Assembler

ABC 80

Assembler

ABC 80 ASSEMBLER

Programpaketet omfattar tre program:

ABC 80-assembler ASM med datafil ASMCON, textredigeringsprogrammet EDITOR samt hjälpprogrammet Hexmon, som är ett testhjälpmedel för assemblerprogram. Programpaketet finns dessutom i två versioner, en kassett-version och en flexskive-version. Bruksanvisningen behandlar främst kassettversionen men alla skillnader mellan de två versionerna beskrivs i Appendix A, sid 34.

I N N E H Å L L S F Ö R T E C K N I N G

VARFÖR ASSEMBLER

ABC 80, liksom varje dator, arbetar i princip med tre språknivåer - maskinspråk, maskinära språk och högnivåspråk. I den lägsta nivån - maskinspråk - är alla instruktioner och alla data skrivna precis som datorn vill ha dem, dvs som en sifferkombination. Maskinspråk är följdaktligen väldigt lämpligt för datorn men tyvärr väldigt olämpligt för människan/programmeraren. Redan efter ett par instruktioner är det väldigt svårt att se vad som händer. Det är till exempel inte särskilt lätt att se att nedanstående sekvens betyder: Lägg ihop två och två och placera resultatet i minnet på adress 65408.

00111110	3E
00000010	2
11000110	C6
00000010	2
00110010	32
10000000	80
11111111	FF

Det blir inte heller mycket lättare av att man vanligen anger dessa noll/ett-kombinationer skrivna i hexadecimal form (som i kolumnen till höger). Det blir bara färre siffror. (Det hexadecimala talsystemet har basen 16, istället för 10 som i det decimala systemet).

Maskinspråk ger alldeles tydligt en för människan tämligen oläslig siffermassa, även för de enklaste instruktioner. Dessutom är maskinspråket alltid bundet till en viss maskin eller processor. Inne i datorn tar dock dessa maskininstruktioner inte upp mer plats än precis vad som är nödvändigt. Därtill är maskininstruktionerna direkta och precisa order till datorn - och kan följdaktligen behandlas mycket snabbt.

Om vi istället tittar på språk ur människans synvinkel, ser det lite annorlunda ut. Motsvarande sekvens skriven i ett högnivå språk som BASIC är betydligt lättare att läsa och att förstå.

```
10 A = 2 + 2
20 PRINT A
```

I datorn kommer BASIC-sekvensen att ta betydligt större plats och kommer dessutom att ta längre tid att utföra. Normalt sett spelar detta ingen som helst roll. Det bekväma och människotillvända sättet att programmera i BASIC har så många fördelar att eventuella nackdelar är att betrakta som marginella. BASIC är dessutom i stort sett maskinoberoende, då de tämligen internationella BASIC-instruktionerna översätts internt av en för varje dator anpassad BASIC-tolk.

I vissa fall vill man dock nå alla datorns möjligheter. Det kan gälla rutiner som är svåra att uttrycka i BASIC eller rutiner som skall utnyttjas väldigt många gånger i ett program och där följdaktligen stora tidsbesparingar kan göras. Så länge som dessa rutiner är korta kan man skriva den vitala delen i maskinkod och lägga in den direkt i minnet med POKE-instruktionen, varefter rutinen anropas av ett BASIC-program med instruktionen CALL. Det finns flera nackdelar med detta tillvägagångssätt. Dels blir programmen, som vi sett tidigare, tämligen snabbt oöverskådliga och dels är man tvungen att ange alla adresser, såväl startadress för programmet som hopp och lägen, redan från början. Arbetet att räkna fram korrekta adresser måste då göras av programmeraren. Detta arbete är inte bara besvärligt - det resulterar dessutom väldigt ofta i att fel smyger sig in. Vad är då mer naturligt än att ta hjälp av ett program för dessa uppgifter?

Sådana hjälpprogram tillhandahålls också av alla dator- och processortillverkare. Med dessa hjälpprogram - assemblatorer - har man kunnat gå ännu något längre för att underlätta programmerarens arbete. Programmeraren kan då skriva instruktionerna i en förkortad operationskod, op-kod eller memokod, varefter assemblatorn tar hand om översättningen till maskinkod. Dessutom kan lägesnamn användas i stället för absoluta adresser, varefter man låter assemblatorn beräkna de resulterande adresserna. Detta sätt att skriva program kallas maskinär programmering eller assembler-programmering.

Tillvägagångssättet blir följande: Med hjälp av tillverkarens assembler-manual, som innehåller en beskrivning av den aktuella datorns eller processorns instruktionsrepertoar samt regler för hur dessa instruktioner får sättas ihop, skriver man ett program i form av op-koder. Detta program kallas källprogram.

Är programmet kort kan man översätta op-koderna till maskinkod för hand (varje instruktion anges i assembler-manualen både med sin op-kod och dess binära och hexadecimala motsvarighet - maskinkoden). Det resulterande programmet kallas objektprogram och kan läggas in i minnet med en eller flera POKE-instuktioner. Är källprogrammet bara något större än "väldigt kort" lönar det sig att låta assemblatorn göra översättningen. Förutom en kontroll av programmets formella riktighet får man då dessutom ut resultatet på en fil i precis den form man vill ha det, dvs som en eller flera POKE-instruktioner.

Det tidigare exemplet med att lägga ihop två och två får följande utseende om det skrivs i form av op-koder:

```
Program                 Kommentar
LD    A,2               ; ladda register A med 2
ADD   A,2               ; addera 2 till register A (result-
                          atet sparas i A)
LD    (65408),A         ; ladda minnesadress 65408 med talet
                          i register A
```

Ett sådant assembler-program är betydligt lättare att förstå (och att skriva) än maskinspråksprogrammet i exempel 1 - trots att programmet utför samma saker.

En beskrivning av assemblerspråket finns i Z 80 Assembly Language Manual.

ALLMÄN BESKRIVNING

Ett assembler-program är ett program, som skrivs i form av förkortade operationskoder - op-koder - där varje op-kod motsvarar en maskininstruktion. Resultatet blir ett källprogram, som sedan skall översättas till lämplig form av ett hjälpprogram - assemblator. Resultatet av översättningen (assembleringen) fås på en fil i form av en eller flera POKE-satser och kallas objektprogram. (Ofta kallas såväl källprogram som objektprogram för assembler-program, eventuellt med tillägget källkod eller objektkod).

Precis som i BASIC består ett assembler-program av ett antal rader med text. Vid assembler-programmering tillåts bara en instruktion, dvs bara en sats per rad. Denna sats kan antingen vara ett direktiv till assemblatorn (en s k pseudoinstruktion) eller en maskininstruktion. Består satsen av en pseudoinstruktion, översätts den inte till maskinkod utan fungerar bara som en order till översättningsprogrammet - assemblatorn. Består satsen av en maskininstruktion, måste den skrivas i en bestämd form med minst ett och upp till fyra ingående element. Dessa element är: Läge, operation, operander samt kommentar. Den aktiva delen i satsen - instruktionsdelen - består av en förkortad operationskod, eller op-kod, samt en eller flera operander. Op-koden är en förkortning av det fullständiga operationsnamnet och beskriver på kortast möjliga sätt vad instruktionen innebär. En instruktion kan till exempel se ut som: LD A,B. Op-koden LD står för Load (ladda) och A respektive B är operander. Instruktionen betyder: Ladda register A med innehållet i register B. Vid assembleringen kommer instruktionen att översättas till maskinkoden 78H (H står för hexadecimal kod).

Vid assembler-programmering av ABC 80 tillåter Z80-assemblatorn 74 olika operationer och 25 olika operandtyper, som kan kombineras till 694 olika maskininstruktioner. En fullständig beskrivning av alla dessa instruktioner finner man i "Z80-Assembly Language Programming Manual".

HUR ETT PROGRAM ANVÄNDS - START, INDATA, UTDATA

Ett assembler-program används vanligen som en subrutin till ett BASIC-program och anropas då från BASIC-programmet med instruktionen CALL (u%) där u% är assembler-programmets startadress. Är assembler-programmet ett självständigt program startas programmet även i detta fall med instruktionen CALL (u%).
I båda fallen måste CALL användas som en funktion, dvs CALL medför att en variabel tilldelas ett värde (hämtat från Z80-processorns HL-register) vid återhoppet från assembler-programmet.
Varje program, och alltså även assembler-program, behöver i regel någon form av indata att arbeta med. Det finns i princip två möjligheter för ett assembler-program att få indata. Den ena metoden är att låta anropsinstruktionen för med sig ett värde, som då hamnar i Z80-processorns DE-register, genom att skriva anropsinstruktionen som CALL (u%, u_1%). I denna instruktion är u% startadressen och u_1% det värde som läggs i DE-registret. Den andra metoden är att assembler-programmet hämtar indata från specificerad minnesadress. I detta fall måste önskade indata ha lagrats i respektive minnesadress i ett tidigare skede - antingen av ett BASIC-program, som med instruktionen POKE har lagt ut data till önskad minnesadress eller manuellt genom att POKE använts som ett kommando för att lägga ut data till en önskad minnesadress.

I regel skall programmet också producera någon form av utdata. Analogt med ovanstående kan även detta ske på i princip två sätt. Den ena möjligheten är genom CALL-instruktionen som, i och med att den arbetar som en funktion, för med sig ett värde (hämtat från HL-registret) tillbaka, ex A% = CALL (u%). Den andra möjligheten är att låta assembler-programmet lägga ut data till önskad minnesadress, där dessa utdata sedan kan hämtas med PEEK-instruktionen. Då PEEK arbetar som en funktion kan hämtningen ske såväl av ett BASIC-program som rent manuellt.

Det är viktigt att ha ovanstående resonemang klart för sig redan vid konstruktionen av assembler-programmet, då det i regel gäller att få till stånd ett fungerande samarbete mellan ett BASIC-program och ett assembler-program. För en närmare beskrivning av PEEK, POKE och CALL hänvisas till ABC 80:s bruksanvisning, sid 44-45 samt sid 58-60.

HUR ETT PROGRAM ÄR UPPBYGGT

Ett assembler-program består av en följd av satser, som tillsammans formar ett program. För att programmet skall kunna assembleras (översättas) måste vissa direktiv till assemblatorn finnas med. Dessutom är det fördelaktigt att förse källprogrammet med inledande kommentarer, namn på programmet, etc för att man även i framtiden skall kunna veta vad det är för program och vad det är för speciellt med programmet. Vanligen är det ju så att man sparar programmet på en fil, som man kanske inte tittar på så ofta. Att göra om samma program en gång till efter ett halvår bara för att man slarvat med dokumentationen är inte roligt. Efter de inledande kommentarerna följer ett eller flera direktiv till assemblatorn. Ett sådant direktiv är absolut nödvändigt - nämligen informationen om var programmet skall placeras i minnet. Om man vill att programmets startadress skall vara 65408 ser instruktionen ut så här: ORG 65408. Man måste naturligtvis se till att startadressen refererar till en tillåten ledig plats i minnet (se ABC 80´s bruksanvisning, sid 50). Därefter följer den aktiva delen av programmet. Givetvis kan pseudoinstruktioner (direktiv till assemblatorn) ingå även i denna del. Den aktiva delen av programmet avslutas lämpligen med op-koden RET, som medför ett returhopp till det anropande BASIC-programmet. (Används assembler-programmet självständigt, medför RET att körningen avslutas). Därefter avslutas assemblerprogrammet med pseudoinstruktionen END som talar om att programmet är slut. Assemblatorn ignorerar allt som står efter END.

RADFORMAT

Varje rad i ett assembler-program måste skrivas i en bestämd form med upp till fyra ingående element (eller fält). Dessa element är: Läge, op-kod, operand eller operander samt kommentar.

Exempel på en fullständig rad:

Läge	Op-kod	Operander	Kommentar
HIT:	SBC	HL, BC	; Beräkna resten

Läget (HIT) används för att man skall kunna referera till raden. Lägesfältet kan antingen användas för ett rent läge, dvs en hoppadress som kan refereras till, eller för en symbol som då står för ett bestämt numeriskt värde. Även symbolen kan naturligtvis refereras till. Ett läge eller en symbol måste alltid följas av kolon om det inte står längst till vänster i raden. (Se även "Symboler och Lägen" på sidan 10 för ytterligare information).

Nästa fält är avsett för operationskoden. Op-koden får inte skrivas längst ut till vänster på raden utan måste då föregås av minst ett mellanslag. Inleds raden med ett läge måste även i detta fall op-koden avskiljas med minst ett mellanslag.

Därefter följer operandfältet som även detta måste avskiljas från föregående fält med minst ett mellanslag. Operanderna avskiljs sinsemellan med ett kommatecken.

Det sista fältet på raden är kommentarfältet som alltid måste inledas av ett semikolon. Kommentarfältet får skrivas var som helst på raden - dock alltid som sista fält. Kommentaren är enbart till för att människan/programmeraren lättare skall förstå vad programmet gör och ignoreras helt av assemblatorn.

Alla dessa delar behöver inte finnas med på varje rad i programmet. Kommentarer och lägen kan alltid utelämnas. Dessutom finns det vissa operationskoder som inte har några operander.

Det är även tillåtet att skriva rader som bara innehåller en kommentar och/eller ett läge. Rader med enbart kommentarer kan många gånger vara praktiska för att beskriva ett helt program, eller avsnitt av program.

PSEUDOINSTRUKTIONER

Som tidigare nämnts består ett assembler-program inte bara av maskininstruktioner. I programmet ingår även nödvändiga direktiv till assemblatorn, s k pseudoinstruktioner. Dessa pseudoinstruktioner översätts inte till maskinkod utan används bara för att styra översättningsprogrammets arbete.

I ABC 80 assemblatorn ingår följande pseudoinstruktioner:

ORG nn — Laddar assemblatorns adressräknare (Location counter) med värdet nn. ORG används för att ange programmets startadress. (Observera att startadressen måste vara tillåten - se ABC 80:s bruksanvisning, sid 50).

EQU nn — Ger symbolen som står i lägesfältet värdet nn. EQU används för att definiera konstanter i programmet.

END — Markerar slutet på programmet. Assemblatorn ignorerar allt som står efter END.

DEFB n — Reserverar en byte i minnet och laddar den med värdet n. Den reserverade byten får den adress som just då anges av adressräknaren.

DEFB 's' — Reserverar en byte i minnet och laddar den med ASCII-representationen av tecknet 's'. (Specialfall av ovan stående). Den reserverade byten får den adress som just då anges av adressräknaren.

DEFW nn — Reserverar två bytes i minnet och laddar dessa med värdet nn. Observera att den minst signifikanta byten lagras först i minnet. Den första reserverade byten får den adress som just då anges av adressräknaren och den andra reserverade byten samma adress+1.

DEFS nn — Reserverar nn bytes i minnet utan att ge något värde. Den första reserverade byten får den adress som just då anges av adressräknaren, den andra reserverade byten får nästföljande adress, osv.

DEFM 'ss' — Reserverar lika många bytes i minnet som det finns tecken i stängen 'ss' och laddar dessa med ASCII-representationen av tecknen. Om strängen skall innehålla apostrofer måste dessa dubbeltecknas. T ex:

SÄG: "HEJ" skrivs som 'SÄG: "HEJ"'. Strängen får innehålla max 16 tecken. Den första reserverade byten får den adress som just då anges av adressräknaren, den andra reserverade byten får nästföljande adress, osv.

Pseudoinstruktionerna får ha lägen och kommentarer precis som alla andra satser. EQU-instruktionen har ingen effekt om den inte har ett läge (eller symbol).

SYMBOLER OCH LÄGEN

Den beteckning som skrivs i lägesfältet på en rad kallas för en symbol eller ett läge. I båda fallen representerar beteckningen ett 16-bitars heltal. Om man kallar beteckningen för en symbol eller för ett läge beror på hur den används. Det 16-bitars heltal som beteckningen representerar kan nämligen vara antingen en adress till något ställe i minnet, och är då ett läge, eller en numerisk heltals-konstant, och är då en symbol. För själva beteckningen gäller samma restriktioner och begreppen symbol eller läge kan då betraktas som synonyma.

En symbol får bestå av 1 - 6 tecken. Det första tecknet måste vara en bokstav och de övriga får vara bokstäver, siffror eller något av tecknen understreck (_) eller frågetecken (?). Som bokstäver räknas A-Ö samt É och ü (se även "Stora och små bokstäver" på sid 14). Symbolerna får inte innehålla mellanslag (blanktecken).

Exempel på tillåtna symboler:

```
PROG1
KLART?
DEL_1
```

För att en symbol skall kunna användas måste den definieras. En symbol definieras genom att den står i lägesfältet på en rad. Symbolen måste börja i första positionen på raden om den inte följs av ett kolon (:). En symbol får bara definieras på ett ställe i ett program. Försök att definiera en symbol på flera ställen resulterar i felutskrift. (Symbolen får givetvis anropas hur många gånger som helst).

I assemblatorn finns en räknare som kallas adressräknare (Location Counter). Den används för att hålla reda på den aktuella adressen i minnet. När ett läge definieras tilldelas läget det värde, som just då anges av adressräknaren. Om en rad ser ut som: PROG1 ORG 65408 ; början av beräkningen
kommer beteckningen PROG1 (alltså läget) att tilldelas värdet 65408, som är adressen till programmets början. Om beteckningen i lägesfältet är en symbol, som står för en konstant so t ex
KONST EQU 73 ; första konstant
kommer beteckningen KONST (alltså symbolen) att tilldelas värdet 73.

Förutom de symboler som definieras i programmet finns det en särskild symbol, sol (¤) som alltid har samma värde som det aktuella värdet i adressräknaren. Denna symbol kan användas när man vill referera ett antal bytes framåt eller bakåt i programmet.

Beteckningen i lägesfältet, som kan vara en symbol eller ett läge, representerar ett 16-bitars binärt heltal. Detta heltal kan antingen betraktas som ett positivt heltal mellan 0 och 65535, eller som ett heltal i 2-komplementär form mellan -32768 och 32767. Eftersom assemblatorn inte bryr sig om overflow vid addition eller subtraktion spelar det ingen roll hur man ser på talen. Så länge man håller sig inom talområdet -32768 till 65535 kommer resultatet att bli riktigt. (Precis samma regler gäller ju även för heltal i ABC 80-BASIC).

Det finns en gräns för hur många symboler eller lägen som får finnas i ett program. Denna gräns beror på hur stort minnet är och på vilken version (kassett eller flexskiva) av assemblatorn som används. Se "Begränsningar - kassettversionen", sidan 18, respektive "Begränsningar - flexskiveversionen", sidan 38.

UTTRYCK

I många instruktioner förekommer uttryck som operander. Ett uttryck (expression) kan i det här sammanhanget bestå av en enda term eller av flera termer åtskilda av operatorer. En term kan vara en konstant eller en symbol. Följande operatorer är tillåtna:

Prioritet	Operator	Betydelse	Exempel
1	-	negation	-2
2	*	multiplikation	2*VÄRDE
2	/	division	TAB/8
3	+	addition	A+B
3	-	subtraktion	BRUTTO-RABATT

Prioriteringen (dvs rangordningen) mellan operatorerna är precis samma som vid vanlig skolmatematik, dvs först utförs negation, därefter multiplikation och division och slutligen addition och subtraktion. Vid samma prioritet utförs operationerna från vänster till höger. Beräkningsgången kan ändras med parenteser, varvid uttrycket inom parenteser behandlas först, precis som vanligt. Observera dock att ett uttryck som är helt omslutet av parenteser betecknar en minnesadress. Exempelvis betyder 65408 värdet 65408 medan (65408) betyder värdet som finns på adress 65408.

Ett uttryck får inte innehålla mellanslag (blanktecken) eller kommatecken eftersom dessa tecken används för att avskilja fälten på raden.

Uttrycken behandlas precis som heltalsuttryck i ABC 80-BASIC. Detta innebär bl a att man inte får overflow vid addition eller subtraktion. (Detta löser de tidigare antydda problemen med negativa och positiva tal). Det innebär också att tecknet " / " står för heltalsdivision, dvs eventuella decimaler huggs av. 5/3 blir alltså 1.

Allt detta kan låta komplicerat. Den som är ovan vid räkning med 2-komplementära tal kan dock trösta sig med att man i praktiken inte märker någonting av allt detta. Uttrycket får helt enkelt de värden man väntar sig.

Ett uttryck behöver givetvis inte innehålla några operationer. De allra flesta uttryck som ingår i ett program består i själva verket av en ensam konstant eller symbol.

KONSTANTER

Som tidigare nämnts får ett uttryck innehålla konstanter. Dessa kan vara av två typer - numeriska konstanter och teckenkonstanter. En numerisk konstant är helt enkelt ett heltal skrivet i någon lämplig bas. En teckenkonstant däremot är ASCII-representationen av ett tecken.

Numeriska konstanter

Assemblatorn kan hantera tal som är skrivna i flera olika talsytem (olika baser). Det är många gånger praktiskt att utnyttja denna möjlighet för att göra programmet lättare att förstå. Det är t ex ofta lämpligt att skriva tal som skall ingå i logiska operationer i binär form. Det är då lättare att se vilka bitar som påverkas.

Ett tal består av en följd av siffror. Inte bara siffrans valör utan även dess plats i talet har betydelse för värdet av talet. Normalt sett uppfattar vi detta positionssystem som helt naturligt. Man förväxlar t ex aldrig 25 med 52. När man skall utnyttja andra talsystem med andra baser än 10 kan det vara bra att veta hur det här med baser och positionssystem egentligen fungerar. Skriver man 352 i ett talsystem med basen 10 (dvs precis som vanligt) betyder det 3 x basen$^{\text{positionen}}$ + 5 x basen$^{\text{pos}}$ + 2 x bas$^{\text{pos}}$ vilket blir 3 x 10^2 + 5 x 10^1 + 2 x 10^0 = 352. (Positionerna räknas från höger till vänster med början vid noll). Precis på samma sätt får man värdet av tal som skrivits i andra baser. Det hexadecimala talsystemet har basen 16 vilket innebär, att om 352 i exemplet ovan från början hade skrivits hexadecimalt, dvs 352H hade värdet varit 3 x 16^2 + 5 x 16^1 + 2 x 16^0 = 1618 decimalt. När man skall skriva tal hexadecimalt räcker de vanliga siffrorna inte till därför fyller man på med A, B, C, D, E och F för att nå upp till 15. Det oktala talsystemet har basen 8 och består därför bara av siffrorna 0-7, medan det binära talsystemet har basen 2 och följdaktligen bara består av nollor och ettor.

Ett tal måste alltid inledas med en siffra, som eventuellt kan vara en inledande nolla. Talet får följas av en bokstav, som anger i vilken bas talet är skrivet. Utelämnas bokstaven tolkas talet som decimalt (bas 10).

Följande baser är tillåtna:

Bas	Bokstav
2 (binärt talsystem)	B
8 (oktalt talsystem)	O eller Q
10 (decimalt talsystem)	D eller ingen
16 (hexadecimalt talsystem)	H

Exempel: Talet 217 kan skrivas på följande sätt:

```
11011001B, 331O, 331Q, 217D, 217, 0D9H
```

OBS: Om den inledande nollan i 0D9H utelämnas kommer D9H att tolkas som ett symbolnamn. Detta resulterar antagligen i felutskriften "Odefinierad symbol".

Teckenkonstanter

En teckenkonstant består av ett ASCII-tecken omgivet av apostrofer. Värdet av en teckenkonstant är lika med tecknets ASCII-representation.

Exempel på teckenkonstanter:

Konstant	Värde
'A'	41H
'a'	61H
'1'	31H
'4'	34H

Observera att citationstecken inte dubbeltecknas om de ingår i en teckenkonstant.

Stora och små bokstäver

Assemblatorn skiljer normalt inte mellan stora och små bokstäver. Det finns tre undantag från denna regel:

1) teckenkonstanter
2) tecken i en DEFB-sats
3) tecken i strängen i en DEFS-sats

Detta innebär att satserna

```
HiT   Ld  A,0FH
    och
hiT   LD  a,0fH
```

tolkas helt lika av assemblatorn. Däremot betecknar 'A' värdet 41H medan 'a' betecknar värdet 61H.

ALLMÄNT OM ASSEMBLATORN

Assemblatorns uppgift är att översätta ett program som är skrivet i assembler, dvs i form av op-koder, till maskinkod. Detta kallas assemblering. Det program som skall översättas (assembleras) kallas källprogram, medan det översatta programmet kallas objektprogram.

Innan ett program kan assembleras måste det matas in och sparas på en fil. Detta görs lättast med hjälp av programmet EDITOR (se sid 20), men kan även göras enligt en annan metod (se sid 19).

Assemblatorn är en s k två-pass assemblator. Detta innebär, att assemblatorn läser källprogrammet två gånger. I första passet översätts alla instruktioner, vilka även kontrolleras med avseende på formell riktighet. I det andra passet beräknas alla relativa adresser. Om källprogrammet finns på kassett bör det därför vara lagrat två gånger efter varandra (vilket sker automatiskt när programmet EDITOR används för att lagra källprogrammet).

Assemblatorn finns i två versioner. En kassett-version och en flexskive-version. Här kommer närmast kassett-versionen att beskrivas. Skillnaderna mellan flexskive-versionen och kassettversionen beskrivs i appendix A, sid 34.

Som utdata ger assemblatorn dels ett objektprogram och dels en utskrift av programmet på bildskärmen. Denna utskrift kallas programlista. I flexskive-versionen skrivs objektprogrammet ut direkt på en fil. Detta innebär, att programmen kan vara i stort sett obegränsat stora. I kassett-versionen däremot sparas objektprogrammet i minnet tills att assembleringen är klar. Därefter skrivs det ut på kassetten. Detta begränsar programmets storlek till ca 300 rader. I kassett-versionen erhålls programlistan på bildskärmen och i flexskive-versionen kan programlistan dessutom erhållas som en fil.

OBS: De felutskrifter som erhålls av assemblatorn hänvisar till appendix B.

Handhavande, kassett-versionen

Kassett-versionen av ABC 80-assemblatorn består av två delar. Dels själva programmet, ASMCAS och dels en fil, ASMCON, som ligger omedelbart efter ASMCAS på kassetten.

När ett källprogram, som är lagrat på en fil på kassett, skall assembleras, blir tillvägagångssättet följande:

1. Ladda in assemblatorn till ABC 80's minne. Följande metod är lättast att använda: Spola tillbaka kassetten till början av programmet ASMCAS. Skriv därefter RUN ASMCAS (eller RUN CAS:), varefter först programmet ASMCAS läses in, vilket i sin tur omedelbart läser in filen ASMCON. När inläsningen är klar, kommer följande att visas på bildskärmen:

   ABC 80 assembler

   Vad skall listas på skärmen?

   H = hela programmet

   F = endast felaktiga rader

   ?

   Efter att man svarat H eller F och tryckt på RETURN visas följande:

   Tryck på RETURN när bandspelaren är klar för PASS 1

2. Lägg i kassetten med källprogrammet så att bandet står vid början av källfilen. Därefter kan man trycka på RETURN, varvid assembleringen börjar.

   I och med att ABC 80-assemblatorn är en två-pass assemblator kommer källprogrammet att läsas två gånger. Är källprogrammet lagrat med kommandot CSAVE av programmet EDITOR är källprogrammet automatiskt lagrat två gånger efter varandra, så att inga speciella åtgärder behöver vidtagas mellan passen. Är källprogrammet däremot bara lagrat en gång på kassetten måste kassetten spolas tillbaka till början av källprogrammet innan pass 2 kan påbörjas.

   OBS: Om fel upptäcks under pass 1 kommer pass 2 inte att utföras.

   OBS: De felutskrifter, som fås av assemblatorn, hänvisar till appendix B.

När assembleringen slutförts frågar assemblatorn om man vill att objektprogrammet skall lagras på en fil. Vill man det så måste man sätta i den kassett som man vill att objektprogrammet skall sparas på och sätta bandspelaren på inspelning. Efter att man talat om vad objektfilen skall heta, och tryckt på RETURN, spelas objektprogrammet in. Objektprogrammet är nu lagrat på kassetten i form av en eller flera POKE-satser med början vid den önskade minnesadressen (assembler-programmets startadress), dvs lagrat som ett Basic-program.

Vill man köra objektprogrammet måste man ladda in detta i ABC 80's minne precis på samma sätt som med ett vanligt BASIC-program - alltså med kommandot LOAD filnamn (alternativt LOAD CAS:). Detta resulterar i att objektprogrammet, som har formen av en eller flera POKE-satser, laddas in i minnet som ett BASIC-program. Kör man detta BASIC-program medför POKE-satserna att maskinkoden laddas in i minnet med början vid assembler-programmets startadress. Maskinkodsprogrammet kan sedan anropas med CALL (u%), där u% är programmets startadress. (Se även "Hur ett program används - Start, Indata").

Detta kanske låter lite rörigt, men studera exemplet på sid 36 så klarnar det säkert.

Objektformat

Resultatet av assembleringen blir ett objektprogram med maskininstruktioner lagrade i form av en eller flera POKE-satser. Maskininstruktionerna är alltså skrivna i maskinkod som POKE-satsen lägger in i minnet när den utförs. POKE-satsen, eller POKE-satserna, är försedda med radnummer och bildar därför ett litet BASIC-orogram. Dessa rader kan numreras om och sedan läggas in i ett större BASIC-program med kommandot MERGE.

Exempel:

På filen UTSUB. BAS finns ett objektprogram som skall användas av BASIC-programmet PROG. Objektprogrammet skall läggas in med början på rad 1000 i programmet PROG: (I programmet PROG har man lämnat plats för objektprogrammet). Det resulterande programmet skall kallas HELPROG. Tillvägagångssättet blir följande:

1. LOAD UTSUB. BAS — laddar objektprogrammet i minnet
2. REN 1000, 10 — numrerar om raderna så att de placeras på önskad plats i PROG.

3. MERGE PROG laddar in programmet PROG och kopplar ihop det med det omnumrerade UTSUB, BAS.

4. SAVE HELPROG sparar det färdiga programmet HELPROG.

Begränsningar - kassettversionen

För kassettversionen i 16K-minne gäller följande begränsningar:

1. Symboler får vara maximalt 6 tecken långa.

2. Det får finnas högst 50 symboler i ett program.

3. Det får sammanlagt finnas högst 5 st ORG eller DEFS -satser i ett program.

4. Objektprogrammet får maximalt bli 500 bytes långt.

5. Maximal radlängd är 78 tecken.

ATT SKAPA EN KÄLLFIL UTAN STÖD AV TEXTREDIGERINGSPROGRAM

Det finns en möjlighet att skapa källfiler direkt med ABC 80, trots att ABC 80 bara tillåter formellt riktiga BASIC-rader. Man får då lägga assembler-raderna som text-data i en datafil. Dessa text-data kan läsas in på filen med direkta kommandon, men det är oftast lämpligare att först göra ett program som sedan läser in text-data på filen när programmet körs. Då är det betydligt lättare att redigera texten om man upptäcker, att man skrivit fel. För inläsning på filen bör man använda den korta versionen av PRINT - alltså semikolon (;) - och den korta versionen av citationstecken - alltså apostrof ('). Annars skapas datafilen på samma sätt som i BASIC.

Exempel: Låt oss skapa en källfil med assembler-programmet

```
          ORG 65408 ; Kort prov
          LD HL,999
          JP LÄGE
STOPP:    RET
LÄGE:     JP STOPP
          END
```

Vi gör då följande program:

```
10 PREPARE "CAS:KP" ASFILE 1
20 ;#1'       ORG 65408 ; Kort prov'
30 ;#1'       LD HL,999'
49 ;#1'       JP LÄGE'
50 ;#1'STOPP:RET'
60 ;#1'LÄGE: JP STOPP'
70 ;#1'       END'
80 CLOSE 1
90 END
```

Sätt i den kassett som källfilen skall lagras på och sätt bandspelaren på inspelning. Kör sedan programmet genom att skriva RUN och trycka på RETURN. Källfilen bör lagras två gånger efter varandra, så skriv RUN och tryck på RETURN en gång till. Källprogrammet är nu lagrat på en fil med namnet KP. Källprogrammet kan assembleras precis på samma sätt som om det hade skrivits med hjälp av textredigeringsprogrammet EDITOR. Se tillvägagångssättet i exemplet på sid 29, fr o m "Assemblering". (Körs programmet skall det ge resultatet 999).

EDITOR - ETT TEXTREDIGERINGSPROGRAM FÖR ABC 80

I ABC 80 finns goda möjligheter att skriva och redigera BASIC-program med hjälp av kommandona ED, LIST, LOAD och SAVE. ABC 80 kontrollerar då hela tiden att det som skrivs är tillåtna BASIC-rader. Detta gör att kommandona inte kan användas för att redigera annan text, t ex brev eller assembler-program.

EDITOR är ett program, som kan användas för att redigera all slags text. Texten kan sedan sparas som en datafil, antingen på kassett eller på flexskiva. Dessutom kan texten skrivas ut på skrivare.

BRUKSANVISNING - EDITOR

Att köra programmet

1. Sätt i kassetten, spola tillbaka bandet till början och tryck på PLAY på bandspelaren.
2. Skriv RUN EDITOR (eller RUN CAS:) och tryck på RETURN (Ligger programmet EDITOR lagrat på flexskiva skriver man bara RUN EDITOR och trycker på RETURN).

När texten EDIT visas på bildskärmen kan man börja skriva in rader.

Att mata in rader

Texten skall matas in radvis som ett BASIC-program. Varje rad måste ha ett nummer, som skall stå främst på raden. Det måste även finnas ett mellanslag mellan radnumret och själva texten (om det skulle saknas sätter programmet dit det). Observera att EDITOR inte översätter små bokstäver till stora och inte ändrar antalet mellanslag (blanktecken) mellan orden. Det kan därför stå vad som helst på raderna.

Tangenterna ← och CTRL-X har samma funktion som vanligt, dvs ← tar bort det senast skrivna tecknet och CTRL-X tar bort hela raden.

Automatisk radnumrering

Kommandot AUTO n
ger automatisk numrering av raderna. Efter varje inmatad rad skrivs ett nytt radnummer ut som är lika med numret på den sista raden + n. Numreringen avbryts genom att man enbart trycker på RETURN, dvs en rad utan någon text.

Exempel: (Det som är understruket är skrivet av programmet).

```
200 Precis som jag trodde. Jag hoppas    RETURN
AUTO 20                                  RETURN
220 verkligen att han inte fortsätter    RETURN
240 på det här sättet. Det kan aldrig    RETURN
260 vara bra att jämt hålla på med       RETURN
280                                      RETURN
```

Att ta bort rader

En rad kan raderas genom att man skriver radnumret följt av RETURN. Vill man däremot ta bort flera rader är det bekvämare att använda kommandot DEL (Delete/ta bort).

DEL n1-n2

tar bort alla rader från och med rad nummer n1 t o m rad nummer n2. Det är också tillåtet att skriva DEL -n2, varvid alla rader t o m rad nummer n2 tas bort, DEL n1- , varvid alla rader från och med rad nummer n1 tas bort, eller DEL n, varvid rad nummer n tas bort.

Att redigera rader

En rad kan redigeras med kommandot

ED n , där n är radnumret

Detta går till på samma sätt som vid redigering med ABC 80´s redigeringsfunktioner (se ABC 80´s bruksanvisning).

Att skriva ut texten på bildskärmen

Det finns två kommandon för att skriva ut texten på bildskärmen. Kommandot LIST skriver ut hela raderna med radnummer, och kommandot DIS skriver ut raderna utan radnummer. Båda kommandona fungerar som ABC 80´s LIST-kommando (se ABC 80´s bruksanvisning).

Att skriva ut texten på skrivare

Ingår en skrivare i systemet kan texten dessutom skrivas ut på skrivaren med kommandona PLIST (med radnummer) och PDIS (utan radnummer). Se sid 34.

Att numrera om raderna

Raderna i texten kan numreras om med kommandot REN. Detta fungerar på samma sätt som ABC 80´s REN-kommando (se ABC 80´s bruksanvisning).

Att spara text

Texten som finns i minnet kan sparas på kassett eller flex-skiva med kommandot:

SAVE filnamn.filtyp

Texten kommer då att sparas rad för rad (med radnummer) på filen filnamn. (Se även "Att spara assembler-program").

Att hämta in text

Text som sparats med SAVE kan hämtas in igen med kommandot:

LOAD filnamn.filtyp (alternativt LOAD CAS:)

Det går givetvis också att läsa in textfiler som skapats på annat sätt, t ex BASIC-program som sparats med LIST filnamn. (Se även "Att hämta in text som lagrats utan radnummer").

Kommandot:

MERGE filnamn.filtyp

hämtar in texten från filen filnamn utan att den gamla texten tas bort. Om det finns en rad med samma nummer både på filen och i minnet kommer raden från filen att användas.

OBS: Filer, som skapats på annat sätt än med programmet EDITOR, kan inte hämtas in (till EDITOR) om de lagrats med kommandot SAVE.

Att börja om från början

Kommandona NEW eller SCR tar bort alla inmatade rader.

Att avsluta körningen

En editering avslutas med kommandot BYE, varvid programmet frågar

Har du sparat filen (JA/NEJ)?

Det är inte möjligt att lämna programmet EDITOR förrän man svarat JA på den frågan. (Man kan naturligtvis alltid avbryta körningen med överordnade ABC 80-kommandon, som t ex CTRL-C).

Att spara assembler-program

Assembler-program som skall behandlas av Editorn skall sparas utan radnummer och helst två gånger efter varandra. Detta görs automatiskt med kommandot

CSAVE filnamn

som sparar programmet på kassett precis som assemblatorn vill ha det.

Skall programmet sparas på flexskiva används kommandot:

DSAVE filnamn

som också sparar programmet utan radnummer.

Att hämta in en text som lagrats utan radnummer

En text, eller ett assembler-program, som lagrats utan radnummer kan hämtas in med kommandot:

CLOAD filnamn

som medför att radnummer läggs till raderna. Raderna numreras då 10, 20, 30, ...osv..

Observera: Om programmet EDITOR används för att redigera program som skall assembleras med ABC 80-assemblatorn, får raderna vara maximalt 78 tecken långa (exkl radnummer).

KOMMANDOLISTA - EDITOR

AUTO n	Automatisk radnumrering med intervallet n.
DEL (n1 - n2)	Tar bort rader.
ED n	För redigering av en rad.
LIST (n1 - n2)	Skriver ut texten på bildskärmen med radnummer.
DIS (n1 - n2)	Skriver ut texten på bildskärmen utan radnummer.
PLIST (n1 - n2)	Skriver ut texten på skrivaren med radnummer.
PDIS (n1 - n2)	Skriver ut texten på skrivaren utan radnummer. Märk! (n1 - n2) i ovanstående kommandon kan betyda: n1, n1-, -n1, n1-n2.
REN n,m	Numrerar om raderna med intervall n med början från rad m.
SAVE filnamn.filtyp	Sparar texten på en fil med radnummer.
LOAD filnamn.filtyp	Hämtar in text från en fil.
MERGE filnamn.filtyp	Hämtar in text från en fil utan att ta bort den gamla texten.
NEW	Tar bort alla rader.
SCR	"-
BYE	Avslutar körningen

Specialkommandon

CSAVE filnamn	Sparar en text (ett program) utan radnummer två gånger efter varandra på kassett.
DSAVE filnamn	Sparar en text (ett program) utan radnummer på flexskiva.
CLOAD filnamn	Hämtar in en text utan radnummer från kassett eller flexskiva och lägger automatiskt till radnummer.

Specifikationer:

EDITOR´s storlek:	ca 9000 bytes
Maximal radlängd:	78 tecken (exkl radnummer)
Maximalt antal rader:	300 (se nedan)
Tillåtna radnummer:	1 - 32000

Den totala mängden text som ryms i minnet beror på hur mycket minne som finns och på om det finns flexskiveenhet ansluten. I en "naken" ABC 80 (16K byte minne, ingen flexskiveenhet)ryms ca 4500 tecken vilket motsvarar en maskinskriven A-4 sida (fylld).

EXEMPEL PÅ ANVÄNDNING AV ABC 80-ASSEMBLATORN (kassett-versionen)

I detta exempel skall vi visa ett kort program som får bildskärmen att blinka. Observera att det finns två metoder för att skapa en källfil (se sid 19).

Vi börjar med att skapa ett källprogram med hjälp av programmet EDITOR. Programmet skall se ut så här:

```
            ORG 65408
            LD HL,7C00H
            LD BC,400H
BR:         SET 7,(HL)
            CPI
            RET PO
            JR BR
```

Sätt nu i kassetten med programmet EDITOR och spola tillbaka bandet till början. Tryck sedan på PLAY. (I exemplet nedan skriver ABC 80 ut allt som är understuket).

```
ABC 80
RUN EDITOR (eller RUN CAS:)                                    RETURN
Found EDITOR.BAC
(Om resultatet blir ERR 21 - gör om samma sak igen)

EDIT

AUTO 10                                                        RETURN
10        ORG   65408 ; Programmet BLINK                       RETURN
20        LD    HL,7C00H                                       RETURN
30        LD    BC,400H                                        RETURN
40 BR:    SET   7,(HL)                                         RETURN
50        CPI                                                  RETURN
60        RET   PO                                             RETURN
70        JR    BR                                             RETURN
80        END                                                  RETURN
90                                                             RETURN
```

Vi skall nu spara källprogrammet på en fil på kassett. Sätt därför in en kassett i bandspelaren och ställ in den för inspelning. Sedan ger vi kommandot:

CSAVE BLINK RETURN

EDITOR

Vi har nu lagrat källprogrammet och kallat det för BLINK. Vi är nu klara med editeringen och skriver därför:

BYE RETURN

Har du sparat filen? (JA/NEJ)? JA RETURN

Assemblering

Sätt nu i kassetten med assemblatorn och spola tillbaka bandet till strax innan början av programmet ASMCAS: Tryck därefter ned PLAY på bandspelaren och skriv:

RUN ASMCAS (eller RUN CAS:) RETURN

Found ASMCAS.BAC
(Om resultatet blir ERR 21 - gör om samma sak igen)

Efter en liten stund har assemblatorn laddats in i minnet och följande visas på bildskärmen: (som tidigare innebär understrykningar att ABC 80 skrivit)

ABC 80 assembler

Vad skall listas på skärmen

H = hela programmet

F = endast felaktiga rader

? H RETURN

Tryck på RETURN när bandspelaren är klar

För PASS 1

Innan vi kan trycka på RETURN måste kassetten med källfilen sättas in i bandspelaren. Glöm ej att spola tillbaka bandet så att det står vid början av källfilen. Tryck sedan ned PLAY på bandspelaren. Därefter kan vi trycka ned RETURN:

RETURN

(listning på skärmen)

Om fel upptäcks under pass 1 utförs inte pass 2. I så fall är det bara att börja om igen. (Detta skall inte hända här, utan vårt program skall vara felfritt - om det matats in rätt).
OBS: De felutskrifter som fås av assemblatorn hänvisar till appendix B.

Efter att pass 1 utförts skrivs följande ut på bildskärmen:

Tryck på RETURN när bandspelaren är klar

För PASS 2

Är källfilen inte lagrad två gånger efter varandra måste bandet spolas tillbaka igen innan vi kan trycka på RETURN. Har källfilen lagrats med kommandot CSAVE av programmet EDITOR är det bara att trycka på RETURN.

RETURN

Objektkoden på band?(J/N)?J RETURN

Filnamn

? BLINK.BAS RETURN

Tryck på RETURN när bansspelaren är klar för inspelning

Innan vi kan trycka på RETURN måste vi lägga i den kassett som vi vill att objektfilen skall sparas på, och ställa bandspelaren för inspelning. Sedan kan vi trycka på RETURN. Alltså:

RETURN

Ny assemblering (J/N)?N RETURN

ABC 80

Nu är objektprogrammet lagrat på en fil med namnet BLINK.BAS. Vi har då fått ett färdigt maskinspråks-program som är lagrat i form av POKE-satser. Låt oss även pröva programmet.

Att köra ett assembler-program

Sätt i kassetten med objektprogrammet och spola tillbaka bandet till början av objektfilen. Tryck därefter på PLAY på bandspelaren och skriv:

LOAD BLINK.BAS (eller LOAD CAS:) RETURN

Found BLINK.BAS
(Om resultatet blir ERR 21 - gör om samma sak igen)

ABC 80

Nu är vårt objektprogram laddat i minnet i form av en POKE-sats till 65408. Vårt program har alltså fått formen av ett mycket kort BASIC-program med en rad - en POKE-sats. Genom att köra

detta program kommer maskininstruktionerna att laddas in i minnet med början vid adress 65408. För att göra detta skriver vi:

RUN RETURN

ABC 80

För att starta programmet anropar vi det med en CALL-instruktion till adress 65408 (startadressen till vårt program). Vi använder här CALL-instruktionen som en direkt funktion, men den kan naturligtvis även ingå som instruktion i ett program.

; CALL (65408) RETURN

och se, bildskärmen blinkar!

Tillvägagångssättet blir alltså följande:

1. Skapa ett källprogram och lagra detta utan radnummer på en fil på kassett två gånger efter varandra. Lättast är att använda programmet EDITOR.
2. Sätt in kassetten med assemblatorn.
3. RUN ASMCAS, eller RUN CAS:
4. Sätt i kassetten med källfilen (tillbakaspolad).
5. Tryck på RETURN för pass 1.
6. Om inga fel upptäcks - tryck på RETURN för pass 2 (förutsatt att källfilen lagrats två gånger efter varandra).
7. Sätt i kassetten som objektprogrammet skall sparas på och sätt bandspelaren på inspelning.
8. Spela in objektprogrammet.
9. Klart.

Observera att pass 2 inte utförts om fel upptäcks under pass 1.

HEXMON

Hexmon är ett program med vars hjälp man kan läsa och skriva direkt i RAM-minnet, anropa maskinspråksprogram samt lägga in en brytpunkt och se på CPU-registren efteråt. Hexmon är skrivet i BASIC, men har en egen assembler-rutin i CASBUF 1. Användarprogrammet får CASBUF 2 som stack-area. Hexmon är framför allt avsett för felsökning av maskinspråksprogram, men kan givetvis även användas för andra ändamål, där man behöver läsa eller skriva direkt i RAM-minnet.

Följande kommandon finns tillgängliga: Hjälp, Dumpa, Ladda, Jump, Break, Register. Endast första bokstaven behöver anges. Varje kommando skall avslutas med RETURN.

Dumpa:

```
D FF80, 10  /Dumpa 10 (decimalt) bytes fr o m adress FF (hex)/
```

Ladda:

```
L FF80   /Ladda minnet med början vid adress FF80 (hex) med
efterföljande hex-värden/ T ex:

         L FF80  RET
         FF80: AA BB CC DD EE FF  RET
         FF 86  RET
```

Break:

```
B FF84   /Sätt brytpunkt vid adress FF84 (hex)/
```

Jump:

```
J FF80 /Call(FF80H), dvs hoppa till maskinspråksprogrammet vid
FF80/
```

Break:

```
B?       /Var finns brytpunkt/
B        /Ta bort brytpunkt/
```

Register:

```
R        /Skriv ut innehållet i CPU-registren/
```

Hjälp:

```
H        /Skriv ut hjälpmeny med tillåtna kommandon/
```

När minnet dumpas erhålls dels en utskrift av innehållet i varje Byte i hex-kod och dels en uppställning av ASCII-betydelsen av denna kod. T ô m minnesardress betecknas med fylld ruta och otryckbara ASCII-tecken betecknas med punkt.

Vill man använda Hexmon för att felsöka ett maskinspråksprogram går man tillväga på följande sätt:

1. Modifiera maskinspråks-programmet genom att byta ut returhoppet till BASIC, dvs instruktionen RET, till CALL FC3E (CD 3E FC hex; 205, 62252 decimalt). Detta för att Hexmon skall återställa CPU-registren innan man kommer tillbaka till BASIC.

2. Är maskinspråks-programmet gjort så att ett värde skall föras över till DE-registret tillsammans med CALL-instruktionen, måste man lägga till en instruktion i början av programmet, som istället laddar DE-registret från någon lämplig minnesadress. Hexmon kan nämligen inte ladda CPU-registren men däremot kan önskad minnesadress laddas med valfritt värde.

3. Ladda in maskinspråks-programmet och ladda därefter in Hexmon.

4. Kör därefter Hexmon och lägg in brytpunkter som önskas. Vid varje brytpunkt sparas CPU-registren, så att användarens värden finns kvar om man vill fortsätta. I så fall är det bara att ta bort eller flytta brytpunkten och göra Jump till värdet på PC-registret.

5. Hexmon-körningen avslutas med CTRL-C.

APPENDIX A

FLEXSKIVE-VERSIONEN

Skillnaden mellan kassett-versionen och flexskive-versionen av ABC 80 ASSEMBLER

Det finns i huvudsak tre skillnader mellan de två versionerna.

1. I flexskive-versionen skrivs objektprogrammet ut direkt på en fil, varför programmet kan vara i stort sett obegränsat stora. I kassett-versionen bevaras objektprogrammet i RAM-minnet tills assembleringen är klar, varför programmets storlek begränsas till ca 300 rader (max 500 bytes).

2. I flexskive-versionen kan man välja mellan att få programlistan på bildskärmen, utskriven på en fil eller utlistad på skrivare. I kassett-versionen fås programlistan enbart på bildskärmen. Detta gäller assembleringsprogrammet.

3. Hanteringen av flexskive-versionen är något annorlunda, vilket beskrivs i det följande.

Handhavande, flexskive-versionen

Flexskive-versionen av ABC 80-assemblatorn består av två delar, dels själva programmet, ASM, och dels en datafil, ASMCON.

När ett källprogram, som är lagrat på en fil, skall assembleras, blir tillvägagångssättet följande:

1. Ladda in assemblatorn till ABC 80´s minne. Skriv RUN ASM och tryck på RETURN, varefter först programmet ASM läses in, vilket i sin tur omedelbart läser in filen ASMCON. När inläsningen är klar kommer följande frågor att visas på bildskärmen efter varandra:

ABC 80-assembler

Infil? Ange namnet (även filtyp) på den fil som ska assembleras

Listfil? Ex. PR: för printer, enbart RETURN för skärmen. Om något annat anges kommer listan att läggas upp som en fil på flexskivan.

Objektfil? Ange det filnamn (även filtyp) du vill lagra objektprogrammet under.

Vad skall listas

H = hela programmet

F = endast felaktiga rader

?

Du har tidigare angett Listfil. Vad skall listas på listfilen? H för hela programmet, F för endast felaktiga rader.

Efter att ovanstående frågor besvarats i tur och ordning utförs assembleringen. Först utförs Pass 1, vilket omedelbart följs av Pass 2 om inga fel upptäcks under Pass 1.

OBS: De felutskrifter som fås av assemblatorn hänvisas till appendix B.

Objektprogrammet är nu lagrat på en fil i form av en eller flera POKE-satser med början vid den önskade minnesadressen (assemblerprogrammets startadress). Vill man köra objektprogrammet förfar man på samma sätt som beskrivs för kassett-versionen (naturligtvis utan att använda kommandot LOAD CAS: utan istället LOAD filnamn).

ATT SKAPA EN KÄLLFIL

Lättast skapar man en källfil med hjälp av textredigerings-programmet EDITOR (se sid 20).

En källfil kan även skapas utan stöd av textredigeringsprogram enligt den metod som beskrivs på sid 19 med följande skillnader:

Dels ändras enheten CAS: i rad 10 i programmet till DR0: eller DR1: , dels är det inte nödvändigt att lagra filen två gånger.

EXEMPEL PÅ ANVÄNDNING AV ABC 80-ASSEMBLATORN (flexskive-versionen)

Vi använder samma exempel som för kassett-versionen (se sid 27) men med några skillnader.

Efter att källprogrammet skapats med hjälp av programmet EDITOR på det sätt som beskrivs i exemplet och därefter lagrats med kommandot DSAVE hoppar vi direkt till assembleringen.

Assemblering

Vi antar nu att källprogrammet är lagrat på en fil på flexskiva under namnet BLINK. Arbetsgången blir då följande:

(Understrykningar innebär även här att ABC 80 skrivit).

RUN ASM RETURN

ABC 80 ASSEMBLER (DISK 790924)

ABC 80-assembler

Infil? BLINK RETURN

Listfil (RETURN = på skärmen)? LISTA RETURN

Objektfil? BLINK.BAS RETURN

Vad skall listas

H = hela programmet

F = endast felaktiga rader

?H RETURN

(Hade ingen listfil angivits, hade först en programlista efter pass 1 listats på skärmen omedelbart följt av en programlista efter pass 2 - förutsatt att inga fel upptäckts under pass 1).

OBS: De felutskrifter som fås av assemblatorn hänvisar till appendix B.

PASS 1 klart 0 fel

Assembleringen klar 0 fel

ABC 80

Därmed är assembleringen klar och resultatet har blivit ett objekt program lagrat på en fil med namnet BLINK.BAS samt en programlista lagrad på en datafil med namnet LISTA.

Vill man köra assembler-programmet förfar man som i exemplet, dvs:

LOAD BLINK.BAS RETURN

ABC 80

Nu är objektprogrammet laddat i minnet i form av en POKE-sats till 65408. Vårt program har resulterat i ett BASIC-program med bara en rad som, när det körs laddar in maskininstruktionerna i minnet med början vid adress 65408. Efter att detta gjorts kan vi starta programmet genom en CALL-instruktion till adress 65408. Alltså:

; CALL(65408) RETURN

varvid bildskärmen blinkar!

Listning av programlistan

Vill man se hur programlistan ser ut använder man lättast följande program:

```
 5  ONERRORGOTO 110
10 ; "DATAFIL";
20 INPUT L¤
30 OPEN L  ASFILE 1
40 INPUTLINE  1, A¤
50 A¤=LEFT  (A¤, LEN(A¤)-2)
60 ; A¤
70 GET X¤
80 IFX¤<>"S" THEN 40
90 CLOSE 1
100 END
110 ; "SLUT PA FILEN":GOTO 90
```

Kör man programmet händer följande:

RUN RETURN

DATAFIL? LISTA RETURN

(Först visas första raden i filen LISTA, därefter fås de följande genom att valfri tangent (utom S) trycks ned. Programlistningen stoppas genom att S trycks ned, annars fortsätter listningen tills filen tar slut.

SAMMANFATTNING

Tillvägagångssättet blir alltså följande:

1. Skapa ett källprogram och lagra detta som en datafil. Lättast är att använda programmet EDITOR.
2. RUN ASM och ange infil, listfil och objektfil efter programmet anvisningar.
3. Objektprogrammet är nu lagrat på objektfilen som ett antal POK satser i en eller flera BASIC-rader och kan laddas in i minnet på samma sätt som ett vanligt BASIC-program. Eventuell program lista är lagrad som en datafil på listfilen.

Begränsningar - flexskiveversionen

1. Symboler får vara maximalt 6 tecken långa.

2. Det får finnas högst 50 symboler i ett program.

3. Det får sammanlagt finnas högst 5 st ORG eller DEFS-satser i ett program.

4. Maximal radlängd är 78 tecken.

APPENDIX B

FELMEDDELANDEN OCH FÖRKLARINGAR

2) NAMN MED OTILLÅTNA TECKEN

Indikerar att ett namn (läge eller operand) innehåller otillåtna tecken.

3) OTILLATEN OP-KOD

Erhålls om op-koden innehåller otillåtna tecken.

4) OTILLATET TAL

Erhålls om ett tal innehåller otillåtna tecken inom specificerad talbas.

5) OTILLATEN OPERATOR

6) SYNTAX FEL

Erhålls om ett uttryck har fel format eller paranteser saknas.

7) ASSEMBLER FEL

Instruktionen har ej kunnat utföras.
Detta kan bero på att raden är felaktig.

8) OKÄND SYMBOL

En symbol i operandfältet har ej definierats. Erhålls för felstavade eller ej definierade lägesnamn.

9) FELAKTIG KOMBINATION AV OPERANDER

Erhålls om ett registernamn eller villkorskod är felstavad eller felaktigt använd.

10) UTTRYCK EJ INOM INTERVALL

Indikerar att värdet för ett uttryck är för stort ellet litet. Erhålls t ex vid overflow för 16-bits aritmetik, division med 0 eller felaktigt värde för en byte.

11) DUBBLERAD DEKLARATION

Indikerar försök till omdefiniering av ett lägesnamn. Erhålls om en variabel är felstavad eller felaktigt använd vid flera tillfällen.

13) CITATTECKEN OBALANSERADE

Indikerar att en sträng ej är korrekt omgiven av citattecken eller att citattecken inom en sträng ej är korrekt grupperade i par.

För ytterligare information hänvisas till sid 280 i Assembly Language Manual.