

# Elektronik

## MIKRODATORTEKNIK

### Häfte 12: TSC 6800 Mnemonic Assembler

---

#### INNEHÅLL

|                                | sid   |
|--------------------------------|-------|
| 12.1 Allmänt                   | 12-1  |
| 12.2 Beskrivning av assemblern |       |
| 12.2.1 Tecken                  | 12-3  |
| 12.2.2 Satser                  | 12-4  |
| 12.2.3 Adresseringsmoder       | 12-7  |
| 12.3 Assemblering              | 12-8  |
| 12.4 Assembler-direktiv        | 12-11 |

## 12 TSC 6800 MNEMONIC ASSEMBLER

## 12.1 Allmänt

Antag, att vi vill att en mikrodator skall addera två tal, 8 och 6. Detta klarar datorn om den har tillgång till ett program som är skrivet på ett språk som datorn förstår, dvs på *maskin-kod* (*objekt-kod*). För mikroprocessorn 6800 skulle programmet då kunna se ut på följande sätt:

```
10110110
00000001
00110000
10111011
00000001
00110001
10110111
00000001
00110010
```

Det finns flera uppenbara nackdelar med att skriva ett program på maskinspråk, dvs i binär form.

- Programmet är svårt att läsa och svårt att förstå
- Det är tidsödande att skriva programmet
- Det är tidsödande att ladda programmet i RAM, eftersom laddningen måste ske med hjälp av en switch-panel (en switch för varje bit = 8 switchar)
- Det är mycket lätt att göra fel men mycket svårt att hitta igen felet, dvs det är svårt att "avlusa" programmet (eng. debug = avlusa).

Betydligt enklare blir det om vi kan skriva programmet med hexadecimala tal och även knappa in programmet via ett hexadecimalt tangentbord. Programmet får då följande utseende:

```
B6
01
30
BB
01
31
B7
01
32
```

Eftersom mikroprocessorn endast förstår maskinkod, måste vi nu ha tillgång till ett hjälp-program, som vid inladdningen av programmet i RAM gör en översättning från hexadecimal kod till binärkod. Detta hjälp-program kan ligga i ett ROM som ett *monitor-program*.

Även ett program skrivet i hexadecimal form är något svårläst bl a på grund av att instruktionskoder, data och adresser ser likadana ut. Programmet blir därför förståeligt endast om det förses med en stor mängd förklaringar och kommentarer.

Det blir påtagligt enklare att skriva och rätta program om man kan tilldela varje instruktionskod (operationskod) ett namn, dvs en s k *mnemonic* (memo-kod). För mikroprocessor 6800 som har 72 olika instruktioner, finns det därför en uppsättning på 72 stycken olika mnemonics. Dessa mnemonics används vid s k *assembler-programmering*. Vid assembler-programmering måste man följa vissa regler och man skall använda vissa tecken med speciell betydelse. Mera om detta senare. Vårt additionsprogram får nu följande utseende:

```
LDAA $0130  (Ladda ackumulator A med innehållet i cell 0130. Där ligger 8.)
ADDA $0131  (Addera ackumulator A med innehållet i cell 0131, dvs med 6.)
STAA $0132  (Lagra resultatet i cell 0132.)
```

För att datorn skall förstå detta program, måste en översättning göras. Denna översättning, *assemblering*, kan vi givetvis göra för hand, men betydligt säkrare och snabbare går det om vi överlåter detta arbete till ett särskilt översättningsprogram, en *assembler*. Assemblern är alltså ett hjälpprogram, som översätter symbolspråk till maskinkod. Fördelarna med assemblerprogrammering är bl a:

- Man kan tilldela vissa minnesceller, grupper av minnesceller, in- och utenheter mm särskilda namn, vilket gör programmet relativt lättläst.
- Assemblern kan översätta tal från olika talsystem (decimala, oktala och hexadecimala tal) liksom ASCII-koder.
- Assemblern kan utföra vissa aritmetiska operationer
- Programmeraren kan välja den form på vilket programmet skall redovisas.

Det finns förstås även vissa nackdelar med assemblerprogrammering, t ex:

- Assemblerprogrammet och den kringutrustning som behövs kostar en hel del pengar.
- Programmeraren måste fortfarande ha en fullständig kännedom om mikroprocessorns uppbyggnad, instruktionsuppsättning mm.
- Programmeraren måste själv bryta ned ett visst problem i en serie instruktioner, varför programmeringen tar relativt lång tid.
- Varje mikroprocessorfamilj har sitt assemblerspråk. Ett assembler-program för Motorolas 6800 kan därför inte direkt användas för t ex Intels 8080.

En del av de problem som hör samman med assemblerprogrammering försvinner om man använder ett *hög-nivå-språk* (typ Basic, Fortran, Pascal). Dessa språk är problem-orienterade i stället för dator-orienterade. Varje *sats* (*statement*) i ett hög-nivå-språk motsvarar i regel ett flertal assembler-instruktioner varför programmeringsarbetet går snabbare. I ett hög-nivå-språk kan vårt additionsprogram få följande utseende:

```
SUM:=(TAL1+TAL2)
```

För att göra en översättning från ett högnivå-språk till maskinspråk, behöver man ett översättningsprogram som kallas *kompilator*. En kompilator är dyr och kräver större minnesutrymme än en assembler, vilket också kostar pengar.

Idag utvecklas mikrodatorprogram i regel med hjälp av s k utvecklings-system. I ett utvecklingssystem finns ett flertal hjälpprogram och speciell utrustning som utnyttjas t ex när man skriver sitt program (Editor), när man vill översätta sitt program till maskinkod (Assembler, Kompilator), när man söker fel i programmet (Debugger) och när man provkör sin konstruktion (Emulator). I detta häfte beskrivs den assembler som finns i utvecklingssystemet Ericsson USE 6800/6809.

## 12.2 Beskrivning av assemblern

### 12.2.1 Tecken

De tecken, karaktärer, som får användas i assemblern är en stor del av de tecken som ingår i ASCII-koden (se figur)

| BITS 4 thru 6 | — | 0   | 1   | 2  | 3 | 4 | 5 | 6 | 7   |
|---------------|---|-----|-----|----|---|---|---|---|-----|
| BITS 0 thru 3 | 0 | NUL | DLE | SP | 0 | @ | P |   | p   |
|               | 1 | SOH | DC1 | !  | 1 | A | Q | a | q   |
|               | 2 | STX | DC2 | "  | 2 | B | R | b | r   |
|               | 3 | ETX | DC3 | #  | 3 | C | S | c | s   |
|               | 4 | EOT | DC4 | \$ | 4 | D | T | d | t   |
|               | 5 | ENQ | NAK | %  | 5 | E | U | e | u   |
|               | 6 | ACK | SYN | &  | 6 | F | V | f | v   |
|               | 7 | BEL | ETB | '  | 7 | G | W | g | w   |
|               | 8 | BS  | CAN | (  | 8 | H | X | h | x   |
|               | 9 | HT  | EM  | )  | 9 | I | Y | i | y   |
|               | A | LF  | SUB | *  | : | J | Z | j | z   |
|               | B | VT  | ESC | +  | ; | K | [ | k | {   |
|               | C | FF  | FS  | ,  | < | L | / | l | /   |
|               | D | CR  | GS  | -  | = | M | ] | m | }   |
|               | E | SO  | RS  | .  | > | N | < | n | ≈   |
|               | F | SI  | US  | /  | ? | O | — | o | DEL |

Figur 1. ASCII-kod (American Standard Code for Information Interchange)

Bl a får alltså följande tecken användas:

- 1 Alfabetets bokstäver A → Z, a → z
- 2 Siffrorna 0 → 9
- 3 De fyra aritmetiska operatorerna +, -, \* och /.
- 4 Karaktärer som används som prefix eller postfix: (exempel senare)

|                  |                | Prefix | Postfix     |
|------------------|----------------|--------|-------------|
| Decimala tal     | (0 → 9)        | inget  | inget       |
| Oktala tal       | (0 → 7)        | @      | 0 eller Q   |
| Binära tal       | (0, 1)         | %      | B           |
| Hexadecimala tal | (0 → 9, A → F) | \$     | H           |
| ASCII-tecken     | (\$20 → \$7F)  | '      | ej tillåtet |
| Immediate mode   |                | #      | ej tillåtet |

- 5 Några ofta använda karaktärer:

SP SPACE = mellanslag (separerande karaktär)  
 , komma (separerande karaktär)  
 CR Carriage Return = vagnretur  
 LF Line Feed = radbyte

## 12.2.2 Satser

Ett program består av ett antal *satser* (*statements*). Varje sats upptar en rad i programmet. När man laddar ett program, skall slutet av en sats markeras med en CR-karaktär (\$ØD). Använder man tangentbord, sker detta genom att man trycker på CR-tangenten. Samtidigt byter man då rad.

En sats består av fyra fält:

A) Labelfält      B) Operatorfält      C) Operandfält      D) Kommentarfält

Fälten skiljs åt genom ett (eller flera) mellanslag (SP-karaktärer)

### A Labelfält

En label används istället för en absolut minnesadress. Följande regler gäller:

- 1 Alla labels i ett program måste vara unika.
- 2 En label måste börja i första karaktärpositionen
- 3 En label kan bestå av maximalt 6 st alfanumeriska karaktärer (A → Z, (a → z, Ø → 9). Försöker man använda fler än 6 st karaktärer, ignoreras dessa.
- 4 Alla labels måste börja med en bokstav (A → Z, a → z)
- 5 En sats kan bestå av enbart en label

Observera speciellt, att en label måste börja i första karaktärpositionen. Detta innebär, att om linjenummer används (t ex 1.00=) skall labeln börja omedelbart efter likhetstecknet (t ex 1.00=START LDAA \$Ø100). Om man har ett eller flera mellanslag efter likhetstecknet, anger detta att raden inte har någon label.

Lägg också märke till, att en label får bestå av högst 6 st karaktärer. Om man i ett program har tänkt sig att EXEMPEL 1 och EXEMPEL 2 skall vara två olika labels, blir det fel eftersom assemblern uppfattar båda som EXEMPE. Assemblern ser alltså två identiska labels och ger därför ett felmeddelande. Dessutom finns det ett mellanslag framför 1:an resp. 2:an varför dessa siffror hamnar i operatorfältet (Se avnitt B nedan)

Stora och små bokstäver uppfattas som helt olika karaktärer. Detta innebär att 'START' och 'start' är två olika labels. Normalt används endast stora bokstäver.

En label får utgöras av en enda bokstav, även om detta är mindre lämpligt. Bokstäverna A och B t ex, bör reserveras helt för att representera ackumulator A resp. B (se sid. 12-7, punkt 1). Bokstaven X ensam, får överhuvudtaget inte användas som en label. STAA X t ex, kommer att uppfattas som STAA 0,X, dvs indexerad adressering (se sid 12-8, punkt 5).

### B Operatorfält

I operatorfältet står en mnemonic, som antingen är en instruktion (t ex INX) eller ett assemblerdirektiv (t ex ORG, se nedan).

En operator består alltså av tre bokstäver (A → Z, a → z), som måste följas av ett mellanslag. Mnemonics som berör ackumulatorerna A och B som exempelvis LDA A och AND B, får dock även skrivas LDAA och ANDB. I sådana fall måste den fjärde karaktärpositionen följas av ett mellanslag.

### C Operandfältet

Vad som står i operandfältet för en viss rad beror bl a på vilken mnemonic-operator, adresseringsmode osv man använder. Följande huvudregler gäller:

- 1 Operandfältet kan bestå av en siffra, en symbol eller ett uttryck eller ett adresseringsmode-tecken tillsammans med en siffra, en symbol eller ett uttryck. (Se avnitt 12.2.3, sid 12-7)
- 2 Adresseringsmode-tecknet är antingen tecknet # (pound sign) som anger Immediate mode adressering eller ,X som anger indexerad adressering (Se avsnitt 12.2.3, sid 12-7 och 12-8)
- 3 Operand kan saknas, t ex vid instruktioner av typen NOP, RTI osv.

Assemblern godtar alltså siffror, symboler och uttryck i operandfältet.

*Siffror* accepteras av assemblern enligt följande varianter, där Nummer står för ett positivt heltal.

|          |                  |                       |                                         |
|----------|------------------|-----------------------|-----------------------------------------|
| Nummer   | decimalt tal     | (Ex. LDAA #15)        | (Betydelsen av #, se sid 12-7, punkt 3) |
| \$Nummer | hexadecimalt tal | (Ex. LDAA #\$0F)      |                                         |
| @Nummer  | oktalt tal       | (Ex. LDAA #@017)      |                                         |
| %Nummer  | binärt tal       | (Ex. LDAA #%00001111) |                                         |

Dessutom tillåts även följande varianter, även om de inte brukar användas så ofta.

|         |                  |
|---------|------------------|
| NummerH | hexadecimalt tal |
| NummerO | oktalt tal       |
| NummerQ | oktalt tal       |
| NummerB | binärt tal       |

*Symboler* får användas i operandfältet enligt följande regler:

- 1 En symbol får bestå av 1 till 6 alfanumeriska tecken, där det första tecknet måste vara en bokstav.
- 2 Tecknet \* ensamt får användas som symbol och representerar då programräknaren (ett undantag från reglerna enligt punkt 1)

*Symboler* måste definieras i källprogrammet enligt följande regler:

- 1 En instruktion i källprogrammet kan innehålla en label, som är lika med den symbol som skall definieras.

```
Ex.   DELAY LDX #$FFFF
      LOOP  DEX
      BNE  LOOP
      RTS
```

- 2 Assemblerdirektiven FCB, FCC, FDB eller RMB kan skrivas med en label som är lika med den symbol som skall definieras.

```
Ex.      ---
          STACK RMB 1
          ---
          LDS #STACK
          ---
```

- 3 En symbol kan definieras med hjälp av assemblerdirektivet EQU.

```
Ex.      ---
          TAL EQU $20
          ---
          LDAA #TAL
          ---
```

Uttryck i operandfältet kan bestå av en kombination av symboler och/eller siffror, som är åtskilda ifrån varandra med en av de aritmetiska operatorerna +, -, \* eller /. De aritmetiska beräkningarna görs med 16-bitars heltalsoperander. Det största tal ett uttryck kan representera blir alltså  $2^{16} = 65\,536$ . Ett 8-bitars resultat tas från de 8 minst signifikanta bitarna.

Uttryck får inte innehålla mellanslag, eftersom detta tolkas som ett byte av fält.

```
Ex.      ---
          NB EQU 8
          ---
          ADCA 2*NB-1,X    (2*NB-1 = 17 = $11, dvs offset = 17)
```

#### 4 Kommentarfältet

Programmeraren kan lägga in kommentarer i klartext i sitt program. Detta kan man göra för att beskriva en funktion, förklara en subrutin och för dokumentation i största allmänhet.

Förutom att man kan lägga in kommentarer i kommentarfältet i en sats kan man reservera en hel rad för en kommentar. Detta gör man genom att kommentaren inleds med en asterisk, \*, i första positionen.

I kommentarerna får man använda alla ASCII-tecken, som motsvarar de hexadecimala talen \$20 → \$7F (SP → DEL) (Se ASCII-tecken, sid. 12-3)

Vid assembleringen bryr sig assemblern inte om kommentarerna på annat sätt, än att de skrivs ut vid programlistningen.

### 12.2.3 Adresseringsmoder

Assemblern undersöker operator och operand för att avgöra vilken adresseringsmode det är som används. Följande adresseringsmoder förekommer:

- 1 Ackumulator-adressering
- 2 Inherent adressering
- 3 Immediate adressering
- 4 Relativ adressering
- 5 Indexerad adressering
- 6 Direkt adressering
- 7 Extended adressering

#### 1 Ackumulator-adressering

Enligt vad som sas i början av avsnitt 12.2.2, sid. 12-4, skiljer man på de olika fälten i en sats genom att slå ett mellanslag (SP-karaktär). Om man alltså skriver LDA A, innebär detta att A:et hamnar i operand-fältet. I detta fall kommer assemblern att tolka A:et som adressen till ackumulator A. På motsvarande sätt gäller för alla instruktioner som berör ackumulatorerna A och B. Observera dock, att det är tillåtet att skriva t ex LDAA (se sid 12-4). I assemblerlistan kommer det däremot att stå LDA A (se sid. 12-10).

#### 2 Inherent adressering

För vissa instruktioner, exempelvis DEX, finns adressen inbyggd i op-koden. Man behöver därför inte skriva någonting i operandfältet.

#### 3 Immediate adressering

Immediate adressering anges med tecknet # (pound sign). I operand-fältet står därför något av följande alternativ.

|          |    |                                                    |
|----------|----|----------------------------------------------------|
| #Nummer  | Ex | LDAA #\$20, LDX #\$403B                            |
| #Symbol  | Ex | LDAA #TAL                                          |
| #Uttryck | Ex | LDAA #2*NB-1 (NB måste vara definierat någonstans) |
| #'C      | Ex | LDAA #'P, LDX #'P (ASCII-kod för P = \$50)         |

'C instruerar assemblern att översätta karaktären C till motsvarande ASCII-kod. I exemplet gäller alltså, att ackumulator A laddas med \$50 och indexregistret med \$0050. Karaktären C får utgöras av varje ASCII-karaktär som motsvarar ett värde \$20 → \$7F.

#### 4 Relativ adressering

Relativ adressering används endast vid BRANCH-instruktioner och BRANCH-instruktioner har heller ingen annan adresseringsmode. När en BRANCH-instruktion används, behöver man därför inte ange vilken adresseringsmode det är som används.

Ex 1.        ---  
               LOOP LDAB #\$20  
               ---  
               BNE LOOP

Ex 2.        ---  
               LOOP BRA LOOP  
                   (snurrar runt på  
                   samma ställe, dvs  
                   en ändlös loop)

## 5 Indexerad adressering

Indexerad adressering anges normalt på följande sätt:

|           |    |               |                                |
|-----------|----|---------------|--------------------------------|
| Nummer,X  | Ex | STAA 15,X     | (15 = offset, 15 = decimaltal) |
| Symbol,X  | Ex | STAA TAL,X    | (TAL definieras någonstans)    |
| Uttryck,X | Ex | STAA 2*TA-1,X | (TA definieras någonstans)     |

Om man använder en symbol eller ett uttryck, översätter assemblern detta till ett numeriskt värde. Källprogrammet måste därför innehålla en sats någonstans, som anger vilket numeriskt värde symbolen motsvarar.

## 6 och 7 Direkt och extended adressering.

Både direkt och extended adressering anges på följande sätt:

Nummer

Symbol

Uttryck

Eftersom det inte anges något särskilt tecken för dessa två adresseringsmoder, kommer assemblern att uppfatta en adress i området  $0 \rightarrow 255_{10}$  som direkt adressering och adresser  $\geq 256_{10}$  som extended adressering.

## 12.3 Assemblering

Det allmänna utseendet för ASMB-kommandot är:

```
ASMB,(filnamn 1){,(filnamn 2)}{,(optionlista)}
```

Klammer {...} anger att dessa uttryck får utelämnas. Namnet på den fil som skall assembleras är (filnamn 1) och detta namn måste alltid anges. "Default" är .TXT och drive 1, dvs textfilen med namn (filnamn 1) ligger på drive 1 (högra skivan).

Om (filnamn 2) finns angiven, kommer binärfilen att få detta namn. Finns (filnamn 2) inte angiven kommer binärfilen att få samma namn som textfilen, dvs (filnamn 1) men med tillägget .BIN i stället för .TXT.

Om en binärfil med samma namn redan existerar på disken, kommer assemblern att fråga om denna fil skall tas bort (DELETE). Svarar man 'Y', dvs YES, tas den gamla binärfilen bort.

Det finns sju olika "options" att välja på i (optionlistan). Optionlistan är avskild ifrån kommandot med ett plus-tecken (+) och varje option representeras med en bokstav.

De options man har att välja på finns beskrivna på nästa sida.

- B Ingen binärfil skapas på disken och detta gäller även om (filnamn 2) finns angiven. Option B är användbar om man, innan hela programmet är klart, vill se om programmet innehåller fel.
- D Instruerar assemblern att ta bort datum från titelraden. (Se för övrigt PAG, sid 12-16)
- G Kopplar in NOG-option (se sidan 12-15) dvs endast en rad skrivs ut för FCB, FDB och FCC.
- L Ingen assemblerlista skrivs ut, vilket normalt sker. Eventuella fel anges med att felaktig rad skrivs ut med radnummer (samma som i EDITORN), samt typ av fel. Även symboltabellen skrivs ut.
- N Radnummer anges på assemblerlistan, vilket normalt inte sker. Vid felsökning (debugging), kan det vara värdefullt att ha varje rad numrerad. Om option N används, får man en radnumrering som överensstämmer med radnumreringen i EDITORN.
- S Ingen symboltabell skrivs ut. Normalt skrivs en symboltabell ut i slutet av assemblerlistan.
- Y Tar bort varje gammal binärfil (.BIN) med samma namn som källfilen (.TXT). Om option Y inte används och en binärfil med samma namn redan existerar, kommer assemblern att fråga om den gamla binärfilen skall tas bort. (Se föregående sida).

Här följer några exempel på ASMB-kommandon:

Ex 1      +++ASMB,TEST

Här vill vi assemblera källfilen TEST.TXT (TXT = default), som ligger på arbetsskivan (drive 1) och skapa en binärfil med namnet TEST.BIN på samma skiva.

Ex 2      +++ASMB,TEST,+LS

Med detta kommando utförs samma operation som i exempel 1, men ingen assemblerlista skrivs ut. Observera, att det får inte finnas något kommatecken mellan L och S.

Ex 3      +++ASMB,Ø.TEST,1.TEST.CMD,+S

Här assembleras textfilen TEST.TXT på drive Ø och en binärfil som kallas TEST.CMD skapas på drive 1. Ingen symboltabell skrivs ut.

Ex 4      +++ASMB,TEST,+BN

Textfilen TEST.TXT assembleras och en lista med radnummer produceras, men ingen binärfil skapas.

Ex 5      +++P,ASMB,TEST,+N

I de föregående exemplen har assemblerlistan redovisats på terminalen (videoskärmen). Om man vill ha listan utskriven på papper via en printer, sker detta genom kommandot 'P'. (Här förutsätts givetvis att utrustning och operativsystem klarar ett print-kommando).

Assemblern översätter normalt källprogrammet i två steg, pass 1 och pass 2.

Under pass 1 bygger assemblern upp en symboltabell. I denna tabell lagras labels och symboler och de tilldelas numeriska värden vilka också lagras i tabellen.

Under pass 2 kommer de symboler som inte definierades under pass 1 att tilldelas numeriska värden. Nu skrivs också assemblerlistan ut, såvida man inte använder option L (se sid 12-9). Om assemblern vid kontrollen av satsernas utseende finner att man har avvikit från källspråkets regler (*syntax-fel*), anges detta genom att felmeddelande ges.

Figuren nedan visar hur en assemblerlista kan se ut. Kommandot var:

+++P,ASMB,TEST,+N

| ①  | ②             | ③ | ④      | ⑤     | ⑥                          | ⑦ (se nedan)                   |
|----|---------------|---|--------|-------|----------------------------|--------------------------------|
|    |               |   |        | NAM   | TEST                       |                                |
| 1  |               |   |        |       |                            |                                |
| 2  |               |   | *      |       |                            |                                |
| 3  |               |   | *      |       | FINNS TALET X I EN TABELL? |                                |
| 4  |               |   | *      |       |                            |                                |
| 5  | 1000          |   |        | ORG   | \$1000                     |                                |
| 6  | AD15          |   | GETCHR | EQU   | \$AD15                     |                                |
| 7  | AD18          |   | PUTCHR | EQU   | \$AD18                     |                                |
| 8  | AD24          |   | PCRLF  | EQU   | \$AD24                     |                                |
| 9  | AD03          |   | FLEX   | EQU   | \$AD03                     |                                |
| 10 | 1000 36       |   | TABELL | FCB   | '6','9','4'                |                                |
|    | 1001 39 34    |   |        |       |                            |                                |
| 11 |               |   | *      |       |                            |                                |
| 12 | 1003 C6 03    |   | START  | LDA B | #3                         | ANTAL VARDEN I TABELLEN        |
| 13 | 1005 CE 10 00 |   |        | LDX   | #TABELL                    | PEKAR PA FORSTA VARDET         |
| 14 | 1008 BD AD 24 |   |        | JSR   | PCRLF                      | NY RAD                         |
| 15 | 100B BD AD 15 |   |        | JSR   | GETCHR                     | HAMTA ETT TAL FRAN TANGENTBORD |
| 16 | 100E A1 00    |   | RAD    | CMP A | 0,X                        |                                |
| 17 | 1010 27 0D    |   |        | BEQ   | JA                         |                                |
| 18 | 1012 08       |   |        | INX   |                            |                                |
| 19 | 1013 5A       |   |        | DEC B |                            |                                |
| 20 | 1014 27 02    |   |        | BEQ   | NEJ                        |                                |
| 21 | 1016 20 F6    |   |        | BRA   | RAD                        |                                |
| 22 | 1018 86 4E    |   | NEJ    | LDA A | #'N                        |                                |
| 23 | 101A BD AD 18 |   |        | JSR   | PUTCHR                     | SKRIV UT N                     |
| 24 | 101D 20 E4    |   |        | BRA   | START                      |                                |
| 25 | 101F 86 4A    |   | JA     | LDA A | #'J                        |                                |
| 26 | 1021 BD AD 18 |   |        | JSR   | PUTCHR                     | SKRIV UT J                     |
| 27 | 1024 7E AD 03 |   |        | JMP   | FLEX                       |                                |
| 28 |               |   |        | END   | START                      |                                |

NO ERROR(S) DETECTED

SYMBOL TABLE:

|        |      |        |      |       |      |        |      |       |      |
|--------|------|--------|------|-------|------|--------|------|-------|------|
| FLEX   | AD03 | GETCHR | AD15 | JA    | 101F | NEJ    | 1018 | PCRLF | AD24 |
| PUTCHR | AD18 | RAD    | 100E | START | 1003 | TABELL | 1000 |       |      |

|              |                                 |                 |
|--------------|---------------------------------|-----------------|
| Kommentarer: | 1 Radnummer                     | 5 Operatorfält  |
|              | 2 Adress                        | 6 Operandfält   |
|              | 3 Operator och operand i hexkod | 7 Kommentarfält |
|              | 4 Symbol- (label-)fält          |                 |

## 12.4 Assembler-direktiv

Assembler-direktiv är instruktioner som ställs direkt till assemblern. Det är inte alltid att dessa direktiv resulterar i att en maskinkod genereras. Följande direktiv finns:

| Mnemonic | Definition                      | Funktion                                                              |
|----------|---------------------------------|-----------------------------------------------------------------------|
| END      | End of source program           | Anger slutet på ett källprogram                                       |
| EQU      | Equate a symbol to an operand   | Ger en symbol ett numeriskt värde, en annan symbol eller ett uttryck. |
| FCB      | Form constant byte              | } Anger värden och tilldelar data och minnesareor olika adresser.     |
| FCC      | Form constant characters        |                                                                       |
| FDB      | Form double constant byte       |                                                                       |
| LIB      | Process library file            | Hoppar till utpekad fil                                               |
| NAM      | Specify name or title           | } Formatangivelser för källprogram och/eller assemblerlistor          |
| OPT      | Assembler control options       |                                                                       |
| ORG      | Assign origin of programcounter | Anger programmets startadress.                                        |
| PAG      | Skip to next page               |                                                                       |
| RMB      | Reserve memory bytes            | Reserverar minnesceller                                               |
| SPC      | Insert spaces of output listing |                                                                       |
| TTL      | Specify name or title           | Se NAM.                                                               |

På de följande sidorna finns mera detaljerade beskrivningar av de olika assemblerdirektiven, placerade i bokstavsordning.

**END — End of Program****END**

The operator in the last statement of a source program must be END. This operator signals the assembler that the end of the source program has occurred.

The "END" directive must not be written with a label.

The "END" directive is not translated into object code.

The "END" directive allows for the assignment of a transfer address to the binary file created. This can be accomplished by putting a label or value in the 'operand field' of the END statement. As an example suppose the program you are assembling is to start executing at location \$100 and in the source file you have the label START on the statement which is ORGed at \$100. The END statement should now include this label in its 'operand field' in order to assign \$100 as the transfer address

```

                                ORG      $100
START      LDX      $FFFF
            .
            .
            END      START

```

**EQU — Equate Symbol****EQU**

The "EQU" directive is used to assign a value to a symbol. The "EQU" statement must contain a label which is identical with the symbol being defined. The operand field may contain the numerical value of the symbol (decimal, hexadecimal, octal, or binary). Alternatively, the operand field may be another symbol or an expression which can be evaluated by the compiler. The EQU statement is not translated into object code.

The following are examples of valid "EQU" statements:

| Location | Data | Label | Operator | Operand |
|----------|------|-------|----------|---------|
|          | 0A01 | SUN   | EQU      | \$A01   |
|          | 0003 | AB    | EQU      | 3       |
|          | 0A01 | AA    | EQU      | SUN     |
|          | 0A04 | AC    | EQU      | AB+AA   |
|          | 0FC1 | ABC   | EQU      | \$FC1   |

Relating to the use of a symbol or an expression in the operand field, only one level of forward referencing will assemble correctly. This reflects a two-pass characteristic of the assembly process. An (illegal) example of two levels of forward referencing would be:

```

E      EQU      Y
Y      EQU      C
C      EQU      5

```

This will not assemble correctly because E will not be assigned a numerical value at the end of pass 2. E and Y are both undefined throughout pass 1. E is undefined throughout pass 2 and will cause an error

**FCB — Form Constant Byte****FCB**

The "FCB" directive may have one or more operands, separated by commas. An 8-bit unsigned binary number, corresponding to the value of each operand is stored in a byte of the object program. If there is more than one operand, they are stored in successive bytes. The operand field may contain the actual value (decimal, hexadecimal, octal or binary). Alternatively, the operand may be a symbol or an expression which can be assigned a numerical value by the Assembler.

An "FCB" directive followed by one or more void operands separated by commas will store zeros for the void operands. Maximum 255 expressions/FCB

An "FCB" directive may be written with a label.

Examples of valid "FCB" directives follow:

| Location | Data | Label | Operator | Operand  |
|----------|------|-------|----------|----------|
| 0000     | FF   | TOP   | FCB      | \$FF     |
| 0001     | 00   | TAB   | FCB      | ,\$F,23, |
| 0002     | 0F   |       |          |          |
| 0003     | 17   |       |          |          |
| 0004     | 00   |       |          |          |
| 0005     | E5   |       | FCB      | *+\$E0   |

**FCC — Form Constant Characters****FCC**

The "FCC" directive translates strings of characters into their 7-bit ASCII codes. Any of the characters which correspond to ASCII hexadecimal codes 20 (SP) thru 7F (DEL) can be processed by this directive.

1. Count, comma, text. Where the count specifies how many ASCII characters to generate and the text begins following the first comma of the operand. Should the count be longer than the text, spaces will be inserted to fill the count. Maximum count is 255.
2. Text enclosed between identical delimiters, each being any single character. (If the delimiters are numbers, the text must not begin with a comma.)

If the string in the operand comprises more than one character, the ASCII codes corresponding to the successive characters are entered into successive bytes of memory.

An "FCC" directive may be written with a label.

The following are examples of valid "FCC" directives:

| Location | Data | Label | Operator | Operand |
|----------|------|-------|----------|---------|
| 0A00     | 54   | MSG1  | FCC      | /TEXT/  |
| 0A01     | 45   |       |          |         |
| 0A02     | 58   |       |          |         |
| 0A03     | 54   |       |          |         |
| 0A04     | 54   | MSG2  | FCC      | 7,TEXT  |
| 0A05     | 45   |       |          |         |
| 0A06     | 58   |       |          |         |
| 0A07     | 54   |       |          |         |
| 0A08     | 20   |       |          |         |
| 0A09     | 20   |       |          |         |
| 0A0A     | 20   |       |          |         |

**FDB — Form Double Constant Byte****FDB**

The "FDB" directive may have one or more operands separated by commas. The 16-bit unsigned binary number, corresponding to the value of each operand is stored in two bytes of the object program. If there is more than one operand, they are stored in successive bytes. The operand field may contain the actual value (decimal, hexadecimal, octal or binary). Alternatively, the operand may be a symbol or an expression which can be assigned a numerical value by the assembler.

An "FDB" directive followed by one or more void operands separated by commas will store zeros for the void operands. The maximum number of operands is 127.

An "FDB" directive may be written with a label.

Examples of valid "FDB" directives follow:

| Location | Data | Label | Operator | Operand          |
|----------|------|-------|----------|------------------|
| 0010     | 0002 | TWO   | FDB      | 2                |
| 0012     | 0000 | MASK  | FDB      | ,\$F,\$EF,,\$AFF |
| 0014     | 000F |       |          |                  |
| 0016     | 00EF |       |          |                  |
| 0018     | 0000 |       |          |                  |
| 001A     | 0AFF |       |          |                  |

**LIB - Library****LIB**

This operator causes FLEX to temporarily leave the current source file and open the file specified (on the working drive and with a default extension of TXT) for source processing. It has the following form:

LIB file spec

Note that no label is allowed. The LIB will not show up in the output listing, but instead the assembled source from the file specified will be listed. The LIB operator is similar to a macro call, except the source is found on an external file and no parameter passing is supported. If an END statement is found in the LIB file it will be ignored. When the file has been depleted, the assembler will switch back to the original file and continue where it left off. The LIB operators can not be nested or in other words, one LIB file may not call another LIB file.

**NAM — Name****NAM**

The NAM and TTL operators are used to assign a title to be printed at the top of all pages (other than page 0) if PAG option is on. If the PAG option is off this operator has no effect. The format as shown below allows up to 32 characters in the title. No label is allowed and no code is generated.

TTL text for the title

If more than one NAM or TTL operator appears the last one "executed" will be printed on the next page.

**OPT — Option****OPT**

The "OPT" directive is used to give the programmer optional control of the format of assembler output. The "OPT" directive is not translated into machine code.

No label may be used with the "OPT" directive.

The options are written in the operand field following the directive, and are separated by commas.

OPT option 1, option 2, ... , option N

The allowable options are:

|     |            |                                                       |
|-----|------------|-------------------------------------------------------|
| SYM | Symbol     | print sorted symbol table after the listing (default) |
| NOS | Nosymbol   | do not print the symbol table                         |
| GEN | Generate   | print code generated by FCB, FDB and FCC (default)    |
| NOG | Nogenerate | print only one line for each FCB, FDB or FCC          |
| LIS | List       | print the assembled source listing (default)          |
| NOL | Nolist     | suppress the printing of the source listing           |
| PAG | Page       | enable page formatting and numbering                  |
| NOP | Nopage     | disable page mode (default)                           |

The disk version operates a little different than the standard assembler. The options NOS, NOG, NOL and PAG have no effect when contained in the source file. The calling option parameters S, G, L and D should be used respectively. (See page 12-9)

**ORG — Origin****ORG**

The assembler directive "ORG" defines the numerical address of the first byte of machine code which results from the assembly of the immediately subsequent section of a source program. There may be any number of "ORG" statements in a program. The "ORG" directive sets the program counter to the value expressed in the operand field.

The operand field may contain the actual value (decimal, hexadecimal, octal or binary) to which the program counter is to be set. Alternatively, the operand field may contain a symbol or an expression which can be assigned a numerical value by the assembler.

The location counter is initialized before each assembly. If no "ORG" statement appears at the beginning of the program, the location counter will begin as if an "ORG" zero had been entered.

An "ORG" directive must not be written with a label.

The ORG statement does not translate into object code.

The following are examples of valid ORG statements:

| Location | Data    | Label   | Operator | Operand |
|----------|---------|---------|----------|---------|
| 0064     |         | (blank) | ORG      | 100     |
| AF23     |         | (blank) | ORG      | \$AF23  |
|          |         | BEGIN   | EQU      | \$1100  |
| 1100     | (blank) | ORG     | BEGIN    |         |

**PAG — Advance Paper to Top of Next Page****PAG**

The PAG operator, if the PAG option is on, causes a page eject and subsequently causes the title (if any), date (if D not set) and page number to be printed at the top of the next page. No label is allowed and no code is produced. Notice that the first page of any listing is page 0 and no title is printed on that page. The PAG operator itself will not appear in the listing. The usual procedure is to have all the options and the title declaration followed by a PAG by the first statements in a program.

**RMB — Reserve Memory Bytes****RMB**

The "RMB" directive causes the location counter to be increased by the value of the operand field. This reserves a block of memory whose length is equal to the value of the operand field. The operand field may contain the actual number (decimal, hexadecimal, octal or binary) equal to the number of bytes to be reserved. Alternatively the operand may be a symbol or an expression which can be assigned a numerical value by the assembler.

The block of memory which is reserved by the "RMB" directive is unchanged by that directive.

The "RMB" directive may be written with a label.

Examples of valid "RMB" directives follow (the data column indicates the number of bytes being reserved):

| Location | Data | Label   | Operator | Operand |
|----------|------|---------|----------|---------|
| 0100     | 0004 |         | RMB      | 4       |
| 0104     | 0016 | TABLE 1 | RMB      | 20      |
| 0118     | 0016 | TABLE 2 | RMB      | 20      |

**SPC — Space N Lines****SPC**

The "SPC" directive provides n vertical spaces for formatting the program listing. It does not itself appear in the listing. The number of lines to be left blank is stated by an operand in the operand field.

The operand would normally contain the actual number (decimal, hexadecimal, octal or binary) equal to the number of lines to be left blank. A symbol or an expression is also allowed.

The "SPC" directive must not be written with a label.

**TTL - Title****TTL**

See NAM