Z80 code te maken.

De object code kan vanzelfsprekend op een duurzaam medium, zoals diskette, worden weggeschreven. Als het programma later gebruikt moet gaan worden, zal u het moeten laden op de juiste plaats in het geheugen en dan starten. Bij deze beide onderdelen moet u enige kennis hebben van de geheugenopbouw en geheugengebruik van uw MSX-computer. Heeft u die kennis niet of niet voldoende dan verwijs ik u hiervoor naar de vele artikelen die hierover in MSX Club Magazine en andere tijdschriften en handboeken zijn verschenen. Bekijk ook de meegeleverde voorbeeldprogramma's. De compiler of beter gezegd het hulpprogramma CONTROL kan u meestal een aantal van de benodigde gegevens verschaffen. Ook deze handleiding zal vermoedelijk voor normaal gebruik voldoende informatie kunnen verschaffen om MCBC te kunnen gebruiken.

Beperkingen

- Er wordt een machinetaalprogramma door MCBC vervaardigd dat slechts in bepaalde delen van het geheugen van de MSX-computer kan werken. Dit zijn niet de delen van het geheugen die door uw BASIC-programma worden gebruikt.
- MCBC gaat uit van een correcte source, zijn er fouten zoals 'PLINT' in plaats van 'PRINT' zal de compilatieslag eindigen met een normale foutmelding. Worden getallen bij berekeningen tijdens de werking van het programma echter te groot zodat we bij de interpreter een 'Overflow in rnr' foutmelding krijgen, zal het object code programma gewoon doorgaan.
- Een object code zal zonder maatregelen van te voren NIET te onderbreken zijn.
- Niet de volledige MSX-BASIC wordt ondersteund wat wel ondersteund wordt leest u in de volgende paragraaf.

Ondersteuning in de Msx Club Basic Compiler :

Bij geen enkele compiler zal ooit de volledige MSX-BASIC gecompileerd kunnen worden omdat een aantal van de instructies direct betrekking heeft op de normale interpreter verwerking van MSX-BASIC. Ook zijn er instructies die niet zinvol gecompileerd kunnen worden. RENUM en LIST zijn, om een tweetal voorbeelden te noemen, vrij onzinnige instructies in een gecompileerde code. Er kan niets meer gelist worden omdat de BASIC nu is omgezet in Z80-code en omdat daarmee ook de regelnummers, sterker, zelfs de regels verdwenen, kan ook bezwaarlijk worden hernummerd.

Een lijst van de wel te gebruiken BASIC-statements, rekentekens en variabelentypes staat in de volgende paragraaf.

Ondersteund worden :

variabelen van het type integer (!)
dat zijn gehele getallen zonder breukdeel, dus geen cijfers achter de decimale komma/punt.
De grootte ligt tussen minimum -32 768 en maximum 32 767.

variabelen van het type string ($)
dat zijn teksten. Zij kunnen wel getallen voorstellen, maar die worden dan nu als tekst met cijfers opgevat.

Eén dimensionale arrays zowel van tekst als integertype

De rekentekens

> voor groter dan
= voor gelijk aan
< voor kleiner dan

en de combinaties hiervan >=, <=, <>

Pas op !
De in feite overbodige mogelijkheden =>, =< en >< kunnen niet gebruikt worden.

\+ voor optellen
\- voor aftrekken
* voor vermenigvuldigen
\ voor delen.....PAS OP !
Dit bent u waarschijnlijk niet gewend.
en * heeft hogere prioriteit dan \
\- om het tegengestelde van een waarde te krijgen.

Nu volgt een lijst met de BASIC-statements die gebruikt kunnen worden en in het kort waar ze voor dienen. In deze lijst staan x en y voor een of andere getalwaarde gebruikt die u zelf als getalconstante dient in te vullen. a,b, enz. voor een variabele die u zelf met naam kunt aangeven. Natuurlijk kunnen in dit geval ook constanten of expressies gebruikt worden. Met t$ geef ik een tekstvariabele aan die overigens altijd ook als constante kan worden opgegeven. Voor een regelnummer wordt de term rnr gebruikt. Voor de duidelijkheid nog even dit : staan in lijst x en of y genoemd betekent dit dat er geen variabele of expressie mag worden ingevuld. Het is echter niet de bedoeling dat ik hier een complete syntax beschrijving van de diverse BASIC-statements ga geven.

Zie handboeken

Bij voorbaat verontschuldig ik mij voor in deze lijst gemaakte fouten en onduidelijkheden. Ik moet te zeer in detail gaan om alles perfect te vertellen. Ik verwijs daarom voor de juiste syntax naar de handboeken. Mochten er problemen zijn probeer deze dan te isoleren en als zij opgelost zijn laat het anderen door middel van ons magazine weten. Lukt het niet de problemen op te lossen vraag dan via het blad om hulp.

`ABS(a)`	Om de absolute waarde van a te bepalen.
`a AND b`	Om een logische 'and' van a en b te doen
`ASC(t$)`	Om de ASCII waarde van het eerste teken van een tekstvariabele t$ te bepalen.
`CHR$(a)`	Om tekstvariabele met ASCII-code a te krijgen.

`CLEAR x` Om geheugendeel te reserveren. Er kan mee worden ingesteld hoeveel geheugen voor stringhandling wordt gebruikt. Maakt geen geheugen vrij voor machinecode daar tweede parameter niet wordt gebruikt. Maakt niets schoon, wist niets, sluit geen bestanden en zal ook niets zoals DEFINT of DEF FN herzetten !

`CLS` Om het scherm schoon te maken.

`COLOR a,b,c` Om kleurenset te kiezen en/of in te stellen. Zonder vermelden van kleur werkt dit als COLOR=NEW.

`COLOR=NEW` Om de kleuren weer volgens de defaultwaarden te krijgen.

`COLOR=RESTORE` Om de kleurenset te herstellen volgens de eerder opgeslagen gegevens.

`COLOR SPRITE(a)`
Om de kleur van de sprite te geven.

`COLOR SPRITE$(a)`
Om de kleur van de sprite per lijn te geven.

`COPY (a,b)-(c,d) TO (e,f)`
Om grafische schermdelen (mag gehele scherm zijn) te copiëren naar andere delen van dezelfde schermpagina of van andere schermpagina's. In syntax kan ook nog schermpagina en indien gewenst een logische operator opgegeven worden.

`DEFDBL`	Om aan te geven dat bepaalde variabelen van het double precision type zijn. Achter DEFDBL de beginletters van de variabelen vermelden. Alleen te gebruiken om ruimte te reserveren. Voorbereiding op eventuele uitbreiding MCBC.
`DEFINT`	Om aan te geven dat bepaalde variabelen van het integertype zijn. Achter DEFINT de beginletters van de variabelen vermelden. Voor MCBC nodig maar wel verplicht opnemen of immer een % achter de naam van de variabele.
`DEFSNG`	Om aan te geven dat bepaalde variabelen van het single precision type zijn. Achter DEFSNG de beginletters van de variabelen vermelden. Alleen te gebruiken om ruimte te reserveren. Voorbereiding op eventuele uitbreiding MCBC.
`DEFSTR`	Om aan te geven dat bepaalde variabelen van het string (=tekst) type zijn. Achter DEFSTR de beginletters van de variabelen vermelden.
`DIM a(x)`	Om de grootte van de array's die gebruikt gaan worden aan te geven. Er kan niet opnieuw worden gedimensioneerd. Moet met getalwaarde, dus constante, opgegeven worden.
`ELSE`	Om alternatief te geven. Alleen in de combinatie met IF...THEN.... en de combinatie IF...GOTO.... te gebruiken.
`a EQV b`	Om een logische 'eqv' met a en b te doen.

`FIX(a)`	Om het geheeltallige deel van een variabele te krijgen. Identiek aan INT voor niet negatieve getallen. Voorbereiding op eventuele uitbreiding MCBC.
`FOR`	Om een programmalus te maken. Alleen te gebruiken in de combinatie FOR a=x TO y in dezelfde regel en NEXT verderop in het programma.
`GOSUB rnr`	Om naar de subroutine met het opgegeven regelnummer te gaan.
`GOTO rnr`	Om naar een andere plaats met het opgegeven regelnummer in het programma te gaan.
`IF`	Om een conditie te testen. Alleen te gebruiken in de combinatie IF...THEN... of IF...GOTO....
`a IMP b`	Om een logische 'imp' met a en b te doen.
`INKEY$`	Om het toetsenbord te scannen.
`INP(a)`	Om een byte van de aangegeven poort te lezen.
`INT(a)`	Kapt het breukdeel van variabele er af. Door de notatie worden negatieve getallen 1 kleiner dan verwacht. Voorbereiding op eventuele uitbreiding MCBC.
`KEY (a) ON`	Om direct na indrukken functietoets ergens heen te springen.Alleen te gebruiken in combinatie met ON KEY ... die voor deze instructie ergens in het programma opgenomen moet zijn.

N.B. Niet om de inhoud van de functietoetsen op scherm te zetten. De (a) is dan ook verplicht.

`KEY (a) OFF` Om niet meer direct na indrukken functietoets ergens heen te springen.
Alleen te gebruiken in combinatie met ON KEY ... en KEY ON die voor deze instructie ergens in het programma opgenomen moet zijn.
N.B. Niet om de inhoud van de functietoetsen van scherm te halen. De (a) is ook verplicht.

`KEY (a) STOP` Om niet meer direct na indrukken functie toets ergens heen te springen maar wel om bij te houden of de toets ingedrukt wordt.
Alleen te gebruiken in combinatie met ON KEY ... en KEY ON die voor deze instructie ergens in het programma opgenomen moet zijn.

`LEFT$(t$,a)` Om het linkerdeel van a tekens lengte van de tekstvariabele t$ te pakken.

`LEN(t$)` Om de lengte van een tekstvariabele t$ te krijgen.

`LET` Om voor een toewijzing te zetten. Niet verplicht om te gebruiken.

`LINE (a,b)-(c,d),k`
Om een lijn te tekenen. De syntax is veel uitgebreider maar daar ga ik nu hier niet verder op in.

`LINE (a,b)-(c,d),k,B`
Om een kader te tekenen. De syntax is veel uitgebreider maar daar ga ik nu hier niet verder op in.

`LINE (a,b)-(c,d),k,BF`
Om een blok te tekenen. De syntax is veel uitgebreider maar daar ga ik nu hier niet verder op in.

`MID$(t$,a,b)` Om een deel vanaf positie a met lengte b uit het midden van een tekstvariabele t$ te krijgen.
N.B. De functie MID$ wordt niet ondersteund.

`a MOD b` Om de rekenkundige a modulo b bewerking te doen.

`NEXT` Om het einde van een programmalus aan te geven.

`NEXT i,j` Om het einde van de programmalus(sen) met index i (,j enz.) aan te geven.

`NEW` Om in combinatie met COLOR de kleuren in de default waarde te zetten.
N.B. : werkt niet om programma te verwijderen.

`NOT a` Om een logische 'not' met a te doen.

`ON...GOTO rnr,rnr`
Om afhankelijk van de waarde na ON naar de regel met het opgegeven regelnummer te springen.

`ON...GOSUB rnr,rnr`
Om afhankelijk van de waarde na ON naar de subroutine met het opgegeven regelnummer te springen.

`ON INTERVAL GOSUB rnr`
Om na verstrijken van een bepaald interval naar de subroutine met het opgegeven regelnummer te springen. Alleen te gebruiken in combinatie met INTERVAL ON.

`ON KEY GOSUB rnr,rnr`
Om na indrukken functietoets naar de subroutine met het opgegeven regelnummer te springen. Alleen te gebruiken in combinatie met KEY (a) ON.

`ON SPRITE GOSUB rnr`
Om bij de botsing van twee sprites naar de subroutine met het opgegeven regelnummer te springen. Alleen te gebruiken in combinatie met SPRITE ON.

`ON STRIG GOSUB rnr`
Om bij indrukken van de actieknop van muis of joystick naar de subroutine met het opgegeven regelnummer te springen. Alleen te gebruiken in combinatie met STRING ON.

`a OR b` Om een logische 'or' met a en b te doen.

`OUT a,b` Om een waarde b naar de opgegeven poort a te zenden. Lijkt in syntax en opzet sterk op POKE.

`PAD(a)` Om de paddle uit te lezen.

`PAINT (a,b),c,d`
Om een willekeurig vlak in te kleuren.

`PDL(a)` Om de stand van de joystick uit te lezen.

`PEEK (a)` Om de inhoud van een geheugenadres (één byte) te lezen. Houd rekening met geheugenpagina's. Voor de adressen tussen &H4000 en &H8000 mag u er vrij zeker van zijn andere waarden te krijgen, uitzonderingen daargelaten dan in de situatie in basic.

`POINT (a,b)` Om de kleur van een beeldpunt te lezen.

`POKE a,b` Om de inhoud van een geheugenadres (één byte) te schrijven. Voor de adressen tussen &H4000 en &H8000 geldt nu dat het RAM-geheugen is en niet zoals normaal ROM-geheugen. Er valt nu dus wel in te POKE'n.

`PRESET (a,b)` Om een punt in de achtergrondkleur te tekenen. De syntax is veel uitgebreider maar daar ga ik nu hier niet verder op in.

`PRINT` Om een variabele of constante op het scherm te zetten. De syntax is veel uitgebreider maar daar ga ik nu hier niet verder op in. Pas op USING kan niet gebruikt worden.

`PSET (a,b)` Om een punt in de voorgrondkleur te tekenen. De syntax is veel uitgebreider maar daar ga ik nu hier niet verder op in.

`PUT SPRITE a,(b,c)`
Om een sprite a op het scherm a in de voorgrondkleur te plaatsen. De syntax is veel uitgebreider maar daar ga ik nu hier niet verder op in.

`REM` Om opmerkingen in de listing te maken. Wordt niet vertaald en u kunt er dus net zoveel commentaar in zetten als u wilt. Ook ' kan gebruikt worden.

`RESTORE` Om de kleurenset te herstellen volgens de eerder opgeslagen gegevens in de combinatie COLOR=RESTORE. Werkt niet om de READ-pointer op een andere plaats te zetten.

`RETURN` Om aan het eind van een subroutine terug te keren naar het punt waar vandaan naar de subroutine werd gesprongen.

`RETURN rnr` Om te gaan naar het gewenste adres. Het verschil met GOTO is dat nu de stack van de returnadressen één wordt verlaagd.
Vaak nodig na een onderbreking door een ON....GOSUB...routine.

`RIGHT$(t$,a)` Om het rechterdeel van a tekens lengte van de tekstvariabele t$ te pakken.

`SCREEN a,b` Om het gewenste scherm te kiezen en een aantal andere instellingen te doen. Zonder parameters werkt het als een CLS. De syntax is veel uitgebreider maar daar ga ik nu hier niet verder op in.

`SET PAGE a,b` Om grafische schermpagina te kiezen voor zichtbare of onzichtbare status, respectievelijk actieve of passieve status.

`SGN(a)` Om het teken van een variabele te krijgen.

`SOUND a,b` Om een waarde b in geluidsregister a te plaatsen. Als meerdere geluidsregisters met de juiste waarden gevuld zijn kan zo een geluid gemaakt worden. Vergelijkbaar met POKE en OUT.

`SPRITE OFF` Om de spritebotsingen detectie uit te zetten.

`SPRITE ON` Om de spritebotsingen detectie aan te zetten.

`SPRITE STOP` Om de reactie op spritebotsingen uit te zetten.

`SPRITE$(a)` Om een sprite in de spritetabel te plaatsen.

`STEP` Om bij grafische instructie de locatie relatief ten opzichte van laatst behandelde punt te nemen.

`STEP a` Om de stapgrootte bij een programmalus aan te geven.
Alleen in combinatie met FOR...TO... NEXT

`STICK(a)` Om de status van cursor of joystick te lezen.

`STRIG(a)` Om de status van actieknoppen (op een joystick of de spatiebalk) te lezen.

`STRIG (a) OFF` Om STRIG (a) ON effect weer af te zetten. Kan dus alleen in combinatie met STRIG ON en ON STRIG GOSUB rnr gebruikt worden.

`STRIG (a) ON` Om na iedere instructie de status van de actieknop te lezen. En indien er actie gevraagd wordt te springen naar de subroutine. Kan alleen in combinatie met ON STRIG GOSUB rnr gebruikt worden.

`STRIG (a) STOP` Om na iedere instructie de status van de actieknop te lezen, maar niet naar de subroutine te springen. Kan alleen in combinatie met STRIG ON gebruikt worden.

`STR$(a)` Om de numerieke waarde a als tekststring te krijgen. Denk eraan dat niet-negatieve waarden beginnen met een spatie !

`THEN` Om gevolg van actie mee te beginnen. Kan alleen in de combinatie met IF...THEN... gebruikt worden.

`TIME` Om de interne teller (een systeemvariabele) te lezen of te schrijven.

`TO` Om de eindgrens bij een programmalus aan te geven. Kan alleen in de combinatie met FOR...TO... in zelfde regel en NEXT gebruikt worden.

`USR(a)` Om een machinetaalroutine aan te roepen. Kan alleen gebruikt worden als de DEF USR a al gedefinieerd is. Dit kan echter NIET in het te compileren deel staan maar dient voor de aanroep van de object code te geschieden.

`VARPTR(a)` Om de plaats waar een variabele in het geheugen staat te weten te komen.

`VDP(a)` Om het betreffende VDP-register te lezen of te schrijven.

`VPEEK (a,b)` Om een geheugenplaats (byte) in het videoRAM te lezen.

`VPOKE a,b` Om een geheugenplaats (byte) in het videoRAM te schrijven.

`a XOR b` Om een logische 'xor' met a en b te doen.

`'` Te gebruiken als REM-statement. Dubbele punt is er niet voor nodig. Wordt niet opgenomen in de object code.

NIET ondersteund

Floating point variabelen

Meer dimensionale arrays

De volgende statements

ATN | BASE | BEEP | BIN$ | CIRCLE | CLOSE | COS |

SRLIN | DATA | DRAW | DSKF | DSKI$ | DSKO$ | EOF |

ERASE | EXP | FIELD | FN | FPOS | GET | HEX$ | INPUT |

INSTR | LOC | LOCATE | LOF | LOG | LSET | OPEN | PLAY |

POS | READ | RESUME | RND | RSET | SIN | SPACE$ | SPC |

SQR | STRING$ | TAB | TAN | USING | VAL | WIDTH

NIET zinnig om te ondersteunen

De volgende statements

ATTR$ | AUTO | BLOAD | BSAVE | CALL | CDBL | CINT |

CLOAD | CMD | CONT | CSAVE | CSNG | CVD | CVI | CVS |

DELETE | END | ERL | ERR | ERROR | FILES | FRE | IPL |

KILL | LFILES | LIST | LLIST | LPOS | LPRINT | LOAD |

MAXFILES | MERGE | MKD$ | MKI$ | MKS$ | MOTOR | NAME |

OCT$ | RESTORE | RENUM | RUN | SAVE | SWAP | TROFF |

TRON

De rekentekens

/ ^

Het is duidelijk dat als de compiler geschikt gemaakt zou worden voor gebruik met floating point getallen een aantal statements die nu bij 'niet zinnig' staan dan bij 'niet ondersteund' komen. Sommige statements worden in de lijst met ondersteunde statements genoemd na aanhalingstekens. Dit is om aan te geven dat de compiler niet op alle punten de toegestane syntax kan volgen.

In sommige gevallen is dit zeer duidelijk DIM werkt nu alleen voor eendimensionale arrays, maar in andere gevallen is er sprake van een soms te betreuren beperking, MID$ kan alleen gebruikt worden als een soort LEFT$ of RIGHT$ dus als functie maar niet als opdracht.

Bij de niet volledig te gebruiken statements staan steeds de beperkingen genoemd. Experimenteer echter gerust het kan zijn dat in de laatste versie van MCBC juist de beperking die u wilde gebruiken is opgeheven.

Lees daarover steeds ons magazine waar regelmatig tips in zullen komen om met MCBC te werken.

Geheugengebruik.

Bij MCBC kan de programmeur het gehele geheugen dat normaal gebruikt kan worden nog steeds gebruiken. Sterker nog; de gecompileerde delen staan in andere pagina's en het oorspronkelijke BASIC programma zou zelfs wel een 80 Kbyte of meer groot kunnen zijn. Van dit programma zijn dan een aantal delen gecompileerd zodat het resterende basic programma wel in de machine (in het normale basic geheugendeel) past.

De compiler staat zelf in een niet door basic gebruikt geheugendeel en neemt tijdens werking alleen tijdelijk basic geheugen in beslag voor opslag variabelen. Een basic programma dat dus met alleen ondersteunde statements geprogrammeerd is en meer dan 20 Kbyte groot is kan verwerkt worden in één compileslag, mits de resulterende code niet boven de 32 Kbyte komt.

Het is met MCBC ook voor de eenvoudige basicprogrammeur mogelijk het gehele geheugen van de MSX-2 (tot 4 Mbyte indien aanwezig in uw computer) te gebruiken. Zie voor verdere details over het gebruiken van het geheugen het hoofdstuk werken met MCBC verderop in deze handleiding.

Een te compileren programma zal zowel groter als kleiner kunnen worden in de gecompileerde versie. Om informatie hierover te krijgen moet in CONTROL optie 3 gekozen worden.

Snelheidswinst

De snelheidswinst is sterk afhankelijk van de gebruikte instructies. Bij PAINT is bijvoorbeeld vrijwel geen winst te verwachten. Ook zullen alle SCREEN 2 acties nauwelijks (of niet) sneller verlopen. In bepaalde gevallen kan de snelheidswinst wel 2000 % of meer bedragen. Vooral de spritebewegingen winnen enorm aan snelheid. Ik heb de standaard benchmarks aangepast op werken met integers en geef hier de tijden van de basic versie versus de gecompileerde versie.

Benchmark #1	voor 0.95 sec	na	.06 sec	dat is 16 x sneller
Benchmark #2	voor 3.98 sec	na	.06 sec	dat is 62 x sneller
	Ja, echt waar tweemaal gecontroleerd !			
Benchmark #3	voor 19.08 sec	na	1.42 sec	dat is 13 x sneller
Benchmark #4	voor 17.98 sec	na	1.36 sec	dat is 13 x sneller
Benchmark #5	voor 10.18 sec	na	1.38 sec	dat is 7 x sneller
Benchmark #6	voor 17.76 sec	na	1.74 sec	dat is 10 x sneller
Benchmark #7	voor 29.66 sec	na	1.88 sec	dat is 16 x sneller

We zien dus een enorm verschil tussen de verschillende benchmarks. Overigens kon benchmark #8 niet gedraaid worden wegens uitdrukkelijk gebruik van floating point getallen.

De tijden vóór zijn NIET van de standaard benchmarks maar de aangepaste integerversies ervan. Het is natuurlijk erg oneerlijk een floating point basic programma te gaan vergelijken met een integer machinetaal routine. Op de diskette staan al deze benchmark programma's zodat u het zelf ook nog eens kunt nagaan.

TIP : gebruik voor hogere snelheid in testen liever een test op gelijk of ongelijk dan een test op kleiner of groter. Bij deze laatste moet ook rekening worden gehouden met het teken (+/-) en bij de eerste niet.

Resetbestendig

Zowel de MCBC als de gegenereerde code zijn reset bestendig. Het is trouwens erg grappig om te zien dat dat met een BASIC programma, waarin een _MEM naar een gecompileerde code staat, rustig aangepast kan worden (de sprites worden bijvoorbeeld anders gevuld) en dan zonder verdere ingrepen weer loopt. Als we even nadenken is dat logisch, als een programma de spritebeweging in een subroutine regelt is het te verwachten dat als de sprites in een ander deel van het programma anders gevuld worden, de subroutine in kwestie nog steeds eender werkt.

Interrupts

Een werkende object code kan niet onderbroken worden zonder speciale maatregelen. U kunt zelf een stop inbouwen, bijvoorbeeld toetsenbord regelmatig scannen of een tijdsinterrupt. Anders voor aanroep van object code een POKE &HFBB0,1 om er voor te zorgen dat met het gelijktijdig indrukken van [SHIFT] & [CODE] & [GRAPH] & [CTRL] de werking kan onderbroken worden.

Pas op 1 : Deze optie werkt pas nadat de toetsenbordscan weer aangezet is. U kunt daarvoor zorgen door in het begin van het programma een BIOS-call te doen, bijvoorbeeld door CLS.

Pas op 2 : Deze onderbreking van het programma is een zogenaamde warme restart, de machine staat nog steeds in het gebruikte, dus vaak grafische, scherm. U dient na de onderbreking zelf (vaak blind) SCREEN 0 in te typen en eventueel kleur in te stellen voordat u iets kunt lezen. Het voorafgaande zal u vermoedelijk doen besluiten maar een onderbreking naar eigen maaksel te verzorgen.

Werken met MCBC

Hoe werken we met MCBC ? Wij bekijken hiervoor twee mogelijkheden die enigszins anders behandeld dienen te worden.

Ten eerste is er de mogelijkheid dat u nu u weet wat de mogelijkheden zijn van MCBC een programma schrijft dat volledig rekening houdt met de mogelijkheden en beperkingen van MCBC.
Ook kan het natuurlijk zo zijn dat één van uw programma's toevallig al geheel uit toegestane statements bestaat. In dit geval kunnen we de gehele source compileren.

Ten tweede is er de mogelijkheid dat we een bestaand basic programma hebben dat we graag gedeeltelijk willen compileren in verband met de snelheid. Alles compileren zou misschien wel leuk zijn, maar zal bijna nooit kunnen omdat er niet ondersteunde statements in het programma voorkomen.
Aangezien deze laatste mogelijkheid de moeilijkste is en ook vrij vaak zal voorkomen in de praktijk begin ik met het bespreken van dit geval.

Let op de extensies

In de volgende verhandeling ga ik er van uit dat u dezelfde extensies gebruikt als ik. Aangezien het gehele verhaal nogal complex kan worden door het vrij grote aantal files dat in het verloop van de verhandeling een rol gaat spelen is het van vrij groot belang dat we er streng op toezien de juiste extensie te gebruiken.

De afspraken voor de extensies berusten op logische gronden en zijn daarom redelijk simpel te onthouden. We gaan uit van een normaal BASIC-programma dat dan ook de extensie .BAS heeft.

Dit programma passen we zodanig aan dat het gecompileerd kan worden. Het programma dat we nu krijgen geven we de extensie .B2M. De extensie .B2M is een soort acroniem voor Basic to (2=two) Memory, zodat we aan de extensie kunnen zien dat dit een programma(deel) is dat gecompileerd kan gaan worden.

Is de .B2M-file eenmaal gecompileerd wordt die weggeschreven onder de extensie .MEM een afkorting voor memory. Om het programma in gecompileerde vorm te laten werken is er natuurlijk nog een loader nodig die alles op de juiste plaats in het geheugen zet.

Deze loader heeft zoals gebruikelijk de extensie .LDR. Tot slot hebben we nog het BASIC-programmadeel dat de gecompileerde versie aanroept en eventueel de niet ondersteunde zaken regelt.

Omdat dit programma altijd met de .MEM wordt gebruikt is de gekozen extensie .B4M dat staat voor Basic for (4=four) Memory. Door de gekozen .B2M en .B4M is het duidelijk dat ook aan .B1M en .B3M betekenissen zijn gegeven.

Deze twee files zijn echter niet altijd nodig. Onder de .B1M-file versta ik de file die bewerkt is om gesplitst te kunnen worden in één te compileren en één niet te compileren deel. Deze .B1M-file kan in principe gelijk zijn aan de .BAS-file maar zoals u in het gebruik van MCBC zult leren zullen er wel verschillen zijn. Ik om hier later nog op terug.

De .B1M-file wordt gesplitst in een .B2M-file die gecompileerd gaat worden tot .MEM-file en een .B3M-file die het niet te compileren deel bevat. Deze .B3M-file wordt minimaal met een _MEM aangevuld tot de .B4M-file. Als u wilt en kunt is het mogelijk om de lader .LDR en het aanroepprogramma .B4M te combineren.

Ik zou hiervoor dan toch de extensie .LDR gebruiken maar ook .RUN is een redelijke keus.

Lees eerst de volgende tekst door en kijk dan nog eens naar deze uitleg van de extensienamen en naar het korte overzicht dat verderop nog eens gegeven wordt.

Programma bevat niet ondersteunde statements

Later als u eenmaal met MCBC hebt leren omgaan kunt u deze aanpak best enigszins aanpassen, maar in dit eerste geval kunnen we het best de volgende stappen nemen :

Begin met een nieuwe schijf en zet daar MCBC.BIN en het programma CONTROL en eventueel LOADER.BAS op en verder het programma dat u wilt gaan compileren.

Bestudeer het basicprogramma (ik geef dat van nu af aan met .BAS aan) goed en zoek het deel dat te compileren is. Is dit deel er niet of nauwelijks en het moet toch met MCBC versneld worden dan zal het programma zodanig aangepast moeten worden dat een deel in toegestane statements staat en alleen integers gebruikt. Het zal in vele gevallen blijken dat een beetje bewuster programmeren het programma al zonder compileren aanzienlijk versnelt.

Aanpassen op compilatiemogelijkheid

Om het programma zo goed mogelijk aan te passen voor compilatie zoeken we eerst alle zeer beslist niet te compileren delen. Kies zoveel mogelijk ondersteunde statements ook in de stukken die toch niet gecompileerd kunnen worden. Als MCBC wordt uitgebreid is uw programma al beter voorbereid. Schrijf bepaalde stukken misschien over.

Als ik bij voorbeeld een klein array van zo'n 12 waarden moet vullen zal ik bijna altijd kiezen voor de oplossing :

```
FOR I=1 TO 12
READ A(I)
NEXT
DATA 2,2,3,4,4,5,6,6,7,8,8,9
```

Maar nu is het misschien het overwegen waard om daar maar eens het volgende van te maken :

```
A(1)=2:A(2)=2:A(3)=3:A(4)=4:A(5)=4:A(6)=5
A(7)=6:A(8)=6:A(9)=7:A(10)=8:A(11)=8:A(12)=9
```

Veel van de variabelen blijken best integer te kunnen zijn en al voor compilatie worden we zo al vaak aangenaam verrast met een sneller programma. Dan zorgen we ervoor dat de niet te compileren stukken netjes bij elkaar staan. En om goed duidelijk te maken welk deel wel gecompileerd kan gaan worden is het een goed idee dat deel in een subroutine aan het eind van het programma te plaatsen. Op de oorspronkelijke plaats komt dan een GOSUB rnr te staan. Vooruitlopend kan ik nu reeds zeggen dat aan het eind van al onze operaties een _MEM op de plaats van de GOSUB rnr komt te staan. Vervolgens gaan we de communicatie tussen de beide programmadelen verzorgen.

Uitwisselen van variabelen

Een normale subroutine gebruikt automatisch dezelfde variabelen als het hoofdprogramma. Maar de communicatie tussen het basic deel (.B4M) en de machinetaalroutine (.MEM) zal anders moeten geschieden. Vergelijk het met de situatie van twee programma's in basic. Een programma met de naam A dat eindigt

met de instructie RUN "B" en het programma met de naam B eindigt met RUN "A". Ook in dat geval zullen alle waarden van variabelen van het ene programma tijdens laden van het andere programma verloren gaan. We moeten de variabelen van het ene (basic-)deel aan het andere (machinetaal-)deel doorgeven.
En aan het eind weer teruggeven voorzover dat nodig is. We kunnen dit doen door vlak voor de _MEM een aantal POKE's te doen.

Weten we bijvoorbeeld dat de geheugenadressen vanaf E000 vrij zijn kunnen we de variabele met de naam A als volgt gaan doorgeven :

In het resterende basicdeel, dat na de splitsing .B3M en later .B4M moet worden zetten we :

```
rnr  POKE &HE000,A MOD 256:POKE &HE001,A\256
```

In het te compileren deel (nu nog subroutine maar straks .B2M) nemen we op :

```
rnr  A=PEEK(&HE000)+256*PEEK(&HE001)
```

Moet de waarde van A weer terug gegeven worden dan moet in het te compileren deel ook de eerst genoemde regel staan en na terugkomst in het resterende basicdeel ook de daarna genoemde regel. Een goede lezer heeft misschien al vastgesteld dat ik de waarde van de variabele A laag/hoog wegzet.
Op zich is dat hier niet noodzakelijk maar ik raad iedereen aan om zich er aan te wennen dat ook te doen omdat er anders snel vergissingen komen ten opzichte van de in de computer gebruikelijke notatievorm. Weten we echter zeker dat de waarde van A nooit meer dan 255 zal bedragen (voor schermcoördinaten is dit vrij waarschijnlijk) dan kan het eenvoudiger met :

```
rnr POKE &HE000,A
```

en later

```
rnr A=PEEK(&HE000).
```

Om te voorkomen dat lange reeksen POKE's en PEEK's ingetoetst moeten worden kunnen we ook meerdere variabelen met een string doorgeven. We slaan de variabelen op in een tekststring b.v. :

```
T$=CHR$(A)+CHR$(B)+CHR$(F)+.....+CHR$(M)
```

en dan kan met

```
POKE &HE000,VARPTR(T$) MOD 256
POKE &HE001,VARPTR(T$)\256
```

het adres van T$ doorgegeven worden. En door wat PEEK's (op E000 en E001) vinden we T$ en zo de variabelen weer terug.

Veilige adressen om te POKE'n zijn meestal de adressen in het videogeheugen. Hiervoor dan natuurlijk wel VPEEK en VPOKE gebruiken. Eventueel in een niet-gebruikte pagina. Voor de MSX-1 schermmodes geldt trouwens een beperking tot adres 16383 ook al bevat uw machine meer videogeheugen is dit nu niet bruikbaar. En bedenk dat ook het machinetaaldeel ze moet kunnen vinden, ga het dus niet in een pagina zetten die straks wordt uitgeschakeld. Probeer het uitwisselen van variabelen echter tot een minimum te beperken, het kan de werking van het eindresultaat negatief beïnvloeden. Het vervelende is dat we zolang we nog met de interpreter werken alle variabelen natuurlijk worden doorgegeven aan de subroutine. Wilt u er echt zeker van zijn dat alles correct werkt dan kunt u zoals al eerder werd gemeld het programma scheiden in twee aparte programma's.

Het ene programma laadt dan het andere en omgekeerd. Alle variabelen worden dan echter steeds bij elke nieuwe run op nul gezet en als het doorgeven met PEEK en POKE respectievelijk VPEEK en VPOKE niet goed is gedaan ziet u dat snel aan het resultaat. Deze voorbereiding op splitsen (het is in feite al gesplitst) is verreweg de omslachtigste en zal vaak niet nodig zijn.

Laatste stappen voor splitsen

Ook zal in het te compileren deel de instructie DEFINT A-Z opgenomen dienen te worden en met een DIM aangegeven moeten worden hoe groot de arrays zijn. Maar nu werkt het niet meer correct in een ongesplitst basicprogramma !!!! We krijgen 'Redimensioned array in rnr' als foutmelding. We moeten dit dus pas als allerlaatste vlak voor compileren er tussen voegen.

De bewerkte .BAS die nog net loopt in BASIC geven we aan met de extensie .B1M. Deze naamgeving wordt verderop duidelijker. De .B1M-file wordt op schijf weggeschreven, niet voor de aardigheid maar om te voorkomen dat u straks alles weer opnieuw moet doen !

Nu wordt naar keuze het te compileren of het resterende deel gewist met DELETE rnr-rnr. Ik raad aan te beginnen met het te compileren deel, enerzijds omdat we dan snel na een compilatiepoging zien dat er nog iets aan het andere deel veranderd moet worden en anderzijds omdat de naamgeving al aanleiding geeft eerst .B2M en dam pas .B3M te behandelen.

Het deel dat gecompileerd gaat worden wordt zonodig aangevuld met de DIM en de extra DEFINT A-Z. Als het inderdaad zoals aanbevolen een subroutine is moet de return in de laatste regel (!!!) vervangen worden door bijvoorbeeld REM EINDE.

Vervolgens wordt deze versie op schijf gezet met de extensie .B2M.

De extensie .B2M staat voor Basic to (2=two) Memory.
Het andere deel (.B1M min .B2M) dat basic blijft gaat naar schijf met de extensie .B3M. Om dit voorelkaar te krijgen laden we eerst de .B1M en verwijderen daaruit de .B2M met DELETE rnr-rnr en houden dan vanzelf .B3M over.

Compileren

Nu gaan we .B2M compileren. Er zijn twee mogelijkheden : de eerste beveel ik zeker voor beginners aan en maakt gebruik van het programma CONTROL, dat op de schijf wordt meegeleverd. De tweede werkt zonder het programma CONTROL en wordt later besproken. We tikken :

```
RUN "CONTROL"
```

en laden en starten zo het programma CONTROL dat een en ander voor ons regelt. Zorg ervoor dat alles goed en wel op schijf staat voordat CONTROL geladen wordt.

```
1
```

Kies de optie 1 en zo laden we de compiler in het geheugen.

```
2
```

Kies nu optie 2 om te gaan compileren.

CONTROL geeft aan dat de te compileren programma(deel) geladen moet worden en dan gecompileerd kan worden. We laden dus de .B2M en tikken vervolgens :

```
_COMP
```

(of CALL COMP)

We krijgen na een korte tijd de cursor terug na de prompt als alles bij compilatie goed ging en anders een normale basicfoutmelding met het re-

gelnummer waar het fout ging. Bij een vermoedelijk correcte compileslag zien we geen fout- melding, nu starten we wederom CONTROL met

```
RUN "CONTROL"
```
of door F3 te in te drukken

We kiezen voor de zekerheid eerst optie 3 om te zien hoe de compilering verlopen is. Eventueel kunnen de gegevens genoteerd worden, maar dit zal in het begin zelden echt nodig zijn.

Wegschrijven

Nu gaan we het machinetaaldeel wegschrijven naar diskette. Hiervoor kiezen we optie 4. Het programma vraagt om een naam. We kiezen de extensie .MEM omdat dit deel straks ook met _MEM opgeroepen zal gaan worden.

Restant basic aanpassen

Nu gaan we .B3M aanpassen. Het programma moet straks het machinetaaldeel kunnen oproepen. Hiervoor zijn een aantal dingen noodzakelijk. Ten eerste moet het machinetaaldeel aangeroepen worden. Zoals reed eerder vermeld kan dit met _MEM, maar dan moet dit wel in het geheugen staan en de machine moet weten waar hij moet zoeken naar die MEM. Ik neem even aan voor het gemak dat het machinetaaldeel wordt ingeladen op de plaats waar hij destijds ook naar toe werd gecompileerd namelijk pagina 3. In .B3M nemen we dan op de regel :

```
rnr OUT &HFD,3:_MEM:OUT &HFD,2
```

Met de OUT kiezen we pagina 3 en na de _MEM zetten we

weer netjes de keuze op de oorspronkelijke pagina 2. We zetten dit programma nu op schijf met de extensie .B4M. De B4M komt van Basic 4 (for) Memory. In veel gevallen zal op deze manier het programma reeds kunnen werken.

We kunnen dus RUN proberen. Stel dat alles naar wens werkt en u zet tevreden de machine uit. Later wilt u nogeens genieten van het programma en dan blijkt het ineens niet meer te werken. De verklaring is simpel; het gecompileerde deel is niet aanwezig en kan dan ook niet werken. Als we met een schone (net aangezette en niet alleen geresette) machine beginnen zal ons MEM-deel natuurlijk niet aanwezig zijn. Maar direct na compilatie staat het nog wel in het geheugen. We moeten er trouwens ook voor zorgen dat ons basicprogramma op een goede plaats in het geheugen staat. Voor beide kunnen we zorgen door een laadprogramma, een voorbeeld laadprogramma staat op de schijf onder de naam : LOADER.BAS. Voor ik hier aan toekom eerst nog eens de extensienamen op een rijtje.

Overzicht extensienamen

- **.BAS** het programma waar we vanuit gaan. Werkt in principe alleen met de interpreter.
- **.B1M** het programma dat de eerste aanpassing heeft ondergaan. Moet inprincipe nog werken met de interpreter. Het te compileren deel staat in een subroutine die met GOSUB rnr wordt aangeroepen.
- **.B2M** het programmadeel dat gecompileerd moet worden. Is in feite de subroutine uit .B1M waarin als het nodig is de DEFINT A-Z en de DIM in is opgenomen en de RETURN uit verwijderd is.
- **.B3M** het deel van .B1M dat niet gecompileerd gaat worden.

- **.B4M** het resterende basicdeel .B3M waarbij de aanroep van het gecompileerde deel is opgenomen.
- **.LDR** het programma dat alles op de juiste plaats in het geheugen laadt en de .B4M opstart.
- **.MEM** de machinetaalcode die MCBC vervaardigde uit .B2M.
- **.M1M .M2M .M3M** ... Als er meerdere .MEM files zijn.

Werking LOADER.BAS

Het programma begint in regel 20 te kijken of het op de goede plaats (vanaf &HC000) in het geheugen staat. Als dit niet het geval is doet het een tweetal POKE's in regel 30 en herlaadt zichzelf op de juiste plaats. Dit is noodzakelijk omdat de loader anders zelf zou staan op de plaats waar hij wil gaan inladen. Staat de LOADER.BAS op de goede plaats wordt het machinetaaldeel ingeladen in regel 60 op de door ons gewenste pagina. Met de OUT &HFE,3 wordt er voor gezorgd dat het op pagina 3 komt. Wilt u om welke reden dan ook een andere pagina moet u dat hier aangeven met de waarde achter de OUT &HFE. In regel 50 worden de gegevens verzameld en verwerkt zodat in regel 60 op de juiste plaats de waarde &H20 gePOKEd kan worden zodat de computer 'weet' dat er een *callable* ROM zit op &H4000.

Anders gezegd : als u dit niet doet werkt _MEM niet. In regel 70 wordt de normale situatie hersteld en alle ROM's weer gelezen kunnen worden. In regel 80 zorgen een tweetal POKE's ervoor dat het basicdeel .B4M op de correcte plaats (vanaf adres &H8000) kan worden ingeladen en dan wordt .B4M in regel 90 geladen en gelijk gestart.

Eigen loader maken

We laden LOADER.BAS in en passen het aan onze wensen aan.

regel 10 : verander in de REM het algemene 'loader' voor de door u gewenste naam en zet er eigen naam en datum bij.

regel 20 : ongewijzigd laten

regel 30 : ongewijzigd laten

regel 40 : zet hier de naam waaronder straks dit laadprogramma zal worden weggeschreven.

Het is aan te raden dezelfde naam als bij de andere files van de serie te gebruiken, maar nu met extensie .LDR.

regel 50 : ongewijzigd laten

regel 60 : zet achter de OUT &HFE, het gewenste paginanummer. Welke paginanummers gekozen kunnen worden hangt af van de grootte van het geheugen van uw computer. Bij de Sony HB-700P (256 Kbyte) tot en met 15, Bij de Philips NMS 8235 (128 KByte) tot en met 7 en bij nog kleinere geheugens (64 KBbyte) slechts tot en met 3. Bij een correct uitgevoerde memoryexpansion kan misschien een veel grotere waarde gebruikt worden. De maximale waarde bepaalt u dan door het totale geheugen in de memorymapper te delen door 16384. U weet dan hoeveel blokken van 16 KByte er zijn. Het grootste getal dat u nu kunt gebruiken is een minder dan dat aantal blokken van 16 KByte. Tevens moet u de naam van de file met extensie .MEM achter de BLOAD nog aanpassen in de door u gebruikte filenaam.

regel 70 : ongewijzigd laten

regel 80 : ongewijzigd laten

regel 90 : vul hier de naam van de gewenste .B4M in. Voor deze filenaam kan zowel voor RUN als LOAD gekozen worden.

Schrijf tot slot de loader onder de in regel 40 gebruikte naam weg op schijf. Met de besproken aanpassingen heeft u nu de beschikking over een loader die geschikt gemaakt is voor uw programma.

Klaar

Als alles goed gegaan is, bent u nu klaar. Ter controle zet u de bestanden .LDR, .B4M en .MEM op een andere schijf en zet de computer enige tijd uit. Na enige tijd, meer dan een minuut wachten a.u.b., tikt u hoopvol RUN "naam.LDR" en was alles goed dan zal het nu perfect gaan. Als we willen kunnen we tot slot nog besluiten de files .B4M en .LDR te combineren.

Verwijder in deze fase de werkfiles nog niet. Pas als u er zeker van bent dat alles goed werkt én dat u het programma nooit meer zult willen wijzigen kunt u de werkfiles kill'en.

Combineren .LDR en .B4M

In bijna alle gevallen is het mogelijk deze twee bestanden te combineren. Bedenk echter dat het programma zichzelf herlaadt en dat dat relatief veel tijd kan kosten als het programma groot is. Daarnaast is het later misschien niet altijd even duidelijk welke POKE's nu voor het laden nodig waren en welke POKE's voor iets anders. Het kan dus wel, maar doe het niet te snel.
Een probleem hierbij is ook dat het basicprogramma in de gekozen opzet soms vanaf adres &HC000 staat en dan niet al te groot mag, zeg maar rustig klein moet, zijn. In dat geval kan de extra uitbreiding met de .LDR net te veel zijn. Moeilijk zal het echter niet zijn en u kunt met een klein experimentje snel uitvinden of het gaat of niet.

Programma kent alleen ondersteunde statements

Dit is gelukkig een stuk gemakkelijker. We laden de compiler met het programma CONTROL en compileren ons .BAS programma. Dan weer CONTROL laden en de gecompileerde versie

op schijf zetten met de .MEM extensie.
De loader laden en aanpassen voor ons .MEM deel. en nu maken we waarschijnlijk niet een .B4M-file maar starten we .MEM natuurlijk wel direct op vanuit de loader. Dus in plaats van LOAD "basic.B4M" nu een _MEM na de OUT.

MCBC zonder CONTROL.

Ook zonder steeds opnieuw met control te werken is het goed mogelijk MCBC te gebruiken. Hoe in dit geval MCBC op de juiste plaats ingeladen moet worden kunt u in CONTROL vinden.
Als deze tip niet voldoende is kunt u er beter niet aan beginnen MCBC te gebruiken zonder CONTROL. Als de compiler echter eenmaal in het geheugen staat kan een basicprogramma dat gecompileerd kan worden er tegelijk bij in het geheugen staan.
De compiler staat in pagina 2 en de gecompileerde versie komt in pagina 3 en eventueel pagina 4 als pagina 3 te klein is dat wil zeggen dat de objectcode meer dan 16 Kbyte groot is.
U start de compiler met _COMP nadat de juiste pagina is ingesteld, dus we tikken :

```
OUT&HFD,2:_COMP
```

Heeft u eenmaal een oorspronkelijke versie van CONTROL laten lopen kunt u hiervoor ook F1 en F2 gebruiken. Om de gecompileerde versie te starten tikken we :

```
OUT &HFD,3:_MEM:OUT &HFD,2
```

Of na gebruik van CONTROL simpel F6,F7 en F1

Na deze laatste OUT (of F1) kan direct weer _COMP gegeven worden maar om fouten te voorkomen is toch voor alle zeker-

heid de OUT &HFD,2 voor de _COMP gezet.

memorymapper : FD en FE

Met OUT &HFE, geven we voor BLOAD "xxxx.MEM" aan in welke geheugenpagina geladen moet gaan worden en met OUT &HFD, geven we voor de _MEM aan in welke pagina gekeken moet worden. Haal deze twee niet door elkaar een fout is hier zo gemaakt. Pas bij het zelf POKE'n en OUT'n op dat u geen vergissingen maakt. In de loader wordt gebruik gemaakt van de OUT &HFE en in CONTROL van OUT &HFD.

Een vergissing is niet altijd snel hersteld en komt bijna altijd ongelegen. Save daarom altijd uw tussenresultaten op tijd, dus regelmatig en zeker vlak voor een 'RUN', POKE of OUT.

Geheugengebruik in detail

Ik ben niet van plan het hele systeem van de memorymapper en aanverwante zaken hier uit te leggen maar wel de informatie te verstrekken die sommigen nodig kunnen hebben om in grensgevallen met MCBC te kunnen werken. Het geheugen dat de Z80 processor kan aanspreken is slechts 64 KByte groot.

Door gebruik te maken van een memorymapper is een veel groter geheugen te bewerken. Het systeem berust er op dat het aanwezige geheugen wordt verdeeld in blokken van 16 KByte. vier van deze blokken worden aangewezen door de memorymapper om het door de Z80 adresseerbare geheugen te zijn. Deze vier blokken zijn normaal als volgt :

0000 t/m 3FFF een ROM met de BIOS van de computer.
4000 t/m 7FFF een ROM met de Basic interpreter.
8000 t/m BFFF een RAM met de gebruikersprogrammatuur.
C000 t/m FFFF een RAM met gebruikersprogrammatuur en de systeem-BIOS van de computer.

Duidelijk is hopelijk dat in ons geval blok 1 geheel en 4 gedeeltelijk niet te gebruiken zijn. Maar tijdens compilatie is er geen basicinterpreter nodig zodat blok 2 voor de compiler gebruikt kan worden.

De compiler wordt ook in pagina 2 van de memorymapper geladen. De compiler compileert het programma naar een andere pagina (3 + eventueel 4) in de memorymapper en compileert het zodanig dat de gecompileerde versie straks zal werken in blok 2 met eventueel uitloop in blok 3. Met de instructies OUT &HFD en OUT &HFE, kunnen we de memorymapper besturen.

Als u hierover doordenkt zult u kunnen inzien waarom er gekozen is voor de 16 KByte grens voor de grootte van de compiler en waarom de compiler niet uit een programma aanroepbaar is.

Tijdens de compilatie moet de compiler echter ook een ruimte reserveren voor variabelen en een zogenaamde stringpool. Deze ruimte wordt bepaald door de CLEAR in .B2M.
De defaultwaarde is hier 200.
De plaats waar die ruimte komt echter wordt bepaald door de situatie tijdens compilatie. Als u dus verstandig bent geeft u als u deze zaken goed wilt regelen voor de compilatie van .B2M eerst een CLEAR die net zo is als de CLEAR die u straks in .B4M wilt gaan gebruiken. Doet u dit zullen er geen problemen zijn, zorg er echter wel voor dat het geheugen maar eenmaal gebruikt kan worden.
Maak de getallen die u bij CLEAR opgeeft dus niet groter dan noodzakelijk anders komen er conflicten. Deze conflicten zijn niet altijd gelijk zichtbaar maar komen aan het licht bij onverklaarbare molest van uw machinecode.
Heeft u dus een machinetaalroutine geschreven, die tot nu toe altijd perfect werkte en nu ineens geen medewerkking verleent; kijk in dat geval eens naar het geheugen waar u uw routine had neergezet.

Is uw machinetaalprogramma duidelijk verknald dan moet u maar eens met CLEAR gaan experimenteren en natuurlijk opnieuw compileren !

Routines

Veel problemen zullen regelmatig voorkomen. Dit betreft zowel zaken die eens hebt geprogrammeerd en die handig in andere programma's zijn op te nemen zoals ook in normaal basic geldt. Maar ook zijn er de oplossingen die u ontwikkelde om bepaalde niet ondersteunde statements te ondervangen.

Het is slim om deze oplossingen redelijk gebundeld te houden om dan in voorkomende gevallen direct te kunnen gebruiken.

Ook zou je zo een 'uitbreiding' van basic kunnen maken door een code aan te maken die iets doet wat zelfs in standaard basic niet ondersteund wordt. Als voorbeeld : stel dat de PAINT niet in MSX-basic bestond, je kan dan een routine in basic maken die een vlak voor je opvult. Deze routine compileer je dan en je hebt de PAINT beschikbaar met _MEM. Je kan wel de standaard PAINT nu uitbreiden tot een PAINT met een patroontje !

Of u maakt een PAINT om een gebied te arceren. En dat kan en handige toepassing zijn voor toepassingen met screendumps op een normale printer zonder kleur.

Wat oplossingen voor niet ondersteunde statements

Voor veel van de niet ondersteunde statements zijn er oplossingen te verzinnen die het gemis aan die statement snel doen vergeten. Ik ben me er van bewust dat er niet voor elk probleem een zinnige oplossing te vinden is maar voor veel gevallen is er best een mouw aan te passen.

Een paar voorbeelden :

BEEP werkt niet maar de combinatie DEF USR 0=&HC9 in .B4M en A=USR(0) in .B2M wel en doet hetzelfde. Ook zal er met een paar sound commando's best een aardige BEEP gemaakt kunnen worden.

SWAP A,B werkt niet maar C=A:A=B:B=C wel

LOCATE 25,9:PRINT "Tekst" werkt niet maar de printopdracht PRINT CHR$(27)+"Y"+CHR$(32+9)+CHR$(32+25);"Tekst" wel. Pas op dat je denkt aan de puntkomma of + voor de te printen tekst anders zit u gelijk al op de volgende regel. Zie hiervoor bijvoorbeeld mijn artikel programmeer technieken in het MSX Club Magazine nummer 24.

Er zijn er zo wel meer te vinden. Ik heb de bedoeling regelmatig in het MSX Club Magazine dit soort aanpassingen te geven.
Het zal wel een beetje van de inzendingen afhangen dus als u een goede *omweg* weet meld die dan.

Tijdmeting

Bent u erg benieuwd naar de snelheidswinst kan het programma dat de .MEM aanroept voorzien worden van een timer.
Aan het begin en aan het eind van het te meten deel zetten we

```
rnr  TIME=0
  .....
rnr  PRINT TIME/50;" seconden"
```

Deze timer niet meecompileren omdat de meting dan niet helemaal eerlijk is.

Samenvatting gebruik MCBC.

Ik geef op de volgende pagina's nog even een overzicht van de stappen die genomen moeten worden om een programma dat in MSX-Basic is geschreven geheel of gedeeltelijk gecompileerd te krijgen met behulp van MCBC.

Zorg liefst voor een lege schijf waar vervolgens de files MCBC.BIN , LOADER.BAS en CONTROL op staan.

We beginnen nu, anders dan bij de voorafgaande uitleg met de simpele variant waar een programma geheel geschikt is voor compilatie door MCBC.

Vervolgens de lastige variant, waar een programma niet geheel geschikt is voor compilatie door MCBC. Ook hier liefst zorgen voor een lege schijf, waar vervolgens de files MCBC.BIN , LOADER.BAS en CONTROL op gezet worden.

Programma volledig compileerbaar

Zet het te compileren programma als eerste op de disk

RUN "CONTROL"

1 Laad MCBC.BIN met optie 1 van CONTROL

2 Stop CONTROL met optie 2

LOAD "naam.BAS"
Laad programma dat gecompileerd moet worden

F2 compileer u kunt ipv F2 ook zelf _COMP geven

F3 run CONTROL

4 kies de SAVE-optie

prog.MEM zet het gecompileerde met de extensie .MEM op disk

eventueel

5 run de gecompileerde versie ter controle

LOAD "LOADER.BAS"
Laad het programma om te veranderen in een eigen versie.

Pas LOADER.BAS aan, in ieder geval op de volgende punten.

- verander de naam van de .MEM in de door u gebruikte naam
- verander indien nodig het paginanummer waar .MEM wordt ingelezen. Het moet overeenstemmen met het paginanummer van de _MEM verderop in de loader
- verander de naam LOADER.BAS in de naam van je eigen programma met de extensie .LDR zowel achter de LOAD als achter de REM in de eerste regel
- zorg dat de _MEM in de zelfde pagina werkt als waar .MEM werd ingelezen
- verander de naam LOADER.BAS in de naam van je eigen programma met de extensie .LDR
- verander het inlezen en starten van de .B4M file in de laatste regel in een _COMP na het instellen van de juiste pagina

```
SAVE "prog.LDR"
```

zet de computer uit

wacht minstens één minuut

zet de computer weer aan en dan de ultime test :

```
RUN "prog.LDR"
```

Is nergens een fout gemaakt moet alles nu goed werken.

Programma niet volledig compileerbaar

- Zet het te compileren programma op de disk met de extensie .BAS

- Bepaal ruwweg het te compileren deel en isoleer dat vervolgens in een subroutine aan het eind van het programma.

- Breid het programma uit met het veilig doorgeven van de noodzakelijke gegevens aan de te compileren subroutine Bijvoorbeeld met een aantal POKE's, VPOKE's of OUTs.

- Breid de te compileren subroutine uit met het correct ophalen van de noodzakelijke gegevens uit het programma. Bijvoorbeeld met een aantal PEEKs, VPEEK's of INs.

Herhaal deze laatste twee punten ook de andere kant op van de te compileren subroutine naar het programma indien nodig.

- Ter controle RUN het programma en kijk of alles nog (zij het trager) werkt.

- SAVE deze versie met de extensie .B1M.

- DELETE het niet te compileren deel

- Breid indien nodig het overblijfsel uit met bijvoorbeeld DIM's en een DEFINT en zorg ervoor dat altijd bij de laatste regel geëindigd wordt.

- SAVE dit deel nu met de extensie .B2M.
- Laad de versie .B1M
- DELETE nu juist het te compileren deel
- SAVE met de extensie .B3M
- RUN "CONTROL"
- Laad MCBC.BIN met optie 1 van CONTROL
- Stop CONTROL met optie 2
- Laad programmadeel dat gecompileerd moet worden met LOAD "naam.B2M"
- compileer .B2M met F2 of _COMP
- run CONTROL met F3
- kies de SAVE-optie met 4
- zet het gecompileerde met de extensie .MEM op disk
- LOAD "LOADER.BAS"

- Pas LOADER.BAS aan

  - verander de naam van de .MEM in de door u gebruikte naam
  - verander indien nodig het paginanummer waar .MEM wordt ingelezen. Het moet overeenstemmen met het paginanummer van de _MEM in .B4M
  - verander de naam LOADER.BAS in de naam van je eigen programma met de extensie .LDR zowel achter de LOAD als achter de REM in de eerste regel
  - verander de naam van de file in de laatste regel in de naam die door u gebruik gaat worden voor de .B4M file

- SAVE "prog.LDR"

- Laad de file met de extensie .B3M

- Vervang in dit programma mistens de GOSUB rnr die de nu gecompileerde subroutine aanriep door een OUT &HFD,x en een _MEM. Zorg ervoor dat het paginanummer overeenstemt met het paginanummer dat in de loader gebruikt werd om te lezen.

- SAVE onder de extensie .B4M

- zet de computer uit

- wacht minstens één minuut

- zet de computer weer aan
- RUN "prog.LDR"

is nergens een fout gemaakt moet alles nu goed werken.

Vragen en antwoorden

Kan MCBC ook op een MSX-1 werken ?

Vermoedelijk wel als de MSX-1 is uitgerust met een memory-mapper bijvoorbeeld die in een memorycartridge en er alleen MSX-1 statements gebruikt worden, maar dit hebben wij nog niet getest.

Hoe kan ik meerdere stukken van mijn programma door MCBC laten compileren ?

Niet zo moeilijk al is het wel veel werk. Splits eerst een deel van het programma waar u van uit gaat af en behandel dat op de hopelijk nu bekende wijze. Als alles goed werkt beschouwen we de .B4M file als een nieuwe .BAS en gaan weer net zo te werk als zoëven. Het probleem schuilt hier in de extensienamen waar we er een zo grote hoeveelheid van krijgen dat we door de bomen het bos niet meer zien. Ik stel voor alle versies van de eerste gang te laten verdwijnen met twee uitzonderingen de loader xxxx.LDR en de machinetaalroutine xxxx.MEM moeten blijven. Het best is het waarschijnlijk om weer een nieuwe schijf te gebruiken waar in het begin alleen de volgende vijf files op staan : MCBC.BIN, CONTROL, xxxx.LDR, xxxx.MEM en xxx.B4M. Hernoem de xxxx.MEM in xxxx.M1M en de xxxx.B4M in xxxx.BAS. Noem het resultaat van de compilatie nu .M2M en zorg er voor dat de nieuwe loader beide .MxM-files in verschillende pagina's inleest en de nieuwe .B4M ook beide aanroept. Herhaal dit proces net zo vaak als nodig is voor een derde, een vierde, enz. machinetaalroutine.

De eerste keer dat ik mijn programma testte ging alles naar wens maar nu werkt hij opeens van geen kant.
Hoe kan dat en vooral wat moet ik doen ?

Als het programma de eerste keer goed werkte betekent dat dat er een goede .MEM in het geheugen gekomen is. Mogelijkheden : Heeft u er wel aan gedacht die versie te saven of staat er nog steeds een oude versie op schijf ? Compileer .B2M opnieuw en save nu wel. Het saven is misschien wel goed gebeurd maar het laden nu niet. Destijds bij die eerste test ging dat ook fout maar toen stond de correcte versie nog op de goede plaats in het geheugen. Nu de machine uitgestaan heeft, of in ieder geval de .MEM niet meer op de juiste plaats heeft, werkt het vanzelf niet meer. De remedie ligt ook nu voor de hand. Pas uw loader programma aan op correct inlezen van de .MEM. U ziet twee mogelijkheden en in beide gevallen een correcte compilatie maar toch de eerste keer de noodzaak om opnieuw te compileren. Omdat u niet zeker weet wat er bij u gebeurd is; kijk als eerste naar de loader en ziet u daar niets vreemds, dan maar opnieuw compileren.

Er is echter nog een andere reden dat een programma de ene keer wel goed loopt en de andere keer iets heel anders doet.

Als we normaal RUN geven worden automatisch alle variabelen op nul gezet, bij de compiler gebeurt dit niet. Als u de eerste keer werkte met geheugenadressen die toevallig schoon of geschikt gevuld waren loopt alles zoals verwacht werd.

Bij elke volgende RUN zal elke keer dat deze geheugenplaatsen gebruikt worden het resultaat identiek zijn als de inhoud van de geheugenplaatsen identiek is aan de eerste keer. Is de inhoud echter veranderd zal het programma ook iets anders doen.

Zorg ervoor dat alle variabelen netjes worden geïnitialiseerd, getallen op nul en strings leeg.

De compilatie verliep uitstekend maar ik kan mijn programma niet meer onderbreken. Wat nu ?

Jammer dan. U kunt resetten en zowel de in uw ogen niet perfecte objectcode als eventueel MCBC blijven in het geheugen aanwezig. Voorkom dit soort problemen door een stop in uw programma in te bouwen.

Bij de laatste poging liep mijn object code vast of beter gezegd bleef maar doorlopen en was niet te onderbreken.
Ik heb toen gereset en ik wil nu een te onderbreken versie maken maar hoe pak ik dat aan ?

Dit kan op verschillende manieren met alle hun eigen nadelen. Alle methodes hebben als nadeel dat de object code er trager door zal worden, soms echter zo weinig dat dat niet merkbaar is.
Een programma dat blijft doorlopen zal een lus moeten bevatten. In deze lus kunt u een keyboardscan doen en bij positief resultaat de lus verlaten.
Uw .B2M ziet er bijvoorbeeld als volgt uit :

```
.........
130 .....
 .....              de lus die blijft doorlopen
220 .....

.....
440 REM einde
```

In het bovenstaande geval voegt u bijvoorbeeld vlak voor regel 220 een nieuwe regel 215 toe.

```
215 IF INKEY$="s" GOTO 440
```

De lus kan nu wel verlaten worden door het indrukken van de kleine letter s. Deze controle is wel traag en zal dus remmend werken, of dat storend is zult u zelf moeten beoordelen.

Een andere mogelijkheid is het onderbreken op tijd. Gemakkelijk te programmeren is bijvoorbeeld in het begin zetten van de tijd op een of andere waarde. Dit zetten doet u bij voorkeur in .B4M omdat het dan later nog aangepast kan worden.

In .B4M neemt u op

```
rnr TIME=y
```

met y een waarde van 0 tot 65536 en een regelnummer zo dat deze regel voor de CALL MEM komt. In .B2M neemt u vervolgens op de regel

```
215 IF TIME=0 GOTO 440
```

Als de y groot gekozen wordt zal er snel een onderbreking volgen. Bij de keuze y=0 zal het een ruime twintig (65536/50/60) minuten duren voordat er onderbroken wordt. Denk er wel aan dat gedurende het zetten van de tijd en het uitlezen ervan ook enige tijd verloopt en de nul zo gepasseerd kan worden.

Een zeer snelle test is een PEEK op adres &HFC9F waar een deel van de TIME staat.

In .B2M neemt u op de regels

```
rnr TIME=0
215 IF PEEK(&HFC9F)=y GOTO 440
```

en uw lus zal na iets meer dan y maal 5 seconden onderbroken worden.

N.B. De TIME=0 natuurlijk niet in de lus zetten !

Ik moet een vrij groot aantal gegevens inlezen door middel van de READ-DATA structuur. MCBC ondersteund dat niet en om nu al de data door te POKE'n lijkt mij te veel werk. Wat is hiervoor een oplossing ?

Dit betreft een groot aantal gegevens die door middel van een READ-DATA combinatie in een of meer array's worden ingelezen. Er zijn een aantal mogelijke oplossingen te geven.

Eerste oplossing is dat u niet de DATA zelf doorgeeft maar alleen de plaats van de array waar ze in staan. Hoe dit aangepakt ziet u in het meegeleverde voorbeeld programma met de sprites.

Tweede oplossing is het doorgeven van de DATA op een supersnelle manier via de videoRAM. In uw .B4M staan al de DATA-statements die u normaal met een READ inleest. Voor de eerste READ echter geeft u een VPOKE 0,0:RD=PEEK(7).
Op adres 7 zal &H98 staan maar voor alle zekerheid lezen we het even uit. Nu leest u de DATA in met een READ A direct gevolgd door OUT RD,A. Alle DATA worden nu razendsnel in de videoRAM gezet. Het best heeft dan ook gezorgd voor een grafisch scherm en gebruikt u de niet zichtbare pagina. Pas wel op dat dit alleen voor gehele waarden van 0 tot 256 geldt anders moeten de getallen in stukken worden gehakt. Het inlezen van de gegevens gaat identiek maar dan met de PEEK op adres 6 in plaats van adres 7 en A=INP(RD) in plaats van OUT RD,A. Het ophogen van de teller waar we zijn gaat automatisch, maar pas wel op dat de waarde niet gereset wordt. Met een PRINT zult u dit bijvoorbeeld al doen. Alleen inlezen en opslaan in een array dan zijn er geen problemen.

Derde mogelijkheid is om in dit geval een klein programmaatje te maken dat de DATA inleest en dan wegschrijft op disk.

Normaal kunnen we dan de gegevens weer inlezen om ze snel paraat te hebben maar ook inlezen van schijf kan niet met MCBC. Wel kunnen we echter de gegevens inlezen in ons .LDR programma dat we toch moeten maken. We kiezen daarvoor een plaats die door de object code gelezen kan worden.

Ik wil een twee dimensionale array DIM G(12,7) gebruiken. Hoe pak ik dit aan?

De array G(12,7) bevat 13 maal 8 is 104 elementen. In .B2M definiëren we daarom H(103) die ook 104 elementen bevat. Willen we nu bijvoorbeeld element G(6,3) uit H halen moeten we element 6 x 8 + 3 = 51 of 3 x 13 + 6 = 45 pakken. De keuze is arbitrair maar leg de eenmaal gemaakte keuze wel goed vast. Ik zelf kies meestal voor de tweede telling vanwege de overeenkomst met lezen.

Ik krijg bij verschillende berekeningen andere antwoorden dan ik in normaal basic krijg. Wat is hiervoor een veklaring ?

Dit kan eigenlijk niet. Ik vermoed echter dat u een fout tegen de prioriteitsregels gemaakt hebt en in uw normale basic nog steeds / voor deling gebruikte. Ik geef een voorbeeld om het toe te lichten.

? 24 / 4 * 2 zal 12 opleveren, want 24 / 4 = 6 en 6 * 2 = 12

maar

? 24 \ 4 * 2 zal 3 opleveren, daar 4 * 2 = 8 en 24 \ 8 = 3

In het eerste geval zijn * en / van dezelfde prioriteit en in het tweede geval heeft * een hogere prioriteit dan \ en moet derhalve ook eerst gedaan worden.

Sinds ik mijn programma heb uitgebreid met een interrupt routine loopt hij keer op keer vast. Hoe komt dat ?

De instructie RETURN rnr is de boosdoener. Dit zal bijna nooit goed kunnen werken in een interruptroutine in een gecompileerd programma. Ik geef toe dat dat juist de plaats is waar u hem het meest zinnig wil gebruiken en dus geef ik een methode om het probleem op te lossen.

Zet in de interrupt routine een vlag met

```
rnr VL=1
```

en test dan in het te interrumperen programma op de daarvoor geschikte plaats(en)

```
rnr ON VL GOTO rnr
```

met hierin als laatste regelnummer het nummer waar u eigenlijk naar toe had willen springen na de RETURN rnr.

Ik krijg steeds REDIMENSIONED ARRAY als foutmelding terwijl ik helemaal geen array heb gedimensioneerd. Hoe komt dat ?

Dat is het juist. U MOET elke array die u wilt gebruiken dimensioneren voor MCBC. Ook de array's die minder dan 11 elementen bevatten.

Op welke adressen kan ik een pointer voor een array veilig doorgeven als ik de video niet kan of wil gebruiken ?

Gebruik de adressen &HFFFC, &HFFFD en &HFFFE.

Ik krijg mij eigen machinetaal routines niet aan de praat. Wat doe ik fout ?

Meerdere mogelijkheden

Uw eigen routine staat in een memorypage die niet toegankelijk is op het moment dat de .MEM draait.

U moet in .B4M de opdracht DEF USR a = ... opnemen en de opdracht DUMMY=USR(A) in .B2M

De opdracht CLEAR is niet goed gebruikt. De CLEAR a,b die u in .B4M gebruikt moet ook gegeven worden vlak voor de _COMP anders zal laten het geheugendeel dat u bij .B4M nodig heeft om de eigen routines op te slaan door de .MEM kunnen worden overschreven.

Waarom ondersteunt MCBC de DISK-commando's niet ?

Omdat daar geen enkele snelheidswinst is te behalen. Alleen voor de programmeur zou een programma misschien sneller te schrijven zijn.

Waarom ondersteunt MCBC de floating-pointgetallen niet ?

Aan de ene kant voor snelheid kiezen en aan de andere kant floating pointgetallen wensen is op zijn zachtst gezegd verbazingwekkend. Waarom heeft een sportauto als Ferrari, Lamborghini, Porsche, en dergelijke geen trekhaak of imperiaal ?
Maar alle gekheid op een stokje het kost veel snelheid en zou in de huidige opzet ook te veel geheugenruimte kosten.

Komt er ooit nog eens een versie van MCBC die meer (liefst alles) zal ondersteunen ?

We hopen het zeker. Er zijn echter problemen met onder meer het geheugengebruik die zo als het er nu uitziet voor zorgen dat we alleen het ene probleem door het andere vervangen. Worden er echter oplossingen voor deze problemen gevonden zal er zeker aan gewerkt worden. Voor alle duidelijkheid : momenteel wordt er NIET gewerkt aan een uitgebreidere versie van MCBC. Pas als er oplossingen voor de geschetste problemen worden gevonden beginnen we (Adriaan dus) er aan. Als dat gaat gebeuren leest u erover in het MSX Club Magazine en u kunt als geregistreerde gebruiker van MCBC rekenen op een speciale prijs om te upgraden.

Help. Ik kom er echt niet meer uit. Zit er een fout in MCBC ?

Tot slot als u er echt niet meer uitkomt en een fout in MCBC vermoedt.

Als u hulp nodig hebt en mij of een ander daarover benaderd zal het voor ons absoluut noodzakelijk zijn u zich aan de extensieconventie gehouden hebt. Anders zal steevast de reactie zijn : zorg eerst voor de juiste extensies en neem dan weer contact op. Zorg er trouwens ook voor de gegevens uit CONTROL bij de hand te hebben als u hulp nodig hebt. Bel voor problemen met MCBC bij voorkeur op de maandagavond.

Frank H. Druijff

Gebruikte termen

.BAS	bestandsnaamextensie voor programma's die in BASIC zijn geschreven.
.B1M	bestandsnaamextensie voor programma's in BASIC die voorbereid zijn voor (gedeeltelijke) compilatie met MCBC.
.B2M	bestandsnaamextensie voor programma of deel ervan in BASIC dat u wilt gaan compileren.
.B3M	bestandsnaamextensie voor programmadeel in BASIC dat niet gecompileerd wordt (.B1M-.B2M)
.B4M	bestandsnaamextensie voor programma(deel) in BASIC dat het niet gecompileerde deel basic bevat met de aanroep(en) voor het machinetaaldeel.
.MEM	gebruikte filenaamextensie voor de objectcode, die door MCBC wordt gegenereerd.
.LDR	bestandsnaamextense voor een loader
alloceren	toewijzen van geheugenruimte.

assembly language
een computertaal waarbij we haast direct in machinetaal met behulp van de zogenaamde mnemonics programmeren.

BIOS het Basic Input Output System.

BIOS-ROM de ROM waar de BIOS in staat.

callable aan te roepen, bereikbaar met een CALL.

compiler het programma dat een in een hogere programmeertaal geschreven programma kan omzetten in machinetaalcode

compileerbaar eigenschap van programma dat alleen toegestane statements bevat en dus door MCBC in machinetaalcode kan worden omgezet.

control het programma dat voor ons de compiler laadt en informatie geeft over de compilatie en de objectcode voor ons wegschrijft.

data de gegevens/informatie die moet worden doorgegeven van het ene naar het andere deel van het programma.

defaultwaarde waarde die aangenomen wordt als er geen (nieuwe) waarde genoemd wordt.

extensie het laatste deel van de naam van een file.
Dus het *woordje* dat staat achter de punt en maximaal drie tekens lang is.

extensieconventie
de afspraken hoe de verschillende files een onderscheidende extensie krijgen.

expressie uitdrukking, een berekening waarmee een of andere waarde berekend kan worden.

integer getal zonder breukdeel tussen -23768 en 32767.

interpreter het programma dat in de computer zit ingebouwd zodat die in staat is programma's die in MSX-Basic zijn geschreven regel na regel en evenzo instructie na instructie te interpreteren en uit te voeren.

keyboardscan het controleren of er een toets, en zo ja welke wordt ingedrukt.

LOADER.BAS een programma dat zichzelf realloceert en de diverse andere files inlaadt en opstart.

Loader (.LDR) een programma om een ander programma in de computer in te laden en meestal op te starten.

MCBC de Msx Club Basic Compiler

memorymapper
een voorziening die er voor zorgt dat er meer geheugen gebruikt kan worden dan de processor kan adresseren.

mnemonic een korte gemakkelijk te onthouden term voor de diverse instructies in machinetaal

modulair programmeren	programmeren op een methode zodat gemakkelijk de verschillende onderdelen van een programma zijn te herkennen.
prioriteit	de *belangrijkheid* van een bewerking. Hierdoor wordt de volgorde bepaald waarin de bewerkingen worden afgehandeld.
object code	het machinetaalprogramma dat met de compilatie door MCBC van uw BASIC gevormd wordt.
realloceren	opnieuw toewijzen van geheugenruimte. Praktisch : zet op een andere plaats in het geheugen.
RAM	het Random Access Memory, het door de gebruiker te vullen geheugen.
ROM	het Read Only Memory populair gezegd de geheugens waar de routines in staan die het systeem zelf laten draaien.
source code	het MSX-BASIC-programma dat u wilt gaan compileren met MCBC.
stack	de 'stapel' met gegevens waarvan altijd de bovenste wordt gepakt.
stringpool	het geheugengedeelte waar de computer strings in manipuleert.
syntax	de grammaticale regels voor notatie in een taal.

videoRAM	het geheugendeel van de computer dat gebruikt wordt om de scherminformatie in op te slaan Kan bij MCBC goed gebruikt worden voor het doorgeven van data.
werkfiles	de files die gebruikt worden om het eindresultaat te krijgen en later niet meer nodig zijn. Het betreft .B1M, .B2M, .B3M en eventueel .BAS.
Z80	de centrale processor in de MSX-computer.

Geleverde software

Op de diskette staan de volgende files :

De MCBC - bestanden

MCBC.BIN het programma waarin de compiler staat.
CONTROL het programma dat het gebruik van MCBC gebruikersvriendelijk maakt.
LOADER.BAS het programma dat aangepast moet worden om eigen loaderprogramma's te maken.

De benchmark - bestanden

BMARK1.BAS tot en met BMARK7.BAS de basicprogramma's die voor de snelheidstesten zijn gebruikt. controleer ze gerust.

van #1 en #2 zijn de .LDR en .MEM erbij gedaan.

De voorbeeld - bestanden

SAMPLE1.BAS tot en met SAMPLE8.BAS de basicprogramma's die u tot voorbeeld kunnen zijn bij het maken van programma's die u wilt compileren met MCBC. Inclusief de werkfiles.

Let vooral op het programma SAMPLE3 die in twee versies op schijf staat. De tweede versie verschilt slechts in de basicaanroep en gebruikt hetzelfde .MEM bestand. Deze parallelversie heet SAMPLE3B.B4M.

De demonstratie - bestanden

Hoofddemo

De hoofddemo voor MCBC is zonder meer de demonstratie de Martijn Hondema als een der testers schreef.

MCBCDEMO.LDR	de loader voor de overige drie
MCBCDEMO.B4M	start .MEM en een verdwenen muziekje komt dit nog boven water vindt u dit op het diskabonnement van ons blad.
MCBCDEMO.MEM	het volledig gecompileerde bestand.
MCBCDEMO.PIC	grafisch scherm bij deze demo.

Demonstratiecyclus

DEMO1.BAS tot en met DEMO5.BAS	de basicprogramma's, die in een doorlopende demonstratie worden afgewisseld met hun door MCBC gecompileerde versie.
DEMO1.LDR tot en met DEMO5.LDR	de loaders voor de volgende serie.
DEMO1.B4M tot en met DEMO5.B4M	de aanroepprogramma's.
DEMO1.MEM tot en met DEMO5.MEM	de gecompileerde delen nu in machinetaal.

Verder staan op de schijf ook nog :

DEMO1.B2M	de werkbestanden waarin
tot en met	u kunt zien hoe ik gewerkt heb
DEMO5.B2M	om deze demonstratie te maken.

Een spel.

Tot slot een spel dat met behulp van onze MCBC geschreven is. U heeft over de ervaringen van Jacco Bikker bij het vervaardigen van X'mas Quarrel kunnen lezen in het maart-april nummer (28) van het MSX Club Magazine.

XMAS.LDR	de loader voor maar liefst drie .MEM bestanden.
XMAS.B4M	een groot programma dat best korter had gekund.
XMAS.M1M	ook hier iets bijzonders in de vorm van niet
XMAS.M2M	minder dan een drietal .MEM bestanden.
XMAS.M3M	zij zijn dan ook netjes 1,2 en 3 genummerd.

Veel plezier.

MSX club

SOFTWARE

Mottaart 20, 3170 Herselt, Tel. 014/54 59 74