sphynx

SPL

macro-assembler

Voorwoord

In de allereerste plaats wensen wij u geluk met de aanschaf van SPL en hopen dat u er veel, plezierig en productief mee zult kunnen werken. Voor het geval dat u nog geen of weinig ervaring met SPL heeft volgt hier een korte beschrijving.

SPL is een assembler voor de DAI personal computer, die de gebruikers in staat stelt macrodefinities en conditionele assemblage toe te passen. Macrodefinities ("Macro's") maken het mogelijk in het assemblerprogramma een groep instructies door 1 woord aan te duiden en conditionele assemblage maakt het mogelijk een programma snel voor meerdere doeleinden aan te passen, bijvoorbeeld aan het al dan niet gebruiken van de mathchip (AM-9511), door het veranderen van slechts 1 instructie.
SPL (de naam is afgeleid van System Programming Language) is door ons in de zomer van 1982 ontwikkeld. Oorspronkelijk was SPL louter voor ons eigen gebruik bestemd. Wij kregen, toen SPL zijn voltooiing naderde, echter zulke enthousiaste reacties van mensen die SPL in werking zagen, dat wij alsnog besloten SPL te gaan 'exporteren'. Door persoonlijke omstandigheden is het voor ons niet mogelijk zelf de commerciele activiteiten te verrichten, dus hebben wij de verspreidingsrechten overgedaan aan DAInamic. Voordat SPL door DAInamic werd uitgebracht hebben wij eerst zelf SPL in de praktijk uitgebreid getest en aan de hand van de verkregen ervaringen aangepast.

Aan SPL is de grootst mogelijke zorg en aandacht besteed. Wij accepteren echter geen enkele verantwoordelijkheid voor eventuele schade, veroorzaakt door het gebruik van SPL. Vragen en opmerkingen m.b.t. de werking van SPL kunt u sturen aan de programmeurs. Het contactadres luidt:
Werkgroep Sphynx, p/a Gebr. Rens, Grote sloot 101, 1754 JC Burgerbrug (Nederland).

Werkgroep Sphynx

[012] SPL V1.1 M01 830302 (8400-B2CF, E:8500)
[071] & IMPLEM V1.1 M01 830302
[092] & RESTORE xxxx
[103] & DISPLAY V1.1
[177] & TRANSLATOR V1.1

Opstarten

U start SPL op de volgende manier:

1) Zet de DAIpc, beeldscherm en dergelijke aan.
2) Als u de geluidscassetteversie van SPL gebruikt, doe dan de cassette met SPL in de cassetterecorder, ga naar Utility (vanuit BASIC: UT [return]), type R [return] en start de cassetterecorder.
   Als u de DCR-versie van SPL gebruikt, doe dan de cassette met SPL in de DCR, ga naar Utility en type R [return].
   Na niet al te lange tijd moet de cursor nu stil gaan staan, gebeurt dit niet, dan kunt u beter in BASIC met behulp van CHECK gaan kijken wat er aan de hand is.
3) Als de cursor weer begint te knipperen en de Utilityprompt (">") op het scherm wordt afgedrukt, zonder dat er een vraagteken is verschenen, type dan Z3 [return] G8500 [return]. Op het scherm verschijnt nu de SPL-titel en er wordt om een startadres gevraagd. Dit adres geeft het begin van het stuk geheugen aan dat SPL zal gebruiken om het assemblerprogramma op te slaan en mag niet groter zijn dan 83FFH. Het E(dit)-commando gebruikt ongeveer 525D bytes extra, om dit commando te kunnen gebruiken mag het startadres dus maximaal 81F0H zijn.
   Het adres wordt in hexadecimale vorm ingetypt, gevolgd door [return]). Als u inplaats van een adres in te typen direct op [return] drukt, dan wordt het defaultadres gebruikt. Dit defaultadres is implementabel, maar bedraagt in SPL V1.1 standaard 400H.
4) U kunt nu SPL-commando's intypen. Als u SPL verlaat (het U of u-commando) en weer opstart (door (UT [return]) Z3 [return] G8500 [return]), dan verschijnt in plaats van de vraag om het startadres een melding of de assembler (system: "S"), de sourcecode (program: "P") en de symboltable (labels: "L") al dan niet intact zijn ("OK", respectievelijk "BAD").

Mocht u de gebruikte termen niet begrijpen, sla dan de verklarende SPL woordenlijst op.

N.B. de bovenstaande adressen gelden uitsluitend voor de standaardversie van SPL (V1.1). Heeft u een andere versie gekregen dan kan het zijn dat daarvoor andere adressen gelden. Dit wordt dan duidelijk vermeld.

Invoer

De invoer van commando's maakt gebruik van de BASIC scherminvoerroutine. Hierdoor is het mogelijk karakters te verwijderen m.b.v. [chardel] of het ingevoerde commando te annuleren door [break] (indien deze voor [return] wordt ingedrukt).
De SPL prompt bestaat uit 4 karakters: "SPL>". Tijdens de commandoinvoer bestaat de cursor uit een knipperende "$". Als u op [return] drukt wordt de cursor een knipperende "_" of, bij lees- en schrijfroutines, een knipperend zwart blok.

Het eerste karakter van de ingevoerde commandoregel geeft het eigenlijke commando aan. Een spatie op deze plaats duidt erop dat er een DCR-commando volgt.
Na de commandoletter kunnen een of meerdere operands volgen. Deze mogen onderling (of van de commandoletter) door ieder gewenst aantal spaties worden gescheiden. Bij het R- (of Y-)commando dient u de eventuele offset echter direct op de "R" (of "Y") te laten volgen, dus zonder spaties ertussen, omdat anders de offset als deel van de naam wordt beschouwd. Overigens beginnen de namen van sourcefiles op achtergrondgeheugens nooit met spaties, omdat inleidende spaties voor de namen door de invoerroutine genegeerd worden. De naam begint dus altijd met het eerste karakter na de spatie(s) na de R, de W of de Y.

U kunt bij het invoeren van een commando dat als operand een regelnummer verwacht inplaats van dat regelnummer ook een naam intypen. SPL zoekt dan de assemblerregel op waar deze naam voor de eerste maal in de labelKolom staat en doet dan net alsof het nummer van die regel was ingevoerd.

Bij commando's die betrekking hebben op namen in de operandkolom (de G- en X-commando's) kunt u inplaats van die naam ook het "dollarlabel" ($) invoeren.

Hoewel de editor verwacht dat iedere nummerieke operand met een cijfer begint en met een "B","D","H" of "O" eindigt, stelt de invoerroutine minder strenge eisen. Hierbij hoeft een nummerieke operand nooit door "B","D","H" of "O" gevolgd te worden, omdat bij een bepaald commando alleen decimale dan wel hexadecimale operands gebruikt kunnen worden, en het is alleen nodig een hexadecimale operand met een cijfer (bijvoorbeeld "0") te laten beginnen als er op die plaats ook een naam gebruikt mag worden (de I-,l- en X-commando's).

De meeste listings zijn sneller dan het oog kan volgen. Drukt u tijdens het listen een willekeurige toets in (even ingedrukt houden), dan wordt het listen onderbroken en de cursor gaat als "~" knipperen. Dit duurt tot u of [break] of [ ] indrukt. Gebruik daarom voor het stoppen liever niet de [break] of de [ ]. De [break] zorgt voor terugkeer naar de commandoinvoer, de [spatie] zorgt voor het vervolgen van de listing.

Na met name het inlezen van een sourcefile van een achtergrondgeheugen, wat bij geluidscassettes wel enige tijd in beslag kan nemen, wordt een geluidssignaal gegeven. Dit signaal kan door het indrukken van [break] weer worden afgezet. De samenstelling van het geluidssignaal is implementabel.

SPL-commando's

De meeste voorbeelden hebben betrekking op de sourcefile van RESTORE XXXX (zie cassette), waarbij is aangenomen dat de sourcebuffer begint op 5000H (pas dit eventueel aan m.b.v. het J-commando).
De commando's C, A, K, R en Y vragen u om een bevestiging. U ziet dan +? op het beeldscherm. Als bevestiging moet u dan [+] indrukken. Het indrukken van een andere toets zorgt er voor dat het commando niet wordt uitgevoerd. Er verschijnt dan 'BREAK' op het scherm.

(C)
Dit veroorzaakt een "cold start", net alsof SPL pas zou zijn ingelezen en opgestart. Er wordt om een nieuw startadres voor de sourcebuffer gevraagd en de pointers worden terug gezet (wat inhoudt dat de oude inhoud van de sourcebuffer niet meer toegankelijk is). Het C-commando vraagt u om een bevestiging. Drukt u bij het invoeren van het startadres direct op [return], dan wordt het default-adres (standaard 400H) gebruikt.

(J) adres0
Dit stelt u in staat het beginadres van de sourcebuffer aan te passen. De sourcecode wordt zodanig verschoven dat het begin met adres0 samenvalt. Hierdoor kunt u het geheugengebruik van SPL beperken, of juist vergroten om meer regels tegelijk te kunnen editeren.

(Z)
Dit print achtereenvolgens:
a) Het beginadres van de sourcebuffer ("B").
b) Het eindadres van de sourcecode ("E").
c) Het beginadres van de symboltable ("S"). Als de symboltable leeg is, dan is dit adres hoger dan het topadres ("T").
d) Het eindadres (of topadres) van de symboltable ("T").
e) Het beginadres van de gecombineerde sourcecode en symboltable bij het wegschrijven naar het achtergrondgeheugen (het wegschrijfadres: "W").
f) Het laagste wegschrijfadres waarmee een samen te voegen sourcefile mag zijn weggeschreven ("M").
g) Het aantal bytes dat nog in de sourcebuffer beschikbaar is (A).
Een voorbeeld:
Het Z-commando list bij RESTORE XXXX het volgende:

```
B=5000
E=50F4
S=8380
T=83FF
W=828C
M=5174
A=328C
```

(E) [regel1] [regel2]
Dit geeft opdracht een bepaald gebied door de editor te laten bewerken. Zijn regel1 en regel2 gelijk, dan wordt met een lege editbuffer begonnen. Als regel2 groter dan regel1 is, dan worden de regels vanaf regel1 tot aan regel2 in de editbuffer gelist. Wij hebben voor een gebied tot aan en niet tot en met regel2 gekozen omdat dit gemakkelijker is bij het gebruik van label- of identifier-namen in plaats van decimale regelnummers. Een naam zal immers meestal de eerste regel van een nieuw gebied aanduiden, dus zal in veel gevallen niet mee hoeven te worden genomen. Dit geldt met name bij de M- en m-commando's. Ontbreekt een regelnummer of is het gelijk aan 0, dan wel groter dan het laatste regelnummer van het assemblerprogramma, dan wordt het vervangen door het laatste regelnummer van het assemblerprogramma + 1.

Een voorbeeld:

Eerst (gelist)		na E46 48 (in editor)		na editeren (in editor)		na [break][ ] (gelist)	
45	POP H		POP B		POP D	45	POP H
46	POP B		POP PSW			46	POP D
47	POP PSW					47	RET
48	RET						

Zie voor verdere informatie het aparte hoofdstuk over de editor.

L regel1 [regel2]
Dit list de regels vanaf regel1 tot aan regel2 in de vorm zoals ze in de editor getoond worden, doch voorafgegaan door regelnummers. De syntaxregels voor dit commando zijn gelijk aan die voor het E-commando.

T
Dit produceert een objectlisting. Deze listing print voor de kolommen zoals die in de editor getoond worden een aantal extra kolommen. Deze kolommen zijn:

1) De adreskolom. Hierin wordt het adres geprint waarop de objectpointer staat na het verwerken van de vorige instructie.
2) De bytekolom. Bij mnemonics worden in deze kolom de door de mnemonic gegenereerde 1 tot 3 bytes objectcode geprint. Dit geld ook voor de DW- en de DB-directive, met dien verstande dat de DB-directive maximaal de eerste 3 bytes in de bytekolom print. Bij de SET en de EQU-directive wordt op de tweede en de derde byteposititie de waarde geprint die aan het label voor de directive wordt toegekend in de volgorde highbyte-lowbyte en op de eerste bytepositie twee (implementabele) karakters (standaard @=). De andere directives printen niets in de bytekolom.

De objectlisting is object georienteerd, wat betekent dat de regels gelist worden in de volgorde waarin bij het A-commando de objectcode in het geheugen geplaatst zou worden. In de standaardversie worden tijdens een objectlisting terwille van de overzichtelijkheid de volgende directives niet gelist:
IF, ELSE, ENDIF, call-macro, MACRO, MEND, control en ###.
Dit is echter implementabel. Na een UNL-directive wordt het printen gestaakt, hoewel de T-routine verder gaat met het verwerken de sourcecode. Het printen wordt hervat door een LST-directive. De objectlisting eindigt bij de END-directive.

B
Dit zet de uitvoerwissel voor adressen (16-bits getallen) op binair. Adressen zullen in binaire vorm worden gelist, tot de adreswissel wordt omgezet door een D-,H- of O-commando of door het listen van een overeenkomstige PUT-directive. Omdat het bij een binaire listing vaak om de bits gaat worden deze allemaal gelist, bv: 0000101001100111B.

b
Dit zet de uitvoerwissel voor bytes (8-bits getallen) op 'binair'. Bytes zullen in binaire vorm worden gelist, tot de bytewissel wordt omgezet door een d,h of o-commando of door het listen van een overeenkomstige PUT-directive. Ook hierbij worden alle bits gelist, bv: 10001110B.

O
Dit werkt hetzelfde als B, maar zet de adreswissel op octaal. SPL gebruikt de doorlopende octale notatie in plaats van de door sommige andere assemblers gebruikte gesplitste octale notatie. 0FFFFH wordt dus geprint als 177770 en niet als 377770.

o
Dit werkt hetzelfde als b, maar zet de bytewissel op octaal.

D
Dit werkt hetzelfde als B, maar zet de adreswissel op decimaal.

d
Dit werkt hetzelfde als b, maar zet de bytewissel op decimaal.

H
Dit werkt hetzelfde als B, maar zet de adreswissel op hexadecimaal.

h
dit werkt hetzelfde als b, maar zet de bytewissel op hexadecimaal.

V
Dit zet de hardcopy vlag om, dus van aan uit of omgekeerd.
Bij het aanzetten wordt de (zelf te implementeren) printerinitialisatieroutine aangeroepen. In de standaardversie wordt op beeldscherm de vraag "Ready?" gesteld. Dit geeft u gelegenheid de printer aan te zetten en dergelijke. Als u klaar bent moet u [ ] indrukken. Het indrukken van een andere toets zorgt er voor dat het V-commando wordt geannuleerd. Staat de hardcopy vlag tijdens listingen aan, dan worden de karakters behalve naar het beeldscherm ook naar de printer gestuurd.
Let op: het V-commando beinvloedt regel- noch paginateller.

v
Dit zet, mits de hardcopy vlag aanstaat, de regel- en de paginateller op 0. Dit commando stelt u in staat een listing op een nieuw blad papier met PAGE 1 te laten beginnen. U moet dan wel zelf het papier doordraaien, of dit laten doen door een P-commando.

P byte0
Dit stuurt byte0 naar de printer, bijvoorbeeld om al meteen aan het begin van een hardcopy met een bepaald karaktertype te kunnen beginnen. Het versturen 0CH en 0DH bytes beinvloedt de regel- (en eventueel pagina)teller.
Een voorbeeld:
P A stuurt een ASCII-linefeed karakter naar de printer.

p getal0
Dit stelt het aantal regels per pagina voor een hardcopy in. Getal0 is een decimaal getal, kleiner of gelijk aan 127, doch groter of gelijk aan 0. Een 0 geeft aan dat er doorlopend gelist moet worden, dus zonder formfeeds en titels.

Q
Dit sorteert de symboltable in ASCII-volgorde. De symboltable wordt gewoonlijk opgebouwd in de volgorde waarin de editor bij het vertalen van de tekst uit de editbuffer in sourcecode nieuwe namen tegenkomt. Het Q-commando stelt orde op zaken en verwijdert namen waar niet naar wordt verwezen.
Het Q-commando maakt gebruik van het schermgeheugen, waardoor bepaalde grafische effecten ontstaan en het aantal namen in de symboltable beperkt moet blijven tot 1624D. Dit lijkt de ontwerpers van SPL echter ruim voldoende. SPL controleert zelf of de symboltable niet te groot wordt.

S
Dit list de namen in de symboltable, in de volgorde zoals de namen daarin staan, met de laatste waarden die aan de namen zijn toegekend. Bij het S-commando wordt geen plaatsfase uitgevoerd (zie het A-commando).

Een voorbeeld:
```
S list bij RESTORE XXXX het volgende (hardcopy vlag uit):
ADMTHL:DE39 CMPLNR:031F COUNTR:0000 DATAC :0123 DATAD :0124
DEC    :FFFF EIND  :0327 ERRORM:D9F5 FINDLN:02FC FRSTLN:0306
HEX    :0000 RESTOR:02EE TXTBGN:029F TYPE  :FFFF getbyt:50EA
loop   :5074
```
Let wel: bij hardcopy worden er, in plaats van vijf, zes namen per regel gelist. Bovendien worden er niet 1 maar 2 spaties tussen de namen gelist (dit is implementabel).

F
Dit list de labels in de volgorde waarin er waarden aan worden toegekend, in dezelfde vorm als bij het S-commando.
Identifiers worden niet gelist en labels evenveel maal als er waarden aan worden toegekend (b.v. meerdere malen in een lusstructuur of helemaal niet als ze in de niet gebruikte tak van een IF-constructie staan).
Voorbeeld:
De sourcefile RESTORE XXXX produceert na F:
```
DEC    :FFFF HEX    :0000 TYPE  :FFFF DATAC :0123 DATAD :0124
TXTBGN:029F ADMTHL:DE39 ERRORM:D9F5 RESTOR:02EE COUNTR:0002
COUNTR:0001 COUNTR:0000 FINDLN:02FC FRSTLN:0306 CMPLNR:031F
EIND   :0327
```
Het F-commando geeft als het ware een sterk vereenvoudigde objectlisting en kan dienen voor het maken van een memorymap ten bate van de Utility L-opdracht.

I naam0
Dit list alle regels waar in de labelkolom naam0 staat, voorafgegaan door het geheugenadres waarin het statusbyte van die regel is opgeslagen.
Een voorbeeld:
```
I COUNTR  list bij RESTORE XXXX het volgende:
506E 21    COUNTR  SET      2H
506A 24    COUNTR  SET      COUNTR-1H
```
U kunt dit commando onder andere gebruiken om, wanneer het objectprogramma de sourcecode heeft aangetast, het adres te vinden waar de verminkte regel begint, om hem daarna in Utility m.b.v. de S-opdracht te proberen te herstellen.
Zie ook het hoofdstuk: "Steken ophalen voor gevorderden".

I adres0
Dit list de eerste regel waarbij de objectpointer (bij de objectlisting de eerste kolom) groter of gelijk is aan adres0. U kunt dit commando bijvoorbeeld gebruiken om, na het geassembleerde programma in Utility met behulp van de L-opdracht getest te hebben, te zien welke sourceregel met een bepaald adres overeenkomt.
Een voorbeeld:
```
I300 list bij RESTORE XXXX het volgende:
0300  33              ORA A
```

i
Dit list de namen uit de symboltable, voorafgegaan door het regelnummer waar ze in de labelkolom staan, in de volgorde waarin de namen in de symboltable staan.
U kunt zo'n listing als referentie voor een (hardcopy) L-listing gebruiken.
Een voorbeeld:
```
i list bij RESTORE XXXX het volgende (hardcopy vlag uit):
   8 ADMTHL   49 CMPLNR   21 COUNTR   24 COUNTR    5 DATAC
   6 DATAD     2 DEC      55 EIND      9 ERRORM   31 FINDLN
  36 FRSTLN    3 HEX      18 RESTOR    7 TXTBGN    4 TYPE
  56 getbyt   22 loop
```

G naam0 [getal1] [getal2]
Dit list alle regels waar naam0 in de operandkolom, of bij identifiers in de instructiekolom, voorkomt. de Getal1 en getal2 geven het aantal voorafgaande, respectievelijk opvolgende, regels aan dat voor en na de regel met naam0 gelist moet worden. Dit stelt u in staat te zien in welk verband naam0 gebruikt wordt. Het is hierbij mogelijk dat sommige regels meerdere malen gelist worden.
Een voorbeeld:
G getbyt 3 list bij RESTORE XXXX het volgende:

```
46              POP B
47              POP PSW
48              RET
49      CMPLNR  getbyt
49      CMPLNR  getbyt
50              CMP B
51              RNZ
52              getbyt
```

Zijn zowel getal1 als getal2 gelijk aan 0, dan worden de regels in dezelfde vorm als bij het I(naam)commando gelist, dus voorafgegaan door het geheugenadres waar het statusbyte van de regel staat.
Inplaats van naam0 kan ook $ worden ingevoerd.

g
Dit list de namen uit de symboltable, voorafgegaan door het regelnummer waar ze in de operandkolom (of bij identifiers in de instructiekolom) staan, in dezelfde vorm als bij het i-commando. U kunt zo'n listing als cross-reference listing gebruiken, wat vooral bij het foutzoeken erg nuttig kan zijn.
Een voorbeeld:
g list bij RESTORE XXXX het volgende (hardcopy vlag uit):

```
  31 ADMTHL    37 CMPLNR    24 COUNTR    25 COUNTR    44 DATAC
  42 DATAD     10 DEC       35 ERRORM    41 ERRORM    39 FINDLN
  30 FRSTLN    29 TXTBGN    10 TYPE      49 getbyt    52 getbyt
  25 loop
```

l instructie0
Dit list alle regels in de source waar een bepaalde instructie wordt gebruikt.
Een voorbeeld:
l END list bij RESTORE XXXX het volgende:

```
55    EIND    END
```

l byte0
Dit list alle regels waar de instructiecode byte0 wordt gebruikt.
Omdat de remark, de call-macro en de control-directive niet met behulp van een bepaalde karaktercombinatie zijn in te voeren, kunnen deze directives alleen aan de hand van hun instructiecode worden gevonden. Deze codes zijn:
30 voor control
5B voor call-macro
FD voor remark (dit moet worden ingevoerd als 0FD)
Een voorbeeld:
l 30 list bij RESTORE XXXX het volgende:

```
22              loop      COUNTR>0H
```

X naam1 naam2
Dit stelt u in staat de namen van labels en identifiers te veranderen, met dien verstande dat een label een label moet blijven en een identifier een identifier. Alle verwijzingen in de sourcecode naar naam1 worden vervangen door een verwijzing naar naam2 (die eventueel eerst in de symboltable wordt opgenomen).

X label0 adres0
Dit vervangt de verwijzingen naar label0 in de operandkolom door de constante adres0.

X adres0 label0
Dit vervangt de constante adres0 in de operandkolom door verwijzingen naar label0, na eventueel eerst de naam van label0 in de symboltable geplaatst te hebben.

X adres1 adres2
Dit vervangt in de operandkolom de constante adres1 door de constante adres2. Het X-commando kan vooral van nut zijn bij het omzetten van de "standaard" labels uit een door de SPL-disassembler geproduceerde sourcefile in meer "functionele" labels.
Inplaats van label1 of label2 kunt u ook $ invoeren.

N
Dit list de regels waar in de labelkolom een naam staat die nergens in de operandkolom, of bij identifiers in de instructiekolom, wordt gebruikt. Op zichzelf leveren zulke namen geen problemen op, maar het zou kunnen dat ze bijvoorbeeld een niet meer gebruikte subroutine aangeven, die dus uit het assemblerprogramma verwijderd kan worden, of juist een subroutine die wel gebruikt moet worden maar door een vergissing niet wordt gebruikt.
Een voorbeeld:

```
N  list bij RESTORE XXXX het volgende:
 3    HEX     EQU      0H
18    RESTOR  PUSH PSW
55    EIND    END
```

M regel1 regel2 regel3
Dit copieert de regels vanaf regel1 tot aan regel2 vlak voor regel3. De regels blijven ook nog op hun oude plaats staan. U kunt dit commando gebruiken om een soortgelijke reeks instructies, maar met bijvoorbeeld een ander registergebruik, te copieren om het daarna m.b.v. de editor bij te schaven.
Let wel: regel3 mag niet tussen regel1 en regel2 inliggen.
Een voorbeeld:
M CMPLNR EIND FINDLN

Listing voor het M-commando

```
30          JMP       FRSTLN
31 FINDLN   CALL      ADMTHL
..          ....
48          RET
49 CMPLNR   getbyt
50          CMP B
51          RNZ
52          getbyt
53          CMP C
54          RET
55 EIND     END
```

Listing na het M-commando

```
30          JMP       FRSTLN
31 CMPLNR   getbyt
32          CMP B
33          RNZ
34          getbyt
35          CMP C
36          RET
37 FINDLN   CALL      ADMTHL
..          ....
54          RET
55 CMPLNR   getbyt
56          CMP B
57          RNZ
58          getbyt
59          CMP C
60          RET
61 EIND     END
```

^
Dit heeft het zelfde effect als MODE 0 in BASIC: het scherm schuift omhoog tot de cursor op de vierde regel van boven staat.

m regel1 regel2 regel3
Dit doet ongeveer hetzelfde als het M-commando, maar het verwijdert tevens de verplaatste regels van hun oude plaats. Dit gebeurt na het verplaatsen, dus de geheugenbehoefte is hetzelfde als bij het M-commando. U kunt het m-commando gebruiken om de sourcecode te reorganiseren door bijvoorbeeld na het samenvoegen van twee sourcefiles (zie het Y-commando) alle macro-definities bij elkaar te zetten.
Een voorbeeld:
m CMPLNR EIND FINDLN

Listing voor het m-commando

```
30          JMP       FRSTLN
31 FINDLN   CALL      ADMTHL
..          ....
48          RET
49 CMPLNR   getbyt
50          CMP B
51          RNZ
52          getbyt
53          CMP C
54          RET
55 EIND     END
```

Listing na het m-commando

```
30          JMP       FRSTLN
31 CMPLNR   getbyt
32          CMP B
33          RNZ
34          getbyt
35          CMP C
36          RET
37 FINDLN   CALL      ADMTHL
..          ....
54          RET
55          END
```

K regel1 [regel2]
Dit verwijdert de regels vanaf regel1 tot aan regel2. Omdat hierdoor sourcecode verloren gaat, wordt eerst om een bevestiging van het commando gevraagd. De syntaxregels voor dit commando zijn gelijk aan die voor het E-commando.
Een voorbeeld:

Listing voor K15 16

```
14          ELSE
15          PUT       "H"
16          ORG       2EEH
```

Listing na K15 16

```
14          ELSE
15          ORG       2EE
```

Wij raden u aan het K-commando voor grote gebieden te gebruiken, maar voor kleine aantallen te verwijderen regels kunt u beter de editor gebruiken, omdat deze een betere visuele controle op het verwijderen geeft. Bovendien zijn bij de editor de oude regels gemakkelijk door [break][<-] te herstellen.

W
Dit schrijft de sourcefile weg naar het achtergrondgeheugen, na eerst een compact blok te hebben gevormd door de sourcecode tegen de symboltable aan te schuiven.
Om deze twee na het inlezen weer te kunnen scheiden wordt in het laatste waardeveld van de symboltable een pointer geplaatst die het beginadres van symboltable aanduidt, en in het op een na hoogste waardeveld een pointer die het begin van de sourcecode aanduidt. Na het wegschrijven wordt de oude toestand weer hersteld. Omdat alleen gesorteerde symboltables gemergd kunnen worden (zie het Y-commando), mogen alleen sourcefiles met gesorteerde symboltables (zie het Q-commando) worden weggeschreven. Hier wordt door SPL op gecontroleerd.

R[adres0] [string0]
Dit dient voor het inlezen van een sourcefile van een achtergrondgeheugen.
Adres0 geeft een offset aan waardoor ook sourcefiles die door een SPL-assembler met een ander top-of-symboltable-adres (zie het Z-commando) zijn weggeschreven correct kunnen worden ingelezen. Hierbij moet de offset ervoor zorgen dat het einde van de sourcefile op het top-of-symboltable-adres van de eigen SPL-assembler terecht komt.
Wordt er geen string0 meegegeven, dan wordt de volgende file van DCR of cassette ingelezen. Wordt er wel een string0 meegegeven dan wordt alleen een file met die naam van DCR, cassette of disk ingelezen.

Zoals bij het W-commando staat beschreven wordt een sourcefile weggeschreven als een compact blok, waarbij de sourcecode tegen de symboltable is geschoven met in de bovenste twee waardevelden de beginadressen van resp. sourcecode en symboltable. Is dit laatste adres kleiner dan het startadres van de sourcebuffer, dan volgt een foutmelding. Is dit niet het geval, dan wordt de sourcecode omlaag geschoven tot het begin hiervan samenvalt met het startadres van de sourcebuffer.
Het R-commando schrijft de eventueel al in de sourcebuffer aanwezige sourcecode en symboltable over. Wil men dit voorkomen, dan moet men gebruik maken van het Y-commando.

Y[adres0] [string0]
Dit dient voor het samenvoegen van de in de sourcebuffer aanwezige sourcecode met een sourcefile dat wordt ingelezen van een achtergrondgeheugen. De symboltables van de twee sourcefiles worden automatisch samengevoegd, waarbij gemeenschappelijke namen tot een enkele worden teruggebracht. Hiervoor is het nodig dat beide symboltables gesorteerd zijn (zie het Q-commando).
Zie voor de betekenis van adres0 en string0 het R-commando.

$
Dit commando word gebruikt om de pointers van de $-operands aan te passen na het inlezen van een sourcefile van een SPL-versie met een ander opstartadres. Het commando vraagt om dit adres met "Start?". Heeft u de sourcefile met een offset in moeten lezen, in verband met een ander top-of-symboltable-adres, dan moet u deze offset na het opstartadres intypen. In verband hiermee doet u er, indien u een SPL-versie met een ander opstartadres dan 8500H gebruikt of met een andere top-of-symboltable als 83FFH, verstandig aan dit bij het uitwisselen van sourcefiles nadrukkelijk te vermelden. Wanneer u het $-commando gebruikt, dan kunt u dit het beste direct na het inlezen doen. Het $-commando kunt u niet gebruiken voor een sourcefile die u met behulp van het Y-commando heeft ingelezen. Zo'n sourcefile moet u dus eerst apart inlezen, daarna met behulp van het $-commando aanpassen en dan weer wegschrijven.

dcr
Indien de opdracht begint met een spatie, dan wordt wat er op volgt geinterpreteerd als een DCR-commando (mits geimplementeerd).
Een voorbeeld:
```
 REW
```
Dit spoelt de DCR-cassette terug tot het begin van de band.

?
Dit commando list de structuurfouten in de sourcecode tot het eind van de sourcecode of totdat een foutboodschap wordt gegeven.
In tegenstelling tot de andere objectroutines, die alleen de code van de foutboodschap printen, list het ?-commando na deze code ook de regel waar de fout werd geconstateerd of, bij #MP#, de aanroepende regel.
Wij raden u sterk aan een A-commando altijd door een ?- en een N-commando vooraf te laten gaan.
Een voorbeeld:
Plaatst u in RESTORE XXXX in regel 56 achter MACRO de operand X, dan zal het ?-commando het volgende listen:
```
#MP#  49      CMPLNR  getbyt
```
Zie ook de aparte hoofdstukken met de beschrijving van de fouten.

A [adres0]
Dit commando geeft SPL opdracht de sourcecode in objectcode om te zetten ("te assembleren"), waarbij de bytes van het objectprogramma in het geheugen worden geplaatst.

De assemblage geschiedt in drie fasen:
- De zoekfase, waarin een aantal foutcontroles geschieden ten behoeve van een snellere verwerking van de volgende twee fasen. In deze fase vindt tevens de controle op de opbouw van het programma plaats (zie het desbetreffende hoofdstuk).
- De zetfase, waarin SPL de waardevelden in de symboltable instelt.
- De plaatsfase waarin de objectcode wordt gegenereerd (en bij deze opdracht in het geheugen wordt geplaatst).

Aangezien iedere call-macro een eigen zet- en plaatsfase voor zijn eigen macrodefinitie uitvoert tijdens zowel de zet- als de plaatsfase van het aanroepende blok (het hoofdprogramma of een andere macro), kan door het gebruiken van veel (geneste) macro's de assemblagetijd aanzienlijk worden verlengd.
Een voorbeeld ter verduidelijking:

```
        ORG     400H
        DB      1H
        alpha
        END
alpha   MACRO
        DB      2H
        beta
        MEND
beta    MACRO
        DB      3H
        MEND
```

Bij een zetfase van alpha wordt voor beta zowel een zetfase als een plaatsfase uitgevoerd. Ditzelfde gebeurt tijdens een plaatsfase van alpha. Aangezien alpha tweemaal zowel een zet- als een plaatsfase doorloopt, in verband met de zet- en de plaatsfase van het hoofdprogramma, doorloopt beta dus viermaal een zet- en een plaatsfase.
Wordt er bij het A-commando geen adres0 (offset) meegegeven, dan komt de objectcode op de adressen terecht zoals die tijdens een objectlisting gelist worden, met andere woorden: het programma staat dan op zijn plaats en zou, indien het foutloos is geschreven, in Utility moeten kunnen worden opgestart m.b.v. de G-opdracht. Wordt er wel een offset meegegeven, dan wordt deze in de plaatsfase telkens bij de operands van de ORG-directives opgeteld. Dit heeft als effect dat een byte van de objectcode dat eerst op adres X geplaatst zou worden nu op adres X+offset terecht komt, zonder dat de bytes zelf veranderen. Het objectprogramma kan m.b.v. de Utility M-opdracht over een afstand, gelijk aan 0H-offset, verplaatst worden, en staat dan klaar om uitgetest te worden.
Een cijfervoorbeeld: loopt het objectprogramma volgens de ORG-directives (en de objectlisting) van 2EEH tot 326H, en is het met een offset van 1000H geassembleerd, dan staat het na de assemblage in het geheugen op de adressen 12EEH tot 326H en kan in Utility door M12EE 1326 2EE op zijn plaats worden gebracht. Uh kunt de objectcode ook naar een achtergrondgeheugen wegschrijven en dan met een egatieve offset inlezen. In het voorbeeld: (Utility) W12EE 1326, gevolgd door RF000. Dit is echter wat omslachtiger.
Het assembleren met een offset is voornamelijk bedoeld om objectcode die normaal ergens geplaatst zou worden waar het de assembler, de sourcecode of de computerhuishouding zou aantasten ergens in het geheugen te plaatsen waar het geen kwaad kan tot de assemblage gereed is.
Tijdens de assemblage mag de objectcode de sourcecode eventueel gedeeltelijk overschrijven, mits er geen objectcode geplaatst wordt op adressen waar sourcecode staat die nog geassembleerd moet worden. Gewoonlijk worden er meer bytes sourcecode verwerkt dan er bytes objectcode worden geproduceerd, maar u moet speciaal rekening houden met ORG, DS, call-macro's en controls. U dient er ook rekening mee te houden dat dit overschrijven de source vernietigt.

U
Dit zorgt voor een sprong naar Utility. Er worden echter eerst checksums van de assembler, de sourcecode en de symboltable opgesteld, die bij het herstarten van SPL worden gecontroleerd.

u
Dit zorgt voor een sprong naar BASIC. Ook hierbij worden checksums opgesteld.

Alfabetische commandolijst
==========

: DCR-commando
$: Aanpassing pointers bij file van andere SPL-versie
?: Error-listing
^: Mode 0
A: Assemblage
B: Adreslisting binair
b: Bytelisting binair
C: Cold start
D: Adreslisting decimaal
d: Bytelisting decimaal
E: Edit
F: Memorymap (labels in volgorde van waardetoekenning)
G: Gebruik van naam in regel (+ gebied eromheen)
g: Cross-reference listing
H: Adreslisting hexadecimaal
h: Bytelisting hexadecimaal
I: Zoek naam in labelkolom
i: Listing van namen in labelkolom
J: Verschuif begin sourcebuffer
K: Verwijder regels
L: Normale listing
l: List gebruik van instructie
M: Copieer regels
m: Verplaats en verwijder regels
N: List niet gebruikte namen
O: Adreslisting octaal
o: Bytelisting octaal
P: Stuur byte (naar printer)
p: Aantal regels per pagina (hardcopy)
Q: Sorteer symboltable
R: Lees file van achtergrondgeheugen
S: List namen + waarden
T: Objectlisting
U: Utility
u: BASIC
V: Hardcopy vlag
v: Initialiseer titel, pagina- en regelteller
W: Schrijf file naar achtergrondgeheugen
X: Verander namen en/of constanten
Y: Voeg files samen
Z: Toon pointers

De editor

Bij het E-commando worden eerst de eventuele namen in de commandoregel vertaald in regelnummers. Hierna wordt in de sourcebuffer ruimte gemaakt voor de editbuffer. Vervolgens wordt het te editeren deel in ASCII-vorm in de editbuffer geplaatst. Past het te editeren deel niet in de editbuffer, dan wordt de oude situatie hersteld en wordt met de boodschap 'BREAK' teruggekeerd naar de commandoinvoer (dit gebeurt ook indien een regel in ASCII-vorm meer dan 224D karakters telt, zoals voor kan komen bij het binaire listen van een call-macro met veel constanten in de operandkolom). Past het te editeren deel wel in de editbuffer, dan kunt u de tekst bewerken zoals bij BASIC's EDIT. Bestudeer hiervoor het instructieboekje van de DAIpc. Na het bewerken van de tekst kunt de editor verlaten door [break] in te drukken. SPL wist het beeldscherm en wacht tot een van de volgende toetsen wordt aangeslagen:

[return] : Sourcecode noch editbuffer wordt gewijzigd en er wordt teruggekeerd naar de editor.
Dit kunt u gebruiken als u [break] per ongeluk heeft aangeslagen.

[<-] : De toestand van voor het E-commando wordt hersteld. De inhoud van de editbuffer wordt dus genegeerd. Dit is te vergelijken met [break] [break] in BASIC'S EDIT, maar omzeilt het eventuele denderen van [break]. Het indrukken van [break][<-] heeft geen effect als er een syntaxfout is geconstateerd, dus als de cursor met een foutmeldingskarakter knippert.

[ ] : De tekst uit de editbuffer wordt omgezet in sourcecode, eventueel met aanvullingen op de symboltable. Bij het constateren van een syntaxfout verschijnt de regel met de fout in de editbuffer bovenaan het scherm, en als cursor symbool knippert er een karakter dat het type syntaxfout aangeeft. Dit wordt herhaald tot de fout hersteld is.

SPL maakt gebruik van de editroutine in het DAIrom. Hierbij is er gebruik gemaakt van de tabulatieroutine, zodat het indrukken van [tab] de cursor naar een bepaalde karakterpositie laat springen. Welke positie dat is hangt af van de vorige plaats van de cursor en de gebruikte tabulatietabel. De [tab] laat op de oude plaats van de cursor een "tab" (pijltje naar rechts) achter. Net zoals u rechts van een "return" (zo'n vishaakje) geen karakters kunt plaatsen, kunt u ook tussen een tab en de kolom waar het naar verwijst geen karakters plaatsen.

De cursor is tijdens de editroutine gewoonlijk geen karakter (zoals '_'), maar een verandering van voor- en/of achtergrond kleur. Standaard worden voor de nieuwe voor- en achtergrondkleur respectievelijk 0D en 12D gebruikt. Dit is echter implementabel.

Het scherm is als het ware in 4 kolommen verdeeld. Dit zijn van links naar rechts de labelkolom, de instructiekolom, de operandkolom en de commentaarkolom. Elke kolom wordt, indien gebruikt, afgesloten door een tab of return.

De labelkolom

In de labelkolom kan een van de vier volgende combinaties worden geplaatst:

1) Een puntkomma, eventueel gevolgd door andere karakters.
   Dit geeft een remark aan, en is voor SPL niet zo interessant. In feite vervallen hierbij de 1e drie kolommen, zodat alleen de commentaarkolom overblijft (hoewel deze wel over de volle breedte van de regel wordt gelist).
2) Een identifier.
   Een identifier wordt gebruikt om een bepaalde plaats in het assemblerprogramma aan te geven, zoals de plaats waar een macrodefinitie of een entryadres voor lusstructuren staat.
   Bv: loop ###

3) Een label.
   Een label geeft een bepaalde waarde aan, bijvoorbeeld een adres in het objectprogramma, een bepaalde constante waarde of een wissel voor conditionele directives.
   Bv: COUNTER SET COUNTR-1H
4) Niets.
   Dit is een sluitpost als er geen gebruik van de drie andere mogelijkheden wordt gemaakt.
   Bv: END

De instructiekolom

In de in de instructiekolom kan een van de volgende vier combinaties worden geplaatst:

1) Een standaard Intel 8080-mnemonic (met uitzondering van IN, OUT of MOV R,R zoals MOV H,H).
   Let wel : In de instructiekolom mogen geen operands staan. Wilt u bijvoorbeeld de accumulator laden met 0DH, dan wordt dit bij andere assemblers meestal aangeduid met MVI A,0DH. Bij SPL komt echter in de instructiekolom "MVI A", gevolgd door een tab en in de operandkolom 0DH. Als vuistregel kunt u hierbij aanhouden dat alleen de representatie van het eerste byte van een machinetaalinstructie in de instructiekolom wordt geplaatst. Zie ook de bijgeleverde sourcefiles.
   Bv: MVI A 0DH
2) Een SPL-directive.
   Zie hiervoor het hoofdstuk over assemblerdirectives. Ook hierbij horen de operands niet in de instructiekolom thuis.
   Bv: PUT "D"
3) Een identifier.
   Het gebruik hiervan valt in twee groepen te verdelen:
   I) Macro-aanroep: hierbij staan in de operandkolom (eventueel) de waarden die als operand bij de aanroep worden meegenomen en aan de labels na de MACRO-directive worden toegekend. Deze operands evalueren dus tot een of meerdere 16-bits nummerieke waarden.
   Bv: getbyt
   II) Controle instructies: hierbij staat in de operandkolom een vergelijking die na evaluatie een 'waar' of 'onwaar' conditie ("vlag") oplevert.
   Bv: loop COUNTR>0H

De operandkolom

Wat hierin mag staan is afhankelijk van wat er in de label en instructiekolom staat.

1) Na mnemonics die een 1-byte machinetaalinstructie aangeven is geen operand toegestaan.
   Bv: STC
2) Na mnemonics die een 2-bytes machinetaalinstructie aangeven mag in de operand kolom het volgende staan:
   a) een combinatie van labels, getallen en operators die na evaluatie een waarde kleiner of gelijk aan 0FFH, doch groter of gelijk aan 0H, oplevert.
   Bv: MVI A .0H-COUNTR&0FFH
   b) 'X', waarbij X een willekeurig karakter voorstelt.
   Dit levert in het objectprogramma een byte op met als waarde de ASCII-waarde van het karakter.
   Bv: CPI ' '
   c) "X", waarbij X een willekeurig karakter voorstelt.
   Dit levert in het objectprogramma een byte op met als waarde 80H, vermeerderd met de ASCII-waarde van het karakter.
   Bv: SUI "X"
3) Na mnemonics die een 3-bytes machinetaalinstructie aangeven mag in de operandkolom een combinatie van labels, getallen en operatoren staan.
   Bv: COUNTR SET COUNTR-1H

4) Wat er na SPL-directives mag staan is afhankelijk van de directive. Zie het desbetreffende hoofdstuk.
   Bv: ORG 750D
Wat er in de operandkolom staat mag eventueel doorlopen tot in de commentaar-kolom.

De commentaarkolom
De commentaarkolom moet altijd beginnen met een puntkomma. Wat er achter deze puntkomma staat wordt als commentaar beschouwd en door de assemblagefasen genegeerd. Als beperking geldt dat commentaar alleen tab's mag bevatten als het onderdeel van een remark vormt (puntkomma aan het begin van de labelkolom).

Indien de editor weigert meer karakters op het scherm te plaatsen, en dit niet wordt veroorzaakt door de positie van de cursor ten opichte van een tab of return, dan is waarschijnlijk de editbuffer vol. De oplossing hiervoor is of het verlaten van de editor of het verwijderen van een aantal karakters.

Mnemonics

SPL maakt gebruik van de standaard Intel-8080 mnemonics, met uitzondering van de MOV R,R mnemonic (b.v. MOV A,A) en de IN en OUT mnemonics, die in SPL niet zijn gedefinieerd. Wenst u toch gebruik te maken van deze machinetaalinstructies, dan kunt u ze met behulp van DB invoeren.
Voor alle mnemomics mag in de labelkolom een label staan.
De mnemonics verhogen naar gelang van het type mnemonic de objectpointer met 1, 2 of 3 bytes.

Assemblerdirectives

Voor de meeste directives mag in de labelkolom een label staan. Bij remark kan er geen label voor staan omdat de labelkolom al met de puntkomma van de remark-regel begint.
Voor EQU en SET moet een label staan en voor ### en MACRO een identifier. Voor de duidelijheid zijn bij de syntaxbeschrijvingen de niet noodzakelijke labels in de labelkolom weggelaten.
De objectpointer wordt door DW met 2 verhoogd, door DB met het aantal bytes objectcode wat hij genereert, en door DS met de waarde van diens operand. Verder wordt de objectpointer door ORG ingesteld en door de andere directives niet beinvloed.

```
          ORG     waarde0
Bv:       ORG     300H
```

ORG zet de objectpointer op waarde0.
Als er voor ORG een label staat, dan krijgt dit ook de waarde waarde0.

```
          END
Bv: EIND  END
```

END stopt bij objectroutines de plaatsfase.

```
          DB      waarde0
Bv:       DB      ESC
```

In deze vorm geeft DB een byte aan die in het objectprogramma op het door de objectpointer aangeven adres moet worden geplaatst. Hierom moet de waarde van waarde0 tussen 0H en 0FFH liggen (het moet in 1 byte passen).

```
          DB      byte1(,byteN)...
Bv:       DB      0A0H,10H,7H
```

In deze vorm geeft DB een reeks van bytes aan die in het objectprogramma opeenvolgend in het geheugen komen vanaf het door de objectpointer aangegeven adres.

```
          DB      'string0'
Bv: STR001 DB     'ASCII'
```

Deze vorm van DB werkt ongeveer gelijk aan de vorige, alleen wordt er een reeks door string0 gedefinieerde karakters als bytes in het geheugen geplaatst met MSB=0.

```
          DB      "string0"
Bv:       DB      "TEKST"
```

Deze vorm van DB werkt analoog aan de vorige, echter met MSB=1.

```
          DW      waarde0
Bv: VAL.XY DW     BUFFER+OFFSET
```

DW definieert een 16-bits waarde die in het objectprogramma in het geheugen komen met het lowbyte op het adres aangegeven door de objectpointer en het highbyte op dit adres +1 (analoog aan SHLD).

```
            DS      waarde0
Bv:         DS      ALFA#FFH
```

DS verhoogt de objectpointer met de waarde van waarde0.
Het is hierbij wel nodig dat deze waarde al tijdens de zetfase gedefinieerd is.

```
   label0   EQU     waarde0
Bv: OUTPUT  EQU     0D695H
```

EQU zet de waarde van waarde0 in het waardeveld van label0. Het is hierbij wel nodig dat deze waarde al tijdens de zetfase gedefinieerd is.

```
   label0   SET     waarde0
Bv: TELLER  SET     $+1
```

SET werkt analoog aan EQU, alleen mag het label0 al vaker in de labelkolom zijn voorgekomen, zonder een Doubledefined error op te leveren. SET is in feite de enige instructie die mag worden gebruikt als een label twee keer in de labelkolom verschijnt. U kunt SET onder meer gebruiken als voorbereiding voor IF en bij call-macro's om extra operands door te kunnen geven.

```
   ident0   MACRO   [label1]([,labelN])...
Bv: alpha   MACRO   A,B
```

MACRO duidt het begin van een macrodefinitie aan.
Telkens wanneer SPL tijdens het assembleren in de instructiekolom ident0 tegenkomt, dan worden de instructies tussen de MACRO en de MEND van ident0 MEND geassembleerd. Alle macrodefinities moeten na de laatste END in het assemblerprogramma zijn opgenomen.

```
            ident0  [waarde1]([,waardeN])...
Bv:         alpha   0CA70H,BUFFER+OFFSET
```

Dit is een macroaanroep ("call-macro"). De waarden van de waarde0 enzovoorts worden d.m.v. een soort SET in de waardevelden van de labels die de MACRO operanden vormen geplaatst. Het aantal waarden moet gelijk zijn aan het aantal labels.

```
            MEND
Bv: RETURN  MEND
```

MEND geeft het eind van een macrodefinitie aan. Als SPL tijdens het assembleren van een macrodefinitie een MEND tegenkomt, dan wordt verder gegaan met de instructies die op de call-macro volgen.

```
            IF      vlag0
Bv: LH1200  IF      $=TELLER
```

Als vlag0 'waar' is, dan worden tijdens de assemblage ook de instructies tussen IF en de bijbehorende ENDIF (of ELSE) geassembleerd en de eventueel aanwezige instructies tussen ELSE en ENDIF niet. Is de vlag 'onwaar' dan gebeurt het tegenovergestelde.

```
            ELSE
Bv:         ELSE
```

ELSE geeft de scheiding aan tussen wat geassembleerd moet worden indien de vlag achter de bijbehorende IF 'waar' is (de instructies tussen IF en ELSE) en wat er anders geassembleerd moet worden (de instructies tussen ELSE en ENDIF). Er hoeft geen ELSE tussen IF en ENDIF te staan.

```
            ENDIF
Bv:         ENDIF
```

ENDIF geeft het einde aan van het gebied dat, afhankelijk van de vlag achter de bijbehorende IF, al dan niet geassembleerd moet worden.

```
   ident   ###
Bv: contrl  ###
```
("entry") geeft een punt in het assemblerprogramma aan waar de assemblage hervat moet worden na het uitvoeren van een control.

```
          ident0  vlag0
Bv: START contrl  X"Z
```
Deze instructie ("control") vervolgt, indien de vlag 'waar' is, de assemblage met de instructies die volgen op de ### waarvoor ident0 staat. Is vlag0 'onwaar', dan worden de instructies die op de control volgen geassembleerd. De control onderscheidt zich uiterlijk van de call-macro doordat de operand tot een vlag evalueert en niet tot een of meer nummerieke waarden.
Terwijl de call-macro een zekere gelijkenis vertoont met een subroutine-call, heeft de control meer weg van een voorwaardelijke sprong. De control mag alleen terug springen, dus de bijbehorende ### moet eerder in de sourcecode staat. Maakt de control deel uit van een macrodefinitie, dan mag er niet verder terug worden gesprongen dan de MACRO van die definitie.
Waarschuwing: het is mogelijk, net als bij machinetaalprogramma's, door een verkeerd gebruik van control een eindeloze lus te creeren. Hier kan door SPL niet op gecontroleerd worden, dus dit moet u zelf doen.

```
          TITL    'string0'
Bv: PART2 TITL    'SPL par SPHYNX'
```
Bij hardcopy print SPL bovenaan elke nieuwe pagina vanaf de labelkolom de string die in de operandkolom van de laatste TITL stond (na een v-commando wordt hiervoor een lege string gebruikt tot er weer een TITL wordt bereikt).

```
          UNL
Bv:       UNL
```
UNL onderdrukt tijdens een objeclisting het sturen van karakters naar scherm en/of printer tot SPL een LST tegenkomt of de objectlisting is afgelopen.

```
          LST
Bv:       LST
```
LST herstelt tijdens een objectlisting het sturen van karakters naar scherm en/of printer. In combinatie met UNL kan dit gebruikt worden om slechts een bepaald gebied te listen.

```
          PUT     "char0"
Bv:       PUT     "O"
```
Char0 bestaat uit "B", "b", "O", "o", "D", "d", "H" of "h". PUT stelt hierbij de uitvoerwissels voor adressen of bytes opnieuw in, net alsof er respectievelijk een B, b, D, d, H, h, O of o commando zou zijn gegeven. Het doel van deze directive is om vooral in listings de nummerieke waarden wat duidelijker uit te laten komen.

```
          PUT     byte0
Bv:       PUT     60D
```
PUT stelt in deze vorm het aantal regel per pagina bij de hardcopy routines in. Dit aantal regels wordt aangegeven door de waarde van byte0, waarbij 0D <= byte0 <= 127D. Is deze waarde 0D, dan wordt er doorlopend gelist (dus geen formfeeds, titels e.d.). Het nieuwe aantal regels per pagina treedt pas bij de volgende pagina in werking, dus de huidige pagina krijgt nog het oude aantal regels. Wilt u toch met een nieuwe pagina beginnen, dan kan dit met behulp van PRT.

```
            PRT     byte1(,byteN)...
Bv:         PRT     0CH,0DH
of          PRT     'string0'
Bv:         PRT     'Nieuwe pagina'
of          PRT     "string0"
Bv:         PRT     "Sommige printers werken met MSB=1"
```

Tijdens de hardcopy routines zendt SPL, als hij een PRT tegenkomt, de bytes uit de operand hiervan naar de printer. Hierdoor kan men de printer speciale opdrachten geven, zoals bijvoorbeeld een extra formfeed (in andere assemblers soms bestuurd door een 'PAGE') of een ander letter type. Zie voor specifieke informatie ook de handleiding van uw printer.
Let wel: als PRT een 0CH of 0DH byte verstuurt beinvloedt dit de regelteller (en eventueel de paginateller). Een 0AH byte heeft hier geen invloed op.

```
  ;[string0]
Bv: ;Main Program.
```

Staat er in de eerste karakterpositie van de labelkolom een puntkomma, dan wordt de rest van de regel als een remark beschouwd en door de meeste routines genegeerd. In een remark zijn wel tabs toegestaan, dit in tegenstelling tot het commentaar achter andere instructies.

Operators

Operators voeren een rekenkundige bewerking uit op het resultaat van de labels, getallen en operators die er links van staan (vanaf de laatste alternator, comparator, komma of tab), aangeduid met X, en het label of getal dat er rechts van staat, aangeduid met Y. Alle operators geven als resultaat een 16-bits waarde. (Met positief of negatief wordt geen rekening gehouden). Een eventuele overflow wordt genegeerd.

```
X#Y : X logisch of Y
X%Y : X exclusief of Y
X&Y : X logisch en Y
X'Y : X modulo Y
X(Y : X shiftleft Y [rechts aangevuld met 0-bits, Y<=16D]
X)Y : X shiftright Y [links aangevuld met 0-bits, Y<=16D]
X*Y : X maal Y
X+Y : X plus Y
X-Y : X min Y
X/Y : X gedeeld door Y
```

De volgende operators zijn niet in SPL opgenomen:

```
  NOT X, te vervangen door X%0FFFFH
MINUS X, te vervangen door 0H-X
 HIGH X, te vervangen door X)8H
  LOW X, te vervangen door X&0FFH
```

Comparators

Comparators worden gebruikt om de combinatie van labels, getallen en operators die er links van staat (vanaf de laatste alternator of tab), aangeduid met X, te vergelijken met de combinatie van labels, getallen en operators die er rechts van staat tot de volgende alternator, tab of return), aangeduid met Y. Alle comparators produceren een 'waar' of 'onwaar' vlag, die alleen door een alternator of een conditionele directive (IF of control) kan worden gebruikt. Alle comparators vergelijken 16-bits waarden, waarbij met positief of negatief zijn geen rekening wordt gehouden. 0FFFFH wordt dus als groter beschouwd dan 0H.

```
X<Y : 'waar' als X kleiner  is dan Y, anders 'onwaar'
X>Y : 'waar' als X groter   is dan Y, anders 'onwaar'
X=Y : 'waar' als X gelijk   is aan Y, anders 'onwaar'
X"Y : 'waar' als X ongelijk is aan Y, anders 'onwaar'
```

Alternators

Alternators worden gebruikt voor het combineren van het resultaat van de combinatie van labels, getallen, operators, comparators en alternators die er links van staat (vanaf de laatste tab), aangeduid met X, en de combinatie van labels, getallen, operators en comparators die er rechts van staat (tot de volgende alternator, tab of return), aangeduid met Y. Alle alternators vergelijken 'waar' en 'onwaar' vlaggen, zoals geproduceerd door comparators of andere alternators, en leveren zelf weer een 'waar' of 'onwaar' vlag.

```
X!Y : 'waar' indien zowel X als Y 'waar' zijn, anders 'onwaar' ("AND")
X?Y : 'waar' indien X of Y 'waar' is, anders 'onwaar' ("OR")
```

Er is sprake van een syntaxfout indien onmiddellijk links of rechts van een operator, comparator of alternator geen label of getal staat.

Separators

Separators worden gebruikt om het einde van een operand aan te geven. Een separator bestaat uit een operator, comparator, alternator, komma, tab, return of spatie. De spatie wordt als separator gebruikt indien u bijvoorbeeld bij het X-commando het einde van de operand aan wilt duiden.

Bij gelijksoortige seperators wordt de operand altijd van links naar rechts geevalueerd.

Speciale karakters in de operand-kolom

Indien na een instructie die een 1-byte waarde verwacht de operandKolom begint met een punt ("."), dan wordt het daarop volgende getal opgevat als een 2-bytes waarde. Het voordeel van zo'n getal is dat het b.v vervangen kan worden door een label of dat het het begin van een samengestelde operand kan vormen.

Indien op een plaats, waar in de operandkolom normaal een label of getal mag staan, een dollarteken ("$") staat (niet toegestaan in de zelfde regel als een MACRO-directive), dan wordt bij de evaluatie aan dit symbool de actuele waarde van de objectpointer toegekend.

U mag alle karakters die geen separators zijn en waarvan de ASCII-waarde tussen de 20H en 7FH (inclusief) ligt in een naam gebruiken, mits de naam er niet mee begint. Dit zijn in de praktijk de volgende karakters:

```
0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
$.:;[]^~
```

Opbouw van een assemblerprogramma

Hoewel gestructureerd programmeren bepaalde nadelen kan hebben, biedt een gestructureerde opbouw van een programma ook voordelen, vooral waar het gaat om het voorkomen van fouten. SPL laat u volkomen vrij in hoe u het objectprogramma wilt opbouwen, maar stelt wel bepaalde eisen met betrekking tot het gebruik van sommige directives en hun onderlinge relaties. Dit is dan ook de reden dat een assemblerprogramma op een bepaalde manier moet zijn opgebouwd. De wijze waarop deze opbouw moet geschieden kan het beste worden weergegeven in zogenaamde syntaxdiagrammen:

```
programma        := (code)...
                    END
                    (list_directive)...
                    (macro_definitie)...
macro_definitie  := MACRO
                    (code)...
                    MEND
                    (list_directive)...
code             := mnemonic
                 := special
                 := blok
blok             := if_blok
                 := controle_blok
                 := call-macro
if_blok          := IF
                    (code)...
                    [ELSE]
                    (code)...
                    ENDIF
controle_blok    := ###
                    (code)...
                    control
special          := ORG
                 := EQU
                 := DW
                 := DS
                 := DB
                 := SET
                 := list_directive
list_directive   := PRT
                 := LST
                 := PUT
                 := TITL
                 := UNL
                 := remark
```

Hierin staan links van de ":=" de woorden waarvan de opbouw wordt beschreven. Rechts van de ":=" staat de opbouw van het woord in andere woorden, of basis-begrippen, van boven naar beneden tot aan de volgende ":=". Meerdere ":=" geven verschillende alternatieven voor de opbouw weer.
Basisbegrippen zijn de woorden in hoofdletters, call-macro, ###, control en remark. Deze behoren tot de assemblerdirectives en staan in het desbetreffende hoofdstuk beschreven. Dit geldt ook voor de mnemonics, die ook tot de basis-begrippen worden gerekend.
Wat tussen "[" en "]" staat mag u desgewenst weglaten. Wat tussen "(" en ")..." staat mag u meerdere keren herhalen, of naar keuze helemaal niet gebruiken.

Als ezelsbruggetje kunt u zich voorstellen dat in listingen de volgende combinaties onderling door boogjes verbonden worden die elkaar niet mogen kruisen (de boogjes staan allemaal voor de linkerkantlijn):

```
begin_sourcecode (~) END (~) eind_sourcecode
MACRO (~) MEND
IF (~) ELSE (~) ENDIF of IF (~) ENDIF
### (~) control (met een gelijke identifier)
```

Dit is dus verkeerd:

```
  /===.........IF        X"Y
 ¦              NOP
 ¦  /-.contrl   ###
 ¦ I            NOP
 )=#==.........ELSE
 ¦ I            NOP
 ¦  \-.........contrl   A=0H
 ¦              NOP
 \====.........ENDIF
```

Zoals u ziet kruisen de boogjes elkaar bij "#".

Een voorbeeld van hoe het wel moet:

```
  /---.callm    MACRO    A,X
 ¦              NOP
 ¦  /=.........IF       A=X
 ¦ ¦            NOP
 ¦  \=.........ENDIF
 ¦              NOP
  \---.........MEND
```

Zoals u ziet kruisen de boogjes elkaar niet.

Syntaxfouten

Dit zijn de fouten die na het verlaten van de editor met [break][ ] worden herkend door de routine die de inhoud van de editbuffer vertaalt naar SPL sourcecode. Wordt een syntaxfout herkend, dan stopt de vertaling van ASCII naar sourcecode, het begin van de editbuffer wordt op het eerste karakter van de regel met de syntaxfout geplaatst en er wordt teruggekeerd naar de editor. Tevens wordt in het cursorsymbooladres (75H) een karakter geplaatst dat het type fout aangeeft.

! Alternator-error.
  Na bv: LXI H A?B
  Er wordt in de regel '!' of '?' gebruikt op een plaats waar dat niet kan.

& Separator-error.
  Na bv: DW $;
  Er staat in de regel iets anders op een plaats waar een separator wordt verwacht.

' Tick-error.
  Na bv: TITL 'SPL
  Er ontbreekt een ' of een " in de regel.
  Na bv: DB ""
  Tussen '' of "" staat niets (een lege string).

, Komma-error.
  Na bv: LXI B 0H,1H
  Er is in de regel een komma gebruikt op een plaats waar dat niet kan.
  Na bv: DB 1H1H
  Er staat in de regel iets anders op een plaats waar een komma word verwacht.

; Remark-error.
  Na bv: RST 5 Call screen
  Voor het commentaar ontbreekt in de regel een puntkomma.

= Comparator-error.
  Na bv: IF A=B=C
  Er ontbreekt een <, =, > of " of er staat er een te veel in de regel.

a Assembler-error
  Na b.v. XCHG
  Dit is eigenlijk geen syntax-fout, maar duidt op een fout in de assembler waardoor hij de "foute" regel niet in SPL-sourcecode kan vertalen. Om de editor toch te kunnen verlaten moet u deze regel verwijderen. Om deze fout te corrigeren moet u SPL opnieuw inlezen.

d Define-error.
  Na bv: LABEL MVI A 0H
  Er is geen ruimte genoeg in de symboltable om een nieuwe naam op te slaan. Dit kan een tijdelijke kwestie zijn (als SPL de benodigde ruimte in de symboltable op dat moment niet vrij kan maken) of permanent (als er al 1624D namen in de symboltable staan, zie het Q-commando). Om de editor te kunnen verlaten moet u de regel met de naam verwijderen.

i IO-error
  Na bv: PUT "Q"
  Er staat in de regel een verkeerde operand na een PUT-directive.
  Na bv: PUT 150D
  Er staat in de regel een te hoge waarde als operand van een PUT-directive.

l Label-error
  Na bv: %ABCD DB   100,350
  Er staat in de regel een verkeerd karakter aan het begin van de labelkolom.
  Na bv: PRT   A
  Er staat in de regel een label (of $) als operand van een PRT-directive.
  Na bv: SET $
  Er staat in de regel geen label voor een EQU- of SET-directive.
  Na bv: alpha NOP
  In de regel wordt een identifier in de labelkolom niet gevolgd door ###- of
  MACRO-directive in de instructiekolom.
  Na bv: OMEGA ###
  Er staat in de regel voor een ###- of MACRO-directive geen identifier.

m Mnemonic-error.
  Na bv: ASC   'STRING'
  In de regel staat een woord in de instructiekolom dat noch als mnemonic noch
  als identifier wordt herkend.

n Numeric-error
  Na bv: identi MACRO   1H
  Er is in de regel een numerieke waarde gebruikt op een plaats waar dat niet
  mag.
  Na bv: MVI A   300D
  In de regel is de als operand gebruikte nummerieke waarde te groot.
  Na bv: PRT   .10H
  In de regel begint de operand van een PRT-directive met een punt en is dus
  gedefinieerd als 2-bytes operand.
  Na bv: DB
  Er staat in de regel een spelfout in de nummerieke waarde die als operand is
  gebruikt.

o Restart-error.
  Na bv: RST 9
  Er staat in de regel geen cijfer (0-7) na RST.

p Passing-error.
  Na bv: label1 MACRO   A+B
  Er is in de regel in de operand van een MACRO-directive een andere separator
  dan komma, tab of return gebruikt.

r Register-error.
  Na bv: STAX H
  Er staat in de regel geen "B" of "D" na LDAX of STAX.
  Na bv: MOV C,C
  In de regel staan na MOV 2 gelijke registers genoemd.

s Space-error.
  Na bv: CALL   X Y
  In de regel wordt een operand gevolgd door een spatie en dat mag niet.

t End-error.
  Na bv: DB'ASCII'
  In de regel ontbreekt een tab of return op een plaats waar zoiets wel wordt
  verwacht.
  Na bv: NAAM   DB   'dit is een buitengewoon lange string die er voor zal
                         zorgen dat de regel te lang wordt'
  De gecodeerde regel zou meer dan 64D bytes in beslag nemen.
  In de regel staat een tab in commentaar.
  Na bv: LDA
  In de regel ontbreekt een operand.

Structuurfouten

Deze fouten worden door de objectroutines herkend tijdens een van de assemblagefasen. Een geconstateerde fout wordt als volgt gelist:
X regelnummer regel
Hierin stelt 'X' een foutcode voor.
In de beschrijvingen staat vermeld in welk assemblagefase de fout wordt herkend. Fouten die normaal in de plaatsfase worden herkend kunnen ook al in de zetfase worden herkend wanneer een instructie dan al een bepaalde waarde moet weten. Als vuistregel is hierbij te hanteren dat de zetfase geen waarden berekent die direct als objectcode in het geheugen worden geplaatst, maar wel waarden die de objectpointer beinvloeden.

D : Double defined error (zoekfase)
De in de labelkolom staande labelnaam is al eerder in de labelkolom voorgekomen. Dit hoeft echter geen problemen op te leveren, bijvoorbeeld indien het label in beide takken van een IF-ELSE-ENDIF constructie wordt gebruikt.
N.B. 'SET' levert geen double defined error op.

O : byte Overflow (plaatsfase)
Er wordt een waarde groter dan 0FFH gebruikt als enkel-byte operand.

S : Shift error (plaatsfase)
De waarde waarover een shift '(' of ')' plaats moet vinden is groter dan 15D.

U : Undefined error (plaatsfase)
Er is (nog) geen waarde aan een label toegekend (het label komt b.v. niet in de labelkolom voor).

Foutboodschappen

Deze bestaan uit het printen van een een foutmelding tussen twee '#', gevolgd door een sprong naar de commandoinvoer.
Het belangrijkste verschil met structuurfouten bestaat hierin dat de fout die een foutboodschap veroorzaakt catastrofale gevolgen kan hebben. Zo kan een shift-error op zich betrekkelijk weinig kwaad en is dus als structuurfout gedefinieerd, terwijl een Division-by-zero-error SPL in een eindeloze lus kan brengen en daarom een foutboodschap geeft en de routine afbreekt.
Bij de ?-routine wordt behalve de foutboodschap ook de regel waar de fout in optrad gelist (behalve bij #MP#, waar de aanroepende regel wordt gelist).
Ook bij de beschrijving van de foutboodschappen staat vermeld wanneer ze worden herkend.
Zie ook het hoofdstuk over de opbouw van assemblerprogramma's.

#AE# : After End error (zoekfase).
Na een END staat (buiten een macro-definitie) iets anders dan een remark, TITL, PUT, PRT, LST of UNL of er staat meer dan 1 END in de sourcecode.

#CT# ConTrol error (zoekfase).
Er staat een control zonder dat er op hetzelfde niveau een bijbehorende entry staat.

#DD# : Double Defined identifier error (zoekfase). LABEL vk 2x
De naam van een identifier staat voor de tweede keer in de labelkolom.

#DZ# : division by zero error (plaatsfase).
Rechts naast een /-operator staat een getal of label met de waarde 0D.

#FO# : File Overflow error (R- of Y-commando).
Bij het inlezen of mergen van een sourcefile was de ingelezen file groter dan de vrije geheugenruimte. U dient eerst de sourcebuffer te vergroten en daarna kunt u de sourcefile opnieuw inlezen. De sourcecode in het geheugen kunt u als vernietigd beschouwen (dus bij het Y-commando zowel de oude als de nieuwe sourcefile).

#IF# : IF error (zoekfase).
Er staat een ELSE of ENDIF, zonder dat er op hetzelfde niveau een IF staat.

#LO# : LOcal error (zoekfase).
Een label wordt met SET of door MACRO gedefinieerd dat al in een eerdere macrodefinitie (of in het hoofdprogramma) is gedefinieerd.

#MA# : MAcro error (zoekfase).
Er staat een MACRO zonder voorafgaande END of MEND.

#MP# : More Passed (of Missing Pass) error (zetfase).
Het aantal waarden in de operandKolom van een call-macro komt niet overeen met het aantal labels in de operandkolom van de aangeroepen MACRO.

#NO# : Nesting Overflow error (zoek- en zetfase).
Tijdens een objectroutine zijn er meer als 80D IF's en/of MACRO's genest, of er zijn meer dan 80D IF's en/of ###'s in de sourcecode genest.

#RC# : ReCursing error (zoekfase).
Een call-macro roept een macrodefinitie aan die eerder dan de call-macro in de sourcecode staat.

#UI# : Undefined Identifier error (zoekfase (control) of zetfase (call-macro)).
Een control of call-macro gebruikt een (nog) niet gedefinieerde identifier.

Invoerfouten

Deze foutmeldingen worden gegeven als u bij het invoeren van een commando een fout heeft gemaakt, of als het commando kan niet worden uitgevoerd. De foutmeldingen bestaan uit twee "#"-tekens met daartussen, ter onderscheiding van de foutboodschappen, twee kleine letters.
Bij de beschrijving van de foutmeldingen staat vermeld bij welke commando's ze kunnen voorkomen. Staat er niets vermeld, dan kunnen ze in principe bij elk commando optreden.
De foutmeldingen zijn:

#ce# : Compatability Error (X).
Een conflict tussen identifier en label of nummerieke waarde.

#cf# : Commando Fout. SYNTAX ERROR
Het eerste karakter is niet als commando herkend.

#cs# : CheckSum error (R,Y).
Er is een fout geconstateerd bij het inlezen van een sourcefile.

#dc# : DCr error (,R,W,Y).
De DCR heeft een fout geconstateerd.

#hc# : HardCopy error (P,v).
De hardcopy vlag stond niet aan.

#in# : INstruction error (I).
Er is geen geldige instructie gebruikt.

#ln# : Line Number error (E,K,L,M,m).
Er mankeerde iets aan de onderlinge verhouding van de regelnummers (bv. in verkeerde volgorde).

#mm# : MeMory error (C,J,M,m,X).
Er is te weinig geheugenruimte voor het uitvoeren van het commando.

#nm# : NaMe error (E,G,I,K,L,M,m,X).
Er is verkeerde (identifier of label) naam gebruikt.

#se# : Source (of Symbols) Empty error (F,g,i,L,S).
Er ontbreekt sourcecode, of de symboltable is leeg.

#sn# : Sorting Needed error (W,Y).
U heeft geprobeerd een ongesorteerde symboltable weg te schrijven of samen te voegen. U dient eerst het Q-commando te gebruiken.

#su# : Symboltable Underflow error (W).
Voor het wegschrijven van een sourcefile zijn minstens 2 (willekeurige) namen in de symboltable nodig, omdat de waardevelden hiervan worden gebruikt om er de beginadressen van de symboltable en de (verschoven) source-code in op te slaan. Als bij het W-commando een #su# foutmelding verschijnt dan moet u, eventueel met behulp van dummylabels, de symboltable tot minimaal 2 namen uitbreiden.

#sy# : SYstem error.
Er is iets mis met de assembler. U dient SPL op nieuw in te lezen.

#vo# : Value Overflow error ($,A,C,E,G,I,J,K,L,M,m,P,p,R,X,Y).
Er is een te grote nummerieke waarde gebruikt.

Steken ophalen voor gevorderden

Dit hoofdstuk is bedoeld voor verder in SPL gevorderden, die willen weten hoe ze een sourcefile, dat door een machinetaal programma is opgeblazen, kunnen herstellen. Hierbij moet u echter wel bedenken dat u slechts bij geringe aantallen fouten succes kunt verwachten. Zijn er in b.v. de sourcefile teveel bytes aangetast, dan kunt u vaak beter direct beginnen met het opnieuw inlezen van de backup sourcefile, indien nodig voorafgegaan door het inlezen van SPL. Verder mag u ook niet vergeten de fout die het ongeluk veroorzaakte op te sporen en te herstellen. Als basiskennis hiervoor is een inleiding in de opbouw van de sourcecode van SPL noodzakelijk, daarom volgt hier daarvan een beschrijving.

Een sourcefile bestaat in feite uit twee delen: de symboltable en de sourcecode. We zullen eerst de sourcecode behandelen.

De basiseenheid in de SPL-sourcecode is de sourceregel. Dit is de gecodeerde vorm van een assemblerregel, dus een regel tekst in de editor (of op een listing). In de sourcecode begint iedere sourceregel met een zogenaamd status-byte. Van dit byte hebben de afzonderlijke bits een bijzondere betekenis.
De 6 laagste bits (bit 0 tot en met bit 5) geven in binaire vorm het aantal bytes aan dat dit statusbyte scheidt van de volgende, met andere woorden de lengte van de sourceregel (exclusief het statusbyte) in bytes.
Als bit 7 (het hoogste bit, ook wel tekenbit genaamd), van het statusbyte gezet is, dan begint de assemblerregel met een naam in de labelkolom.
Bit 6 van het statusbyte heeft verschillende betekenissen, afhankelijk van de gebruikte instructie.

Bij 1-byte mnemonics, zoals ORA M, wordt bit 6 van het statusbyte niet gebruikt en is gelijk aan 0. Dit type mnemonics wordt na de instructiecode alleen door eventueel commentaar gevolgd. De aanwezigheid van commentaar blijkt altijd uit een grotere waarde van bit 0 tot en met bit 5 van het statusbyte, dan het totale aantal bytes dat voor het eventuele voorafgaande label of identifier, de instructiecode en de operands wordt gebruikt, dus bij 1-byte mnemonics groter dan 1B (geen label) of 11B (wel een label). Dit commentaar staat in ASCII-vorm (tekenbit = 0). De puntkomma aan het begin van het commentaar wordt niet in de sourceregel opgenomen.

Bij 2-byte mnemonics, zoals CPI 2H, is bit 6 van het statusbyte niet gezet (gelijk aan 0) als de operand als enkel-byte moet worden weergegeven. De waarde van dit byte wordt direct aangegeven door het op de instructiecode volgende byte. Hierachter volgt eventueel commentaar.
Is bit 6 van het status-byte bij 2-byte mnemonics gezet, dan zijn er weer verscheidene mogelijkheden:

- Wordt de instructiecode gevolgd door een byte 00H, of 80H, dan duidt dit op een operand, die na evaluatie een 16-bits waarde oplevert, waarvan het hoogste byte gelijk moet zijn aan 0. In de listings en in de editor wordt dit aangegeven door bij een nummerieke operand er een punt voor te plaatsen. Zie voor de opslagmethode van de operand de beschrijving bij de 3-byte mnemonics. Bit 7 van het 00H of 80H byte speelt hierbij dezelfde rol als bit 6 van het statusbyte bij 3-byte mnemonics (0: operand begint met getal, 1: operand begint met label).
- Wordt de instructiecode gevolgd door een byte met een waarde ongelijk aan 00H of 80H, dan stelt dit de ASCII-waarde van een karakter voor. ASCII-karakters met bit 7 = 0 worden gelist tussen twee ', ASCII-karakters met bit 7 = 1 worden gelist als het overeenkomstige ASCII-karakter met bit 7 = 0, echter tussen twee ".

Bij 3-byte mnemonics, zoals CALL OUTPUT, is de operand als volgt opgebouwd:
Na de instructiecode volgen twee bytes. Is bit 6 van het statusbyte niet gezet, dan stellen deze twee bytes een 16-bits nummerieke waarde voor, in de volgorde lowbyte highbyte (analoog aan 8080-instructies). Is bit 6 van het statusbyte wel gezet, dan vormen deze 2 bytes een pointer naar de symboltable (zie verderop in dit hoofdstuk). Na deze twee bytes zijn er drie mogelijkheden:

- Het aantal bytes in de regel (zoals aangegeven door bit 0 tot en met bit 5 van het statusbyte) is uitgeput. Het volgende byte is dan het statusbyte van een nieuwe sourceregel.
- Bit 7 van het volgende byte is niet gezet. Dit duidt op het begin van commentaar.
- Bit 7 van het volgende byte is gezet, terwijl de regel nog niet is uitgeput. Dit duidt op een operator, waarvan de ASCII-waarde zou zijn weergegeven door bit 0 tot en met bit 5 van dit byte, maar dan met bit 6 en bit 7 niet gezet. Bit 6 van het operatorbyte geeft aan of de volgende twee bytes als een nummerieke waarde of als een pointer naar een label beschouwd moeten worden, analoog aan bit 6 van het statusbyte met betrekking tot de eerste twee bytes van de operand. Na het operator byte met zijn twee waardebytes ontstaan opnieuw de drie mogelijkheden, net zolang tot er commentaar volgt of het eind van de sourceregel is bereikt.

Na de mnemonics behandelen we nu de diverse directives.
END, ###, ELSE, ENDIF, MEND, LST, UNL en remark worden in de sourceregels net zo opgeslagen als een 1-byte mnemonic (alleen kan er voor een remark geen naam staan en voor ### staat altijd een identifier).

ORG, EQU, SET, DW, DS en IF worden analoog aan 3-byte mnemonics opgeslagen, alleen moet er voor EQU en SET een label staan en gebruikt IF naast operators ook comparators en alternators.

Voor MACRO moet altijd een identifier staan, maar hij hoeft niet door operands te worden gevolgd. In dat geval is bit 6 van het statusbyte niet gezet. Wordt MACRO wel door operands gevolgd, dan is bit 6 van het statusbyte gezet.
De operand van MACRO kan alleen uit een of meerdere labels bestaan, gescheiden door Komma's, die op dezelfde manier als operators, comparators en alternators zijn opgeslagen. Dit om compatible te blijven met 3-byte mnemonics in verband met het listen.

Call-macro wordt als volgt opgeslagen:

- Statusbyte
- (Eventeel) pointer naar label
- Call-macro instructiebyte
- Pointer naar identifier
- Eventueel gevolgd door een Komma-seperator en een of meerdere operands, ieder opgeslagen zoals de operands van 3-byte mnemonics maar onderling gescheiden door Komma-separators.
- Eventueel gevolgd door commentaar.

Control wordt als volgt opgeslagen:

- Statusbyte
- (Eventueel) pointer naar label
- Control instructiebyte
- Pointer naar identifier, gevolgd door een Komma-seperator
- Operand, opgebouwd zoals een operand bij IF
- Eventueel gevolgd door commentaar.

Bij DB wordt het instructiebyte gevolgd door een lengtebyte dat aangeeft hoeveel bytes DB tijdens het assembleren in het geheugen moet plaatsen, gevolgd door deze bytes. Of deze bytes als nummerieke waarden of als een ASCII-string moeten worden weergegeven hangt af van bit 6 van het statusbyte, analoog aan dat bij een 2-byte mnemonic. Bij een ASCII-string bepaalt bit 7 van het eerste karakter of de string tussen ' of tussen " geplaatst moet worden. Overigens zijn in een ASCII-string de tekenbits van alle karakters gelijk. Labels of samengestelde operands zijn bij DB ook mogelijk, analoog aan 2-byte mnemonics. Er kan dan echter slechts 1 byte gedefinieerd worden.

TITL wordt analoog aan DB opgeslagen, alleen mag de operand alleen uit een ASCII-string tussen ' bestaan (dus met bit 7 = 0).

PRT wordt analoog aan DB opgeslagen, alleen zijn labels en samengestelde operands niet toegestaan.

PUT wordt analoog aan een 2-byte mnemonic opgeslagen, alleen zijn als karakters slechts B,D,H,O,b,d,h en o toegestaan, allen tussen 2 " (tekenbit gezet). Als waarde voor een byte-operand is alleen een waarde kleiner of gelijk aan 7FH, doch groter of gelijk aan 00H, toegestaan.

Het einde van de sourcecode wordt weergegeven door een 00H byte op de plaats waar anders een statusbyte zou staan.

De instructie-codes

De mnemonics worden weergegeven door de standaard INTEL 8080 codes. Het is echter niet mogelijk de MOV R,R (b.v MOV A,A), de IN en de OUT mnemonics te gebruiken, omdat hun codes voor directives worden gebruikt.
Bij de directives horen de volgende instructiecodes:

08H : ORG	40H : IF	0CBH : END
10H : EQU	49H : ELSE	0D3H : PRT
18H : DW	52H : ENDIF	0D9H : LST
20H : ###	5BH : call-macro	0DBH : PUT
28H : DS	64H : MACRO	0DDH : TITL
30H : control	6DH : MEND	0EDH : UNL
38H : DB	7FH : SET	0FDH : remark

Symboltable

De symboltable wordt vanaf een (per assembler) vast adres naar beneden opgebouwd (dit adres is altijd 0XX00H). Per ingevoerde naam zijn er 8 bytes beschikbaar. De eerste 6 bytes van dit veld bevatten de naam van het ingevoerde label of identifier. Hiervoor worden per byte alleen de laagste 7 bits gebruikt, het tekenbit kan als vlag worden gebruikt, maar moet tijdens de meeste routines 0 te zijn. Het eerste karakter van de naam staat op het laagste adres van deze zes bytes, de tweede op de volgende enz. Is de naam korter dan 6 karakters, dan worden de opengebleven bytes gevuld met 00H-bytes.
De andere 2 bytes van het veld dienen als opslagplaats voor de waarde die er aan een label of identifier kan worden toegekend. De 6 en 2 bytes worden ook wel aangeduid met naamveld, respectievelijk waardeveld.
Als er in de sourcecode naar een naam in de symboltable wordt verwezen, dan gebeurt dit door middel van een 2-byte pointer in de volgorde lowbyte highbyte. Deze pointer wijst naar het laagste geheugenadres van het waardeveld.
Het "dollarlabel" ($), dat de actuele waarde van de objectpointer tijdens de assemblage aangeeft, is niet in de symboltable opgenomen, maar heeft een vaste plaats in de assembler zelf. Dit is dan ook dè reden dat u na het inlezen van een sourcefile van een andere SPL-versie de verwijzingen naar het dollarlabel door het $-commando moet aanpassen.

Tips voor het herstellen van aangetaste sourcefiles

Bij het verlaten van SPL (door het U- of u-commando) worden bepaalde checksums opgesteld, die na terugkeer in SPL automatisch worden gecontroleerd. Blijkt er dan een fout in de assembler te zitten, dan kunt u het beste zowel de assembler als de laatste backup sourcefile van het programma opnieuw inlezen.
Is de symboltable aangetast, dan betekent dit meestal dat een van de namen is verminkt. Een symbollisting zal dit spoedig aantonen, vooral indien de symboltable nog niet zo lang geleden is gesorteerd. Zo'n fout kan in Utility hersteld worden door door middel van de D-opdracht de naam op te zoeken en deze door middel valn de S-opdracht te herstellen. Ziet u niets bijzonders in de symbollisting, dan is de fout in een van de waardevelden opgetreden. Deze fout kan geen kwaad en wordt vanzelf hersteld.
Blijkt dat de sourcecode is aangetast, dan kan dit vaak door een listing worden opgespoord. Soms zal de fout duidelijk te zien zijn, indien bijvoorbeeld het statusbyte van een regel is aangetast (waardoor ook vaak de listing van de volgende regels is verminkt), een symboltable pointer is gewijzigd (waardoor er een andere naam wordt afgedrukt of, wat waarschijnlijker is, een reeks zinloze tekens, of een instructiecode is veranderd in een instructiecode met andere operandvereisten (b.v. een 1-byte mnemonic door een 3-byte mnemonic). In zulke gevallen doet u er verstandig aan het laatste label (of identifier) in de label- of operandkolom op te zoeken en met behulp van het I-, respectievelijk G-, commando het adres op te vragen waar het statusbyte van die regel staat. Door daarna in Utility met behulp van de D-opdracht het gebied met de fout te tonen kan de fout vaak gelocaliseerd worden, waarna deze meestal met behulp van de S-opdracht hersteld kan worden. Levert dit te veel problemen op, dan kunt u ook het statusbyte van de aangetaste regel vervangen door een byte waarvan bit 6 en bit 7 nul zijn en de andere 6 bits het aantal bytes tot het volgende statusbyte aangeven, gevolgd door een 00H byte (NOP) en zoveel 20H (commentaarspatie) bytes als nodig is om het volgende statusbyte te bereiken. Het is ook mogelijk om in plaats van het 00H byte een 38H (DB) byte te gebruiken. Dit moet dan wel door een extra lengtebyte gevolgd worden dat aangeeft hoeveel databytes er nog volgen tot het volgende statusbyte. Na de sourceregel op deze manier provisorisch hersteld te hebben kunt u terug gaan naar SPL om daar met de editor de source verder te corrigeren. Blijkt in eerste instantie niet duidelijk uit de listing wat er fout is gegaan, dan kan de fout soms nog door het ?-commando opgespoord worden. Het is hierbij dan meestal niet nodig het in Utility te herstellen, vaak kan dit met de editor gebeuren. Gebruik de editor echter alleen als u zeker weet dat de te editeren regels normaal gelist worden. Wees ook voorzichtig met objectroutines, in ieder geval als labelpointers zijn aangetast. Levert ook het ?-commando niets op, dan kunt u vaak beter opnieuw de backup sourcefile inlezen.
Na het herstellen van de sourcefile dient de fout in het programma die het ongeluk veroorzaakte nog te worden opgelost.
Hiervoor geven we enkele vuistregels:

1) Stackniveau: dit openbaart zich vaak door een sprong naar een adres buiten het programma, en doet vaak vreemde dingen met de cursor (hollen of stilstaan).
   Meestal komt dit door een onevenwichtig gebruik van PUSH en POP mnemonics, of bijvoorbeeld een JMP waar een CALL had moeten staan. Dit komt vaak voor als u midden in of uit een subroutine springt. Vooral voor nog niet zo ervaren assemblerprogrammeurs verdient het aanbeveling het wat minder ingewikkeld te doen en soms wat meer bytes en/of machinetijd te gebruiken. Als het programma eenmaal goed draait kunt u het altijd nog optimaliseren.

2) Conditionele sprongen: Wordt er op de juiste voorwaarde gesprongen? Vooral als u "test-subroutines" gebruikt, die een vlag (bijvoorbeeld Z of NC) als resultaat geven, kunt u zich hiermee vergissen. Dit soort fouten kenmerkt zich door iets niet, of juist te veel, te doen. "Grafische effecten" hebben dit vaak als oorzaak, bijvoorbeeld een lus met een STAX-mnemonic erin. Als zoiets op het beelscherm verschijnt, dan zal voor het effect het videoram heeft bereikt vaak eerst SPL zijn aangetast.

Bij de voorbereiding van conditionele sprongen kunt u als ezelsbruggetje gebruiken dat een CPI (of CMP) op de vlaggen dezelfde invloed heeft als een SUI (of SUB) echter zonder het A-register te beinvloeden. Let er in het algemeen op hoe de mnemonics de vlaggen beinvloeden. Zo hebben INX en DCX geen invloed op de vlaggen terwijl DCR en INR de carryvlag ongemoeid laten.

3) Registerfouten: Soms zijn dit eenvoudige fouten, zoals een MOV A,B waar een MOV B,A moest staan, soms is het een geval van overbelasting wanneer u bijvoorbeeld in een subroutine C als teller gebruikt en daarbij bent vergeten dat BC in een hoofdroutine als (niet synchroon lopende) pointer wordt gebruikt. Een PUSH-POP combinatie kan hier vaak uitkomst bieden.
Het kan ook gebeuren dat bijvoorbeeld een subroutine een bepaald gegeven in E verwacht, terwijl de hoofdroutine het in L heeft geplaatst. Hier kan een omweg, die voor de aanpassing zorgt, uitkomst bieden, indien het niet praktischer is de hoofd- of subroutine aan te passen of een nieuwe subroutine te schrijven.

4) Diverse fouten: dit is bijvoorbeeld het niet verhogen van een teller, of een het aanroepen van een verkeerde subroutine.

SPL-handwoordenboek

In deze handleiding zijn hier en daar vaktermen gebruikt.
Het is de bedoeling dat hieronder enkele van de meest gebruikte zullen worden verklaard.

In syntaxbeschrijvingen van commando's, directives etc.
[en] : wat tussen [en] staat is optioneel.
( )... : wat tussen (en) staat mag herhaald worden.
onderstreept : hier is een standaardwoord onderstreept.
Dit kan vervangen worden door iedere reeks karakters die aan de de syntaxeisen van het standaardwoord voldoet. In de syntaxbeschrijvingen worden de standaardwoorden ter onderscheiding gevolgd door 0,1,2 of N.
Standaardwoorden:

adres: hexadecimaal getal van maximaal 16D bits.
byte: hexadecimaal getal van maximaal 8D bits
char: Karakter waarvan de ASCII-waarde tussen de 20H en de 7FH (inclusief) ligt.
dcr: DCR-commamdo, bv LOOK of REW3.
getal: nummerieke waarde.
ident: identifiernaam: kleine letter gevolgd door maximaal 5 karakters die geem separators zijn (zie het desbetreffende hoofdstuk).
label: labelnaam: hoofdletter gevolgd door maximaal 5 karakters die geen separators zijn.
regel: regelaanduiding, bestaande uit decimaal regelnummer of label of identifier (zie respectievelijk label en ident).
string: reeks van karakters waarvan de ASCII-waarden tussen de 20H en de 7FH (inclusief) liggen.
vlag: Combinatie van getallen, labels, operators, comparators en/of alternators die na evaluatie een logische conditie 'waar' of 'onwaar' opleveren.
waarde: Combinatie van labels, operators en/of getallen, resulterend in een 16-bits waarde.
Bv: 0FFFFH#0C0C0H of 127D%10D.

Elders

Achtergrond : (geheugen) cassetterecorder, DCR of disk.
Adreskolom : kolom in listingen met de actuele waarde van de objectpointer.
Adreswissel : hierdoor wordt bepaald of de 16-bits waarden in de operands binair, octaal, decimaal of hexadecimaal gelist worden.
Assemblage : het omzetten van de sourcefile in de objectcode.
Assemblerregel : deel van het assemblerprogramma, zoals dit door de editor in 1 regel gelist wordt.
Bytekolom : kolom in listingen die maximaal 3 bytes toont welke tijdens de assemblage door een sourceregel geproduceerd worden.
Bytewissel : hierdoor wordt bepaald of de 8-bits waarden in de operands binair, octaal, decimaal of hexadecimaal gelist worden.
Bevestiging : De C, A, K, R en Y-commando's vragen voor alle zekerheid om het indrukken van [+].
Commando : opdracht voor SPL, in te typen na de prompt.
Commentaarkolom : zie de editorbeschrijving.
Digit : de cijfers en letters waaruit een getal is opgebouwd (bv: de hexadecimale digits zijn de cijfers 0-9 en de letters A-F).
Directive : in een assemblerregel vervatte opdracht aan de assembler om iets speciaals te gaan doen.
Editbuffer : ruimte in de sourcebuffer waar de editor tijdens het editeren de tekst bewaard.
Fase : bepaald stadium tijdens objectroutines. Zie het A-commando.
Hardcopyvlag : hierdoor wordt bepaald of er tijdens listingen ook karakters naar de printer gestuurd worden.

Identifier : symbolische naam, gebruikt om een bepaald adres in de source-file aan te geven (voor ### of MACRO).
Implementabel : door de gebruiker aan te passen d.m.v. de IMPLEM sourcefile.
InstructieKolom: zie de editorbeschrijving.
Kopregel : regel zoals die tijdens hardcopy listingen bovenaan de pagina's worden afgedrukt. Bij SPL V1.1 bestaat deze regel uit 'SPL V1.1 PAGE', paginanummer en (eventueel) een titel.
Label : symbolische naam om een waarde in het assemblerprogramma aan te geven.
Labelkolom : zie de editorbeschrijving.
Macro : reeks instructies in het assemblerprogramma, beginnende met MACRO, eindigend met MEND en aangeduid door een identifier. Kan op iedere plaats in het objectprogramma worden ingevuld door middel van een call-macro.
Mnemonic : symbolische weergave van een voor de microprocessor begrijpelijke instructiecode.
Offset : afstand waarover iets verplaatst wordt in vergelijking met zijn normale plaats.
Opstartadres : het adres waarmee SPL wordt aangeroepen door (utility) G of (BASIC) CALLM. In SPL V1.1 standaard 8500H.
Objectcode : de bytes van het machinetaalprogramma, zoals dit door de assembler uit de sourcecode worden opgebouwd.
Objectlisting : uitgebreide listing, waarbij wordt getoond hoe de objectcode tijdens de assemblage in het geheugen komt.
Objectpointer : een teller die tijdens objectroutines bijhoudt vanaf welk adres de door een sourceregel geproduceerde objectcode tijdens assemblage in het geheugen geplaatst zou worden.
Objectroutine : routine waarbij assemblagefasen worden uitgevoerd. Dit zijn bij SPL de volgende routines: A, F, I(adres), S, T en ?.
OperandKolom : zie de editorbeschrijving.
Paginateller : teller van het aantal pagina's tijdens hardcopy listingen ten behoeve van het paginanummer in de kopregel.
Regelnummer : decimaal getal maarmee een assemblerregel kan worden aangeduidt. SPL nummert de regels 1,2,3,...
Regelteller : teller van het aantal regels tijdens hardcopy listingen ten behoeve van het genereren van formfeeds.
Routine : een deel van het SPL-programma dat door een bepaald commando wordt aangeroepen.
Sourcebuffer : deel van het geheugen waar de sourcefile en de editbuffer worden opgeslagen.
Sourcecode : de gecodeerde vorm van het assemblerprogramma, zoals dit door SPL in het geheugen wordt opgeslagen.
Sourcefile : combinatie van sourcecode en symboltable.
Sourceregel : de gecodeerde vorm van een assemblerregel.
Startadres : beginadres van de sourcebuffer.
Tab : tabulatiekarakter.
Topadres : hoogste adres van de symboltable.
Uitvoerwissel : adres- of bytewissel.
Waardevelden : twee bytes achter een naam in de symboltable waar de waarde van het label, of het adres van de identifier, wordt opgeslagen.
Wegschrijfadres :beginadres van de gecombineerde sourcecode en symmboltable bij het wegschrijven naar een achtergrondgeheugen.
[ ] : spatiebalk.
[+] : + toets.
[<-] : cursor-left toets.
[break] : breaktoets.
[chardel] : character-delete toets.
[return] : returntoets.
[tab]: tabulatietoets.

DISPLAY.
========

DISPLAY is een hulpprogramma van de conditionele macro-assembler SPL (verder aangeduid met "SPL"). De naam DISPLAY is afgeleid van DISassembler en SPL, met -AY voor de uitspraak. Voor het begrijpen van de werking van DISPLAY verdient het sterke aanbeveling eerst de hoofdstukken over de assembler grondig door te nemen, daar u zonder deze kennis voor problemen zult komen te staan.
De ontwerpers van DISPLAY stellen zich bij voorbaat niet verantwoordelijk voor enig gevolg van het gebruik van dit programma, waarbij zij dan vooral denken aan eventuele copyrights van gedisassembleerde programma's e.d. Zij realiseren zich welliswaar, dat eventuele 'krakers' met dit programma, in combinatie met SPL, een zeer krachtig inbraakwerktuig in handen krijgen, maar beschouwen een eventueel misbruik als niet onder hun verantwoordelijkheid vallend. Wel doen zij bij deze een klemmmend beroep op de u om dit programma uitsluitend voor legale doeleinden te gebruiken.
DISPLAY is speciaal ontworpen voor het decoderen (disassembleren) van machinetaal-programma's. Door de assemblage van DISPLAY wordt de commandoset van SPL met enige opdrachten uitgebreid. Tevens kan SPL in de zgn.'koude start' toestand worden gebracht (door TOFILE op TRUE te SETten), zodat u naast de machinetaal versie van DISPLAY een bijbehorende SPL-versie kan wegschrijven.
Naast de echte disassembler commando's in DISPLAY zijn er een paar hulp-commando's, die kunnen helpen een overzicht van het onder handen zijnde programma te krijgen. Dit zijn:

% [adres1] [adres2]
Hierbij zijn adres1 en adres2 door 16D deelbaar, of worden afgekapt.
Deze opdracht toont de geheugen-inhoud van adres1 tot adres2. Dit geschiedt in blokken van 16 bytes per regel, voorafgegaan door een adres, dat altijd door 16D deelbaar is. De blokken van bytes worden op de regel achtereenvolgens getoond als getallen van 2 hexits, waarbij telkens na elk byte op een oneven adres een spatie volgt en als ASCII-tekens. Karakters met een waarde van minder als 20H worden getoond als een punt ('.'). Desgewenst kunt u hiervoor een ander karakter kiezen (door het aanpassen van label CONTRL).
Dit tonen kunt u, net zoals in andere listings in SPL, naar wens tijdelijk of definitief onderbreken.

+ [adres1]
Deze routine zoekt alle plaatsen op waar een bepaald adres in het geheugen vermeld staat en toont deze. Dit kunt u bijvoorbeeld gebruiken om te zien vanwaar een bepaalde subroutine wordt aangeroepen, waar een bepaalde tabel wordt gebruikt etc.

¥ [adres]
Deze opdracht doet hetzelfde als de bovengenoemde, maar heeft als extra voorwaarde, dat het betreffende adres in het geheugen dient te worden voorafgegaan door een machinecode, die als operand een adres gebruikt. Deze code wordt dan tesamen met het adres getoond. Tevens wordt de instructie in SPL-mnemonics getoond.

Het belangrijkste onderdeel van DISPLAY is een verzameling disassembler-opdrachten. Deze wijken echter in belangrijke mate af van de vroegere DAInamic disassembler. Ze genereren namelijk geen direct zichtbare output, maar inplaats daarvan maken ze een SPL-sourcefile aan, die met behulp van de standaard SPL commando's kan worden bewerkt. Dit heeft twee in het oog springende voordelen:
1) Doordat DISPLAY niet op het langzame menselijke oog of op printers hoeft te wachten, kan hij een hoge snelheid bereiken. U dient er echter rekening mee te houden dat het disassembleren desondanks toch duidelijk langzamer is als het assembleren van eenzelfde programma. Voor grote programma's kan het wel in de minuten lopen. (Langer als 5 minuten zal het waarschijnlijk zelden duren).

2) Doordat u nu al over een source van het programma beschikt, hoeft u deze niet meer apart in te typen. Wanneer u ervaring heeft met het overtypen van programma's uit tijdschriften e.d. zult u wel weten hoeveel tijd er juist in dat intypen kan gaan zitten.

De nieuwsgierigen onder ons kunnen met de DISPLAY-SPL combinatie vrij eenvoudig de werking van een bepaald programma nagaan, de source-listing van commentaar voorzien, de labels doeltreffende benamingen geven en al zulke dingen meer, waardoor u de werking van het programma beter zult kunnen doorgronden. Al dit soort dingen kunnen ook de eigen machinetaal programmeringsvaardigheid bevorderen (hier spreken wij uit eigen ervaring).
DISPLAY is een zogenaamde lijstgestuurde disassembler. Dit houdt in, dat eerst een bepaalde commandolijst wordt opgesteld, waarin vermeld staat of een bepaald geheugengebied als bijvoorbeeld machinecode's, als ASCII, of als nog wat anders vertaald moet worden. Sphynx heeft voor deze constructie gekozen terwille van een van de krachtigste eigenschappen van de disassembler, namelijk het zelf genereren van labels. Nu worden er ook voor tabellen e.d. op de juiste plaatsen labels geplaatst (mits u natuurlijk een goede commandolijst heeft opgesteld). Ondermeer terwille van een hoge verwerkingssnelheid is gekozen voor een gesorteerde opbouw van de symboltable. Hierdoor wordt de zoektijd van in het bijzonder de labelkolom-labels aanmerkelijk bekort. Dit heeft echter wel tot gevolg gehad dat elke disassemblage met een schone symboltable moet beginnen. Hiertoe wordt voor elke disassemblage een koude start gegenereerd. Ook dit was een reden voor de tabulaire opbouw. De commandolijst blijft bewaard, zodat u hem desgewenst opnieuw kan gebruiken als bv. blijkt dat u een bepaald geheugengebied als machinecode heeft gedisassembleerd, terwijl het eigenlijk een ASCII-tabel was. Door de gesorteerde opbouw is een apart Q-commando niet meer nodig, u kunt de ontstane source desgewenst direct wegschrijven (indien er tenminste minimaal 2 labels in de symboltable staan).
Het opstellen van de lijst geschiedt als volgt: u voert een werkcommand in (" # ' : = ~ , ; < > of een zelf gedefinieerd werkcommando), gevolgd door een adres (in hexits). Dit betekent dan: disassembleer, tijdens de uitvoering van de lijst, het gebied vanaf het voorafgaande adres in de lijst (of vanaf 0H, indien het de eerste opdracht in de lijst is), tot aan het meegegeven adres, op de manier zoals deze door de opdracht wordt aangegeven. In de lijst staat in het begin al de opdracht ~ FFFF. De commando's worden automatisch naar opklimmende adreswaarde gerangschikt. Bij het invoeren van een bestaand adres wordt de oude opdracht door de nieuwe vervangen (niet mogelijk voor FFFF). In de standaardversie is deze lijst in de BASIC-ENVELOPE buffer geplaatst. Dit houdt in dat de lijst maximaal 80H bytes (dus 42D opdrachten) lang kan zijn. Bij het opstellen van de commandolijst waakt DISPLAY dan ook tegen het overschrijden van deze grens. U dient er tevens rekening mee te houden, dat SPL zelf deze buffer gebruikt voor de evaluatie van MACRO's. Indien u dus tussentijds een programma met MACRO's assembleert, dan wordt uw lijst overschreven. Indien U een grotere lijst wenst te gebruiken, dient U de grenzen van de lijst (DMPBUF en BUFMAX) aan te passen. Voor het bewerken van de lijst heeft U de beschikking over de volgende lijst-commando's:

! Toon de disassemblagelijst.
  Deze vertoning geschiedt in de vorm: code, [spatie] adres (hexadecimaal).
  De vertoning kunt u desgewenst op de bekende manier tijdelijk of definitief onderbreken.

) Maak de lijst leeg (op ~ FFFF na). Deze opdracht wacht op een bevestiging.

/ [adres]
  Verwijder dit adres uit de lijst (niet mogelijk voor FFFF).

&[adres]

Disassembleer aan de hand van de lijst, waarbij het meegegeven adres een offset aanduidt, zoals die bij het SPL-commando A meegegeven zou zijn om een dergelijke offset te bewerkstelligen. Zo'n offset kunt u bijvoorbeeld gebruiken wanneer het te disassembleren programma zich normaal op adressen bevindt die nu door SPL worden ingenomen.
Er wordt automatisch een koude start gegenereerd, zodat de eventueel al in het geheugen aanwezige SPL-source wordt vernietigd. Bij deze koude start wordt mbv. 'Start?' het beginadres van de te vormen SPL-source gevraagd. Dit startadres dient u zodanig te kiezen, dat het te disassembleren machinetaalprogramma niet wordt overschreven.
Indien tijdens het afwerken van de lijst het gebied tussen E000H t/m EFFFH anders als dmv. '~' moet worden gedisassembleerd, wordt u het nummer van de E-bank gevraagd. Het gelijktijdig disassembleren van meerdere E-banken is niet mogelijk, dit o.m. met het oog op de labels. Indien het eerste commando niet uit '~' bestaat, wordt aan het begin een ORG 0H geplaatst. Aan het eind van de SPL-source wordt automatisch een END geplaatst. Bij het adres van elke '~' wordt een ORG geplaatst, behalve bij de ~ FFFF. Dit alles geschiedt onder het voorbehoud van genoeg geheugenruimte. Bij een eventueel tekort wordt de sourceproduktie voortijdig afgebroken.
Tijdens een disassemblage worden eerst alle als 16-bits operands, bij DW of bij ';' gebruikte getallen in opklimmende volgorde als labels in de symboltable opgeslagen. Deze labels bestaan uit de letters LH, gevolgd door de hexadecimale uitbeelding van de waarde, maar u mag hiervoor ook een andere combinatie kiezen. Tijdens de tweede fase van de disassemblage worden deze labels in de labelkolom van de sourceregel geplaatst, die op het desbetreffende adres begint of, bij het ontbreken hiervan, mbv. een EQU gedefinieerd.

De disassembler kent de volgende werk-opdrachten:

```
# Disassembleer als machinetaalinstructies
,       "       "  nummerieke 8-bit getallen
'       "       "  8-bit ASCII karakters (MSB 0)
"       "       "    "     "       "     (MSB 1)
=       "       "    "     "       "     (MSB 0 of 1)
:       "       "  16-bit nummerieke getallen (low-high)
;       "       "    "       "         "     (high-low)
~ Definieer alleen de labels in dit gebied.
```

Bij ' " en = worden de getallen, die niet aan de desbetreffende MSB-voorwaarde voldoen, en/of waarvan de laagste 7 bits een waarde, kleiner als 20H aangeven, als numerieke 8-bit waarden gedisassembleerd.
Bij de # opdracht worden waarden, die een speciale SPL-code vertegenwoordigen, als 8-bit getal verwerkt.
In de bovenstaande twee speciale gevallen staat er 1 8-bit getal op een regel. Bij de verwerking van ',' komen er gewoonlijk 8 op een regel, tenzij het top-adres eerder bereikt is, of er eerder een label staat. De standaard lengte voor ASCII functies is 16 karakters.
Bij ; wordt, aangezien SPL geen speciale directive voor 16-bits getallen in high-low volgorde kent, een speciale methode toegepast. Stel dat adres 1234 in de machinecode in high-low volgorde staat, dan geeft ';' het volgende:

```
[label0] DB LH1234)8H
[label1] DB LH1234&0FFH
```

In DISPLAY zijn standaard twee commando's voor het decoderen van samengestelde tabellen opgenomen. Dit zijn:
< decodeert byte, adres in low-high volgorde.
> decodeert lengtebyte, ASCIIstring, adres in low-high.
U kunt gemakkelijk eigen samengestelde routine's definieren. Als opdrachtcodes kunt u hiervoor bv '[' en ']' gebruiken. In dit geval moet u niet vergeten het commando op te nemen in DECTBL, ZETTBL en PLCTBL. In deze tabellen zult u verwijzingen naar als voorbeeld te gebruiken routines (< en >) vinden.
Mocht u een of meerder opdachten niet nodig hebben, dan kunt u de betreffende routines desgewenst uit DISPLAY verwijderen, om zo meer ruinte voor machine-taalprogramma en SPL-source te creeren.
Omdat bij de DAIpc RST 1,4 en 5 altijd een databyte meenemen, wordt deze ook standaard als zodanig gedecodeerd. Wanneer u van deze mogelijkheid wilt afzien moet u het label RSTDAT op FALSE SETten.

TRANSLATOR

Translator is een hulpprogramma voor de SPL-assembler, dat bedoeld is om programma's, die m.b.v. de DNA-assembler geschreven zijn, om te zetten in SPL-editorcode, die dan door de SPL-editor weer in SPL-sourcecode wordt vertaald.
Dit vertalen naar editorcode houdt ondermeer in:
Het op 0 zetten van de MSB's van de ASCII-karakters
Het verwijderen van de linefeed na de carriage return
Het op de juiste wijze plaaten van [tab]'s
Het verwijderen van overbodige spaties
Het vervangen van de speciale DNA-mnemonics door standaard mnemonics
Het aanpasen van de notatie van de constanten
Het op de daartoe bestemde plekken plaatsen van puntkomma's

Het werken met TRANSLATOR gaat als volgt:
U begint met TRANSLATOR op de gewenste plaats te assembleren, en desgewenst de machinetaalversie ook op band te zetten. U kunt natuurlijk ook, indien aanwezig, meteen de machinetaalversie inlezen. In ieder geval heeft u ook SPL zelf nodig. Indien u een versie van DNA heeft, die zijn sourcefiles onder een "#" wegschrijft, leest u vervolgens eerst deze assembler in en met behulp daarvan de gewenste sourcefile. Wanneer uw versie onder een "1" wegschreef, kunt u deze sourcefile ook mbv. utility inlezen. Deze sourcefiles mogen SPL en/of TRANSLATOR natuurlijk niet overschrijven. Het begin van de sourcefile dient gelijk te zijn aan de waarde DNABAS in TRANSLATOR (standaard 3000H). Daarna start u TRANSLATOR, op het adres waar u zijn origin op had ingesteld. Na de opstart vraagt TRANSLATOR met "start?" het startadres van de te vormen SPL-buffer. Dit adres dient minstens 200H kleiner te zijn als DNABAS. Hierna is het uiterlijk even rustig, terwijl TRANSLATOR de source vertaalt. Als dit geschied is, ziet u de typische edit-mode op het scherm en dan "zit" u in de editor van SPL. Nu kunt u het beste op [break][ ] drukken, want meestal zal de editor dan zonder foutmeldingen de editbuffer inhoud naar SPL-source vertalen. Mochten er toch problemen zijn (bijvoorbeeld door een fout in de oorspronkelijke DNA-source), dan ziet u die vanzelf wel op het scherm verschijnen.
Tijdens deze vertaalfase kunt u de editor niet met [break][<-] verlaten. Dit is om te voorkomen dat u per ongelijk wel deze uitweg zou kiezen en dan opnieuw zou moeten beginnen.

IMPLEMENTATIE

Het implementeren van veranderingen op SPL gaat als volgt:
U begint met SPL in te lezen en deze daarna in een veilig gebied te copieren m.b.v. de utility M-opdracht. Hierna leest u de sourcefile IMPLEM in, waarna u de gewenste veranderingen kiest. Vervolgens assembleert u IMPLEM met een offset die overeenkomt met de verplaatste versie van SPL. Vervolgens schuift u deze (met veranderingen) weer op de juiste plaats en schrijft hem naar het achtergrondgeheugen.
De veranderingen selecteert u door het bijbehorende "vlag"-label op een bepaalde waarde te zetten. In de default situatie zou het assembleren van IMPLEM niets veranderen, alleen de labels die u een andere waarde geeft hebben effect. Hieronder een lijst van de vlaglabels met hun mogelijke standen, de bedoeling daarvan. Als er "extern" na staat, dient u ook op de plaats waar de vlag gebruikt wordt nog wat te veranderen.

TOFILE	TRUE/FALSE	Het in de koude start toestand brengen van SPL.
SRCTOP	SAME/CHANGD	Het veranderen van de top van de buffer, bijvoorbeeld om SPL en DISPLAY als 1 file te kunnen wegschrijven en inlezen Verander ook TOPADR.
HIDING	SAME/CHANGD	Het al dan niet listen van bepaalde instructies tijdens de objectlist. Er kunnen maximaal 8 instructies onderdrukt worden (geen NOP).Extern.
DFWORD	SAME/CHANGD	De default listvorm van 16-bits constanten. Extern.
DFBYT	SAME/CHANGD	De default listvorm van 8-bits constanten. Extern.
FILECH	SAME/CHANGD	Het lees/schrijf karakter. Extern.
RGDEFL	SAME/CHANGD	Default aantal regels op een blad (zonder hoofd). Extern.
BASGEB	KILLED/SAVED	Bewaren of vernietigen BASIC-programma.
BASPRG	SAME/CHANGD	Begin BASIC (extern).
SOUND	SAME/CHANGD	De signaaltoon. Code analoog aan TALK. Extern.
COLORS	SAME/CHANGD	De cursorkleuren. Extern.
WIDTH	getal	De karakterbreedte van de printer.
TABELS	SAME/CHANGD	De tabulatietabellen. Extern.
DCR	PRSENT/NPRSNT	Al dan niet een DCR aanwezig.
PRINTR	SERIAL/PARALL/NOPRIN	Type printer (NOPRIN: geen printer).Extern.
BDRATE	getal	Byte voor baudrateregister 5501

RESTORE XXXX

RESTORE XXXX is niet alleen een leuk voorbeeld om de werking van diverse SPL-comando's duidelijk te maken, het heeft ook een ander doel. Als u deze source-file assembleert vormt het een leuke aanwinst voor uw BASIC-programma's.

Een van de (weinige) nadelen van de DAI-BASIC is dat het niet mogelijk is door middel van RESTORE de DATA-pointer op een ander regelnummer dan de eerste regel met DATA te zetten. Bij veel andere BASIC's is dit wel mogelijk door middel van het statement RESTORE XXXX, waarbij XXXX het regelnummer voorstelt van de regel waar u de volgende DATA met een READ-statement uit wilt lezen. Voor dit probleem vormt RESTORE XXXX de oplossing. Hiervoor moet u het volgende doen:

-Assembleer RESTORE XXXX of lees een eerder weggeschreven objectfile van RESTORE XXXX in.
-Zet de HEAP-pointer (adressen 29BH en 29CH) op 327H, dus voorbij RESTORE XXXX.
-Voer in BASIC een NEW of een CLEAR XXXX uit, zodat ook de andere BASIC-pointers worden aangepast.
-Typ of lees uw BASIC-programma in.
-Plaats in uw BASIC-programma's, overal waar u RESTORE XXXX wilt gebruiken, de combinatie P%=XXXX:CALLM 750,P% of statements met een gelijk effect. Hierbij stelt P% een willekeurige integer variabele voor en XXXX het regelnummer waar de DATA-pointer naar moet komen te wijzen.
-RUN uw programma.

RESTORE XXXX plaatst de DATA-pointer op het regelnummer wat in de integer variabele staat. Komt dit regelnummer niet in het BASIC-programma voor, dan volgt de (BASIC) foutmelding "UNDEFINED LINE NUMBER", tijdens een RUN gevolgd door "IN LINE NUMBER NNNN", waarbij NNNN het nummer is van de regel met het CALLM-statement.
De regel waar u de DATA-pointer naar laat verwijzen hoeft zelf geen DATA te bevatten. Ontbreekt de DATA, dan zal het eerstvolgende READ-statement de eerste regel (na de regel waar u de DATA-pointer naar heeft laten verwijzen) opzoeken die wel DATA bevat. Wordt zo'n regel niet gevonden, dan volgt de gebruikelijke foutmelding "OUT OF DATA", tijdens een RUN gevolgd door "IN LINE NUMBER NNNN", waarbij NNNN het nummer van de regel met het READ-statement voorstelt.

De RESTORE XXXX sourcefile is een beetje omslachtig opgesteld, om de werking van diverse SPL-directives duidelijk te maken. Hij kan vereenvoudigd worden, maar dat laten wij graag aan uw eigen inzicht over.

Inhoudsopgave documentatie SPL V1.1
==========

Pag.	hoofdstuk
1	Voorwoord
2	Opstarten
3	Invoer
4	SPL-commando's
13	Alfabetische commandolijst
14	De editor
17	Mnemonics
17	Assemblerdirectives
21	Operators
21	Comparators
21	Alternators
22	Separators
22	Speciale karakters in de operand-kolom
23	Opbouw van een assemblerprogramma
25	Syntaxfouten
27	Structuurfouten
28	Foutboodschappen
29	Invoerfouten
30	Steken ophalen voor gevorderden
35	SPL-handwoordenboek
37	DISPLAY
41	TRANSLATOR
42	IMPLEMENTATIE
43	RESTORE XXXX
44	Inhoudsopgave documentatie SPL V1.1

Zilog Incorporated
10460 Bubb Road
Cupertino
California 95014
USA

AMD
901 Thompson Place
PO Box 453
Sunnyvale
California 94086
USA

Intel Corporation
3065 Bowers Avenue
Santa Clara
California 95051
USA

Motorola Incorporated
2553 North Edgington Street
Franklin Park
Illinois 60151

SPL documentatie V1.1

INHOUDSOPGAVE

1	voorwoord
2	opstarten
3	invoer
4	SPL-commando's
13	alfabetische commandolijst
14	de editor
17	Mnemonics
17	Assemblerdirectives
21	Operators
21	Comparators
21	Alternators
22	Separators
22	Speciale karakters in de operand-kolom
23	Opbouw van een assembler-programma
25	Syntaxtfouten
27	Structuurfouten
28	Foutboodschappen
29	Invoerfouten
30	Steken ophalen voor gevorderden
35	SPL-handwoordenboek
37	Display
41	Translator
42	Implementatie
43	Restore XXXX

***