PC-2 Software

****
***** ASM65 *****
****

De omzetting van een symbolische schrijfwijze van een programma naar de voor de 6502 uitvoerbare instructies heet assembleren. Het programma dat deze omzetting doet heet een assembler. Als het bedoeld is om op een 6502-systeem te werken heet het een 'resident' assembler.

ASM65 is een 2-pass resident assembler die voldoet aan de standaard MOS TECHNOLOGIE 6502 taal. De programmatekst wordt met de editor voorbereid en evt op casstte opgeslagen waarna het vanuit het geheugen wordt doorgegeven aan de assembler.

De assembler werkt, nadat een aantal vragen zijn beantwoord geheel zelfstandig, haalt programmatekst op, verwerkt deze en zet de OBJECT-code in geheugen.
De assemblerlisting wordt in de tweede 'pass' afgedrukt.

De assembler kent een aantal optie's voor het in en uitschakelen van de listing, code uitvoer, symboltable afdruk en 'format' funkties. Grote programma's kunnen in kleinere blokken met de editor worden verwerkt en met behulp van het 'file' commando achter elkaar gekoppeld worden. Het programma wordt dan van floppy-disk gehaald. Uiteraard geldt dit uitsluitend voor systemen die zijn uitgerust met floppydisk.

Symbolen:

Symbolen zijn namen die worden gebruikt om getallen voor te stellen. Ze mogen 1 tot 6 letters of cijfers bevatten, maar de eerste moet altijd een letter (a-z) zijn. Uitzonderingen vormen de 56 instructies van de 6502 processor en de symbolen a, x, y, s en p. Deze hebben een speciale betekenis voor de assembler.

Constanten

Constanten kunnen op verschillende manieren en in verschillende getalstelsels worden voorgesteld. Het getalstelsel wordt bepaald door een voorvoegsel dat direct voor de cijfers gezet wordt. De voorzetsels zijn:

voorvoegsel	Stelsel	Basis
(geen)	Decimaal	10
$	Hexadecimaal	16
@	Octaal	8
%	Binair	2

bv.

```
LDA $38FA
LDA @733
LDA 1024
LDA %11100100
```

ASCII constanten worden ingesloten met ' (quotes). Hierdoor weet de assembler dat de waarde van de ASCII-code in het geheugen moet worden geplaatst.
Bv.

```
LDA  #';'
.BYT 'abcdefg'
CMP #'X'
```

Locatieteller:
De locatieteller wordt in de 6502 assembler-taal met het een ster (*) aangegeven. Deze teller wordt zo opgehoogd dat de assembler weet op welk adres in het geheugen hij is. De locatieteller mag overal in expressies gebruikt worden.
Op de locatieteller zijn 2 speciale bewerkingen mogelijk: .EX1 en .EX2 die de inhoud van de locatieteller en een speciaal register uitwisselen. Deze instructies dienen voor het bijhouden van de laatst gebruikte locatie in zeropage (EX1) en ram (EX2). Met deze instructies is het mogelijk 'blokgestruktureerde' programma's te schrijven, waarbij blokken als komplete module kunnen worden opgeslagen en later gebruikt in andere programma's (zoals bij een 'load-module').

Operatoren
Voor het doen van berekeningen binnen ASM65 zijn de volgende operatoren beschikbaar:

+	Optellen
-	Aftrekken
*	Vermenigvuldigen
/	Delen

het verwerken van en expressie geschiedt van links naar rechts, er geldt geen voorrang van operatoren.
Er zijn nog twee speciale operatoren:

>	High-byte selectie
<	Low-byte Selectie

de operatoren < en > verkleinen een 2-byte getal naar zijn low-byte of high-byte.

Assembler directives
ASM65 kent 14 directives. Deze zijn bedoeld voor het zetten van de locatieteller of een symbool op een waarde (=), het reserveren en initieren van geheugenruimte (.BYTE, .SBYTE, .WORD, .DBYTE) en het besturen van de assembler in en uitvoer (.OPT, .FILE, .END) en de assemblerlisting (.PAGE, .TOP, .SKIP). Voor blokgestructureerde software dienen .EX1, .EX2, en .LOC .
Deze directives zijn instructies die tijdens het assembleren worden uitgevoerd en niet tijdens het uitvoeren van het eindprogramma.

=
Toekenning van een waarde aan het linkse symbool. De expressie aan de rechterzijde mag geen voorwaartse referenties bevatten (dwz alle gebruikte symbolen van de expressie dienen vooraf gedefinieerd te zijn).
Bv.
```
MODE1=12
*=$4000

VARPTR=RAMPTR+$300-7
```

.BYTE (of .BYT)
Initieer geheugenlocaties. Meerdere argumenten, gescheiden door kommas, mogen in een commando gespecificeerd worden.
Bv. MSG .BYT 'DIT IS EEN TEKST',13,10,%1000,$10
.BYT $12,>MSG

.SBYTE (of .SBY)
Werkt identiek aan .BYTE, doch de laatste letter van een ASCII-tekst wordt vertaald met het 'sign-bit' (bit7) hoog. Dit kan worden gebruikt om het einde van een tekst aan te geven.
Bv. MSG .SBY 'Dit is een tekst'
genereerd:
44, 69, 74, 20, 69, 73, 20, 65, 65, 6E, 20, 74, 65, 6B, 73, F4
D i t i s e e n t e k s t

.WORD
Initieer een 2-byte geheugenlocatie. De twee byte's worden in low-byte/high-byte volgorde geplaatst, direct geschikt voor 'jump'-tabellen.
Bv. JMPTAB.WOR MSG,MSG+12,*-3,<JMPTAB

.DBYTE
Initieer een 2-byte geheugenwoord. De twee byte's worden in high-byte/low-byte volgorde geplaatst.

.PAGE
Genereert een speciale kop in de programmalisting. Na .PAGE kan een tekst worden opgegeven, die onder de kopregel geplaatst wordt.
Bv. .PAG ' ## testregel 1 ##'
in de listing:

---
testregel 1

.TOP
Idem als .PAGE maar genereert eerst een 'topform' code voor de regelprinter.

.TIT
Zorgt voor een titel in de kopregel van elke pagina.
Bv. .TIT ' ### Test-programma ###'

.FILE & .END
Het is gebruikelijk om lange programma's in verschillende delen te splitsen welke afzonderlijk ge-'edit' worden. Om het programma te assembleren moeten deze afzonderlijke blokken aan elkaar gekoppeld worden. Deze koppeling gebeurt met .FILE <filename> en .END <filename>.
Het .FILE commando koppelt de file met de gespecificeerde naam achter de huidige file. De laatste file van het programma wordt afgesloten met .END met daarachter de filenaam van de eerste file van het programma. In dit geval worden de files van floppydisk gehaald

NB. Het FILE-directive werkt uiteraard alleen op systemen die met een floppydisk zijn uitgerust.

Bv.
```
<< file 'PROG1' >>
   .FILE PROG2

<< file 'PROG2' >>
   .FILE PROG3

<< file 'PROG3' >>
   .END PROG1
```

Wanneer een programma over meerdere floppy-disks verdeeld staat, kan nog een drive specificatie worden opgenomen achter de filenaam.
Bv.
Op de 2 floppy-disks staat:
```
1.  PROG1
    PROG2
2.  PROG3
    PROG4
```

de file/end directives worden dan:
```
<< file PROG1 >>
    .FILE PROG2/1
<< file PROG2 >>
    .FILE PROG3/2
<< file PROG3 >>
    .FILE PROG4/2
<< file PROG4 >>
    .END PROG1/1
```

.SKIP
Maakt een vrije regel in de programmalisting.

.LOCAL
Dit directive definieert het begin van een nieuw blok binnen het programma. Het zorgt ervoor dat labels die binnen dat blok gebruikt worden niet van buiten het blok geadresseerd kunnen worden en dat labels die buiten het blok gedefinieerd zijn niet binnen het blok bekend zijn. Hierdoor is het mogelijk dezelfde namen voor labels in verschillende blokken te gebruiken.
Labels die wel buiten het blok bekend moeten zijn (entry-points) worden gedefinieerd met een dubbele punt achter de naam.
Bv.
```
   INPUT:   LDY ACIANR
   LOOP1    LDA  ACIA,Y
```
hier is 'INPUT' een global label en 'LOOP1' een local label.

.OPT
De 7 options specificeren het 'format' van de assemblerlisting.
Geldige options zijn:
```
.OPT LIST,GENERATE,PRINT,MEMORY,SYMBOL,OBJECT,PUSH
en
.OPT NOLIST,NOGENERATE,NOPRINT,NOMEMORY,NOSYMBOL,NOOBJECT,PULL
```

deze mogen al gezet worden tot:
.OPT LIS,GEN,PRI,MEM,SYM,OBJ,PUS
en
.OPT NOL,NOG,NOP,NOM,NOS,NOO,PUL

de funktie van de verschillende optie's is als volgt:

LIST [NOLIST]
  Schakelt de assembler-listing in (uit).
  Errors worden altijd afgedrukt.

GENERATE [NOGENERATE]
  Bestuurt het printen van objectcode van ASCII strings in het
  .BYT directive. Onder 'nogen' worden alleen de eerste 2 bytes
  Van de string afgedrukt.
  Standaard is 'NOG'.

PRINT [NOPRINT]
  Bestuurt het printen van objectcode; onder 'nop' worden
  Geen regels afgedrukt die geen programmatekst bevatten (coderegels)
  Zoals bij het .WORD en .DBYTE directive.
  Standaard is 'NOP'.

MEMORY [NOMEMORY]
  Bestuurt het genereren van code direct in het geheugen.
  Standaard is 'MEM'.

SYMBOL [NOSYMBOL]
  Bestuurt het al dan niet afdrukken van de symbol table aan het
  einde van de assemblerlisting.
  Standaard is 'NOS'.

OBJECT [NOOBJECT]
  Bestuurt het genereren van code naar een output-device.
  Delen van het programma kunnen hierdoor worden overgeslagen.
  Standaard is 'OBJ'.

PUSH [PULL]
  Push zet de huidige option-informatie in een geheugen. Met
  Pull kan deze informatie weer teruggehaald worden naar de
  Assembler. Wordt gebruikt in 'load-modules'.

De options count/nocount (COU/NOC of CNT/NOC) en ERRORS/NOERRORS
(ERR/NOE) worden wel herkend, maar hebben geen funktie voor de
assembler.

Onderstaand programma is een voorbeeld van toepassing van de ASM65
directives.

```
.PAGE '  ## voorbeeld van een blok ##'
```

```
        ;
        .LOCAL
        ;
        .OPT PUSH    ; (save list-option)
        .OPT NOL     ; (don't list this module)
        ;
        .EXl         ; define zeropage variables
VARl    *=*+1
VAR2    *=*+4
LINE    *=*+80
        .EXl         ; restore locatieteller & save zeropage pointer
        ;
input:  LDY ACIANR   ; index for acia   ('input' is a global label)
        LDA  ACIA,Y
        ...
        ...

        ...
        RTS          ; return & exit from block.
        ;
        .OPT PUL
                     ; (restore old list-option)
                     ;
```

Assembler foutmeldings code's

```
01    Undefined symbol or symbol not in this block
02    Label previously defined or forward reference to page 0 symbol
03    Illegal or missing opcode
04    Addres not valid
05    Accumulator mode not allowed
06    Forward reference to page zero
07    Ran off end of line
08    Label does not begin with alphabetic character
09    Label greater than 6 characters
10    Label or opcode contains non-alphanumeric
11    Forward reference in equate
12    Invalid index, must be X or Y
13    Invalid expression
14    Undefined assembler directive
17    Relative branch out of range
18    Illegal operand type for this instruction
19    Out of bounds on indirect addressing
20    A,X,Y,S and P are preserved labels
21    Programcounter negative, reset to 0
22    Too many local blocks
```