SPECTRAVIDEO

MASKINSPRÅKSMANUAL
FÖR SPECTRAVIDEO SV318/SV328
VERSION 2
SKRIVEN FÖR RONEX COMPUTER AB

Av Andrzej Felczak

Viktigt Viktigt Viktigt Viktigt Viktigt Viktigt Viktigt
==========

Viktigt information till köparen av den här manualen.

Den här manualen är avsedd som ett uppslagsverk för den som redan är bekant med maskinspråksprogrammering och är inte tillräckligt pedagogisk och utförlig för att användas som grundkurs i maskinspråksprogrammering.
För den som vill lära sig programmera Z-80 förslår jag därför någon av följande böcker:

"Z80 Assembly language programming" av Lance A. Leventhal
"Programming the Z80" av Rodney Zaks
"the Z-80 microcomputer handbook" av William Barden,Jr

Dessa och flera andra böcker finns hos välsorterade data och bokhandlare.
Manualen tillsammans med någon av de nämnda böckerna och Spectravideos monitor utgör ett kraftfullt paket för maskinspråksprogrammering på Spectravideo.

Denna manual är skriven på en SV328 med Wordstar.

Andrzej Felczak

P.S. Många har frågat om denna manual saknar sidor. Det är inte så utan manualen är sidnumrerad på "Basic-sätt" d.v.s. (nästan) varje kapitel börjar på jämnt tiotal sidnummer.

Viktigt Viktigt Viktigt Viktigt Viktigt Viktigt Viktigt
==========

Innehållsförteckning
====================

Ritningar 1
Beskrivning av Bus-signalerna till expandern 15
I/O karta 20
Körning av maskinspråksprogram ihop med Basic 30
Hopptabell (Kernel) 40
Detaljerad beskrivning av vissa Kernel-rutiner 45
ROM-adresser för kommandon 50
Minneskarta 60
Beskrivning av videoprocessor 70
Exempel på hur BASIC-program ligger i minnet 80
Användning av minnesbankar i Spectravideo 90
Användning av joystick från maskinspråk 110
Bruksanvisning Spectravideo monitor 120
Närmare beskrivning av 80-kolumnskortet 130
Närmare beskrivning av RS-232 interfacet 140

Detaljerad beskrivning av Spectravideos bus-signaler för
expander.

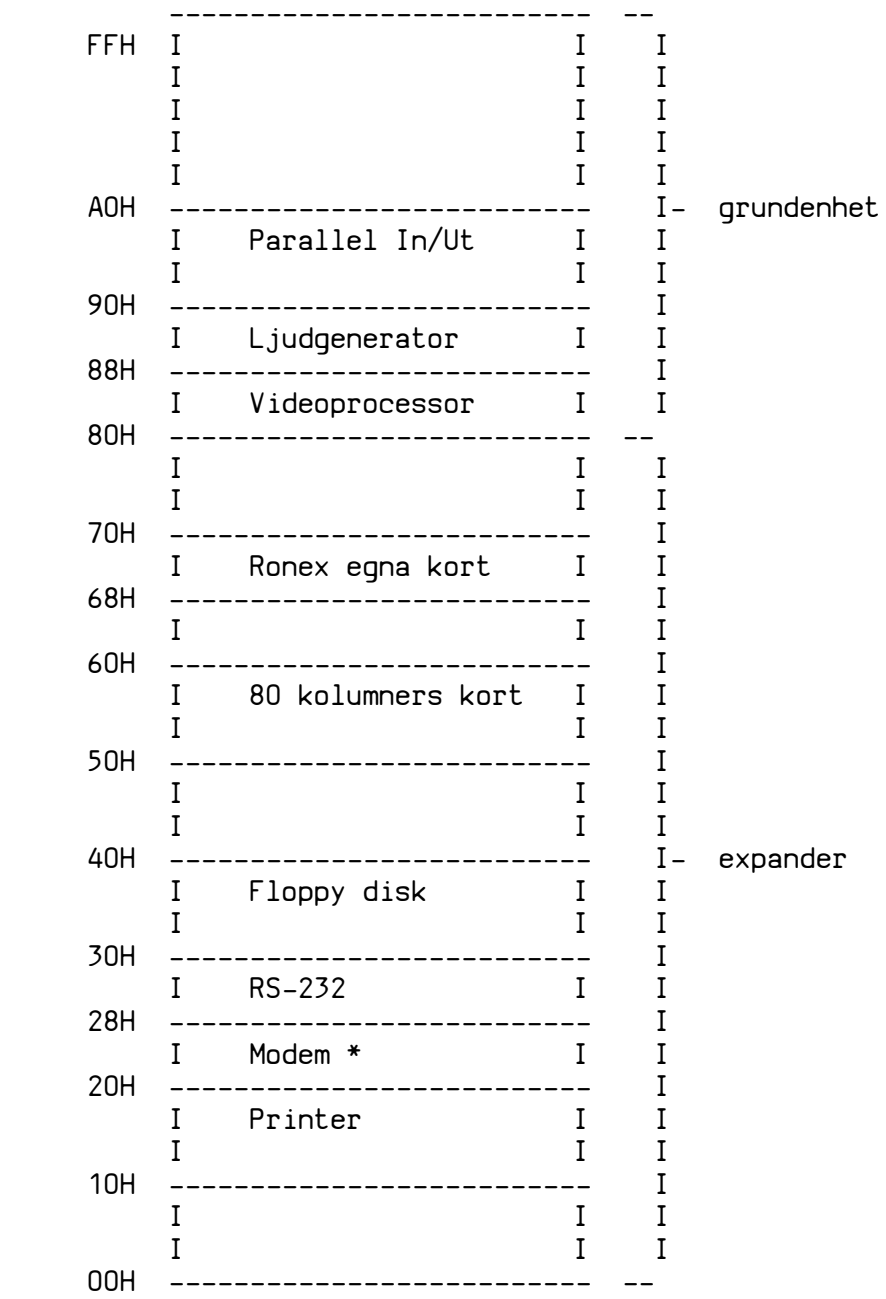
Stift	Namn	I/O	Beskrivning
1	+5V	O	+5V spänningsmatning. Kan belastas med max 300 mA.
2	$\overline{\text{CNTRL2}}$	I	Kontrollsignal för Spectravideo Coleco adapter (normalt hållen hög via ett 3.3 Kohm motsånd till +5V). Denna signal kontrollerar dataöverföringen mellan CPU:n och adaptern under adressering av yttre In/Ut enheter.
3	+12V	O	+12V spänningsmatning. Kan belastas med max 100 mA.
4	-12V	O	-12V spänningsmatning Kan belastas med max 50 mA.
5	$\overline{\text{CNTRL1}}$	I	Kontrollsignal för Spectravideo Coleco adapter (normalt hållen hög via ett 1 Kohm motstånd till +5V). När denna signal dras låg kopplas all intern avkodning av In/Ut enheterna bort och A15 inverteras.
6	$\overline{\text{WAIT}}$	I	När denna signal dras låg indikerar den för Z80 CPU:n att adresserade enheter inte är färdiga för dataöverföring.
7	$\overline{\text{RST}}$	I	När denna signal går låg sätts CPU:ns adress, data och kontrollsignaler i frisvävande läge. När signalen går hög igen startar datorn upp igen som efter spänningstillslag.
8	CPUCLK	O	Buffrad systemklocka med frekvensen 3.58 MHz.
9-24	A15-A0	O	Buffrad adressbus. Den här 16-bitars bussen skickar ut adress för dataöverföring mellan CPU och In/Ut enheter eller minne.
25	$\overline{\text{RFSH}}$	O	Buffrad REFRESH signal för det dynamiska RAM-et i expandern. När denna signal får låg indikerar den att de 8 lägsta bitarna av adressbussen innehåller en refresh-adress.

Stift	Namn	I/O	Beskrivning
26	$\overline{\text{EXCSR}}$	I	Det här är en yttre CPU-från-VDP LÄS signal som används av Spectravideos Colecoadapter.
27	$\overline{\text{M1}}$	O	Buffrad MACHINE CYCLE ONE signal. Denna signal indikerar att CPU:n hämtar en instruktion från minnet.
28	$\overline{\text{EXCSW}}$	I	Det här är en yttre CPU-från-VDP SKRIV signal som används av Spectravideos Colecoadapter.
29	$\overline{\text{WR}}$	O	Buffrad WRITE signal. Indikerar att CPU:ns databus innehåller giltig data för skrivning i det adresserade minnet eller den adresserade Ut-porten.
30	$\overline{\text{MREQ}}$	O	Buffrad MEMORY REQUEST signal. Denna signal indikerar att adressbussen innehåller en giltig minnesadress.
31	$\overline{\text{IORQ}}$	O	Buffrad INPUT/OUTPUT REQUEST. Denna signal indikerar att adressbussen innehåller en giltig In/Ut adress.
32	$\overline{\text{RD}}$	O	Buffrad READ signal. Denna signal indikerar att CPU:n vill läsa data från det adresserade minnet eller den adresserade In-porten.
33-40	D0-D7	I/O	Buffrad dubbelriktad DATA bus. Det här är en 8-bitars dubbelriktad bus för överföring av data mellan CPU och minne eller In/Ut enheter.
41	CSOUND	I	Audioingång från Spectravideo Colecoadapter.
42	$\overline{\text{INT}}$	I	Denna signal (INTERRUPT) indikerar för CPU:n att In/Ut enheter begär avbrott.
43	$\overline{\text{RAMDIS}}$	I	Dras denna signal låg kopplas Spectravideos användar-RAM-minne bort. Denna signal hålls normalt hög via ett 1 Kohm motstånd till +5V.
44	$\overline{\text{ROMDIS}}$	I	Dras denna signal låg kopplas Spectravideos BASIC-ROM-minne bort.

Stift	Namn	I/O	Beskrivning
45	$\overline{\text{BK32}}$	O	Buffrad MEMORY BANK CONTROL signal. Går denna signal låg kopplas minnesbank 32 in och det inbyggda RAM-minnet kopplas bort.
46	$\overline{\text{BK31}}$	O	Buffrad MEMORY BANK CONTROL signal. Går denna signal låg kopplas minnesbank 31 in och det inbyggda BASIC-ROMet kopplas bort.
47	$\overline{\text{BK22}}$	O	Buffrad MEMORY BANK CONTROL signal. Går denna signal låg kopplas minnesbank 22 in och det inbyggda RAM-minnet kopplas bort.
48	$\overline{\text{BK21}}$	O	Buffrad MEMORY BANK CONTROL signal. Går denna signal låg kopplas minnesbank 21 in och det inbyggda BASIC-ROMet kopplas bort.
49-50	GND		Elektrisk jord.

Översiktskarta för In/Ut portarna i SV-318/SV-328 samt expander
==========

Adress (Hex) Enhet



* Modemet finns inte i Sverige eftersom det fungerar enligt amerikansk standard och kan därför inte användas här.

Detaljerad beskrivning av portarna i expandern

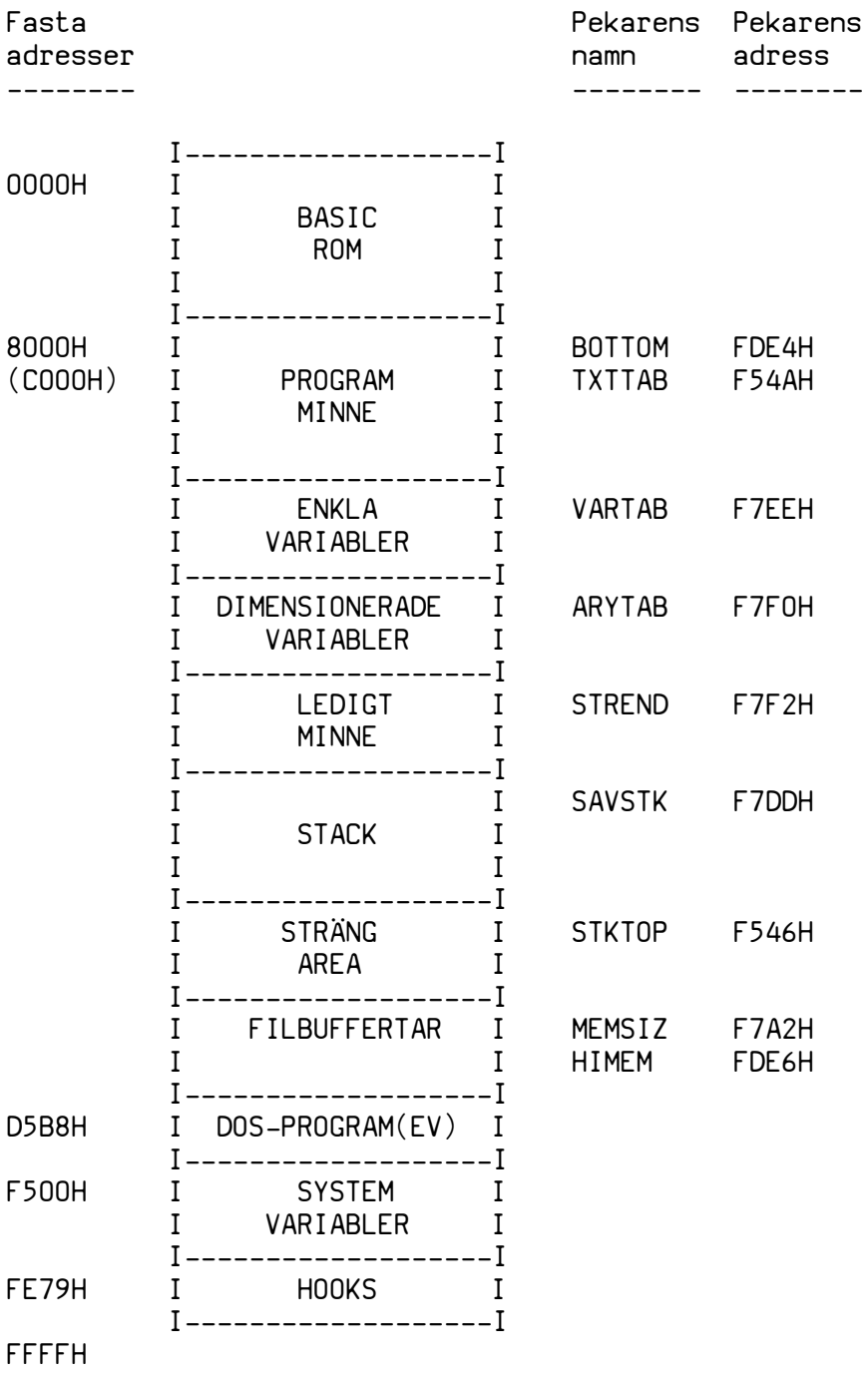
Adress Hex	Dec	R/W	Beskrivning	Enhet
10H	16	W	Write data port	Printer
11H	17	W	Data strobe	Printer
12H	18	R	Status (Bit 0="0" for ready)	Printer
20H	32	R	Receiver buffer register	Modem
		W	Divisor latch (least significant)	Modem
		W	Transmitter holding buffer reg.	Modem
21H	33	W	Divisor latch (most significant)	Modem
		W	Interrupt enable register	Modem
22H	34	W	Interrupt identification register	Modem
23H	35	W	Line control register	Modem
24H	36	W	Read modem control register	Modem
25H	37	R	Line status register	Modem
26H	38	R	Read modem status register	Modem
28H	40	R	Receiver buffer register	RS-232
		W	Divisor latch (least significant)	RS-232
		W	Transmitter holding buffer reg.	RS-232
29H	41	W	Divisor latch (most significant)	RS-232
		W	Interrupt enable register	RS-232
2AH	42	W	Interrupt identification register	RS-232
2BH	43	W	Line control register	RS-232
2CH	44	W	Modem control register	RS-232
2DH	45	R	Line status register	RS-232
2EH	46	R	Modem status register	RS-232
30H	48	R	FD-1793 Status register	Floppy disk
		W	FD-1793 Command register	Floppy disk
31H	49	R/W	FD-1793 Track register	Floppy disk
32H	50	R/W	FD-1793 Secotr register	Floppy disk
33H	51	R/W	FD-1793 Data register	Floppy disk
34H	52	R	Read INTRQ and DRQ from FD-1793	Floppy disk
		W	Disk select register	Floppy disk
			(Bit 0 = "0" to select disk 1	
			Bit 1 = "0" to select disk 2)	
38H	56	W	Density select register	Floppy disk
			(Bit 0 = "0" for double density	
			Bit 0 = "1" for single density)	
50H	80	W	Adress register select	80-column card
51H	81	W	CRT controller register (R0-R17)	80-column card
58H	88	W	CRT bank control	80-column card
			(Bit 0 = "0" for bank off	
			Bit 0 = "1" for bank on)	
68H	104	R/W	Datavision status & programmering	Eget RS-232
69H	105	R/W	Datavision receive/transmit buffer	Eget RS-232
6AH	106	R/W	Datavision status & programmering	Modem
6BH	107	R/W	Datavision receive/transmit buffer	Modem
6CH	108	R/W	PIO-kort lägsta 8 bitarna	PIO-kort
6DH	109	R/W	PIO-kort högsta 8 bitarna	PIO-kort

Detaljerad beskrivning av portarna i expandern

Adress Hex	Dec	R/W	Beskrivning	Enhet
80H	128	W	TMS-9918A Write Mode=0	Video processor
81H	129	W	TMS-9918A Write Mode=1	Video processor
84H	132	R	TMS-9918A Read Mode=0	Video processor
85H	133	R	TMS-9918A Read Mode=1	Video processor
88H	136	W	AY-3-8910 Latch adress	Ljudgenerator
8CH	140	W	AY-3-8910 Write	Ljudgenerator
90H	144	R	AY-3-8910 Read	Ljudgenerator
96H	150	W	Write 8255 port C	Parallel In/Out
97H	151	W	Write 8255 control word register	Parallel In/Out
98H	152	R	Read 8255 port A	Parallel In/Out
99H	153	R	Read 8255 port B	Parallel In/Out

Körning av maskinspråksprogram ihop med Basic

Om du tittar på minneskartan nedanför ser du hur Basic-tolken använder RAM-minnet.
Ska du kunna köra ett maskinspråksprogram ihop med ett Basic-program måste du se till att programmen inte kolliderar med varandra och måste därför reservera plats.
Lyckligtvis är detta relativt lätt gjort genom att ställa om några pekare.



Reservering av plats för egna maskinspråksprogram

Det finns två sätt att reservera utrymme för egen användning.
Antingen kan du höja "golvet" för RAM-minnet eller sänka "taket".
Metoden att höja golvet passar bäst om programmet ska användas på en SV328 med och utan disk medan metoden att sänka taket är bäst om programmet ska passa både SV318 och SV328.
De två följande exemplen visar båda principerna.

Reservering av minnesutrymme genom höjning av "golvet"

De pekare som bestämmer golvet är BOTTOM (adress FDE4H) och TXTTAB (F54AH). Dessa pekare består av två byte så när jag säger att en pekare finns på adress FDE4H menar jag egentligen att den lägre halvan finns på adress FDE4H och den högre halvan finns ett steg högre, alltså på adress FDE5H.
Börja med att titta vilka värden pekarna har från början.
Följande rad skriver ut värdet på pekaren BOTTOM.

```
PRINT HEX$(PEEK(&HFDE4H) + 256*PEEK(&HFDE4+1))
```

Studera raden noga och försök förstå hur den fungerar.
Orsaken till att jag multiplicerar den övre halvan med 256 är att den ska ha rätt "viktning".
BOTTOM har värdet 8000H (C000H). Värdet inom parantes är för SV318.
Skriv därefter ut värdet av TXTTAB. Det ska vara 8001H (C001H).
Titta därefter vad som finns på adress 8000H (C000H) och adress 8001H (C001H).
Följande rad skriver ut innehållet på adress 8000H (C000H).

```
PRINT PEEK(&H8000)    ( PRINT PEEK(&HC000) )
```

Värdet på båda adresserna ska vara 0.

Säg nu att vi behöver reservera 1024 bytes för något ändamål.
Då måste vi flytta båda pekarna 1024 steg upp.
1024 decimalt är 400 hexadecimalt. BOTTOM ska därför få värdet 8400H (C400H) och TXTTAB 8401H (C401H).
Följande rader ändrar pekarnas värden.

```
POKE &HFDE4,&H00            ' lägsta adresshalvan BOTTOM
POKE &HFDE5,&H84   (C4)     ' högsta adresshalvan BOTTOM
POKE &HF54A,&H01            ' lägsta adresshalvan TXTTAB
POKE &HF54B,&H84   (C4)     ' högsta adresshalvan TXTTAB
```

Dessutom måste du nollställa de två byten som dessa pekare pekar på. Det gör du lätt med följande rad.

```
POKE &H8400,0 : POKE &H8401,0 (POKE &HC400,0 : POKE &HC401,0)
```

Till sist får du ge kommandot NEW för att datorn själv ska ändra pekarna VARTAB och ARYTAB.

Nu har du reserverat minnet mellan 8000H-83FFH (C000H-C3FFH)
för eget bruk.

Om du ger följande kommando:

PRINT FRE(0)

Ser du att det finns 1024 bytes mindre tillgängligt än vanligt.

Reservering av minnesutrymme genom sänkning av "taket"

Den pekare som bestämmer taket är HIMEM (FDE6H).
Skriv ut den och kontrollera att den har värdet F500H (D5B8H).
Värdet inom parantes är om du har Disk-Basic.
Säg nu att vi som i förra exemplet ska reservera 1024 bytes.
Då ska taket sänkas från F500H (D5B8H) till
F500H - 400H = F100H (D5B8H - 400H = D1B8H).
Detta görs mycket enkelt med kommandot CLEAR.

```
CLEAR 200,&HF100     ( CLEAR 200,&HD1B8)
```

Detta sänker taket till F100H (D1B8H) och reserverar minnesutrymmet
F100H - F4FF (D1B8H - D5B7H) för egna rutiner.
Ger du nu kommandot:

PRINT FRE(0)

Så ser du på samma sätt att 1024 bytes minne har "försvunnit".

Dessutom sätter CLEAR här storleken på strängarean till 200 bytes.
Eftersom strängarna lagras intill varandra kan du, om du
dimensionerat strängarean till 200 bytes, lagra 10 strängar på
20 bytes eller 20 strängar på 10 bytes eller 100 strängar på 2 bytes
eller varje annan kombination där summan av strängarnas längder
inte överstiger 200 bytes.

Om du överskrider begränsningen får du felmeddelandet OUT OF STRING
SPACE.

Inmatning av maskinspråksprogram i minnet

Vi ska nu mata in nedanstående program i minnet. Studera programmet och jämför koderna med dem som står i någon av de Z-80 läroböcker som du (förhoppningsvis) har köpt.

Adress (Hex)	Kod (Hex)	Mnemonic	Kommentar
C000	C5	PUSH BC	Sparar BC-registerparet
C001	E5	PUSH HL	Sparar HL-registerparet
C002	F5	PUSH AF	Sparar Accumulator-registret
C003	06	LD B,20	Laddar B-registret med 20
C004	14		B används här som loop räknare
C005	3E	LD A,"C"	Laddar A med ASCII-värdet
C006	43		av "C"
C007	F5	PUSH AF	Sparar A:s värde
C008	CD	CALL OUTDO	Mata ut ASCII-koden i A
C009	18		på skärmen
C00A	00		
C00B	21	LD HL,FFFFH	Laddar HL med antalet varv
C00C	FF		
C00D	FF		
C00E	2B	DEC HL	Minskar HL ett steg
C00F	7C	LD A,H	Enda gången (H OR L) är noll
C010	B5	OR L	är när HL är noll
C011	C2	JP NZ,C00EH	Fortsätt om HL<>0
C012	0E		
C013	C0		
C014	F1	POP AF	Återställ A:s värde
C015	3C	INC A	Öka ASCII-koden i A
C016	05	DEC B	Minska loopräknaren
C017	C2	JP NZ,C007H	Fortsätt om inte 20 tkn än
C018	07		
C019	C0		
C01A	F1	POP AF	Återställ A
C01B	E1	POP HL	Återställ HL
C01C	C1	POP BC	Återställ BC
C01D	C9	RET	Återvänd till Basic

Programmet är placerat på adress C000H för att passa både SV318 och SV328 med och utan disk. Detta medför att du inte kan använda allt minne på SV328 men det är väldigt lätt att ändra programmet så att det ligger på adress 8000H. Försök själv.
Glöm inte att ändra hoppadresserna.
Programmet fungerar så att det laddar A-registret med ASCII-värdet för C. Därefter skrivs bokstaven ut.
På adress C00BH är en fördröjning på c:a 1/2 sekund är insatt.
På adress C015H ökas ASCII-koden. Till sist hoppar programmet tillbaka till adress C007 så länge B inte är noll. Programmet kör slingan totalt 20 ggr och skriver ut bokstäverna C till V med paus. Därefter återvänder programmet till Basic:en.

Det finns tre sätt att få in ett maskinspråksprogram i minnet.

1 Genom användning av POKE.
2 Med kommandot BLOAD.
3 Med Spectravideos monitor. Detta sätt beskrivs i bruksanvisningen på sid 110.

Följande Basc-program POKE:ar in det föregående maskinspråks-programmet, ändrar pekarna och raderar sig själv.
Har du en SV318 börjar Basic-programmet på adress C000H och då skriver programmet faktiskt över sig själv. Om du i så fall utesluter fr.o.m. rad 20, d.v.s. delen som ändrar pekarna och försöker lista programmet händer det underliga saker. Det är list-rutinen som enbart är avsedd att lista Basic-program och reagerar konstigt om den stötet på maskinspråk.

```
10 DATA C5,E5,F5,06,14,3E,43,F5
11 DATA CD,18,00,21,FF,FF,2B,7C
12 DATA B5,C2,0E,C0,F1,3C,05,C2
13 DATA 07,C0,F1,E1,C1,C9,*
14 ADR%=&HC000
15   READ A$
16   IF A$="*" GOTO 20
17   POKE ADR%,VAL("&H"+A$)
18   ADR%=ADR%+1
19   GOTO 15
20 POKE &HFDE4,&H20     ' lägsta adresshalvan BOTTOM
20 POKE &HFDE5,&HC0     ' högsta adresshalvan BOTTOM
20 POKE &HF54A,&H21     ' lägsta adresshalvan TXTTAB
20 POKE &HF54B,&HC0     ' högsta adresshalvan TXTTAB
24 POKE &HC020,0        ' nollställ (BOTTOM)
25 POKE &HC021,0        ' nollställ (TXTTAB)
26 NEW
```

Om du jämför Basic-programmets data-satser med de hex-koder som skrevs på sid 33 ser du att dom är identiska.
Du kan nu spara maskinspråksprogrammet genom att ge kommandot

```
BSAVE "1:DEMO",&HC000,&HC01D
```

Nästa gång du vill köra programmet behöver du bara kör rad 20-26 av Basic-programmet som du reserverar den plats som behövs. Själva maskinspråksprogrammet laddar du snabbt in med följande:

```
BLOAD "1:DEMO"
```

Att använda BLOAD är speciellt fördelaktigt att använda om man har väldigt långa program eftersom det går mycket snabbare att ladda in och tar mindre plats på diskett.

Anrop av egna maskinspråksprogram

När du kört det lilla Basic-programmet ligger maskinspråks-programmet dolt i minnet på adress C000H-C01DH.
Du anropar programmet genom sekvensen:

```
DEF USR0=&HC000   ' definiera USR0 att börja på adress C000H
A=USR0(0)         ' kalla på programmet
```

Då ser du att tecknena C till V skrivs ut med en kort paus mellan och därefter visar datorn att den är färdig.
Du har nu kört ditt (kanske) första maskinspråksprogram på Spectravideo. Läs noga igenom texten, jämför med din Z-80 lärobok och försök förstå. Fortsättnigen är svårare och kräver att du verkligen kan och begriper vad som skrivits tills nu.

Överföring av parametrar mellan Basic och maskinspråk

Du kan skicka med parametrar (variabler) från Basic:en till maskinspråksprogrammet och vice versa.
Det medför att du kan skicka med en variabel av valfri typ (heltal, enkel och dubbel precision eller sträng) direkt i anropet av maskinspråksprogrammet. Du kan också få tillbaka en variabel av valfri typ. Se på exemplet nedan.

```
A%=USR0(B%)
```

Där skickar du med variabeln B% och får tillbaka variabeln A%.
Jag ska nu förklara hur maskinspråksprogrammet tolkar de medföljande variablerna.
När Basic:en kallar på maskinspråksprogrammet så ligger det ett värde i A-registret som säger vilken typ av variabel det är. T.ex. betyder värde 2 heltal. Adressen till variabeln ligger dessutom i HL-registret.
Se på sammanfattningen på sid 38 för att få en komplett uppställning hur överföringen går till.
På nästa sida finns ett nytt maskinspråksprogram som demonstrerar överföringen av variabler. Det är mycket likt det förra men studera noga skillnaderna.

```
Adress    Kod       Mnemonic       Kommentar
(Hex)     (Hex)
------    -----     --------       ---------

C000      C5        PUSH BC        Sparar BC-registerparet
C001      D4        PUSH DE        Sparar DE-registerparet
C002      E5        PUSH HL        Sparar HL-registerparet
C003      F5        PUSH AF        Sparar Accumulator-registret
C004      23        INC  HL        Flyttar pekaren till
C005      23        INC  HL        eventuellt heltal
C006      FE        CP   2         Testa om heltal
C007      02
C008      C2        JP   NZ,C00FH  Hoppa om inte heltal
C009      0F
C00A      C0
C00B      46        LD   B,(HL)    Laddar B med överförd variabel
C00C      C3        JP   C011H     Hoppa för att inte ladda B igen
C00D      11
C00E      C0
C00F      06        LD   B,20      Laddar B-registret med 20.
C010      14                       B används här som loopräknare
C011      3E        LD   A,"C"     Laddar A med ASCII-värdet
C012      43                       av "C"
C013      F5        PUSH AF        Sparar A:s värde
C014      CD        CALL OUTDO     Matar ut ASCII-koden i A
C015      18                       på skärmen
C016      00                       (Se Kernel för mer info)
C017      11        LD   DE,8000H  Laddar DE med antalet varv
C018      00
C019      80
C01A      1B        DEC  DE        Minskar DE ett steg
C01B      7A        LD   A,D       Enda gången (D OR E) är noll
C01C      B3        OR   E         är när DE är noll
C01D      C2        JP   NZ,C01AH  Forsätt om DE<>0
C01E      1A
C01F      C0
C020      F1        POP  AF        Återställ A:s värde
C021      3C        INC  A         Öka ASCII-koden i A
C022      05        DEC  B         Minska loopräknaren
C023      C2        JP   NZ,C013H  Fortsätt om inte slut än
C024      13
C025      C0
C026      36        LD   (HL),FFH  Ladda lägsta halvan av talet
C027      FF                       med 255
C028      23        INC  HL        Flytta pekaren till övre halvan
C029      36        LD   (HL),FFH  Ladda högsta halvan av talet
C02A      FF                       med 255
C02B      F1        POP  AF        Återställ A
C02C      E1        POP  HL        Återställ HL
C02D      D1        POP  DE        Återställ DE
C02E      C1        POP  BC        Återställ BC
C02F      C9        RET            Återvänd till Basic
```

När du har förstått hur programmet fungerar så ändrar du de gamla DATA-satserna i det föregående Basic-programmet så att du istället matar in det nya programmet.
Glöm inte att ändra pekarna till adress C03OH istället för adress C02OH eftersom detta nya program är längre än det gamla.
När du skrivit det nya Basic-programmet och kört det ligger maskinspråksprogrammet dolt på adress C000H-C02FH.
Du kallar på det genom följande Basic-sekvens.

```
10 DEF USR0=&HC000
20 INPUT B%
30 IF B%>255 GOTO 20
40 A%=USR0(B%)
50 PRINT
60 PRINT A%,B%
```

Du ser då att maskinspråksprogrammet skriver ut så många tecken som finns i B%. Dessutom sätts värdet av den utgående variabeln A% till -1 (försök med PRINT &HFFFF).

Om du ändrar programmet så att det ser ut som nedanstående märker du att oberoende vilket värde du ger B# skriver den ut 20 tecken. Detta därför att maskinspråksprogrammet testar om den ingående variabeln är ett heltal, om inte så sätter den värdet till 20.

```
10 DEF USR0=&HC000
20 INPUT B#
30 A#=USR0(B#)
50 PRINT
60 PRINT A#,B#
```

Som du ser blir A#:s värde lite konstigt.
Det beror på att du har byggt upp talet på ett felaktigt sätt.

Du ska efter den här genomgången kunna länka in och använda dina egna maskinspråksprogram. Har du haft problem med att förstå vad som står här bör du läsa på din Z-80 lärobok.
Vill du ha flera exempel på program så finns programmet för att komma åt de 32K RAM som finns dolt i maskinen.
Programmet står på sid 90 och framåt.

Sammanfattning av parameteröverföring med USR

Vid anrop med USR ligger variabeltypskoden i A.

2 Heltal
3 Sträng
4 Flyttal enkel precision
5 Flyttal dubbel precision

Dessutom ligger adress till variabeln i HL enligt följande system.

Heltal :	HL+2	LSB
	HL+3	MSB
Sträng :	HL+2	LSB adress till strängbeskrivning
	HL+3	MSB " " "
Sträng-beskrivning:	adr	stränglängd
	adr+1	LSB adress till sträng
	adr+2	MSB " " "
Flyttal enkel prec:	HL	exponent
	HL+1	2 högsta siffror i BCD-kod
	..HL+4	2 lägsta " i "
Flyttal dubbel prec:	HL	exponent
	HL+1	2 högsta siffror i BCD-kod
	..HL+4	2 lägsta " i "

Hopptabell (Kernel)

Kernel "Kärna" är den benämning man ger den hopptabell som brukar ligga i början på varje större program. I denna hopptabell finns hopp till de flesta rutiner som man kan ha nytta av separat, t.ex. diskhanteringsrutiner, skärm och tangentbordsrutiner m.m. Fördelen med att ha alla anrop till rutinerna samlade till ett ställe är att man kan stuva om resten av programmet så mycket som man önskar utan att program som använder dessa rutiner behöver skrivas om. Kernel-tabellen ligger nämligen kvar på samma ställe hela tiden.

Beskrivning av hopptabellen som ligger i början av Spectravideons BASIC-tolk.

Listan är upplagd på följande sätt:
Hexadecimal adress, namn och kort beskrivning.

Förklaring till vissa förkortningar:

Z,C	CPU-flaggor
B,C,D,E,H,L,A	CPU-register
BC,DE,HL	CPU-registerpar
Tgb	Tangentbord
Lpt	Skrivare (centronicsinterface)
FAC	Accumulator för flyttal (F923H)
FACLO	Accumulator för heltal (F925H)

	0008 SYNCHK	Looks at the current (HL) character to check if it equals the character after SYNCHK, otherwise "Syntax error"
	0010 CHRGET	Incr HL, get character into A and set flags: C=numeric, Z=EOL (":" or 0)
	0018 OUTDO	Output char in A using PRTFLG, TTYPOS etc.
	0020 COMPAR	Compares HL & DE : HL>DE sets C, HL=DE sets Z
	0028 SIGN	Returns A=-1 if FAC<0, 0 if FAC=0, 1 if FAC>0, works on single and double precision constants.
	0030 GETYPR	Get valtyp & set condition codes (flags) INT=2,SIGN STR=3,Z SNG=4,ODD P DBL=8,NC
0038	3CC2 KEYINT	Interrupt handler
003B	3DCA CHSNS	Testa om tecken från Tgb finns, Z om finns ej
003E	403D CHGET	Väntar på tecken från Tgb. Tecknet hamnar i A
0041	3938 CHPSTT	Testar om Lpt Ready. Z om ej Ready.
0044	3915 CHPLPT	Skriver ut tecknet i A på Lpt
0047	3541 INITXT	Initiera screen 0
004A	3610 INIGRP	Initiera screen 1
004D	3665 INIMLT	Initiera screen 2
0050	3B95 FNKSB	Visa funktionstangenterna om screen 0 och flagga (CNSDF) satt.
0053	3B86 ERAFNK	Radera funktionstangentvisningen
0056	3B9F DSPFNK	Visa funktionstangenterna
0059	3498 RSTFNK	Initiera funktionstangenterna

```
005C 3512 BREKX   Kolla om CTRL-STOP. C om nertryckt
005F 3476 JMPBNK  Hoppa till minnesbank
0062 3480 CALBNK  Kalla på subrutin i en annan minnesbank
0066 0138 NMI     NMI-handler
0069 203A CSRDON  Sätt på kassett-motor och vänta SYNC
                  och HEADER
006C 2016 CASIN   Läs data byte för byte till A från
                  kassett-bandspelaren
006F 207C CTOFF   Stäng av kassett-motor
0072 2059 CWRTON  Sätt på kassett-motor och skriv SYNC
                  och HEADER
0075 2026 CASOUT  Skriv data byte för byte från A
                  på kassett-bandspelaren
0078 206C CTWOFF  Skriva nolla på kassett och stanna motorn
007B 6474 CRDO    Put CR to output device
007E 6463 CDRONZ  Put CR to output device if not at left position
0081 6415 OUTDLP  Put char in A to Lpt and decode TAB, CR and skip
                  rest of the Ctrl-characters
0084 692C STRINI
0087 6959 PUTNEW
008A 6AD5 FRESTR
008D 697D STROUT
0090 08C9 READYR  Reset stack when BASIC externally stopped & then
                  restarted
0093 08ED SNERR   Print "Syntax error"
0096 0907 ERROR   Print error number in E
0099 0981 NTDERR  ?
009C 09AF READY   Warmstart
009F 0AE5 LINKER  Goes through the program and fixes links
00A2 0E3E NEWSTT  New statement fetcher
00A5 0F9E FCERR   Function call error
00A8 643D FINLPT  Reset output device to video and move Lpt head
                  to left margin
00AB 162D EVAL    Evaluate variable, constant, function call
                  Put result in FAC
00AE 14CA FRMEVL  Formula evaluation routine
00B1 1AA6 GETBYT  ?
00B4 1CB9 FRMQNT  Get integer value from formula
00B7 1AA9 CONINT  Convert FAC to an integer in DE
00BA 183C SNGFLT  ?
00BD 1A99 GETIN2  ?
00C0 0B44 CRUNCH  Translate all reserved words into tokens
00C3 0AE9 CHEAD   Goes through program and fixes links from
                  position in DE
00C6 55C9 CONIA   Convert signed number in A to integer
00C9 5968 INEG2   Convert integer into single precision, put in DAC
00CC 56C4 MAKINT  Put HL in FACLO, set valtyp to int
00CF 204D DIOERR
00D2 3966 POPALL
00D5 74B0 EOF     Close file with # in FAC
00D8 59C9 POPHRT
00DB 5B44 LINPRT
```

```
00DE 6545 OMERR   Fixes program links, resets SAVSTK and prints
                  "OUT OF MEMORY"
00E1 656A RUNC    Intitiates for run
00E4 6FD3 NAMSCN  Scan filename and device
00E7 702F SCNBLK  Scan block
00EA 7033 GETFLP  Get pointer to file with number in FAC
00ED 7036 GETPTR  Same as above but number in A
00F0 7073 SETFIL  Tests if file wit # in a A is open and returns
                  pointer
00F3 70C3 NULOPN  Open file: Drive in D, mode in E (4=filemode=
                  no "FOR", 1="INPUT", 2="OUTPUT", 8="APPEND"),
                  file # in A
00F6 70EA CLSFIL  Close file # in A
00F9 710A NOCLSB
00FC 737D CLSALL  Close all files if NLONLY<>0
00FF 73CA FILOU1  Writes character in (A) onto disc
0102 73F1 INDSKC  Gets a char from disk into A. Carry if EOF
0105 7460 CLRBUF  Clears data buffer of selected file. On return HL
                  points at start of buffer
0108 7474 DOCLR   Clear B # of bytes starting at HL
010B 75D0 NOSKCR
010E 75FA DERBFN  "BAD FILE NAME"
0111 75FD DERFAO  "FILE ALREADY OPEN"
0114 7603 DERFNF  "FILE NOT FOUND"
0117 7607 DERFNO  "FILE NOT OPEN"
011A 760F DERIER  "INTERNAL ERROR"
011D 7612 DERRPE  "READ PAST EOF"
0120 170B MAKUPL
0123 0BCA CRN2ND
0126 346A PUTBNK
0129 3463 GETBNK
012C 721A GETDEV  Returns device # in A from PTRFIL (F997H)
012F 7210 NOROOM  No room for program loaded from disk: erase, close
                  files and print "OUT OF MEMORY"
0132 7209 CHKTOP  Check if enough room, else fall into NOROOM
0135 747D GETBF1
0138 7609 DERFOV  "FIELD OVERFLOW"
013B 6FD6 NAMSC1
013E 7615 DERSAP
0141 7067 FILSCN
0144 747A BETBUF
0147 40BE BEEP
014A 3C5A CNVCDO
014D 3CBC GETLEN
0150 3CA7 GETTRM
0153 3C39 GETCOD
0156 3CB5 SETTRM
0159 3CB3 TERMIN
015C 1365 FINPRT
015F 1AA5 GTBYTC
0162 0F99 INTIDX
0165 09FB INILIN
0168 5783 FRCSTR
016B 6993 GETSPA
016E 405D CKCNTC
```

```
0171 6557 SCRTCH  New command
0174 56B5 FRCINT
0177 7402 INDSKE
017A 6066 PTRGET  Get pointer to variable
017D 71AB SPSVEX
```

Namn : EVAL Adress : 162DH Kernel : 00ABH

Rutinen beräknar (evaluate) värdet av variabeln, konstanten eller funktionsanropet som HL pekar på. Efter anropet hamnar typen av variabel i VALTYP (F793H) där 2=heltal, 3=sträng, 4=enkel precision och 8=dubbel precision.
Tal i enkel & dubbel precision hamnar i FAC (F923H), heltal i FACLO (F925H) och för strängar hamnar adress till en sträng-beskrivning i FACLO.
Strängbeskrivningen är uppbyggd av 3 byte, 1 byte stränglängd och 2 byte strängadress.
Efteråt pekar HL på tecknet efter den uträknade funktionen.

Namn : PTRGET Adress : 6066H Kernel : 017AH

Hämtar adressen till variabeln vars namn HL pekar på. Adressen hamnar i DE och HL pekar på positionen efter variabelnamnet vid uthopp. Finns inte variabeln får DE värdet noll.

Här följer en lista på var rutinerna för att exekvera kommandon och instruktioner ligger i BASIC-tolken Rev 1.0.
Dessutom visas internkoden (Token) för alla kommandon.

Kommando	Token	Adress
ABS	06	55B1
AND	F8	1804
ASC	15	6B10
ATN	0E	5139
ATTR$	E9	34D3
AUTO	A9	11F5
BEEP	C0	40BE
BIN$	1D	6904
BLOAD	CD	7684
BSAVE	CE	7624
CDBL	20	5765
CHR$	16	6B20
CINT	1E	56B5
CIRCLE	BC	2652
CLEAR	92	67A6
CLICK	C8	31AF
CLOAD	9B	1EAA
CLOSE	B4	7375
CLS	9F	3777
CMD	D7	34C4
COLOR	BD	4552
CONT	99	671B
COPY	D6	34BF
COS	0C	50B8
CSAVE	9A	1E15
CSNG	1F	56DD
CSRLIN	E8	31C7
CVD	2A	7331
CVI	28	732B
CVS	29	732E
DATA	84	109B
DEF	97	188A
DEFDBL	AE	0F65
DEFINT	AC	0F5F
DEFSNG	AD	0F62
DEFSTR	AB	0F5C
DELETE	A8	1C6C
DIAL	D0	79C2
DIM	86	6061
DRAW	BE	29DA
DSKF	26	34C9
DSKI$	EA	34CE
DSKO$	D1	34A6
ELSE	A1	109D
END	81	66CF
EOF	2B	74B0
EQV	FB	181C
ERASE	A5	676E

Kommando	Token	Adress
ERL	E1	1671
ERR	E2	1663
ERROR	A6	11EA
EXP	0B	526B
FIELD	B1	72CD
FILES	B7	73B2
FIX	21	57E9
FN	DE	18AD
FOR	82	0D65
FPOS	27	74C6
FREE	0F	6CF7
GET	B2	2FB4
GOSUB	8D	0FF6
GOTO	89	1028
HEX$	1B	68FF
IF	8B	1225
IMP	FC	1824
INKEY$	EC	64F3
INP	10	1A37
INPUT	85	13D2
INSTR	E5	6BF0
INT	05	3048
IPL	D5	34BA
KEY	C7	3120
KILL	D4	34B5
LEFT$	01	6B66
LEN	12	6B04
LET	88	10C0
LFILES	BB	73AD
LINE	AF	1374
LIST	93	1AB8
LLIST	9E	1AB3
LOAD	B5	7121
LOC	2C	7484
LOCATE	D8	2FD1
LOF	2D	749A
LOG	0A	5197
LPOS	1C	1834
LPRINT	9D	125D
LSET	B8	7228
MAX	CA	7CBA
MDM	CF	3036
MERGE	B6	7122
MID$	03	6C73
MKD$	30	7318
MKI$	2E	7312
MKS$	2F	7315
MOD	FD	32BD
MON	CB	7B44
MOTOR	CC	2BE5
NAME	D3	34B0
NEW	94	6556
NEXT	83	6821
NOT	E0	????
OCT$	1A	68FA

Kommando	Token	Adress
OFF	EB	6909
ON	95	1124
OPEN	B0	7080
OR	F9	17FE
OUT	9C	1A4C
PAD	25	32BD
PAINT	BF	24FC
PDL	24	3280
PEEK	17	1CA6
PLAY	C1	2C24
POINT	ED	2346
POKE	98	1CAD
POSE	11	1839
PRESET	C3	2328
PRINT	91	1265
PSET	C2	232D
PUT	B3	2FB1
READ	87	1405
REM	8F	109D
RENUM	AA	1CF0
RESTORE	8C	66AE
RESUME	A7	119D
RETURN	8E	1061
RIGHT$	02	6B96
RND	08	5300
RSET	B9	7227
RUN	8A	0FE2
SAVE	BA	7167
SCREEN	C5	459A
SET	D2	34AB
SGN	04	55C6
SIN	09	50D1
SOUND	C4	2BFD
SPACE$	19	6B4D
SPRITE	EE	45D2
SQR	07	5222
STEP	DC	????
STICK	22	3206
STOP	90	66C8
STR$	13	6909
STRIG	23	3263
STRING$	E3	6B2E
SWAP	A4	6735
SWITCH	C9	337F
TAB	DB	????
TAN	0D	5120
THEN	DA	????
TIME	EF	31D3
=TIME		31BD
TO	D9	????
TROFF	A3	6730
TRON	A2	672F
USING	E4	????
USR	DD	1842
VAL	14	6BC0

Kommando	Token	Adress
VARPTR	E7	6066
VPEEK	18	46F2
VPOKE	C6	46D8
WAIT	96	1A52
WIDTH	A0	1AC6
XOR	FA	1812
>	F0	
=	F1	
<	F2	
+	F3	
-	F4	
*	F5	
/	F6	
Ü	F7	

Memory-mapp

Lista på användbara variabler i Spectravideons BASIC-tolk.

Listan är upplagd på följande sätt:
Hexadecimal adress, antal byte, namn och kort förklaring.

B=byte (1) W=Word (2) I=Instruktion (3)

```
F500   RAMLOW
F504   RNDCNT
F506   RNDTAB
F52B   USRTAB
F53F B ERRFLG  Error number
F540   LPTLST  Last Lpt operation 0=LF, 0<>print
F541   LPTPOS  Position of Lpt print head
F542   PRTFLG  =0  output device = CRT
               =1     "     "    = LPT
F543   LINLEN  Line length (39,40,80)
F545 B RUBSW   Rubout switch=1 means inside the processing of a
               rubout (inlin)
F546 W STKTOP  Top of stack, start of strigarea, 2:nd parameter in
               CLEAR statement
F548 W CURLIN  Current line #, FFFFH at direct statement in execution
F54A W TXTTAB  Start of Basic-program
F54C W VLZADR  Adress of character replaced by VAL
F54E B         ":": a colon for restarting input
F54F   KBUF    This is the CRUNCH buffer
F68D   BUFMIN
F68E   BUF     Input line buffer
F790 B ENDBUF  Make sure overrun stops
F791 B TTYPOS  Store terminal position
F792 B DIMFLG
F794 B VALTYP  Type of variable
F798   CONSAV
F7A2 W MEMSIZ  End of string area, start of file-buffer area
F7A4 W TEMPPT  Temporary pointer
F7A6   TEMPST
F7C4 B DSCTMP  Character count ) temporary string description
F7C5 W DSCPTR  pointer         )
F7C7 W FRETOP
F7C9   TEMP3
F7CB   TEMP8
F7CD   ENDFOR
F7D1   SUBFLG
F7D2   USFLG
F7D3   TEMP
F7D5   PTRFLG
F7D6 B AUTFLG  <>0 means AUTO mode
F7D7 W AUTLIN  Initial line for auto
F7D9 W AUTINC  Increment in auto
F7DB W SAVTXT
F7DD W SAVSTK  Stack temporary saved during command execution
F7DF W ERRLIN  Line number where last error occurred
```

```
F7E1 W DDT     Current line for edit & list
F7E3 W ERRTXT  Text pointer for use by "RESUME"
F7E5 W ONELIN  Current line to go to when error occurs
F7E7 B ONEFLG  =1 if executing error trap routine, else 0
F7E8 W TEMP2   Text pointer
F7EA W OLDLIN  Old line number (setup by ÜC, "STOP" or "END")
F7EC W OLDTXT  Old text pointer
F7EE W VARTAB  Start of simple variables
F7F0 W ARYTAB  Start of array variables
F7F2 W STREND  End of array variables
F7F4 W DATPTR  Adreess-1 of line where read DATA
F7F6-  DEFTBL  Default valtyp for each letter of the alphabet.
F80F           Setup by clear and changed by DEFSTR, DEFSNG,
               DEFDBL, DEFINT and used when a variable isn't
               followed by !,#,$ or %
F810 W PRMSTK
F812 W PRMLEN
F814 W PARM1
F878   PRMPRV
F87A   PRMLN2
F8E0   PRMFLG
F8E1   ARYTA2
F8E3   NOFUNS
F8E4   TEMP9
F8E6   FUNACT
F8E8   VLZDAT
F8E9   SWMTMP
F8F1   TRCFLG  Zero means no trace in progress
```

RAM temporary area for the math package routines

```
F8F2-  FBUFFR
F91A   FMLTT1
F91B   FMLTT2
F91D W DECTMP  Used by decimal int to float
F91F W DECTM2  Used by divide
F921 B DECCNT  Used by divide
F922 B         Temp complement of sign
F923-  DACFAC  Decimal accumulator
F932
F925   FACLO   Integer accumulator
               Holding registers for decimal multiply (8 byte each)
F933   HOLD8   X*128
               X*64
               X*32
F948   HOLD5   X*16
               X*8
               X*4
F963   HOLD2   X*2
F968   HOLD    X*1
F973           Temp sign complement
F974-  ARG     Argument accumulator
F983
F984-  RNDX    Holds last random number generated
F98C
```

Variables for disk

```
F98C B MAXDRV  Highest legal drive #
F98D B MAXFIL  Highest legal file #
F98E W FILTAB
F990   ORVTAB
F992   NULBUF
F994   CURDRW
F995   DRWPTR
F997   PTRFIL  Point at file data
F999   FREPLC
F99B   LSTFRE
F99D   RUNFLG  Non-zero if run after load
F99E   FILNAM
F9A7   FILNM2  Holds filename for NAME
F9B0 B LSTTRK  Last track accessed by DSKI$ or DSKO$
F9B1 B LSTSCT
F9B2 B NLONLY  Non-zero if loading program
F9B3 B SAVFLG  Non-zero if saving program
F9B4 W SAVEND  End adress for BSAVE
F9B6   DSKBSY
F9B7   ERRCNT
F9B8   ERRCN1
F9B9   RAWFLG  Zero for read afer write
F9BA   EBCFLG
F9BB   SAVEBC
F9BC   STAT0
F9BD   STAT1
F9BE   TSTACK
F9EF           Buffert för formatterat filnamn
```

Följande variabler initieras vid uppstart

```
FA02 B CLIKSW  Click on/off (0=off, 1=on)
FA03 B CSRY    Cursorns vertikalposition
FA04 B CSRX    Cursorns horisontalposition
FA05 B CSRSW   Cursorvisningsflagga (0=visa ej, FF=visa, 40 tkn)
FA06 B CNSDFG  Funktionstangentsvisning (0=visa ej, FF=visa)
FA07 B RG1SAV  ?
FA08 B TRGFLG  ?
FA09 B SPCFLG  ?
FA0A B FGRCLR  Förgrundsfärg
FA0B B BAKCLR  Bakgrundsfärg
FA0C B BRDCLR  Ramens färg
FA0D I MAXUPD  Hopp
FA10 I MINUPD  Hopp
FA13 B ATRBYT  Atrribute
FA14 I PUTFN   Adress satt av PGINIT
FA17 W QUEUES  Adress av kötabell, används av rutin QUEUTL
FA19 B REPCNT  Something with STOP-key
FA1A W PUTPNT  Var spara nästa tecken i radbufferten
FA1C W GETPNT  Var hämta nästa tecken i radbufferten
```

```
FA1E-FA2D      Betydelse av funktionstangent 1
FA2E-FA3D      F2
FA3E-FA4D      F3
FA4E-FA5D      F4
FA5E-FA6D      F5
FA6E-FA7D      F6
FA7E-FA8D      F7
FA8E-FA9D      F8
FA9E-FAAD      F9
FAAE-FABD      F10
FABE B XODFLG  ?
FABF B COMMSK  ?
FAC0-FADD      Rutin för test om cartridge finns, endast initiering


Lista på variabler som inte initieras vid uppstart

FAC0 W CLOC    ?
FAC2 B COMMSK  ?

Variabler för cirkelritning

FAC3 W ASPECT  Hjöd/breddförhållande
FAC5 W CENCNT  Sluträknare
FAC7 B CLINEF  Flagga om rita linje till centrum
FAC8 W CNPNTS  Punkter att plotta
FACA B CPLOTF  Flagga för plottningspolaritet
FACB W CPCNT   1/8 av antalet punkter som ska plottas
FACD W CPCNTB  Antal punkter i cirkeln
FACF W CRCSUM  Cirkel summa
FAD1 W CSTCNT  Start count
FAD3 B CSCLXY  Skalning X Y
FAD4 W CSAVEA  ADVGRP C save area
FAD6 B CSAVEM  ADVGRP C save area
FAD7 W CXOFF   X offset from center save location
FAD9 W CYOFF   Y offset save location


Variabler för paint

FADB B LOHMSK  RAM save area for left overhang
FADC B LOHDIR  ?
FADD W LOHADR  ?
FADF W LOHCNT  ?
FAE1 W SKPCNT  Skip count
FAE3 W MOVCNT  Move count
FAE5 B PDIREC  Paint direction
FAE6 B LFPROG  ?
FAE7 B RTPROG  ?


Variabler för PUT/GET

FAE8 B PUTFLG  ?
FAE9 W MINDEL  ?
FAEB W MAXDEL  ?
FAED W ARYPTR  ?
```

Variabler för MACLNG

```
FAEF W MCLTAB  ?
FAF1 B MCLFLG  Indicates PLAY/DRAW
```

Köer för PLAY kommando

```
FAF2   QUETAB  4 köer, 6 byte i varje
FB0A   QUEBAK  4 byte för BCKQ
FB0E   VOICAQ  128 byte kö för kanal A
FB8E   VOICBQ  128 byte kö för kanal B
FC0E   VOICCQ  128 byte kö för kanal C
FC8E   RS2IQ   64 byte in-kö för RS-232
```

Diverse musik-variabler

```
FCCE B PRSCNT  D1-D0 = # strängar analyserade
               D7=0 om pass 1, 1 om inte
FCCF W SAVSP   Spara huvudstackenpekaren under spelning
FCD1 B VOICEN  Vilken kanal som analyseras
FCD2 W SAVVOL  Spara volym för paus
FCD4 B MCLLEN  ?
FCD5 W MCLPTR  ?
FCD7 B QUEUEN  Used bu intime-action-deque

FD68 B FNKSWI  Shift status, 1=ej nertryckt, 0=nertryckt
FD69-FD72 FNKFLG  Flaggor för funktionstangenter
FD73 B CNGSBF  Global event flag
FD74 B CLIKFL  ?
FD75-FD7F OLDKEY Tangenbordets föregående status
FD80-FD8A NEWKEY Tangenbordets nuvarande status
FD86 B SETKEY  Funktionstangentvisning FF=F1-F5, FE=F6-F10
FD8B-FDB2 Radbuffert
FDE4 W BOTTOM  RAM-minnets lägsta adress
FDE6 W HIMEM   RAM-minnets högsta adress
FDE8 W TXPSAV  Text pointer save area
FDEA B CASATR  Cassette attribute of file being read
FE06 B         KEY ON=! OFF! STOP!  (0=off, 1=on, 2=stop)
FE09 B         STOP      - II -     (0=off, 1=onm 3=stop)
FE0C B         SPRITE         - II -
FE18 B         INTERVAL       - II -
FE1B B         MDM            - II -
FE2A B RTYCNT  Antal försök att boota
FE2B B INTFLG  ?
FE2C B PADX    ?
FE2D B PADY    ?
FE2E W JIFFY   Realtidsklockan
FE30 W INTVAL  ?
FE38 B CAPST   Caps lock (0=av, 20=på)
FE39 B FLBMEM  0 om laddar BASIC-program, FF om sparar binärfil
FE3A B SCRMOD  Screen mode (0, 1, 2)
FE3B B SPRSIZ  Storlek på spriten
FE3C B RGOSAV  ?
FE3D B STATFL  ?
FE3E B KBDPRV  ?
```

```
FE3F B CASPRV  ?
FE40 B MSMPRV  ?
FE41 B BRDATR  Kantfärg för PAINT
FE42 W GXPOS   ?
FE44 W GYPOS   ?
FE46 W GRPACX  Grafisk accumulator
FE48 W GRPACY  ?
FE4A B DRWFLG  ?
FE4B B DRWSCL  DRAW skal-faktor - 0 = ingen skalning
FE4C B DRWANG  DRAW-vinkel (0-3)
```

Variabler för modem

```
FE4D   DATCNT  4 byte reserverade
FE51 B SIDFLG  Flagga för SI/SO kontroll logik
FE52 B RCVXOF  Flagga för Xoff-mottaget
FE53 B SNTXOF  Flagga för Xoff-redan-sänt
FE54 B RCVSFT  Shift-status för mottagning
FE55 B SNDSFT  Shift-status för sändning
FE56 B ADDPM   Dial pulse speed
```

Variabler för BLOAD och SAVE

```
FE57 B RUNBF   "R" om RUN efter laddning
FE57 W SAVENT  Startadress för binärfil
```

Följande minnesarea används för att spara CPU:ns registerinnehåll vid start och stopp av program i monitorn.

```
FE5A W REGPC   Programräknare (Program Counter)
FE5C W REGSP   Stackpekare
FE5E W REGHL   Registerpar HL
FE60 W REGDE        "      DE
FE62 W REGBC        "      BC
FE64 B REGA    Accumulator
FE65 B REGF    Flaggregistret
FE66 W REGIY   IY-registret
FE68 W REGIX   IX-registret
FE6A W REGHLT  Andra uppsättningen av registerparet HL
FE6C W REGDET  Andra uppsättningen av registerparet DE
FE6E W REGBCT  Andra uppsättningen av registerparet BC
FE70 B REGAT   Andra accumulatorn
FE71 B REFFT   Andra flaggregistret
FE72 B MONFLG  FF om inne i monitorn, annars 00
FE73 W SAVESP  ?
FE75 B SWIFLG  ?
FE76 W SPSAVE  Stack pointer save area for SWITCH statement
FE78 B SCNCNT  ?
```

Definition av programlänkar (Hooks)

En Hook är precis vad det låter som, en "Hake" för att hänga på egna programdelar i BASIC-ROM-minnet.
Eftersom ROM-minnet är väldigt svårt att ändra har man lagt ett stort antal CALL från ROM-minnet till RAM. I RAM-minnet ligger det normalt en RET (return) och programmet återvänder till ROM-minnet och fortsätter som om ingenting hade hänt. Nu kan man väldigt enkelt ändra i RAM-minnet genom att använda instruktionen POKE.
Ex. Titta på adress FEF4. Dit hoppar BASIC-rutinen för LIST-kommandot. Man kan nu väldigt enkelt stänga av LIST-funktionen genom att ändra på denna adress. Om vi skriver följande:

```
POKE &HFEF4,&HC3 : POKE &HFEF5,0 : POKE &HFEF6,0
```

Detta lägger in ett hopp till adress 09AFH som är varmstart (se Kerneltabellen) och när man försöker lista svarar datorn bara "Ok" även om det finns program inne.
Du kan köra programmet och göra allt annat, utom just at lista.
Man återfår LIST-funktionen genom: POKE &HFEF4,&HC9
Som åter lägger in en RET.

Listan är upplagd på följande sätt:
Hexadecimal adress, antal byte, namn och kort förklaring.
Värdena inom parantes är varifrån rutinen kallas.
B=byte (1) W=Word (2) I=Instruktion (3)

```
FE79 I H.KEYI (3CC6H)
FE7C I H.PRTF (129FH)
FE7E I H.DGET
FE82 I H.INDS
FE85 I H.SCNE (19D5H)
FE88 I H.SNGF
FE8B I H.FPOS FPOS (74C6H)
FE8E I H.READ
FE91 I H.SIRE (0BEDH)
FE94 I H.MAIN
FE97 I H.RSLF
FE9A I H.LOC  (7994J)
FE9D I H.BAKU (75CAH)
FEA0 I H.STKE (65C0H)
FEA3 I H.PARD (770BH)
FEA6 I H.FRET
FEA9 I H.NTFL (7104H)
FEAC I H.NTFN (0C16H)
FEAF I H.CLEA (6571H)
FEB2 I H.SAV0 (7884H, 749AH, 74B0H, 74C6H)
FEB5 I H.SAVE
FEB8 I H.FILE FILES (73B2H)
FEBB I H.LOF  (74AAH)
FEBE I H.NTPL (150CH)
FEC1 I H.NODE
FEC4 I H.DOGR
FEC7 I H.MERG
FECA I H.EOF  (74C0H)
```

```
FECD I H.PTRG Get pointer to variable PTRGET (606BH)
FED0 I H.NOFO (70BFH)
FED3 I H.PRGE PRGEND (08CFH)
FED6 I H.BUFL
FED9 I H.CRDO CR to output device (6474H)
FEDC I H.OKNO
FEDF I H.GETP
FEE2 I H.LOPO (657FH)
FEE5 I H.DEVN
FEE8 I H.SETF SETFIL (7073H)
FEEB I H.NULO NULOPN (70D8H)
FEEE I H.RETU (1061H)
FEF1 I H.CLRC
FEF4 I H.LIST List (1AB8H)
FEF7 I H.RUNC (656AH)
FEFA I H.EVAL Evaluate routine (163FH)
FEFD I H.ISMI
FF00 I H.COMP COMPRT (12DEH)
FF03 I H.FRQI (1CC9H)
FF06 I H.DIRD
FF09 I H.OUTD OUTDO (0019H)
FF0C I H.NOTR (0D2CH)
FF0F I H.GEND
FF12 I H.FILO FILOU1 (73CAH)
FF15 I H.ISFL (6812H)
FF18 I H.ERRP Selection of error message (096CH)
FF1B I H.ERRF Error routine (0989H)
FF1E I H.TRMN TRMNOK (13B3H)
FF21 I H.CRUS (0BC4H)
FF24 I H.CRUN (0B4BH)
FF27 I H.FINP FINPRT (1365H)
FF2A I H.FRME Formula evaluator (14D3H)
FF2D I H.BINS
FF30 I H.FINI
FF33 I H.BINL
FF36 I H.FINE
FF39 I H.FING (17A0H)
FF3C I H.INCH
FF3F I H.WIDT Width (1A6FH)
FF42 I H.PINL Screen editor (6D13H)
FF45 I H.QINL
FF48 I H.INLI
FF4B I H.DSKC
FF4E I H.ERAF Erase function key (3B86H)
FF51 I H.DSPF Display function key (3B95H)
FF54 I H.NEWS (0E3EH)
FF57 I H.GONE (0E88H)
FF5A I H.DMOT
FF5D I H.MDMO
FF60 I H.MDMC
FF63 I H.MDMW
FF66 I H.MDMI
FF69 I H.MOME
FF6C I H.MDMB
FF6F I H.DIAL
```

```
FF72 I H.RS2I Modem (79D8H)
FF75 I H.DNGO
FF78 I H.KYCL
FF7B I H.KYEA
FF7E I H.NMI  NMI handler (0180H)
FF81 I H.KEYC
FF84 I H.MON
FF87 I H.BADC
FF8A I H.DSKO DSKO$ (34A6H)
FF8D I H.SETS SETS  (34ABH)
FF90 I H.NAME NAME  (34B0H)
FF93 I H.KILL KILL  (34B5H)
FF96 I H.IPL  IPL   (34BAH)
FF99 I H.COPY COPY  (34A6H)
FF9C I H.CMD  CMD   (34C4H)
FF9F I H.DSKF DSKF  (34C9H)
FFA2 I H.DSKI DSKI$ (34CEH)
FFA5 I H.ATTR ATTR$ (34D3H)
FFA8 I H.MONE
FFAB I H.INIP
FFAE I H.CHPU
FFB1 I H.TOTE
FFB4

End of work-area

FFFE W PWRFLG "SM" if not first power-up
```

Beskrivning av videprocessorn (VDP) och videominnet

Kommunikation med videominnet

När du ska skriva eller läsa i videominnet måste du använda ett internt register VPD:n. Detta register används som adress-register och bestämmer var data ska skrivas eller läsas.

Eftersom VDP:n kan adressera max 16K videominne är registret på 14 bitar (2 Ü 14 = 16384). När du ska ställa registret matar du ut 2 byte (totalt 16 bitar) till VDP:n. De resterande 2 bitarna som inte är adress används för att bestämma om VDP:n ska läsa eller skriva i videominnet eller om man ska programmera andra register. Vi säger att du t.ex. ska skriva någonting på adress 10 decimalt. 10 decimalt är 000A hexadecimalt.
Om du ser på tabellen på sidan 75, Skriv till VRAM, så står där att man först ska börja med att mata ut den lägsta adressbyten på porten där Mode=1, d.v.s. port 81H.

Mata alltså ut 0AH på port 81H.

Nästa steg är att mata ut den högsta byten med bit 6 satt.
Kom ihåg att bit 6 & 7 är de 2 bitarna som inte behövs för att bestämma adressen. Mata alltså ut 40H på port 81H.
Till sist mata du ut den byte du vill ha skriven i videominnet på poten där Mode=0, d.v.s. port 80H.

Därefter ökas registret automatiskt ett steg så att det pekar på adress 11 decimalt. Om du vill att nästa byte hamnar på adress 11 kan du alltså skriva direkt utan att behöva ställa om adress-registret.

En demonstrationssekvens i Basic ser ut på följande sätt:

```
10 SCREEN 1
20 OUT &H81,&HA
30 OUT &H81,&H40
40 OUT &H80,&HF0
50 GOTO 40
```

Ett demonstrationsexempel i maskinspråk finns på sidan 75.

Programmering av videoprocessor

VDP:n innehåller 8 interna skriv-register (0-7). Av dessa är enbart register 0, 1 och 7 intressanta. De övriga registren bestämmer var i minnet olika tabeller ska ligga och det initierar datorn själv.

Register 0
Bit 0 External video input (alltid 0)
Bit 1 Mode M3

Register 1

Bit 0	Spriteförstoring	0=x1, 1=x2
Bit 1	Spritestorlek	0=8x8 pixels, 1=16x16 pixels
Bit 2	Alltid 0	
Bit 3	Mode M2	
Bit 4	Mode M1	
Bit 5	Enablar 50 Hz interrupt	0=disabled, 1=enabled
Bit 6	BLANK enable/disable	0=active display area blanked 1=enabled
Bit 7	Alltid 1	

M1	M2	M3	Mode
0	0	1	Högupplösningsmode
1	0	0	Textmode

Register 2
Name table base adress

Register 3
Color table base adress

Register 4
Pattern generator

Register 5
Sprite attribute table base adress

Register 6
Sprite pattern generator base adress

Register 7
Förgrundsfärg och bakgrundsfärg i TEXT-mode (screen 0)
De 4 lägsta bitarna=bakgrundsfärg dessutom kantfärg i screen 1 & 2
De 4 högsta bitarna=förgrundsfärg

VDP:ns status

VDP:n har ett read-only statusregister som indikerar när två sprites kolliderar eller när det finns mera än 4 sprites på en horisontell linje. VDP:n klarar bara av 4 sprites på samma horisontella linje. Annars visas bara de 4 som har högst prioritet.

Bit 0-4	Numret på den 5:e spriten på en horisontell linje.
Bit 5	1 om två eller flera sprites kolliderar
Bit 6	1 om det finns flera än fyra sprites på samma horisontella linje.
Bit 7	Kopia av interruptutgången (1=interrupt finns).

Beskrivning screen 0

Bildminnet ligger på adress 0 - 959 (3BFH). Adress 0 är positionen längst upp till vänster, adress 959 positionen längst ner till höger " " har kod 0, "!" har kod 1 o.s.v. t.o.m. "Ü"=94, därefter kommer tecknen inverterade med början på " "=96 och till sist de grafiska tecknen med början på kod 192.

Om du provkör följande program blir det klarare.

```
10 CLS
20 FOR I%=0 TO 255
30   VPOKE I%,I%
40 NEXT I%
```

Programmet matar ut hela teckenuppsättningen som finns med början på adress 0.
Karaktärsgeneratorn ligger i videominnet på adress 2048 - 4095 (800H - FFFH). Tecknen upptar 8 byte var. Du kan alltså själv ändra din teckenuppsättning.
Provkör och studera följande program.

```
10 CLS
15 PRINT "!!!!!!!!!!"
20 FOR I%=0 TO 7
25   READ A$
30   VPOKE 2048+8+I%,VAL("&B"+A$)
35   FOR J=0 TO 100 : NEXT J
40 NEXT I%
45 DATA 01110000
50 DATA 10001000
55 DATA 10000000
60 DATA 01100000
65 DATA 10000000
70 DATA 10001000
75 DATA 01110000
80 DATA 00000000
```

Programmet skriver först ut 10 st "!" längst upp till
vänster och programmerar därefter om utseendet på tecknen.
Prova att ändra i programmet och testa ut hur mönstret på
tecknen blir.

Beskrivning av screen 1

Följande areor används i screen 1

Pattern generator	0	- 6143	(17FFH)
Sprite attribute table	6912 (1B00H)	- 7093	(1B7FH)
Colour table	8192 (2000H)	- 14335	(37FFH)
Sprite pattern generator	14336 (3800H)	- 15359	(3BFFH)

Byte på adress 0 i pattern generatorn är de 8 horisontella
punkterna längst upp till vänster. Ettorna på positionen i minnet
väljer förgrundsfärg, nollorna bakgrundsfärg. Första
positionen i colour table (adress 2000H) bestämmer för- och
bakgrundsfärg på mönstret på adress 0 i minnet. De 4 högsta
bitarna på adressen är förgrundsfärg, de 4 lägsta bakgrundsfärg.

Kör följande program:

```
10 SCREEN 1
20 FOR I%=512 TO 512+255
30   VPOKE I%,(I% AND 4)/4*255
40 NEXT I%
99 GOTO 99
```

Då ser du hur mönstret byggs upp.
Om du sedan lägger till följande rader:

```
50 FOR I%=0 TO 15
55   FOR J%=0 TO 7
60     VPOKE 8704+I%*8+J%,I%+16
65   NEXT J%
70 NEXT I%
75 FOR I%=0 TO 15
80   FOR J%=0 TO 7
85     VPOKE 8704+128+I%*8+J%,I%*16+1
90   NEXT J%
95 NEXT I%
```

Så ser du hur färgläggningen fungerar. Programmet börjar att
färglägga de 128 första byten med svart förgrundsfärg och
16 olika bakgrundsfärger och fortsätter med att färglägga
de nästföljande 128 byte med svart bakgrundsfärg och 16 olika
förgrundsfärger.

Sprite attribute tabellen upptar 4 bytes/sprite enligt följande mönster.

Byte	Förklaring
1	Vertikal position
2	Horisontell position
3	Namn (ett 1-byte namn som pekar på en area om 8 byte i sprite pattern generatorn).
4	
Bit 0-3	Spritens färg
Bit 7	Om 1 så flyttas spriten 32 punkter åt vänster (används om en sprite ska komma ut bakom bildskärmens vänstra kant).

Sprite pattern tabellen bestämmer mönstret och byggs upp på i princip samma sätt som i Basic.
Nedanstående exempel demonstrerar hur man kan använda sprites i maskinspråk.

```
100 SCREEN 1
110 COLOR 1,4,1
120 FOR I%=0 TO 7
130   READ A$
140   VPOKE 14336+I%,VAL("&B"+A$)
150 NEXT I%
160 VPOKE 6914,0
170 VPOKE 6915,15
180 X%=0 : Y%=0 : X1%=1 : Y1%=1
190   X%=X%+X1% : IF X%=0 OR X%=248 THEN X1%=-X1% : BEEP
210   Y%=Y%+Y1% : IF Y%=0 OR Y%=184 THEN Y1%=-Y1% : BEEP
230   VPOKE 6912,Y% : VPOKE 6913,X%
240   GOTO 190
250 DATA 11111111
260 DATA 11000011
270 DATA 10100101
280 DATA 10011001
290 DATA 10011001
300 DATA 10100101
310 DATA 11000011
320 DATA 11111111
```

Rad 120-150 POKE:ar in spritens form på adress 14336-14343. Namnet den då får blir 0 eftersom spriten ligger först i Sprite Pattern Generatorn. Spriten på adress 14336+1*8 skulle fått namnet 1, 14336+2*8 skulle fått namnet 2 o.s.v.

I sprite attribute table ligger beskrivningen till första spriten på adress 6912-6915, 2:a på 6916-6919 o.s.v.
Rad 160 sätter namnet till 0
Rad 170 sätter färgen till vit
Rad 230 POKE:ar in spritens koordinater.

VDP:n ligger på följande I/O adresser

```
80H  Write     Mode=0
81H  Write     Mode=1
84H  Read      Mode=0
85H  Read      Mode=1
```

Operation	Bit 7	6	5	4	3	2	1	0	Mode
Skriv till VDP register									
Byte 1 Data write	D7	D6	D5	D4	D3	D2	D1	D0	1
Byte 2 Register select	1	0	0	0	0	RS2	RS1	RS0	1
Skriv till VRAM									
Byte 1 adress setup	A7	A6	A5	A4	A3	A2	A1	A0	1
Byte 2 adress setup	0	1	A13	A12	A11	A10	A9	A8	1
Byte 3 data write	D7	D6	D5	D4	D3	D2	D1	D0	0
Läs från VDP register									
Byte 1 data read	D7	D6	D5	D4	D3	D2	D1	D0	1
Läs från VRAM									
Byte 1 adress setup	A7	A6	A5	A4	A3	A2	A1	A0	1
Byte 2 adress setup	0	0	A13	A12	A11	A10	A9	A8	1
Byte 3 data read	D7	D6	D5	D4	D3	D2	D1	D0	0

Exempel på ett maskinspråksprogram som använder videominnet.

Detta program kan placeras var som helst i minnet eftersom det inte innehåller några absoluta adresser utan bara relativa.

```
E5        PUSH HL
F5        PUSH AF
3E0A      LD   A,0AH     ; videoramadressens lägsta halva
D381      OUT  (81H),A
DE40      LD   A,40H
D381      OUT  (81H),A
210004    LD   HL,1024
3E0F      LD   A,0FH
D380      OUT  (80H),A
AF        XOR  A
DDE2      EX   (SP),IX
DDE2      EX   (SP),IX
3D        DEC  A
20F9      JR   NZ,-5
7C        LD   A,H
B5        OR   L
20F0      JR   NZ,-14
F1        POP  AF
E1        POP  HL
C9        RET
```

Exempel på hur Basic-program ligge i RAM-minnet
==========

Nedanstående program ligger i SV328:ans minne.

```
10 INPUT A$
20 A%=VAL("&H"+A$)
30 LPRINT HEX$(A%)"  ";
40 FOR I%=0 TO 7
50   CH$=HEX$(PEEK(A%+I%))
55   IF LEN(CH$)<2 THEN CH$="0"+CH$
56   LPRINT CH$ " ";
60 NEXT I%
70 LPRINT"  ";
80 FOR I%=0 TO 7
90   CH%=PEEK(A%+I%)
100  IF CH%<32 OR CH%>127 THEN LPRINT"."; ELSE LPRINT CHR$(CH%);
101 NEXT I%
120 LPRINT
130 A$=INKEY$
140 IF LEN(A$)=0 THEN GOTO 130
150 A%=A%+8
160 IF A$<>CHR$(13) GOTO 30
170 GOTO 10
```

Så här ser det ut om du listar minnesinnehållet i HEX-kod

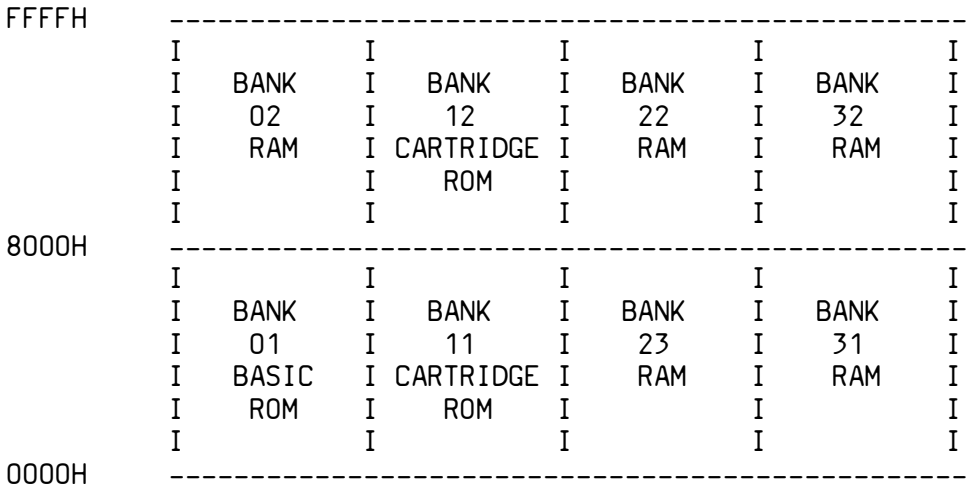
```
8000 00 0A 80 0A 00 85 20 41   ...... A
8008 24 00 1D 80 14 00 41 25   $.....A%
8010 F1 FF 94 28 22 26 48 22   ...("&H"
8018 F3 41 24 29 00 2F 80 1E   .A$)./..
8020 00 9D 20 FF 9B 28 41 25   .. ..(A%
8028 29 22 20 20 22 2B 00 3E   )"  ";.>
8030 80 28 00 82 20 49 25 F1   .(.. I%.
8038 11 20 D9 20 18 00 56 80   . . ..V.
8040 32 00 20 20 43 48 24 F1   2.  CH$.
8048 FF 9B 28 FF 97 28 41 25   ..(..(A%
8050 F3 49 25 29 29 00 76 80   .I%)).v.
8058 37 00 20 20 8B 20 FF 92   7.  . ..
8060 28 43 48 24 29 F2 13 20   (CH$)..
8068 DA 20 43 48 24 F1 22 30   . CH$."0
8070 22 F3 43 48 24 00 87 80   ".CH$...
8078 38 00 20 20 9D 20 43 48   8.  . CH
8080 24 20 22 20 22 2B 00 90   $ " ";..
8088 80 3C 00 83 20 49 25 00   .<.. I%.
8090 9B 80 46 00 9D 22 20 20   ..F.."
8098 22 3B 00 AA 80 50 00 82   ";...P..
80A0 20 49 25 F1 11 20 D9 20    I%.. .
80A8 18 00 BE 80 5A 00 20 20   ....Z.
80B0 43 48 25 F1 FF 97 28 41   CH%...(A
80B8 25 F3 49 25 29 00 EB 80   %.I%)...
80C0 64 00 20 8B 20 43 48 25   d. . CH%
80C8 F2 0F 20 20 F9 20 43 48   ..  . CH
80D0 25 F0 0F 7F 20 DA 20 9D   %.. . .
80D8 22 2E 22 3B 20 3A A1 20   "."; :.
80E0 9D 20 FF 96 28 43 48 25   . ..(CH%
80E8 29 3B 00 F4 80 65 00 83   );...e..
80F0 20 49 25 00 FA 80 78 00    I%...x.
80F8 9D 00 03 81 82 00 41 24   ......A$
8100 F1 EC 00 18 81 8C 00 8B   ........
8108 20 FF 92 28 41 24 29 F1    ..(A$).
8110 11 20 89 20 0E 82 00 00   . . ....
8118 24 81 96 00 41 24 F1 41   $...A%.A
8120 25 F3 19 00 3B 81 A0 00   %...;...
8128 8B 20 41 24 F2 F0 FF 96   . A$....
8130 28 0F 0D 29 20 89 20 0E   (..) . .
8138 1E 00 00 45 81 AA 00 89   ...E....
8140 20 0E 0A 00 00 00 00 0A    .......
8148 9D D3 02 43 48 00 00 03   ...CH...
8150 00 8B 20 3A 20 8F 20 4E   .. : . N
8158 4F 20 49 4E 54 45 52 4C   O INTERL
```

Här kommer en förklaring till hur det medföljande BASIC-
programmet är upplagt i RAM-minnet

Pekar-namn	Pekar-adress (HEX)	Minnes-adress (HEX)	Minnes-innehåll (HEX)	Kommentar
RAMLOW	F500	8000	00	Alltid noll
TXTTAB	F54A	8001	0A	Minnespekare till nästa rad
		8002	80	(800AH)
		8003	0A	Radnummer (000AH = 10)
		8004	00	
		8005	85	Token för "INPUT"
		8006	20	Mellanslag
		8007	41	"A"
		8008	24	"$"
		8009	00	0=slut på rad
		800A	1D	Början på rad 20, pekare till
		800B	80	rad 30
		.....		
		813B	45	Pekare till 8145H
		813C	81	
		813D	AA	Radnummer 170
		813E	00	
		813F	89	Token för "GOTO"
		8140	20	" "
		8141	0E	
		8142	0A	Hopp till 10
		8143	00	
		8144	00	Slut på rad
		8145	00	Här skulle nästa rad legat men
		8146	00	0000 betyder programslut
VARTAB	F7EE	8147	03	03=Strängvariabel
		8148	41	Variabelnamn "A"
		8149	00	00=Inget andra tecken
		814A	01	Strängen består av ett tecken
		814B	FB	Pekare till sträng (D2FBH)
		814C	D2	
		814D	02	02=Heltalsvariabel
		814E	41	Variabelnamn "A"
		814F	00	Inget andra tecken
		8150	50	Värdet av heltalsvariabeln
		8151	81	(2 byte)
		.....		
		8157	03	Strängvariabel
		8158	43	Variabelnamn "CH"
		8159	48	
		.....		

Pekar-namn	Pekar-adress (HEX)	Minnes-adress (HEX)	Minnes-innehåll (HEX)	Kommentar
ARYTAB	F7F0	8162		Eftersom ARYTAB=STREND finns
STREND	F7F2			inga indexerade variabler
SAVSTK	F7DD	D2BD		Här är stackens botten
STKTOP	F546	D2D8		Här börjar stränglagringsarean
MEMSIZ	F7A2	D3A0		Början på filbufferterna. För närvarande finns det plats för en fil.
HIMEM	FDE6	D5B8		2:a parametern i CLEAR-kommandot

Minnesuppbyggnad i Spectravideo
==========



Minnet i Spectravideo SV-318/SV-328 är organiserat som 8 st minnesbanker. Varje minnesbank har en storlek på 32K. Utöver dessa minnesbanker finns ett separat 16K minne för bildskärms-hanteringen. Detta betyder att Spectravideo kan hantera max 176K RAM och 96K ROM.

Bank 01 består av ett ROM-minne och innehåller Microsoft Extended Basic. Detta minne är standard.

Bank 02 består av ett RAM-minne. Detta ingår som standard i SV-328. I detta minne finns alla systemparametrar som tar 3K. Bank 02 används för ega program under Basic. Programmet kan vara max 29K stort. Vid optimal programmering bör man använda denna bank endast för kod, d.v.s. ej för datavariabler.

Bank 11 består av ett ROM-minne. Detta ingår ej som standard. Bank 11 används för cartridgemoduler som t.ex. spel-program.

Bank 12 samma som bank 11.

Bank 13 består av ett RAM-minne. Detta ingår som standard i SV-328. Man kan komma åt detta minne under Basic med ett program som beskrivs i detalj på sidan 94. Det medföljer också din Basic-disk eller kan kopieras upp hos din återförsäljare.
Bank 21 kan under Basic endast användas som dataarea. Det betyder att man bör lägga in alla variabler och tabeller här.
Dessutom används Bank 21 under CP/M.

Bank 22 består av ett RAM-minne. Detta ingår ej som standard utan man måste expandera med ett 64K-kort (SV-807) för att få tillgång till det*. För att komma åt Bank 22 används SWITCH-kommandot. Bank 22 kan användas för programkod som kan exekveras separat, d.v.s. utöver programmet som finns i Bank 02. Detta betyder att man kan skriva två program som utnyttjar 29K (21K om Disk Basic version används) i Bank 02 och 29K (21K om Disk Basic version används) i Bank 22, d.v.s. 58K totalt (42K om Disk Basic version används).
Exemplet nedan visar hur två program kan arbeta ihop.

```
10 REM program i bank 02        10 REM proram i bank 22
20 FOR I=0 TO 9                 20 PRINT "BANK 22"
30 PRINT "BANK 02"              30 SWITCH
40 SWITCH                       40 GOTO 20
50 NEXT I
```

```
1 Mata in det vänstra programmet
2 Skriv SWITCH
3 Mata in det högra programmet
4 Skriv SWITCH
5 Skriv RUN
6 Skriv RUN
6 Skriv RUN en gång till
```

Du ser nu hur exekveringen hoppar mellan Bank 02 och Bank 22. På nästa sida visas hur du kan koppla ihop två program så att de fungerar som ett. Observera att om du använder 80-kolumnerskortet måste detta initieras efter först SWITCH. Ska du stanna både programmen får du först trycka CTRL-STOP och därefter mata in SWITCH STOP.

Bank 31 består av ett RAM-minne. Detta ingår ej som standard utan man måste expandera med ett 64K-kort (SV-807) för att få tillgång till det*. Du kommer åt Bank 31 genom programmet på sid 94. Programmet behöver justeras för Bank 31. Se beskrivning i programmet. Bank 31 används på samma sätt som bank 21, d.v.s. som dataarea.

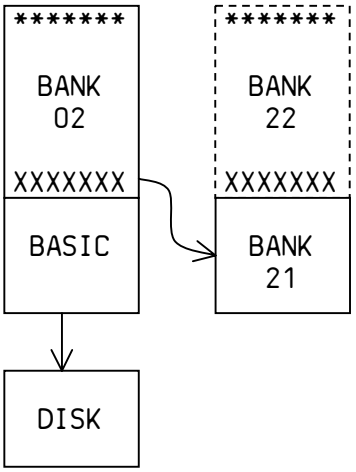
Bank 32 är avsedd för CP/M 3.0 och kan ej på ett enkelt sätt användas från Basic.

* Vill du bygga ut minnet på en SV-328 (eller utbyggd SV-318) så kopplar du inte ett 64K-kort och ställer omkopplarna BK22 och BK31 åt vänster, övriga åt höger.
Minnet i en SV-318 kan byggas ut med ett 64K-kort. Vill du ha samma minnesuppsättning som på en SV-328 så ställer du omkopplarna BK21 och BK02 på 64K-kortet åt vänster och övriga åt höger.
En SV-318 behöver alltså två 64K-kort och en SV-328 behöver ett 64K-kort för att utnyttja BK02, BK21, BK22 och BK31.

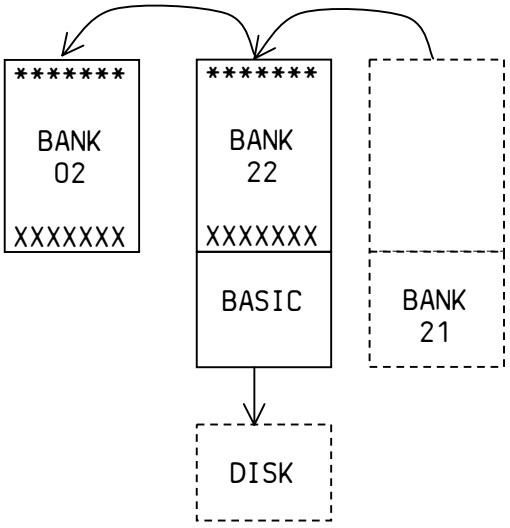
Så här fungerar switchningen mellan minnesbank 02 och 22

******* Här ligger systemvariablerna och eventuellt disk-
rutinerna för Basic.

XXXXXXX Här ligger eventuellt programmet för att komma åt
Bank 21 och Bank 31.
Se beskrivning längre fram angående användning
av Bank 21 och Bank 31.
Tänk på att om du ska använda Bank 21 måste du initiera
programmet både i Bank 02 och Bank 22.



1.
Spara de parametrar (variabler)
som du behöver i den andra
delen av programmet.
Parametrarna kan sparas på disk
eller i Bank 21 (eller 31).



2.
Kör SWITCH.
Då kopplar du ur Bank 02 och
kopplar in Bank 22.

Beskrivning av program för att komma åt 32 K RAM i SV-328

Det medföljande maskinspråksprogrammet innehåller tre användardefinierade rutiner.

1	Hämta en variabel i minnet	USR0
2	Spara en variabel i minnet	USR1
3	Sätta adresspekaren	USR2

När du ska använda minnet gör du på följande sätt:

a Sätt adresspekaren var i minnet du vill lagra variabeln.
Minnet ligger på adress 0-32767
Ex. B%=USR2(100) eller B%=USR2(A%).
Adressen (värdet inom parantes) måste vara en heltalsvariabel eller konstant, alltså ej t.ex. A, A# eller 100!.
Heltalsvariabeln till vänster, i detta fall B%, är en slask-variabel för syntaxens skull.

b Spara variabel B#, B! och A$ i minnet på följande sätt:
S#=USR1(B#), S!=USR1(B!) eller S#=USR1(A$).
Dessa variabler sparas med början på adress enligt adresspekaren.
Dessutom ökas adresspekaren enligt följande regler:

Heltal	2 steg
Enkel precision	5 steg
Dubbel "	9 steg
Strängar	Strängens längd (Ex. "ABC" 3 steg)

Detta innebär att du kan lagra så många variabler i följd som du önskar så länge minnet inte tar slut eller du kolliderar med annat minnesinnehåll. Variabeln till vänster, i detta fall S, är en slaskvariabel som ska vara av samma typ som den inom parantesen.

c När du ska hämta variablerna ur minnet måste du först sätta adresspekaren på den adressen där du sparade variabeln. Du måste alltså själv hålla reda på var i minnet du har sparat variabeln. Det innebär att du är mest fördelaktigt att lagra indexerade variabler eftersom det är lätt att räkna ut adressen.

Därefter kan du hämta variablerna genom USR0.
Ex. B#=USR0(0#), B!=USR0(0!) eller B$=USR0(" ")
Innehållet i minnet hamnar i B#, B! resp B$. Värdet inom parantes måste vara en variabel eller konstant av samma typ som variabeln som ska hämtas ur minnet.
Adresspekaren ökas enligt samma regler som vid lagring så du kan hämta flera variabler i en följd.

Kompletterande sidor:

Grafisk beskrivning hur programmet fungerar sid 96.

Initiering av maskinspråksprogram sid 98.
Kör in detta program en gång och maskinspråksprogrammet ligger inne så länge datorn är påslagen.

Programexempel i BASIC sid 99.
I ditt eget program behöver du bara ha med de tre första raderna. Resten är exempel.

Grafisk förklaring hur variablerna lagras i RAM-minnet sid 100.
Visar hur det medföljande BASIC-exemplet lagrar variablerna. Jämför.

Källkodslistningen till maskinspråksprogrammet sid 101.
Denna listning är huvudsakligen avsedd för de som kan programmera i maskinspråk.

Så här fungerar maskinspråksprogrammet som skriver och läser i Bank 21 och Bank 31 i princip

Åtkomst av joysticksignalerna från maskinspråk

Joystickutgångarna är kopplade till A-porten på ljudkretsen AY-3-8910 samt till A-porten på parallell In/Ut kretsen 8255.

Joystickarnas riktning ligger på ljudkretsens A-port (se ritning) som du kommer åt på följande sätt:

Mata ut 14 på port 88H. Då väljer du att använda A-porten på ljudkretsen.

Ta sedan in värdet på adress 90H. Då ligger joystick 1's riktningar som de 4 lägsta bitarna och joystick 2's riktningar som de 4 högsta bitarna. Riktningarna ligger med framåt som den lägsta biten, därefter bakåt, vänster och höger.

Exempel i Basic:

```
10 OUT &H88,14     ' välj joystickport
20 A%=INP(&H90)    ' ta in riktningar
30 PRINT BIN$(A%)  ' skriv ut riktningar
40 GOTO 10
```

Provkör programmet och prova med olika joystickriktningar. Samma exempel i maskinspråk (Z80 mnemonics) ser ut på följande sätt:

```
3E0E      LD   A,14
D388      OUT  (88H),A
DB90      IN   A,(90H)
```

Joystickarnas värde ligger sedan i A-registret och kan testas på normalt sätt.

Trigger-signalerna från joystickarna ligger på 8255:ans A-port (se ritningar). Joystick 1's triggersignal ligger på bit 4 och joystick 2's triggersignal ligger på bit 5.
Se följande Basic-exempel hur lätt dessa plockas in.

```
10 A%=INP(&H98)
20 A%=A% AND &B00110000 ' maska bort alla signaler utom trigger
30 PRINT A%
40 GOTO 10
```

Provkör programmet och testa att trycka ner trigger-knappen.

Ett motsvarande program i maskinspråk kan se ut på följande sätt:

```
DB98      IN   A,(98H)
E630      AND  00110000B
```

Dokumentation för Spectravideo Monitor

Av Andrzej Felzcak 840415

Det finns två monitorprogram på kassetten.
spvmon.318 och spvmon.328. Det ena är avsett för SV-318 och det andra SV-328.
Du initierar monitorn genom att mata in bload"spvmon.318",r eller bload"spvmon.328",r.

Då kommer en initieringstext upp. Monitorn hamnar sedan längst ner i RAM-minnet i datorn (adress C000H i SV-318 och 8000H i SV-328). Monitorn flyttar därefter upp "golvet" för basicprogram och rensar basic-minnesarean. Eventuella program som låg inne raderas.

Kör du list syns ingenting i minnet. Monitorn ligger dold under basic-arean och kan inte listas på normalt sätt. Du ser dock att den finns genom att ge kommandot "? FRE(0)".
Då upptäcker du att det finns ca 1.5K mindre arbetsminne tillgängligt. Det är utrymmet som monitorn upptar.

Du kan nu ladda och köra basic-program som vanligt. Du får dock tänka på två saker:
Du har mindre arbetsminne tillgängligt och vissa program får kanske inte plats.
Programmet som kommer åt det 32K "dolda" minnet i SV-328 ligger på samma adress som monitorn och måste flyttas för att kunna användas.

Monitorn går man in med kommandot "mon".
Monitorn svarar med ">" och väntar på kommando.

Kommandolista

Alla kommandorader avslutas med ENTER. Tal inom parantes behöver inte matas in om man vill använda de förutbestämda värdena. De förutbestämda värdena anges normalt inom parantes i kommandobeskrivningen.
Alla tal anges hexadecimalt. Tal kan förkortas om så önskas (Ex. 0000 kan anges som 0).

ÜB

Åter Basic

A-SSSSSSS...
(xxxx) (yyyy) Find ASCII values

Letar efter teckensträngen SSSSSSS... som kan vara upp till 22 tecken lång. Börjar leta på adress xxxx (0000) och letar t.o.m. adress yyyy (FFFF). "-" skrivs ut av datorn.
Utskriften stannas och startas tillfälligt med någon tangent.
ÜC avbryter utskriften.

Bxxxx Set BREAKpoint

Lägger in en CALL-instruktion på 3 byte till break-rutinen. CALL-instruktionen hamnar på adress xxxx.

Cxxxx Enter & substitute CHARACTERS

Ändrar i minnet i ASCII-format med början på adress xxxx. ENTER ökar adressen, BACKSPACE minskar avbrytes med ÜC.

Dxxxx (yyyy) DISPLAY memory

Visar minnesinnehållet i HEX och ASCII-kod med början på adress xxxx t.o.m. adress yyyy. Anges inte adress yyyy visas 128 tecken (ungefär en skärm). Ascii-koder mindre än 32 "Space" ersätts med punkt.
Utskriften stannas och startas tillfälligt genom någon tangent.
ÜC avbryter utskriften.

Fxxxx yyyy zz FILL memory with constant value

Fyller minnesarean fr.o.m. xxxx t.o.m. yyyy med koden zz.

G GO execute

Börjar exekvering av program med registerinnehåll enligt specifikationen i X-kommandot.
Trycks någon annan tangent än ENTER efter G avbryts kommandot.

H-aa bb cc... Find HEX values
(xxxx) (yyyy)

Letar efter talföljden aa bb cc... som kan vara upp till 22 tal lång. Börjar att leta på adress xxxx (0000) och letar t.o.m. adress yyyy (FFFF). "-" skrivs ut av datorn.
Utskriften stannas och startas tillfälligt med någon tangent.
ÜC avbryter utskriften.

Mxxxx yyyy zzzz MOVE memory area

Flyttar minnesarean mellan xxxx och yyyy till en ny plats med början på zzzz.

R(xxxx) READ from cassette

Laddar program från kassett med offset xxxx. Anges inte xxxx placeras programmet där det sparades ifrån.
Ex. Programmet sparades på 9000H, offset är 100H. Då laddas programmet på 9100H.

Sxxxx SUBSTITUE memory values

Ändrar i minnet hexadecimalt en byte i taget med början på adress xxxx. ENTER ökar adressen, SPACE minskar adressen och ÜC avbryter.

Wxxxx yyyy WRITE on cassette

Sparar minnesarean fr.o.m. xxxx t.o.m. yyyy på kassett.

X EXAMINE registers

Visar innehållet i registrena.

X'

Visar innehållet i Z80:ns andra uppsättning register samt IX och IY registrena.

Xreg eller X'reg

Skriver ut innehållet i registret reg och innehållet kan därefter ändras.

Register beteckningar

A	Ackumulator
B	BC-registret
D	DE-registret
H	HL-registret
F	Flaggregistret
S	Stackpekaren
P	Programräknaren
X	Indexregister IX
Y	Indexregister IY

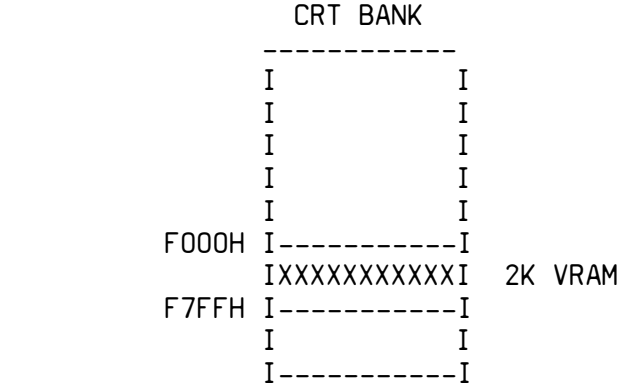
Specification for 80 column cartridge
=====================================

Character: 5x7 dot matrix

Characters (Column & Row) total: 80 by 24 (1920)
Display characters total: 176 (128 ASCII characters - 48 graphic characters)

Graphic: SV-318/SV-328 graphic character comparative
Video output: Composite video 2Vp-p

Memory map



I/O Port:
- 50H — Adress register of 6845 CRT controller
- 51H — CRT register of 6845 CRT controller
- 0F000H — Video RAM starting address
- 58H — CRT bank control (D0 = 1 -> Bank ON, D0 = 0 -> Bank OFF)

Dokumentation RS-232 Interface
==============================

Lista över portarna i RS-232 Interfacet

DLAB	Adress	R/W	Register
0	28H	R	Receiver Buffer
0	28H	W	Transmitter Holding Register
1	28H	R/W	Divisor Latch (least significant byte)
0	28H	R/W	Interrupt Enable
1	29H	R/W	Divisor Latch (most significant byte)
x	2AH	R	Interrupt Identification
x	2BH	R/W	Line Control Register
x	2CH	R/W	Modem Control Register
x	2DH	R/W	Line Status Register
x	2EH	R/W	Modem Status Register

Input Signals

Receiver Clock (RCLK), Pin 9: This input is the 16x baud rate clock for the receiver section of the chip.

Serial Input (SIN), Pin 10: Serial data input from the communications link (peripheral device, Modem or data set).

Clear to Send ($\overline{CTS}$), Pin 36: The $\overline{CTS}$ signal is a Modem control function input whose condition can be tested bu the CPU by reading bit 4 (CTS) of the Modem Status Register. Bit 0 (DCTS) of the Modem Status Register indicates whether the CTS input has changed state since the previous reading of the Modem Status Register.

Note: Whenever the CTS bit of the Modem Status Register changes state, an interrupt is generated if the Modem Status Interrupt is enabled.

Data Set Read ($\overline{DSR}$), Pin 37: When low, indicates that the Modem or data set is ready to establish the communications link and transfer data with the INS8250. The DSR signal is a Modem-control function input whose condition can be tested by the CPU by reading bit 5 (DSR) of the Modem Status Register. Bit 1 (DDSR) of the Modem Status Register indicates whether the DSR input has changed state since the previous reading of the Modem Status Register.

Note: Whenever the DSR bit of the Modem Status Register changes state, an interrupt is generated if the Modem Status Interrupt is enabled.

Received Line Signal Detect ($\overline{\text{RLSD}}$), Pin 38: When low, indicates that the data carrier has been detected by the Modem or data set. The RLSD signal is a Modem-control function input whose condition can be tested by the CPU by reading bit 7 (RLSD) of the Modem Status Register. Bit 3 (DRLSD) of the Modem Status Register indicates whether the RLSD input has changed state since the previous reading of the Modem Status Register.

Note: Whenever the RLSD bit of the Modem Status Register changes state, an interrupt is generated if the Modem Status Interrupt is enabled.

Output Signals

Data Terminal Ready ($\overline{\text{DTR}}$), Pin 33: When low, informs the Modem or data set that the INS8250 is ready to communicate. The DTR output signal can be set to an active low by programming bit 0 (DTR) of the Modem Control Register to a high level. The DTR signal is set high upon a Master Reset operation.

Request to Send ($\overline{\text{RTS}}$), Pin 32: When low, informs the Modem or data set that the INS8250 is ready to transmit data. The RTS output signal can be set to an active low by programming bit 1 (RTS) of the Modem Control Register. The RTS signal is set high upon a Master Reset operation.

Baud Out (BAUDOUT), Pin 15: 16x clock signal for the transmitter section of the INS8250. The clock rate is equal to the main reference oscillator devided by the specified divisor in the Baud Generator Divisor Latches. The BAUDOUT may also be used for the receiver section by tying this output to the RCLK input of the chip.

Interrupt (INTRPT), Pin 30: Goes high whenever any one of the following interrupt types has an active condition and is enabled via the IER: Receiver Error Flag, Receiver Data Available, Transmitter Holding Register Empty and Modem Status. The INTRPT signal is reset low upon the appropriate interrupt service or a Master Reset operation.

INS8250 Line Control Register

The system programmer specifies the format of the asynchronous data communications exchange via the Link Control Register. In addition to controlling the format, the programmer may retrieve the contents of the Line Control Register for inspection. This feature simplifies system programming and eliminates the need for separate storage in system memory of the line characteristics. The contents of the Line Control Register are described below.

Bit 0 and 1: These two bits specify the number of bits in each transmitted or received character. The encoding of bits 0 and 1 is as follows:

Bit 1	Bit 0	Word Length
0	0	5 Bits
0	1	6 Bits
1	0	7 Bits
1	1	8 Bits

Bit 2: This bit specifies the number of stop bits in each transmitted or received serial character. If bit 2 is a logic 0, 1 Stop bit is generated or checked in the transmit or receive data, respectively. If bit 2 is a logic 1 when a 5-bit word length is selected via bits 0 and 1, 1-1/2 Stop bits are generated or checked. If bit 2 is a logic 1 when either a 6, 7 or 8-bit word length is selected, 2 Stop bits are generated or checked.

Bit 3: This is the Parity Enable bit. When bit 3 is a logic 1, a Parity bit is generated (transmit data) or checked (receive data) between the last data word bit and Stop bit of the serial data. (The Parity bit is used to produce an even or odd number of 1s when the data word bits and the Parity bit are summed.)

Bit 4: This is the Even Parity Select bit. When bit 3 is a logic 1 and bit 4 is a logic 0, and odd number of logic 1s are transmitted or checked in the data words bits and Parity bit. When bit 3 is a logic 1 and bit 4 is a logic 1, an even number of bits is transmitted or checked.

Bit 5: This is the Stick Parity bit. When bit 3 is a logic 1 and bit 5 is a logic 1, the Parity bit is transmitted and then detected by the receiver as a logic 0 if bit 4 is a logic 1 or as a logic 1 if bit 4 is a logic 0.

Bit 6: This bit is the Set Break Control bit. When bit 6 is a logic 1, the serial ouput (SOUT) is foreced the Spacing (logic 0) state and remains there regardless of other transmitter activity. The set break is disabled by setting bit 6 to a logic 0. This feature enables the CPU to alert a terminal in a computer communications system.

Bit 7: This bit is the Divisor Latch Access Bit (DLAB). It must be set high (logic 1) to access the Divisor Latches of the Baud Rate Generator during a Read or Write operation. It must be set low (logic 0) to access the Receiver Buffer, the Transmitter Holding Register, or the Interrupt Enable Register.

INS8250 Programmable Baud Rate Generator

The INS8250 contains a programmable Baud Rate Generator that is capable of taking the clock input and dividing it by any divisor from 1 to (2Ü16 - 1). The output frequency of the Baud Generator is 16x the Baud rate. Two 8-bit latches store the divisor in a 16-bit binary format. These Divisor Latches must be loaded during initialization in order to insure desired operation of the Buad Rate Generator. Upon loading either of the Divisor Latches, a 16-bit Baud counter is immediately loaded. This previous long counts on initial load.

Line Status Register

This 8-bit register provides status information to the CPU concerning the data transfer. The contents of the Line Status Register are described below.

Bit 0: This bit is the receiver Data Ready (DR) indicator. Bit 0 is set to a logic 1 whenever a complete incoming character has been received and transferred into the Receiver Buffer Register. Bit 0 may be reset to a logic 0 either by the CPU reading the data in the Receiver Buffer Register or by writing a logic 0 into it from the CPU.

Bit 1: This bit is the Overruin Error (OE) indicator. Bit 1 indicates that data in the Receiver Buffer Register was not read by the CPU before the next character was transferred into the Receiver Buffer Register, thereby destroying the previous character. The OE indicator is reset whenever the CPU reads the contents of the Line Status Register.

Bit 2: This bit is the Parity Error (PE) indicator. Bit 2 indicates that the received data character does not have the correct even or odd parity, as selected by the even-parity-select bit. The PE bit is set to a logic 0 whenever the CPU reads the contents of the Lines Status Register.

SVI-805 RS-232 Interface