

Buffered Commands ('E' prompt)

When loading a new file or exiting the editor you will be prompted to save the current buffer if it has been modified since last saved. Any response other than an 'N' or 'n' will save the old buffer. A response of 'C' or 'c' will allow you to change the file name before saving.

```

Quit _____ Exit editor.                                original version
4 _____ move cursor left one character.                J
6 _____ move cursor right one character.                 L
H-19 8 _____ move cursor up one line.                    I
      2 _____ move cursor down one line.                 ;
      5 _____ alternate between left and right of line.  K
      0 _____ move cursor 1/2 screen up                  O (letter)
      . _____ move cursor 1/3 screen down
space _____ move cursor to next identifier.
BT or cntrl T _____ move cursor to next tab.
cntrl R _____ move cursor backwards to previous tab.
cntrl S _____ single character delete. deleting the carriage
                                     return will merge lines, with multiple leading
                                     spaces compressed into one space.
                                     original version
cntrl X _____ delete current line.
9 _____ Overwrite one character.                          I (digit one)
3 _____ overwrite characters until escape.                V
H-19 1 _____ enter append mode.                            A
      7 _____ enter insert mode.                           ;

```

```
Find _____ will position cursor, response is
CR = stop command
Space = continue
M = multiple
      occurrences on line

Replace _____ will position cursor, response is
CR = stop command
Space = continue
R = Replace and continue
M = multiple
      occurrences on line
E = entire buffer
```

Insert or append node

```

BS or DEL _____ backspaces over character.
ET or cntrl T _____ forward tab.
cntrl R _____ reverse tab.
cntrl A _____ copies remainder of last line entered.
cntrl X _____ removes all characters entered so far.

```

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
[ROM-] Monitorprogrammet SBUG-E	18
Adresslista/or	

Elektronik

MINI-BOTATORTNIK

Werte 10.000 bis 100.000,00

1	10.000	10.000	10.000
2	20.000	20.000	20.000
3	30.000	30.000	30.000
4	40.000	40.000	40.000
5	50.000	50.000	50.000
6	60.000	60.000	60.000
7	70.000	70.000	70.000
8	80.000	80.000	80.000
9	90.000	90.000	90.000
10	100.000	100.000	100.000
11	110.000	110.000	110.000
12	120.000	120.000	120.000
13	130.000	130.000	130.000
14	140.000	140.000	140.000
15	150.000	150.000	150.000
16	160.000	160.000	160.000
17	170.000	170.000	170.000

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^x beskrivs de förändringar som uppkommer när systemet skall användas för utveckling av 6809-program.

^x som aldrig blev av.

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)

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

[⊗] g. av oss.

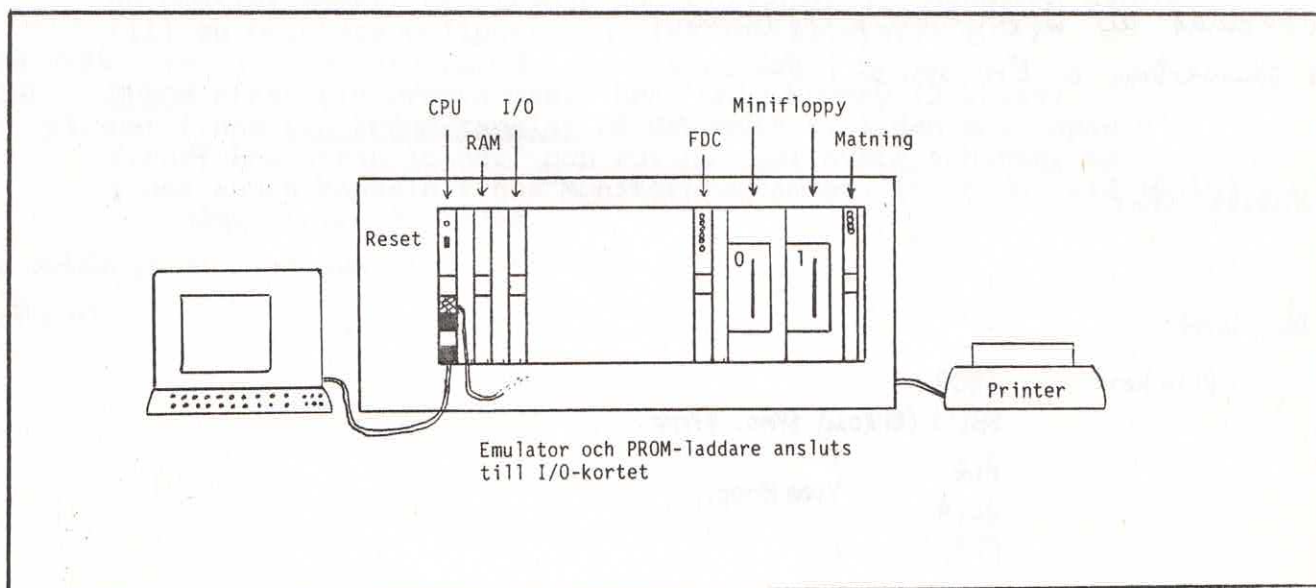


Fig 1. Utvecklingssystem USE 6800 i grundversion

LISE 6809

I/O-kort

FLEX Operativsystem
(ersatte smån av OS/09)

SE fullSkärmsEditor

ASMB
ö.sätter mmem. + op.kop

SBLG (i Monitorn)

DRBLG
går djupare ner i progr.

LISE 6809

utvecklat av Ulf Brydsten & Ulf Nordin
i samverkan c Ericsson i Östersund

Diskette - stur

Din kort:

CPU-kort	6809
	SBLG (EPROM) \$F000..\$FFFF
	PIA
	ACIA
	PTM *

} \$0000..\$0000..

PIA 1

PIA 2

PIA 3

ACIA 1

ACIA 2

PTM bärn räknare 3 disponibel
(2 + 2 2 av 5 bärn-
rate-generator + ACIA.na)Låg förr på \$D02x
Nu flyttat + \$E02x..
f a g krocka c RAM-adr.na

Diskett - stur

#0 avsedd f systemdisk.

#1 " " jobb

Systemet u. håller bl.a

80 spår	(96 tpi)
40 spår	(48 tpi)

80 spårsformatting ger	2844 tillgängl. sektorer
40 " " "	1404 " "

Båda: 18 sektorer/spår

36 sektorer används alltid av systemet.
(se s 17-8)

* ①
② bärn-rate generator (gentemot dominalen)
③ 83 tpi f används när indvikl gä - 2
progr. an

17.2.1 CPU-kort (ROF 137 3204)

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

- M6809 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 ^①generera 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 ^②fungera 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 ^③generera 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 ^①"Bootstrap-loader" och rutiner för avbrottshantering. I den andra kapseln finns ^②Monitorprogrammet. (Se EBUG, sid 17-10)

Minneskarta

	SBLG	\$FFFF..
\$E020 \$E000	I/Okort System I/O	\$F000
	FLEX	\$E000 \$DFFF \$C000
	RAM	\$0000

+++ NEWDISK 2,1

Dis frågor; ge svar

DOUBLE SIDE DISK

40 TRACKS

DOUBLE DENSITY DISK

VOLUME NAME? WLF

VOLUME NUMBER? 1

256 by/sekter

Nu tiden är:

Alla adr. + FLEX ändrat

f \$Axxx .. \$Cxxx

+ \$Bxxx .. \$Dxxx

+++

Kommando
s ligger på
systemdisken

skilje-
tecken

gäller
av. disken

tex

+++ LIST, WLF.TXT

Listar filen WLF.TXT på/r
disken 1

+++ LIST, 0-FLEXLIB.TXT

Listar filen ...
på/r disken 0.

Demo

W RESET

hanterar man i BFFF

där SBLG:n residerar

F a komma i + FLEX (OS)

Stoppa systemdisk i #0

Tryck L

> L

Bootar Systemet

Laddar in ett Boot-progr
fr spår 0

Da progr laddar sen
FLEX

+++ DATE

(MM, DD, YY)?

+++ = FLEX' prompt.

Allt i sälls + stora bokst.

Esc fogglar stopp på serolln.

Ret avbryter

F a skr i +

+++ SE

Skr i ett memoriserat progr.

F a assemblera

+++ ASMB, ADD, +N

skr ut allt (=kl op koder) på skärmen

+++ PO, ASMB, ADD, +N

skr ut på skrivaren

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

\$E000 - \$EOFF	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 (6809-uppläggen anslutna till sid 17-9)

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 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

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

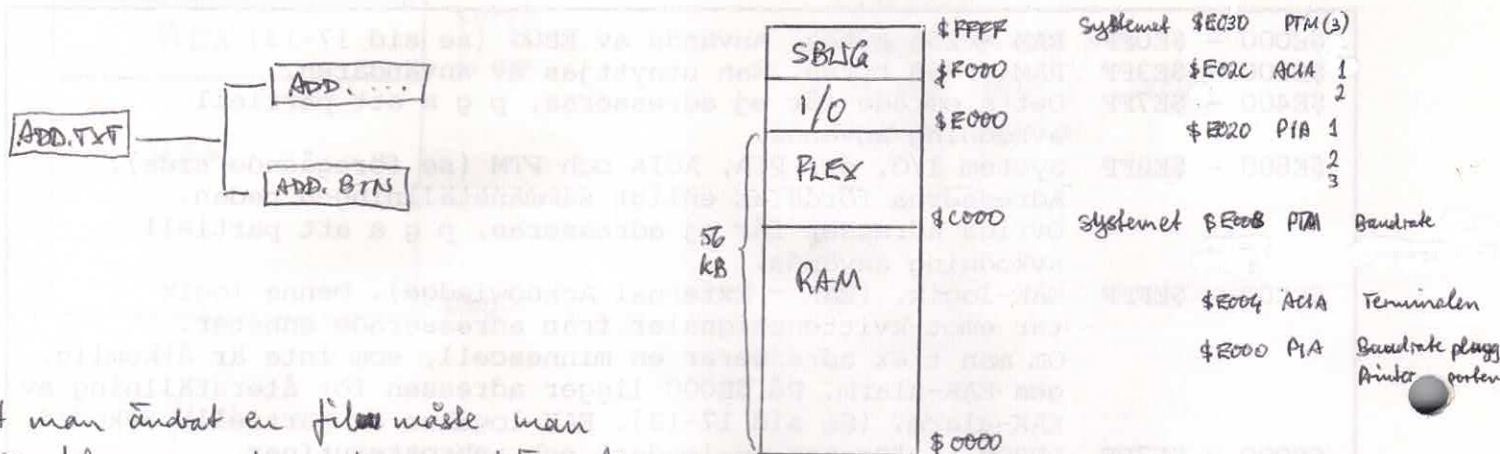
Fig 3. Adresser för CPU-kortets PIA, ACIA och PTM

1. ADD. BIN

871201 hi 2

LSR 6809

Minneskarta



A man ändrat en film måste man
assemblera om f a kunna köra den
nya versionen.

	A	B	C		A B C		
02	PB7	CB2	PB3	<u>Printer</u>	o o o	grön	
04	PB6	P.u.	PB2	(parallell-	o o o		lyser när
06	PB5	GND	PB1	snitt)	o o o		systemet
08	PB4	CB1	PB0		o o o	RESET	är i gång
10	PA7	CA2	PA3	<u>BAUD-rate-</u>	o o o		
12	PA6	P.u.	PA2	<u>plugg</u> för	o o o		
14	PA5	GND	PA1	bildskärms-	o o o		
16	PA4	CA1	PA0	terminal	o o o		
18	*	*	*		o o o		
20	*	DTR	CALL.IND		o o o		
22	*	*	*		o o o		
24	*	*	*		o o o		
26	RXD	*	RTS	<u>Bildskärms-</u>	o o o		
28	GND	CTS	DTR	<u>terminal</u>	o o o		
30	TXD	*	DCD		o o o		
32	GND	*	*		o o o		
* = ej ansluten					nedre	övre	
GND = 0 V					halva	halva	
P.u. = Pull up, 1 kohm till +5 V					Frontkontakt		

Fig 4. Signaler och frontkontakt på CPU-kortet17.2.2 RAM-kort (ROF 137 3201)

Byggt: 6809-versionen.

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)
 varav \$0000 - \$DFFF (8 kb) reserverad för FLEX
 kvar: \$0000 - \$BFFF (48 kb) användarfrött

871210 to 17

Flödeschema

PAU Problemlösungsstips

Da Hälfte 12

871211 für 8

Flödeschema

Progr. u. v.

Segel. u. v.

MT
Lösblad

Tentamen

10 880116 Skr. sal 2 9-15

684 uppgifter

Utrecklingsystem

Assembler programming 6809

Skriver progr mot hårdvara
tex enkertsdatorer!

Kommunik mit FLEX

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

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

Fig 6 I/O-kontakte signaler og kontakt

P.N. = Pull up, 1 kohm till +5 V			
GND = 0 V			
* = ej ansluttet			
Front-kontakt			
A B C			
02	PB7	CA2	PB3
04	PB6	P.U.	PB2
06	PB5	GND	PB1
08	PB4	CA1	PB0
10	PA7	CA2	PA3
12	PA6	P.U.	PA2
14	PA5	GND	PA1
16	PA4	CA1	PA0
18	PB7	CA2	PB3
20	PB6	P.U.	PB2
22	PB5	GND	PB1
24	PB4	CA1	PB0
26	PA7	CA2	PA3
28	PA6	P.U.	PA2
30	PA5	GND	PA1
32	PA4	CA1	PA0
02	PB7	CA2	PB3
04	PB6	P.U.	PB2
06	PB5	GND	PB1
08	PB4	CA1	PB0
10	PA7	CA2	PA3
12	PA6	P.U.	PA2
14	PA5	GND	PA1
16	PA4	CA1	PA0
18	PB7	*	PB3
20	GND	PB2	
22	PB1	*	
24	GND	*	
26	PB0	PB7	
28	GND	PB6	
30	PB5	PB5	
32	GND	PB4	
02	PB7	CA2	PB3
04	PB6	P.U.	PB2
06	PB5	GND	PB1
08	PB4	CA1	PB0
10	PA7	CA2	PA3
12	PA6	P.U.	PA2
14	PA5	GND	PA1
16	PA4	CA1	PA0
18	PB7	*	PB3
20	GND	PB2	
22	PB1	*	
24	GND	*	
26	PB0	PB7	
28	GND	PB6	
30	PB5	PB5	
32	GND	PB4	
02	PB7	CA2	PB3
04	PB6	P.U.	PB2
06	PB5	GND	PB1
08	PB4	CA1	PB0
10	PA7	CA2	PA3
12	PA6	P.U.	PA2
14	PA5	GND	PA1
16	PA4	CA1	PA0
18	PB7	*	PB3
20	GND	PB2	
22	PB1	*	
24	GND	*	
26	PB0	PB7	
28	GND	PB6	
30	PB5	PB5	
32	GND	PB4	

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.

röd	o	Drive 0
röd	o	Drive 1
röd	o	Drive 2 (option)
röd	o	Drive 3 (option)
grön	o	Sida 2
grön	o	Dubbel densitet
röd	o	Spår 0
röd	o	Motor på

\$D014	Drive Select Register. Skriv och läsbart.
\$D018 - \$D01B	Floppy Disk Controller (se tabell)

	$\overline{WE} = 0$	$\overline{RE} = 0$
\$D018	Write Command Register	Read Status Register
\$D019	Write Track Register	Read Track Register
\$D01A	Write Sector Register	Read Sector Register
\$D01B	Write Data Register	Read Data Register

Fig 7. FDC-kortets lysdioder och adresser

6809-versionen har adresser \$Exxx

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.

grön	o	-12 V
gul	o	+12 V Diskdrive
gul	o	+12 V
röd	o	+5 V

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.

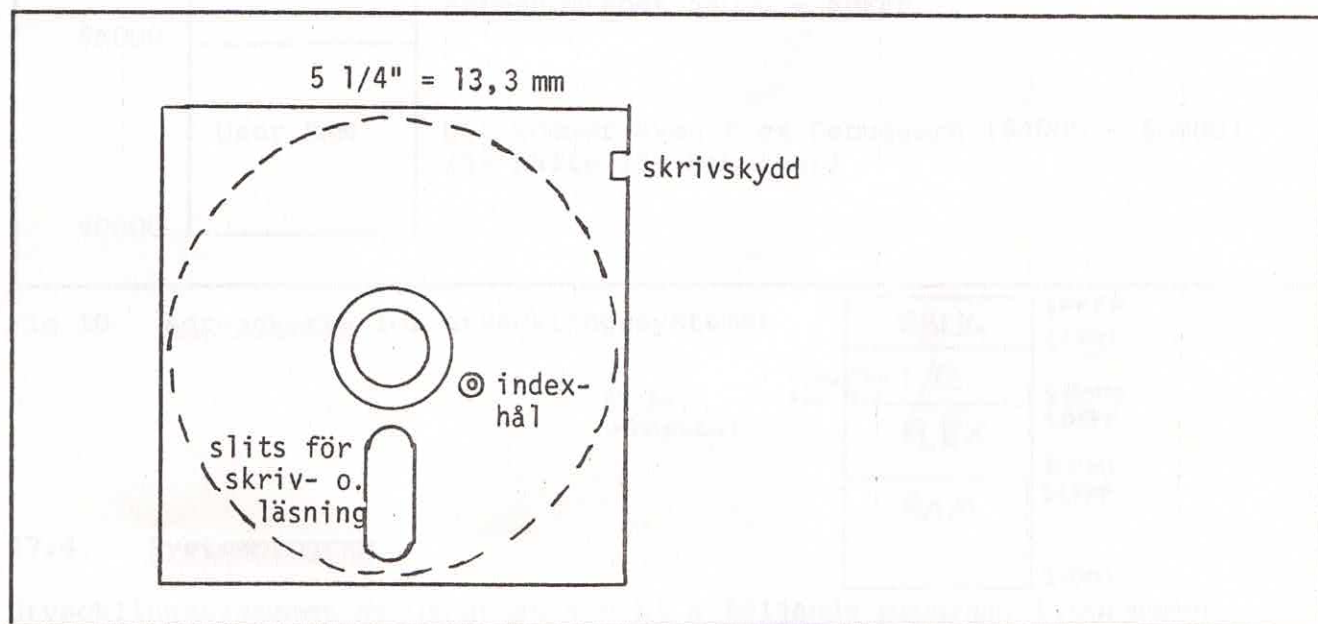


Fig 9 Flexskiva

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 $80 \cdot 10 \cdot 256 \text{ bytes} = \underline{200 \text{ kb}}$.

^{dvs 18 sektorer/sida (w DB)}
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

Dubbelsid. Dubbel densitet:	80 spår	⇒	2844 t.gängl sektorer	+	^{2·18=} 36 systemsektor	=	2880 sektorer totalt.
	40 " "		1404 " "		" "	=	1440 " "

Dubbel densitet ⇒ 18 sektorer/spår.

17.3 Adresskarta

Bygets: 6809-versionen.

Systemet har ⁵⁶52 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
\$D000	User I/O	(Se sid 17-5)
\$C000	User RAM	
\$A000	RAM/FLEX	FLEX 2.0 operativsystem läggs i adressområdet \$A000 - \$BFFF
\$0000	User RAM	Hit kommer även t ex Debuggern (\$4000 - \$6000) (Se häfte 13: Debugger)

Fig 10 Adresskarta för utvecklingssystemet

6809-
versionen:

SBLG	\$FFFF
I/O	\$F000
FLEX	\$E000
	\$DFFF
	\$C000
	\$BFFF
RAM	\$0000

17.4 Systemprogram

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

- 2.8:4
- | | |
|--|---------------|
| 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)

qL Se EBUG ist.

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 RAMFormat: **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 a1 a2** **n** = antal bytes i bitmönstret
 m = det sökta bitmönstret (1-3 bytes = 2-6 hextecken)
 a1 = 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 kodenFormat: **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 a1 a2** **a1** = startadress för kontrollsummeringen
 a2 = stoppadress för kontrollsummeringen

K-kommandot summerar minnesinnehållet från adress **a1** till 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.

M Manipulation av innehåll

Form: M a - skattemässiga

M-kommandor skriver ut innehåll i angiven adress.

Ex. *M 0190 7E (7E = innehåll i adressen 00190)

Efter det att innehåll har skrivits ut, kan man välja mellan några olika alternativ enligt punkterna nedan. Vi nämner i varje alternativ ett exempel ovan.

1. Vi vill lista på nästa adress.

Tryck på tangenten LINE NEXT
Innehåll i nästa adress (00190) lämnas oförändrat och
nästa adress (00191) och dess innehåll skrivs ut.

2. Vi vill lista på föregående adress.

Tryck på tangenten SHIFT 5 (PIL UPP).
Innehåll i nästa adress (00190) lämnas oförändrat och
föregående adress (00189) och dess innehåll skrivs ut.

3. Vi vill ändra innehåll i angiven adress.

Skriv det nya värdet efter det gamla.

Ex. *M 0190 7E 45

00190 har nu innehåll 45. Endast hexadecimala tecken är tillåtna. Siffror som inte är hexadecimala tecken, skrivs kommandor.
(Se punkt 5)

4. Vi vill veta offset från angiven adress (00190) till

en annan adress (00230).

Tryck på tangenten O (offset) av en adress (innehåll)
adress: Minneshållare (00190) skrivs ut.

Ex. *M 0190 7E 00230 7E

5. Vi vill skriva M-kommandor.

Siffror som inte är hexadecimala tecken, skrivs mellanrädd.

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 a1 a2 a1 = 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 franslag 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 uppåt 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
 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 släppta
- 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 ^{CD03 (6809)} 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 släppta.
Öppna dörrarna till respektive drive.
- ③ Slå av strömmen till utvecklingssystemet
- ④ Slå av strömmen till terminal och printer.

* I 6809-versionen går SBUS in i skrotar i översta regionen av FLEX' RAM-are → man bör alltid kallstarta fr SBUS, f a återställa FLEX.

1000000

Man kann sich die Lage so vorstellen, dass die beiden Seiten sich in einem

Verständnis der Lage und der Lage der Lage

1000000

Man kann sich die Lage so vorstellen, dass die beiden Seiten sich in einem

Verständnis der Lage und der Lage der Lage

Man kann sich die Lage so vorstellen, dass die beiden Seiten sich in einem

Verständnis der Lage und der Lage der Lage

1000000

Man kann sich die Lage so vorstellen, dass die beiden Seiten sich in einem

1000000

Man kann sich die Lage so vorstellen, dass die beiden Seiten sich in einem

Verständnis der Lage und der Lage der Lage

Man kann sich die Lage so vorstellen, dass die beiden Seiten sich in einem

Verständnis der Lage und der Lage der Lage

Man kann sich die Lage so vorstellen, dass die beiden Seiten sich in einem

Verständnis der Lage und der Lage der Lage

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 sk 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**.

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

Byte <u>0-1</u>	<u>Spår- och sektorsnummer till nästa sektor i kedjan av fria sektorer.</u>
Byte <u>2-3</u>	Här skall det <u>logiska ordningsnummret</u> på sektorn lagras, när sektorn ingår i en fil.
Byte <u>4-255</u>	<u>Datasektorer</u> . 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 formaterad 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=*			Fel s ASMB uppmärksammar (s17-18c)
13.00=SUM	RMB	1	SE SID 12-16
14.00=*			
15.00=START	LDAA	#\$15	
16.00=	LDAB	#\$39	
17.00=	ABA		PSHS B
18.00=	DAA		ADDA ,S+
19.00=	STAA	SUM	
20.00=	JSR	PCRLF*	NY RAD *Rubin för <CR> + Line Feed
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 #.

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 funktioner.

Uppgift: Skriv ett program som skriver ut namnet på skolorna i en lista som finns på skolorna.

Vi kallar vårt program för "skola". Programmet skall skrivas i BASIC (se häfte 1) och om vi vill köra det i BASIC 2.0, kan vi använda BASIC 2.0.

17.8.1.1. Skriv programmet

I och med detta, har vi skapat en textfil med namnet "skola.TXT". Den fil ligger på skolorna i en katalog som heter "skola". (Se skola.TXT för information om hur vi gör det i BASIC).

Vi skriver vårt program i BASIC och använder oss av (se häfte 1). Varje rad avslutas med ett uttryck på skolorna. De som skrivs ut och upprepas flera gånger kan vi skriva på skolorna. Den som är kort en eller flera gånger kan vi skriva på skolorna. Den som är kort en eller flera gånger kan vi skriva på skolorna. Den som är kort en eller flera gånger kan vi skriva på skolorna. (Se sid 11-12)

17.8.1.2. Köra programmet

Rad	Skola	Skola	Skola
1	1.00	1.00	1.00
2	2.00	2.00	2.00
3	3.00	3.00	3.00
4	4.00	4.00	4.00
5	5.00	5.00	5.00
6	6.00	6.00	6.00
7	7.00	7.00	7.00
8	8.00	8.00	8.00
9	9.00	9.00	9.00
10	10.00	10.00	10.00
11	11.00	11.00	11.00
12	12.00	12.00	12.00
13	13.00	13.00	13.00
14	14.00	14.00	14.00
15	15.00	15.00	15.00
16	16.00	16.00	16.00
17	17.00	17.00	17.00
18	18.00	18.00	18.00
19	19.00	19.00	19.00
20	20.00	20.00	20.00
21	21.00	21.00	21.00
22	22.00	22.00	22.00
23	23.00	23.00	23.00
24	24.00	24.00	24.00
25	25.00	25.00	25.00
26	26.00	26.00	26.00

Det programmet är skrivet i BASIC och körs i BASIC 2.0. Programmet avslutar sig med att skriva ut namnet på skolorna i en lista som finns på skolorna. (Se sid 11-12)

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 SKÄRMEN
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 (Editorbufferten) 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 (P = 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 (Programmet ADD.BIN förs från skiva 1 till
 54 RAM och exekveras)
 +++P,ASMB,ADD,+NB (Assemblerlista med radnummer (N), skrivs ut
 +++ på printern. Ingen ny binärfil skapas (B))

Generalization of the program for assembly of
program 13-17

NAME 1.0
*
FILE 2.0

DATE (MM/DD/YY) 01.12.82
(name: 13-17)

++EDIT,ADD
(text: 13-17) (name: 13-17)

NEW FILE
1.00=

24.00=

24.00=

++ASMB,ADD,IN
(text: 13-17) (name: 13-17)

(Assembly: 13-17)

++EDIT,ADD
(text: 13-17) (name: 13-17)

11.00-FILE
*AD03

11.00-FILE
*AD03

11.00-FILE
*AD03

++ASMB,ADD,IN
(text: 13-17) (name: 13-17)

(Assembly: 13-17)

++1,ADD,IN
24
++1,ADD,ADD,IN
24

6809 SBUG-E[®] Monitor ROM Version 1.5

MAKAN JOELSON
871130

1871207

The 6809 SBUG monitor ROM is provided to enable the computer to communicate with a terminal for the purpose of various programming and debugging functions. It has been designed to work in a SWTPC MP-09 processor board. It will not work in other processor boards. SBUG requires an MP-S interface installed in I/O port 1, at least 4K of RAM memory installed at any address at or below address D000[®] (52K) and an MP-B or MP-B2 motherboard patched to address I/O devices at 56K (E000 hex). Refer to the MP-09 instruction set for changes that must be made to several computer system boards for proper system operation. While SBUG provides some functional compatibility with 6800 DISKBUG[®], SWTBUG[®], and MIKBUG[®] monitors; in general, programs utilizing the ROM functions of the older monitors will need to be changed. Note that while the 6800 is a predecessor to the 6809, 6800 object code will not execute properly on a 6809. A cross-assembler is available to aid in processor transition.

If the full 56K memory capability is not desired, it is possible to modify the SBUG-E[®] Version 1.5 code to allow the MP-09 processor board to run in an unmodified 6800 mainframe. With the modified code, the system memory capacity is limited to 40K bytes, and the system clock frequency is limited to 1 MHz. The SBUG monitor is supplied in a 16K masked ROM that is pinout compatible with the INTEL 5V-only 2716 EPROM. This compatibility provides the user the capability of duplicating the SBUG monitor in EPROM with custom modifications to meet his particular needs. A detailed explanation of how to make this change is provided later in this document.

SBUG-E[®] Monitor Commands

When the SBUG-E[®] monitor completes power up (or reset) processing, it will display its identification and enter the command mode. In this mode, it prompts the user for monitor commands with a right arrow character, ">". Commands consist of one character and may be followed by one or two arguments. All arguments are entered in hexadecimal, with the number of digits corresponding to the precision required of the argument. For example, a 16-bit argument requires four hex digits. Commands requiring arguments may be aborted by typing a carriage return character instead of a hex digit. A list of available commands follows:

Alter A-accumulator — control "A"

The alter accumulator commands display the current contents of the specified register and then allow the user to alter its value by typing in a new value in hex. Should the user not wish to alter the register value, he can enter a carriage return at the terminal device and the SBUG-E[®] monitor will retain the old register value. The alter A-accumulator command takes the following form:

```
> ^A      A=43 6F                                —change accumulator from 43 to 6F
>
```

Alter B-accumulator — control "B"

The alter B-accumulator command takes the following form:

```
> ^B      B=7F 2A                                —change accumulator from 7F to 2A
>
```

Alter Condition Codes — control "C"

The alter condition code command first displays the current condition flags in binary, as described under Display Registers. The monitor will then allow the contents of the condition flags to be changed by typing in a new value in hex. Note that the "E" flag cannot be cleared. The alter condition code command takes the following form:

```
> ^C      CC: E F — I — — — D4                    —set the zero condition
> ^C      CC: E F — I — Z — — [cr]                 —no change
>
```

⊗ ⇒ krock med FLEX, es FLEX läggs i 3000..3DFF

Az: när SBUG är använt, måste FLEX laddas på nytt (sk principiellt räkna a ladda den

Alter Direct Page Register — control "D"

The alter direct page register command takes the following format:

> ^D DP=00 01 —alter direct page register to \$01
>

Alter Program Counter — control "P"

The alter program counter command has the following format:

> ^P PC=4655 434B —change program counter to \$434B
>

Alter User Stack Pointer — control "U"

The alter user stack command has the following format:

> ^U US=5348 4954 —change user stack to \$4954
>

Alter X Index Register — control "X"

The alter X-register command has the following format:

> ^X IX=4355 4E54 —change X-register to \$4E54
>

Alter Y Index Register — control "Y"

The alter Y-register command has the following format:

> ^Y IY=434F 434B —change Y-register to \$434B
>

Set Breakpoint — "B" [addr]

The breakpoint command causes the SBUG—E^o monitor to save the byte at the specified address in its internal breakpoint table and replace it with a software interrupt instruction. Whenever the monitor is entered via a software interrupt instruction, the internal breakpoint table is searched for a matching program counter entry. If this entry is found, the saved byte replaces the software interrupt instruction, the entry is removed from the breakpoint table, and the program counter is adjusted to point back at the restored byte. The user can continue execution of the interrupted program via the Go monitor command. § -3-

Up to five breakpoints may be active at one time. If the monitor finds that there is no space in the breakpoint table, it will output a question mark after the address and the breakpoint will not be set. Breakpoints may be explicitly cleared via the "X" command. § -5-

Disk Bootstrap — "D"

The disk boot command invokes the SBUG—E^o bootstrap program for the DMAF1 or DMAF2 8-inch floppy disks. The bootstrap program issues a restore to drive zero, then attempts to read sector one of track zero into memory at address \$0000 (logical). If the read indicates a CRC failure, the bootstrap program will continue retrying the read until the sector is read properly or the reset switch is pressed. When the sector is successfully loaded, control is passed to the Flex internal bootstrap program in the loaded sector. Note that if there is no disk in the drive, or if no disk interface is attached, the bootstrap program will hang until the reset button is pressed.

Since the ROM-based disk read function does not contain the extensive error detection and retry functions found in the disk operating system, it is possible to have a disk boot fail on an undetected soft read error. If this is the case, the disk drive will deselect without booting the operating system. If this occurs, the reset button should be pressed and the bootstrap repeated. This type of soft failure should be very rare—if it should occur with any regularity, it indicates a problem with either the disk media, disk drive or controller.

Examine Memory – "E [addr] – [addr]"

The examine memory command displays the contents of memory between the first address specification and the second address specification. The memory is displayed in both hexadecimal and ASCII, and is always displayed in multiples of 16 bytes. The format of the display is shown below:

```
>E 579B-57A2
- 5790  00 00 22 53 ... 38 30 39 20  .. "SWTPC S/6809
- 57A0  20 69 73 20 ... 21 22 0D 0A  i s t h e b e s t !".
>
```

Since the display of large blocks of memory can take considerable time at low baud rates, the display may be terminated at the end of a line by typing the carriage return at the terminal device.

Continue Execution – "G"

The Go command causes the SBUG-E[©] monitor to resume processing of an interrupted program. The interrupt may have been caused by a software interrupt or breakpoint. In the case of a reset interrupt, the Go command simply re-enters the monitor code and requests another command. If the monitor was entered via a software interrupt, execution resumes at the first instruction after the software interrupt. In the case of a breakpoint, which is a special case of software interrupt, the monitor removes the breakpoint and resets the program counter. Execution will resume at the instruction at the breakpoint address. If the user wishes to change the address at which execution will resume, the program counter can be changed prior to issuing the Go command.

Load MIKBUG[®] Format Tape – "L"

The tape load command causes the SBUG-E[©] monitor to load a MIKBUG[®] formatted paper tape (or cassette) into memory. At the beginning of the load, the monitor sends a reader-on control character (\$11) to the terminal device. The tape is then read and any object record blocks loaded into memory. The loading process stops when either an end of tape indicator is read, or a bad checksum is detected on a load block. For information on block formats, see the Punch command.

Memory Alter – "M [addr]"

The memory alter command allows the user to examine and alter the contents of (logical) memory on a byte-by-byte basis. When the memory alter command is invoked, the specified address is made the "pointed" address. For each byte, the SBUG-E[©] monitor displays the "pointed" address, and contents of memory at that address, and then allows the user to alter that byte, and/or move forward or backward through memory. An example is provided as explanation:

```
>M 93F3
- 93F3  46 .           - period moves forward in memory
- 93F4  55 ,           - so does a comma
- 93F5  43             - in fact, most characters move forward
- 93F6  4B 48          - typing a hex value alters the byte
- 93F7  49 ^           - typing an up-arrow goes backward
- 93F6  48 [cr]        - typing carriage return ends the alter
>
```

If a new value is specified for a memory alter, and for some reason or other the monitor cannot write the new value into memory (write into ROM, protected memory, etc.), the alter function displays a question mark after the new value, like this – F833 3C 2A ? – and continues.

Punch MIKBUG[®] Tape — "P [addr] — [addr]"

The "P" command causes the SBUG—E[®] monitor to output the specified range of memory addresses as a MIKBUG[®] formatted ASCII tape. The tape is punched in 32-byte blocks starting at the first specified address and continuing through the second specified address, inclusive. Each block consists of an object record header, a block count field, a block address field, the data block itself, and a checksum. The checksum is calculated by taking the complement of the sum of the byte count, the two bytes of address, and all of the data bytes output in the block. The punch routine outputs a punch-on control character (\$12) at the beginning of the punch operation and outputs a punch-off control character (\$14) at the end of the punch operation.

Memory Test — "Q [addr] — [addr]"

The "Q" command causes the SBUG—E[®] monitor to perform an address variable memory test starting at the first address specified and continuing through the second specified address, inclusive. The memory test will display a plus sign "+" after completing each pass through memory, and will perform 65,535 passes over the address range before terminating. Since this large number of passes can take a substantial period of time (just under 90 hours for 56K of memory), the test can be interrupted at the end of each pass by typing a carriage return at the terminal.

The memory test is performed in two parts. The first part computes a test pattern for each byte of memory by adding together the MSP and LSP of the memory address and the MSP and LSP of the pass number. This test pattern is stored into the memory and subsequently checked by the second part of the memory test. This test will detect address convergence errors, row or column decode errors, and bit write or read failures with high reliability.

If the memory test detects an error, the following message is displayed on the terminal device:

```
— ERROR AT XXXX=> YYYYYY, PASS ZZZZ, BITS IN ERROR BBBBBBBB
```

The logical address of the failing memory is specified by XXXX while the actual physical address is specified as YYYYYY. The pass number is shown (in hex) as ZZZZ. The bits-in-error field is displayed as a series of eight characters, each specifying a bit in the memory word. Bits that were read (or written) incorrectly are specified by their bit number, valid bits are shown as hyphens. If the memory problem is due to slow memories, the bits-in-error field may be shown as all hyphens.

For example, a user having 20K of memory in his 6809 system suspects that he has a problem with memory. He would then run the monitor memory diagnostic and receives the following indications:

```
>Q C000-DF7F ++++++++
— ERROR AT D13F => 0713F, PASS 000C, BITS IN ERROR --5-----
— ERROR AT D17F => 0717F, PASS 000C, BITS IN ERROR --5-----
— ERROR AT D1BF => 071BF, PASS 000C, BITS IN ERROR --5-----
— ERROR AT D1FF => 071FF, PASS 000C, BITS IN ERROR --5-----
```

This display shows that the memory test detected a memory malfunction at logical address D13F hex (in the system area) on pass 12. The physical address of this memory board is 0713F hex. The high order digit of the address is zero, since extended addressing is not being used in this system, and the particular bank of memory chips responds at 7000 hex. In the case of an MP-8M memory board, this is the upper row of memory chips. Since in each case, bit 5 is the bit found to be in error, the third memory chip from the left in the top row should be suspected.

Since the SBUG—E[®] monitor assigns its stack storage starting at \$DFBF, it is suggested that the uppermost address specified in the memory test command be lower than \$DF80. If the memory test overruns its own stack, it will appear to continue operation but the results of the test are no longer valid.

Display Registers — "R"

The display register command causes the SBUG-E^c monitor to display the contents of all accessible processor registers. Most register values are displayed in hexadecimal, however, the condition code register is shown in an expanded binary format. If a flag bit in the condition code register is set the corresponding flag name is shown, otherwise the flag position is denoted with a hyphen. The format of the display is as follows:

- SP=hhhh US=hhhh DP=hh IX=hhhh IY=hhhh
- PC=hhhh A=hh B=hh CC: E F H I N Z V C

The condition flag letters are "E" — for "entire state" (always set), "F" for "FIRQ masked", "H" for "half-carry", "I" for "IRQ masked", "N" for "negative", "Z" for "zero" (or equal), "V" for "two's compliment overflow", and "C" for "carry".

Display Stack — "S"

The display stack command causes the SBUG-E^c monitor to enter the memory examine function using the current stack pointer as a lower limit and using its default internal stack pointer (\$DFC0) as an upper limit. This results in the stack contents being displayed, if the stack space assigned by the SBUG-E^c monitor is utilized. If the stack pointer has been changed via a load stack instruction, the operation of this command may be unpredictable. The format of the stack display is the same as that of memory examine.

Boot MF-68 Minifloppy — "U"

The "U" command invokes the SBUG-E^c bootstrap program for the MF-68 5-inch floppy disk. The disk interface should be installed in processor interface slot 6 with appropriate modifications to MP-B or MP-B2 motherboards. The bootstrap program issues a restore command to drive zero, then attempts to read sector one on track zero into memory at SC000 (logical). If the sector is read successfully, control is transferred to the disk boot sector. Note that if there is no disk in the drive, or no interface attached, the disk boot command will hang until the reset switch is pressed.

Remove Outstanding Breakpoints — "X"

The "X" command causes the SBUG-E^c monitor to remove any breakpoints that may have been set in memory. Note that a reset (or NMI) interrupt implicitly calls the "X" command, clearing any left-over breakpoints.

s-2-

The "U" command invokes the 5B10-E² monitor program for the MPE-3200 floppy disk. The disk interface would be installed in processor interface slot 2 with appropriate modifications to the 5B10-E² monitor. The monitor program uses a store command to drive test data into the 5B10-E² monitor. If the stack contains data, the stack space is used. If there is no data in the stack, the data is taken from the 5B10-E² monitor. The data is then transferred to the 5B10-E² monitor. If the stack contains data, the data is taken from the 5B10-E² monitor. The data is then transferred to the 5B10-E² monitor. If the stack contains data, the data is taken from the 5B10-E² monitor. The data is then transferred to the 5B10-E² monitor.

The condition flag bits are "E" for "error", "N" for "negative", "V" for "overflow", "H" for "half carry", "C" for "carry", and "Z" for "zero". The condition flag bits are "E" for "error", "N" for "negative", "V" for "overflow", "H" for "half carry", "C" for "carry", and "Z" for "zero". The condition flag bits are "E" for "error", "N" for "negative", "V" for "overflow", "H" for "half carry", "C" for "carry", and "Z" for "zero".

Display Stack - 2

The display stack command uses the 5B10-E² monitor to enter the monitor stack function using the current stack pointer as a lower limit and using the default stack pointer (5B10-E²) as an upper limit. The results in the stack contents being displayed. If the stack space contains data, the data is taken from the 5B10-E² monitor. If the stack contains data, the data is taken from the 5B10-E² monitor. The data is then transferred to the 5B10-E² monitor. If the stack contains data, the data is taken from the 5B10-E² monitor. The data is then transferred to the 5B10-E² monitor.

Boot MPE-3200 Floppy - 1

The "U" command invokes the 5B10-E² monitor program for the MPE-3200 floppy disk. The disk interface would be installed in processor interface slot 2 with appropriate modifications to the 5B10-E² monitor. The monitor program uses a store command to drive test data into the 5B10-E² monitor. If the stack contains data, the stack space is used. If there is no data in the stack, the data is taken from the 5B10-E² monitor. The data is then transferred to the 5B10-E² monitor. If the stack contains data, the data is taken from the 5B10-E² monitor. The data is then transferred to the 5B10-E² monitor.

Remove Outstanding Breakpoints - 1

The "X" command causes the 5B10-E² monitor to remove any breakpoints that may have been set in memory. Note that a test for (X) interrupt pin-hole is the "X" command, clearing any leftover breakpoints.

MOTION

NEXTING

INCH

The SBUG-E[®] monitor resides in the uppermost 2K of memory space in the system (from F800 hex to FFFF hex) and allocates RAM memory from DFC0 hex to DFFF for internal use as vectors, tables, and pointers. A set of routine addresses is provided at address F800 hex for use by user programs. These routines may be called via Indirect Jump to Subroutine instruction. A group of vectors is provided in RAM in locations DFC0 to DFCF hex to provide for user control of interrupt processing. A list of significant monitor addresses follows:

Addr	Name	Description of Address
SF800	MONITOR	Re-enter the monitor and initialize memory.
SF802	NEXTCMD	Re-enter monitor and prompt for command. → \$F861
SF804	INCH	Get input character from terminal device.
SF806	INCHE	Get input character from terminal and echo.
SF808	INCHEK	Check for input character.
SF80A	OUTCH	Output character to terminal device.
SF80C	PDATA	Mikbug [®] compatible print data string.
SF80E	PCRLF	Print carriage return, line feed, nulls.
SF810	PSTRNG	Call PCRLF, then print data string.
SF812	LRA	Load real address of memory byte.
SD8C0	USER-V	User Interrupt Vector
SD8C2	SW13-V	Software Interrupt III Vector
SD8C4	SW12-V	Software Interrupt II Vector.
SD8C6	FIRQ-V	Fast Maskable Interrupt Vector.
SD8C8	IRQ-V	Slow Maskable Interrupt Vector.
SDFCA	SWI-V	Software Interrupt Vector
SDFCC	SVC-VO	Supervisor Call Vector Origin Address.
SDFCE	SVC-VL	Supervisor Call Vector Limit Address.

Note that these vector entries are addresses, and jumping directly into the address vector is invalid and will produce unpredictable results.

User-callable ROM Routines

The following paragraphs provide entry parameterization and functional descriptions of the provided user-callable ROM routines in the SBUG-E[®] monitor. It is inadvisable to call directly into the ROM routines as opposed to using the entry vector since future releases of the SBUG ROM may have different internal addresses while the entry vector will remain valid. The proper method for calling a routine in SBUG is to use an indirect subroutine call, e.g.

INCHE	EQU	\$F806	INPUT ROUTINE ADDRESS
	JSR	[INCHE]	CALL INPUT CHARACTER ROUTINE
	STA	,X+	STORE INTO BUFFER, etc.

MONITOR — SF800

Re-enter the SBUG-E[®] monitor. Re-establish the RAM interrupt vectors and initialize the serial interface to the terminal device. Establish a dummy stack, compute memory size, and display the signon message, then enter NEXTCMD.

NEXTCMD — SF802

Enter the SBUG-E[®] monitor at the point where it prompts for a command. This entry point does not initialize the stack, which implies that the SBUG monitor can be called as a subroutine.

INCH — SF804

Get a character from the terminal device. The character is returned in the A register with all other registers preserved. The input character is 8-bits long, and is not echoed to the terminal device.

Chief

Address	Function
0000	Reset
0001	Power On Reset
0002	Watchdog Timer
0003	Interrupt Vector
0004	Interrupt Vector
0005	Interrupt Vector
0006	Interrupt Vector
0007	Interrupt Vector
0008	Interrupt Vector
0009	Interrupt Vector
000A	Interrupt Vector
000B	Interrupt Vector
000C	Interrupt Vector
000D	Interrupt Vector
000E	Interrupt Vector
000F	Interrupt Vector
0010	Interrupt Vector
0011	Interrupt Vector
0012	Interrupt Vector
0013	Interrupt Vector
0014	Interrupt Vector
0015	Interrupt Vector
0016	Interrupt Vector
0017	Interrupt Vector
0018	Interrupt Vector
0019	Interrupt Vector
001A	Interrupt Vector
001B	Interrupt Vector
001C	Interrupt Vector
001D	Interrupt Vector
001E	Interrupt Vector
001F	Interrupt Vector
0020	Interrupt Vector
0021	Interrupt Vector
0022	Interrupt Vector
0023	Interrupt Vector
0024	Interrupt Vector
0025	Interrupt Vector
0026	Interrupt Vector
0027	Interrupt Vector
0028	Interrupt Vector
0029	Interrupt Vector
002A	Interrupt Vector
002B	Interrupt Vector
002C	Interrupt Vector
002D	Interrupt Vector
002E	Interrupt Vector
002F	Interrupt Vector
0030	Interrupt Vector
0031	Interrupt Vector
0032	Interrupt Vector
0033	Interrupt Vector
0034	Interrupt Vector
0035	Interrupt Vector
0036	Interrupt Vector
0037	Interrupt Vector
0038	Interrupt Vector
0039	Interrupt Vector
003A	Interrupt Vector
003B	Interrupt Vector
003C	Interrupt Vector
003D	Interrupt Vector
003E	Interrupt Vector
003F	Interrupt Vector
0040	Interrupt Vector
0041	Interrupt Vector
0042	Interrupt Vector
0043	Interrupt Vector
0044	Interrupt Vector
0045	Interrupt Vector
0046	Interrupt Vector
0047	Interrupt Vector
0048	Interrupt Vector
0049	Interrupt Vector
004A	Interrupt Vector
004B	Interrupt Vector
004C	Interrupt Vector
004D	Interrupt Vector
004E	Interrupt Vector
004F	Interrupt Vector
0050	Interrupt Vector
0051	Interrupt Vector
0052	Interrupt Vector
0053	Interrupt Vector
0054	Interrupt Vector
0055	Interrupt Vector
0056	Interrupt Vector
0057	Interrupt Vector
0058	Interrupt Vector
0059	Interrupt Vector
005A	Interrupt Vector
005B	Interrupt Vector
005C	Interrupt Vector
005D	Interrupt Vector
005E	Interrupt Vector
005F	Interrupt Vector
0060	Interrupt Vector
0061	Interrupt Vector
0062	Interrupt Vector
0063	Interrupt Vector
0064	Interrupt Vector
0065	Interrupt Vector
0066	Interrupt Vector
0067	Interrupt Vector
0068	Interrupt Vector
0069	Interrupt Vector
006A	Interrupt Vector
006B	Interrupt Vector
006C	Interrupt Vector
006D	Interrupt Vector
006E	Interrupt Vector
006F	Interrupt Vector
0070	Interrupt Vector
0071	Interrupt Vector
0072	Interrupt Vector
0073	Interrupt Vector
0074	Interrupt Vector
0075	Interrupt Vector
0076	Interrupt Vector
0077	Interrupt Vector
0078	Interrupt Vector
0079	Interrupt Vector
007A	Interrupt Vector
007B	Interrupt Vector
007C	Interrupt Vector
007D	Interrupt Vector
007E	Interrupt Vector
007F	Interrupt Vector
0080	Interrupt Vector
0081	Interrupt Vector
0082	Interrupt Vector
0083	Interrupt Vector
0084	Interrupt Vector
0085	Interrupt Vector
0086	Interrupt Vector
0087	Interrupt Vector
0088	Interrupt Vector
0089	Interrupt Vector
008A	Interrupt Vector
008B	Interrupt Vector
008C	Interrupt Vector
008D	Interrupt Vector
008E	Interrupt Vector
008F	Interrupt Vector
0090	Interrupt Vector
0091	Interrupt Vector
0092	Interrupt Vector
0093	Interrupt Vector
0094	Interrupt Vector
0095	Interrupt Vector
0096	Interrupt Vector
0097	Interrupt Vector
0098	Interrupt Vector
0099	Interrupt Vector
009A	Interrupt Vector
009B	Interrupt Vector
009C	Interrupt Vector
009D	Interrupt Vector
009E	Interrupt Vector
009F	Interrupt Vector
00A0	Interrupt Vector
00A1	Interrupt Vector
00A2	Interrupt Vector
00A3	Interrupt Vector
00A4	Interrupt Vector
00A5	Interrupt Vector
00A6	Interrupt Vector
00A7	Interrupt Vector
00A8	Interrupt Vector
00A9	Interrupt Vector
00AA	Interrupt Vector
00AB	Interrupt Vector
00AC	Interrupt Vector
00AD	Interrupt Vector
00AE	Interrupt Vector
00AF	Interrupt Vector
00B0	Interrupt Vector
00B1	Interrupt Vector
00B2	Interrupt Vector
00B3	Interrupt Vector
00B4	Interrupt Vector
00B5	Interrupt Vector
00B6	Interrupt Vector
00B7	Interrupt Vector
00B8	Interrupt Vector
00B9	Interrupt Vector
00BA	Interrupt Vector
00BB	Interrupt Vector
00BC	Interrupt Vector
00BD	Interrupt Vector
00BE	Interrupt Vector
00BF	Interrupt Vector
00C0	Interrupt Vector
00C1	Interrupt Vector
00C2	Interrupt Vector
00C3	Interrupt Vector
00C4	Interrupt Vector
00C5	Interrupt Vector
00C6	Interrupt Vector
00C7	Interrupt Vector
00C8	Interrupt Vector
00C9	Interrupt Vector
00CA	Interrupt Vector
00CB	Interrupt Vector
00CC	Interrupt Vector
00CD	Interrupt Vector
00CE	Interrupt Vector
00CF	Interrupt Vector
00D0	Interrupt Vector
00D1	Interrupt Vector
00D2	Interrupt Vector
00D3	Interrupt Vector
00D4	Interrupt Vector
00D5	Interrupt Vector
00D6	Interrupt Vector
00D7	Interrupt Vector
00D8	Interrupt Vector
00D9	Interrupt Vector
00DA	Interrupt Vector
00DB	Interrupt Vector
00DC	Interrupt Vector
00DD	Interrupt Vector
00DE	Interrupt Vector
00DF	Interrupt Vector
00E0	Interrupt Vector
00E1	Interrupt Vector
00E2	Interrupt Vector
00E3	Interrupt Vector
00E4	Interrupt Vector
00E5	Interrupt Vector
00E6	Interrupt Vector
00E7	Interrupt Vector
00E8	Interrupt Vector
00E9	Interrupt Vector
00EA	Interrupt Vector
00EB	Interrupt Vector
00EC	Interrupt Vector
00ED	Interrupt Vector
00EE	Interrupt Vector
00EF	Interrupt Vector
00F0	Interrupt Vector
00F1	Interrupt Vector
00F2	Interrupt Vector
00F3	Interrupt Vector
00F4	Interrupt Vector
00F5	Interrupt Vector
00F6	Interrupt Vector
00F7	Interrupt Vector
00F8	Interrupt Vector
00F9	Interrupt Vector
00FA	Interrupt Vector
00FB	Interrupt Vector
00FC	Interrupt Vector
00FD	Interrupt Vector
00FE	Interrupt Vector
00FF	Interrupt Vector

Note that these vector entries will all return and jumping directly into the interrupt vector is not valid and will produce unpredictable results.

Use-callable ROM Routines

The following paragraphs provide only a generalization and not a detailed description of the operations of the ROM routines in the 2800-EP processor. This information is provided for the user's reference only. The ROM routines are designed to be used in a variety of applications and the user should consult the 2800-EP processor data sheet for more information.

ROUTINE ADDRESS	ROUTINE ADDRESS
CALL INPUT CHARACTER ROUTINE	0000
STORE INTO BUFFER	0001

For the 2800-EP processor, the ROM routines are located in the 2800-EP processor memory and are accessible to the user. The user should consult the 2800-EP processor data sheet for more information.

For the 2800-EP processor, the ROM routines are located in the 2800-EP processor memory and are accessible to the user. The user should consult the 2800-EP processor data sheet for more information.

For the 2800-EP processor, the ROM routines are located in the 2800-EP processor memory and are accessible to the user. The user should consult the 2800-EP processor data sheet for more information.

INCHE — \$F806

Get a character from the terminal device and echo. The character is returned in the A register with all other registers preserved. The input is masked to 7-bits in length, and then echoed back to the terminal device.

INCHEK — \$F808

Check for a character available from the terminal device. The serial interface is checked to see if it has become read ready. All registers are preserved, and the routine will return zero clear if a character can be read from the terminal device.

OUTCH — \$F80A

Output a character to the terminal device. The character to be output is passed in the A register, and return is made with all registers preserved.

PDATA — \$F80C

Output a MIKBUG[®] compatible string to the terminal device. The routine is entered with the IX register pointing at the first character of the string to be output. The string is terminated with a byte having the value 04 hex. Return is made with the A register containing the 04, the IX register pointing at the byte after the 04, and all other registers preserved.

PCRLF — \$F80E

This routine outputs a carriage return, a line feed, and three null characters to the terminal device. The A register returns with a 04, all other registers are preserved.

PSTRNG — \$F810

This routine is entered with the IX register containing the address of a string to be output. PCRLF is called to obtain a new line, then PDATA to print the string. The A register contains a 04 on exit, the IX register points to the byte after the 04, and all other registers are preserved.

LRA — \$F812

Load the 20-bit physical address of a memory byte into the A and X registers. This routine is entered with the logical address of a memory byte in the IX register. Exit is made with the high-order four bits of the 20-bit physical address in the A register, and the low-order 16-bits of the 20-bit physical address in the IX register. All other registers are preserved. This routine is required since the DMAF1 and DMAF2 disk controllers must present physical addresses on the system bus.

User-referable RAM Locations

The SBUG monitor reserves memory from DFC0 hex to DFFF for use as internal tables and addresses. Included in this area are the interrupt handler addresses for the maskable and software interrupts. These interrupts are available to the user by placing the address of an interrupt service routine in the appropriate vector address. Note that RESET and NMI are not vectored through RAM and hence are not available to the user. The interrupt vector addresses and explanations follow:

Software Interrupt Vector — \$DFCA

This vector contains the address of the software interrupt service routine entered whenever an SWI instruction is executed. After power up, this vector points to SBUG breakpoint processing, however, if breakpoints are not being used, this vector may be altered by the user.

Maskable Interrupt Vector — \$DFC8

This vector contains the address of the IRQ interrupt service routine. After power up processing, this vector points at a return from interrupt instruction.

When a write operation is performed, the data is written to the memory location specified by the address. The data is then read back from the memory location and compared to the data that was written. If the data is the same, the write operation is successful. If the data is different, the write operation is failed.

When a read operation is performed, the data is read from the memory location specified by the address. The data is then compared to the data that was written. If the data is the same, the read operation is successful. If the data is different, the read operation is failed.

When a write operation is performed, the data is written to the memory location specified by the address. The data is then read back from the memory location and compared to the data that was written. If the data is the same, the write operation is successful. If the data is different, the write operation is failed.

When a read operation is performed, the data is read from the memory location specified by the address. The data is then compared to the data that was written. If the data is the same, the read operation is successful. If the data is different, the read operation is failed.

When a write operation is performed, the data is written to the memory location specified by the address. The data is then read back from the memory location and compared to the data that was written. If the data is the same, the write operation is successful. If the data is different, the write operation is failed.

When a read operation is performed, the data is read from the memory location specified by the address. The data is then compared to the data that was written. If the data is the same, the read operation is successful. If the data is different, the read operation is failed.

When a write operation is performed, the data is written to the memory location specified by the address. The data is then read back from the memory location and compared to the data that was written. If the data is the same, the write operation is successful. If the data is different, the write operation is failed.

When a read operation is performed, the data is read from the memory location specified by the address. The data is then compared to the data that was written. If the data is the same, the read operation is successful. If the data is different, the read operation is failed.

When a write operation is performed, the data is written to the memory location specified by the address. The data is then read back from the memory location and compared to the data that was written. If the data is the same, the write operation is successful. If the data is different, the write operation is failed.

When a read operation is performed, the data is read from the memory location specified by the address. The data is then compared to the data that was written. If the data is the same, the read operation is successful. If the data is different, the read operation is failed.

Fast Maskable Interrupt Vector — \$DFC6

This vector contains the address of the FIRQ interrupt service routine. After power up processing, this vector points at a return from interrupt instruction.

Software Interrupt II Vector — \$DFC4

This vector contains the address of the secondary software interrupt service routine entered whenever an SWI2 instruction is executed. After power up, this vector points to a return from interrupt instruction. The secondary software interrupt is not used on a system level to make it available at a user level.

Software Interrupt III Vector — \$DFC2

This vector contains the address of the tertiary software interrupt service routine entered whenever an SWI3 instruction is executed and there is no supervisor call vector specified (see below). After power up, this vector points to a return from interrupt instruction.

Supervisor Call Vector Origin — \$DFCC, and Supervisor Call Vector Limit — \$DFCE

The supervisor call (SVC) vectors are used in conjunction with the SWI3 instruction to provide a second level decode for interrupt processing. When an SWI3 interrupt occurs, the SBUG monitor checks the SVC Vector Origin for an address of \$FFFF. If this value is contained in the vector, SWI3 processing defaults to the SWI3 vector (see above), otherwise, the byte of data following the SWI3 instruction is fetched and the program counter adjusted to bypass this byte. The byte is multiplied by two, and used as an index into the SVC vector. The indexed address is compared to the address contained in the SVC Vector Limit, and if it is not greater, an indirect jump is made to the SVC routine. If the address is found to be out of bounds, the processing proceeds through the SWI3 vector.

When the SVC routine is entered, the US register contains the value of the stack pointer on entry to the SVC handler. The stack pointer should be restored to this value before returning from the SVC routine. The supervisor call service routine usually terminates its processing via a TFR U.S: RTI sequence.

Note that an invalid parameter (either vector or SVC number) passed to the SVC service routine will cause the SWI3 vector to be used. In this manner, a supervisor program can detect invalid parameterization. After power up processing, both the origin and limit vectors contain \$FFFF.

Duplicating the SBUG-E[©] Monitor in EPROM

Since the SBUG-E[©] monitor ROM is pinout compatible with 2716 EPROM's, the MP-R PROM programmer can be used to duplicate the SBUG code in PROM. In addition, since the MP-09 board has on-board address translation, it is possible to modify the SBUG-E[©] code to recognize I/O devices addressed at 32K (8000 hex), and hence to run an MP-09 board in a system with an unmodified motherboard. This provides the user with the capability of easily changing from a 6800 system to a 6809 system and back, but costs the user an additional 16K of memory capability, and limits the 6809 system to running at 1MHz, and using the MF-68 minifloppy. If the user wants to use the DMAF1 or DMAF2 disk controller, the controller must be modified to respond at F000 hex (60K) in order to properly run DMAF software.

The SBUG-E[©] monitor ROM should be placed in the MP-R PROM programmer, and read into the program buffer using the MP-R software read command. The entire ROM must be read. The edit command can then be used to change the byte at FF79 (hex) from F1 (hex) to F7 (hex). This changes the way the SBUG monitor sets up the address translation table at power up time to recognize I/O at 8000 (hex). A blank 2716 is inserted into the PROM programmer and a new PROM burned, which will then replace the SBUG ROM in the MP-09 board.

The first section contains the code for the SWI3 instruction. It is located at address 00000000. The code is as follows:

SWI3 instruction (SWI3)

This section contains the code for the SWI3 instruction. It is located at address 00000000. The code is as follows:

SWI3 instruction (SWI3)

The second section contains the code for the SWI3 instruction. It is located at address 00000000. The code is as follows:

SWI3 instruction (SWI3)

The third section contains the code for the SWI3 instruction. It is located at address 00000000. The code is as follows:

SWI3 instruction (SWI3)

The fourth section contains the code for the SWI3 instruction. It is located at address 00000000. The code is as follows:

SWI3 instruction (SWI3)

The fifth section contains the code for the SWI3 instruction. It is located at address 00000000. The code is as follows:

ADRESSER TILL VEKTORER I MONITORPROGRAM

se SBLG-E-manual
2-6-

\$DFC0	USER VEKTOR
\$DFC2	SWI3 VEKTOR
\$DFC4	SWI2 VEKTOR
\$DFC6	FIRQ VEKTOR
\$DFC8	IRQ VEKTOR
\$DFCA	SWI VEKTOR
\$DFCC	SVC ORIGIN
\$DFCE	SVC LIMIT
\$DFD0	IRQ0 RQI-0, PIA B
\$DFD2	IRQ1 PIA A, PIA B
\$DFD4	IRQ2 RQI-1
\$DFD6	IRQ3 ACIA
\$DFD8	IRQ4 RQI-2
\$DFDA	IRQ5 PTM
\$DFDC	IRQ6 RQI-3
\$DFDE	IRQ7 EAK-ALARM
\$DFE0	ADRESS TERMINAL ACIA
\$DFE2	EKO-FLAGGA
\$DFE3	START AV BREAKPOINTTABELL (DFE3-DFFA)
\$DFFB	LAGRA MINNESMANGD
\$FFEE	RESET AV EAK-ALARM

BORTMASKNING AV AVBROTT GENOM SKRIVNING PÅ FÖLJANDE ADRESSER

\$FFE0	INGET
\$FFE1	IRQ0
\$FFE2	IRQ0-IRQ1
\$FFE3	IRQ0-IRQ2
\$FFE4	IRQ0-IRQ3
\$FFE5	IRQ0-IRQ4
\$FFE6	IRQ0-IRQ5
\$FFE7	IRQ0-IRQ6
\$FFE8	IRQ0-IRQ7
\$FFE9	IRQ0-IRQ

*** ADRESSER TILL IO-KRETSAR PÅ CPU-KORTET

\$E000-\$E003	PIA
\$E004-\$E005	ACIA
\$E008-\$E00F	PTM

*** ADRESSER TILL IO-KRETSAR PÅ I/O-KORTET

\$E020-\$E023	PIA 1
\$E024-\$E027	PIA 2
\$E028-\$E02B	PIA 3
\$E02C-\$E02D	ACIA 1
\$E02E-\$E02F	ACIA 2
\$E030-\$E037	PTM 3

command	cmd	page
ASN[,w=<drive>I,S=<drive>]	ASN	10-9
CAT[,<drive list>I,<match list>]	CAT	10-11
COPY,<filespec>,<file spec>	COPY	10-13
COPY,<file spec>,<drive>		
COPY,<drive>,<drive>,<match list>]		
DATE[,mm,dd,yy]	DATE	10-15
DELETE,<file spec>,<file list>]	DELETE	10-16
GET,<file spec>,<file list>]	GET	10-7
default extension : .BIN		
LIST,<file spec>,<line range>I,N]	LIST	10-17
MON	MON	10-7
O,<file spec>,<command>	O	10-18
default extension : .OUT		
P,<command>	P	10-19
PROT,<file spec>,<options>]	PROT	10-20
RENAME,<file spec 1>,<file spec 2>	RENAME	10-21
default extension : .TXT		
SAVE,<file spec>,<begin adr>,<end adr>,<transfer adr>]	SAVE	10-22
default extension : .BIN		
SAVE.LOW	SAVE.LOW	10-22
STARTUP	STARTUP	10-23
TTYSETI,<parameter list>]	TTYSET	10-25
VERSION,<file spec>	VERSION	10-29
default extension : .CMD		
GETCHR Get character	\$AD15	10-33
PUTCHR Put character	\$AD18	10-33
PSTRNG Print string	\$AD1E	10-33
INBUFF Input to line buffer	\$AD1B	10-34
PCRLF Print CR + LF	\$AD24	10-34
OUTADR Output hexadecimal address	\$AD45	10-34
OUTDEC Output decimal number	\$AD39	10-35
OUTHEX Output hexadecimal number	\$AD3C	10-35
GETHEX Get hexadecimal number	\$AD42	10-35
INDEC Input decimal number	\$AD48	10-35
Line buffer	\$A080 - \$A0FF	10-32
System date registers	\$AC0E - \$AC10	10-32
Line buffer pointer	\$AC14 - \$AC15	10-32
Escape return register	\$AC16 - \$AC17	10-32

Omegasoft Screen Editor

SCREEN EDITOR COMMANDS

Line syntax : " " = current, "g" = last, digits = offset or relative line number, # and - allowed.

Buffered Commands ('E' prompt)

Edit <filename> _____ Clear buffer and read file.
 New <filename> _____ Clear buffer and set filename.
 Set <filename> _____ Get file past cursor.
 Save <filename> _____ Save entire file.
 Save Partial <lines>,<lines> _____ Save entire file and quit.
 Move <lines>,<lines> _____ Save part of file.
 Copy <lines>,<lines> _____ will move lines to after cursor position.
 Delete <lines>,<lines> _____ will delete lines.
 Position <lines> _____ positions cursor to line.
 Tabs Repetitive <incr> _____ sets tabs at columns 0, incr, 2*incr, 3*incr, etc.
 Tabs Non-repetitive (<column>) _____ sets tab at entered column.
 Tabs _____ Clear all tabs.
 T+ _____ Enable display of tab markers.
 T- _____ Disable display of tab markers.
 T _____ Display filename, and line and memory status. Enter any key to exit.

When loading a new file or exiting the editor you will be prompted to save the current file if it has been modified since last saved. Any response other than "N" will save the file in the old buffer. A response of 'C' or 'c' will allow you to change the file name before saving.

Non-buffered commands (see page 10-9)

Quit _____ Exit editor.
 9 _____ move cursor left one character.
 0 _____ move cursor right one character.
 8 _____ move cursor up one line.
 7 _____ move cursor down one line.
 5 _____ alternate between left and right of line.
 0 _____ move cursor 1/2 screen up.
 0 _____ move cursor 1/2 screen down.
 space _____ move cursor to next identifier.
 ctrl Y _____ move cursor backwards to previous tab.
 ctrl M _____ single character delete, deleting the carriage return will merge lines, with multiple leading spaces compressed into one space.
 ctrl X _____ original version
 9 _____ overwrite one character.
 0 _____ overwrite characters until escape.
 ctrl Y _____ enter append mode.
 ctrl M _____ enter insert mode.

Text search commands

Find _____ will position cursor, response is
 CR = stop command
 Space = continue
 M = multiple occurrences on line

Replace _____ will position cursor, response is
 CR = stop command
 Space = continue
 M = multiple occurrences on line
 E = entire buffer

Note : When the 'F' prompt appears enter the find pattern - if the pattern is not found the cursor returns then this will terminate the command. When the 'R' prompt appears enter the Replace pattern. A '-' is used as a don't care in the find and Replace patterns. The entire buffer option will enable the replacement throughout the rest of the buffer. The entire line option will enable the replacement on each occurrence of the find pattern on any one line, else only the first match on a line is recognized. The search will start on the current line, continue to the end of the buffer, start back at the top, and stop on the current line. The cursor will be positioned at the last match when the command terminates.

Insert or append mode

Character entry until an escape is entered. Insert mode will insert characters at the current cursor position. If a carriage return is entered during insert mode then append mode will be entered with the last part of the line carried over until append mode is exited. Append mode will start new lines with the current line with no partial line carried over. In addition the following special characters are recognized :

BS or DEL _____ backspaces over character.
 RT or ctrl Y _____ forward tab.
 ctrl X _____ reverse tab.
 ctrl M _____ copies remainder of last line entered.
 ctrl I _____ removes all characters entered so far.

DIRECTIVES

END	end of source program	12-12
EQU	equate a symbol to operand	12-12
FCB	form constant byte	12-13
FDB	form double constant byte	12-14
FCC	form constant characters	12-13
LIB	process library file	12-14
NAM	specify name or title	12-14
ORG	assign origin of program counter	12-15
PAG	skip to next page	12-16
RMB	reserve memory bytes	12-16
SPC	insert spaces of output listing	12-16
TTL	specify name or title	12-14
OPT	assembler control options	12-15
	PAG enable page formatting	

COMMAND

[P,]ASMB,<filename 1>[,<filename 2>[,<optionlist>]

optionlist

B	no binary file
D	no date on list
G	print only one line for FCB, FDB and FCC
L	no list
S	no symbol table
Y	delete binary file

DEBUG COMMAND SUMMARY

General system control

CALC	Calc	13-16
DELAY=<value>	DElay	13-17
DEPTH	DEpTh	13-17
EXIT	Exit	13-17
FLAG[=<address>]	FLag	13-18
IND=ON or OFF	IND	13-18
MACH	MAch	13-18
MODE=1 or 0	MOde	13-18
REG	Reg	13-19
RESET	RESet	13-19
RET	RET	13-19
SET,<register list>	Set	13-20
STACK[,<value>]	STACK	13-20
STATES	STATes	13-20
TRAIL	TRAIL	13-20
X,<FLEX command>	X	13-21

Memory commands

ASM[,<address>]	Asm	13-21
DIS,<start address>,<stop address>	Dis	13-22
DUMP,<address>	DUmp	13-22
FILL,<start address>,<stop address>,<byte>	FILL	13-22
FIND,<start address>,<stop address>,<string>	FIND	13-22
MEM,<address>	Mem	13-23

Simulation control

GO	Go	13-24
JUMP,<address>	Jump	13-24
PAST[,<value>]	PAst	13-24
SIM[,<count>]	SIM	13-24
START,<address>	StArt	13-25
STEP[,<count>]	STEP	13-25
TRACE=<value>	Trace	13-25
TSM[,<count>]	TSM	13-26
'ctrl'C	operator halt	13-26
escape	halt on EOL	13-26
'ctrl'F	faster	13-26
'ctrl'S	slower	13-26

Breakpoints

B,<actions>,<address>,<modifier>,<count>,<I,IF-condition>]	B	13-27
Ba<address>,<modifier>,<count>,<I,IF-condition>]		
R...Print register contents Z...Zero the states counter T...Trace mode on U...Trace mode off (untrace) H...Histogram counter M...Print message J...Jump to new address S...Stop simulation		
Bp[,<address>,<address>]	BP	13-29
CLB[,<address>,<address>]	CLB	13-29
CLH[,<address>,<address>]	CLH	13-29
CLM	CLM	13-29
HIST[,<address>,<address>]	Hist	13-30
RT[,<address>]	RT	13-30

Memory protection

BOUND\$[,<types>]	BOunds	13-31
CLP[,<type>]	CLP	13-31
PROT,<address>,<address>,<type>	Prot	13-32

Execution traps

INST=<count>	INST	13-32
ITRAP=ON or OFF	ITrap	13-33
NEST=<value>	Nest	13-33
STOP=<address>	STOP	13-33
XFR=ON or OFF	XFR	13-33

Interrupt control

IRQ=<count>	IRQ	13-34
NMI=<count>	NMI	13-34
'ctrl'I	'Ctrl'I	13-34
'ctrl'N	'Ctrl'N	13-34

DEBUG COMMAND SUMMARY

General system control

CALC	Calc	13-16
DELAY=<value>	DElay	13-17
DEPTH	DEPth	13-17
EXIT	Exit	13-17
FLAG[=<address>]	FLag	13-18
IND=ON or OFF	IND	13-18
MACH	MAch	13-18
MODE=1 or 0	MOde	13-18
REG	Reg	13-19
RESET	RESet	13-19
RET	RET	13-19
SET,<register list>	Set	13-20
STACK[,<value>]	STACK	13-20
STATES	STATes	13-20
TRAIL	TRAIL	13-20
X,<FLEX command>	X	13-21

Memory commands

ASM[,<address>]	Asm	13-21
DIS,<start address>,<stop address>	Dis	13-22
DUMP,<address>	DUMp	13-22
FILL,<start address>,<stop address>,<byte>]	FILL	13-22
FIND,<start address>,<stop address>,<string>	FIND	13-22
MEM,<address>	Mem	13-23

Simulation control

GO	Go	13-24
JUMP[,<address>]	Jump	13-24
PAST[,<value>]	PAst	13-24
SIM[,<count>]	SIM	13-24
START,<address>	STart	13-25
STEP[,<count>]	STEP	13-25
TRACE=<value>	Trace	13-25
TSM[,<count>]	TSM	13-26
'ctrl'C	operator halt	13-26
escape	halt on EOL	13-26
'ctrl'F	faster	13-26
'ctrl'S	slower	13-26

Breakpoints

B,<actions>B<address>[,<modifier><count>],IF<condition>]	B	13-27
Ba<address>[,<modifier><count>],IF<condition>]		
R...Print register contents		
Z...Zero the states counter		
T...Trace mode on		
U...Trace mode off (untrace)		
H...Histogram counter		
M...Print message		
J...Jump to new address		
S...Stop simulation		
BP[,<address>{-<address>}]	BP	13-29
CLB[,<address>{-<address>}]	CLB	13-29
CLH[,<address>{-<address>}]	CLH	13-29
CLM	CLM	13-29
HIST[,<address>{->address>}]	Hist	13-30
RT[,<address>]	RT	13-30

Memory protection

BOUNDS[,<types>]	BOunds	13-31
CLP[,<type>]	CLP	13-31
PROT,<address>[-<address>],<type>	Prot	13-32

Execution traps

INST=<count>	INST	13-32
ITRAP=ON or OFF	ITrap	13-33
NEST=<value>	Nest	13-33
STOP=<address>	STOP	13-33
XFR=ON or OFF	XFR	13-33

Interrupt control

IRQ=<count>	IRQ	13-34
NMI=<count>	NMI	13-34
'Ctrl'I	'Ctrl'I	13-34
'Ctrl'N	'Ctrl'N	13-34

HÅKAN JOÉLSON
Björnvägen 426
902 60 UMEÅ
Sweden
Tel. 090-13 68 61