LUXOR

Dator ABC 800

Manual BASIC II

ABC 800®

LUXOR Datorer

Förord

Denna manual beskriver programspråket BASIC II för ABC 800. Manualen är inte avsedd som lärobok i BASIC, utan läsaren förutsätts ha erfarenhet av programmering sedan tidigare.

Kapitel 1 presenterar programspråket BASIC. I kapitel 2 behandlas uppbyggnaden av datorprogram i BASIC II.

Kapitlen 3, 4 och 5 beskriver de data, som kan hanteras i ett program.

I kapitel 6 talas om det praktiska arbetet med BASIC II. Detta kapitel innehåller många råd och tips om hur man skriver och ändrar ett program.

Kapitel 7 beskriver, hur datorns instruktioner och kommandon används direkt, utan att bilda något program. Detta är särskilt användbart vid felsökning.

Kapitlen 8, 9 och 10 innehåller utförliga beskrivningr av de kommandon, funktioner och instruktioner, som ingår i BASIC II. De flesta beskrivningar är kompletterade med exempel, som visar hur varje programdel byggs upp.

Kapitlen 11 och 12 behandlar grafiken i ABC 800. Där beskrivs TELETEXT-grafik och högupplösningsgrafik med animation-mod.

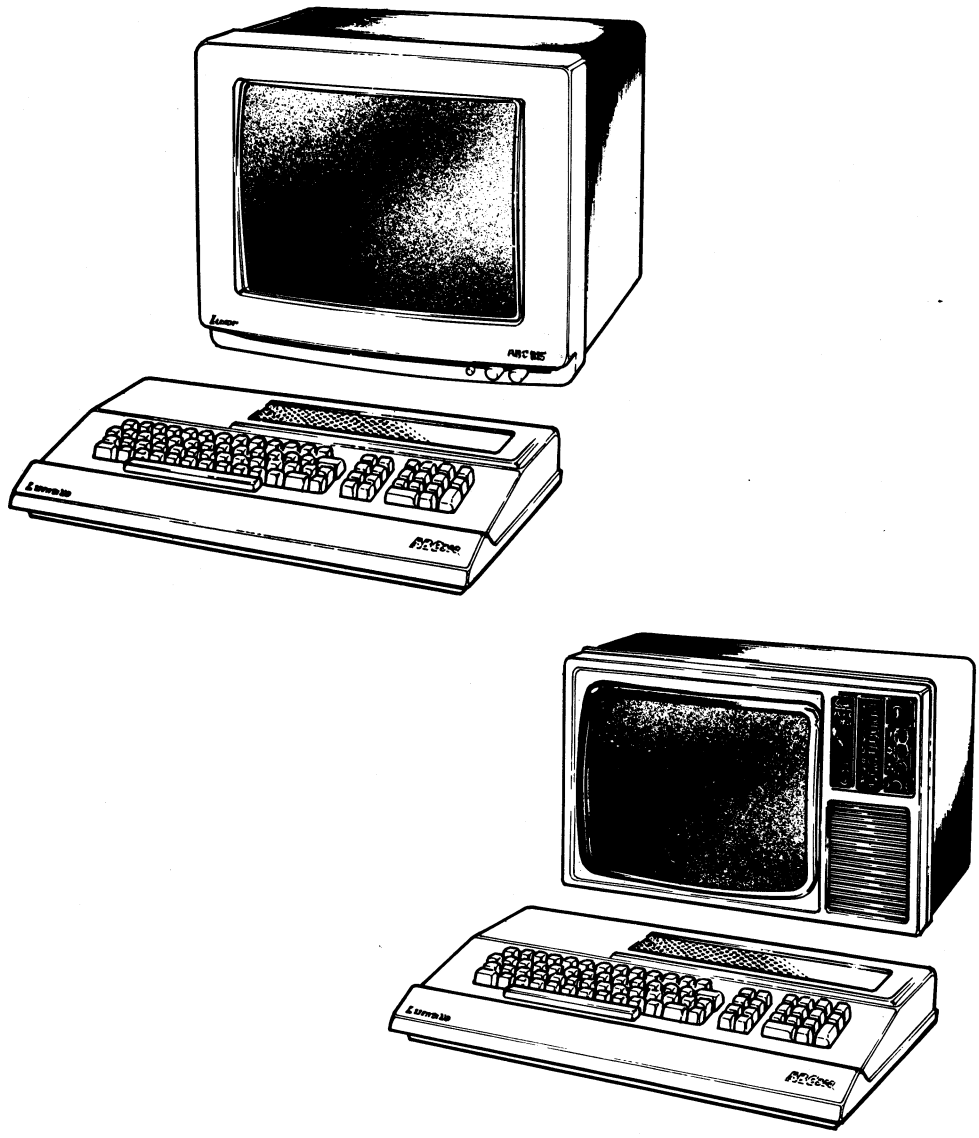
Kapitel 13 talar om hur man använder datorns funktionstangenter.

Kapitel 14 beskriver skillnaderna mellan ABC 800 och ABC 80.

Kapitel 15 innehåller en lista med felmeddelanden och kommentarer till dessa.

Kapitel 16, som är markerat med grått i kanten, innehåller kortfattade beskrivningar av instruktionerna, funktionerna och kommandona, samlade i bokstavsordning. Listan i kapitel 16 är avsedd som ett uppslagsregister, som ger syntax och sidhänvisningar till de mera omfattande beskrivningarna tidigare i manualen (kapitel 8, 9 och 10).

Kapitel 17 är en litterturförteckning och kapitel 18 innehåller ett antal bilagor. Manualen avslutas med ett sakregister.



Innehåll

1 Om programspråket BASIC

För att tala om för datorn, vad den ska göra, använder vi ett formellt språk eller programmeringsspråk. Det formella språket är uppbyggt av vissa nyckelord på engelska.
BASIC är ett av de allra enklaste programspråk, som finns. Alla instruktioner, kommandon och funktioner är lätta att förstå och använda. Trots detta är språket tillräckligt omfattande för att ge en smidig och effektiv problemlösning.

Namnet BASIC kommer av Beginner's All-purpose Symbolic Instruction Code. Programspråket BASIC var från början avsett enbart för elementär undervisning i programmering. Det visade sig dock fungera så bra, att det numera används i många tillämpningar.

Många programspråk bygger på principen att hela programmet först matas in i datorn och därefter översätts till maskinspråk (kompileras). Vid kompileringen undersöks, om programmet innehåller formella fel. Datorn skriver då ut en fellista. Programmeraren rättar programmet och matar in det på nytt. Det kompileras igen och eventuella fel skrivs ut.

När man arbetar med BASIC, däremot, finns ett program, som kallas interpretator (BASIC-tolk), i datorn. För varje programrad, som skrivs, kontrollerar BASIC-tolken om den är formellt korrekt. Formella fel ger en felutskrift direkt på skärmen, medan man matar in programmet. Man kan också när som helst under inmatningen provköra sitt program.Detta brukar kallas interaktiv programmering. I många fall är detta mycket effektivare än traditionell programmering med kompilering.

Den interaktiva programmeringen löser givetvis inte alla problem. När programmet är fritt från formella fel, återstår kanske logiska fel, som man bara kan upptäcka vid provkörning.

BASIC har grammatikregler, precis som alla andra språk. Grammatiken för ett programspråk är dock mycket enklare än motsvarande regler för ett naturligt språk. Nedanstående exempel beräknar medelvärdet av fem tal, som användaren matar in. Här kan Du se, hur språket är uppbyggt.

Exempel

```
10 INPUT A,B,C,D,E
20 LET S=(A+B+C+D+E)/5
30 PRINT S
40 END
```

BASIC II innehåller dels de elementära instruktioner, som behövs för att skriva enkla program, dels också instruktioner och funktioner, som gör det möjligt att skriva mera avancerade program med högre effektivitet. Nyckelordet för denna typ av programmering är effektivitet. Efterhand som användaren skaffar sig större programmeringserfarenhet, ökar effektiviteten och man vill använda alltmer avancerad databehandling. BASIC II är så omfattande, att man kan lösa praktiskt taget vilket problem som helst.

BASIC II ger möjlighet till AUTO START. AUTO START funktionen behandlas utförligt i bruksanvisningen för flexskiveenheten.

2 Program

Ett program består av programrader, som innehåller satser. Satserna innehåller instruktioner till BASIC-tolken. Varje programrad börjar med ett unikt radnummer. Efter radnumret skrivs en eller flera BASIC-satser. Mellan två satser på samma rad ska finnas ett kolon (:). Programraderna utförs i nummerordning. Varje sats inleds av ett nyckelord, som anger vilken operation, som ska utföras.

Satsen ger datorn en instruktion (i detta fall **PRINT**):

```
30 PRINT S
```

Vissa instruktioner kräver operander, som specificerar vilken variabel eller vilken programdel, instruktionen ska utföras på. I exemplet ovan är operanden "S".

Sista satsen i ett program är en **END**-sats:

```
10 INPUT A,B,C,D,E
20 LET S=(A+B+C+D+E)/5
30 PRINT S
40 END
```

Satsen **END**, som talar om för datorn, att programmet är slut, är inte obligatorisk, men bör vara den sista sats, som genomlöps i programmet. Den medför, att alla filer stängs, men variablerna behåller sina värden.

2.1 Radnummer

Varje programrad inleds med ett radnummer, som bl.a. fyller följande funktioner:

1. Anger i vilken ordning satserna genomlöps. Det är likgiltigt i vilken ordning de matas in.

2. Gör det möjligt, att ändra den normala genomloppsordningen med hjälp av t.ex. satser som **GOTO** eller **GOSUB**. Radnumret fungerar som lägesangivelse och hoppadress.

3. Gör det möjligt, att ändra vilken rad som helst, utan att resten av programmet påverkas.

Du, som programmerar, väljer radnummer. Det får vara ett godtyckligt heltal från **1** t.o.m. **65535**.

Varje rad ska ha ett, unikt radnummer. Datorn använder radnumren för att identifiera och hålla reda på instruktionerna. Om man skriver in en ny rad med ett radnummer, som redan finns, ersätts den existerande raden med den nyskrivna.

Satserna kan skrivas in i godtycklig ordning. Datorn tar hänsyn till nummerordningen. Om Du t.ex. skriver in raderna i följande ordning: 30, 10, 20, så sorterar datorn dem: 10, 20, 30.

Raderna bör numreras i jämna multipler av 10 eller 5, för att det ska vara lätt att föra in nya rader. Det finns kommandon för automatisk radnumrering (**AUTO**) och för omnumrering av raderna (**RENUMBER**).

2.2 Kommentarer

BASIC II har **två sätt** att beteckna kommentarer i programmet.

1. **REM**-satser (som i standard BASIC)

```
10 A=7: REM sju
```

2. Utropstecken. Behöver inte avgränsas med kolon.

```
10 A=7 ! sju
```

Kommentarer är en del av ett BASIC-program. De skrivs ut, när programmet listas på skärmen eller på skrivare. Kommentarsatserna exekveras inte, utan hoppas över då programmet körs. Vilka tecken som helst (utom RETURN) kan användas i kommentaren. För att göra det lättare att se kommentarerna, brukar man markera dem med något väl synligt tecken:

```
10 REM ********* MÄTDATA IN *****************
200 GOSUB 3100 ! ### TILL  DIFF-BERÄKNING ###
3240 RETURN ! &&&&& X7=DIFFERENSEN &&&&&
```

OBS!

En kommentar kan inte avslutas med kolon. Hela satsen, inklusive kolon, betraktas som en kommentar.

```
150 REM *** Beräkningsmetod *** : LET R1=3.52E2.1+Y5
```

Den avslutande satsen blir inte utförd. Hela raden behandlas som en kommentar.

2.3 BASIC-satser

Efter radnumret följer en BASIC-sats. Satsens nyckelord anger, vilken typ av sats det är. På detta sätt vet BASIC-tolken (det program, som översätter BASIC till datorinstruktioner) vad, som ska utföras och hur de eventuella data, som står efter nyckelordet, ska behandlas.

Användaren får lov att skriva flera BASIC-satser på samma programrad. Mellan två satser ska finnas ett kolon (:). En rad med flera satser exekveras något snabbare än samma satser på var sin rad, vilket kan vara väsentligt i vissa tillämpningar.

```
100 PRINT A,B,C
```

är en enkel programrad

```
200 LET X=X+1 : PRINT X : IF Y=1 THEN 100
```

är en sammansatt programrad med de tre satstyperna:

LET, PRINT och **IF-THEN**

Praktiskt taget alla instruktioner kan användas i sammansatta programrader. De undantag, som finns, är klart angivna i beskrivningarna av respektive instruktion.

OBS!
Enligt god programmeringssed bör man dock skriva endast en sats per rad.

2.4 Uttryck

Ett uttryck består av symboler, som representerar konstanter, variabler, funktioner eller någon kombination av dessa, åtskilda av aritmetiska, relations- eller logiska operatorer.
Exempel

Aritmetiska uttryck

4.123
3%+A%
B6*(C** 3+1.0)

Relationsuttryck

X > Y
Y8 >= 0
A = B

Logiska uttryck

(A < 1.) **AND** (B=5)
((B < A) **OR** (D=C)) **AND** B/A <> D/C

Aritmetiska uttrycks värde är antingen heltal eller flyttal.

Relationsuttryck har värdet sant eller falskt, då de båda ingående värdena jämförts.

Logiska uttryck har värdet sant eller falskt, beroende på om villkoren i uttrycket är uppfyllda eller inte.

Beträffande stränguttryck se kapitel 5.

2.5 Logiska enheter

BASIC II gör användaren oberoende av vilka in/ut-enheter, som faktiskt används. Den aktuella enheten tilldelas ett filnummer. Filnumret kan behandlas som en logisk enhet och hanteras med instruktionerna **OPEN, PREPARE** och **CLOSE.** Filnumret kan representera t.ex. skrivare eller en fil på kassett/flexskiva.

Exempel

```
10 - -
20 OPEN "PR:" AS FILE 2 !Öppna skrivaren
30 - -
40 - -
50 CLOSE 2 !Stäng skrivaren
60 END
```

Observera **CON:** (tangentbord/bildskärm) läggs upp som standardenhet .

2.6 Felhantering

Under körning av ett program kan BASIC upptäcka vissa typer av fel. Dessa fel kan t.ex. vara beräkningsfel (t.ex. division med 0) eller I/O-fel (t.ex. när en end-of-file läses som indata till en **INPUT**-sats). Huvudregeln är, att då ett fel upptäcks, avbryts körningen av programmet och ett felmeddelande skrivs ut.

I vissa tillämpningar kan det vara nödvändigt, att programmet fortsätter genomlöpas även efter ett fel. För att uppnå detta, kan användaren skriva en **ON ERROR GOTO** <radnr>-instruktion i programmet. Programmet hoppar då till ett underprogram, som börjar på den angivna raden. Underprogrammet kan t.ex. innehålla en felhanterare, som analyserar det fel, som inträffar då programmet körs.

ON ERROR GOTO-satsen placeras i programmet före alla exekverbara satser, som ska behandlas av underprogrammet.

När ett fel inträffar i ett program, undersöks om någon **ON ERROR GOTO** <radnr>-sats har genomlöpts. Om detta inte har skett, skrivs ett felmeddelande ut på skärmen och programmet avbryts. Om en **ON ERROR GOTO** <radnr>-sats har genomlöpts innan felet, sker uthopp till det angivna radnumret, där användarens felhanteringsrutin t.ex. kan undersöka funktionen **ERRCODE**s värde, för att få veta, vilket fel det rör sig om och behandla det.

I vissa delar av programmet ska kanske alla fel hanteras av systemet istället för av användarens felrutin. En sådan del av programmet ska inledas med satsen

radnummer **ON ERROR GOTO**

Datorn tar då hand om alla fel.

Felhanteringen avslutas med **RESUME. RESUME**-satsen fungerar ungefär som **RETURN** i slutet av ett vanligt underprogram. Återhopp sker till uthoppspunkten (om det finns någon) i den sats, som orsakade feluthoppet. Om man måste hoppa in i någon annan rad i programmet, för att kunna fortsätta, anges radnumret i **RESUME**-satsen.

Exempel på felhantering

```
10 ON ERROR GOTO 100 !Vid felaktig inmatning hoppa till rad 100
20 INPUT "Ålder, Vikt "A,W
30 ON ERROR GOTO !Stäng av felhanteraren
40 STOP
100 PRINT !Felhanterare
110 PRINT "Felaktig inmatning!"
120 RESUME !Återhopp till rad 20
```

3 Data

3.1 Talområden

Flyttal

Talområdet för flyttal är det största talområdet i BASIC och omfattar:
±1.7E-37 - ±1.7E+38

I enkel precision (**SINGLE**) finns **sju signifikanta siffror** och i dubbel precision (**DOUBLE**) **16 signifikanta siffror**. Talen avrundas internt, så att de anges med aktuell precision.

Talen kan matas in och skrivas ut på tre olika format:

Exempel 153, 34.53, 134E-2

Heltal

Talområdet för heltal är **-32768 - +32767**

Strängar

En teckensträng kan innehålla hur många tecken som helst.

OBS!
Strängar, som används **vid strängaritmetik, får** maximalt innehålla **125 tecken** inklusive + eller - tecken och decimalpunkt.

3.2 Konstanter

Numeriska konstanter behåller sitt konstanta värde genom hela programmet. De kan vara positiva eller negativa. Numeriska konstanter kan skrivas enligt följande:

Exempel
+3%
-4.765
1234.6
-.0001

De tre sista konstanterna i exemplet lagras som flyttal, eftersom de inte följs av något %-suffix. Man bör alltid ange antingen %-suffix eller decimalpunkt, för att undvika onödiga omvandlingar och för att göra dokumentationen mera lättläst.

3.3 Variabler

Variabler är data, vars värde kan ändras medan programmet körs. En variabel betecknas med ett specifikt variabelnamn.

Ett variabelnamn består av en bokstav eller en bokstav följd av en siffra. När man använder **EXTEND**, kan man använda långa variabelnamn (alfanumeriska tecken med en bokstav först).
Tillåtna tecken

A,B,C,.....,Ö
0,1,2,.....,9

Ett namn kan också ha prefixet **FN**, som betecknar en användarfunktion, suffixet %, som betecknar heltal, suffixet ¤, som betecknar en sträng eller ett indexsuffix, som består av index inom parentes.

Ett stränguttryck har ett värde, som består av en teckenföljd, där varje tecken upptar en byte. En sträng kan skrivas antingen som en teckenföljd inom citationstecken eller som en strängvariabel betecknad med ett variabelnamn följt av suffixet ¤.

Data av olika typ bör inte blandas i samma sats. Använd i första hand heltal, där det är möjligt. Heltal tar mindre minnesutrymme och behandlas snabbare i datorn.

Samma namn kan förekomma med olika prefix och suffix i ett och samma program. Det representerar då helt olika variabler. Flyttalsvariabeln A är inte samma sak som heltalsvariabeln A%. Namnet A kan användas på följande, skilda sätt:

A	flyttalsvariabeln A
A%	heltalsvariabeln A%
A¤	strängvariabeln A¤
A(d)	flyttalsvektorn A med dimensionen d
A%(d)	heltalsvektorn A med dimensionen d
A¤(d)	strängvektorn A¤ med dimensionen d
FNA	flyttalsfunktionen A
FNA%	heltalsfunktionen A%
FNA¤	strängfunktionen A¤

I **EXTEND**-mod kan man skriva enligt följande:

Antal	flyttalsvariabeln Antal
Antal%	heltalsvariabeln Antal%
Antal¤	strängvariabeln Antal¤
Antal(d)	flyttalsvektorn Antal med dimensionen d
Antal%(d)	heltalsvektorn Antal% med dimensionen d
Antal¤(d)	strängvektorn Antal¤ med dimensionen d
FNAntal	flyttalsfunktionen Antal
FNAntal%	heltalsfunktionen Antal%
FNAntal¤	strängfunktionen Antal¤

Variabler kan tilldelas värden med bl.a. instruktionerna **LET, INPUT** och **READ**. Innan programmet genomlöps, nollställs variablerna, om de inte är skyddade av en **COMMON**-deklaration. Det är inte nödvändigt att tilldela en variabel ett startvärde, annat än när den måste ha ett startvärde skilt från 0.

3.4 Indexerade variabler (vektorer) och DIM-satsen

Man får använda indexerade variabler (vektorer) förutom de enkla variablerna. Indexerade variabler ger programmeraren utökade möjligheter att hantera listor, tabeller, matriser eller liknande variabelgrupper. Variablerna får indexeras i hur många dimensioner som helst.

En indexerad variabel betecknas med ett namn, som består av ett godtyckligt valt, tillåtet variabelnamn följt av ett antal heltal inom parentes. En lista kan t.ex. skrivas A(I), där I går från 0 till 5:

A(0), A(1), A(2), A(3), A(4), A(5)
Programmeraren kan nu använda vart och ett av listans sex element. De kan anses utgöra en endimensionell, algebraisk vektor enligt nedan:
A(0)
A(1)
A(2)
A(3)
A(4)
A(5)

En tvådimensionell matris B(I,J) kan definieras på samma sätt. Man kan avbilda den så här:

B(0,0)	B(0,1)	B(0,2)	...	B(0,J)
B(1,0)	B(1,1)	B(1,2)	...	B(1,J)
B(2,0)	B(2,1)	B(2,2)	...	B(2,J)
B(3,0)	B(3,1)	B(3,2)	...	B(3,J)
.....				
B(I,0)	B(I,1)	B(I,2)	...	B(I,J)

Varje index i en vektorvariabel måste vara ett heltal. Om index är ett flyttal, avrundas det till ett heltal.

En **DIM**-sats används ofta för att ange maximalt index i en vektor. Om en vektorvariabel används utan att vara dimensionerad, antas varje index i den vara dimensionerat till maximala värdet 10. Begynnelsevärdet för varje index kan ändras med instruktionen **OPTION BASE.** Normalt är detta värde 0, men det kan ändras till 1, så att en standardvektor omfattar 10 element i varje dimension i stället för 11. Alla **DIM**-satser ska placeras i början av programmet.

3.5 Filhantering

BASIC II innehåller instruktioner för datalagring och åtkomst av data på kassett eller flexskiva.

En datafil består av en följd av data, som överförs mellan ett BASIC-program och en yttre I/O-enhet. Den yttre enheten kan t.ex. vara skrivare, kassett eller flexskiva. Med instruktionen **OPEN** anger man, vilka enheter, som är tillgängliga och deras referensnummer. Varje enhet har ett namn, som används för att identifiera den i systemet. Ex: Drivenhet 0 i flexskiveenheten betecknas DR0:.

Varje fil har ett unikt namn. Ex. ABC123.BAC betecknar en fil på flexskiva. Inom programmet anropas filen med ett filnummer. Filnumret anges i programmet med någon av instruktionerna **PREPARE** eller **OPEN.** Dessa instruktioner öppnar kanalen, d.v.s. upprättar förbindelsen. För att stänga en sådan dataöverföringskanal används instruktionen **CLOSE.** Instruktionerna **INPUT** och **PRINT** eller **GET** och **PUT** används för att utföra dataöverföringen.

Då en fil öppnas, skapas en buffertarea i systemet. All dataöverföring går via buffertutrymmen.

3.5.1 Att öppna en fil

En fil, som redan finns, öppnas med **OPEN.** Om filen är ny, öppnas den med **PREPARE.**

Exempel

```
10 OPEN "FIL1.FIL" AS FILE 1
```

öppnar den existerande filen med namnet FIL1.FIL för in- och utmatning med filnummer1.

3.5.2 Dataöverföring till/från en fil

Dataöverföring sker direkt mellan den interna kanalen och angiven strängvariabel eller värdet av angivet uttryck. All dataöverföring avser antingen en byte (bitgrupp) eller en sträng (tecknen före vagnretur).
Följande instruktioner kan användas:

INPUT # läser ett värde till en variabel eller sträng från filpekarens läge till vagnretur.

INPUT LINE # läser ett värde till en strängvariabel inklusive vagnretur (CR) och radframmatning (LF).

PRINT # skriver variabelns innehåll på filen

GET # COUNT läser en byte eller angivet antal bytes från filpekarens läge

PUT # skriver en post på filen

POSIT # flyttar filpekaren till önskat läge

Om inget filnummer anges i **GET**-instruktionen, läser den från tangentbordet. Om **COUNT** ej anges, läser **GET** en byte, d.v.s. ett tecken.

Exempel 20 **GET** #1,D2¤**COUNT** 6%

läser på filen med filnummer 1 från filpekarens läge sex tecken framåt. Dessa tecken läggs i strängen D2¤.

Instruktionen **POSIT** används för att positionera filpekaren på angiven position i filen. Antalet tecken räknas alltid från filens början (position 0). **POSIT** kan användas tillsammans med vilken som helst av de övriga filhanteringsinstruktionerna.

Exempel fil 1 innehåller ABCDEFGHIJK

```
I
I
50 POSIT #1,5
60 GET #1,A¤COUNT 3
70 PRINT A¤
80 END
RUN
FGH
```

Funktionen POSIT(<filnr>) läser filpekarens läge. I detta fall har **POSIT**(1) värdet 8, då exemplet ovan har genomlöpts. **POSIT** har flyttalsvärde, för att kunna omfatta långa filer.

VARNING! **POSIT** bör inte användas i samband med sekventiella filer, d.v.s. filer som hanteras med **PRINT** och **INPUT/INPUT LINE.** Om man vill använda **POSIT** på sekventiella filer, måste en **PRINT**-sats åtföljas av en **GET**-sats, annars skrivs filslutsmärke (EOF) där filpekaren står, då **POSIT** eller **CLOSE** används nästa gång. En dummy **GET** kan se ut så här:

40 **GET** #2,Q¤**COUNT** 0

Tag för vana att betrakta **PRINT** och **GET** som en sekvens av instruktioner, som följs åt då **POSIT** används på sekventiella filer.

3.5.3 Att stänga en fil

Dataöverföring på en fil avslutas inte korrekt, förrän filen stängs. Då överförs innehållet i buffertarean, och filen får ett filslut (EOF).

Filen kan stängas på två sätt:

CLOSE 2 Stänger filen med filnummer 2

CLOSE Stänger samtliga filer

4 Heltal och flyttal

Alla numeriska värden (variabler och konstanter), som används i ett BASIC-program, lagras normalt som flyttal. Om heltal används i ett program, kan man spara mycket minnesutrymme genom att deklarera dessa data som heltal. Aritmetiken är också snabbare för heltal än för flyttal. Heltalsdeklarationen för konstanter, variabler eller funktioner görs med ett procenttecken, % efter namnet.

Exempel A%, **FN**X%(Y), -8%, Z3%

Observera, att %-tecknet måste finnas med varje gång ett heltal ska genereras, annars antas talet vara ett flyttal. Variabeln H% kan aldrig vara samma sak som variabeln H.

Om ett tal ska upphöjas till en heltalsexponent, skall exponenten alltid tydligt anges som heltal.

Här ovan förutsätts att BASIC II arbetar i sitt normala läge, d.v.s. **FLOAT.** Med instruktionen **INTEGER** kan man få BASIC II att förutsätta, att alla tal är heltal, så länge inget annat anges.

4.1 Matematiska operationer

Om mer än en operation ska utföras i ett och samma uttryck, tillämpas nedanstående prioritetsordning mellan de olika operatorerna, där nr 1 har högst prioritet.

1. Beräkningar inom parentes utförs först. Resultatet av en sådan beräkning används sedan i de följande stegen. För kapslade parenteser:
   (A+(B * (C** 3)))
   gäller, att den innersta beräknas först.

2. I övrigt gäller följande prioritetsordning:
   a. Inbyggda eller användardefinierade funktioner
   b. Exponentiering (**)
   c. Multiplikation och division (* , /)
   d. Addition, subtraktion (+, -) och minustecken som negerande prefix.
   e. Relationsoperatorer (=, <>, >=, <=, <,>)
   f. **NOT**
   g. **AND**
   h. **OR och XOR**
   i. **IMP**
   j. **EQV**

   Så är t.ex. -A** B ett tillåtet uttryck och betyder det samma som -(A**B). Detta innebär, att -2**2 blir -4. Uttrycket A**-B är inte tillåtet. Det ska skrivas som A**(-B).

3. Bortsett från parentesuttryck utförs alla operationer med samma prioritet från vänster till höger, så som uttrycket har skrivits.

4.2 Heltalsaritmetik

Heltalsaritmetiken utförs modulo 2**16. Ett BASIC-heltal kan ligga mellan -32768 och +32767. Heltalsrepresentationen kan betraktas som en obruten cirkel, där -32768 ligger efter +32767.

Heltalsdivision innebär, att en eventuell rest trunkeras. Jämför dock funktionen **MOD**, som gör det möjligt att ta fram resten från divisionen.

Exempel 3%/4%=0% och 283%/100%=2%

En och samma beräkning kan innehålla både heltal och flyttal. Resultatet får då den form, som passar resultatvariabelns deklaration.

Exempel

```
LET B%=Z%+3/X
```

Resultatet avrundas, så att B% får ett heltalsvärde.

4.3 In- och utmatning av heltal och flyttal

In- och utmatning av heltal görs på samma sätt som för flyttal.

De tal, som kan skrivas med högst sju siffror (vid **SINGLE**) eller högst sexton siffror (vid **DOUBLE**) skrivs utan exponentialformat.

De flyttal, som har ett heltalsvärde, skrivs som heltal men lagras fortfarande som flyttal.

Om ett tal omfattar mer än sju resp. sexton siffror, skrivs det automatiskt ut på följande sätt:

[-].nE[-]m

där n är ett tal med upp till sju siffror och m är en exponent med högst två siffror.

Vid inmatning kan alla de format användas, som används vid utmatning. Om en heltalsvariabel tilldelas ett flyttalsvärde, avrundas värdet till heltal.

4.4 Användardefinierade funktioner

En funktion deklareras som heltalsfunktion, genom att ett procenttecken (%) sätts direkt efter funktionsnamnet.

Exempel

```
10 DEF FNV%(X%)=X%*(Z%+X%)
```

En flyttalsfunktion kan skrivas på följande sätt:

Exempel

```
10 DEF FNV(X%)=X%*(Z+X%)
```

4.5 Heltal som logiska variabler

Heltalsvariabler eller heltalsuttryck kan användas i **IF**-satser på samma sätt som logiska uttryck. Alla värden skilda från 0 tolkas då som "sant" och värdet 0 som "falskt". De logiska operatorerna (**AND, OR, NOT, XOR, IMP** och **EQV**) utför bitorienterade operationer på logiska data (eller heltalsdata).

OBS!
Normalt motsvaras värdet "sant" av heltalet -1 och "falskt" av heltalet 0. Detta gäller de logiska värden, som kommer från BASIC II.

4.6 Logiska operationer på heltalsdata

BASIC II gör det möjligt för användaren att kombinera heltalsvariabler och heltalsuttryck med logiska operatorer, som arbetar databit för databit.

Sanningtabellen nedan gäller för de logiska operatorerna. A är tillståndet hos en viss databit i ett heltal och B är tillståndet hos motsvarande databit i ett annat heltal.

A	B	A **AND** B	A **OR** B	A **XOR** B	A **EQV** B	A **IMP** B	**NOT** A
1	1	1	1	0	1	1	0
1	0	0	1	1	0	0	0
0	1	0	1	1	0	1	1
0	0	0	0	0	1	1	1

Resultatet av en logisk operation är ett heltalsvärde, som bildas då motsvarande databitar i de båda heltalen utsätts för operationer enligt tabellen ovan.

En heltalsvariabel eller flyttalsvariabel kan tilldelas värdet av detta resultat.

Exempel

```
10 A%=13% OR 14%
20 PRINT A%
RUN
15
```

AND, OR, XOR,EQV,IMP och **NOT** kan användas på variabler och beräknade uttryck. De opererar då på en databit i taget. Flyttalsvariabler, som behandlas med logiska operatorer, omvandlas till heltal innan operationen utförs.

5 Teckensträngar

BASIC II bearbetar inte bara numeriska data utan också data i form av teckensträngar. En teckensträng är en följd av tecken t.ex. alfanumeriska och skiljetecken.

5.1 Strängkonstanter

På samma sätt som man kan använda numeriska konstanter, är också strängkonstanter tillåtna. Strängkonstanter omges av ett av två avgränsningstecken, citationstecken (") eller apostrof ('). Vill man använda något av dessa tecken inne i texten, ska det dubbeltecknas, om det samtidigt används som gränstecken. Om det andra tecknet används som gräns, går det bra att skriva strängen, som den ska se ut.

Strängen MATS' BIL skrivs "MATS' BIL" eller 'MATS" BIL'.

5.2 Strängvariabler

Tillåtet variabelnamn för en strängvariabel är ett godtyckligt, tillåtet variabelnamn, följt av tecknet ¤, valutatecken. Detta tecken motsvaras i engelskspråkig litteratur av dollartecken.

Exempel D¤, G1¤ är enkla strängvariabler.
A¤(8), G5¤(M,N), J¤(I) är vektorsträngar.

OBS!
Samma namn, utan ¤, definierar en helt annan variabel, som kan användas i samma program som strängvariabeln.

Exempel F, F¤ och F% kan användas parallellt.

5.3 Strängvektorer

Instruktionen **DIM** används för att definiera en- eller flerdimensionella strängvektorer. Följande **DIM**-satser finns att välja på:
Exempel

DIM W¤(2,4)=8 stränglängd 8, högsta index 2 och 4

DIM R5¤(9,9) strängl. max. 80, högsta index 9 och 9

DIM NAMN¤(7,6,3,2)=10 strängl.=10, fyrdimensionell matris med högsta index 7,6,3 och 2

5.4 Strängstorlek

Längden av en icke dimensionerad strängvariabel sätts automatiskt till den aktuella längden första gången strängen tilldelas ett värde, som är skilt från "tom sträng" (<>"").

Om färre än 80 tecken används, får strängen standardlängden 80 tecken.

Varje sträng, även vektorer, har två längder:

1. Den maximala längden, d.v.s. det antal bytes, som reserverats för strängen.
2. Den aktuella längden, d.v.s. det antal bytes, som för tillfället används. Den aktuella längden kan växla mellan noll och maximala längden. Det är den aktuella längden, som kan undersökas med **LEN.**

Båda dessa längder sätts till noll, då programmet startas. De ändras efter hand som strängar dimensioneras eller tilldelas värden.

Om en sträng tilldelas värdet "tom sträng" (=" "), sätts dess aktuella längd till noll. Inga andra åtgärder vidtas.

När en sträng tilldelas ett värde skilt från "tom sträng" och har en maximal längd skild från noll, kontrolleras att det nya värdet får plats. Om den reserverade längden inte räcker, fås en felutskrift. Om längden räcker till, tilldelas strängvariabeln det nya värdet, och den aktuella längden blir då aktuellt antal bytes.

5.5 Strängfunktioner

Ett helt sortiment av funktioner används på teckensträngar. Med hjälp av dessa kan man i programmet utföra aritmetiska operationer på numeriska strängar, sammanfoga två strängar, ta fram en del av en sträng, ta reda på antalet tecken i strängen, skapa den teckensträng som motsvarar ett visst tal eller tvärtom, söka efter en viss delsträng i en större sträng o.s.v. Se avsnitt 10.2.

5.6 Strängaritmetik

Med strängaritmetiken kan man behandla numeriska strängar med aritmetiska operatorer. På detta sätt kan man få högre precision i beräkningarna. Numeriska strängvariablers namn måste avslutas med ett ¤-tecken. Numeriska strängkonstanter måste avgränsas av citationstecken (") eller apostrofer(').

De numeriska strängar, som ska behandlas med strängaritmetiken, får ha en största längd av 125 tecken.

5.7 Stränginmatning

Precis som övriga variabler kan strängvariabler ges värden med instruktionerna **READ, DATA** och **INPUT.**

Exempel

```
10 INPUT "Adress, Namn "A4¤,A5¤
```

är samma sak som

```
10 PRINT "Adress, Namn ";
20 INPUT A4¤,A5¤
```

Instruktionen **INPUT LINE** är mycket användbar, då man vill mata in strängar. Den tar emot en rad från tangentbordet.

Exempel

```
45 INPUT LINE D¤
```

Exempel

```
210 READ A,B,C¤
290 DATA 17,14,61
```

Variablerna tilldelas följande värden:

A=17
B=14
C¤="61"

Instruktionen **INPUT** används för att mata in numeriska värden på samma sätt som teckensträngar.

5.8 Utmatning av strängar

När en sträng skrivs med en **PRINT**-instruktion, matas enbart tecknen inom citationstecken eller apostrofer ut. Avgränsningstecknen syns aldrig.

Exempel

```
10 PRINT "Javisst!"
RUN
Javisst!
```

Strängar kan också lagras i filer på någon yttre enhet.

5.9 Relationsoperatorer på strängar

Relationsoperatorer, när de tillämpas på strängar, avser alfabetisk ordningsföljd.

Exempel

```
15 IF A¤(I%)<A¤(I%+1%) THEN GOTO 115
```

När rad 15 genomlöps, sker följande: A¤(I%) jämförs med A¤(I%+1%). Om A¤(I%) kommer före A¤(I%+1) i alfabetisk ordning sker uthopp till rad 115.

När strängarna har olika längd, jämförs den kortare strängen med de första tecknen i den längre, tecken för tecken. När den kortare strängen är slut, gäller det resultat, som då finns, utom i det fall att de första tecknen i den längre strängen är lika med den kortare. Då anses den kortare strängen vara minst.

Relationsoperatorer, som används på strängvariabler

Operator	Exempel	Betydelse
=	A¤=B¤	strängarna A¤ och B¤ är lika
<	A¤<B¤	A¤ är före B¤ i alfabetisk ordning
<=	A¤<=B¤	A¤ är före B¤ eller lika med B¤
>	A¤>B¤	A¤ är efter B¤ i alfabetisk ordning
>=	A¤>=B¤	A¤ är efter B¤ eller lika med B¤
<>	A¤<>B¤	strängarna A¤ och B¤ är ej lika

6. Att arbeta med BASIC II

6.1 Hur man skriver programrader

6.1.1 Fritt format i varje sats

BASIC tillåter "fritt textformat". Datorn bryr sig inte om extra mellanslag i en sats. Dessa fyra satser är likvärdiga:

```
30 PRINT S
30 PRINT S
30PRINTS
30P RINT S
```

Då programmen skrivs ut på skärm eller skrivare, listar datorn dem alltid på sitt vanliga sätt, oavsett hur Du skrivit in satserna.

OBSERVERA! I följande fall är mellanslagen viktiga:
- **EXTEND**-mod
- **DATA**-satser

6.1.2 Skrivsätt

En rad kan antingen utföras omedelbart (direktexekveras) eller lagras i programminnet för att exekveras senare och slutligen lagras på yttre minnesenhet (flexskiva eller kassett).

Då en sats är färdigskriven, ska **RETURN**-tangenten tryckas ned.

Exempel

```
10 INPUT A,B,C,D,E          (tryck RETURN)
20 LET S=(A+B+C+D+E)/5      (tryck RETURN)
30 PRINT S                  (tryck RETURN)
40 END                      (tryck RETURN)
```

RETURN-tangenten talar om, att programsatsen är avslutad. Om satsen innehåller fel, skrivs ett felmeddelande på skärmen.

6.1.3 Att rätta fel

Tangenten ← fungerar som radertangenten på en skrivmaskin. Då den trycks ned, raderas närmast föregående tecken på skärmen.

Att skriva så här:
```
20 LR← ET S=10
```
är likvärdigt med:
```
20 LET S=10
```

På samma sätt är:
```
30 LET←←← PRINT S
```
likvärdigt med:
```
30 PRINT S
```

En avslutad rad, som ger felmeddelande, kan ändras med piltangenterna (→ och ←).

Exempel
```
10 LE S=10
```
Detta ger ett felmeddelande. Stega fram med →och ändra programraden.

För att ändra en färdigskriven rad använder Du kommandot **ED.**

6.1.4 Att ta bort en rad.

För att ta bort den sats, Du håller på att skriva, trycker Du antingen CTRL och X samtidigt, eller tangenten CE. Då raderas hela den programrad, Du just skrivit.

OBS!
Om Du vill ta bort en redan inskriven progamrad, skriv radnumret följt av RETURN, varvid raden med det angivna numret raderas.

Exempel
```
5 LET S=0
10 INPUT A,B,C,D,E
20 LET S=(A+B+C+D+E)/5
```
För att ta bort rad 5 skriver Du:

```
5
```
(tryck RETURN)

Kontrollera med **LIST**-kommandot.

6.1.5 Att ändra en programrad

Ett sätt att ändra en redan skriven programrad, är att skriva om den i dess nya form. Då Du avslutar raden med RETURN, ersätter den nya raden den gamla med samma nummer.
För att ändra rad 5 ovan kan Du t.ex. skriva:

```
5 LET S=5
```
(tryck RETURN)

Den gamla rad 5 ersätts nu med den nya.

Om Du bara vill göra små ändringar i programraden, kan Du använda kommandot **ED.**

6.2 Att ändra i ett program

Du har stora möjligheter att redigera Dina program.
Rader kan tas bort, läggas till eller ändras enligt 6.1. Kommandot **MERGE** gör det möjligt att länka program i minnet med en uppsättning programrader, som lagrats på ett yttre minne. Med kommandot **ERASE** raderar Du flera programrader åt gången. Kommandot **ED** gör det lätt att ändra några tecken i en redan inskriven rad.

När Du redigerar (eller editerar) Dina program, kan radnumreringen behöva ändras. Använd då kommandot **RENUMBER.**

6.3 Att köra ett program

Kommandot **RUN** startar körningen (exekveringen) av programmet. När Du skrivit **RUN** och avslutar med RETURN, börjar datorns BASIC utföra satserna på programraderna i minnet för användarprogram. Den rad, som har det lägsta numret, genomlöps först. Exekveringen fortsätter tills BASIC finner någon av följande tre orsaker att stanna:

STOP
END
feluthopp

När programmet genomlöper någon av satserna **STOP** eller **END,** stannar det. Alla variabler behåller sina värden. Du kan undersöka variablerna genom att adressera dem med variabelnamnen.
Ex: Du vill veta värdet på A,S och K%. Skriv då så här

```
PRINT A,S,K%                                    (RETURN)
```

Datorn skriver då ut de värden, variablerna hade, då exekveringen avbröts.

Fel presenteras med ett felmeddelande på skärmen.

Ett program, som körs, kan stoppas genom att Du trycker

CTRL/C (båda tangenterna på en gång)

Datorn kan då utföra en instruktion i taget om Du trycker

CTRL/S (båda tangenterna på en gång)

För fortsatt exekvering kan valfri tangent tryckas ner.

För att avbryta programexekveringen måste Du trycka

CTRL/C ytterligare en gång.

6.4 Vilka satser används var?

Syfte:
Att skriva meddelanden och förklaringar i programtexten
Använd:
REM-satser eller !

Syfte:
Att tilldela en variabel ett värde
Använd:
instruktionen **LET**

Syfte:
Att tilldela variabler värden
Använd:
instruktionerna **READ, DATA, ON - RESTORE** och **RESTORE**

Syfte:
Dataöverföring till och från systemet
Använd:
instruktionerna **INPUT, INPUT LINE, PRINT, GET, PUT, PREPARE, OPEN, CLOSE**
instruktionen **OUT** och funktionen **INP** för att styra in- och utmatning via I/O-portarna

Syfte:
Att styra programexekveringen

1. Ovillkorligt hopp
Använd:
instruktionen **GOTO**

2. Villkorligt hopp
Använd:
instruktionerna **IF - THEN, ON** ....- och **WHILE - WEND**

3. Programloopar (slingor)
Använd:
instruktionerna **FOR - NEXT**

4. Modulär programmering med underprogram
Använd:
instruktionerna **GOSUB - RETURN** och **DEFIN - FNEND**

Syfte:
Felhantering
Använd:
instruktionerna **ON ERROR GOTO - RESUME**
funktionen **ERRCODE**

Syfte:
Kombinera BASIC-program med program i ASSEMBLER-språk
Använd:
funktionen **PEEK**
instruktionen **POKE**
instruktionen **CALL**

Syfte:
Definiera och flytta datablock
Använd:
instruktionerna **PREPARE, OPEN, CLOSE, PRINT, GET, PUT, INPUT** och **INPUT LINE**

Övriga instruktioner:
COMMON och **DIM** för att reservera utrymme för variabler
STOP, TRACE och **NOTRACE** förenklar felsökning av program

I kapitel 10 finns en lista över de många, tillgängliga matematiska, logiska och andra funktioner, som utökar användarens möjligheter att skriva avancerade program.

6.5 Deklarationer

Följande deklarationssatser förekommer:

- **FLOAT/INTEGER**
- **SINGLE/DOUBLE**
- **NO EXTEND/EXTEND**
- **OPTION BASE** (0/1)

Om deklarationssatser används, måste de placeras först i programmet. Därefter ska eventuella **COMMON**- och **DIM**-satser följa.

7 Direktanvändning

Med BASIC II är det lätt att använda datorn för att lösa sådana problem, vanligtvis rent matematiska, som inte kräver ett datorprograms förmåga att upprepa samma förfarande.

Enklare uttryckt: BASIC II gör det möjligt att använda datorn som en avancerad, elektronisk räknare, genom att BASIC-satserna kan exekveras direkt.

När BASIC väntar på kommandon kan en BASIC-sats skrivas utan radnummer. Då en sådan sats avslutas med RETURN, utförs den omedelbart. Detta kallas direktanvändning eller körning i direktmod.
De flesta BASIC-satser kan användas i direktmod.

Exempel

```
A=1.5 : B=3
PRINT "(A+B*A)=";(A+B*A)
```

En sats, som skrivs efter ett radnummer, antas vara en programsats, som ska utföras vid ett senare tillfälle.

Körning i direktmod är mycket användbart vid programutveckling och felsökning. Man kan lätt ändra eller läsa av variablernas värden, och programmets förlopp kan på detta sätt övervakas och styras.

Direktanvändning i samband med program kan användas

- när man tryckt CTRL/C två gånger

- vid felmeddelande

- efter **STOP** eller **END**

8 Kommandon

BASIC är tillgänglig för direkt kommunikation via tangentbordet, om man skriver vissa, direkta kommandon. Ett flertal andra satser kan också exekveras direkt, när de skrivs utan radnummer. Se avsnitt 7.

Då ett kommando är skrivet och avslutat med RETURN, utför datorn den beordrade aktiviteten omedelbart. Ett BASIC-program med radnummer lagras däremot i minnet och börjar utföras, då man ger kommandot **RUN.**

På skärmen visas texten ABC 800, då BASIC väntar på nästa kommando. Kommandon ska skrivas utan radnummer.

Olika kommandon styr programeditering, körning av program och viss filhantering. Varje kommando identifieras av ett nyckelord i början av raden.

I det följande anges

- reserverade ord - med fet stil, t.ex. **LOAD, SAVE** och **RUN**
- uppgifter som kan utelämnas - inom hakparentes, t.ex. [enhet:]
- alternativa uppgifter - med snedstreck, t.ex. "data"/strängvariabel
- ytterligare uppgifter - med punkter, t.ex. ["data"/strängvariabel,...,...]

Allmänt gäller att

- enhet adresseras med DR0:, DR1:, CAS: och PR:
- om enhet utelämnas, adresseras alltid drivenhet 0 (DR0:) först och därefter drivenhet 1 (DR1:). Om både flexskiveenhet och kassettminne är inkopplade, måste CAS: anges, om kommandot ska utföras på kassetten.
- filnamn får bestå av maximalt åtta bokstäver/siffror, varav det första tecknet måste vara en bokstav. Dessutom kan filtyp (3 tecken) användas valfritt för förtydligande av filnamnet.
- filtyp inte behöver anges. Undantag finns dock. Detta anges i så fall vid respektive syntax. Om filtyp utelämnas, utförs kommandot först på filtyp BAC och därefter på BAS.
- RETURN-tangenten måste tryckas ned efter varje avslutad inmatning.

Nedan följer en lista med kort beskrivning av varje **kommando:**

AUTO	Automatisk radnumrering
BYE	Övergång till skivoperativsystemet (DOS)
¤BAS	Återgång till BASIC

CLEAR	Nollställer alla variabler och stänger alla filer
CON	Fortsätter exekveringen av ett stoppat program
ED	Möjliggör editering (korrigering)
ERASE	Raderar flera programrader
LIST	Listar programmet
LOAD	Laddar in ett program
MERGE	Länkar programfiler
NEW	Raderar programminnet
RENUMBER **REN**	Ändrar radnumreringen
RUN	(Laddar in och) Exekverar ett program
SAVE	Lagrar ett program
SCR	Raderar programminnet
UNSAVE	Raderar en fil

Nedan följer en utförlig beskrivning av samtliga kommandon.

AUTO

Format

AUTO [argument 1 [, argument 2]]

Där <**argument 1**> anger **första radnummer**, som ska skrivas och <**argument 2**> anger **radintervall**.

Argumenten är **ej nödvändiga**. Om inga argument anges, börjar radnumreringen med första jämna tiotal efter de redan skrivna raderna. Intervall sätts till 10.

Funktion

Skriver automatiskt ett nytt radnummer efter varje vagnretur. Man slipper alltså skriva radnummer vid programmering.

Användning

Kommandot kan användas när som helst under pågående programmering. För att **avsluta** den automatiska radnumreringen trycker man **RETURN som första tecken på en ny rad**. Om inmatad rad ger felmeddelande, avbryts radnumreringen och raden kan rättas. Numreringen kan återupptas med **AUTO**.

Exempel **AUTO** 10,5

Första raden får nummer 10 och radnumren stegas upp med 5 för varje rad.

```
AUTO 10,5
10 LET A=1
15 —  —
20 —  —
25 —  —
```

o.s.v

BYE

Format **BYE**

Funktion Övergång till skivoperativsystemet (DOS).

Verkan Stänger de filer, som är öppna, och laddar kommandointerpretatorn CMDINT.SYS.

¤BAS

Format **¤BAS**

Funktion Återgång till BASIC.

Verkan Avbryter arbetet under DOS och återgår till BASIC.

CLEAR

Format **CLEAR**

Funktion Nollställer alla variabler och stänger alla öppna filer.

Verkan **Påverkar ej programmet, som finns i minnet.**

CON

Format **CON** (eller **CONTINUE**)

Funktion Fortsätter programexekveringen, där den har stoppats med antingen två CTRL/C eller en **STOP**-sats.

Användning Direktkommandon kan användas för att ändra eller skriva ut variabelvärden, innan programmet startas igen med **CON.**

Observera **CON** kan inte användas om programmet ändrats eller om ett feluthopp skett.

ED

Format

ED [radnummer]

Där <radnummer> är numret på den rad, som ska ändras.

Funktion

Gör det möjligt att editera (ändra) i en programrad.

Verkan

När kommandot avslutats med RETURN, använder man tangenten högerpil→för att stega sig framåt i raden. Varje gång→trycks ned, visas ytterligare ett tecken. För att ändra innehållet, skriver man det nya innehållet på sin plats. Tangenten vänsterpil← används för att radera bort felaktiga tecken.

Exempel

Rad 100 har följande innehåll:

```
100 A=B+C+E
```

Vi antar nu, att C skall ersättas av D och att man inte vill skriva om hela raden. Gör då på följande sätt:

Skriv **ED** 100. Tryck RETURN. På skärmen visas då rad 100:

```
100 A=B+C+E
```

Stega fram med→tills följande text syns:

```
100 A=B+C+E
100 A=B+C
```

På raden under den ursprungliga framträder alltså Din nya text. Radera bort C med en tryckning på←. Skriv D. Den understa raden ser nu ut så här:

```
100 A=B+D
```

Tryck på→, så att resten av raden blir synlig.

```
100 A=B+C+E
100 A=B+D+E
```

Läs igenom hela raden och kontrollera. Tryck RETURN för att lagra den nya raden i stället för den ursprungliga.

Observera

Då felmeddelande erhålls i smband med programmering, finns den felaktiga raden kvar i datorn och kan editeras med →och←enligt ovan, utan föregående **ED**-kommando.
Om **ED** skrivs utan något radnummer, hämtas programmets första rad.

ERASE

Format

ERASE radnummer [-radnummer]
ERASE radnummer -
ERASE - radnummer

Funktion

Raderar en del av det program, som finns i minnet.

Verkan

Alla rader fr.o m. radnummer t.o.m radnummer utplånas.

Exempel

ERASE 20-200	Raderar raderna 20 - 200
ERASE -200	Raderar raderna 1 - 200
ERASE 20-	Raderar raderna 20 - programmets slut

LIST

Format

LIST [enhet:] filnamn[.typ]
LIST [enhet:][radnummer[-radnummer]]
LIST radnummer -
LIST - radnummer
LIST enhet:, [radnummer -radnummer]

Där <filnamn.typ> är namnet på en programfil. <Enhet> kan vara t.ex. CAS:, PR:, DR0: eller DR1:.

Funktion

Listar hela eller del av program.

Verkan

1. **LIST** [enhet:] filnamn [typ]
Lagrar programmet, som är i arbetsminnet, på yttre minne i textformat. Programmet lagras under namnet <filnamn>. Jämför **SAVE.** Filtypen blir BAS, om inget annat anges.

2. **LIST**
Hela programmet listas på skärmen.

3. **LIST** radnummer
Endast den angivna raden listas på skärmen.

4. **LIST** radnummer I - radnummer II
Alla rader fr.o.m. radnummer I t.o.m. radnummer II listas.

5. **LIST** PR:
Hela programmet listas på skrivare.

6. **LIST** - radnummer
Programmet listas t.o.m. angivet radnummer.

7. **LIST** radnummer -
Programmet listas fr.o.m. angivet radnummer.

Observera

Ett långt program listas på skärmen, tills skärmen är full. Nästa rad framträder, då man trycker på mellanslagstangenten. Listningen stoppas med CTRL/C, RETURN eller valfritt BASIC-kommando.

Exempel

```
LIST ABC800
LIST !Listar
LIST 100
LIST 100-500
LIST PR:
LIST PR:, 100-200
```

Lagrar filen ABC800 på externt minne
hela programmet på bildskärmen
Listar rad 100
Listar raderna 100-500
Listar hela programmet på skrivare
Listar raderna 100-200 på skrivare

LOAD

Format

LOAD [enhet:] filnamn[.typ]

Där <filnamn.typ> är namnet på det program, som ska laddas in. Enhet kan vara CAS:, DR0: eller DR1:.

Funktion

Laddar in ett program från yttre minne.

Exempel

```
LOAD ABC
LOAD DR1: PROG.BAS
```

Observera

Om filtyp (de tre tecknen efter punkten) inte anges, söker datorn först efter filtyp .BAC och sedan filtyp .BAS.
Datorn läser hela filen fram till EOF (end of file) och inte bara till **END.**

MERGE

Format

MERGE [enhet:] filnamn[.typ]

Där <filnamn> är namnet på en fil på en yttre minnesenhet. Filen ska vara lagrad i textform (med **LIST**).

Funktion

Länkar programfiler så att programraderna kommer i nummerordning. Programmet i arbetsminnet överlagras med den begärda programfilen från en yttre enhet.

Verkan

De numrerade BASIC-raderna från den begärda filen läggs in i programmet i datorns arbetsminne. Varje rad felsöks, då den läggs in. Radnummer, som inte finns i programmet förut, läggs på sin plats i nummerordning. Om en ny rad har samma nummer som en rad i programmet, läggs den nya raden på den gamlas plats.

Den yttre filen läses till EOF(end of file).

Exempel

I datorn finns följande program:

```
5 Y = 10
10 PRINT
20 FOR L=1 TO 10
30 PRINT L;TAB(Y);"I";
40 READ Y
50 FOR I=1 TO Y
60 PRINT " * ";
70 NEXT I
80 PRINT
90 PRINT TAB(Y);"I"
100 NEXT L
```

På en yttre minnesenhet (flexskiva, kassett) finns följande programfil under namnet TABELL:

```
200 DATA 5,4,2,3,1
300 DATA 10,15,28,15,6
999 END
```

För att länka dessa båda filer ges kommandot:

```
MERGE TABELL
```

Raderna 200, 300 och 999 läses då in till programmet i arbetsminnet.

NEW

Format **NEW**

Funktion Raderar innehållet i arbetsminnet.

Verkan Raderar arbetsminnet, nollställer alla variabler och återställer pekare. Kommandot rensar på detta sätt minnet från alla spår av programmet och börjar på nytt.

Alla öppna filer stängs.

Använd detta kommando innan Du börjar skriva in ett nytt program.

Observera **Kommandot SCR (scratch) kan också användas. SCR har samma funktion som NEW.**

RENUMBER
REN

Format **REN[UMBER]** [radnr[,intervall[,fr.o.m. rad -t.o.m. rad]]]

Där <radnr> är det nummer, som första raden ska få. Om inget radnummer anges, får första raden nummer 10.

<Intervall> anger intervallet mellan två rader. Om inget intervall anges, ökas radnumret med 10 för varje ny rad.

<Fr.o.m. rad> - <t.o.m. rad> anger vilka rader, som ska numreras om. Om inget anges, numreras alla rader i programmet om.

För att <intervall> ska kunna anges, måste <radnr> anges.

Både <radnr> och <intervall> måste anges, om man ska kunna ange <fr.o.m rad>-<t.o.m. rad>.

Funktion Ändrar radnumreringen i det aktuella programmet.

Verkan

Alla radnummer i programmet ändras på det sätt, som begärts i kommandot **RENUMBER.**

Alla radnummer, som används i satser med **GOSUB, GOTO, IF, ON** eller **RESUME** ändras, om det behövs för att hoppen ska ske till samma programsats som tidigare.

Om någon rad i programmet ger ett hopp till en icke existerande rad, skrivs ett felmeddelande ut på skärmen. I detta fall behåller raderna sin ursprungliga numrering.

Exempel

```
REN
REN 10
RENUMBER 10,5
REN 10,5,10-50
REN 10,5,-100
REN 10,5,100-
```

RUN

Format

RUN [enhet:] [filnamn[.typ]]

Där <filnamn> är namnet på den programfil, som ska laddas in och köras. Om ingen filtyp anges, söker datorn först efter .BAC och i andra hand efter .BAS.

Funktion

Laddar och exekverar ett BASIC-program eller exekverar (kör) programmet i datorns minne.

Verkan

1. **RUN**
Alla variabler och fält i programminnet nollställs. Datasatser återställs och programmet i minnet genomlöps med början på den rad, som har lägst nummer.

2. **RUN** filnamn[.typ]
Programmet <filnamn[.typ]> laddas in på samma sätt som med kommandot **LOAD.** Det nyss inlästa programmet genomlöps med början på den rad, som har lägst nummer.

Exempel

Ex.1

```
10 READ A,B
20 LET A=A+B
30 PRINT A
40 DATA 2,3
50 END
RUN
5
```

Ex.2
Om samma program hade varit lagrat på en yttre minnesenhet under namnet APLUSB, ser bildskärmen ut så här:

```
RUN APLUSB
5
```

SAVE

Format

SAVE [enhet:] filnamn[.typ]

Där <filnamn> är namnet på den nya filen i form av en teckensträng. Om ingen filtyp anges, lagras filen som typ .BAC.

Funktion

Skapar en programfil på ett yttre minne och lagrar programmet, som finns i datorns arbetsminne, i denna fil.

Verkan

Kommandot medför, att programmet i arbetsminnet lagras under det angivna filnamnet.
Programmet lagras i internkodsformat, så att det kan laddas snabbt.

Exempel

```
10 - -
900 - -
999 END
SAVE TEST
```

Observera

Om filen redan finns på flexskiva,kommer den gamla filen att förstöras och ersättas av det nya programmet, om inte filen på skivan är skrivskyddad.

UNSAVE

Format

UNSAVE [enhet:] filnamn[.typ]

Funktion

Raderar en fil på flexskiva.

Exempel

När man har slutfört allt arbete med filen XYZ, raderas filen med följande kommando:

```
UNSAVE XYZ
```

Observera

Om man inte anger filtyp, söker datorn först efter .BAC och i andra hand .BAS.
Kommandot **UNSAVE** kan inte användas på en raderskyddad fil.

9 Instruktioner

Detta kapitel behandlar programsatserna i BASIC II. En programsats är en instruktion, som beordrar BASIC att utföra en viss operation. De flesta instruktioner kan även användas som kommandon.

Nedan följer en instruktionslista med en kort beskrivning av varje instruktion.

CHAIN	Laddar en programfil och exekverar den
CLOSE	Stänger filer
COMMON	Överför variabler till nästa program vid **CHAIN**
DATA	Datasats, värdet hämtas med **READ**
DEF FN	Definierar en användarfunktion
DIGITS	Anger max. antal siffror, som skrivs ut
DIM	Anger storlek för vektor/matris och sträng
DOUBLE	Dubbel precision (16 siffror)
END	Logiskt sista instruktion i BASIC-program
EXTEND	Möjliggör användning av långa variabelnamn
FLOAT	Återställer till flyttalsformat
FNEND	Avslutar flerradiga användarfunktioner
FOR	Inleder programloop (**NEXT** avslutar)
GET **GET COUNT**	Läser ett tecken/angivet antal tecken
GOSUB	Anropar en subrutin
GOTO	Hoppar till angivet radnummer
IF-THEN-ELSE	Styr villkorlig exekvering
INPUT **INPUT LINE**	Läser in data
INTEGER	Ändrar till heltalsformat
KILL	Tar bort en fil
LET	Tilldelar en variabel ett värde
NAME	Ger en fil nytt namn

NEXT	Stegar räknevariabeln i programloop
NO EXTEND	Stänger av **EXTEND**-mod
NO TRACE	Stänger av **TRACE**-funktion
ON ERROR GOTO	Hantering av feluthopp
ON-GOSUB	Villkorssats för hopp till subrutin
ON-GOTO	Villkorssats för hopp till olika radnummer
ON-RESTORE	Villkorlig återställning av datapekare
ON-RESUME	Villkorligt hopp från felhantering
OPEN-AS FILE	Öppnar en fil och ger den ett filnummer
OPTION BASE	Sätter lägsta värde för vektorindex
POSIT	Positionerar filpekare
PREPARE-AS FILE	Skapar en ny fil och ger den ett filnummer
PRINT **PRINT USING**	Skriver ut data med angivet format
PUT	Skriver en post
RANDOMIZE	Ger slumptalsgeneratorn ett nytt startvärde
READ	Tilldelar en variabel ett värde från **DATA**-sats
REM (!)	Inleder kommentar
RESTORE	Ställer datapekare
RESUME	Återhopp från felhantering
RETURN	Återhopp från subrutin
SINGLE	Ändrar precisionen till sju siffror
STOP	Stoppar programexekveringen
TRACE	Startar **TRACE**-funktion (felsökning)
WEND	Avslutar **WHILE**-funktion
WHILE	Ger uthoppsvillkor för **WHILE**-funktion

Nedan följer en utförlig beskrivning av samtliga instruktioner.

CHAIN

Format

CHAIN "filnamn.typ"/strängvariabel

Där <filnamn.typ> är namnet på det program, som ska laddas in. Om filnamnet avser en fil på flexskiva, och filtypen inte är angiven, söker datorn först efter .BAC och i andra hand .BAS.

Funktion

Det program, som anropas, laddas in och körs.

Användning

Om användarprogrammet är för stort för att laddas in i arbetsminnet, kan man dela upp programmet i flera delprogram. **CHAIN**-instruktionen används som logisk avslutning på ett program för att hämta in nästa. Varje program anropas med sitt namn. Det förra programmet raderas, och det nya laddas in. Den programrad, som har lägst nummer, utförs först, på samma sätt som vid kommandot **RUN. CHAIN**-instruktionen blir den sista instruktion, som utförs. Det sista programmet i en sådan kedja behöver alltså ingen **CHAIN**-sats, men det är vanligt, att man återgår med **CHAIN** till ett program, där användaren kan välja, vilket delprogram, som ska köras.

Observera

För att överföra variabler från ett program till ett annat, används instruktionen **COMMON.**

Exempel

```
550 CHAIN "PROGRAM4.BAC"
```

CLOSE

Format

CLOSE (filnummer, ...)

<Filnummer> anger den fil, som ska stängas.

Funktion

Avslutar in/utmatning och stänger filen.

Användning

Instruktionen **CLOSE** används för att stänga en eller flera filer. Om inte filnummer anges, stängs samtliga filer.

Observera

Instruktionen **END** stänger alla öppna filer.

Vid utmatning med **PRINT** matas innehållet i den sista buffertarean ut, när filen stängs.

COMMON

Format

COMMON variabel[(n,...)] [,variabel,...]
COMMON strängvariabel [(n,...)] = längd[,strängvariabel]

<n>=vektorindex och <längd>=strängens maximala längd

Funktion

Överför variablers värden från ett program till ett annat vid **CHAIN.**

Användning

Deklarationerna måste se likadana ut i alla de berörda programmen.

Observera

COMMON-strängvariablers längd måste deklareras. **COMMON**-satserna måste placeras omedelbart efter deklarationssatserna.

Exempel

```
10 COMMON A%,B(7),C¤(15)=25
```

DATA

Format

DATA värde [,värde, ...]

Där alla **DATA**-satser, var de än står i programmet, skapar en datalista. **DATA**-satsen måste stå ensam på en programrad.

Funktion

Utgör tillsammans med **READ**-instruktionen en metod att tilldela variabler värden.

Användning

DATA används enbart tillsammans med **READ** och omvänt. Se vidare **READ**.

Exempel

```
100 DATA ....en.text...,"..en.text."
110 DATA ' "...en.text..." '
120 FOR I=1 TO 3
130 READ A¤
140 PRINT "!";A¤;"!"
150 NEXT I
```

ger följande utskrift:

```
!....en.text...!
!..en.text.!
!"...en.text.."!
```

DEF FN

Format

Enradig funktion:
DEF FN identifierare [(argument)]=funktion
Flerradig funktion:
DEF FN identifierare [%/¤][(argument)] [**LOCAL** variabel [,variabel,...]]

Funktion

Definierar enradiga och flerradiga funktioner.

Användning

En flerradig **DEF**-funktion skiljer sig från enradiga användarfunktioner, genom att funktionsnamnet på första raden inte följs av något likhetstecken. Vilka typer av argument som helst kan användas, även 0. Det är också tillåtet att blanda olika typer av argument. Den flerradiga funktionsdefinitionen ska innehålla satser med följande utseende:

RETURN <uttryck>
FNEND

När **RETURN** genomlöps, returneras funktionsvärdet varefter återhopp sker. Det kan finnas mer än en sådan sats, som framgår av exempel nedan.

Exempel på enradig funktion:

```
10 DEF FNA(X,Y)=X+X * Y
```

I nedanstående exempel väljs det större av två tal. Detta tal returneras som funktionsvärde. Det är vanligt, att man på detta sätt använder **IF - THEN** instruktionen i flerradiga funktioner:

```
10 DEF FNM(X,Y)
20 IF Y<=X THEN RETURN X
30 RETURN Y
40 FNEND
```

Nästa exempel visar en rekursiv funktion, som beräknar N! (N-fakultet). (Det finns effektivare, icke-rekursiva metoder att beräkna N!)

```
10 DEF FNFak(M%)
20 IF M%=1% THEN RETURN 1% ELSE
RETURN M% * FNFak(M%-1%)
30 FNEND
35 INPUT "Beräkna fakultet för följande tal: ";X
40 PRINT X;"-fakultet är ";FNFak(X)
50 END

RUN
Beräkna fakultet för följande tal: 4
4-fakultet är 24
```

En variabel, som används inne i själva funktionen, och inte är ett argument eller en lokal variabel för just den funktionen, behåller sitt värde i det överordnade programmet. En flerradig användarfunktion kan anropa en annan, men denna andra måste då vara helt innesluten i den första. Här gäller samma regler för programmets uppbyggnad som vid programslingor. De radnummer, som används i definitionen, får inte anropas från resten av programmet. Om man använder **ON ERROR GOTO** i definitionen, stängs denna felhantering av vid återhopp till det program, som anropat funktionen.

Om man behöver tillfälliga variabler i en definierad funktion, bör dessa variabler vara lokala, så att man inte råkar skada de globala variablerna. Man behöver då inte leta fram "oanvända" variabler.

Modifieraren **LOCAL** gör, att man kan använda ett lokalt variabelnamn. Vektorer kan inte deklareras som lokala variabler. Strängvariabler måste anges med sin längd.

```
10 DEF FNA(X) LOCAL A,A¤=10
20 A=33: A¤="Lokal"
30 PRINT A¤
40 PRINT A
50 RETURN 5*X
60 FNEND
100 A=22: A¤="Global"
110 PRINT A¤
120 PRINT A
130 PRINT FNA(8)

 RUN
Global
 22
Lokal
 33
 40
```

Nedanstående exempel visar en strängfunktion:

```
100 PRINT FNV1¤("AaBbCcDdEeFf",5%,10%)
110 END
120 DEF FNV1¤(A¤,B%,C%)
130 IF B%=C% THEN RETURN LEFT¤(A¤,B%) ELSE
 RETURN RIGHT¤(A¤,C%-B%)
140 FNEND
 RUN
CcDdEeFf
```

FNEND

Format	**FNEND**
Funktion	Avslutar en flerradig funktion.
Användning	Se instruktionen **DEF FN.**
Observera	Denna sats får aldrig genomlöpas. Innan **FNEND** ska funktionen ha gett ett uthopp med **RETURN.**

DIGITS

Format	**DIGITS** antal siffror
Funktion	Ger antal siffror, som skrivs ut med **PRINT**
Verkan	Det tal, som skrivs ut med **PRINT,** avrundas till angivet antal siffror. Talvärden, som är för stora för att skrivas med detta antal siffror, skrivs på exponentiell form med angivet antal siffror.
Observera	Instruktionen **DIGITS** påverkar inte beräkningarnas noggrannhet. Default: 6/16 siffror beroende på precision.

DIM

Format

DIM variabel(n)[,variabel(n,...),...]
DIM strängvariabel[(n,...)] [=uttryck]

Där värdet efter ''='' anger stränglängden. n,... är högsta indexvärde. Begynnelseindex är <undre gräns> om inget annat angivits. <undre gräns> är antingen 0 eller 1 beroende på **OPTION BASE.** Om ingen sådan instruktion har genomlöpts, har <undre gräns> värdet 0.

Funktion

Ger maximala antalet element i indexerade variabler och strängar.

Användning

Man kan ha hur många index som helst.
Alla värden, som anges i en **DIM**-sats, avrundas till heltal.
Om man använder en indexerad variabel, utan att dimensionera den, antas dimensionen till 10 för varje index.
Alla variabler har värdet 0 tills de tillordnats ett annat värde.
Om en strängvariabel inte dimensionerats, sätts dess maxlängd automatiskt till den längd strängen har, då den för första gången tillordnats ett värde skilt från tom sträng. För en sträng, som inte dimensionerats, reserveras dock aldrig kortare längd än 80 tecken.

Exempel

```
DIM A¤(N)
```

En strängvektor med strängarna A¤(< undre gräns>) - A¤(N). Varje sträng har längden 80.

```
DIM A¤(N)=14
```

Som ovan men varje sträng har längden 14 tecken.

```
10 DIM X(5),Z(4,3),A(10,10)
12 DIM A4(100)
14 DIM A¤(20),B¤(10,20)
16 DIM C¤(40)=4
18 DIM D¤(10,10)=8
20 DIM Q¤=253
```

AVANCERAD PROGRAMMERING:

Den undre gränsen 0 eller 1 kan vid behov ändras för varje index. Man ersätter maxvärdet med två värden åtskilda av kolon.

Exempel

```
DIM A(-2:2)
```

Här får vi en vektor med fem element A(-2), A(-1), A(0), A(1), A(2) oberoende av den undre gräns, som gäller för tillfället.
Omdimensionering av tidigare dimensionerad variabel är tillåten, om den nya DIM-satsen medför en mindre dimension.

DOUBLE

Format **DOUBLE**

Funktion Ändrar alla variabler och uttryck med flyttal till dubbel precision (16 siffror).

Användning Deklarationen **DOUBLE** ska stå innan variablerna används i programmet och kan inte ändras när programmet väl har startats med **RUN.** Ändringen kan göras om en programrad har editerats eller om man har använt kommandot **CLEAR.**
Om precisionen inte deklareras i programmet, gäller **SINGLE.**

Observera Det går inte att blanda **SINGLE** och **DOUBLE** i ett och samma program.

END

Format **END**

Funktion Avslutar pogrammet.

Användning Instruktionen **END** ska ha det högsta förekommande radnumret i huvudprogrammet. Efter **END** får bara förekomma underprogram (subrutiner) och funktioner, vilka ger återhopp till huvudprogrammet. **END** stänger alla filer.

Observera Variablerna behåller sina värden efter **END.**
END måste stå först på raden.

EXTEND

Format **EXTEND**

Funktion Tillåter långa variabelnamn.

Observera Då **EXTEND** begärts, är BASIC känslig för mellanslag, utom efter radnummer och intill aritmetiska operatorer (− + ∗ /). Hopskrivna nyckelord kan misstolkas som långa variabelnamn.
Variabelnamnen får vara hur långa som helst och är signifikanta till alla delar.

Exempel

```
10 EXTEND
20 LET MOMS=MOMSGRUNDANDE* MOMSPROCENT
```

FLOAT

Format **FLOAT**

Funktion Samtliga tal tolkas som flyttal. Heltal betecknas med %.

Användning Se instruktionen **INTEGER**

Observera Som defaultvärde gäller **FLOAT.** Det är inte tillåtet att blanda **FLOAT** och **INTEGER** i samma program.

Exempel

```
10 A=125.5 !Flyttal
20 A%=12% !Heltal
```

FOR

Format

FOR variabel=uttryck **TO** uttryck [**STEP** intevall]

Där variabeln i **FOR...TO**-satsen får det begynnelsevärde, som ges av det första uttrycket. Därefter utförs instruktionerna mellan **FOR** och **NEXT**. När **NEXT** påträffas, adderas det angivna intervallet till variabeln. Se även under **NEXT.**

Om variabeln har ett värde, som är större än uttrycket efter **TO,** sker uthopp till satsen efter **NEXT.**

Om variabeln redan från början har ett värde, som är större än den övre gränsen efter **TO,** utförs inte satserna i loopen.

Uttrycken i **FOR**-satsen beräknas en enda gång vid inhoppet i loopen. Variabeln kontrolleras mot den övre gränsen varje gång loopen ska genomlöpas. Beträffande uttryck se avsnitt 2.4.

Funktion

Instruktionerna **FOR** och **NEXT** används tillsammans för att skapa programloopar. En loop innebär, att en eller flera instruktioner utförs ett visst antal gånger.

Användning

En programloop består av fyra delar:

1. Initiering av de villkor, som måste gälla, för att loopen ska genomlöpas en första gång.

2. Programloopens innehåll, d.v.s. de instruktioner, som ska upprepas.

3. Modifieringen, som ändrar variabelns värde, så att varje genomlöpning av loopen skiljer sig från de övriga.

4. Uthoppsvillkoret, som kontrolleras vid varje genomlöpning. När det är uppfyllt, sker uthopp till programsatsen närmast efter loopen.

Om **STEP**-uttrycket inte anges i **FOR**-satsen, antas intervallet vara +1. Detta medför, att de flesta **FOR**-satser inte innehåller något **STEP**-uttryck.
Variabeln kan också modifieras inne i själva loopen. När uthopp sker, har variabeln sitt nya värde, d.v.s. senast använda variabelvärdet + intervall.

FOR-loopar kan inte överlappa varann utan måste vara antingen totalt inneslutna i varann (kapsling) eller totalt fristående från varann.

Även om variabeln inte har uppnått sitt gränsvärde, kan man hoppa ur programloopen med ett villkorligt eller ovillkorligt uthopp. Om man vill hoppa tillbaka in i en loop, som inte hade genomlöpts fullständigt, måste man vara noga med att sätta korrekta värden för övre gräns och steglängd.

Exempel

Ex.1 Exempel på en **FOR** - **NEXT**-loop. Loopen utförs 20 gånger. Före uthoppet ur loopen skrivs A=20. **FOR**-satsen innehåller inget **STEP**-uttryck, alltså antas intervallet vara +1.

```
10 FOR A%=1% TO 20%
20 PRINT "A=";A%
30 NEXT A%
40 PRINT "A=";A%
```

Loopen består av raderna 10, 20 och 30. Då loopen genomlöps, skrivs A=1, A=2,..., A=20. När A% har värdet 20 och rad 30 genomlöps, ökas A% med 1 och rad 10 genomlöps. Eftersom A% nu är större än övre gränsvärdet, ger rad 10 uthopp till rad 40. Utskriften från rad 40 blir A=21.

Ex.2 Godtagbar kapsling

```
50 FOR A=1 TO 10
60 FOR B=2 TO 11
70 NEXT B
80 FOR C=1 TO 10
90 NEXT C
100 NEXT A
```

Ej godtagbar kapsling

```
150 FOR A=1 TO 10
160 FOR B=2 TO 11
170 NEXT A
180 NEXT B
```

Observera

I **FOR**-satser bör man använda heltalsvariabler, eftersom loopen då genomlöps mycket snabbare.

NEXT

Format

NEXT variabel

Där <variabel> är den variabel, som specificeras i tillhörande **FOR**-sats. Satserna **FOR** och **NEXT** avgränsar programloopen. När **NEXT**-satsen genomlöps, adderas intervallet till variabelns värde och programmet undersöker, om variabeln överskridit den övre gräns, som anges i **FOR**-satsen. När variabelns värde är större än det övre gränsvärdet, utförs satsen närmast efter **NEXT**-satsen.

Funktion

NEXT anger slutet på en programloop, som inletts med en **FOR**-sats. Vid **NEXT** inkrementeras variabeln (ökas med intervallet).

Användning

Se **FOR**.

GET

Format

GET strängvariabel

Funktion

Läser in ett tecken från tangentbordet till en strängvariabel.

Observera

Om tangentbordsbufferten är tom, väntar BASIC-tolken tills en tangent trycks ned. Vilket tecken som helst kan hämtas in.

GET#

Format

GET # filnummer, strängvariabel [**COUNT** antal tecken]

<Filnummer> är det filnummer, som definierats med instruktionen **OPEN.**

<Strängvariabel> är den sträng, dit tecknet(nen) överförs.

COUNT <antal tecken> anger antalet tecken, som ska läsas från filen.

Funktion

Läser in ett eller flera tecken från angiven fil till angiven strängvariabel.

Användning

Se kapitel 3.5

GOSUB

Format

GOSUB radnummer

<Radnummer> är det första radnumret i den anropade subrutinen. Programexekveringen fortsätter med detta radnummer.

Funktion

Ger uthopp till en subrutin.

Användning

En subrutin är en följd av programinstruktioner, som utför en uppgift, vilken återkommer vid flera tillfällen i ett program. Subrutiner och funktioner gör det möjligt, att anropa en sådan instruktionsföljd från flera ställen i programmet.

En subrutin är en programdel, som kan anropas med en **GOSUB**-instruktion. När subrutinen har fullgjort sin uppgift, sker ett återhopp via en **RETURN**-instruktion till programsatsen närmast efter den anropande **GOSUB**-satsen.

Exempel

```
50 GOSUB 1300
|
|
1290 REM Detta är en subrutin
1300 LET K=1
|
|
1360 RETURN
|
9999 END
```

Observera

Uthopp ur en subrutin får endast ske med **GOSUB** eller **RETURN.**

GOTO

Format

GOTO radnummer

Där <radnummer> vanligtvis inte är närmast följande rad i programmet.

Funktion

Ovillkorligt hopp till angivet radnummer i programmet.

Användning

Instruktionen **GOTO** används, när man vill åstadkomma ett ovillkorligt hopp till en annan programrad än den nästföljande. Med instruktionen **GOTO** kan man hoppa framåt eller bakåt i programmet.
När **GOTO** används i en programrad med flera satser, ska denna instruktion alltid stå sist på raden. Om någon annan programsats står senare på raden, utförs den aldrig

Exempel

```
10 X=20
20 PRINT X
30 X=X+1
40 GOTO 20
50 END
```

Observera

GOTO kan användas i direktmod i stället för **CON** (Continue), om man vill att programmet ska exekveras från en viss rad.

IF-THEN-ELSE

Format

IF villkor **THEN** sats[er]/radnummer [**ELSE** sats[er]/radnummer]

Där test sker med avseende på det angivna villkoret. Om det är uppfyllt (logiskt sant) utförs det, som står efter **THEN.** Om villkoret inte är uppfyllt (logiskt falskt), utförs den programrad, som följer närmast efter **IF**-satsen.

Funktion

Används för villkorlig styrning av ordningsföljden mellan programraderna.

Användning

Efter **THEN** i **IF**-satsen får finnas antingen ett radnummer eller en eller flera BASIC-satser. Om där finns BASIC-satser, och villkoret är uppfyllt, utförs dessa satser innan programmet fortsätter med raden närmast efter **IF**-satsen. Villkoret gäller för alla satser, som följer på samma rad som **IF**-satsen.

Efter **ELSE** anges antingen ett radnummer, dit uthopp sker, eller en eller flera satser, som utförs innan raden närmast efter **IF**-satsen. Om villkoret är uppfyllt, utförs de instruktioner, som finns mellan **THEN** och **ELSE.**

Vid beräkningen av villkorstuttryck utförs aritmetiska operationer i sin vanliga prioritetsordning. Relationsoperatorerna har samma inbördes prioritet och utförs efter de aritmetiska operatorerna men före de logiska operatorerna.

Relationsoperatorer

=	lika med
<>	icke lika med
<	mindre än
>	större än
<=	mindre än eller lika med
>=	större än eller lika med

Villkorsuttryck har värdet -1, om de är sanna, och värdet 0 om de är falska.

Ex. 5+6*5>15*2 är sant

Exempel

```
170 IF A<B+3 THEN 160
180 IF A=B+3 THEN PRINT "A har värdet ";A
190 IF A=>B THEN T1=B
200 IF A=B THEN PRINT "Lika ": A=1/B
210 IF A>B THEN PRINT "Större" ELSE PRINT "Mindre"
```

I rad 200 gäller villkoret både för **PRINT**-satsen och tilldelningssatsen.

INPUT

Format

INPUT [#filnummer/"ledtext"] variabel [,variabel,...]

Där <filnummer> är den beteckning filen givits i instruktionen **OPEN.**

<Variabel> innehåller namn på aritmetiska variabler, element i numeriska vektorer, strängvariabler eller element i strängvektorer.

Om #<filnummer> inte anges, antar systemet, att indata kommer från tangentbordet. Data läses från den fil eller den enhet, som tilldelats det angivna filnumret.

<Ledtext> är en teckensträng inom citationstecken. Kan endast användas vid inmatning från tangentbordet.

Funktion

Hämtar in data till det program, som körs.

Användning

Medan programmet exekveras, kan användaren mata in data efterhand som programmet begär in dem. **INPUT** låter datorn vänta på ett svar. Om ingen ledtext angivits, skrivs ett frågetecken på skärmen.

Det är ofta lämpligt att låta programmet skriva en ledtext, för att påminna användaren om vilka data, som ska skrivas in. Se Ex. 1 nedan. Efter en ledtext skrivs inget frågetecken.

Exempel

Ex. 1

```
10 INPUT "Ditt namn : ?" A¤
20 INPUT "Din adress : ?" B¤
```

är samma sak som:

```
10 PRINT "Ditt namn : ";
20 INPUT A¤
30 PRINT "Din adress : ";
40 INPUT B¤
```

Ex. 2

```
50 INPUT #3,C¤
```

Instruktionen medför, att data läses från fil 3. Data placeras i strängen C¤.

INPUT LINE

Format

INPUT LINE [# filnummer,] strängvariabel

<Filnummer> är det nummer, som ges i instruktionen **OPEN** och betecknar en yttre enhet eller en fil som logisk enhet.

Funktion

Tar emot inmatning av en rad tecken.

Användning

Programmet tar emot en rad tecken från den angivna filen. Alla tecken på raden läses in, även mellanslag, skiljetecken och citationstecken. De avslutande tecknen vagnretur (CR) och radframmatning (LF) läses också in.

Med instruktionen **INPUT LINE** kan man inte få någon ledtext, utan den får skrivas med en **PRINT**-sats.

Exempel

Ex. 1

```
10 PRINT "Din adress : ";
20 INPUT LINE A¤
```

Ex. 2

```
10 INPUT LINE A¤
20 A¤=LEFT¤(A¤,LEN(A¤)-2)
```

Exempel 2 tar bort CR och LF från strängen A¤.

INTEGER

Format

INTEGER

Funktion

Vid inmatning och listning av program antas alla variabler vara heltalsvariabler, om inget annat anges.

Användning

När ett program skrivs in, och man givit instruktionen INTEGER, behöver programmeraren inte ange suffixet % på heltalsvariablerna. Däremot ska alla flyttalsvariabler markeras med suffixet . (decimalpunkt), och strängvariabler ska som vanligt ha suffixet ¤.

Ett program, som är lagrat i textform och innehåller flyttalsvariabler, kan köras som ett rent heltalsprogram, om man ger kommandot **INTEGER** innan det laddas. När ett sådant program lagras, har man omformat det till heltalsprogram.

Observera

Om inget annat anges, gäller flyttalsformat.
Det är inte tillåtet att blanda **INTEGER** och **FLOAT** i samma program.

Exempel

```
100 REM Visar utskrift av flyttal/heltal
110 INTEGER
120 A.=12.345
130 B.=123
140 C=B.
150 D1=A.
160 PRINT A.,B.,C,D1
170 END
RUN
12.345 123 123 12
```

KILL

Format

KILL "[enhet:] filnamn[.typ]"

Där filen med namnet <filnamn.typ> inte är raderskyddad. Användaren kan inte radera en fil, som är raderskyddad.

Funktion

Den fil, som anges med filnamnet, raderas från det yttre minnet.

Exempel

När användaren inte längre behöver filen XYZ.TXT på flexskivan, kan filen raderas från skivan med följande sats:

```
460 KILL "XYZ.TXT"
```

LET

Format

[LET] variabel=uttryck

Ordet **LET** kan utelämnas.

Användning

Instruktionen **LET** används för att tilldela en variabel ett värde.

Exempel

```
10 LET A=5.02
20 LET B9=5*(X/2)
30 D=(3*A)/2-B
```

ON-GOTO

Format

ON uttryck **GOTO** radnummer[,radnummer,...]

Där värdet av <uttryck> (heltalsvärdet) används som pekare i radnummerlistan.

Funktion

Instruktionen **ON-GOTO** gör det möjligt att hoppa till en av flera rader, beroende på uttryckets värde.

Exempel

```
100 ON A/B GOTO 1000,1500,1700
```

ger uthopp enligt följande tabell:
1. rad nr 1000 om 0.5 <= A/B < 1.5
2. rad nr 1500 om 1.5 <= A/B < 2.5
3. rad nr 1700 om 2.5 <= A/B < 3.5
4. fel om A/B < 0.5
5. fel om A/B >= 3.5

ON-RESTORE

Format

ON uttryck **RESTORE** radnummer[,radnummer,...]

Heltalsvärdet av <uttryck> flyttar **DATA**-pekaren till angivet radnummer.

Funktion

Ställer **DATA**-pekaren efter samma urvalsförfarande som **ON-GOTO**-satsen.

Användning

Satsen **ON-RESTORE** kan på detta sätt användas för att ge villkorlig ändring av **DATA**-pekaren till en viss plats i data-buffern.

Exempel

```
10 FOR X=1 TO 3
20 READ A,B,C
30 ON X RESTORE 60,70,80
40 PRINT A;B;C
50 NEXT X
60 DATA 1,2,3
70 DATA 4,5,6
80 DATA 7,8,9
90 END
RUN
1 2 3
1 2 3
4 5 6
```

ON-RESUME

Format

ON uttryck **RESUME** radnummer[,radnummer,...]

Där värdet av <uttryck> (heltalsvärdet) används som pekare i radnummerlistan.

Funktion

Instruktionen **ON-RESUME** gör det möjligt att hoppa till en av flera rader, beroende på uttryckets värde, då satsen utförs. Felhantering återställs.

Användning

ON-RESUME används för villkorligt återhopp efter felhantering.

Exempel

```
10 ON ERROR GOTO 100
 |
 |
100 REM Felhantering
 |
 |
150 ON A RESUME 1000,2000
```

Observera

ON-RESUME används tillsammans med **ON ERROR GOTO.**

OPEN

Format

OPEN "[enhet:][filnamn[.typ]]" **AS FILE** filnummer

Där <enhet:> kan vara t.ex.
DR0: Drivenhet 0
DR1: Drivenhet 1
PR: Skrivare
CAS: Kassettminne

Uttrycket efter **AS FILE** ska vara ett heltalsvärde mellan 0 och 255.

Om skrivare öppnas, utelämnas <filnamn.typ>.

Funktion

Används för att öppna en fil med ett filnummer, som gäller inom det aktuella BASIC-programmet.

Användning

OPEN används för filer, som redan existerar.

Datafiler eller enheter har både sitt eget namn, som identifierar dem i systemet, och ett filnummer, som identifierar dem i programmet. **OPEN**-satsen ger sambandet mellan namnet och filnumret.

Skrivning på och läsning från filen sker med bl.a. **INPUT, PRINT, GET** och **PUT.**

Observera

För att kunna läsa data i en redan existerande fil, ska man alltid öppna filen med **OPEN.** Maximalt sju filer får vara öppna samtidigt.

Exempel

Ex. 1

```
50 OPEN "TEST.TXT" AS FILE 1
```

Ex. 2

```
10 OPEN "DATA.TXT" AS FILE 2
20 INPUT #2,A
30 INPUT #2,B
40 INPUT #2,C7¤
```

Värdet på variablerna A, B och C7¤ läses in från den fil, som öppnats som fil 2. Dessa värden läses direkt efter de senast lästa värdena. Om filen ska läsas från början, måste den öppnas igen med en **OPEN**-sats.

OPTION BASE

Format

OPTION BASE n

där n=0 eller 1

Funktion

Anger lägsta värdet för vektorindex.

Observera

Om ingen **OPTION BASE** angivits, gäller 0 som lägsta index. Deklarationen av **OPTION BASE** måste ligga före all dimensionering och användning av vektorer.

POSIT

Format

POSIT # filnummer, antal tecken

Funktion

Positionerar filpekare.

Användning

POSIT används för flyttning av filpekaren angivet antal positioner räknat från filens början (första positionen). Första positionen = 0. **POSIT** kan användas tillsammans med samtliga filhanteringsinstruktioner. Instruktionen pekar ut ett speciellt tecken eller första tecknet i en teckenföljd, som ska läsas eller skrivas. **POSIT**(filnummer) används för läsning av filpekarens läge (position).Se avsnitt 3.5.

Exempel

Ex. 1

```
10 POSIT #1,5
```

Filpekaren flyttas till position 5, d.v.s. pekar på det sjätte tecknet i fil 1.

Ex. 2

```
50 A=POSIT(1)
```

A = filpekarens position. I Ex. 1 befinner sig filpekaren i position 5, d.v.s. A=5.

Observera

Om filpekaren positioneras med **POSIT** efter en **PRINT**-instruktion, kommer filslut att skrivas innan **POSIT** exekveras. För att förhindra detta måste man lägga in en dummy **GET**-sats mellan **PRINT** och **POSIT**.

PREPARE

Format

PREPARE "[enhet:][filnamn.typ]" **AS FILE** filnummer

Funktion

Skapar och öppnar en ny fil med ett filnummer, som gäller inom det aktuella BASIC-programmet.

Användning

PREPARE används som **OPEN**, men skapar en ny fil. **OPEN** används då filen redan finns.

Exempel

```
10 PREPARE "DATA.TXT" AS FILE 2
20 PRINT #2,A
30 PRINT #2,B
40 PRINT #2,C¤
```

Värdet på variablerna A, B och C ¤ skrivs på fil 2 (DATA.TXT).

PRINT

Format

PRINT [#filnummer] "data"/variabel [,"data"/variabel]
Där # <filnummer> motsvarar filnumret i instruktionerna **OPEN** och **PREPARE**. Om filnumret utelämnas, skrivs värdena på bildskärmen. **PRINT** kan bytas mot ; (semikolon).

Om ett element inte är en enkel variabel eller konstant, beräknas uttryckets värde, innan data skrivs ut. Instruktionen kan också innehålla teckensträngar inom citationstecken.

Funktion

Matar ut data i ASCII-form.

Användning

Positionerna på en rad är numrerade från 0 till 39 alternativt 79. Raden är indelad i kolumner, fasta tablägen, som börjar i pos. 0, 15, 30, 45, 60 och 75. Ett kommatecken (,) efter en variabel eller en sträng i **PRINT**-listan innebär, att nästa element i listan börjar skrivas ut i nästa kolumn. Två kommatecken efter varann i listan innebär, att en kolumn hoppas över vid utskriften.

Ett semikolon (;) efter en variabel eller en sträng i listan innebär, att nästa element i listan börjar skrivas ut i nästa position, d.v.s. omedelbart efter det föregående tecknet. Om listan avslutas med ett semikolon, ges ingen radframmatning efter **PRINT**-satsen.

Då en rad är fullskriven, görs en radframmatning, och utskriften fortsätter på nästa rad. För att styra **PRINT**-satsens utskrifter till vissa, bestämda lägen används funktionerna **TAB** och **CUR**. Dessa funktioner ger information om vilken position utskriften ska placeras i.

En **PRINT**-sats utan argument ger vagnretur och radframmatning, d.v.s. en blankrad.

Exempel

Ex. 1

```
110 PRINT X;Y;"5"
120 PRINT
130 PRINT "Värde= ";X3,"SAM2= ";A+2
140 END
```

Ex. 2

```
10 LET A=5
20 LET B=2
30 PRINT A,B,A+B,A*B,A-B,B-A,A/B
40 END
```

Ex. 3

```
100 PREPARE "DR0:SKRFIL.DAT" AS FILE 3
110 PRINT #3,A
```

skapar filen SKRFIL.DAT på skivan i DR0: och skriver värdet av variabeln A i filen.

PRINT USING

Format

PRINT USING "formatsträng"; "data"/variabel [,"data"/variabel,...]

Där <"formatsträng">,med citationstecken, är sammansatt av speciella formateringstecken. Dessa tecken, som beskrivs nedan, bestämmer uppställning och format för de strängar och tal, som ska skrivas.

Funktion

Skriver tal och strängar med angivet format.

STRÄNGFÄLT

När **PRINT USING** används för att formatera strängar, kan man välja mellan tre styrtecken för att ange formatet:

"!"

Anger att bara det första tecknet i strängen ska skrivas ut.

"Ö n blankaÖ"

Anger, att strängens 2+n första tecken ska skrivas ut. <n blanka> är n stycken mellanslag. Om enbart bokstäverna "ÖÖ" (versaler) skrivs, kommer två tecken att skrivas ut o.s.v. Om strängen är längre än det angivna antalet tecken i fältet, skrivs de första tecknen ut och resten ignoreras. Om det angivna fältet är större än strängen, blir strängen vänsterjusterad i fältet. Resten av fältet fylls med blanka tecken (mellanslag).

Exempel

```
10 A¤="Se" : B¤="upp"
20 PRINT USING "ÖÖ";B¤
30 PRINT USING " ÖÖ";A¤,B¤
40 PRINT USING " Ö Ö ";A¤,B¤,"!!"
RUN
up
Se up
Se upp !!
```

"&" Anger ett strängfält med variabel längd. När ett fält specificeras med "&", skrivs strängen ut precis som den ser ut.

Exempel

```
10 A¤="Se": B¤="upp"
20 PRINT USING "!";A¤;
30 PRINT USING "&";B¤
RUN
Supp
```

NUMERISKA FÄLT Följande specialtecken kan användas för att formatera ett numeriskt fält.

Nummertecknet # används för att beteckna varje sifferposition. Alla sifferpositioner fylls ut vid utskrift. Om det tal, som ska skrivas, har färre siffror än det reserverade antalet positioner, högerjusteras talet i fältet och föregås av mellanslag.

En decimalpunkt kan sättas i valfri position i fältet. Om formatsträngen anger, att en siffra ska stå före decimalpunkten, skrivs siffran alltid ut (ev. 0). Talen avrundas vid behov.

PRINT USING "##.##";.78 ger utskriften
0.78

PRINT USING "###.##"; 987.654 ger utskriften
987.65

PRINT USING "##.## ";10.2,5.3,66.789,.234
10.20 5.30 66.79 0.23

I det sista exemplet avslutas formatsträngen med två mellanslag, för att de utskrivna värdena inte ska hamna alldeles intill varann på raden.

+ Ett plustecken i början eller slutet av formatsträngen medför, att talets tecken (+ eller -) alltid skrivs ut före eller efter talet.

- Ett minustecken i slutet av formatsträngen medför, att de negativa talens minustecken placeras efter talet i stället för före, som är det normala.

PRINT USING"+##.## ";-68.95,2.4,55.6,-.9
-68.95 +2.40 +55.60 -0.90

PRINT USING"##.##- ";-68.95,22.449,-7.01
68.95- 22.45 7.01-

** Två asterisker i början av formatsträngen medför, att blanka tecken i början av ett tal fylls med asterisker. Tecknen** reserverar också plats för två siffror.

PRINT USING"** #.# ";12.39,-0.9,765.1
* 12.4 *-0.9 765.1

¤¤

Två ¤¤ gör att ett valutatecken skrivs omedelbart till vänster om talet. ¤¤ reserverar också två platser, varav den ena används för ¤ i utskriften. ¤¤ kan inte användas vid exponentialformat. Negativa tal måste ha minustecken i slutet av talet. Denna funktion används mest med amerikansk teckenuppsättning, där ¤ motsvaras av $ (dollartecken).

```
PRINT USING "¤¤###.## ";456.78
¤456.78
```

**¤

¤ i början av en formatsträng kombinerar verkan av och ¤¤. Mellanslag i början av ett tal fylls med* och ¤-tecken skrivs före talet.**¤ reserverar tre positioner, varav en används för ¤-tecknet.

```
PRINT USING"**¤###.## ";2.34
**¤2.34
```

Ett kommatecken till vänster om decimalpunkten i en formatsträng medför, att ett mellanslag skrivs som avgränsare före var tredje siffra räknat från decimalpunkten. Ett kommatecken i slutet av formatsträngen skrivs ut som en del av strängen. Detta kommatecken är en avgränsare mellan två tal. Kommatacknet har ingen inverkan, om det används med exponentialformat.

```
PRINT USING"####,.## ";1234.5
1 234.50

PRINT USING"#####.##,";1234.5
1234.50,
```

ÜÜÜÜ

Fyra versala Ü kan placeras i slutet av formatsträngen för att ange exponentialformat. De fyra tecknen reserverar utrymme för E+xx. Decimalpunkten kan sättas i valfritt läge. Exponenten anpassas efter det valda formatet. Om man inte särskilt anger, var + eller - ska placeras, används en sifferposition i början av talet som teckenposition. I denna position skrivs antingen ett mellanslag eller ett minustecken.

```
PRINT USING "##.##ÜÜÜÜ";234.56
2.35E+02

PRINT USING ".####ÜÜÜÜ";-888888
-.8889E+06

PRINT USING "+.##ÜÜÜÜ";123
+.12E+03
```

_

Ett understrykningstecken i formatsträngen gör, att nästa tecken skrivs ut som det står i strängen.

```
PRINT USING"_!###.##_!";12.34
!12.34!
```

Om man vill skriva ut ett _- tecken, ska formatsträngen innehålla "__".

%

Om talet, som ska skrivas, är större än det format, som reserverats, skrivs ett procenttecken ut före talet. Procenttecken skrivs också, om en avrundning får talet att bli större än det reserverade fältet.

```
PRINT USING"##.##";111.22
%111.22

PRINT USING".##";.999
%1.00
```

PUT

Format

PUT #filnummer,strängvariabel

Där #<filnummer> ger det filnummer, som definierats med någon av satserna **OPEN** eller **PREPARE.**
<strängvariabel > kan vara en strängvariabel eller ett strängutryck.

Funktion

Skriver en strängvariabel i binär form.

Användning

Se avsnitt 3.5.

RANDOMIZE

Format

RANDOMIZE

Funktion

Sätter ett slumpmässigt startvärde för funktionen **RND** (slumptalsgeneratorn).

Användning

Ska stå före det första anropet av slumptalsgeneratorn **RND** i programmet. **RANDOMIZE** får slumptalsgeneratorn att ge olika värden för varje gång programmet körs, genom att **RND** får ett nytt startvärde då **RANDOMIZE** påträffas.

VARNING

Bör bara användas en gång i varje program.

READ

Format

READ variabel [,variabel,...]

Funktion

Utgör, tillsammans med **DATA**-satser, ett sätt att tilldela variabler värden.

Användning

Instruktionen **READ** tilldelar variablerna i listan i tur och ordning värden från **DATA**-satserna. Innan programmet körs, har BASIC sammanfört alla data från **DATA**-satserna till ett datablock. Varje gång en **READ**-sats exekveras, hämtas nästa data från datablocket.
READ används tillsammans med **DATA.**

Om man behöver använda samma data mer än en gång i ett program, kan man återställa datapekaren inom blocket med instruktionen **RESTORE** eller **ON RESTORE** Se vidare under dessa instruktioner.

Exempel

Ex. 1

```
100 READ A,B,C,D,X1,X2
|
|
150 DATA 3,6,1.8
200 DATA 6.83E-3,-86.4,3.14
```

När programmet genomlöps, får variablerna följande värden:

```
A=3
B=6
C=1.8
D=6.83E-3
X1=-86.4
X2=3.14
```

Ex. 2

```
10 READ A¤,B¤,C¤
20 PRINT A¤,B¤,C¤
30 DATA OSKAR, MATS, """STEN"""
RUN
OSKAR        MATS         "STEN"
```

Observera

Om ett kommatecken, citationstecken eller apostrof ska ingå i en sträng, måste det stå inom citationstecken.

REM

Format

REM text
! text

Där <text> får innehålla alla tecken på tangentbordet. BASIC-tolken ignorerar allt, som står efter **REM** eller ! på en rad.

Funktion

Inleder kommentar i program.

Användning

Man bör lägga in täta kommentarer i sina program, för att göra dem mera lättlästa och för att underlätta underhåll.

Exempel

```
10 REM Detta program beräknar
20 ! medelvärdet.
2010 !***** Skriver ut en tabellrad*****
```

RESTORE

Format

RESTORE [radnummer]

Funktion

Gör det möjligt att använda innehållet i **DATA**-satser flera gånger.

Exempel

Ex. 1

```
60 RESTORE
```

Ställer datapekaren till början av den första **DATA**-satsen i programmet.

Ex. 2

```
50 RESTORE 100
```

Ställer datapekaren till **DATA**-satsen med radnummer 100.

RESUME

Format

RESUME [radnummer]

Funktion

Återhopp från felhanteringsrutin.

Användning

När felet är åtgärdat kan man fortsätta att köra programmet, genom att placera en **RESUME**-sats i slutet av felhanteringsrutinen.

Om man vill hoppa in på någon annan rad i programmet anges radnumret i **RESUME**-satsen.

Exempel

```
2000 RESUME
2010 RESUME 100
```

Rad 2000 ger återhopp till den rad, där felet genererades. Rad 2010 ger återhopp till rad 100.

RETURN

Format

RETURN [variabel]

Funktion

Ger återhopp från subrutin eller flerradiga funktioner.

RETURN ger återhopp från subrutin till satsen närmast efter anropet.

RETURN <variabel> ger återhopp från funktionen med funktionsvärdet <variabel>.

Användning

Se **GOSUB** och **DEF FN.**

SINGLE

Format

SINGLE

Funktion

Ändrar alla variabler och uttryck, som är flyttal, till enkel precision (7 siffror).

Användning

Deklarationen **SINGLE** måste göras innan variablerna används och kan inte ändras, när programmet väl startats med **RUN.** Om en rad har ändrats (editerats) eller om kommandot **CLEAR** har använts, kan man ändra **SINGLE** till **DOUBLE** eller tvärtom.
Om inget annat deklareras, gäller **SINGLE.**

Observera

Det är inte tillåtet att blanda **SINGLE** och **DOUBLE** i samma program.

STOP

Format **STOP**

Funktion Stoppar programexekveringen.

Användning Instruktionen **STOP** stoppar exekveringen av programmet. Variablernas värden kvarstår och inga filer stängs. **STOP**-instruktionen används med fördel vid felsökning. **STOP**-satser får finnas på flera ställen i ett program. Följande utskrift erhålls:.

```
STOP IN LINE radnummer
```

Observera Programexekveringen kan fortsättas med något av kommandona **CON** eller **GOTO.**

TRACE

Format **TRACE** [# filnummer]

Funktion Skriver ut radnumren på de programrader,som genomlöps.

Användning Vid felsökning i program, för att spåra hur programmet utförs.

Exempel

```
100 OPEN "PR:" AS FILE 1%
110 A=12.345
115 TRACE #1%
120 B=123
125 IF A=0 THEN STOP
130 C%=B
135 X=A*2
140 D1%=A
145 NOTRACE
150 PRINT #1% A,B,C%,D1%,X
160 CLOSE 1%
170 END
RUN
```

Följande utskrift erhålls på skrivare:
120 125 130 135 140 145
12.345 123 123 12 24.69

WEND

Format **WEND**

Funktion **WEND** avslutar en loop, som inleds av **WHILE.**

Användning Se **WHILE.**

WHILE

Format **WHILE** uttryck

Funktion Anger villkor för uthopp ur en programloop.

Användning I programloopar där de värden,som testas i uthoppsvillkoret, ändras då loopen genomlöps. Jämför med **FOR**-loopar, där uthoppsvillkoret automatiskt uppnås, oavsett innehållet i loopen.

Det kan vara önskvärt, att utföra loopen tills ett visst värde har uppnåtts.

Exempel

```
10 WHILE X<10
20 X=X*X+1
30 WEND
```

Innan loopen genomlöps första gången och vid varje nytt varv undersöks villkoret X<10. Så länge detta är sant, fortsätter iterationen.

10 Funktioner

10.1 Matematiska funktioner

Varje programmerare träffar ofta på diverse, vanliga, matematiska operationer. Deras resultat brukar finnas i matematiska tabellverk. Detta gäller t.ex. sinus, cosinus, kvadratrötter, logaritmer och många andra. Datorn kan beräkna dessa värden snabbt och noggrant. Man har därför valt att bygga in ett antal sådana funktioner i BASIC II. När ett av dessa funktionsvärden behövs i en beräkning, anropas de inbyggda funktionerna t.ex.:

```
SIN(23 * PI/180)
LOG(144)
```

Här följer en tabell över de matematiska funktionerna.

ABS(X)	absolutvärdet av X
ATN(X)	arcustangens för X
COS(X)	cosinus för X (X i radianer)
EXP(X)	e**X, där e = 2.71828 (i enkel precision)
FIX(X)	trunkerat värde av X som **SGN**(X)* **INT(ABS**(X))
HEX¤(X)	omvandlar decimalt tal till hexadecimal sträng
INT(X)	största heltal <= X
LOG(X)	naturliga logaritmen för X
LOG10(X)	tiologaritmen för X
MOD(X,Y)	resten vid heltalsdivisionen X/Y
OCT¤(X)	omvandlar decimalt tal till oktal sträng
PI	konstant med värdet 3.14159 (enkel precision)
RND(X)	slumptal mellan 0 och 0.999999. **RND** genererar samma följd av slumptal varje gång ett program körs, om inte en **RANDOMIZE**-sats finns före **RND** i programmet.
SGN(X)	0 för X = 0, -1 för X < 0, +1 för X > 0
SIN(X)	sinus för X (X i radianer)
SQR(X)	kvadratroten ur X
TAN(X)	tangenten för X (X i radianer)

ABS

Format **ABS**(argument)

Funktion Ger absolutvärdet av argumentet.

Exempel
```
10 Y=ABS(-3.1)
```

Detta ger Y=3.1

ATN

Format **ATN**(argument)

Funktion Ger arcustangens för argumentet i radianer.

Exempel
```
10 Y=ATN(PI/2)
```

Detta ger Y=1

COS

Format **COS**(argument)

Funktion Ger cosinus för argumentet. Argumentet ska anges i radianer.

Exempel
```
10 Y=COS(0)
```

Detta ger Y=1

EXP

Format **EXP**(argument)

Funktion Ger e** argumentet, där e = 2.71828 (i enkel precision).

Exempel
```
10 Y=EXP(1)
```

Detta ger Y=2.71828

FIX

Format **FIX**(argument)

Funktion Ger trunkerat värde av argumentet (X) d.v.s.
SGN(X) * **INT**(**ABS**(X)).

Exempel
```
10 Y%=FIX(-.5)
```

Detta ger Y=0

Observera Jämför **INT.**

HEX¤

Format **HEX¤**(argument)

Funktion Omvandlar decimalt tal till hexadecimal sträng

Exempel
```
10 Y¤=HEX¤(255)
```

Detta ger Y¤=''FF''

INT

Format **INT**(argument)

Funktion Ger värdet av det största heltal, som är mindre än eller lika med argumentet. Jämför **FIX.**

Användning **INT** kan användas, då man vill ha ett tal avrundat. Man använder då **INT**(X+.5)
Funktionen **INT** kan också användas vid avrundning till önskat antal decimaler enligt följande:

INT(X∗10∗∗ D%+.5)/10 ∗∗ D%

där D% är antalet decimaler, som önskas.

Om talet är negativt, ger **INT** det största heltal, som är mindre än talet självt.

Exempel Ex. 1
```
10 Y=INT(34.67)
```
Detta ger Y=34

Ex. 2
```
10 Y=INT(34.67+.5)
```
Detta ger Y=35

Ex. 3
```
10 Y=INT(-23.15)
```
Detta ger Y=-24

LOG

Format **LOG**(argument)

Funktion Ger naturliga logaritmen för argumentet.

Exempel
```
10 Y=LOG(2)
```
Detta ger Y=.693147

LOG10

Format **LOG10**(argument)

Funktion Ger tiologaritmen för argumentet.

Exempel
```
10 Y=LOG10(10)
```
Detta ger Y=1

MOD

Format **MOD**(argument1,argument2)

Funktion Ger resten vid heltalsdivision av argumenten.

Exempel
```
10 Y=MOD(22,4)
```
Detta ger Y=2

OCT¤

Format **OCT¤**(argument)

Funktion Omvandlar decimalt tal till oktal sträng.

Exempel
```
10 Y¤=OCT¤(59)
```
Detta ger Y¤="73"

PI

Format **PI**

Funktion Konstant med värdet 3.14159 (enkel precision)

Exempel
```
10 Y=2 * PI
```
Detta ger Y=6.28318

RND

Format **RND**

Funktion Ger ett slumptal mellan 0 och 0.999999. **RND** genererar samma följd av slumptal varje gång ett program körs, om inte en **RANDOMIZE**-sats finns före **RND** i programmet.

Exempel Ex. 1
```
10 Y=RND
```
Ex. 2
```
10 Y=(B-A) * RND+A
```
Y tilldelas ett slumptal i intervallet (A,B).

SGN

Format **SGN**(argument)

Funktion Funktionen **SGN**(X) har värdet +1 om X är positivt, 0 om X är 0 och -1 om X är negativt.

Exempel Ex. 1

```
10 Y=SGN(3.42)
```

Detta ger Y=1

Ex. 2

```
20 Y=SGN(-42)
```

Detta ger Y=-1

Ex. 3

```
10 Y=SGN(23-23)
```

Detta ger Y=0

SIN

Format **SIN**(argument)

Funktion Ger sinus för argumentet (argumentet i radianer).

Exempel

```
10 Y=SIN(PI/2)
```

Detta ger Y=1

SQR

Format **SQR**(argument)

Funktion Ger kvadratroten ur argumentet.

Exempel

```
10 Y=SQR(121)
```

Detta ger Y=11

TAN

Format **TAN**(argument)

Funktion Ger tangenten för argumentet (argumentet i radianer).

Exempel

```
10 Y=TAN(PI/4)
```

Detta ger Y=1

10.2. Strängfunktioner

Förutom de rent matematiska funktionerna (t.ex. **SIN** eller **LOG**), finns även inbyggda funktioner, som opererar på strängvariabler. Dessa funktioner gör det bl.a. möjligt att utföra aritmetiska operationer på numeriska strängar, sammanfoga två strängar, ta fram en del av en sträng, bestämma antalet tecken i en sträng, skapa den teckensträng, som motsvarar ett visst tal eller tvärtom.

Följande strängfunktioner finns i BASIC II:

ADD¤(A¤,B¤,P%)	numerisk summa A¤+B¤ med P% decimaler
ASCII(A¤)	ASCII-värdet för första tecknet i A¤
CHR¤(X%)	tecknet med ASCII-värdet X%
COMP%(A¤,B¤)	sant eller falskt, numerisk jämförelse
DIV¤(A¤,B¤,P%)	kvoten A¤/B¤ med P% decimaler
INSTR(I%,A¤,B¤)	startposition för strängen B¤ i A¤
LEFT¤(A¤,I%)	de I% första tecknen av A¤
LEN(A¤)	antalet tecken i A¤
MID¤(A¤,P%,K%)	tecken nr P% t.o.m. P%+K% av A¤ tilldelas ett värde
MUL(A¤,B¤,P%)	numerisk produkt A¤*B¤ med P% decimaler
NUM¤(V)	numerisk sträng motsvarande talvärdet V
RIGHT¤(A¤,I%)	de sista tecknen fr.o.m. I% av A¤
SPACE¤(N%)	sträng med N% mellanslag
STRING¤(I%,C%)	sträng med I% tecken med ASCII-kod C%
SUB¤(A¤,B¤,P%)	numerisk differens A¤-B¤ med P% decimaler
VAL(A¤)	numeriska värdet av A¤
A¤+B¤	konkatenerar (sammankedjar) två strängar

ADD¤

Format **ADD¤**(A¤,B¤,P%)

Funktion Ger numerisk summa A¤ + B¤ med P% decimaler.

Exempel:

```
10 A¤="123.76"
20 B¤=ADD¤(A¤,"957.63359",3)
```

Observera — Beräkningar med ASCII-aritmetik kan göras med upp till 125 tecken.

ASCII

Format — **ASCII**(A¤)

Funktion — Ger ett heltal, som är lika med ASCII-värdet (decimalt) för första tecknet i A¤.

Exempel

```
10 A = ASCII("T")
```

Detta ger A = 84

CHR¤

Format — **CHR¤**(argument [,argument, ...])

Funktion — Ger en teckensträng, som motsvarar argumentens ASCII-värden.

Exempel

```
PRINT CHR¤(65)
```

Detta ger utskriften A.

COMP%

Format — **COMP%**(A¤,B¤)

Funktion — Ger värdet -1, 0 eller 1 som resultat av en numerisk jämförelse av två numeriska strängar. Funktionsvärdet är -1 om A¤ < B¤, 0 om A¤=B¤ och 1 om A¤ > B¤.

Exempel

```
30 A¤="123.456" : B¤="12.8907"
40 T%=COMP%(A¤,B¤)
50 PRINT T%
60 PRINT COMP%(B¤,A¤)
100 END
RUN
1
-1
```

DIV¤

Format — **DIV¤**(A¤,B¤,P%)

Funktion — Ger kvoten A¤/B¤ avrundad till P% decimaler

Exempel

```
100 LET C¤="3.5"
110 V9¤=DIV¤(C¤,"1.7777",3%)
120 PRINT V9¤
200 END
RUN
1.969
```

Observera — Beräkningar med ASCII-aritmetik kan göras med upp till 125 tecken.

INSTR

Format **INSTR**(N%,A¤,B¤)

Funktion Söker efter strängen B¤ i A¤ med början i position N%. Ger värdet 0, om B¤ ej finns i den undersökta delen av A¤, annars värdet för den position i A¤, där B¤ börjar. Positionen räknas från strängens början. Första tecknet står i position 1.

Exempel

```
10 A¤="AaBbCcDdEeFf"
20 PRINT INSTR(5%,A¤,"eF")
30 END
RUN
10
```

LEFT

Format **LEFT**[¤](A¤,I%)

Funktion Ger de I% första tecknen av strängen A¤.

Exempel

```
10 D2¤='ABCDEFGHIJKLMNOPQRSTUVXYZÅÄÖ'
20 T8¤=LEFT¤(D2¤,6%)
40 PRINT T8¤
100 END
RUN
ABCDEF
```

LEN

Format **LEN**(A¤)

Funktion Ger antalet tecken i strängen A¤, d.v.s. stränglängden (inklusive mellanslag).

Exempel

```
10 S¤="MOTALA"
20 PRINT LEN(S¤)
200 END
RUN
6
```

MID

Format **MID**[¤](A¤,P%,K%)

Funktion Tillordnar tecken nr P% t.o.m. P%+K%-1 av A¤ det nya värdet, d.v.s. byter tecken i strängen.

Exempel

```
10 A¤="ABCDEFGHI"
20 MID¤(A¤,6%,2%)="MM"
30 PRINT A¤
60 END
RUN
ABCDEMMHI
```

MID

Format **MID**[¤](A¤,P%,K%)

Funktion Ger den delsträng av A¤, som börjar i position P% och är K% tecken lång, d.v.s. tecknen nr P% t.o.m. P%+K%-1.

Exempel

```
200 W¤="radframmatning"
210 A2¤=MID¤(W¤,8%,3%)
220 PRINT A2¤
500 END
RUN
mat
```

MUL¤

Format **MUL¤**(A¤,B¤,P%)

Funktion Ger produkten A¤ * B¤ med P% decimaler.

Exempel

```
10 LET A¤="12345.6789"
20 LET B¤="987.54321"
30 Y¤=MUL¤(A¤,B¤,6%)
40 PRINT Y¤
50 END
RUN
12191891.370535
```

NUM¤

Format **NUM¤**(argument)

Funktion Ger den numeriska sträng, som motsvarar argumentet. Strängen skrivs eligt följande: positivt tal - teckenposition markeras inte, negativt tal - minustecken skrivs ut.

Exempel

```
10 PRINT NUM¤(345709702134)
20 END
RUN
3.45709E+12
```

RIGHT

Format **RIGHT**[¤](A¤,N%)

Funktion Ger de sista tecknen i A¤, fr.o.m position N%.

Exempel

```
10 M¤="ABCDEFGHIJKL"
20 H¤=RIGHT¤(M¤,7%)
30 PRINT H¤
90 END
RUN
GHIJKL
```

SPACE¤

Format **SPACE¤**(N%)

Funktion Ger en sträng, som består av N% mellanslag.

Exempel

```
10 Y¤=SPACE¤(10)
```

Detta ger en sträng (Y¤) med 10 blanka.

STRING¤

Format **STRING¤**(I%,K%)

Funktion Ger en sträng med I% st ASCII-tecken, d.v.s. strängen har längden I% och består av likadana tecken, vars ASCII-värde är K%.

Exempel

```
10 G5¤=STRING¤(4%,33%)
20 PRINT G5¤
30 END
RUN
!!!!
```

SUB¤

Format **SUB¤**(A¤,B¤ ,P%)

Funktion Ger den aritmetiska differensen A¤-B¤ för de numeriska strängarna A¤ och B¤ med P% decimaler.

Exempel

```
10 LET H¤="9876.54321"
20 PRINT SUB¤(H¤,'98.76',5%)
30 END
RUN
9777.78321
```

Observera Beräkningar med ASCII-aritmetik kan göras med upp till 125 tecken.

VAL

Format **VAL**(A¤)

Funktion Beräknar numeriska värdet av den numeriska strängen A¤. En numerisk sträng får innehålla siffror, +, -, . och E. Resultatet ges i form av ett flyttal.

Exempel

```
330 V4=VAL("14.3E-5")
340 PRINT V4
400 END
RUN
.000143
```

A¤+B¤

Format A¤+B¤

Funktion Sammanfogar (konkatenerar) strängar.

Exempel

```
10 D¤="God"
20 S¤="Jul"
30 A¤=D¤+" "+S¤
```

Detta ger A¤="God Jul"

10.3 Övriga funktioner

CALL(A%,(D%)) anropar maskinspråksrutiner.

CUR(M%,N%) sätter markören på rad M%, position N%

CVT%¤(X%)
CVT ¤%(X¤) byter typ på variabel från heltal till sträng och vice versa

CVTF¤(X)
CVT¤ F(X¤) byter typ på variabel från flyttal till sträng och vice versa

ERRCODE ger värdet av senast genererade felkod

FN användardefinierad funktion

INP(I%) ger datavärde från inport I%

OUT skickar data till utport

PEEK(I%) ger minnesinnehållet i adress I%

PEEK2(I%) **PEEK(I%)+256*PEEK**(I%+1%)

POKE skriver i angiven minnesadress

SWAP%(N%) första och andra byten av N% skiftar plats

SYS(I%) ger systemstatus

TAB(I%) flyttar skrivhuvud eller markör till position I% på raden

TIME¤ ger datum och tid

VAROOT(X) ger adressen till variabeln X

VARPTR(X) ger adressen till värdet för X

CALL

Format — **CALL**(A%[,D%])

Funktion — Anropar maskinspråksrutiner med början i adress A% (decimalt). Ger ett funktionsvärde från Z80-processorns HL-register vid återhoppet till BASIC. D%, om det anges, läggs i Z80-processorns DE-register vid anropet.

VARNING — Detta är en maskinorienterad funktion, som endast bör användas vid avancerad programmering. Eftersom den adresserar processorn direkt, kan en körning förstöras, om **CALL** används på fel sätt.

CUR

Format — **CUR**(R%,N%)

där R% (rad) finns i intervallet 0 - 23 och N% (position) i intervallet 0 - 39/79.

Funktion — Sätter markören på rad R%, position N%

Användning — Vid utskrifter eller i samband med grafik.

Exempel

```
10 PRINT CUR(12,20)"Text"
```

CVT

Format — **CVT%¤**(heltalsvariabel)
CVT¤%(strängvariabel)

CVTF¤(flyttalsvariabel)
CVT¤F(strängvariabel)

Funktion — Lagrar tal som strängar respektive återskapar talen.

Användning — CVT-funktionen används för att spara utrymme på skiva. Numeriska värden, som lagras på flexskiva, tar lika stor plats, som när de skrivs ut med **PRINT**. Heltal kräver upp till sex tecken, flyttal i enkel precision (**SINGLE**) tolv tecken och flyttal i dubbel precision (**DOUBLE**) behöver tjugotvå tecken. Varje tecken lagras i en byte. Med hjälp av funktionen **CVT** (av engelska convert) kan man spara dessa data i respektive 2, 4 och 8 bytes.

Exempel

```
10 PREPARE "TAL.DAT" AS FILE 1
20 I%=15973%
30 PUT #1,CVT%¤(I%)
40 CLOSE 1
```

Heltalet I% har nu lagrats på filen TAL.DAT i två bytes. För att återskapa talet används följande program användas:

```
10 OPEN "TAL.DAT" AS FILE 1
20 GET #1,A¤ COUNT 2
30 I%=CVT¤%(A¤)
40 CLOSE 1
```

Nedan följer ett exempel på hur ett flyttal kan sparas. Flyttalet i exemplet kan vara antingen enkel eller dubbel precision:

```
10 DIM A(100)
20 PREPARE "TAL2.DAT" AS FILE 1%
30 FOR I%=1% TO 100%
40 PUT #1%,CVTF¤(A(I%))
50 NEXT I%
60 CLOSE 1%
```

Nästa exempel visar, hur talet i ovanstående exempel återskapas.

```
10 DIM A(100): L%=LEN(CVTF¤(0))
20 OPEN "TAL2.DAT" AS FILE 1%
30 FOR I%=1% TO 100%
40 GET #1%,A¤ COUNT L%: A(I%)=CVT¤F(A¤)
50 NEXT I%
60 CLOSE 1%
```

Observera — Man använder **LEN(CVTF¤(0))** för att undersöka om enkel eller dubbel precision används.

ERRCODE

Format — **ERRCODE**

Funktion — Ger värdet av senast genererade felkod. Är 0 om inga fel indikerats.

FN

Format — **FN**identifierare[%/¤] [[parameter [,parameter,...]]]

där <identifierare> utgör funktionens namn.

Funktion — Anrop av användardefinierad funktion. Jämför instruktionen **DEF FN**.

INP

Format — **INP**(I%)

Funktion — Ger datavärdet från inport I%.

VARNING — Detta är en maskinorienterad funktion, som ska användas enbart vid avancerad programmering.

OUT

Format — **OUT** port,data [port,data, ...]

där portnummer och data anges i decimal form.

Funktion — Används för att adressera utportar vid utmatning av data.

VARNING — Detta är en maskinorienterad instruktion för avancerad programmering. Denna instruktion och instruktionen **INP** ger användaren tillgång till I/O-funktionerna i ABC 800 och dess I/O-buss.

Observera — För att välja en I/O-kanal måste man alltid först ge instruktionen **OUT** för denna kanal. I/O-kanalen i fråga förblir sedan tillgänglig, tills ett nytt val görs med **OUT.**

PEEK

Format — **PEEK**(I%)

Funktion — Ger minnesinnehållet i adress I%.

VARNING — Denna funktion är avsedd att användas vid avancerad programmering.

PEEK2

Format — **PEEK2**(B0%)

Funktion — Läser innehållet i två bytes enligt följande:
J%=**PEEK**(B0%)+256∗ **PEEK**(B0%+1%)

VARNING — Denna funktion är avsedd att användas vid avancerad programmering.

POKE

Format — **POKE** adress,data [,data, ...]

Där <adress> anges decimalt. Om flera <data> anges, räknas adressen upp för varje nytt värde.

Funktion — Används för att ladda in ett värde i en minnescell.

Användning — **POKE** används huvudsakligen då BASIC samarbetar med maskinspråksrutiner.

VARNING — Denna instruktion är maskinorienterad och ska endast användas vid avancerad programmering. Om den används på fel sätt, kan innehållet i datorns minne förstöras.

SWAP%

Format **SWAP%**(N%)

Funktion Första och andra byten av N% skiftar plats.

VARNING Denna funktion är avsedd att användas vid avancerad programmering.

SYS

Format **SYS**(I%)

Funktion Ger systemstatus enligt lista nedan.

Användning Programmeraren får tillgång till följande systemstatus:

SYS(2) Totalt minnesutrymme
SYS(3) Programmets storlek
SYS(4) Kvarvarande minnesutrymme
SYS(5) Tangentbordsflagga. Nollställs med **GET, INPUT** eller INPUT LINE.
SYS(6) Lägger tillbaka senast inlästa tecken i tangentbordsbufferten.
SYS(11) Programmets startadress
SYS(12) Variabelrot

Exempel

```
PRINT SYS(3)
```

Detta ger utskrift av programmets storlek.

TAB

Format **TAB**(I%)
där I% kan ha värdet 1 - 40/80

Funktion Flyttar skrivhuvud eller markör till position I% på raden.

Exempel

```
10 PRINT TAB(20)"Data"
```

TIME¤

Format **TIME¤**

Funktion Ger tidsangivelse år-mån-dag tim.min.sek

Användning Datorns klocka kan sättas med följande program:

```
10 PRINT CHR¤(12%)
20 PRINT "** ABC800 Sätt klockan! ** "
30 INPUT "Datum : ÅÅ,MM,DD ";Y%,M%,D% !Mata in datum
40 INPUT "Tid : HH,MM,SS ";H%,M1%,S% !Mata in tid
50 POKE -17,Y%,M%,D%,H%,M1%,S% !Skriv tid i minnet
60 PRINT CUR(12,12);TIME¤ !Skriv tiden på bildskärm
70 GOTO 60
80 END
```

VAROOT

Format	**VAROOT**(variabel)
Funktion	Ger adressen till en tabell, som innehåller uppgifter om en variabel.
VARNING	Denna funktion är avsedd att användas vid avancerad programmering.

VARPTR

Format	**VARPTR**(variabel)
Funktion	Ger adressen till en variabels värde.
VARNING	Denna funktion är avsedd att användas vid avancerad programmering.

11 Grafik och färger

11.1 Allmänt

ABC800C har grafik motsvarande standarden för Teletext. I grafisk mod tolkas varje utmatat tecken som en figur, bildad av sex grafiska punkter.

Vid utskrift av text eller grafik på bildskärmen styr man val av färg m.m. genom speciella argument i den aktuella **PRINT**-satsen. Satsen påverkar en rad åt gången. Varje argument lägger ut ett styrtecken på skärmen. Dessa styrtecken syns inte, men tar upp en position. De kan skrivas över med en bakgrundsfärg, om styrargumenten ges i lämplig ordningsföljd.

Följande färger är tillgängliga:

Röd (**RED**)
Grön (**GRN**)
Gul (**YEL**)
Blå (**BLU**)
Magenta (**MAG**)
Cyan (**CYA**)
Vit (**WHT**)

Nedan visas en lista över tillgängliga tecken i ABC800. Tabellen ger ASCII-koden för varje tecken, dess betydelse i teckenmod och i grafisk mod. Ett sätt att planera en bild är att rita den på en grafikkarta och sedan mata in lämpliga data till programmet.

När Du ritat bilden på grafikkartan, skriver Du en rad i taget. Glöm inte att ta hänsyn till styrtecken, om Du arbetar med varierande styrargument för Din bild.

Observera att versaler (stora bokstäver) inte förändras i grafisk mod. Grafiska tecken och versaler kan alltså blandas fritt.

I grafisk mod, får man 72 grafiska rader (0-71) med 78 grafiska positioner på varje rad (0-77).

ASCIIKod	T	G	ASCIIKod	T	G	ASCIIKod	T	G	ASCIIKod	T	G
32	Blank		56	8		80	P	P	104	h	
33	!		57	9		81	Q	Q	105	i	
34	"		58	:		82	R	R	106	j	
35	#		59	;		83	S	S	107	k	
36	¤		60	<		84	T	T	108	l	
37	%		61	=		85	U	U	109	m	
38	&		62	>		86	V	V	110	n	
39	'		63	?		87	W	W	111	o	
40	(		64	É	É	88	X	X	112	p	
41	)		65	A	A	89	Y	Y	113	q	
42	*		66	B	B	90	Z	Z	114	r	
43	+		67	C	C	91	Ä	Ä	115	s	
44	,		68	D	D	92	Ö	Ö	116	t	
45	-		69	E	E	93	Å	Å	117	u	
46	.		70	F	F	94	Ü	Ü	118	v	
47	/		71	G	G	95	_	_	119	w	
48	0		72	H	H	96	é		120	x	
49	1		73	I	I	97	a		121	y	
50	2		74	J	J	98	b		122	z	
51	3		75	K	K	99	c		123	ä	
52	4		76	L	L	100	d		124	ö	
53	5		77	M	M	101	e		125	å	
54	6		78	N	N	102	f		126	ü	
55	7		79	O	O	103	g		127		

Koder tolkade i teckenmod (T) och grafmod (G)

11 Grafik och färger

11.1 Allmänt

ABC800C har grafik motsvarande standarden för Teletext. I grafisk mod tolkas varje utmatat tecken som en figur, bildad av sex grafiska punkter.

Vid utskrift av text eller grafik på bildskärmen styr man val av färg m.m. genom speciella argument i den aktuella **PRINT**-satsen. Satsen påverkar en rad åt gången. Varje argument lägger ut ett styrtecken på skärmen. Dessa styrtecken syns inte, men tar upp en position. De kan skrivas över med en bakgrundsfärg, om styrargumenten ges i lämplig ordningsföljd.

Följande färger är tillgängliga:

Röd (**RED**)
Grön (**GRN**)
Gul (**YEL**)
Blå (**BLU**)
Magenta (**MAG**)
Cyan (**CYA**)
Vit (**WHT**)

Nedan visas en lista över tillgängliga tecken i ABC800. Tabellen ger ASCII-koden för varje tecken, dess betydelse i teckenmod och i grafisk mod. Ett sätt att planera en bild är att rita den på en grafikkarta och sedan mata in lämpliga data till programmet.

När Du ritat bilden på grafikkartan, skriver Du en rad i taget. Glöm inte att ta hänsyn till styrtecken, om Du arbetar med varierande styrargument för Din bild.

Observera att versaler (stora bokstäver) inte förändras i grafisk mod. Grafiska tecken och versaler kan alltså blandas fritt.

I grafisk mod, får man 72 grafiska rader (0-71) med 78 grafiska positioner på varje rad (0-77).

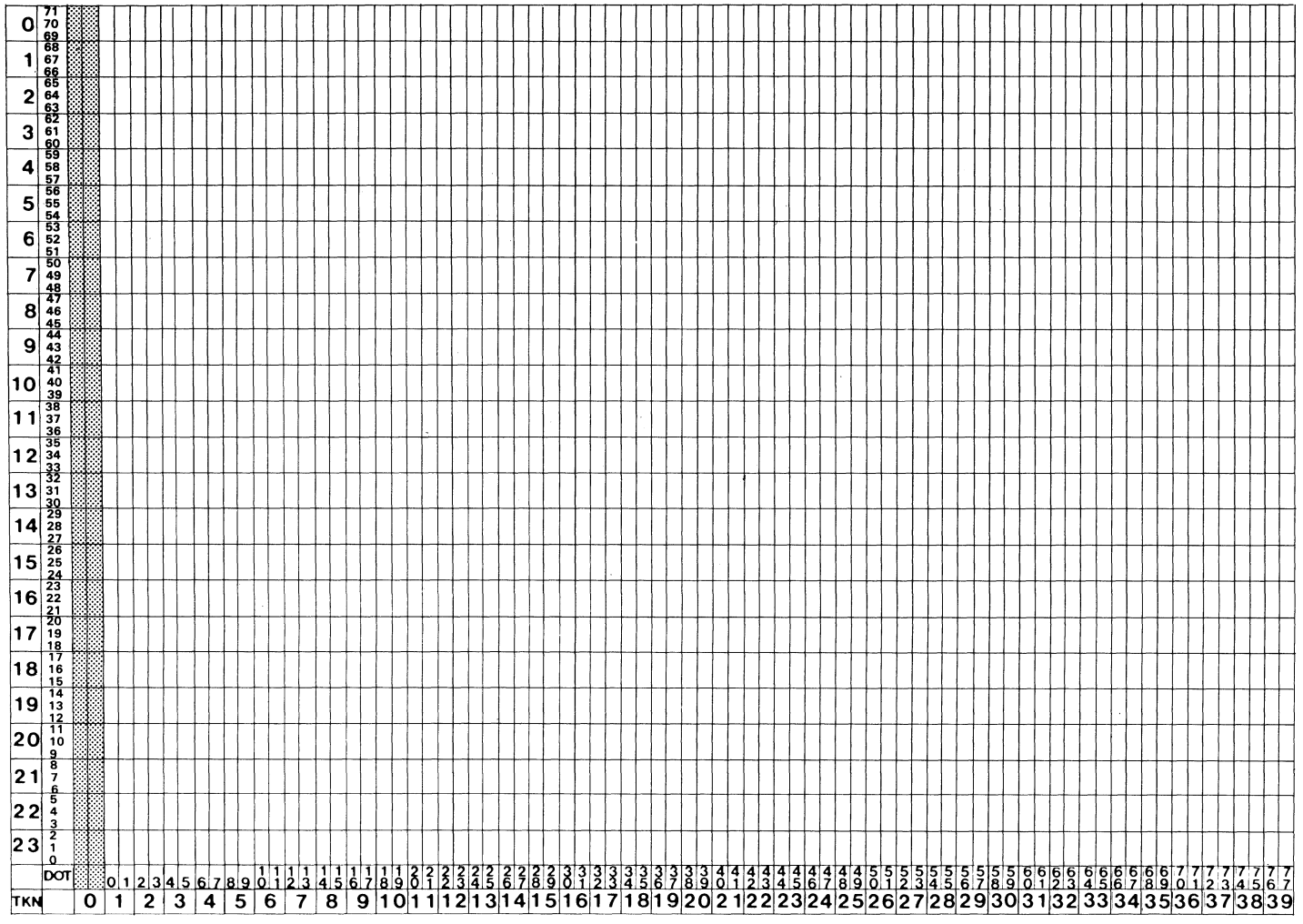
ASCIIKod	T	G	ASCIIKod	T	G	ASCIIKod	T	G	ASCIIKod	T	G
32	Blank		56	8		80	P		104	h	
33	!		57	9		81	Q		105	i	
34	"		58	:		82	R		106	j	
35	#		59	;		83	S		107	k	
36	¤		60	<		84	T		108	l	
37	%		61	=		85	U		109	m	
38	&		62	>		86	V		110	n	
39	'		63	?		87	W		111	o	
40	(		64	É		88	X		112	p	
41	)		65	A		89	Y		113	q	
42	*		66	B		90	Z		114	r	
43	+		67	C		91	Ä		115	s	
44	,		68	D		92	Ö		116	t	
45	-		69	E		93	Å		117	u	
46	.		70	F		94	Ü		118	v	
47	/		71	G		95	–		119	w	
48	0		72	H		96	é		120	x	
49	1		73	I		97	a		121	y	
50	2		74	J		98	b		122	z	
51	3		75	K		99	c		123	ä	
52	4		76	L		100	d		124	ö	
53	5		77	M		101	e		125	å	
54	6		78	N		102	f		126	ü	
55	7		79	O		103	g		127		

Koder tolkade i teckenmod (T) och grafmod (G)

ABC800®

VIDEOGRAFIK

PROGRAM ....



— plats för grafikstyrtecken

11.2 Instruktioner

PRINT

Format

PRINT [**CUR**(R,K)]argument[;argument;...] "text"

Funktion

Används för att skriva text och grafik. Argumenten styr färgval m.m. Ett G i början av färgvalsargument (t.ex. **GRED**) sätter raden i grafisk mod, varvid tecken inom citationstecken tolkas som grafik (se ASCII-tabell). Om **CUR**(R,K) används, börjar utskriften i den angivna positionen R=rad (0-23) och K=position (0-39).

Följande argument finns:

RED, GRN, YEL, BLU, MAG, CYA, WHT	alfanumeriska tecken i färg
GRED,GGRN,GYEL,GBLU, GMAG,GCYA,GWHT	grafik i färg
FLSH, STDY	blinkande resp. fast
NRML, DBLE	normal resp. dubbel höjd
GCON, GSEP	sammanhängande resp. separerad grafik
NWBG, BLBG	ny bakgrund resp. svart bakgrund
GHOL, GREL	håll kvar grafik resp. ta bort grafik
HIDE	dold text/grafik

Följande ordningsföljd gäller för styrtecknen:

PRINT <tecken för bakgrundsfärg> <tecken för ändring av bakgrundsfärg> <styrtecken för textfärg> "Text" <styrtecken för svart bakgrund>

Exempel

10 **PRINT RED NWBG GYEL** "I,6 VOV"
Resultatet blir en gul "hund" på röd bakgrund.

TXPOINT

Format

TXPOINT X,Y[,1/0]
där X går från 0–71 och Y från 0–77

Funktion

Tänder (1 kan utelämnas) eller släcker (0) en grafisk punkt på rad Y i position X.
Observera att origo finns i nedre vänstra hörnet

Exempel

```
10 PRINT CHR$(12)
20 FOR I=0 TO 23
30 PRINT CUR (I,0) GGRN;
40 NEXT I
50 FOR I=0 TO 77
60 TXPOINT I,32+SIN(I/5)*30
70 NEXT I
80 PRINT CUR(0,15) RED FLSH DBLE "SINUS"
90 END
```

Rad 10 - 40 tömmer bildskärmen och sätter den i grafisk mod (grön). Rad 50 - 70 ritar en sinuskurva. Rad 80 skriver SINUS med röd text, blinkande och med dubbel texthöjd.

TXPOINT kan även användas som funktion, d.v.s. kontrollera om en angiven punkt är tänd (-1) eller släckt (0) . **TXPOINT**(X,Y).

SET DOT

Format **SET DOT** R%,K%

där R%= 0-71 och K%= 2-79

Funktion Tänder en grafisk punkt (origo uppe till vänster).

CLR DOT

Format **CLR DOT** R%,K%

där R%= 0-71 och K%= 2-79

Funktion Släcker en grafisk punkt (origo uppe till vänster).

DOT

Format **DOT**(R%,K%)

där R%= 0-71 och K%= 2-79

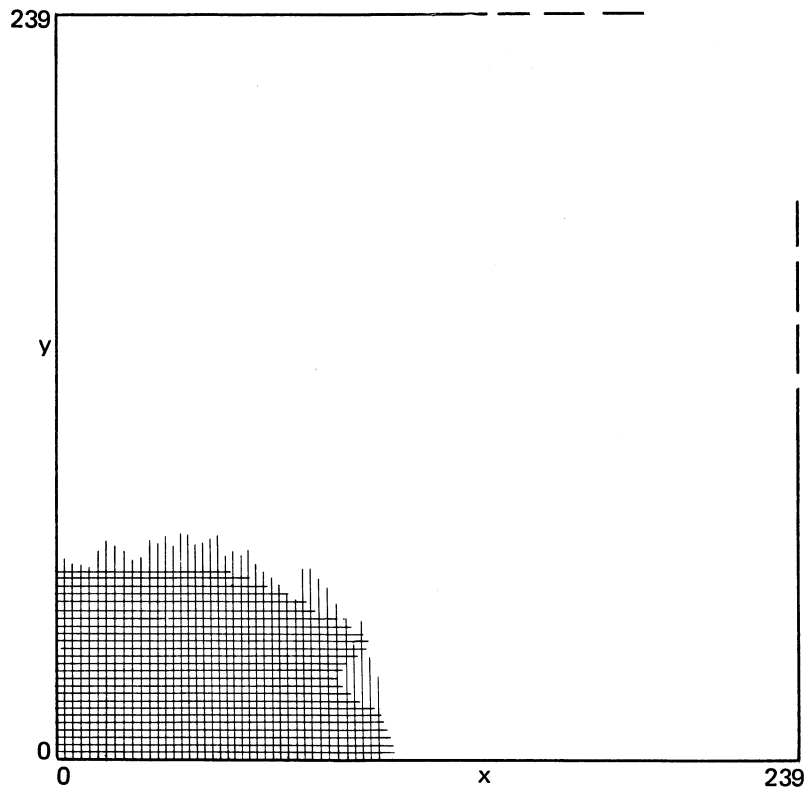
Funktion Ger -1 (sann) om punkten är tänd, annars 0 (falsk).

12 Högupplösningsgrafik

12.1 Allmänt

Högupplösningsgrafiken, som är en option, kan användas med både ABC 800 C och ABC 800 M.

Bildytan är indelad i 240 x 240 bildelement (pixels). Varje sådant bildelement kan adresseras oberoende av de övriga. Till varje bildelement hör två databitar.Dessa databitar kan användas för att välja en av fyra färger. Högupplösningsgrafiken kan visas tillsammans med den ordinarie bilden. Bildens origo ligger i nedre, vänstra hörnet och positionerna är numrerade från 0 till 239.



Bilden ställs in så att följande höjd/breddförhållande erhålls:

	Höjd (mm)	Bredd (mm)	B/H
ABC 810	185 +- 2	225 +- 2	1.2
ABC 815	166 +- 2	250 +- 2	1.5

12.2 Instruktioner

Allmänt gäller att:

- färgnummer ges med en siffra från 0 till 3, där 0 anger bakgrundsfärg. Dess betydelse framgår av färgvalstabellen (avsnitt 12.4).
- färgnummer får utelämnas. I så fall gäller föregående färgnummer.
- startpositionen för bilden väljs med OUT 6,radnummer, där radnummer ligger i intervallet 0 - 255.

FGCTL

Format **FGCTL** färgvalkommando

Där <färgvalkommando> följer färgvalstabellen (avsnitt 12.4).

Funktion Väljer den färgkombination, som ska användas. Varje kombination består av fyra av de tillgängliga färgerna. Svart och vitt behandlas som färger.

FGFILL

Format **FGFILL** x,y [färgnr]

Funktion Fyller en rektangel från föregående position till det läge koordinaterna (x,y) anger.

FGLINE

Format **FGLINE** x,y [färgnr]

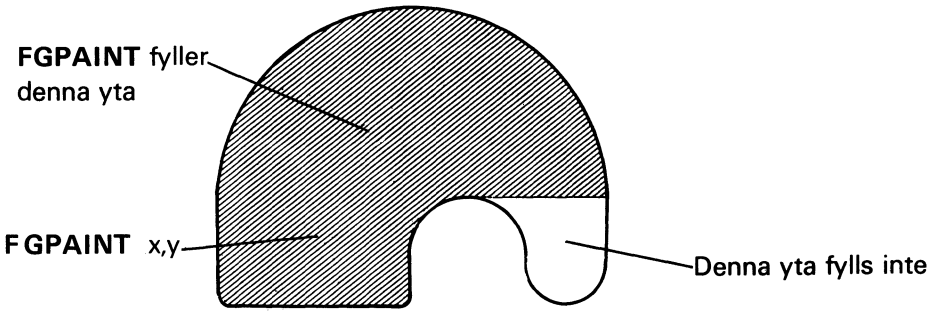
Funktion Ritar en linje från föregående position till det läge, som koordinaterna (x,y) anger.

FGPAINT

Format **FGPAINT** x,y [färgnummer]

Funktion Fyller en sluten yta

Observera Vissa begränsningar finns.



FGPOINT

Format	**FGPOINT** x,y(,färgnr)
Funktion	Sätter ett bildelement (x,y) på skärmen.
Observera	**FGPOINT(x,y) ger färgnumret för bildelementet (x,y).**

FGPOINT

Format	**FGPOINT** (x,y)
Funktion	Ger färgnumret för bildelementet (x,y)

12.3 Animation-mod

I animation-mod arbetar man med två färger.
Tillvägagångssättet är följande:

1. Välj färgvalsgrupp (72-127, 200-255). Färgvalsgrupperna används två och två, t.ex. 72-73, 74-75 o.s.v.

2. Rita bild med färgnr 1 alt 2. Välj samma färg som den bilden ritas på. Detta innebär, att den ritade bilden ej framträder.

3. Byt färgvalsgrupp så att den i punkt 2 ritade bilden framträder.

4. Rita ny bild enligt punkt 2 ovan.

5. Byt färgvalsgrupp, varvid den bild, som ritades i punkt 2, försvinner och den bild, som ritades i punkt 4, visas.

6. Sudda bilden, som ritades i punkt 2, och rita ny bild.

7. Byt färgvalsgrupp, varvid bilden, som ritades i punkt 6, visas.

Upprepa förfarandet enligt punkterna 6 och 7.

För att skydda den bild, som visas, tills man ska växla bild, kan man t.ex. skriva:

```
100 FGLINE 100,100,256*2+1
```

Ovanstående instruktion medför, att en linje ritas från föregående position till punkten 100,100 med färg nr 1. Färg nr 2 är skyddad och skrivs inte över.

12.4 Färgvalstabell

Färgvalskommandot (enligt tabellen nedan) ligger i intervallet 0-255. Värden mindre än 128 betyder, att ordinarie bildminne skrivs ut överlagrat på grafikminnet. Från 128 och uppåt visas grafikminnet. Värden från 72 till 127 och 200 till 255 används vid animation-mod (se 12.3).

B=blå, C=cyan, G=gul, GR=grön, M=magenta, R=röd, S=svart, V=vit

Färgvals-kommando Grafik + text	Färgnr 0	1	2	3	Färgvals-kommando Enbart grafik
0	S	S	S	S	128
1	S	V	V	V	129
2	S	R	GR	G	130
3	S	R	GR	B	131
4	S	R	GR	M	132
5	S	R	GR	C	133
6	S	R	GR	V	134
7	S	R	G	B	135
8	S	R	G	M	136
9	S	R	G	C	137
10	S	R	G	V	138
11	S	R	B	M	139
12	S	R	B	C	140
13	S	R	B	V	141
14	S	R	M	C	142
15	S	R	M	V	143
16	S	R	C	V	144
17	S	GR	G	B	145
18	S	GR	G	M	146
19	S	GR	G	C	147
20	S	GR	G	V	148
21	S	GR	B	M	149
22	S	GR	B	C	150
23	S	GR	B	V	151
24	S	GR	M	C	152
25	S	GR	M	V	153
26	S	GR	S	V	154
27	S	G	B	M	155
28	S	G	B	C	156
29	S	G	B	V	157
30	S	G	M	C	158
31	S	G	M	V	159

Färgvals-kommando Grafik + text	Färgnr 0	1	2	3	Färgvals-kommando Enbart grafik
32	S	G	C	V	160
33	S	B	M	C	161
34	S	B	M	V	162
35	S	B	C	V	163
36	S	M	C	V	164
37	R	GR	G	B	165
38	R	GR	G	M	166
39	R	GR	G	C	167
40	R	GR	G	V	168
41	R	GR	B	M	169
42	R	GR	B	C	170
43	R	GR	B	V	171
44	R	GR	M	C	172
45	R	GR	M	V	173
46	R	GR	C	V	174
47	R	G	B	M	175
48	R	G	B	C	176
49	R	G	B	V	177
50	R	G	M	C	178
51	R	G	M	V	179
52	R	G	C	V	180
53	R	B	M	C	181
54	R	B	M	V	182
55	R	B	C	V	183
56	R	M	C	V	184
57	GR	G	B	M	185
58	GR	G	B	C	186
59	GR	G	B	V	187
60	GR	G	M	C	188
61	GR	G	M	V	189
62	GR	G	C	V	190
63	GR	B	M	C	191
64	GR	B	M	V	192
65	GR	B	C	V	193
66	GR	M	C	V	194
67	G	B	M	C	195
68	G	B	M	V	196
69	G	B	C	V	197
70	G	M	C	V	198
71	B	M	C	V	199
72	S	R	S	R	200
73	S	S	R	R	201
74	S	GR	S	GR	202
75	S	S	GR	GR	203
76	S	G	S	G	204
77	S	S	G	G	205
78	S	B	S	B	206
79	S	S	B	B	207

Färgvals-kommando Grafik + text	Färgnr 0	1	2	3	Färgvals-kommando Enbart grafik
80	S	M	S	M	208
81	S	S	M	M	209
82	S	C	S	C	210
83	S	S	C	C	211
84	S	V	S	V	212
85	S	S	V	V	213
86	R	GR	R	GR	214
87	R	R	GR	GR	215
88	R	G	R	G	216
89	R	R	G	G	217
90	R	B	R	B	218
91	R	R	B	B	219
92	R	M	R	M	220
93	R	R	M	M	221
94	R	C	R	C	222
95	R	R	C	C	223
96	R	V	R	V	224
97	R	R	V	V	225
98	GR	G	GR	G	226
99	GR	GR	G	G	227
100	GR	B	GR	B	228
101	GR	GR	B	B	229
102	GR	M	GR	M	230
103	GR	GR	M	M	231
104	GR	C	GR	C	232
105	GR	GR	C	C	233
106	GR	V	GR	V	234
107	GR	GR	V	V	235
108	G	B	G	B	236
109	G	G	B	B	237
110	G	M	G	M	238
111	G	G	M	M	239
112	G	C	G	C	240
113	G	G	C	C	241
114	G	V	G	V	242
115	G	G	V	V	243
116	B	M	B	M	244
117	B	B	M	M	245
118	B	C	B	C	246
119	B	B	C	C	247
120	B	V	B	V	248
121	B	B	V	V	249
122	M	C	M	C	250
123	M	M	C	C	251
124	M	V	M	V	252
125	M	M	V	V	253
126	C	V	C	V	254
127	C	C	V	V	255

12.5 Exempel

Ex. 1

```
 10 FGPOINT 0,0,0 :REM Sätter punkt 0,0 i färg 0
 15 PRINT CHR  (12) :REM Tömmer bildminnet
 20 FGFILL 239,239 :REM Tömmer högupplösningsminnet
 30 FGCTL 3 :REM Väljer färg S, R, GR, B + text
 35 REM Rita kvadrat
 40 FGPOINT 20,20,2 :REM Sätter punkt 20,20 i färg 2 (GR)
 50 FGLINE 220,20 :REM Ritar linje till 220,20 i färg 2
 60 FGLINE 220,220,3 :REM Ritar linje till 220,220 färg 3 (B)
 70 FGLINE 20,220,2 :REM Ritar linje till 20,220 färg 2 (GR)
 80 FGLINE 20,20 :REM Ritar linje till 20,20 färg 2
 90 PRINT CUR(12,15);CYA DBLE "KVADRAT";
100 END
```

Ex. 2

```
 10 !Rita klot
 20 EXTEND
 30 C=1.2 ! Bredd/höjd korrigeringsfaktor
 40 ; CHR¤(12) ! Tömmer bildskärmen
 50 FGPOINT 0,0,0 ! Sätter punkt 0,0 i färg 0
 60 FGFILL 239,239 ! Fyller skärmen med färg 0
 70 FGCTL 7 ! Väljer färgkombination
 80 Origo=119
 90 Radie=95
100 Färg=3
110 FGPOINT Origo+Radie,Origo,Färg
120 WHILE Xposition <=2 * PI
130    FGLINE COS (Xposition) * Radie+Origo, SIN (Xposition) * Radie * C+Origo
140    Xposition=Xposition+1/18
150 WEND
160 FGPAINT Origo,Origo
170 END
```

13 Funktionstangenter

Datorn är utrustad med åtta funktionstangenter, placerade mellan de alfanumeriska tangenterna och de numeriska. Funktionstangenterna är märkta PF1, PF2, . . ., PF8.

Funktionstangenterna kan tilldelas olika fasta funktioner i program för t. ex. markörförflyttning, sidbyte, hopp till programmodul etc.

Med funktionstangenterna kan man åstadkomma 32 olika ASCII-koder enligt följande tabell

		SHIFT	CTRL	SHIFT+CTRL
PF1	192	208	224	240
PF2	193	209	225	241
PF3	194	210	226	242
PF4	195	211	227	243
PF5	196	212	228	244
PF6	197	213	229	245
PF7	198	214	230	246
PF8	199	215	231	247

En tryckning på en funktionstangent kan leda till ett anrop av ett underprogram.

Exempel:

```
10 ON ERROR GOTO100
20 INPUT"Tal",P(I)
30 I=I+1
40 GOTO 20
    |
    |
    |
    |
100 IF ERRCODE < > 53 THEN RESUME
110 A=SYS(6)
120 GET X¤
130 ON ASC (X¤)-191 RESUME 400, 500, 600
```

När en funktionstangent tryck ned vid **INPUT** eller **INPUT LINE**,genereras ett fel med **ERRCODE**=53. Programmet bör alltså innehålla en rutin, som tar hand om fel 53. Genom att använda funktionen **SYS**(6) och därefter läsa tecknet med **GET** kan man ta reda på vilken funktionstangent, som har tryckts ned.

14 Skillnader i BASIC mellan ABC 800 och ABC 80

De förändringar, som är gjorda i förhållande till ABC 80 BASIC, är en anpassning till ANSI-standard. Dessutom är minnesdispositionen och internkoden ändrad.

1. Vid tilldelning av ett flyttalsvärde till en heltalsvariabel sker avrundning.

   Exempel A%=3.567
   Variabeln A% erhåller värdet:
   ABC 80: 3
   ABC 800: 4

2. Vid utskrift med hjälp av **TAB** anges utskriftspositionen med **TAB**(1) och uppåt.

   Exempel
   ```
   PRINT TAB(5)"B"
   ```
   ABC 80: B skrivs i position 6 (d.v.s. position 0-39).
   ABC 800: B skrivs i position 5 (d.v.s. position 1-40/80)

3. Vid utskrift av en variabels värde med **NUM¤** tas den position, som är reserverad för plustecknet, bort.

   Exempel
   ```
   20 I=1234
   30 PRINT NUM¤(I)
   ```
   ger som resultat:
   ABC 80 : 1234
   ABC 800:1234

4. Vid utskrift av numeriska variabler åtskilda av semikolon (;), läggs ett extra blanktecken in.

   Exempel
   ```
   PRINT X;Y
   ```
   ger följande utskrift
   ABC 80 : 0 0
   ABC 800: 0 0

5. **CALL**-instruktionerna ersätts av **POSIT, GET** ... **COUNT** och **PUT**.

   Exempel Läsning:
   ABC 80 : Z=**CALL**(28666,filnr)+**CALL**(28668,sektornr)
   ABC 800: **POSIT** #filnr,sektornr*253: **GET**#filnr,Q0¤ **COUNT** 253
   Skrivning:
   ABC 80 : Z=**CALL**(28666,filnr): Q0¤=A¤: Z=**CALL**(28670,sektornr)
   ABC 800: **POSIT** #filnr,sektornr*253: **PUT** #filnr,A¤

6. Instruktionen **CHAIN** " " tas bort eller ersätts av **END** i ABC 800 program.

7. Instruktionen **END** skall stå ensam på raden. **END** stänger filer men nollställer ej variabler.

8. Instruktionen **ON ERROR GOTO** 0 ersätts av **ON ERROR GOTO**.

9. För att överföra program från ABC 80 till ABC 800 måste man ha lagrat programmen i textformat (.BAS) d.v.s. med kommandot **LIST** filnamn. Rader, som ej är kompatibla, ger felmeddelande, och är märkta med ett frågetecken efter radnumret.

15 Felmeddelanden

Fel 19 - 68: I/O-fel
Fel 130 - 176: Fel vid programkörning
Fel 180 - 191: Logiska fel
Fel 200 - 211: Allmänna fel
Fel 220 - 234: Formella BASIC-fel

Fel	Meddelande	Kommentar
19	Kan ej öppna fler filer	Sju filer är öppnade
20	För lång rad (>160 tecken)	En rad får innehålla max. 160 tecken
21	Hittar ej filen	Filen finns inte eller har sökts under fel namn
32	Filen ej öppnad	
34	Slut på filen	Försökt läsa efter filslut
35	Checksummafel vid läsning	Skivan eller kassettbandet är skadat
36	Checksummafel vid skrivning	Skivan är skadad
37	Felaktigt sektorformat	Fel på skiva eller kassett
38	Sektornummer utanför filen	Försök att läsa längre än filen medger
39	Filen skrivskyddad	
40	Filen raderskyddad	
41	Skivan full	Filen får ej plats på skivan
42	Skivan ej klar	Ingen flexskiva isatt eller luckan öppen
43	Skivan skrivskyddad	
44	Logisk fil ej öppnad	
45	Fel logiskt filnummer	
46	Fel enhetsnummer	
47	Fel trapnummer	
48	Fel i biblioteket	
49	Felaktigt fysiskt filnummer	Fel på skivan
51	Enheten upptagen	
52	Ej till denna enhet	
53	Funktionstangent	Funktionstangent har tryckts ned i **INPUT**-eller **INPUT LINE**-sats
54	IEC både sändare och mottagare	IEC-option
55	IEC-mottagare ej aktiv	IEC-option
56	IEC-sändare ej aktiv	IEC-option
57	Tecken från tangentbord ej i tid	
58	Ogiltigt tecken inläst	
64	Felaktigt **"NAME"**	Nya filnamnet existerar redan
68	Felaktig tidsspecifikation	
130	För stort flyttal	
131	Index utanför tillåtet område	Försök att använda index större än motsvarande **DIM**
132	För stort heltal	
133	Fel i ASCII-aritmetiskt uttryck	
134	Index utanför strängen	Index för stort eller negativt

Fel	Meddelande	Kommentar
135	Negativ **"SPACE¤","STRING¤"** eller **"TAB"** < 1	
136	För lång sträng	För liten dimension på den mottagande strängen
137	Ej tillåtet öka **"DIM"**	Ett fält får inte ökas utöver sin ursprungliga längd
138	Fel värde i **"ON"**-uttryck	
139	**"RETURN"** utan **"GOSUB"**	En **RETURN**-sats påträffad utan att en föregående **GOSUB**-sats har blivit utförd
140	Felaktigt **"RETURN"**-variabel	
141	Data slut	Datalistan har blivit tömd och en **READ**-sats efterfrågade ytterligare data
142	Felaktigt argument i funktion	
143	Felaktig **"SYS"**-funktion	
144	Ej tillåten rad	
145	**"FNEND"** utan föregående **"RETURN"**	
146	**"PRINT USING"**-fel	Felaktigt format i **PRINT USING**-sats
147	Felaktiga data	
148	För lite indata	För få data inmatade vid **INPUT**
149	**"RESTORE"** ej på en **"DATA"**-rad	
150	För mycket indata	För många data inmatade vid **INPUT**
151	**"RESUME"** utan fel	
176	Grafisk punkt utanför bildskärmen	
180	Hittar ej detta radnummer	Referens till ett radnummer som inte finns i programmet
181	Felaktigt inhopp i funktion	
182	**"NEXT"** eller **"WEND"** saknas	
183	**"FOR"** eller **"WHILE"** saknas	
184	Fel variabel efter **"NEXT"**	
185	Blandade **"FOR"**-loopar med samma variabel	
186	**"FOR"**-loop med lokal variabel ej tillåtet	Gäller i flerradiga funktioner
187	Funktion ej definierad	Anrop till ej definierad funktion
188	Flera funktioner med samma namn	
189	Felaktig funktion	Ej tillåtet att blanda flera **DEF**
190	Fel antal index	Antalet index överensstämmer ej med **DIM**
191	Ej tilldelningsbar i funktion	Funktionens argument är ej tilldelningsbart i funktionen
200	Enheten ej ansluten	
201	Minnet fullt	Datorns primärminne har ej plats för program och data
202	**"LIST"**-skyddat program	

Fel	Meddelande	Kommentar
203	Fel programformat	Programmet är sparat under en icke kompatibel BASIC-version
204	"**MERGE**" går ej på "BAC"-fil	
205	"**COMMON**"-fel	
206	Använd kommandot "**RUN**"	
207	Kan ej fortsätta	Gäller **GOTO** radnr och **CON**
208	Otillåtet som kommando	Instruktionen kan ej användas som kommando
209	Fel data till kommando	Felaktigt argument till kommandot, t.ex. **LIST** # #
210	Felaktigt tal	Talet innehåller tecken som inte är siffror
211	Precision får ej ändras	Ej tillåtet ändra precision efter tilldelning av variabel
220	Förstår ej	Formellt BASIC-fel
221	Otillåtet tecken efter satsen	Formellt BASIC-fel. Datorn förväntade sig Return, kolon (:) eller utropstecken (!)
222	Måste vara först på en rad	
223	Fel antal eller typ av argument	
224	Otillåten blandning av tal och strängar	
225	Ej enkel variabel	Ej tillåtet ha index på variabel t.ex. i **FOR**-loop
226	Felaktig sats efter "**ON**"	Formellt BASIC-fel
227	"," saknas	Formellt BASIC-fel
228	"=" saknas	Formellt BASIC-fel
229	")" saknas	Formellt BASIC-fel
230	"**AS FILE**" saknas	Förekommer i **OPEN**- och **PREPARE**-satser
231	"**AS**" saknas	Fel i **NAME AS**
232	"**TO**" saknas	Förekommer i **FOR**-loopar
233	Radnummer saknas	
234	Felaktig variabel	

16 Kommando- och instruktions-sammanfattning

ABS (funktion) Sid. 63
Format **ABS**(argument)
Funktion Ger absolutvärdet av argumentet

ADD¤ (funktion) Sid. 67
Format **ADD¤**(A¤,B¤,P%)
Funktion Adderar värdet av strängarna A¤ och B¤. Svar med P% decimaler.

ASCII (funktion) Sid. 68
Format **ASCII**(A¤)
Funktion Ger ASCII-värdet för första tecknet i A¤.

ATN (funktion) Sid. 63
Format **ATN**(argument)
Funktion Arcustangens för argumentet i radianer.

AUTO (kommando) Sid. 25
Format **AUTO** [argument 1 [,argument 2]]
Där < argument 1 > anger första radnummer, som ska skrivas och < argument 2 > anger radintervall.
Funktion Automatisk radnumrering.

¤BAS (kommando under DOS) Sid. 26
Format **¤BAS**
Funktion Övergång till BASIC

BYE (kommando) Sid. 26
Format **BYE**
Funktion Övergång till DOS

CALL (funktion) Sid. 73
Format **CALL**(A%[,D%])
Funktion Anrop av assemblerprogram.
VARNING Man kan förstöra en körning, om **CALL** används på fel sätt.

CHAIN (instruktion) Sid. 35
Format **CHAIN** "filnamn.typ"/strängvariabel
Funktion Laddar och startar exekveringen av ett program.

CHR¤ (funktion) Sid. 68
Format **CHR¤**(argument[,argument,...])
Funktion Ger teckensträng som motsvarar argumentens ASCII-värden.

CLEAR (kommando) Sid. 26

Format **CLEAR**

Funktion Nollställer alla variabler och stänger alla öppna filer.

CLOSE (instruktion) Sid. 35

Format **CLOSE** [filnummer, ...]

Funktion Stänger angiven fil.

CLR DOT (instruktion - grafik) Sid. 82

Format **CLR DOT** R%,K%

Funktion Släcker en grafisk punkt på rad R% (0-71) i position K% (2-79).

Observera Origo är i övre, vänstra hörnet.

COMMON (instruktion) Sid. 35

Format **COMMON** variabel[(n,...)] [,variabel, ...]
COMMON strängvariabel[(n,...)]=längd [,strängvariabel, = längd,...]

Funktion Deklaration av de variabler, vars värden ska överföras till ett annat program.

COMP% (funktion) Sid. 68

Format **COMP%**(A¤,B¤)

Funktion Jämför två numeriska strängar.

CON (kommando) Sid. 26

Format **CON** (eller **CONTINUE**)

Funktion Fortsätter programexekvering.

COS (funktion) Sid. 63

Format **COS**(argument)

Funktion Ger cosinus för argumentet (argumentet i radianer).

CUR (funktion) Sid. 73

Format **CUR**(R%,N%)

Funktion Flyttar markören till rad R%, (0–23) position N% (0–39/79).

CVT (funktion) Sid. 73

Format **CVT%¤**(heltalsvariabel)
CVT¤%(strängvariabel)
CVTF¤(flyttalsvariabel)
CVT¤F(strängvariabel)

Funktion Överför tal till strängar respektive återskapar talen.

DATA (instruktion) Sid. 36

Format **DATA** värde [,värde, ...]

Funktion Utgör tillsammans med **READ**, ett sätt att tilldela variabler värden.

DEF (instruktion) Sid. 36

Format Enradig funktion:
DEF FNidentifierare[(argument)]=funktion
Flerradig funktion:
DEF FNidentifierare [%/] [(argument)] [**LOCAL** variabel [,variabel, ...]]

Funktion Definierar enradiga och flerradiga funktioner.

DIGITS (instruktion) Sid. 38
Format **DIGITS** antal siffror
Funktion Ger antal siffror för utskrift.

DIM (instruktion) Sid. 39
Format **DIM** variabel(n)[, variabel(n,...), ...]
DIM strängvariabel[(n,...)] [=uttryck]
Funktion Reserverar utrymme för strängar och vektorer.

DIV¤ (funktion) Sid. 68
Format **DIV¤**(A¤,B¤,P%)
Funktion Ger kvoten A¤/B¤ avrundad till P% decimaler.

DOT (instruktion - grafik) Sid. 82
Format **DOT**(R%,K%)
Funktion Ger -1 (sann) om punkten är tänd, annars 0 (falsk).
R%=rad (0-71), K%=position (0-79)
Observera Origo är i övre, vänstra hörnet.

DOUBLE (instruktion) Sid. 40
Format **DOUBLE**
Funktion Flyttal får dubbel precision (16 siffror).

ED (kommando) Sid. 27
Format **ED** [radnummer]
Funktion Inleder ändringar i program.

END (instruktion) Sid. 40
Format **END**
Funktion Avslutar programmet.

ERASE (kommando) Sid. 28
Format **ERASE** radnummer I [- radnummer II]
ERASE radnummer -
ERASE - radnummer
Funktion Raderar en eller flera programrader.

ERRCODE (funktion) Sid. 74
Format **ERRCODE**
Funktion Ger värdet av senast genererade felkod.

EXP (funktion) Sid. 63
Format **EXP**(argument)
Funktion Ger e**argument.

EXTEND (instruktion) Sid. 40
Format **EXTEND**
Funktion Tillåter långa variabelnamn.

FGCTL (instruktion - högupplösningsgrafik) Sid. 84
Format **FGCTL** färgvalskommando
Funktion Väljer färgkombination.

FGFILL (instruktion - högupplösningsgrafik) Sid. 84

Format **FGFILL** x,y[,färgnummer]

Funktion Fyller en rektangel från föregående läge till det läge koordinaterna x,y anger, x=0–239, y=0–239.

Observera Origo är i nedre, vänstra hörnet.

FGLINE (instruktion - högupplösningsgrafik) Sid. 84

Format **FGLINE** x,y[,färgnummer]

Funktion Ritar en linje från föregående position till det läge, som koordinaterna x,y anger, x=0–239, y=0–239.

Observera Origo är i nedre, vänstra hörnet.

FGPAINT (instruktion - högupplösningsgrafik) Sid. 84

Format **FGPAINT** x,y[,färgnummer]

Funktion Fyller en sluten yta. x=0–239, y=0–239.

Observera Origo är i nedre, vänstra hörnet.

FGPOINT (instruktion - högupplösningsgrafik) Sid. 85

Format **FGPOINT** x,y[,färgnummer]

Funktion Sätter ett bildelement på den position som koordinaterna x,y anger. x=0–239, y=0–239.

Observera Origo är i nedre, vänstra hörnet.

FIX (funktion) Sid. 63

Format **FIX**(argument)

Funktion Ger trunkerat värde av argumentet.

Observera Jämför **INT**(X).

FLOAT (instruktion) Sid. 40

Format **FLOAT**

Funktion Alla variabler antas vara flyttade om inget annat anges.

FN (funktion) Sid. 74

Format **FN**identifierare [%/¤] ((parameter [,parameter,...])

Funktion Anrop av användardefinierad funktion.

Observera Jämför **DEF FN**.

FNEND (instruktion) Sid. 38

Format **FNEND**

Funktion Avslutar flerradig funktion.

FOR (instruktion) Sid. 41

Format **FOR** variabel=uttryck **TO** uttryck [**STEP** intervall]

Funktion Inleder programloop.

GET (instruktion) Sid. 42

Format **GET** strängvariabel

Funktion Läser ett tecken från tangentbordet.

GET # (instruktion) Sid. 43

Format **GET** # filnummer, strängvariabel [**COUNT** antal tecken]

Funktion Läser från fil.

GOSUB (instruktion) Sid. 43
Format **GOSUB** radnummer
Funktion Ger ovillkorligt hopp till subrutin.

GOTO (instruktion) Sid. 44
Format **GOTO** radnummer
Funktion Ger ovillkorligt hopp till angivet radnummer.

HEX¤ (funktion) Sid. 64
Format **HEX¤**(argument)
Funktion Omvandlar decimalt tal till hexadecimal sträng.

IF - THEN - ELSE (instruktion) Sid. 44
Format **IF** villkor **THEN** sats[er]/radnr [**ELSE** sats[er]/radnr]
Funktion Villkorlig styrning av ordningsföljden mellan programraderna.

INP (funktion) Sid. 74
Format **INP**(I%)
Funktion Ger datavärdet från inport I%.
VARNING Detta är en maskinorienterad funktion, som ska användas enbart vid avancerad programmering.

INPUT (instruktion) Sid. 45
Format **INPUT** [#filnr/"ledtext"] variabel [,variabel, ...]
Funktion Hämtar in data till det program, som körs.

INPUT LINE (instruktion) Sid. 46
Format **INPUT LINE** [#filnummer,]strängvariabel
Funktion Tar emot inmatning av en rad tecken.

INSTR (funktion) Sid. 69
Format **INSTR**(N%,A¤,B¤)
Funktion Söker efter strängen B¤ i A¤ med början i position N%.

INT (funktion) Sid. 64
Format **INT**(argument)
Funktion Ger värdet av det största heltal, som är mindre än eller lika med argumentet. Jämför **FIX**.

INTEGER (instruktion) Sid. 46
Format **INTEGER**
Funktion Alla variabler antas vara heltalsvariabler, om inget annat anges.

KILL (instruktion) Sid. 47
Format **KILL** "[enhet:]filnamn.typ"
Funktion Den angivna filen raderas från det yttre minnet.

LEFT (funktion) Sid. 69
Format **LEFT**[¤](A¤,I%)
Funktion Ger de I% första tecknen av strängen A¤.

LEN (funktion) Sid. 69
Format **LEN**(A¤)
Funktion Ger antalet tecken i strängen A¤ , d.v.s. stränglängden (inklusive mellanslag).

LET (instruktion) Sid. 47
Format [**LET**] variabel=uttryck
Funktion Tilldelar en variabel ett värde.

LIST (kommando) Sid. 28
Format **LIST** [enhet:]filnamn[.typ]
LIST [radnummer [-radnummer]]
LIST radnummer -
LIST - radnummer
LIST enhet: [radnummer - radnummer]
Funktion Listar hela eller del av program.

LOAD (kommando) Sid. 29
Format **LOAD** [enhet:]filnamn[.typ]
Funktion Laddar in ett BASIC-program.

LOG (funktion) Sid. 64
Format **LOG**(argument)
Funktion Ger naturliga logaritmen för argumentet.

LOG10 (funktion) Sid. 65
Format **LOG10**(argument)
Funktion Ger tiologaritmen för argumentet.

MERGE (kommando) Sid. 29
Format **MERGE** [enhet:]filnamn[.typ]
Funktion Länkar BAS-filer.

MID (instruktion) Sid. 69
Format **MID**[¤](A¤,P%,K%)
Funktion Tillordnar tecken nr P% t.o.m. P%+K%-1 av A¤ ett nytt värde, d.v.s. byter tecken i strängen.

MID (funktion) Sid. 70
Format **MID**[¤](A¤,P%,K%)
Funktion Ger den delsträng av A¤, som börjar i position P% och är K% tecken lång, d.v.s. tecknen nr P% t.o.m. P%+K%-1.

MOD (funktion) Sid. 65
Format **MOD**(argument 1, argument 2)
Funktion Ger resten vid heltalsdivision av argumenten..

MUL¤ (funktion) Sid. 70
Format **MUL¤**(A¤,B¤,P%)
Funktion Ger produkten A¤ * B¤ med P% decimaler.

NAME (instruktion) Sid. 48
Format **NAME** "[enhet:]filnamn1.typ" **AS** "filnamn2.typ"
Funktion Ändrar namnet på en fil.

NEW (kommando) Sid. 30
Format **NEW**
Funktion Raderar innehållet i minnet.
Observera Kommandot **SCR** (scratch) kan också användas.

NEXT (instruktion) Sid. 42
Format **NEXT** variabel
Funktion **NEXT** anger slutet på en programloop, som inletts med en **FOR**-sats.

NO EXTEND (instruktion) Sid. 49
Format **NO EXTEND**
Funktion Avslutar arbete i **EXTEND**-mod.

NO TRACE (instruktion) Sid. 48
Format **NO TRACE**
Funktion Avslutar den utskrift av radnummer, som begärts med instruktionen **TRACE**.

NUM¤ (funktion) Sid. 70
Format **NUM¤**(argument)
Funktion Ger den numeriska sträng, som motsvarar argumentet.

OCT¤ (funktion) Sid. 65
Format **OCT¤** (argument)
Funktion Omvandlar decimalt tal till oktal sträng.

ON ERROR GOTO (instruktion) Sid. 49
Format **ON ERROR GOTO** [radnummer]
Funktion Ger hopp till angivet radnummer vid fel.

ON - GOSUB (instruktion) Sid. 49
Format **ON** uttryck **GOSUB** radnummer[,radnummer,...]
Funktion Används för att villkorligt hoppa till en av flera subrutiner eller en av flera ingångar i en subrutin.

ON - GOTO (instruktion) Sid. 50
Format **ON** uttryck **GOTO** radnummer[,radnummer,...]
Funktion För att hoppa till en av flera rader, beroende på uttryckets värde.

ON - RESTORE (instruktion) Sid. 50
Format **ON** uttryck **RESTORE** radnummer [radnummer,...]
Funktion Ställer **DATA**-pekaren efter samma urvalsförfarande som **ON - GOTO**.

ON - RESUME (instruktion) Sid. 51

Format **ON** uttryck **RESUME** radnummer[,radnummer,...]
Funktion För att hoppa till en av flera rader, beroende på uttryckets värde, då satsen utförs. Felhantering återställs. Används tillsammans med **ON ERROR GOTO.**

OPEN (instruktion) Sid. 51

Format **OPEN** "[enhet:][filnamn[.typ]]" **AS FILE** filnummer
Funktion Används för att öppna en befintlig fil.

OPTION BASE (instruktion) Sid. 52

Format **OPTION BASE** n
Funktion Anger lägsta värde för vektorindex (n = 0 eller 1).

OUT (instruktion) Sid. 75

Format **OUT** port,data [,port,data, ...]
Funktion Används för att adressera utportar vid utmatning av data.
VARNING Detta är en maskinorienterad instruktion för avancerad programmering. Denna instruktion och instruktionen **INP** ger användaren tillgång till I/O-funktionerna i ABC 800 och dess I/O-buss.

PEEK (funktion) Sid. 75

Format **PEEK**(I%)
Funktion Ger innehållet i adress I%.
VARNING Denna funktion är avsedd att användas vid avancerad programmering.

PEEK2 (funktion) Sid. 75

Format **PEEK2**(B%)
Funktion Läser innehållet i två bytes.
VARNING Denna funktion är avsedd att användas vid avancerad programmering.

PI (funktion) Sid. 65

Format **PI**
Funktion Konstant med värdet 3.14159 (enkel precision)

POKE (funktion) Sid. 75

Format **POKE** adress,data (,data, ...)
Funktion Används för att ladda in ett värde i en minnescell.
VARNING Denna instruktion är maskinorienterad och ska endast användas vid avancerad programmering. Om den används på fel sätt, kan innehållet i minnet förstöras.

POSIT (instruktion) Sid. 52

Format **POSIT** #filnummer, position
Funktion Positionerar filpekare.

PREPARE (instruktion) Sid. 53

Format **PREPARE** "[enhet:][filnamn.typ]" **AS FILE** filnummer
Funktion Skapar och öppnar en ny fil.

PRINT (instruktion) Sid. 53

Format **PRINT** [#filnummer,] "data"/variabel (,"data"/variabel, ...)

Funktion Matar ut data i ASCII-form.

PRINT (instruktion - färgval m.m.) Sid. 81

Format **PRINT** [**CUR**(R,K)]argument [;argument;...]"text"

Funktion Används för att skriva text och grafik. Argumenten styr färgval m.m. Bilden ritas med början på rad R (0-23), position K (0-39/79).

PRINT USING (instruktion) Sid. 54

Format **PRINT** [#filnummer] **USING** "formatsträng";"data"/variabel [,"data"/variabel, ...]

Funktion Skriver tal och strängar med angivet format.

PUT (instruktion) Sid. 57

Format **PUT** #filnummer, strängvariabel

Funktion Skriver en strängvariabel.

RANDOMIZE (instruktion) Sid. 57

Format **RANDOMIZE**

Funktion Sätter ett slumpmässigt startvärde för funktionen **RND** (slumptalsgeneratorn).

VARNING Bör bara användas en gång i varje program.

READ (instruktion) Sid. 57

Format **READ** variabel[, variabel, ...]

Funktion Utgör, tillsammans med **DATA**-satser, ett sätt att tilldela variabler värden.

REM (instruktion) Sid. 58

Format **REM** text

Funktion Inleder kommentar i program.

Observera **REM** kan bytas mot !

RENUMBER (kommando) Sid. 30

REN

Format **REN**[**UMBER**] [radnummer[,intervall[,fr.o.m. rad -t.o.m. rad]]]

Funktion Ändrar radnumreringen i det aktuella programmet.

RESTORE (instruktion) Sid. 58

Format **RESTORE** [radnummer]

Funktion Gör det möjligt att använda innehållet i **DATA**-satser flera gånger.

RESUME (instruktion) Sid. 59

Format **RESUME** [radnummer]

Funktion Återhopp från felhanteringsrutin.

RETURN (instruktion) Sid. 59
Format **RETURN** [variabel]
Funktion Ger återhopp från subrutin eller flerradiga funktioner.

RIGHT (funktion) Sid. 70
Format **RIGHT**[¤](A¤,N%)
Funktion Ger de sista tecknen i A¤, fr.o.m position N%.

RND (funktion) Sid. 65
Format **RND**
Funktion Ger ett slumptal mellan 0 och 0.999999.

RUN (kommando) Sid. 31
Format **RUN**[[enhet:]filnamn[.typ]]
Funktion Laddar och exekverar ett BASIC-program eller exekverar (kör) programmet i datorns minne.

SAVE (kommando) Sid. 32
Format **SAVE** [enhet:]filnamn[.typ]
Funktion Skapar en programfil på ett yttre minne och lagrar programmet, som finns i datorns arbetsminne, i denna fil.

SET DOT (instruktion - grafik) Sid. 82
Format **SET DOT** R%,K%
Funktion Tänder en grafisk punkt på rad R% (0-71) i position K% (2-79).
Observera Origo är i övre, vänstra hörnet.

SGN (funktion) Sid. 66
Format **SGN**(argument)
Funktion Funktionen **SGN**(X) har värdet +1 om X är positivt, 0 om X är 0 och -1 om X är negativt.

SIN (funktion) Sid. 66
Format **SIN**(argument)
Funktion Ger sinus för argumentet (argumentet i radianer).

SINGLE (instruktion) Sid. 59
Format **SINGLE**
Funktion Ändrar alla variabler och uttryck, som är flyttal, till enkel precision (7 siffror).
Observera Det är inte tillåtet att blanda **SINGLE** och **DOUBLE** i samma program.

SPACE¤ (funktion) Sid. 71
Format **SPACE¤**(N%)
Funktion Ger en sträng, som består av N% mellanslag.

SQR (funktion) Sid. 66
Format **SQR**(argument)
Funktion Ger kvadratroten ur argumentet.

STOP (instruktion) Sid. 60

Format **STOP**

Funktion Stoppar programexekveringen.

Observera Programexekveringen kan fortsättas med något av kommandona **CON** eller **GOTO.**

STRING¤ (funktion) Sid. 71

Format **STRING¤**(I%,K%)

Funktion Ger en sträng med I% st tecken, som har ASCII-värdet K%.

SUB ¤ (funktion) Sid. 71

Format **SUB¤**(A¤,B¤,P%)

Funktion Ger den aritmetiska differensen A¤-B ¤ med P% decimaler.

SWAP% (funktion) Sid. 76

Format **SWAP%**(N%)

Funktion Första och andra byten av N% skiftar plats.

SYS (funktion) Sid. 76

Format **SYS**(I%)

Funktion Ger systemstatus enligt följande:

SYS(2) Totalt minnesutrymme
SYS(3) Programmets storlek
SYS(4) Kvarvarande minnesutrymme
SYS(5) Tangentbordsflaggan
SYS(6) Lägger tillbaka sist inlästa tecken i tangentbordsbufferten.
SYS(11) Programmets startadress
SYS(12) Variabelrot

TAB (funktion) Sid. 76

Format **TAB**(I%)

Funktion Flyttar skrivhuvud eller markör till position I% på raden.

TAN (funktion) Sid. 66

Format **TAN**(argument)

Funktion Ger tangens för argumentet (argumentet i radianer).

TIME¤ (funktion) Sid. 76

Format **TIME¤**

Funktion Ger tidsangivelse år-mån-dag tim.min.sek

TRACE (instruktion) Sid. 60

Format **TRACE** [#filnummer]

Funktion Skriver ut radnumren på de programrader,som genomlöps.

TXPOINT (instruktion - grafik) Sid. 81

Format **TXPOINT** x,y[,1/0]

Funktion Tänder (1) eller släcker (0) en grafisk punkt i position x,y. x=0–77. y=0–71.

Observera Origo är i nedre, vänstra hörnet.

UNSAVE (kommando) Sid. 32
Format **UNSAVE** [enhet:]filnamn[.typ]
Funktion Raderar en fil på skiva.

VAL (funktion) Sid. 71
Format **VAL**(A¤)
Funktion Beräknar numeriska värdet av den numeriska strängen A¤.

VAROOT (funktion) Sid. 77
Format **VAROOT** (variabel)
Funktion Ger adressen till en tabell, som innehåller uppgifter om en variabel.
VARNING Denna funktion är avsedd att användas vid avancerad programmering.

VARPTR (funktion) Sid. 77
Format **VARPTR** (variabel)
Funktion Ger adressen till en variabels värde.
VARNING Denna funktion är avsedd att användas vid avancerad programmering.

WEND (instruktion) Sid. 60
Format **WEND**
Funktion **WEND** avslutar en loop, som inleds av **WHILE.**

WHILE (instruktion) Sid. 61
Format **WHILE** uttryck
Funktion Anger villkor för uthopp ur en programloop.

A¤+B¤ (funktion) Sid. 72
Format A¤+B¤
Funktion Sammankedjar (konkatenerar) två strängar.

17 Litteraturförteckning

- "Mikrodatorns ABC" av Gunnar Markesjö.
  Beskriver hur ABC 80 fungerar.

- "ABC om BASIC" av Anders Andersson, Arne Kullbjer, Jan Lundgren och Sören Thornell.
  Behandlar grunderna i ABC 80 BASIC.

- "Avancerad programmering på ABC 80" av Anders Isaksson och Örjan Kärrsgård.
  Denna bok vänder sig till dem, som redan kan en del om datorer och vill veta hur man gör professionella program på ABC 80.

- "Styr och mät med ABC 80" av Åke Westh

- "ABC om programmering och dokumentation" av Jan Lundgren och Bengt Lundin

- "Bygg ut ABC 80 med Databoard 4680". SATTCO AB.

- "Att programmera ABC 80" av Lennart Rodhe.

- "ABC om användardokumentation". Luxor AB.

- "Lärobok i PASCAL" av Anders Haraldsson.

- "Datoranvändning med IEC-buss" av Sune Windisch.

- "Vår elektroniska framtid" av B-G Wennersten.

- "Privatdatorn - din egen dator" av B-G Wennersten.

- "BASIC Computer Games" av David H. Ahl.

- "More BASIC Computer Games" av David H. Ahl.

- "Z80, Technical Manual". Zilog.

- "Z80, Programming Manual". Zilog.

- "Dataordboken". SIS handbok 142.

Arbete pågår med framtagning av motsvarande litteratur för ABC 800.

18 Bilagor

Bilaga 1: BASIC II errata.

Tilldelning av variabelvärden eller dimensionering (**DIM**) av variabler före **COMMON** gör att ABC 800 "går vilse".

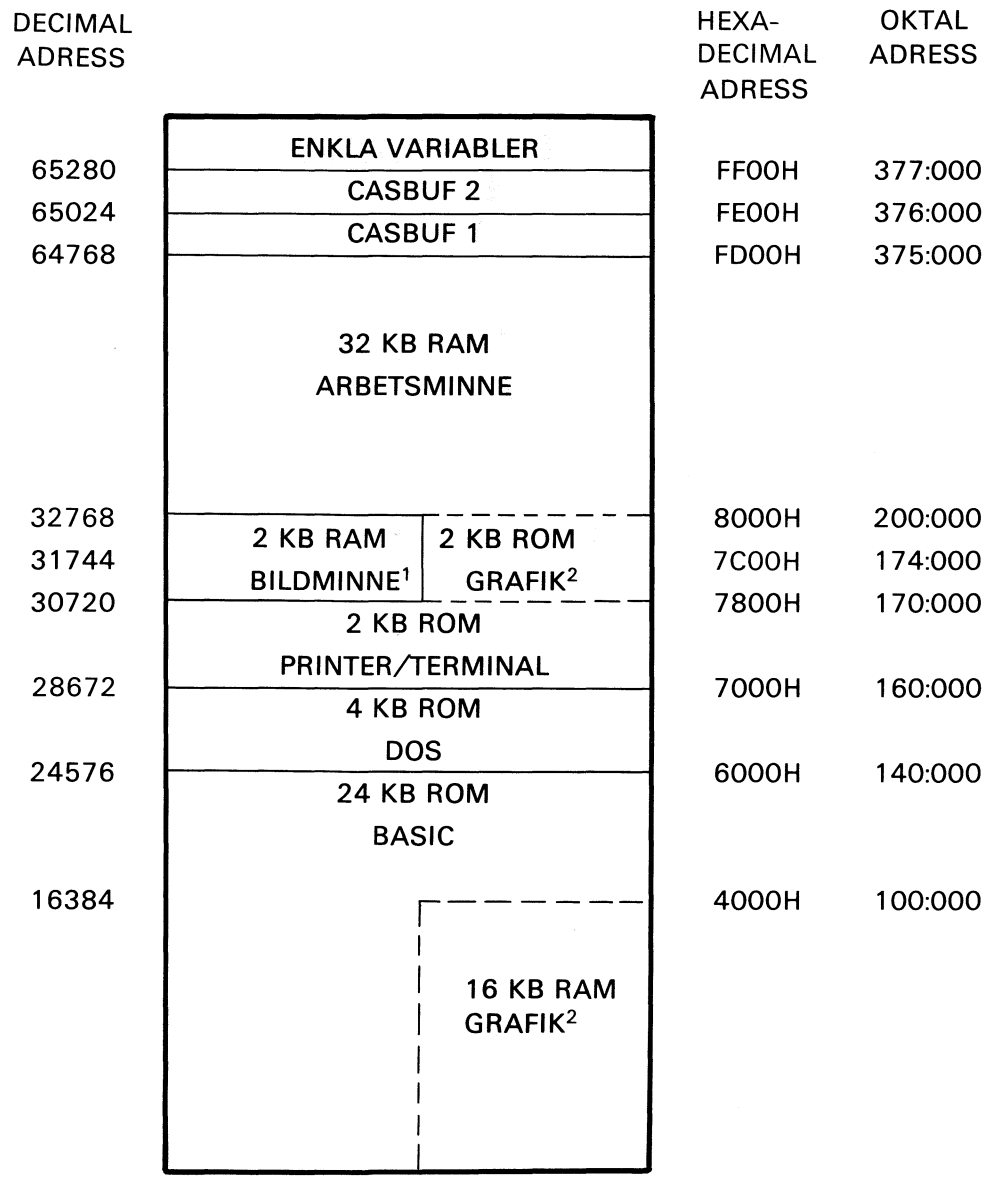
Bilaga 2: I/O-portar i ABC 800

Port	Adress bit 7 bit 0		
CTC	011XXX00	kanal 0	
	011XXX01	kanal 1	
	011XXX10	kanal 2	
	011XXX11	kanal 3	
SIO/2	010XXX00	V24 data	kanal A
	010XXX01	V24 styr	kanal A
	010XXX10	kassett data	
	010XXX11	kassett styr	
DART	0010XX00	skrivare data	kanal B
	0010XX01	skrivare styr	kanal B
	0010XX10	tangentbord data	
	0010XX11	tangentbord styr	

X = godtyckligt värde

Bilaga 3: Minnesdisposition

Minneskarta ABC 800 M/C HR utan flexskiveenhet ansluten

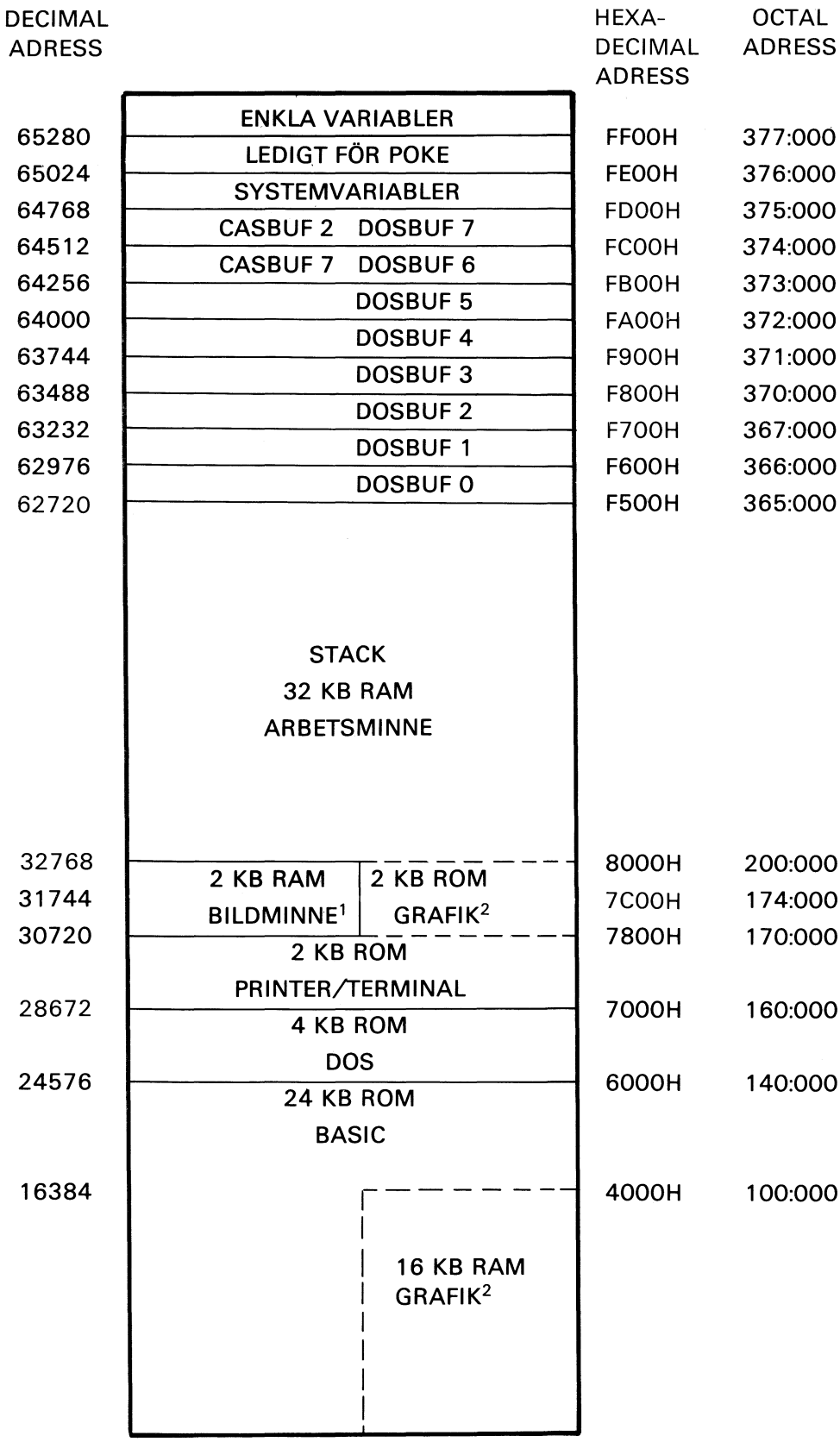


1. ABC 800 C använder endast 1 kB bildminne (31744 - 32786).

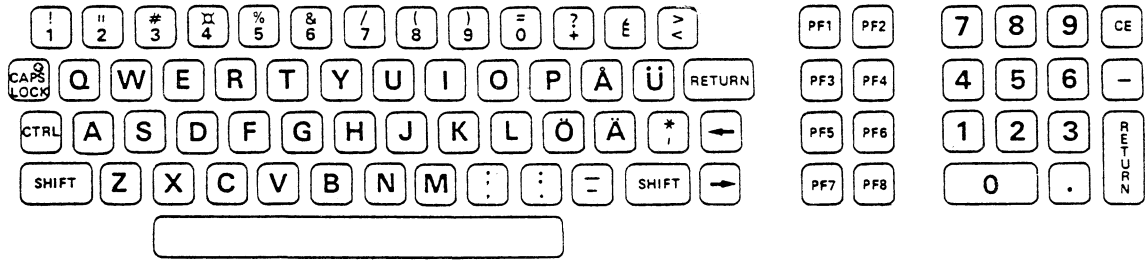
2. Bildminnet (2 kB) på VU-kortet ligger parallellt med systemprogrammet för grafik (2 kB) på PU-kortet. Likaså ligger bildminnet för grafik (16 kB) parallellt med systemprogrammet BASIC. De olika minnesareorna inkräktar dock inte på varandra utan ABC 800 går över i en specialmod, då grafikminnet adresseras.
   Om minnesutrymme ska reserveras för maskinspråksrutiner, ändras följande adresser:

   - Pekare till lägsta minnesadress för BASIC-program(BOTTOM): 65292
   - Pekare till högsta minnesadress för BASIC-program (TOP): 65294

Minneskarta ABC 800 med flexskiveenhet ansluten



Bilaga 4: Tangentbordslayout, ASCII-koder



Koder på tangentbordet

ASCII-kod	Ctrl	Shift	Tangent	ASCII-namn	Funktion
0	X		E	NUL	Tidsutfyllnadstecken
1	X		A	SOH	–
2	X		B	STX	–
3	X		C	ETX	Stoppar exekvering
4	X		D	EOT	–
5	X		E	ENQ	–
6	X		F	ACK	–
7	X		G	BEL	–
8	X		H	BS	*)"←" tangenten
9	X		I	HT	*)"→" tangenten
10	X		J	LF	Radframmatning
11	X		K	VT	–
12	X		L	FF	*)Raderar skärmen
13	X		M	CR	*)"RETURN" tangenten
14	X		N	SO	–
15	X		O	SI	–
16	X		P	DLE	–
17	X		Q	DC1	–
18	X		R	DC2	–
19	X		S	DC3	Stegar en programinstruktion
20	X		T	DC4	–
21	X		U	NAK	–
22	X		V	SYN	–
23	X		W	ETB	–
24	X		X	CAN	*) Tar bort skriven rad
25	X		Y	EM	–
26	X		Z	SUB	–
27	X		Ä	ESC	–
28	X		Ö	FS	–
29	X		Å	GS	–
30	X		ü	RS	–
31	X	X	O	US	–
127	X		<	DEL	Ger fylld kvadrat (■).

*) Dessa tecken påverkar skärmen direkt.

ASCIIKod	T	G	ASCIIKod	T	G	ASCIIKod	T	G	ASCIIKod	T	G
32	Blank		56	8		80	P	P	104	h	
33	!		57	9		81	Q	Q	105	i	
34	"		58	:		82	R	R	106	j	
35	#		59	;		83	S	S	107	k	
36	¤		60	<		84	T	T	108	l	
37	%		61	=		85	U	U	109	m	
38	&		62	>		86	V	V	110	n	
39	'		63	?		87	W	W	111	o	
40	(		64	É	É	88	X	X	112	p	
41	)		65	A	A	89	Y	Y	113	q	
42	*		66	B	B	90	Z	Z	114	r	
43	+		67	C	C	91	Ä	Ä	115	s	
44	,		68	D	D	92	Ö	Ö	116	t	
45	-		69	E	E	93	Å	Å	117	u	
46	.		70	F	F	94	Ü	Ü	118	v	
47	/		71	G	G	95	_	_	119	w	
48	0		72	H	H	96	é		120	x	
49	1		73	I	I	97	a		121	y	
50	2		74	J	J	98	b		122	z	
51	3		75	K	K	99	c		123	ä	
52	4		76	L	L	100	d		124	ö	
53	5		77	M	M	101	e		125	å	
54	6		78	N	N	102	f		126	ü	
55	7		79	O	O	103	g		127		

Koder tolkade i teckenmod (T) och grafmod (G)

Decimala koder från funktionstangenterna

		SHIFT	CTRL	SHIFT + CTRL
PF1	192	208	224	240
PF2	193	209	225	241
PF3	194	210	226	242
PF4	195	211	227	243
PF5	196	212	228	244
PF6	197	213	229	245
PF7	198	214	230	246
PF8	199	215	231	247

Följande kommandon används som styrfunktioner och matas in från tangentbordet:

RETURN	verkställighetskommando
CTRL/C	stoppar exekveringen (OBS! Två CTRL/C avbryter exekveringen.)
CTRL/H eller ←	raderar ett tecken
CTRL/I eller →	används vid editering
CTRL/L	tömmer skärmen
CTRL/S	single-step exekvering
CTRL/X eller CE	raderar inskriven rad

19 Sakregister

G

H

I

K

L

M

N

O

P

R

S

LUXOR
Datorer

Art. nr. 66 79210—12

TEKNOTRANS AB Göteborg 1981/Tryckt av MINAB, Surte 1981