

Elektronik

MIKRODATORTEKNIK

Häfte 17: Utvecklingssystem USE 6800

INNEHÅLL

	sid
17.1 Allmänt	1
17.2 Uppbyggnad	1
17.2.1 CPU-kort (ROF 137 3204)	2
17.2.2 RAM-kort (ROF 137 3201)	4
17.2.3 I/O-kort (ROF 137 3202)	5
17.2.4 FDC-kort (ROF 137 3203)	7
17.2.5 Spänningskort (ROF 137 3299)	7
17.2.6 Flexskivor	8
17.3 Adresskarta	9
17.4 Systemprogram	9
17.5 Monitor-EBUG	10
17.5.1 EBUG:s kommandon	10
17.5.2 EBUG:s data-areor	13
17.6 På- och frånslag av utvecklingssystemet	14
17.7 Formattering av en ny flexskiva	16
17.8 Exempel på hur man skriver ett program	17

17 Utvecklingssystem USE 6800

17.1 Allmänt.

Denna manual beskriver uppbyggnad och handhavande av ett utvecklings-system, som är konstruerat och byggt av IM Ericsson, Östersund, i nära samarbete med Elektronikgruppen vid Fysiska institutionen, Umeå universitet.

Utvecklingssystemet kan användas för utveckling av program för Motorolas mikroprocessorer 6800 och 6809. I denna manual beskrivs vad som gäller, när systemet är avsett att användas för utveckling av 6800-program. Diskoperativsystemet för systemet är då FLEX 2.0 (Se även sid. 17-9) I en annan manual beskrivs de förändringar som uppkommer när systemet skall användas för utveckling av 6809-program.

17.2 Uppbyggnad

Utvecklingssystemet är utrustat med bl a följande hårdvara:

- 1 CPU-kort, med M6800 mikroprocessor (2 MHz utförande)
- 2 RAM-kort, med 64 kb dynamiskt RAM
- 3 I/O-kort, med 3 st PIA (M6821), 2 st ACIA (M6850) och 1 st PTM (M6840)
- 4 FDC-kort, dvs Floppy Disk Controller kort
- 5 Spänningskort
- 6 2 st Floppy-disk-drivers (5 1/4")
- 7 Tangentbordsmodul (hex) med display. Kopplas direkt till I/O-kortet.
- 8 LED-modul med bl a 8 st switchar och 8 st LED. Sätts på I/O-kortet.
- 9 Emulator för 6802 (ej realtid)
- 10 PROM-programmerare för UV-EPROM 2716 (2 kb) och 2732 (4 kb)

Utrustningen under punkt 1-6 beskrivs i detta häfte. Övrig utrustning beskrivs i andra häften.

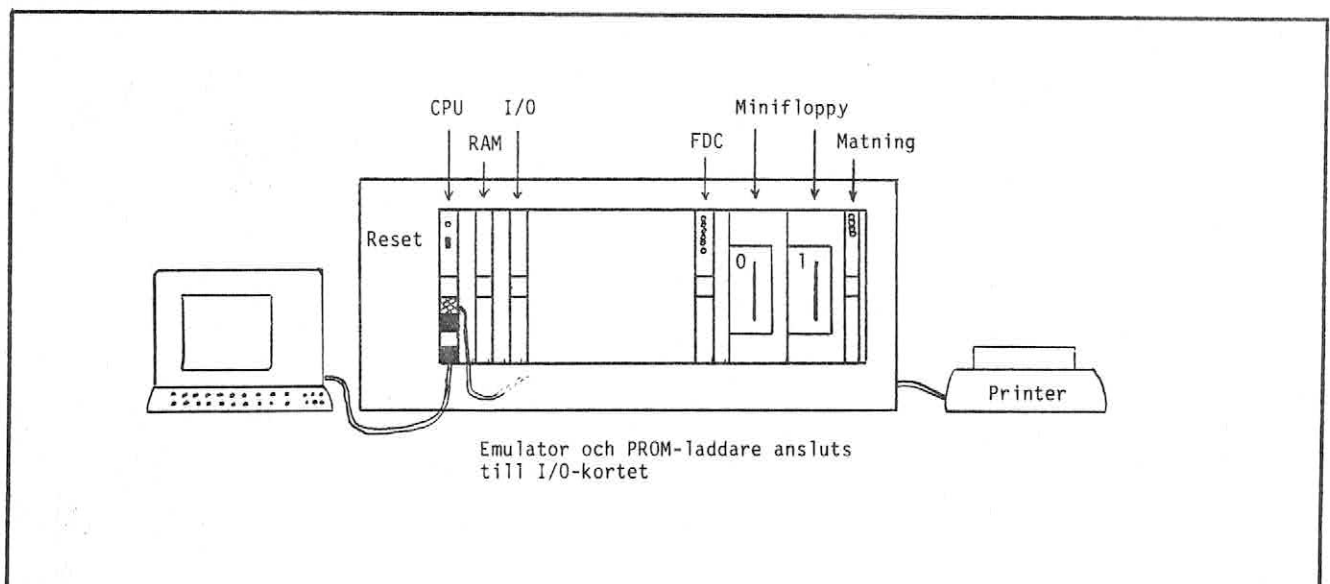


Fig 1. Utvecklingssystem USE 6800 i grundversion

17.2.1 CPU-kort (ROF 137 3204)

CPU-kortet innehåller följande huvudkretsar (2 MHz-utförande):

- M6800 **MPU**-Micro Processor Unit
- M6875 **Tvåfas-klocka**
M6800 behöver två icke överlappande klocksignaler och får dessa från den kristallstyrda klockkretsen M6875. Oscillatorfrekvensen är 4,9152 MHz, som delas ned med en faktor 4. Systemets klockfrekvens är alltså 1,23 MHz.
- M6828 **PIC**-Priority Interrupt Controller
Denna krets används för att generera avbrott (IRQ) i MPU:s huvudprogram efter en viss prioritetsordning. M6828 har 8 prioritetsnivåer och har alltså möjlighet att adressera 8 st olika avbrottsrutiner.
- M6821 **PIA**-Periferial Interface Adapter
PIA M6821 har två portar, A och B. Port A går som ingång och används för att avläsa den BAUD-rate plugg, som sitter på CPU-kortets frontkontakt. (Se sid 17-4). Med olika BAUD-rate pluggar, väljer man alltså den överföringshastighet man vill ha mot bildskärms-terminalen.
Port B går som utgång mot en printer med parallellsnitt. Printerkontakten ansluts på CPU-kortets frontkontakt. (Se sid 17-4).
- M6850 **ACIA**-Asynchronous Communication Interface
Denna krets används för asynkron seriell kommunikation med en bildskärmsterminal. Gränssnittet är RS-232C. Bildskärmsterminalens kontakt ansluts på CPU-kortets frontkontakt. (Se sid 17-4).
- M6840 **PTM**-Programmable Timer Module
Denna timer har tre funktioner. En funktion är att generera en klocksignal till ACIA:n på CPU-kortet (se M6850 ovan). Frekvensen bestäms av BAUD-rate pluggen (se M6821 ovan). En annan funktion är att fungera som RTW = Real Time Watch. Det innebär, att systemet övervakas och om systemet helt spårar ur, görs en automatisk RESET. Den tredje funktionen är att generera avbrott till en printerspooling-rutin. (Ej installerat).
- 2716 **EPROM**-Eraseable Programmable Read Only Memory (2 kbyte)
Det finns två EPROM-kapslar på CPU-kortet. I den ena kapseln finns "Bootstrap-loader" och rutiner för avbrottshantering. I den andra kapseln finns Monitorprogrammet. (Se EBUG, sid 17-10)

Tabellen nedan, visar vilka adresser som gäller för kretsarna på CPU-kortet. (Se även adresskarta på sidan 17-9).

\$E000 - \$E0FF	RAM - 256 bytes. Används av EBUG (se sid 17-13)
\$E100 - \$E3FF	RAM - 768 bytes. Kan utnyttjas av användaren.
\$E400 - \$E7FF	Detta område får ej adresseras, p g a att partiell avkodning används.
\$E800 - \$EBFF	System I/O, dvs PIA, ACIA och PTM (se föregående sida). Adresserna fördelas enligt sammanställningen nedan. Övriga adresser får ej adresseras, p g a att partiell avkodning används.
\$EC00 - \$EFFF	EAK-logik. (EAK = External Acknowledge). Denna logik tar emot kvittenssignaler från adresserade enheter. Om man t ex adresserar en minnescell, som inte är åtkomlig, ges EAK-alarm. På \$E000 ligger adressen för återställning av EAK-alarm. (Se sid 17-13). EAK-logiken är partiellt avkodad.
\$F000 - \$F7FF	EPROM 1. "Bootstrap-loader" och avbrottsrutiner
\$F800 - \$FFFF	EPROM 2. EBUG-monitorprogram (se sid 17-10).

Fig 2. Adresser för CPU-kortets kretsar

PIA 6821

\$E810
\$E810
\$E811
\$E812
\$E812
\$E813

RS1	RS0	Control Register Bit		Location Selected
		CRA-2	CRB-2	
0	0	1	X	Peripheral Register A
0	0	0	X	Data Direction Register A
0	1	X	X	Control Register A
1	0	X	1	Peripheral Register B
1	0	X	0	Data Direction Register B
1	1	X	X	Control Register B

X = Don't Care

\$E821

\$E821

\$E820

\$E820

ACIA
6850

Data Bus Line Number	Buffer Address			
	RS • R/W Transmit Data Register	RS • R/W Receive Data Register	RS • R/W Control Register	RS • R/W Status Register
	(Write Only)	(Read Only)	(Write Only)	(Read Only)
0	Data Bit 0*	Data Bit 0	Counter Divide Select 1 (CR0)	Receive Data Register Full (RDRF)
1	Data Bit 1	Data Bit 1	Counter Divide Select 2 (CR1)	Transmit Data Register Empty (TDRE)
2	Data Bit 2	Data Bit 2	Word Select 1 (CR2)	Data Carrier Detect (DCD)
3	Data Bit 3	Data Bit 3	Word Select 2 (CR3)	Clear to Send (CTS)
4	Data Bit 4	Data Bit 4	Word Select 3 (CR4)	Framing Error (FE)
5	Data Bit 5	Data Bit 5	Transmit Control 1 (CR5)	Receiver Overrun (OVRN)
6	Data Bit 6	Data Bit 6	Transmit Control 2 (CR6)	Parity Error (PE)
7	Data Bit 7***	Data Bit 7**	Receive Interrupt Enable (CR7)	Interrupt Request (IRQ)

* Leading bit = LSB = Bit 0

** Data bit will be zero in 7 bit plus parity modes

*** Data bit is "don't care" in 7-bit plus parity modes

PTM
6840

\$E808
\$E809
\$E80A
\$E80B
\$E80C
\$E80D
\$E80E
\$E80F

Register Select Inputs			Operations	
RS2	RS1	RS0	R/W = 0	R/W = 1
0	0	0	CR20 = 0 Write Control Register #3 CR20 = 1 Write Control Register #1	No Operation
0	0	1	Write Control Register #2	Read Status Register
0	1	0	Write MSB Buffer Register	Read Timer #1 Counter
0	1	1	Write Timer #1 Latches	Read LSB Buffer Register
1	0	0	Write MSB Buffer Register	Read Timer #2 Counter
1	0	1	Write Timer #2 Latches	Read LSB Buffer Register
1	1	0	Write MSB Buffer Register	Read Timer #3 Counter
1	1	1	Write Timer #3 Latches	Read LSB Buffer Register

Fig 3. Adresser för CPU-kortets PIA, ACIA och PTM

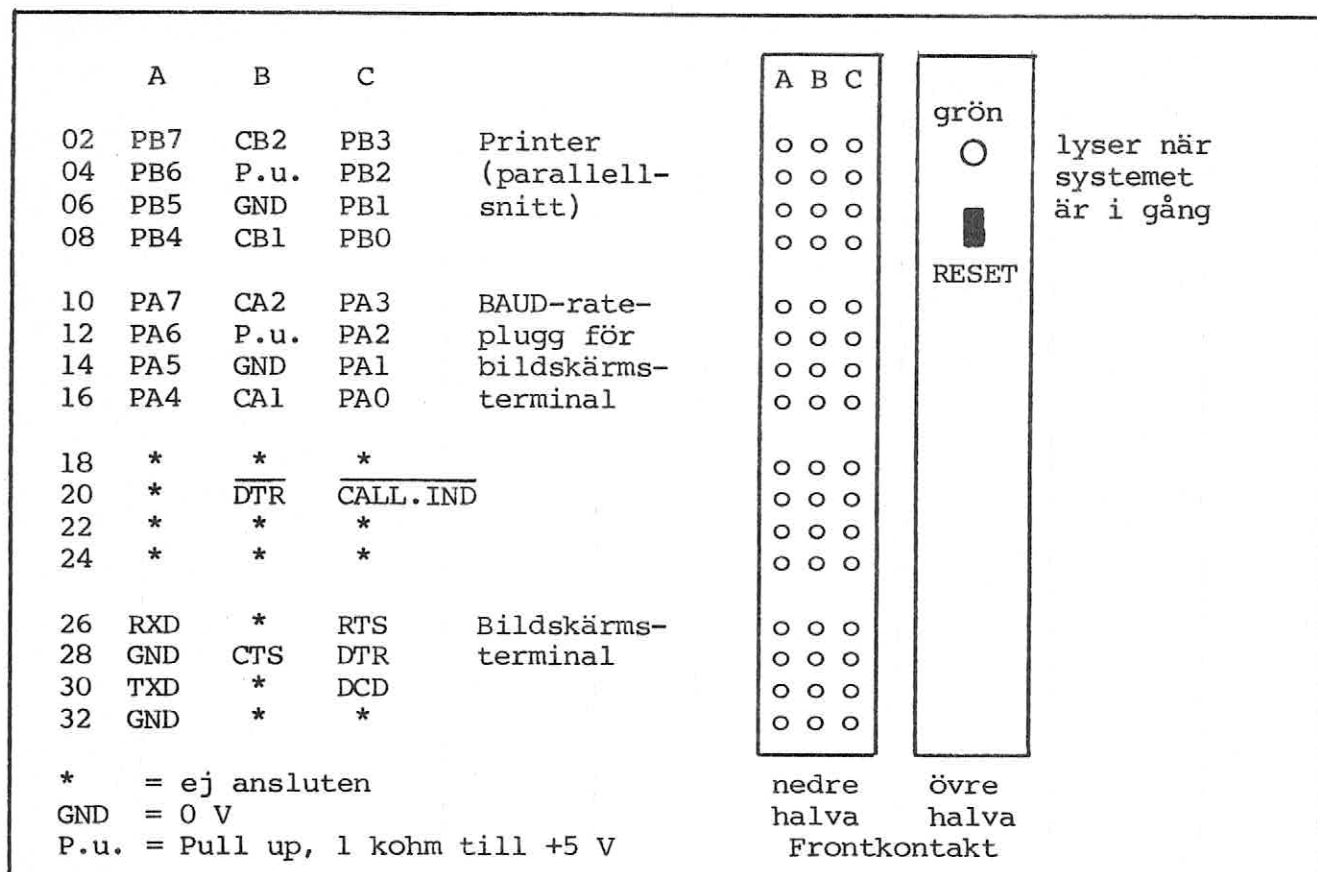


Fig 4. Signaler och frontkontakt på CPU-kortet

17.2.2 RAM-kort (ROF 137 3201)

Minneskortet är på 64 kb, varav 52 kb är tillgängligt. Den övriga minnesarean är tilldelad andra enheter som monitorns EPROM, I/O-kort mm. (Se adresskarta på sid 17-9).

Eftersom minnet är ett dynamiskt RAM, måste minnet "friskas upp" (refresh) med jämna mellanrum, för att minnescellerna skall behålla sin information. Till denna minnesrefresh används en för detta ändamål särskilt avpassad krets, Intel 3242. Denna krets styrs av en oscillator på 25 MHz.

Adressområde för tillgängligt RAM är \$0000 - \$CFFF. (52 kb)

17.2.3 I/O-kort (ROF 137 3202)

I/O-kortet är utrustat med 3 st PIA (M6821), 2 st ACIA (M6850) och 1 st PTM (M6840). Kretsarna används inte alls av systemet och kan därför användas fritt av användaren, som därmed också måste initiera kretsarna. Till I/O-kortet skall t ex Emulator och PROM-programmerare anslutas via I/O-kortets frontkontakt. (Se nästa sida).

De två ACIA-kretsarna utgör seriella portar med gränssnitt enligt RS-232C. Portarna är försedda med anpassningskretsar (1488 - 1489) för TTL- till RS-232C-nivåer

PTM M6840 innehåller tre programmerbara klocksignaler, varav två används för de två ACIA-kretsarna. En timerfunktion kan alltså användas externt via frontkontakten. (Se fig 6 på nästa sida).

Figur 5 visar adresserna till kretsarna på I/O-kortet.

PIA 1 \$D020 - \$D023
PIA 2 \$D024 - \$D027
PIA 3 \$D028 - \$D02B
ACIA 1 \$D02C - \$D02D
ACIA 2 \$D02E - \$D02F
PTM 3 \$D030 - \$D037

PIA 1

RS1	RS0	Control Register Bit		Location Selected
		CRA-2	CRB-2	
0	0	1	X	Peripheral Register A
0	0	0	X	Data Direction Register A
0	1	X	X	Control Register A
1	0	X	1	Peripheral Register B
1	0	X	0	Data Direction Register B
1	1	X	X	Control Register B

X = Don't Care

Data Bus Line Number	Buffer Address			
	RS • R/W	RS • R/W	RS • R/W	RS • R/W
	Transmit Data Register	Receive Data Register	Control Register	Status Register
	(Write Only)	(Read Only)	(Write Only)	(Read Only)
0	Data Bit 0*	Data Bit 0	Counter Divide Select 1 (CR0)	Receive Data Register Full (RDRE)
1	Data Bit 1	Data Bit 1	Counter Divide Select 2 (CR1)	Transmit Data Register Empty (TDRE)
2	Data Bit 2	Data Bit 2	Word Select 1 (CR2)	Data Carrier Detect (DCD)
3	Data Bit 3	Data Bit 3	Word Select 2 (CR3)	Clear to Send (CTS)
4	Data Bit 4	Data Bit 4	Word Select 3 (CR4)	Framing Error (FE)
5	Data Bit 5	Data Bit 5	Transmit Control 1 (CR5)	Receiver Overrun (OVRN)
6	Data Bit 6	Data Bit 6	Transmit Control 2 (CR6)	Parity Error (PE)
7	Data Bit 7***	Data Bit 7**	Receive Interrupt Enable (CR7)	Interrupt Request (IRQ)

* Leading bit = LSB = Bit 0

** Data bit will be zero in 7 bit plus parity modes

*** Data bit is "don't care" in 7-bit plus parity modes

PTM	Register Select Inputs			Operations	
	RS2	RS1	RS0	R/W = 0	R/W = 1
\$D030	0	0	0	CR20 = 0 Write Control Register #3 CR20 = 1 Write Control Register #1	No Operation
\$D031	0	0	1	Write Control Register #2	Read Status Register
\$D032	0	1	0	Write MSB Buffer Register	Read Timer #1 Counter
\$D033	0	1	1	Write Timer #1 Latches	Read LSB Buffer Register
\$D034	1	0	0	Write MSB Buffer Register	Read Timer #2 Counter
\$D035	1	0	1	Write Timer #2 Latches	Read LSB Buffer Register
\$D036	1	1	0	Write MSB Buffer Register	Read Timer #3 Counter
\$D037	1	1	1	Write Timer #3 Latches	Read LSB Buffer Register

Fig 5. Adresser för I/O-kortets PIA, ACIA och PTM

Kretsarna på I/O-kortet är tillgängliga via kortets frontkontakter enligt sammanställningen nedan.

	A	B	C		A B C
02	PB7	CB2	PB3	PIA 1	○ ○ ○
04	PB6	+5 V	PB2		○ ○ ○
06	PB5	GND	PB1		○ ○ ○
08	PB4	CB1	PB0		○ ○ ○
10	PA7	CA2	PA3	PIA 2	○ ○ ○
12	PA6	+5 V	PA2		○ ○ ○
14	PA5	GND	PA1		○ ○ ○
16	PA4	CA1	PA0		○ ○ ○
18	PB7	CB2	PB3	PIA 3	○ ○ ○
20	PB6	P.u.	PB2		○ ○ ○
22	PB5	GND	PB1		○ ○ ○
24	PB4	CB1	PB0		○ ○ ○
26	PA7	CA2	PA3	ACIA 1	○ ○ ○
28	PA6	P.u.	PA2		○ ○ ○
30	PA5	GND	PA1		○ ○ ○
32	PA4	CA1	PA0		○ ○ ○
02	PB7	CB2	PB3	ACIA 2 + PTM 3	○ ○ ○
04	PB6	P.u.	PB2		○ ○ ○
06	PB5	GND	PB1		○ ○ ○
08	PB4	CB1	PB0		○ ○ ○
10	PA7	CA2	PA3	ACIA 1	○ ○ ○
12	PA6	P.u.	PA2		○ ○ ○
14	PA5	GND	PA1		○ ○ ○
16	PA4	CA1	PA0		○ ○ ○
18	RXD	*	RTS	ACIA 2 + PTM 3	○ ○ ○
20	GND	CTS	DTR		○ ○ ○
22	TXD	*	DCD		○ ○ ○
24	GND	*	*		○ ○ ○
26	RXD	PTM G3	RTS	ACIA 2 + PTM 3	○ ○ ○
28	GND	CTS	DTR		○ ○ ○
30	TXD	PTM C3	DCD		○ ○ ○
32	GND	PTM 03	*		○ ○ ○

* = ej ansluten
 GND = 0 V
 P.u. = Pull up, 1 kohm till +5 V

Front-kontakt

Fig 6 I/O-kortets signaler och frontkontakt

17.2.4 FDC-kort (ROF 137 3203)

För styrningen av floppydiskenheter finns ett särskilt kretskort, FDC-kort. (FDC = Floppy Disk Controller). Drivkretsarna klarar dubbelsidiga flexskivor med dubbel densitet, dubbelsidiga flexskivor med enkel densitet och enkelsidiga flexskivor, enkel densitet. När systemet går i 6800-version, kan dubbelsidiga och enkelsidiga flexskivor med enkel densitet användas. Diskkontroller är WD1797 med extern dataseparator

Ett antal lysdioder på FDC-kortets front, visar vilken typ av aktivitet som pågår. (Se figur 7 nedan). Figur 7 visar även FDC-kortets adresser.

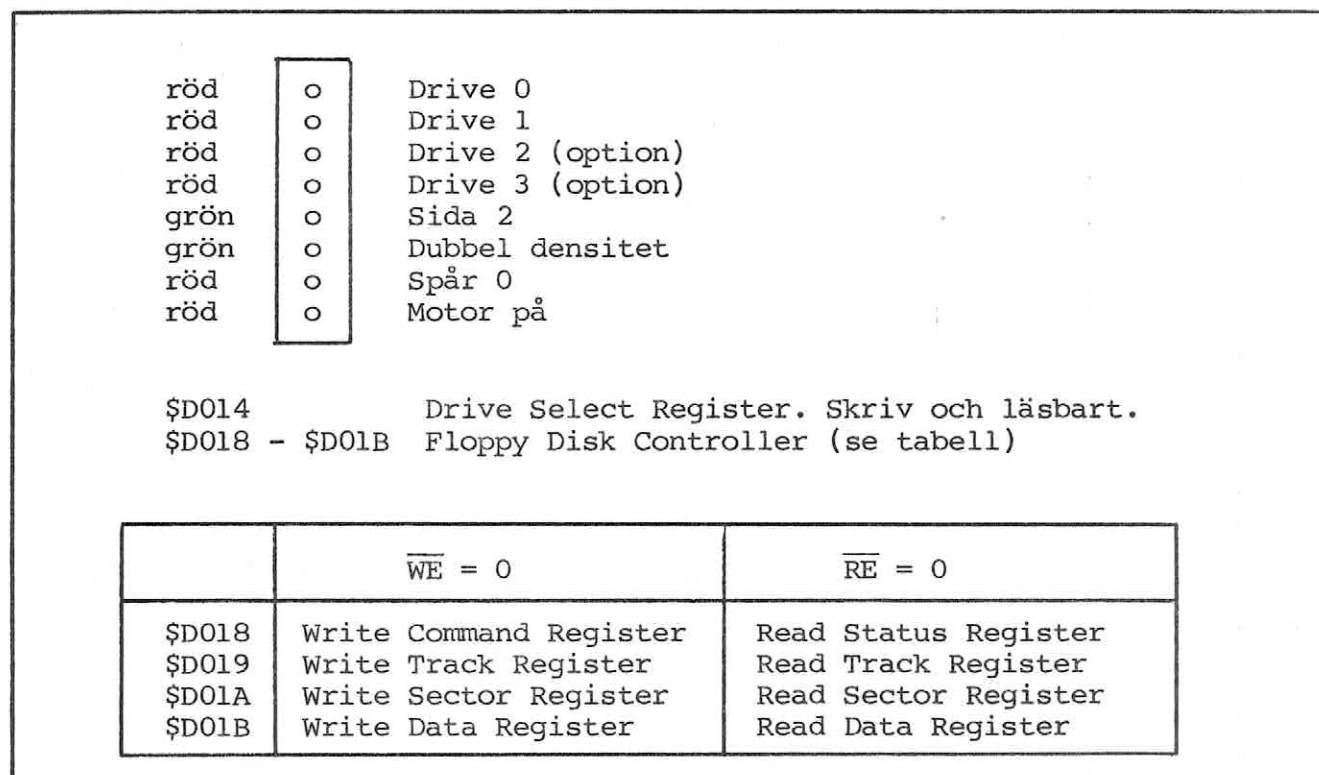


Fig 7. FDC-kortets lysdioder och adresser

17.2.5 Spänningskort (ROF 137 3299)

I systemet används matningsspänningarna +12 V, -12 V och +5 V. Spänningarna +12 V och -12 V behövs bl a för det seriella RS-232C-snittet. Spänningen +5 V finns tillgänglig bl a på frontkontaktarna på både CPU- och I/O-kortet (se sid 17-4 och 17-6). Lysdioder på spänningskortets front, markerar om matningsspänningarna fungerar.

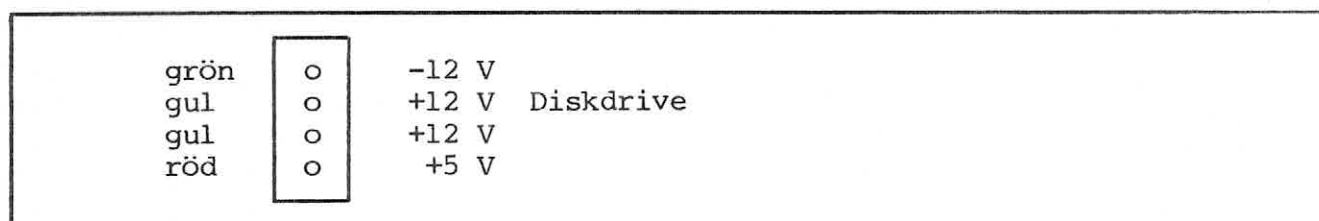


Fig 8. Lysdioder på spänningskortets front

17.2.6 Flexskivor

Sekundärminnet utgörs av två stycken flexskivor (5 1/4"). Normalt har man en systemskiva för systemprogrammen och en arbetsskiva för användarens program. Systemskivan skall sitta i drive nr 0 (vänster drive) och arbetsskivan i drive nr 1 (höger drive). Vid normal körning skall man endast läsa ifrån systemskivan och därför skall man skrivskydda den genom att sätta en ogenomskinlig tejp över ett jack i skivfodralet. (Se figur nedan)

Flexskivan (även benämnd diskett eller floppydisk) utgörs av en tunn plastskiva, som är överdragen med ett tunt magnetiserbart skikt. Skivan ligger alltid innesluten i ett pappersfodral och läsning och eventuell skrivning sker genom en slits i fodralet.

Eftersom skivan plockas in och ut i diskdrivern, måste systemet på något sätt kunna fastlägga skivans exakta läge, så att systemet vet var ett spår börjar. Detta sker med hjälp av ett hål (index-hål) på skivan. I och med att hålet passerar en lysdiod-fotodiod-kombination, vet systemet vilken position skivan har vid en viss tidpunkt. Skivan skall sitta med **skrivskyddet** uppåt och **etiketten** åt höger.

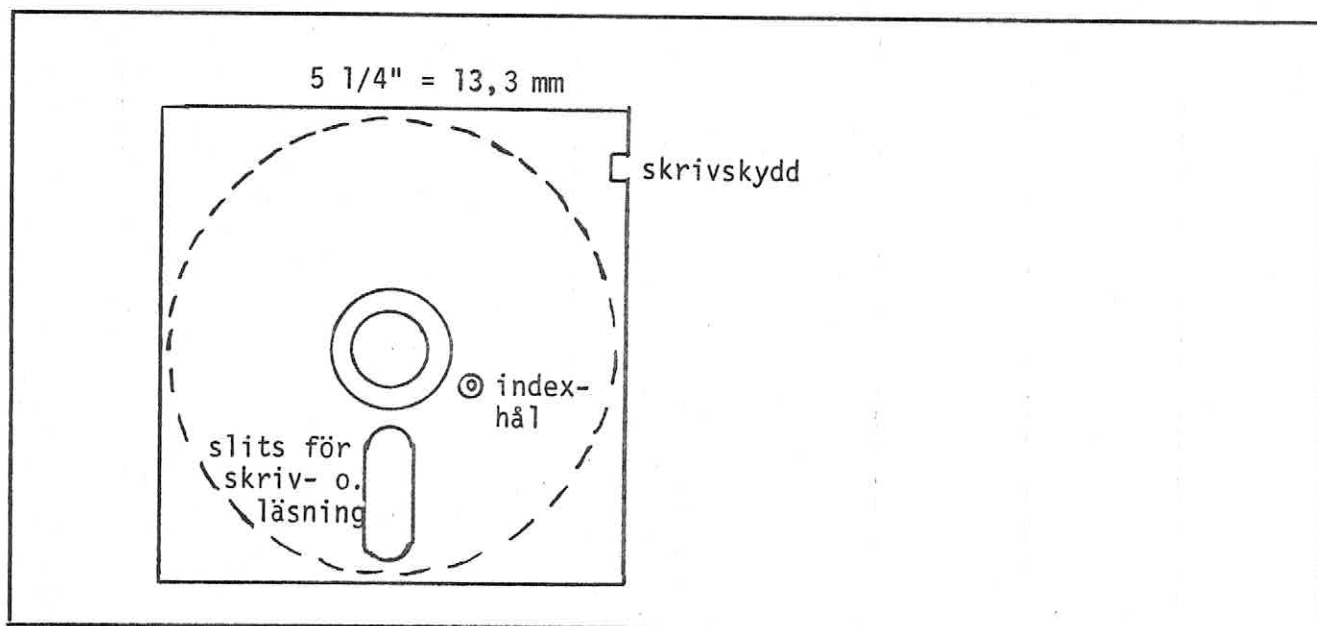


Fig 9 Flexskiva

Om man använder enkelsidiga flexskivor med enkel densitet, kommer sidan att innehålla 80 spår, där varje spår består av 10 sektorer på vardera 256 bytes. En enkelsidig flexskiva med enkel densitet har alltså en total lagringskapacitet på $80 \cdot 10 \cdot 256$ bytes = 200 kb.

Spår 0 är reserverat för systemet. (Se sid 17-16). Det är alltså 79 spår som är tillgängliga för användaren.

På vardera diskdrivern sitter en lysdiod, som lyser när drivern arbetar.
OBS! Öppna inte dörren till en drive, när lysdioden på drivern lyser

17.3 Adresskarta

Systemet har 52 kb RAM åtkomligt. Av detta behöver operativsystemet FLEX 2.0 8 kb (\$A000 - \$BFFF), varför användaren normalt har tillgång till 44 kb RAM (\$0000 - \$9FFF, \$C000 - \$CFFF). Minneskartan nedan visar vad som gäller för systemet.

\$FFFF	EPROM EBUG	Monitor. (Se sid 17-10)
\$F000	Intern I/O	(Se sid 17-3)
\$E000	1 kb RAM	EBUG:s dataarea, \$E000 - \$E0FF
	User I/O	(Se sid 17-5)
\$D000	User RAM	
\$C000		
	RAM/FLEX	FLEX 2.0 operativsystem läggs i adressområdet \$A000 - \$BFFF
\$A000		
	User RAM	Hit kommer även t ex Debuggern (\$4000 - \$6000) (Se häfte 13: Debugger)
\$0000		

Fig 10 Adresskarta för utvecklingssystemet

17.4 Systemprogram

Utvecklingssystemet är utrustat med bl a följande program. Programmen ligger på systemskivan, som skall sitta i drive nr 0.

- | | | | |
|---|--------------------------------------|----|---------------|
| 1 | FLEX 2.0 operativsystem | *) | (Se häfte 10) |
| 2 | Editor (radorienterad) | *) | (Se häfte 11) |
| 3 | Assembler (absolut) | *) | (Se häfte 12) |
| 4 | Debugger | *) | (Se häfte 13) |
| 5 | Emulator (ICE = In circuit emulator) | | (Se häfte 15) |

* Programmen är utvecklade av TSC. (TSC = Technical System Consultants)
FLEX 2.0 är omkonfigurerad, för att passa till USE 6800.

17.5 Monitor - EBUG

På CPU-kortet finns det två EPROM-kapslar. (Se sid 17-3). I dessa finns ett program, som bl a initierar utvecklingssystemet, när man slår på strömmen. Detta program, som kallas EBUG, kommer vid påslag av systemet bl a att läsa av den BAUD-rate-plugg som sitter på CPU-kortet. (Se sid 17-2). Därefter initieras ACIA:n och PTM:n på CPU-kortet, så att kommunikation med bildskärmsterminalen kan ske. Efter själva initieringen lägger sig EBUG i ett vänteläge, vilket markeras med att EBUG:s prompt * kommer upp på skärmen.

När EBUG:s prompt * visar sig på skärmen, kan ett antal kommandon som finns i EBUG utnyttjas. (Se nedan). Det vanligaste kommandot är E, som medför att ett "boot"-program som ligger på spår 0 på systemskivan laddas till RAM. Detta "boot"-program laddar sedan i sin tur ner operativsystemet FLEX 2.0 till RAM i adressområdet \$A000 - \$BFFF. (Se sid 17-9)

17.5.1 EBUG:s kommandon

I detta avsnitt beskrivs EBUG:s kommandon i bokstavsordning. Följande kommandon finns. (Dessa kan alltså ges, om EBUG:s prompt * är uppe).

D	Dump av minnesinnehåll
E	Laddar ner operativsystemet FLEX 2.0 från systemskiva till RAM
F	Finn bitmönster i minnet
G	Gå till adress och exekvera koden
J	Exekvera subrutin
K	Kontrollsummering av minnesinnehåll
M	Manipulering av minnesinnehåll
R	Registerutskrift
T	Test av minne

D Dump av minnesinnehåll

Format: D a1 a2 a1 = startadress för dump
 a2 = stoppadress för dump

D skriver ut innehållet i angiven minnesarea, både i hexadecimal form och som ASCII-tecken.

Ex. *D 0000 0100 Skriver ut innehållet i minnesarean \$0000 - \$0100

D-rutinen är skriven som en subrutin och kan därför även anropas från egna program, t ex vid "avlusning". Anrop sker då via anropsvektorn \$F842. Observera, att om man anropar D-rutinen från egna program, måste start- och stoppadresserna först läggas in i BEGA och ENDA. (Se sid 17-13)

E Laddar operativsystemet FLEX 2.0 från systemskiva (nr 0) till RAM

Format: E

Operativsystemet FLEX 2.0 ligger på systemskivan (nr 0). Vill man utnyttja operativsystemet och det är det normala, måste det laddas ner i RAM. Detta sker med hjälp av ett särskilt "boot"-program, som också ligger på systemskivan. (Spår 0). Kommandot E laddar alltså mer "boot"-programmet, som i sin tur laddar ner FLEX 2.0 till RAM-arean \$A000 - \$BFFF.

F Finn bitmönster i minnet

Format: F n m al a2 n = antal bytes i bitmönstret
 m = det sökta bitmönstret (1-3 bytes = 2-6 hextecken)
 al = startadress för sökningen
 a2 = stoppadress för sökningen

F-kommandot söker igenom angivet adressområde och jämför innehållet i varje adress med angivet bitmönster. Om mönstret påträffas, skrivs adressen ut där mönstret börjar.

Ex. *F 3 BD2000 0000 0100

Adressområdet \$0000 till \$0100 söks igenom och om \$BD2000 påträffas, skrivs adressen ut där \$BD ligger.

G Gå till adress och exekvera koden

Format: G a a = startadress

Om man har laddat ned FLEX 2.0 i RAM via *E och sedan går tillbaka till EBUG via FLEX-kommandot MON (se häfte 10), kan man åter gå tillbaka till FLEX 2.0 på följande sätt:

Ex. *G AD03 \$AD03 är "varmstart"-adressen för FLEX 2.0.

J Exekvera subrutin

Format: J a a = startadress

J startar exekvering av en subrutin (dvs en rutin som avslutats med RTS) på den adress, som anges med a.

K Kontrollsummering av minnesinnehåll

Format: K al a2 al = startadress för kontrollsummeringen
 a2 = stoppadress för kontrollsummeringen

K-kommandot summerar minnesinnehållet f o m adress al t o m adress a2 och skriver ut erhållen summa i hexadecimal form.

M Manipulering av minnesinnehåll

Format: **M a** a = startadress

M-kommandot skriver ut innehållet i angiven adress.

Ex. ***M 01F0 7E** (\$7E = innehållet i adressen \$01F0)

Efter det att innehållet har skrivits ut, kan man välja mellan några olika alternativ enligt punkterna nedan. Vi utgår i varje alternativ ifrån exemplet ovan.

1. Vi vill titta på **nästa** adress.

Tryck på tangenten **LINE FEED**

Innehållet i nuvarande adress (\$01F0) lämnas oförändrat och nästa adress (\$01F1) och dess innehåll skrivs ut.

2. Vi vill titta på **föregående** adress.

Tryck på tangenten **Shift 6 (pil uppåt)**.

Innehållet i nuvarande adress (\$01F0) lämnas oförändrat och föregående adress (\$01EF) och dess innehåll skrivs ut.

3. Vi vill **ändra** innehållet i angiven adress.

Skriv det nya värdet efter det gamla.

Ex. ***M 01F0 7E 65**

\$01F0 har nu innehållet \$65. Endast hexadecimala tecken är tillåtna. Slår man ett icke hexadecimalt tecken, avbryts kommandot. (Se punkt 5)

4. Vi vill veta **offset** från angiven adress (\$01F0) till en annan adress (\$0230).

Tryck på tangenten **O (bokstaven O)** följt av en absolut hexadecimal adress. Minnesinnehållet påverkas ej.

Ex. ***M 01F0 7E 00230 3F**

5. Vi vill **avbryta** **M**-kommandot.

Slå ett icke hexadecimalt tecken, t ex mellanslag.

R Registerutskrift

Format: R

R-kommandot används för att skriva ut stackens innehåll och är användbart efter avbrott. Utskrift sker enligt nedanstående lista:

CC Condition Code Register
 B Accumulator B
 A Accumulator A
 X Index Register
 PC Program Counter
 SP Stack Pointer

T Test av minne

Format: T al a2 al = startadress för test
 a2 = stoppadress för test

Detta är ett enkelt test av RAM-minnet (minneskapslarna). Först laddas minnescellerna med \$FF och efter ca 2 s görs en verifiering. Därefter upprepas proceduren med värdet \$00.

Om det är något fel på en minnescell, skrivs cellens adress ut, samt inskrivet och avläst värde. Testet fortsätter tills hela den angivna minnesarean är testad, oavsett om fel upptäcks eller ej.

17.5.2 EBUG:s data-areor

\$E000		IRQ vektor	
\$E002		Interruptvektor 0	vektor
\$E004		"	1 "
\$E006		"	2 "
\$E008		"	3 "
\$E00A		"	4 "
\$E00C		"	5 " Printerspooling, PTM
\$E00E		"	6 "
\$E010		"	7 " EAK-alarm (högsta prioritet)
\$E012		NMI-vektor	
\$E014		SWI-vektor	
\$E016	BEGA	Startadress	
\$E018	ENDA	Slutadress	
\$E01E	ACIAP	Pekare för in- och utmatning	
\$E020	USERPS	Egna programs startadress	
\$E022	USERCT	Egen kommandotabells startadress	
\$E024	CRNULS	Antal nollor i 'CR'-rutinen	
\$E02A	EKO	Ekoflagga: '0' = Full duplex, '1' = Half duplex	
\$E0C6	Stack	Stackens startadress	

17.6 På- och frånslag av utvecklingssystemet

När man slår på strömmen, måste systemet initieras innan det kan användas. Denna initiering görs av Monitorprogrammet EBUG. (Se sid 17-10) EBUG har bl a till uppgift att föra över ett "boot"-program från systemskivan till RAM. "Boot"-programmet i sin tur laddar ner operativsystemet FLEX 2.0 till RAM. Det normala är alltså, att man arbetar med hjälp av operativsystemet FLEX 2.0 (Se häfte 10).

Starta systemet på följande sätt:

- 1 Se till att disk-driver dörrarna är **öppna**. Flexskivorna får dock ligga i respektive drive. Skivorna skall ligga med **skrivskyddet** **uppat** och **etiketten** åt höger.
- 2 Slå på strömmen till terminal och printer (om en sådan är inkopplad)
- 3 Slå på strömmen till systemet
- 4 På terminalen skall det nu stå:

```
EBUG 1.0
*
```

- 5 Stäng dörren till respektive drive. Tryck ned tangenten E.

Efter en stund svarar systemet med FLEX 2.0 och frågar efter dagens datum. Knappa in månad, dag och år (observera ordningsföljden). På skärmen skall nu FLEX-prompten (+++) komma upp.

```
EBUG 1.0
*E
FLEX 2.0
```

```
DATE (MM,DD,YY)? 01,15,85    (Datum lagras i $ACOE. Se sid 10-32)
```

```
+++
```

Systemet är nu klart att användas.

- OBS 1! Tag **alltid** av båda luckorna till utvecklingssystemet. Överhettning av systemet kan annars inträffa.
- OBS 2! Lämna aldrig utrustningen för någon längre tid, utan att du ligger i FLEX +++ eller i EBUG *. Då gör det ingenting om strömmen bryts (av misstag eller av t ex strömavbrot).
- OBS 3! Gör ingenting så länge systemet arbetar med diskarna, dvs så länge en lysdiod lyser på någon av drivenheterna.

VÄND!

OBS 4! Man kan byta en flexskiva, utan att man behöver slå av systemet.
Se dock till att:

- 1 Lysdioderna på drivenheterna är släkt
- 2 Du befinner dig i FLEX +++ eller EBUG *

OBS 5! Om ett program sparat ur eller fastnat, tryck då på RESET-knappen på CPU-kortet. (Se sid 17-4). Då hamnar du åter i EBUG *.

Efter det att man gjort en RESET, kan man återvända till FLEX 2.0 på två sätt:

- 1 "Varmstart". Slå *G AD03. \$AD03 är adressen till FLEX 2.0.
(Se sid 17-11).
- 2 "Kallstart". Slå *E och datum enligt punkt 5 på föregående sida.

OBS 6! Ta **aldrig** ut ett kort ur systemet, om strömmen är på.
Kretsarna går sönder!

Slå av systemet på följande sätt:

- 1 Se till att du ligger i FLEX +++ eller EBUG *
- 2 Kontrollera att lysdioderna på de båda drivenheterna är släkt.
Öppna dörrarna till respektive drive.
- 3 Slå av strömmen till utvecklingssystemet
- 4 Slå av strömmen till terminal och printer.

17.7 Formattering av en ny flexskiva

En ny, icke använd flexskiva, kan inte användas, förrän den har formatterats. Detta gör man med hjälp av FLEX-kommandot **NEWDISK, <drive>** där <drive> = nummer på den drive där den nya skivan ligger.

Efter kommandot **NEWDISK,1** ställer systemet i tur och ordning följande frågor: ARE YOU SURE? SCHRATCH DISK IN DRIVE 1? DOUBLE SIDED DISK? VOLUME NAME? VOLUME NUMBER? Vi besvarar frågorna på lämpligt sätt och kan alltså nu ge skivan ett namn och ett ordningsnummer.

Efter dessa frågor, börjar **NEWDISK** undersöka skivan, för att se om det finns några felaktigheter på den. Om felaktiga sektorer upptäcks, meddelas detta. Även om fel finns, kan dock skivan användas, eftersom **NEWDISK** plockar bort felaktiga sektorer från den s k fria listan.

Eftersom formatteringen är en komplicerad process, kan den misslyckas även om det inte är något allvarligt fel på skivan. Om formatteringen avbryts, bör man göra åtskilliga nya försök innan man kasserar skivan.

Om formatteringen lyckas, får man ett meddelande om hur många tillgängliga sektorer skivan har och därefter är skivan klar att användas som **arbetsskiva**.

Spår 0 på skivan intar en särställning. Inga sektorer på spår 0 är tillgängliga som datafiler, utan spår 0 är helt reserverad för andra ändamål. De två första sektorerna på spår 0 innehåller "boot"-programmet. På spår 0 finns också bl a spår- och sektorsadresser till första och sista sektorn i kedjan av fria sektorer. På spår 0 ligger dessutom 'directory' till alla filer på skivan.

Direkt efter formatteringen har alla övriga spår på skivan samma format.

Byte 0-1	Spår- och sektorsnummer till nästa sektor i kedjan av fria sektorer.
Byte 2-3	Här skall det logiska ordningsnumret på sektorn lagras, när sektorn ingår i en fil.
Byte 4-255	Datasektorer. Omedelbart efter formatteringen har de värdet noll.

När skivan börjar användas och data lagras i en sektor, kommer länk-adressen i början av sektorn att ändras, så att den pekar på nästa sektor i filen och inte på nästa sektor i den fria listan. I sista sektorn i filen har byte 0 och 1 värdet noll, vilket fungerar som en 'End of file' markering. (De sektorer på spår 0 som lagrar 'directory', har liknande länkadresser).

Om en ny, icke formatterad skiva skall användas som **systemskiva**, gör man på följande sätt: (Något förkortat redovisat).

- 1 Sätt "gamla" systemskivan i drive nr 0 och "nya" skivan i drive nr 1.
- 2 **+++NEWDISK,1**
- 3 **+++COPY,0,1**
- 4 **+++LINK,1.FLEX**
- 5 Prova att "boota" den nya skivan.

Efter detta har vi formatterat en ny skiva, kopierat en "gammal" systemskiva och via **LINK** talat om för "boot"-programmet på spår 0 var **FLEX 2.0** ligger på skivan.

17.8 Exempel på hur man skriver och kör ett program

Vi skall nu skriva och köra ett litet program, för att få en demonstration av några vanliga kommandon.

Uppgift: Skriv ett program, som adderar de decimala talen 15 och 39 och visar summan på skärmen.

Vi kallar vårt program för 'ADD'. Programmet skall skrivas i Editorn (Se häfte 11) och om vi utgår ifrån att vi ligger i FLEX 2.0, kan vi anropa Editorn:

+++EDIT,ADD

I och med detta, har vi skapat en textfil med namnet **ADD.TXT**. Denna fil ligger på arbetsskivan (nr 1). Programmet skrivs dock i RAM och filen ADD.TXT får inte något innehåll förrän vi går ut ur Editorn.

Vi skriver vårt program i Assemblerspråk. (Se häfte 12). Varje rad avslutas med att vi trycker på CR-tangenten. Om man skriver fel och upptäcker detta innan man tryckt på CR-tangenten, kan man ta bort en eller flera av de sist inslagna karaktärerna, genom att trycka på CTRL- och H-tangenterna samtidigt. (Se sid 11-32)

+++EDIT,ADD

NEW FILE:

1.00=	NAM	ADD	SE SID 12-14
2.00=*			SE SID 12-6
3.00=*	Programmet adderar tva decimala tal, 15 och 39		
4.00=*	Summan laggs i en minnescell med namnet SUM		
5.00=*	Summan skrivs också ut på skärmen		
6.00=*	1985-01-15. Version 1. Kalle Pettersson		
7.00=*			
8.00=	ORG	\$1000	SE SID 12-15
9.00=OUTHEX	EQU	\$AD3C	SE SID 10-35
10.00=PCRLF	EQU	\$AD24	SE SID 10-34
11.00=FLEX	EQU	#AD03	"VARMSTARTADRESSEN" FOR FLEX 2.0
12.00=*			
13.00=SUM	RMB	1	SE SID 12-16
14.00=*			
15.00=START	LDAA	#\$15	
16.00=	LDAB	#\$39	
17.00=	ABA		
18.00=	DAA		
19.00=	STAA	SUM	
20.00=	JSR	PCRLF	NY RAD
21.00=	LDX	#SUM	
22.00=	JSR	OUTHEX	SKRIVER SUMMAN PÅ SKARMEN
23.00=	JMP	FLEX	TILLBAKA TILL FLEX
24.00=	END	START	SE SID 12-12
25.00=#			
24.00=	END	START	SE SID 12-12

#

När programmet är färdigskrivet, anger vi detta med tecknet # i första kolumnen och slår <CR>. Systemet svarar då med att skriva ut sista raden (rad 24) en gång till. Därefter kommer prompten för Editorn upp, dvs #.

Vi har alltså nu Editorns prompt # på skärmen. Om vi redan nu vet att det finns något fel i programmet (t ex på grund av felskrivning), går vi givetvis in och ändrar detta fel med hjälp av de kommandon som finns beskrivna i häfte 11.

I detta fall tycker vi emellertid, att programmet ser riktigt ut och vill därför assemblera programmet, dvs översätta programmet från assembler-språk (källkod) till maskinspråk (objektkod).

Vi går därför ut ur Editorn med kommandot S (eller STOP) (Se sid 11-22)

#S
+++

Programmet har nu förts över till arbetsskivan och till den fil som kallas ADD.TXT och som vi skapade med kommandot +++EDIT,ADD. Filen ADD.TXT har alltså nu också fått ett innehåll, dvs vårt program.

Vi ligger nu åter i FLEX +++ och ger order om assemblering.

+++ASMB,ADD,+N (N = radnummer skrivs ut. Se sid 12-8, 12-9.)

Assemblerlistan skrivs nu ut på skärmen. Eftersom hela programmet normalt inte ryms på skärmen, följer vi utskriften för att se om det skrivs ut felmeddelanden. Om ett sådant felmeddelande skrivs ut, stoppar vi utskriften genom att trycka på CTRL- och C-tangenterna samtidigt. När vi har noterat felet, trycker vi på samma tangenter en gång till och utskriften fortsätter. I vårt fall får vi då följande lista:

```

1          NAM      ADD      SE SID 12-14
2          *                SE SID 12-6
3          * Programmet adderar tva tal, 15 och 39
4          * Summan laggs i en minnescell med namnet SUM
5          * Summan skrivs också ut på skärmen
6          * 1985-01-15. Version 1. Kalle Pettersson
7          *
8 0100          ORG      #0100      SE SID 12-15
9 AD3C          OUTHEX EQU      #AD3C      SE SID 10-35
10 AD24          PCRLF  EQU      #AD24      SE SID 10-34
11 0000          FLEX   EQU      #AD03      "VARMSTARTADRESSEN" FOR FLEX 2.0
** ILLEGAL CHARACTER FOR SPECIFIED BASE
12          *
13 0100          SUM     RMB      1          SE SID 12-16
14          *
15 0101 86 15    START  LDA A   ##15
16 0103 C6 39    LDA B   ##39
17 0105 1B       ABA
18 0106 19       DAA
19 0107 B7 01 00 STA A   SUM
20 010A BD AD 24 JSR     PCRLF      NY RAD
21 010D CE 01 00 LDX     #SUM
22 0110 BD AD 3C JSR     OUTHEX     SKRIVER UT SUMMAN PÅ SKARMEN
23 0113 7E 00 00 JMP     FLEX      TILLBAKA TILL FLEX
24          END      START      SE SID 12-12

```

ERROR(S) DETECTED

SYMBOL TABLE:

FLEX 0000 OUTHEX AD3C PCRLF AD24 START 0101 SUM 0100

På rad 11 finns ett fel angivet och vi ser nu att vi råkat skriva # i stället för \$. Vi måste rätta till detta fel och går därför tillbaka till Editorn, dvs

+++EDIT,ADD

Innehållet i filen ADD.TXT förs nu över från arbetsskivan till RAM (Editor-bufferten) och dessutom kommer filen ADD.TXT på arbetsskivan att byta namn till **ADD.BAK** (BAK = BACK UP). Det gamla programmet finns alltså kvar på arbetsskivan under namnet ADD.BAK och en ny fil med namnet ADD.TXT skapas på skivan. I denna fil skall så småningom det nya, rättade programmet in.

Eftersom vi nu har en kopia av det gamla programmet undanstoppat i back-up-filen ADD.BAK, är det ingen katastrof om vi under rättningsarbetet t ex råkar ta bort hela programmet i stället för en rad. Vi kan ju gå tillbaka och hämta det gamla felaktiga programmet och börja om med rättningen från början. Hur detta exakt skall göras tas inte upp här.

Vi har nu efter anropet av Editorn fått upp Editorns prompt på skärmen och kan rätta felet på t ex följande sätt:

- 1 Vi anropar rad 11. (Se sid 11-29)
Systemet svarar med att rad 11 skrivs ut.
- 2 Vi rättar felet. (Se sid 11-25)
Systemet svarar med att skriva ut den ändrade raden, så att vi kan kontrollera att det blev som vi hade tänkt oss.
- 3 Vi går ut ur Editorn. (Se sid 11-22)

#11P

11.00=FLEX EQU #AD03 "VARMSTARTADRESSEN" FOR FLEX 2.0

#C/#/\$

11.00=FLEX EQU \$AD03 "VARMSTARTADRESSEN" FOR FLEX 2.0

#S

+++

Det rättade programmet finns nu i filen ADD.TXT och det gamla programmet finns i filen ADD.BAK.

Vi vill nu assemblera det rättade programmet. När vi ger kommando om detta, ser det ut så här på skärmen:

+++ASMB,ADD,+N (Alt. **+++ASMB,ADD,+NY** Se sid 12-8, 12-9)
DELETE OLD BINARY (Y-N)?

När vi assemblerade programmet ADD.TXT första gången, skrevs en assemblerlista ut på skärmen enligt föregående sida. Den översättning som assemblern då gjorde från assemblerspråkets symbolkod till maskinkod finns utskriven på listan i hex-form, men dessutom lagrades maskinkoden i binär form på en s k binärfil, med namnet **ADD.BIN**.

När vi nu efter rättningen på nytt vill assemblera ett program med namnet ADD.TXT, vill systemet veta om vi vill ha kvar den gamla binärfilen eller om den skall tas bort (=DELETE).

Eftersom det var fel på programmet, är det också fel i maskinkoden och vi svarar därför Y (=YES). Därefter startar assembleringen och vi får en utskrift enligt nedan.

```

1          NAM      ADD      SE SID 12-14
2          *                      SE SID 12-6
3          * Programmet adderar två tal, 15 och 39
4          * Summan laggs i en minnescell med namnet SUM
5          * Summan skrivs också ut på skärmen
6          * 1985-01-15. Version 1. Kalle Pettersson
7          *
8 0100          ORG      $0100      SE SID 12-15
9 AD3C          OUTHEX EQU      $AD3C      SE SID 10-35
10 AD24         PCRLF  EQU      $AD24      SE SID 10-34
11 AD03         FLEX   EQU      $AD03      "VARMSTARTADRESSEN" FOR FLEX 2
12          *
13 0100         SUM     RMB      1          SE SID 12-16
14          *
15 0101 86 15   START  LDA  A  ##15
16 0103 C6 39   LDA  B  ##39
17 0105 1B      ABA
18 0106 19      DAA
19 0107 B7 01 00 STA  A  SUM
20 010A BD AD 24 JSR   PCRLF      NY RAD
21 010D CE 01 00 LDX   #SUM
22 0110 BD AD 3C JSR   OUTHEX     SKRIVER UT SUMMAN PÅ SKÄRMEN
23 0113 7E AD 03 JMP    FLEX      TILLBAKA TILL FLEX
24          END     START      SE SID 12-12

```

NO ERROR(S) DETECTED

SYMBOL TABLE:

```

FLEX  AD03  OUTHEX AD3C  PCRLF  AD24  START  0101  SUM  0100

```

Inga felmeddelanden ges och programmet innehåller därför inga formella fel (syntaxfel). Detta är dock ingen garanti för att programmet fungerar, eftersom det kan finnas logiska fel. Vi testar programmet genom att göra en första provkörning.

+++1.ADD.BIN

54

+++

När programmet körs, svarar systemet med att skriva ut 54 på skärmen, dvs summan av de två tal som vi skulle addera. Programmet fungerar alltså. Eftersom programmet avslutades med JMP FLEX, är vi nu tillbaka i FLEX 2.0.

Vill vi ha en papperskopia på programmet, kopplar vi in en printer och ger kommando om en utskrift: (P = Print). Om utskriften inte kommer igång, kontrollera på printern att "ON LINE"-knappen är i läge ON och att READY lyser.

+++P,ASMB,ADD,+NB (B = ingen ny binärfil skapas. Vi har ju redan en).

Sammanfattning av de kommandon som ges, när programmet för exemplet på sidan 17-17 skrivs:

EBUG 1.0

***E**

(FLEX 2.0 går till RAM, \$A000)

FLEX 2.0

DATE (MM,DD,YY)? 01,15,85

(Datum lagras i \$ACOE)

+++EDIT,ADD

(Textfilen **ADD.TXT** skapas på arbetsskivan)

NEW FILE

1.00=

NAM

ADD

(Programmet ADD skrivs i RAM)

-

24.00=

END

START

25.00=#

24.00=

END

START

#S

(Prog. ADD förs till **ADD.TXT** på arbetsskivan)

+++ASMB,ADD,+N

(Binärfilen **ADD.BIN** skapas på arbetsskivan och får innehåll = maskinkod)

(Assemblerlista med felangivelse på rad 11)

+++EDIT,ADD

(Filen **ADD.TXT** byter namn till **ADD.BAK**.
En ny fil med namnet **ADD.TXT** skapas)

#11P

11.00=FLEX

EQU

#AD03

"VARMSTARTADRESSEN" FOR FLEX 2.0

#C/#/\$

11.00=FLEX

EQU

\$AD03

"VARMSTARTADRESSEN" FOR FLEX 2.0

#S

(Det rättade programmet går till **ADD.TXT**)

+++ASMB,ADD,+N

DELETE OLD BINARY (Y-N)? Y

(Binärfilen **ADD.BIN** får nytt innehåll)

Ny assemblerlista. Inga fel rapporteras.

+++1.ADD.BIN

54

(Programmet **ADD.BIN** förs från skiva 1 till RAM och exekveras)

+++P,ASMB,ADD,+NB

+++

(Assemblerlista med radnummer (N), skrivs ut på printern. Ingen ny binärfil skapas (B))