SPECTRAVIDEO™

Andersson Kullbjer

MÄTNING STYRNING REGLERING

Liber

Mätning—styrning—reglering
med
SPECTRAVIDEO SV-318 och SV-328

Anders Andersson • Arne Kullbjer

Liber Läromedel

Förord

Ett viktigt användningsområde för datorer är för kontroll och styrning av processer och för automatisk insamling av mätvärden.

Denna bok är avsedd att ge grundläggande kunskaper om mätning och styrning med datorer och att använda SV-318 och SV-328 i några processer.

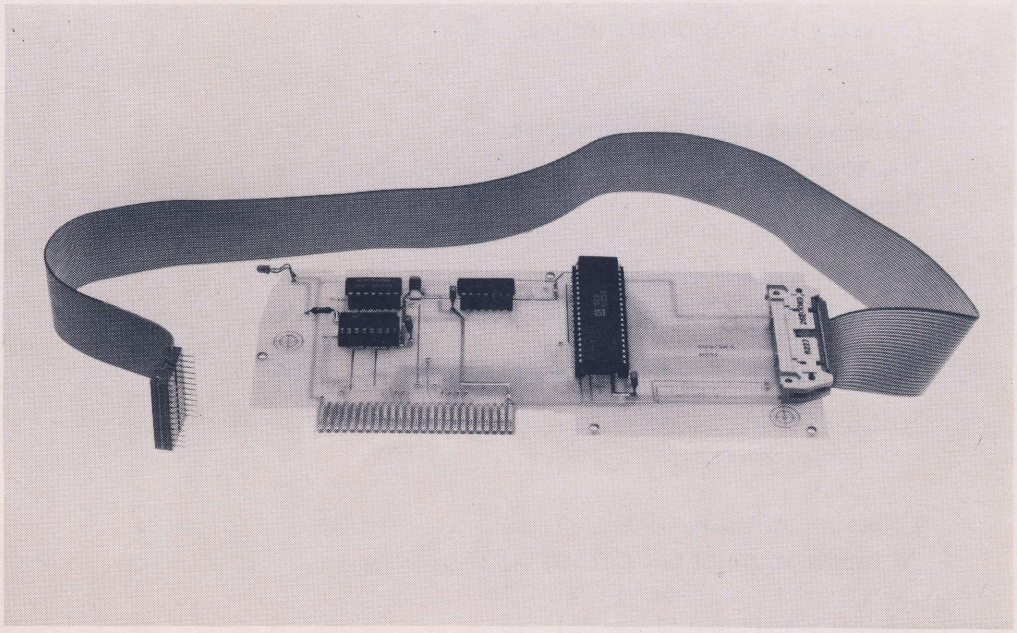
För att kunna tillgodogöra sig innehållet i boken krävs grundläggande kunskaper i BASIC-programmering och för några experiment viss kännedom om filhantering. Dessutom är det bra med kännedom om digitalteknikens grunder.

Boken är granskad, bearbetad och producerad av Lennart Bergström, Computer Press Förlags AB.

För experimenten i denna bok krävs ett parallellt in/ut-kort, (6522 VIA, se foto nedan), som finns att köpa som tillbehör hos Spectravideoåterförsäljarna.

Linköping i juli 1984

Författarna



Parallellt in/ut-kort, 6522 VIA

Innehåll

1. Hur datorn fungerar

Introduktion

Hittills har du förmodligen använt din dator för beräkningar, för att lagra och redigera data och för att presentera resultat på bildskärmen eller printern. Det här är mycket vanliga tillämpningsområden för datorer. Uppskattningsvis utnyttjas åtminstone 90% av persondatorbeståndet för texthantering och beräkningar. Datorer har emellertid en annan viktig fördel, de är utmärkta redskap att använda för kontroll och styrning av processer och för automatisk insamling av mätvärden. SV-318 och SV-328 utgör definitivt inga undantag, de är tvärtom mycket lämliga även för sådana tillämpningar.

I denna bok ska vi dels titta på några grundläggande begrepp när det gäller mätning och styrning med datorer, dels handgripligen koppla in SV-318 eller SV-328 i några processer för att ge åtminstone en antydan om vilket förnämligt instrument vi har i vår hand.

Allra först måste vi emellertid ta en titt på hur en dator egentligen fungerar. Med kännedom om de grundläggande principerna är det sedan lätt att gå vidare och även förstå vilka begränsningar som finns och vilka metoder som står till buds för att kringgå dem.

Datorns funktion:

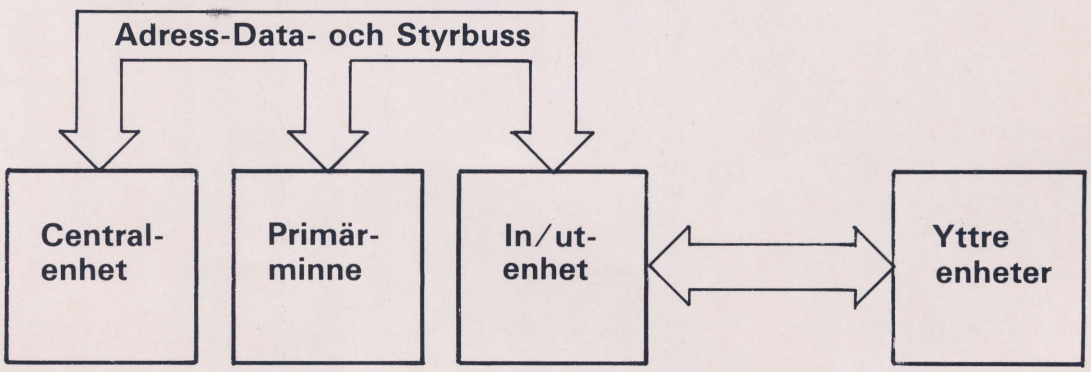


Fig 1.1 Blockschema för en dator

Figuren visar blockschemat för en dator. Det här blockschemat är mycket generellt, i princip är alla existerande datorer uppbyggda på det här sättet, alltså inte bara SV-318 och SV-328. Låt oss analysera funktionen!

Centralenheten utför beräkningar och administration av data från minne och yttre enheter.

Minnet innehåller dels program, som instruerar centralenheten om vad som ska göras, dels data. Din SV-318 eller SV-328 innehåller två olika typer av minnen, dels s.k. läsminnen eller ROM (eng. "Read Only Memory"), dels skriv/läsminnen eller RAM (eng. "Random Access Memory"). Båda typerna är halvledarminnen, d v s integrerade kretsar som består av tiotusentals transistorer på en liten kiselbricka, stor som en halv lillfingernagel.

Läsminnena är fast programmerade från fabriken och kan inte ändras! I SV-318 och SV-328 innehåller läsminnet i huvudsak två olika program, operativsystemet och Basictolken.

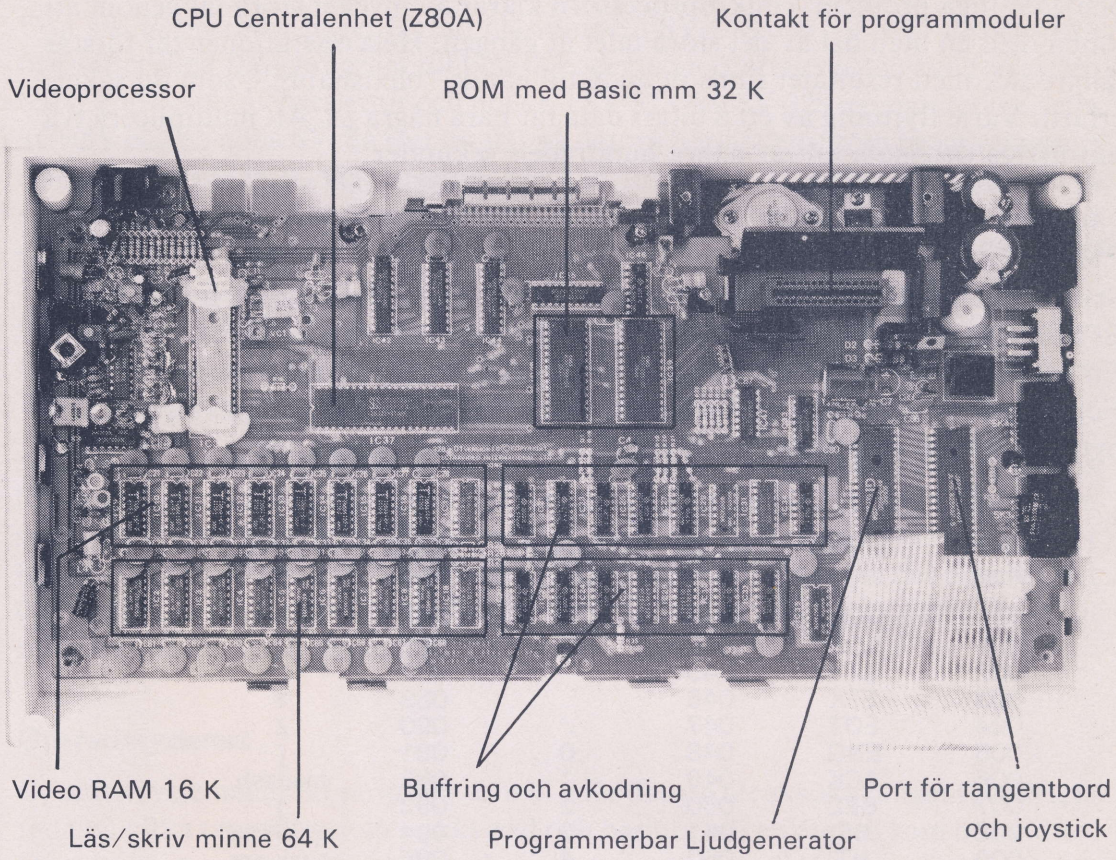


Fig. 1.2 Spectravideo SV-328 kretskort

Adressbussen

Låt oss gå tillbaka och titta på figur 1.1 ett ögonblick! De symboler som förbinder centralenhet, minne och in/utportar brukar i verkligheten kallas *bussar* och består av en mängd elektriska ledare som var och en kan överföra en etta eller en nolla. Man skiljer mellan adressbuss, databuss och styrbuss.

Adressbussen (som i SV-318 och SV-328 består av 16 ledningar) låter centralenheten välja vilken plats, adress i minnet eller vilken in/utport den avser att läsa eller skriva i. Man brukar säga att centralenheten *adresserar* minnet resp. in/ut-porten. De 16 ledningarna i adressbussen medger totalt $2^{16} = 65\,536$ olika adresser.

Databussen

Databussen består i SV-318 och SV-328 av 8 ledningar. Den används för att utbyta information mellan centralenheten å ena sidan och en adresserad minnesposition eller in/ut-port å andra sidan. De 8 dataledningarna kan överföra ett 8-siffrigt binärt tal åt gången. Datortekniker brukar kalla ett sådant åttasiffrigt binärt tal för "åtta bitar" eller "en byte" (utalas "bajt"). Ordet bit kommer egentligen från engelskans "binary digit", binär siffra. "Byte" är engelsk dataslang och betyder ordagrant "munfull". En byte är den mängd information som centralenheten, *mikroprocessorn* i SV-318 och SV-328 kan "tugga i sig" åt gången! Det passar ju verkligen utsökt för ASCII-koden – de 128 olika koderna går ju till och med att klämma in i bara 7 bitar ($2^7 = 128$), men hur i all världen kan datorn klara av

beräkningar med många siffror? Med 8 bitar finns det ju bara plats för tal mellan 0 och 255. Inga problem – mikroprocessorn klarar av mycket stora tal genom att jobba med en liten del av det stora talet åt gången. Hela operationen tar förstås längre tid, men resultatet finns ju ändå till hands "blixtsnabbt", som du säkert erfarit. Varje flyttning av ett 8 bitars data tar bara några μs. Att multiplicera två niosiffriga decimala tal tar några hundradels sekunder.

Styrbussen

Styrbussen består av ett tiotal ledningar, var och en med en speciell styrfunktion. Bl a finns en ledning som anger om mikroprocessorn ska skriva eller läsa i minnet. En ledning används vid start av datorn för att ge ett definierat starttillstånd osv. För en datorkonstruktör är styrbussen väsentlig, men vi har faktiskt ingen större nytta av någon detaljkännedom om den, så låt oss förbigå den.

ASCII Kod	Tecken	ASCII Kod	Tecken	ASCII Kod	Tecken
000	NULL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	bkslash
007	BEL	050	2	093	]
008	BS	051	3	094	up arrow
009	HT	052	4	095	back arr
010	LF	053	5	096	space
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	;
038	&	081	Q	124	<
039	'	082	R	125	=
040	(	083	S	126	>
041	)	084	T	127	DEL
042	*	085	U		

Fig. 1.3 ASCII-tabell

Basictolken

Den s k *Basictolken* ligger lagrad i läsminnet i SV-318 och SV-328. Basictolken översätter dina Basicprogram till binära koder, som centralenheten kan förstå. En binär kod består av bara ettor och nollor. I SV-318 och SV-328 representeras en etta av +5V och en nolla 0V. Så fort du slår på datorn börjar den utföra programavsnitt från operativsystemet. Via tangentbordet kan du nu mata in det Basicprogram du så småningom tänker låta datorn utföra. Varje tecken du matar in lagras i datorns skriv/läsminne (RAM) i kodad form som ettor och nollor. Den kod som används kallas ASCII-koden (eng *A*merican *S*tandard *C*ode for *I*nformation *I*nterchange), (se fig. 1.3). ASCII-koden har du förmodligen träffat på tidigare, åtminstone om du läst boken "BASIC-boken för Spectravideo", men då verkade ASCII-koden bestå av vanliga decimala tal, från 0—127, inte sant?

ASCII-koden för bokstaven A tycks vara 65 och 49 för siffran 1. Nåväl, gåtans lösning är som väl är ganska enkel. Datorn hanterar koderna ifråga i binär form som åttasiffriga binära tal. Bokstaven A, med ASCII-koden 65 lagras som det binära talet 01000001, siffran 1 med ASCII-koden 49 lagras som det binära talet 00110001 o s v. När du slår av spänningen till datorn försvinner omedelbart all information ur skriv/läsminnet!

Operativsystemet

Operativsystemet är ett program som bl a får datorn att avsöka tangentbordet, ta reda på vilka tangenter som trycks ned och besluta vilken åtgärd som bör vidtas. Operativsystemet gör förvisso mycket mera, men i huvudsak är operativsystemet ansvarigt för kommunikationen mellan tangentbordet, bildskärmen, flexskiveenheten, kassettbandspelaren, skrivaren och användarens program i datorn.

Portar

Ur centralenhetens synpunkt är det ingen som helst skillnad mellan skriv/läsminnet och in/ut-portar. Ur användarens synpunkt är det dock en fundamental skillnad. Information som centralenheten matar ut till en port i form av ett 8-bitars binärt tal är normalt åtkomlig på en kontakt och kan användas t ex för styrning av en yttre enhet. Den information som centralenheten matar ut till ett minne är däremot höljd i dunkel för alla utom centralenheten själv, eftersom det bara är centralenheten som kan läsa i minnet! På motsvarande sätt är det med en inport. Här kan användaren utifrån mata in en 8-bitars binär kod, som centralenheten sedan kan utnyttja genom att läsa i inporten.

Användarporten

I SV-318 och SV-328 finns ett antal portar, som bl a används för kommunikation mellan datorn och tangentbordet och för inkoppling av kassettbandspelare. Det finns ingen inbyggd användarport i vare sig SV-318 eller SV-328, d v s en port som du själv kan nyttja efter behag, t ex för styrning och mätning som den här boken ska handla om. Har du tillgång till en expanderlåda är emellertid detta inget problem. I en av kortplatserna i expanderlådan kan du sätta en speciell användarportmodul. Denna användarport möjliggör inkoppling av åtminstone 16 binära datakanaler, i form av två 8-bitars "portar", via ett kontaktdon i modulen. I fortsättningen förutsätter vi att du använder denna användarportmodul!

Användarporten består egentligen av en integrerad krets med typbeteckningen 6522, "VIA", (eng. Versatile Interface Adapter). I datorsammanhang utgör ett gränssnitt en anordning som gör att datorn kan anslutas till olika yttre enheter, t ex tangentbord, flexskivenhet, mätinstrument, etc.

Användarportens kontaktdon

De olika signalerna från porten är anslutna till kontaktdonet enligt fig 1.4. I nästa kapitel ska vi titta närmare på vad en del av de olika signalerna betyder och hur de kan användas.

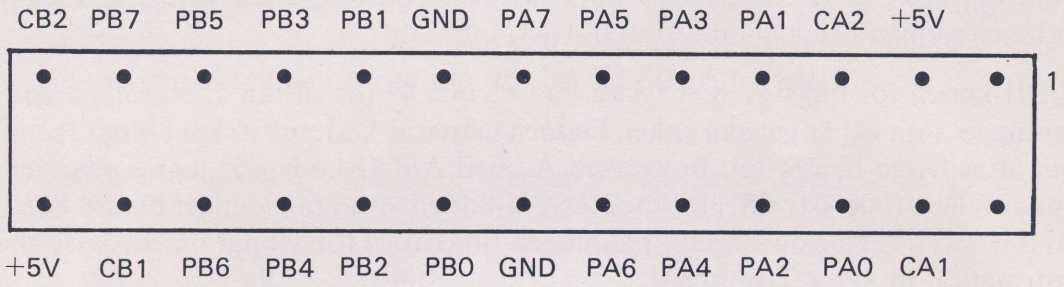


Fig. 1.4 Användarportens kontaktdon

Användarportens adress

Användarporten återfinns i in/ut-adresserna 0—15. I nästa kapitel ska vi titta närmare på hur porten kan ges information för att användas som in- eller utgång. Låt oss därför raskt gå vidare till nästa kapitel och se hur man gör.

2. Att utnyttja användarporten

Kontaktdon och kabel

För att kunna experimentera med användarporten behöver du en kabel med ett lämpligt kontaktdon och helst också ett s k kopplingsdäck, där du snabbt och enkelt kan göra små uppkopplingar. Den kabel som levereras med användarport-modulen är en flatkabel som i ena änden har en 26-polig s k "socket"-kontakt (ansluts till modulen) och i andra en 24-polig DIP-kontakt, som kan anslutas till kopplingsdäcket. I de följande experimenten utgår vi ifrån att du använder denna kabel.

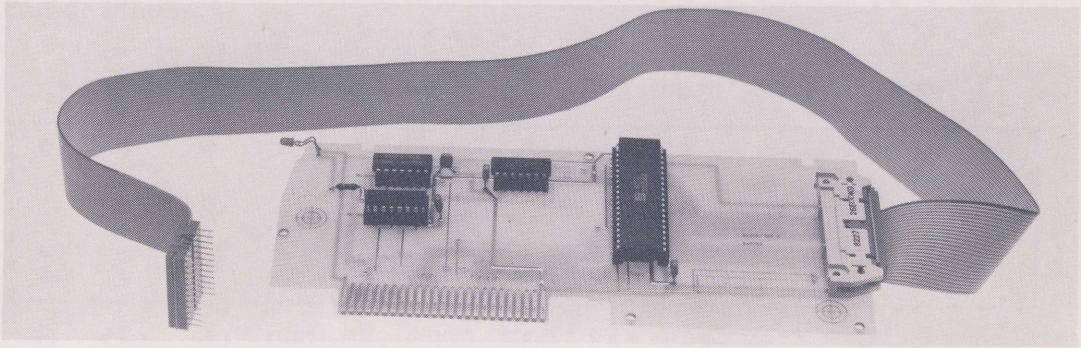


Fig. 2.1 Användarportmodul och kabel

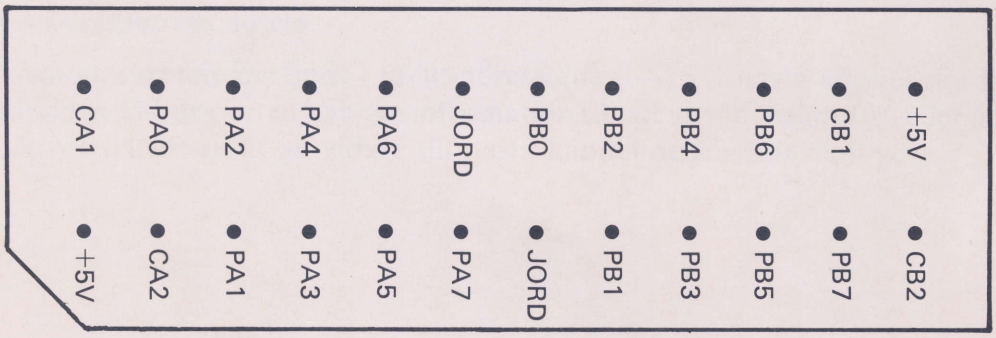


Fig 2.2 DIP-kontaktens koppling sedd uppifrån.

Binärkod

Vi börjar närma oss det första verkliga målet för det här kapitlet, nämligen att låta datorn påverka elektronik utanför själva datorn via användarporten. För att det hela ska bli meningsfullt, måste vi dock först fördjupa oss något i binära tal. Datorn jobbar ju internt med 8-bitars binära tal.

Vi människor har ju sedan barnsben dresserats i konsten att använda decimaltal (tal som innehåller tio siffersymboler, 0 – 9), så i början kanske det binära talsystemet med sina två siffersymboler (0 och 1) verkar förvirrande. Decimalsystemet sägs f ö ha *basen 10,* det binära systemet *basen 2.*

Hur kommer det sig egentligen att vi kan uttrycka godtyckligt stora (eller små) tal i decimalsystemet, trots att det bara finns tio siffersymboler? Hur vet vi att talet 186,37 betyder "etthundraåttiosex komma trettiosju"? Jo, knepet består i att siffrornas position i förhållande till decimalkommat tilldelar siffrorna olika vikt, beroende på läget. 186,37 är egentligen ett förkortat skrivsätt för

$1 \times 100 + 8 \times 10 + 6 \times 1 + 3 \times 1/10 + 7 \times 1/100$

En synnerligen användbar förkortning, inte sant? Matematiker skriver kanske hellre så här:

$1 \times 10^2 + 8 \times 10^1 + 6 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2}$

Du inser nu säkert varför vi säger att decimalsystemets bas är 10!

Det binära systemet är också ett sådant *positionssystem,* dvs siffrornas lägen anger deras vikt. I det binära fallet är förstås viktsfaktorn *2*, inte tio.

Det binära talet 1 0 1 1 0 1.0 1 är följaktligen ett förkortat skrivätt för

$1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$

eller om du så vill

$1 \times 32 + 0 \times 16 + 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 + 0 \times 1/2 + 1 \times 1/4$

Summerar du ihop dessa termer får du f ö den decimala motsvarigheten till det binära talet vi började med. Summan blir 45,25. Så nu vet du ett bra sätt att översätta ett givet binärt tal till motsvarande decimala tal.

Användarport-Modul	DIP-kontakt	Signal
1	1	+5V
2	24	CA1
3	2	CA2
4	23	PA0
5	3	PA1
6	22	PA2
7	4	PA3
8	21	PA4
9	5	PA5
10	20	PA6
11	6	PA7
12	19	JORD
13	7	JORD
14	18	PB0
15	8	PB1
16	17	PB2
17	9	PB3
18	16	PB4
19	10	PB5
20	15	PB6
21	11	PB7
22	14	CB1
23	12	CB2
24	13	+5V
25	—	—
26	—	—

Tabell 2.1 Signaler i gränssnitt

Den här bilden av ett godtyckligt 8-bitars binärt tal kommer väl till pass när du vill översätta från basen 2 till basen 10 (och vice versa).

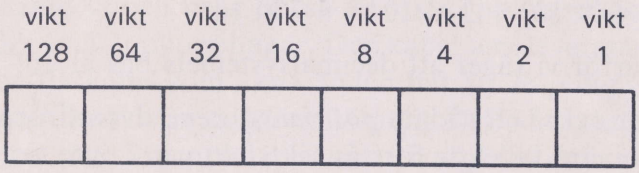


Fig. 2.3

Varje "box" kan innehålla en nolla eller en etta. $1\,0\,1\,0\,1\,1\,0\,0_2$ är alltså lika med 172_{10}.

Betecknar bas 10 betecknar bas 2

Åt andra hållet blir det lite värre, men det går! T ex 135_{10} kan delas upp i termerna $128 + 4 + 2 + 1$, d v s $135_{10} = 1\,0\,0\,0\,0\,1\,1\,1_2$

Som väl är finns det tabeller för omvandlingen – du hittar en sådan i appendix C.

Elementärt om in/utkretsen VIA

VIA-kretsen är en synnerligen generell in/ut-krets med mängder av möjligheter. För vårt vidkommande räcker det, som väl är, med en ytlig bekantskap och i de följande avsnitten får du veta tillräckligt mycket om dessa för att kunna börja experimentera med SV-318 eller SV-328 som "styrdator". Vill du på ett senare stadium veta mera bör du läsa igenom fabrikantens datablad över kretsen 6522 (VIA).

Register i in/ut-kretsarna

In/ut-kretsarna innehåller inte mindre än 16 olika 8-bitars register, alla åtkomliga för användaren. Ett register är ur datorns synpunkt lika med en minnesposition och centralenheten kan både läsa och skriva i vart och ett av de 16 registren. Varje register har en unik adress. Registren i den in/ut-krets som utgör användarporten har adresser i området 0—15 (I/O - adresser). Genom att användaren skriver lämpliga data i registren kan han kontrollera funktionen hos in/ut-kretsen in i minsta detalj. VIA:n är som sagt mycket generell. Den innehåller t ex två timers (för att mäta tid och antal) ett skiftregister (för att mata ut information seriellt, en bit åt gången), ett antal kvittenskanaler (för kontroll av signalflödet mellan datorn och t ex en plotter) och mycket, mycket mera. Registren styr och övervakar allt detta och mera därtill.

Vi behöver som väl är till att börja med inte bekymra oss om fler än två av de 16 registren. Det ena är själva användarporten, som är det register som utväxlar information med omvärlden. Det andra kallas *datariktningsregistret.* Varje in/ut-krets består egentligen av två halvor, kallade A och B. Det finns alltså två portar och två datariktningsregister. Den del vi ska använda är B-delen. Vi har följaktligen tillgång till port B och datariktningsregister B.

Port B har adressen 0 och datariktningsregister B har adressen 2. De åtta anslutningarna till port B kallas PB0 - PB7. Du har tillgång till dem på din kabels DIP-kontakt:

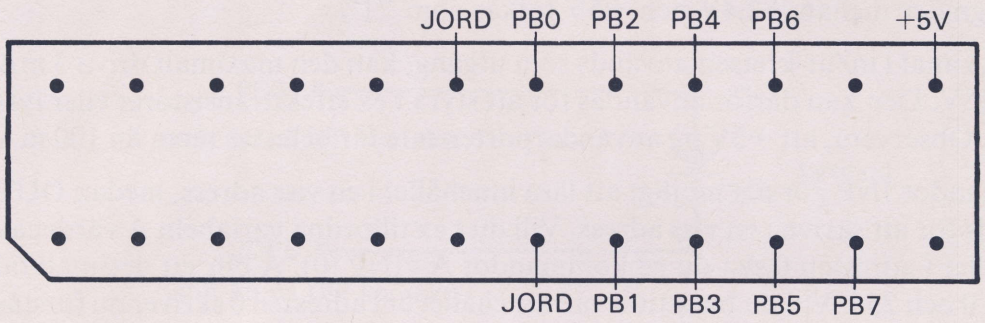


Fig. 2.4

Om du låter datorn mata ut information på användarporten (port B), finns signalerna tillgängliga på DIP-kontakten som "1" eller "0" (+5V eller 0V). PB0 är den minst signifikanta biten, PB7 den mest signifikanta. Eftersom bara 8 bitar är tillgängliga, är det största tal som kan matas ut $255_{10}=1\ 1\ 1\ 1\ 1\ 1\ 1\ 1_2$. T ex talet $135_{10} = 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1_2$ åstadkommer följande signalkombination på port B:

PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
+5V	0V	0V	0V	0V	+5V	+5V	+5V

Innan du provar måste du dock veta vad datariktningsregistret har för inverkan. Datariktningsregister B innehåller också 8 bitar. Varje bit kontrollerar en av de åtta datakanalerna i port B med avseende på riktning (d v s ingång eller utgång). Vill du att alla de åtta datakanalerna PB0–PB7 ska vara utgångar, ska du först skriva åtta stycken ettor i datariktningsregister B, dvs $1\ 1\ 1\ 1\ 1\ 1\ 1\ 1_2 = 255_{10}$. Om du istället vill att alla datakanalerna PB0–PB7 ska vara ingångar, skriver du åtta stycken nollor i datariktningsregister B, dvs $0\ 0\ 0\ 0\ 0\ 0\ 0\ 0_2 = 0_{10}$. Du kan blanda in- och utgångar efter behag. Vill du t ex att PB0 och PB1 ska vara utgångar och de övriga ingångar gör du så här:

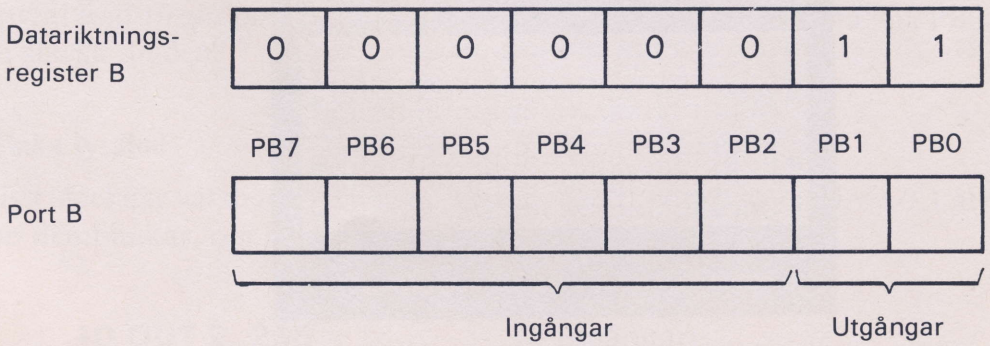


Fig. 2.5

Du skriver alltså talet $0\ 0\ 0\ 0\ 0\ 0\ 1\ 1_2 = 3_{10}$ i datariktningsregister B, d v s i adress 2. I nästa avsnitt ska vi se hur man bär sig åt för att skriva och läsa i registren, men först några varningens ord.

Restriktioner

Det är möjligt att förstöra in/ut-kretsen om inte fabrikantens krav på signaler följs. Tänk därför på att när en kanal i in/ut-kretsen används som ingång *måste* insignalerna ligga i området 0 till +5V. Signaler mellan 0V och + 0,8V tolkas som "0", signaler mellan +2,4V och +5V tolkas som "1".

När en kanal i in/ut-kretsen används som utgång, kan den maximalt driva 3 mA vid +1,5V. Den kan därför användas för att styra t ex effekttransistorer eller lysdioder. Observera, att +5V på användarporten inte får belastas mera än 100 mA.

Kommandot INP gör det möjligt att läsa innehållet i en viss adress, medan OUT används för att skriva i en viss adress. Vill du t ex tillordna variabeln A värdet av innehållet i adressen 0 ska du ge kommandot A=INP (0). A blir ett decimalt tal mellan 0 och 255. Vill du bara titta vad innehållet är i adressen 0 skriver du förstås PRINT INP (0) och trycker på returntangenten, så svarar datorn med det decimala innehållet i den angivna adressen. Det är förstås INP du använder för att hämta in data från användarporten för vidare behandling i datorn.

Om du vill skriva ett tal (mellan 1 och 255 decimalt) i en viss minnesadress, ska du använda kommandot OUT.

OUT 2, 255 skriver talet 255 (d v s 11111111) i adressen 2, vilket för övrigt är ett utmärkt sätt att tala om för VIA:n att port B ska vara åtta utgångar.

Med kommandot OUT 0,1 skrivs talet 1 (d v s 00000001_2) i adressen 0, d v s +5V läggs ut på användarportens kontakt PB0.

Låt oss nu göra ett par enkla experiment.

Några experiment

Vi föreslår, att du använder ett kopplingsdäck för att lätt kunna testa dina experimentuppkopplingar. Det finns många bra kopplingsdäck på marknaden t ex Protoboard PB6.

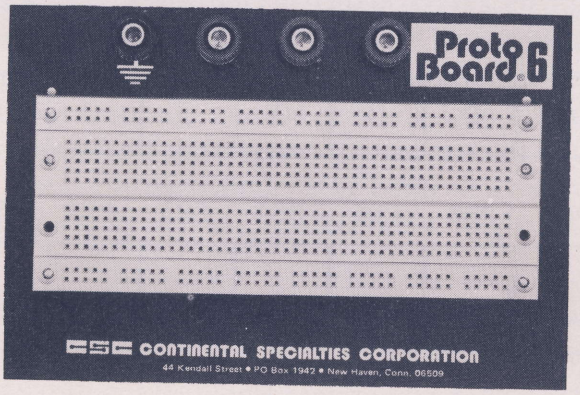


Fig. 2.6 Kopplingsdäck

För förbindningar använder du vanlig 0,6 mm PVC-isolerad enkelledare. Övriga komponenter (motstånd, kondensatorer, lysdioder, IC-kretsar) passar direkt i däcket, liksom DIP-kontakten från användarporten.

Tänd lysdiod

Koppla in en lysdiod mellan PB0 och jord. Lysdiodens katod ska kopplas till jord. Katodbenet brukar vara lite kortare än det andra, men eventuellt kan det istället vara en liten avfasning i plasten vid katodbenet.

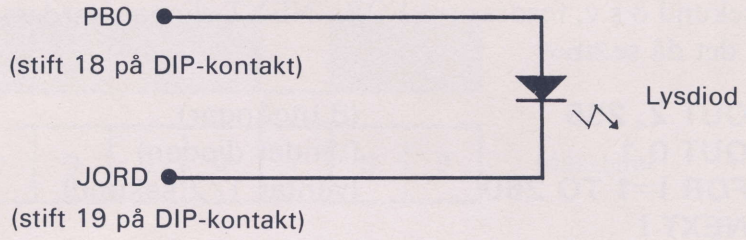


Fig. 2.7 Kopplingsschema

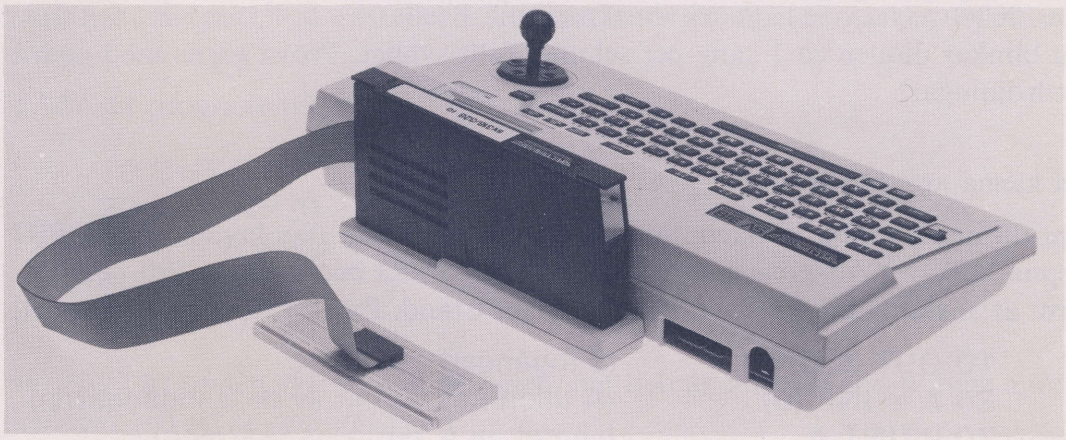


Fig. 2.8 Experimentuppkoppling

Följande program tänder lysdioden!

```
10 OUT 2, 255
20 OUT 0,1
```

Prova! Rad 10 gör port B till 8 utgångar. Rad 20 matar ut en "etta" på PB0. Kan du släcka dioden?

Blinka lysdiod

Visst slocknar väl dioden om man skriver talet 0 i port B, d v s adress 0. Låt oss se om den blinkar, om man gör en slinga i programmet nedan.

```
10 OUT 2, 255          (8 utgångar)
20 OUT 0,1             (tänder dioden)
30 OUT 0,0             (släcker dioden)
40 GOTO 10             (hoppar tillbaka och tänder igen)
```

Kör det här programmet. Du ser inte några blinkningar, eller hur? Orsaken är att det går på tok för fort. Det tar bara någon tusendels sekund för datorn att utföra

varje rad, d v s den blinkar ca 500 gånger/sek. Det mänskliga ögat förmår inte reagera så snabbt (det tar ungefär 1/30 sekund för ögat att reagera), så det ser ut som om dioden lyser kontinuerligt, fast lite svagare än förut.

Vill du verkligen se att dioden blinkar återstår bara att göra processen långsammare. Ett vanligt sätt är att använda en FOR...NEXT-slinga som fördröjning – varje varv i slingan tar ca 2 ms. Om du vill att dioden ska lysa 1/2 sekund, vara släckt 1/2 sekund o s v, fordras två FOR...NEXT-slingor, vardera om 250 varv. Så här kan det då se ut.

```
10 OUT 2, 255          (8 utgångar)
20 OUT 0,1             (tänder dioden)
30 FOR I=1 TO 250      (väntar 1/2 sekund)
40 NEXT I
50 OUT 0,0             (släcker dioden)
60 FOR I=1 TO 250      (väntar 1/2 sekund)
70 NEXT I
80 GOTO 20             (hoppar tillbaka och tänder igen)
```

Nu blinkar dioden ca 1 gång per sekund, fullt synligt. Prova gärna med andra fördröjningar!

Att känna av strömbrytare

Om du inte kopplar in någonting till användarporten, utan lämnar PB0 - PB7 öppna, tolkar in/ut-kretsen detta som "ettor". Detta beror på att PB0 - PB7 internt är anslutna till +5V via högohmiga motstånd. Prova det här programmet:

```
10 OUT 2, 0            (8 ingångar)
20 A = INP (0)         (A tillordnas värdet av de 8 ingångarna)
30 PRINT A             (värdet av A skrivs på skärmen)
40 GOTO 20
```

Programmet skriver kontinuerligt på skärmen den decimala motsvarigheten till det binära tal, som matas in på port B. Med alla ingångar öppna skrivs talet 255, d v s 11111111_2. Om du nu jordar PB0 visas istället 254, och jordar du PB7 visas 127. Jordas *alla* ingångar visas förstås 0. Prova gärna några olika kombinationer!

Hur gör man nu, om det bara är en enda ingång som är intressant? Jo, datorn har en uppsättning logiska funktioner, som är som klippta och skurna för detta. Den logiska AND-funktionen används bl a för att "maska bort" ointressant information. Låt oss anta att tre strömbrytare är anslutna:

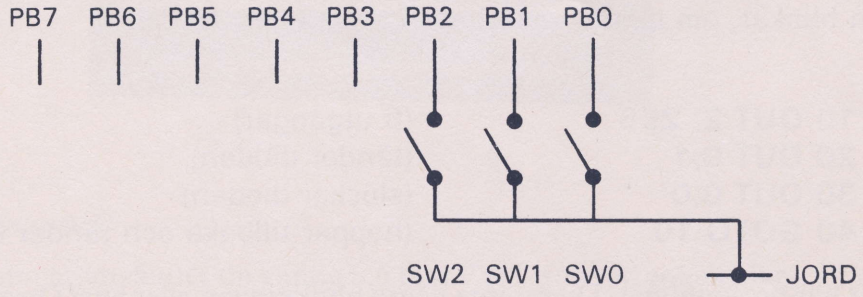


Fig. 2.9

Låt oss vidare förutsätta att vi bara är intresserade av tillståndet hos strömbrytaren SW2, som är ansluten till PB2. Låt då datorn samla in all information från port B med ett INP-kommando, (d v s alla åtta ledningarnas status). Maska sedan bort de sju onödiga bitarna med lämpliga maskbitar och en AND-funktion.

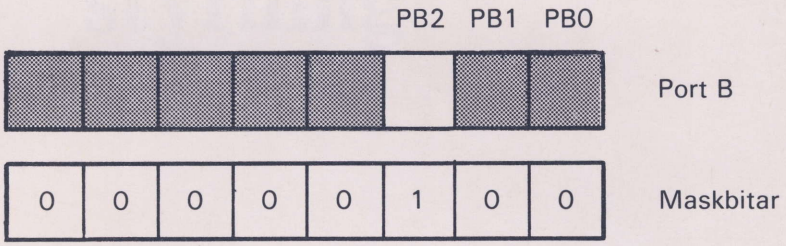


Fig. 2.10

Resultatet efter AND-funktionen blir nollor i alla de positioner där maskbitarna var nollor. I den position där maskbiten var "1" återfinns den ursprungliga informationen från port B, d v s 0 eller 1. I ovanstående exempel är masken $00000100_2 = 4_{10}$.

Så här ser programmet ut:

```
10 OUT 2,0          (8 ingångar)
20 A = INP (0)
30 B = A AND 4
40 PRINT B
50 GOTO 20
```

När SW2 är öppen skrivs 4 på skärmen, när den är sluten skrivs 0. Inga andra ingångar påverkar resultatet.

Du kan föreställa dig masken som pappskiva med "titthål" där det står ettor. Bara de bitar du ser genom hålet påverkar resultatet (med sin respektive vikt).

3. Att använda datorn för styrning

Programverk

I samband med sekvensstyrning används ofta elektromekaniska programverk. Du hittar dem bl a i hushållsmaskiner (disk- o tvättmaskiner) och i industriella processtyrsystem. En timer av den typ som brukar användas för att förleda potentiella inbrottstjuvar att tro att lägenhetsinnehavaren är hemma, eftersom golvlampan tänds klockan 19.15 och släcks 22.35 varje kväll, även när nämnda lägenhetsinnehavare gonar sig på Kanarieöarna, är också ett enkelt elektromekaniskt programverk.

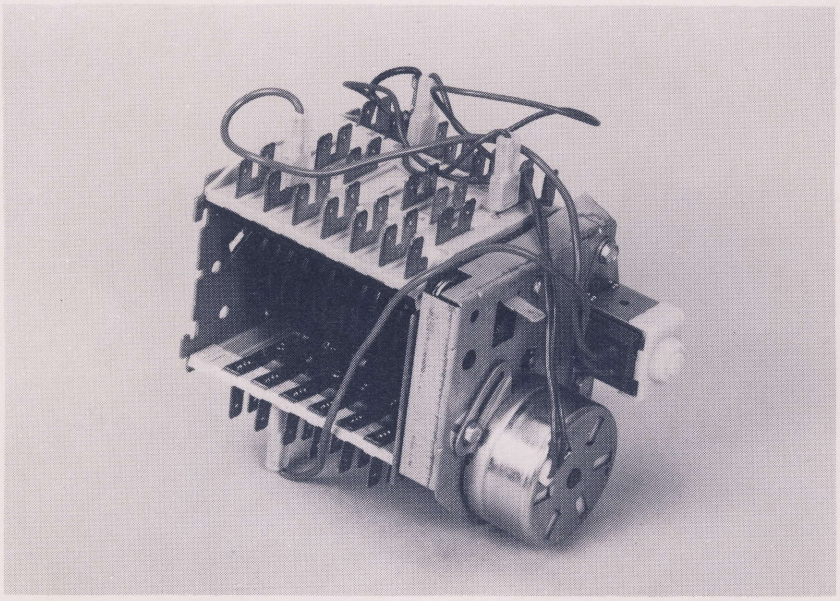


Fig. 3.1 Elektromekaniskt programverk

Programverket i t ex en tvättmaskin består av en synkronmotor och en växel, som får utgående axeln att rotera långsamt, ca 1 varv per timme. På axeln sitter ett antal kamskivor, som slår till eller från mikroswitchar under sin långsamma rotation. Mikroswitcharna kontrollerar diverse motorer och pumpar i maskinen, enligt det schema som bestäms av kamskivorna. Normalt finns också möjlighet att stoppa programverkets rotation till dess att ett visst villkor är uppfyllt (t ex att en givare känner av att tvättmaskinen verkligen är full med vatten).

Moderna hushållsmaskiner innehåller små mikrodatorer, som via portar styr reläer och motorer. Inom processindustrin används större datorer för sådan sekvensstyrning. I det här kapitlet ska vi titta på några olika enkla sådana styrutrustningar, där Spectravideodatorerna kan tänkas göra nytta. Vi kommer också att detaljstudera de gränssnitt som behövs för styrning av höga strömmar och för generering av analoga signaler.

Styrning av trafiksignaler

För styrning av trafiksignaler i en gatukorsning fordras en styrsekvens som tänder och släcker ett antal röda, gula och gröna lampor. I vissa fall ska sekvensen kunna påverkas av givare, som aktiveras av passerande fordon. Till att börja med ska vi titta på hur en periodisk styrsekvens kan genereras av din dator. Du använder lämpligen röda, gula och gröna lysdioder anslutna till användarporten, för att kunna simulera trafiksignalerna.

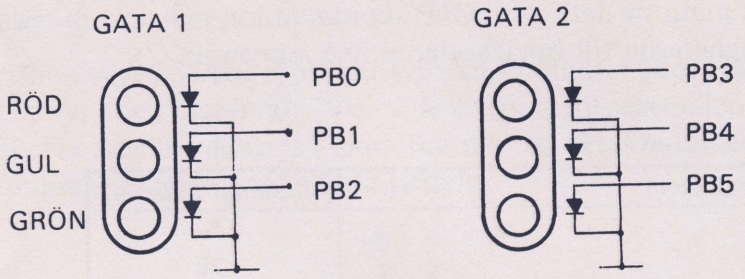


Fig. 3.2 Trafiksignaler

Trafiksignalerna ska styras periodiskt med en periodtid på 20 sekunder enligt figur 3.3.

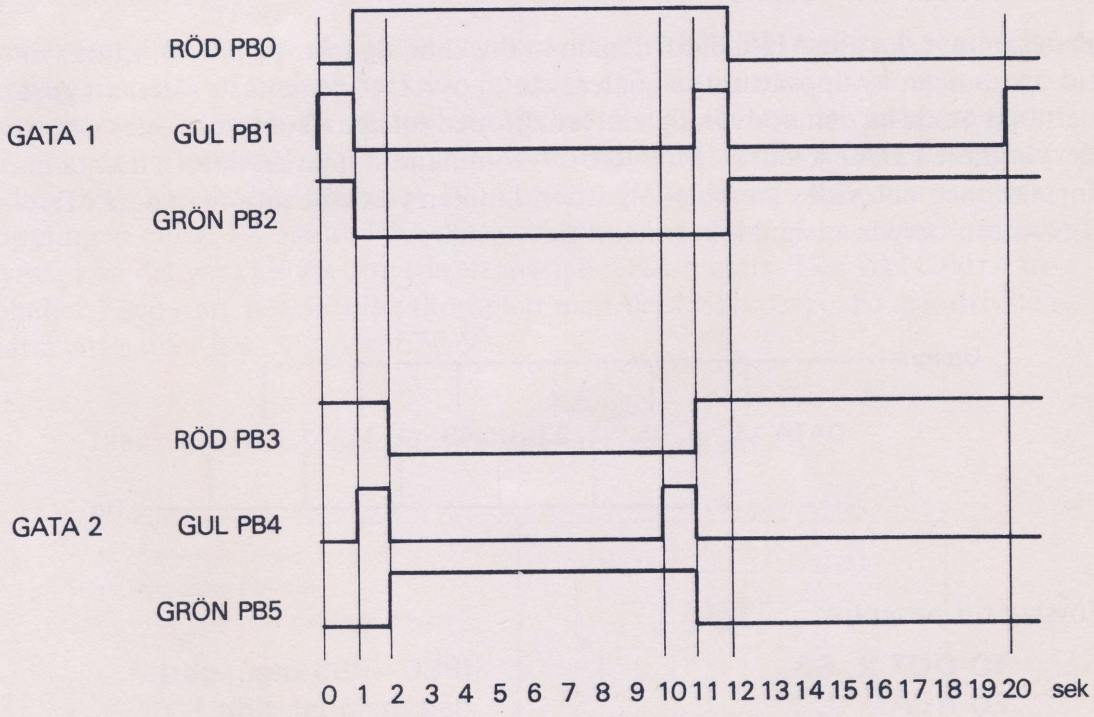


Fig 3.3 Styrsekvens för trafiksignal

Första steget blir att ta reda på vilken signalkombination som fordras i varje tidsintervall! Genom att studera fig 3.3 kan vi fylla i följande tabell:

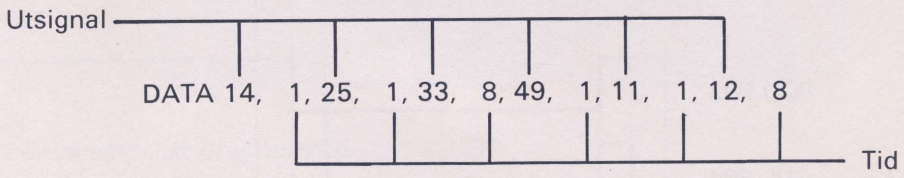
Tidsintervall (sek)	Utsignal på användarporten							
	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
0-1	0	0	0	0	1	1	1	0
1-2	0	0	0	1	1	0	0	1
2-10	0	0	1	0	0	0	0	1
10-11	0	0	1	1	0	0	0	1
11-12	0	0	0	0	1	0	1	1
12-20	0	0	0	0	1	1	0	0

Eftersom vi måste mata ut data med OUT-kommandon måste vi ta reda på de decimala motsvarigheterna till binärkoderna, (se Appendix C).

Tidsintervall (sek)	Decimal utsignal
0-1	14
0-2	25
2-10	33
10-11	49
11-12	11
12-20	12

De nödvändiga tidsfördröjningarna går lätt att ordna med FOR...NEXT-slingor, som du redan sett i kap. 2.

Programmet ska alltså låta datorn mata ut önskade signaler på port B, vänta viss tid, mata ut en ny uppsättning signaler, vänta, osv. Det elegantaste sättet att göra detta på är att ha den nödvändiga informationen (utsignalkombinationer och tider) lagrade i DATA-satser. Med READ-kommandot låter du datorn hämta informationen och vidta lämpliga åtgärder. Du kan t ex låta varannan data i DATA-satsen betyda utsignal, varannan tid – såhär:



Förslag till program:

```
10 OUT 2, 63                          (PB0—PB5 utgångar)
20 READ U, T                          (läs utsignal, tid)
30 IF U = 12 THEN RESTORE             (börja om med första data)
40 OUT 0,U                            (mata ut)
50 FOR I = 1 TO T * 500               (fördröjer ca T sekunder)
60 NEXT I
70 GOTO 20
80 DATA 14, 1, 25, 1, 33, 8, 49, 1, 11, 1, 12, 8
```

Fördröjningen på raderna 50 och 60 är experimentellt bestämd, med hjälp av stoppur! Fundera gärna över hur t ex en givare i gata 1 kan fås att påverka sekvensen.

Drivsteg för högre strömmar

Eftersom in/ut-kretsens drivförmåga inskränker sig till några få milliampere, fordras ofta någon form av extra drivsteg. Låt oss studera hur man kan driva reläer från en UT-port.

Relädrivning

Miniatyrreläer av s k "reed-typ" har typiska kontakter som klarar av att bryta 100V DC,0,5A, dock maximalt 10W. För styrning av reläspolen åtgår ca 20 mA vid 5V, vilket är mer än vad en UT-port klarar av direkt. Med ett enkelt transistordrivsteg fungerar emellertid det hela perfekt!

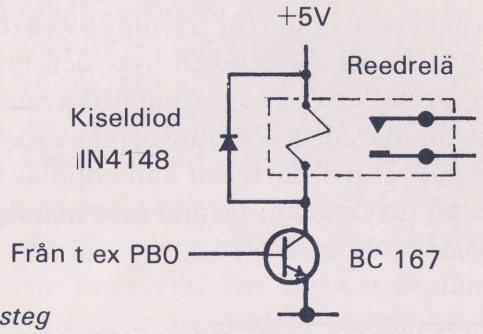


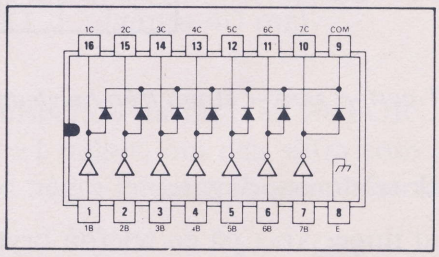
Fig. 3.4 Relädrivsteg

Transistortypen är inte alls kritisk, vilken lågeffekt NPN-transistor som helst fungerar utmärkt. Diodens funktion är att skydda transistorn för de spänningstransienter som uppstår då reläspolen switchas. Reläer för styrning av högre effekter, t ex för nätspänning 220 VAC, 5A, fordrar mera effekt för spolen. Ett typiskt sådant relä kan fordra uppemot 100 mA vid 5V matning. Även här fungerar kopplingen enligt fig 3.4 alldeles utmärkt. Om det är flera olika reläer som ska styras, kan det vara idé att utnyttja en speciell relädrivkrets. T ex ULN2001A innehåller 7 separata relädrivsteg (komplett med skyddsdioder) och varje drivsteg klarar att sänka hela 0,5 A vid 50 V.

TYPE ULN2001A, DARLINGTON TRANSISTOR ARRAY

- 500 mA Rated Collector Current
- High-Voltage Outputs . . . 50 V
- Output Clamp Diodes
- Inputs Compatible with Various Types of Logic
- Relay Driver Applications
- Designed to be Interchangeable with Sprague ULN2001A Series

DUAL-IN-LINE PACKAGE (TOP VIEW)



description

The ULN2001A, ULN2002A, ULN2003A, and ULN2004A are monolithic high-voltage, high-current darlington transistor arrays. Each comprises seven n-p-n darlington pairs. All units feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads. The collector-current rating of each darlington pair is 500 milliamperes. Outputs and inputs may each be paralleled for higher current capability. Applications include relay drivers, hammer drivers, lamp drivers, display drivers (LED and gas discharge), line drivers, and logic buffers. For 100-volt (otherwise interchangeable) versions, see the SN75466 through SN75469.

The ULN2001A is a general-purpose array and may be used with DTL, TTL, P-MOS, CMOS, etc. The ULN2002A is specifically designed for use with 14- to 25-volt P-MOS devices and each input has a zener diode and resistor in series to limit the input current to a safe limit. The ULN2003A has a series base resistor to each darlington pair. This allows operation directly with TTL or 5-volt CMOS. The ULN2004A has an appropriate series input resistor to allow its operation directly from CMOS or P-MOS utilizing supply voltages of 6 to 15 volts. The required input current is below that of the ULN2003A while the required voltage is less than that required by the ULN2002A.

schematic (each darlington pair)

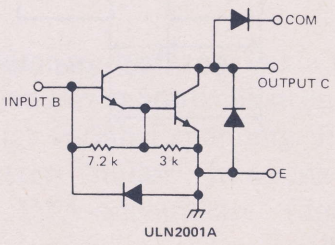


Fig. 3.5 ULN 2001A drivsteg

Halvledarrelä

Ett halvledarrelä består av en optokopplare och en triac (och ofta någon form av elektronik för s k nollgenomgångsstyrning). Optokopplaren isolerar styringångarna galvaniskt från resten av kretsen. Sett från styringången utgör halvledarreläet helt enkelt en lysdiod! När lysdioden tänds, påverkar ljuset en fototransistor, som i sin tur via nollgenomgångsstyrningen får triacen att leda. Observera att denna typ av halvledarrelä bara går att använda för styrning av växelström! Detta beror på att triacen bara kan släckas då spänningen över den blir 0, alltså två gånger per period. För styrning av nätlaster är dock halvledarreläer synnerligen lämpliga, dels klarar de höga strömmar, dels gör frånvaron av rörliga kontakter att tillförlitligheten blir extremt god. Halvledarreläet kan kopplas in till UT-porten på två olika sätt, beroende på drivkraven. Denna kan nämligen bara *driva* några få mA, men den kan *sänka* ca 10 mA.

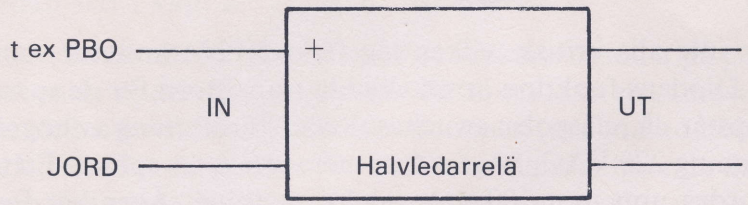


Fig. 3.5 UT-porten driver halvledarreläet

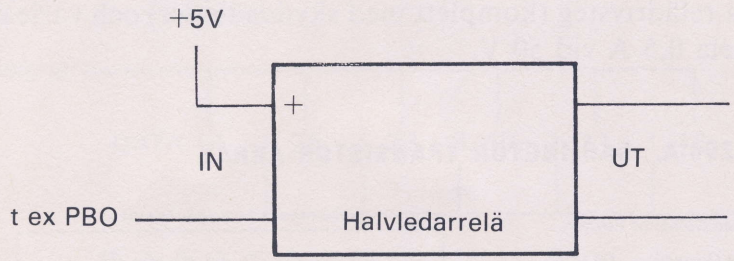


Fig. 3.6 UT-porten sänker ström från halvledarreläet

Automatisk telefonuppringare

En normal fingerskiva på en telefon bryter under sin fjäderdrivna återgång en strömkrets N+1 gånger, om den valda siffran är N. Vid en "Nolla" erhålls alltså *ett* avbrott och vid en "nia" *tio* avbrott. Återgångsrörelsens hastighet regleras av en centrifugalregulator, så att periodtiden blir 100 ms. Därav utgör avbrottstiden 60 ms.

Exempelvis siffran "fyra" ger följande pulssekvens:

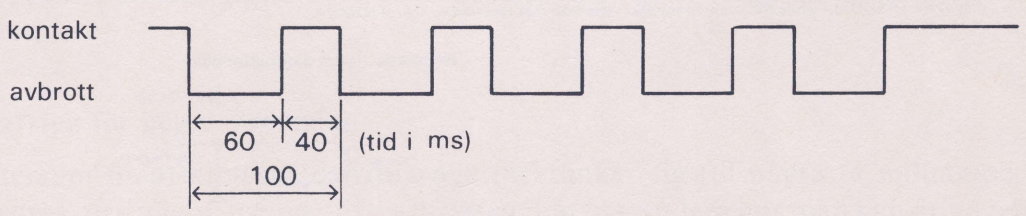


Fig. 3.7

Vare sig periodtid eller avbrottstid är speciellt kritiska. Avvikelser upp till ± 10% tolereras normalt av telefonväxlarna.

Mikrodatorn kan givetvis programmeras, så att en pulssekvens enligt ovan genereras på t ex PB0. Siffrorna kan skrivas in i en tabell i minnet, varför datorn kan utnyttjas både för lagring av telefonnummer och för nummertagning.

Du vet redan hur man genererar pulser med viss längd, så svårigheten med det här programmet är snarare att lagra respektive hämta nummerinformationen på ett effektivt sätt. Det enklaste sättet är definitivt att lagra numret i DATA-satser, siffra för siffra. T ex numret 1234567 lagras som DATA 1, 2, 3, 4, 5, 6, 7. Vidare behövs en READ-sats för att läsa data och en subrutin för att generera pulser.

Låt oss definiera "kontakt" som "nolla" och "avbrott" som "etta" i utsignalen från PB0. Det går givetvis utmärkt att koppla ett relä till PB0 för att ersätta fingerskivan, men Televerket har mycket bestämda åsikter om lämpligheten, detta är inte tillåtet! För att studera utsignalen från PB0 duger emellertid en lysdiod utmärkt.

```
10 OUT 2,1                          (PB0 utgång)
20 READ N                           (läs siffra)
30 FOR I = 0 TO N+1
40 GOSUB 90                         (generera en pulsperiod)
50 NEXT I
60 FOR J = 1 TO 150 : NEXT J        (vänta 300 ms mellan siffrorna
70 GOTO 20
80 DATA 1, 2, 3, 4, 5, 6, 7,        (telefonnumret)
85 END
90 OUT 0,1                          (mata ut en etta)
100 FOR J = 1 TO 30 : NEXT J        (vänta 60 ms)
110 OUT 0,0                         (mata ut en nolla)
120 FOR J = 1 TO 20 : NEXT J        (vänta 40 ms)
130 RETURN
```

Subrutinen för utmatning av en enda pulsperiod startar i rad 90. FOR ...NEXT-slingan på raderna 30--50 matar ut N+1 pulser, om den siffra som löstes ur DATA-satsen var N. Prova gärna med några andra telefonnummer i DATA-satsen.

En mera raffinerad variant av programmet skulle kunna vara att utnyttja telefonnummer, lagrade som strängar. Med hjälp av strängfunktioner (LEFT$, MID$, LEN, etc) går det att plocka fram de enskilda siffrorna i numret och generera pulssekvensen enligt programmet ovan. Detta ger förstås oanade möjligheter till att länka den automatiska nummertagaren med ett namn/nummer-register på flexskiva. Hisnande utsikter, inte sant? Använd dina kunskaper i programmering och filhantering på flexskiva för att skriva ett sådant program. Din "elektroniska" telefonkatalog ska givetvis kunna uppdateras.

Analoga signaler från datorn

Det är minsann inte alla periferienheter som låter sig styras av enkla till/från signaler från reläer eller liknande. Vill du t ex styra farten hos en likströmsmotor fordras en *analog* signal, d v s en kontinuerligt variabel spänning. Eftersom datorer producerar digitala signaler, "ettor" resp "nollor" från sina portar, behövs gränssnitt för omvandling, s k D/A-omvandlare (digital-till-analogomvandlare).

Resistansnät

De flesta D/A-omvandlare utnyttjar principen med summation av binärt viktade strömmar, där respektive delströms magnitud beror på en binär siffras vikt. I t ex en 8-bitars D/A-omvandlare låter man den minst signifikanta biten i insignalen styra en strömgenerator som ger t ex 1 μA vid "etta" (och 0 μA vid "nolla"), nästa bit 2μA, nästa 4μA osv. Den mest signifikanta biten styr en strömgenerator som ger 128 μA vid "etta". Alla strömmarna summeras och summan utgör utsignalen från D/A-omvandlaren. Kopplingen i fig. 3.8 skulle kunna användas där switcharna är reläer, styrda av PB0–PB7.

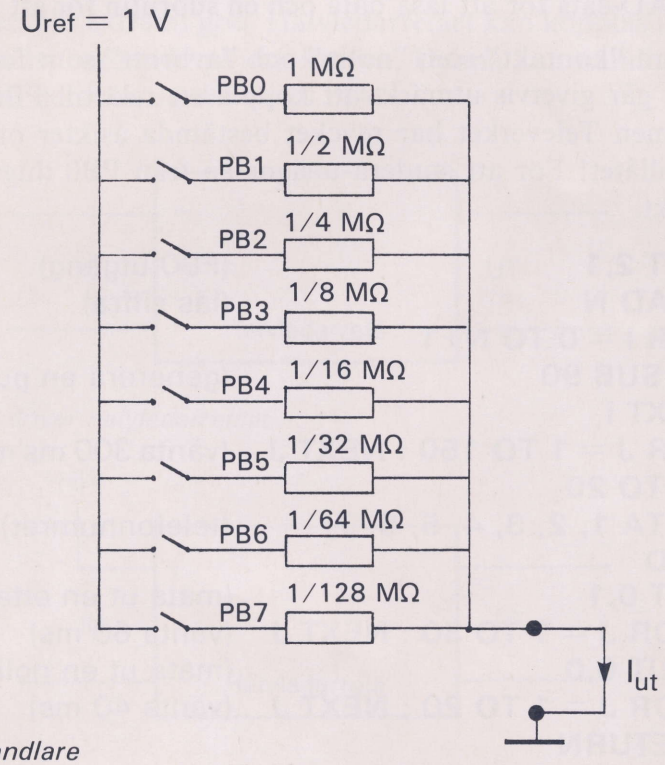


Fig. 3.8 D/A-omvandlare

Strömgeneratorerna utgörs helt enkelt av switchade, binärt viktade motstånd, och en konstant referensspänning. Utsignalen Iut kan anta alla heltalsvärden mellan 0 och 255 μA. Tomgångsspänningen över utgången blir f ö också proportionell mot den binära insignalen. En 8-bitars D/A-omvandlare får upplösningen 1/256 d v s 0,4%. De ingående motstånden måste förstås ha motsvarande precision! I verkligheten utnyttjas inte reläer utan halvledarswitchar. Som väl är finns D/A-omvandlare att köpa färdiga att koppla in till datorn, i form av IC-kretsar.

För en 8-bitars omvandlare får man betala några tior, så det är knappast möda värt att bygga själv. I nästa avsnitt ska vi för övnings skull koppla upp en enkel hembyggd D/A-omvandlare, men låt oss först titta på en kommersiell produkt, en IC med beteckningen AD 7523.

AD 7523

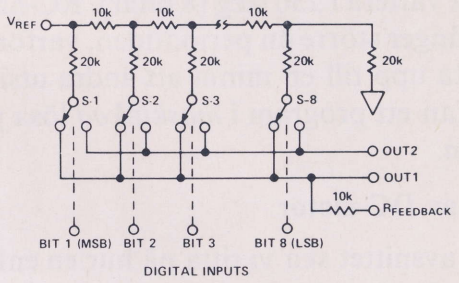
AD 7523 från Analog Devices är ett typexempel på en integrerad D/A-omvandlare. Internt utnyttjas ett resistansät för strömgenerering och styrningen sker med inbyggda halvledarswitchar. Resistansnätet ser lite annorlunda ut. Det är en s k R/2R-stege, där man utnyttjar strömgrening i varje knutpunkt för att erhålla binärt viktade delströmmar, fig. 3.9.

AD 7523 DIGITAL/ANALOG-OMVANDLARE

FEATURES
Low Cost
Fast Settling: 100ns
Low Power Dissipation
Low Feedthrough: ½LSB @ 200kHz
Full Four-Quadrant Multiplying

APPLICATIONS
Battery Operated Equipment
Low Power, Ratiometric A/D Converters
Digitally Controlled Gain Circuits
Digitally Controlled Attenuators
CRT Character Generation
Low Noise Audio Gain Control

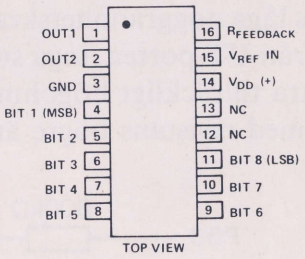
FUNCTIONAL DIAGRAM

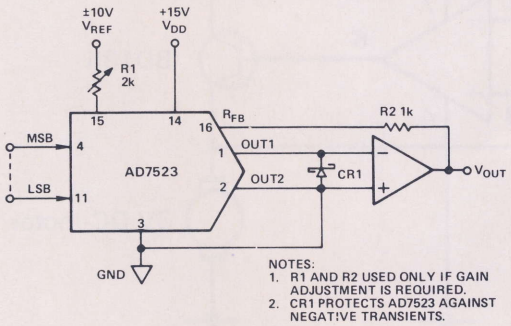


GENERAL DESCRIPTION

The AD7523 is a low cost, monolithic multiplying digital-to-analog converter packaged in a 16-pin DIP. The device uses an advanced monolithic, thin-film-on-CMOS technology to provide 8-bit resolution with accuracy to 10-bits and very low power dissipation.

The AD7523's excellent multiplying characteristics and low cost allow it to be used in a wide ranging field of applications such as: low noise audio gain control, CRT character generation, motor speed control, digitally controlled attenuators, etc.





DIGITAL INPUT MSB LSB	ANALOG OUTPUT
11111111	$-V_{REF}\left(\frac{255}{256}\right)$
10000001	$-V_{REF}\left(\frac{129}{256}\right)$
10000000	$-V_{REF}\left(\frac{128}{256}\right) = -\frac{V_{REF}}{2}$
01111111	$-V_{REF}\left(\frac{127}{256}\right)$
00000001	$-V_{REF}\left(\frac{1}{256}\right)$
00000000	$-V_{REF}\left(\frac{0}{256}\right) = 0$

Note: $1LSB = (2^{-8})(V_{REF}) = \frac{1}{256}(V_{REF})$

Fig. 3.9 AD 7523 Digital/Analog-omvandlare

Pulsbreddsmodulering

Ett helt annat sätt att generera en analog signal, är att utgå från ett pulståg med konstant frekvens, där pulsbredden är proportionell mot det tal som ska omvandlas. Efter medelvärdesbildning (i ett RC-filter) erhålls DC-komponenten, som är proportionell mot pulsbredden och följaktligen också proportionell mot det ursprungliga talet.

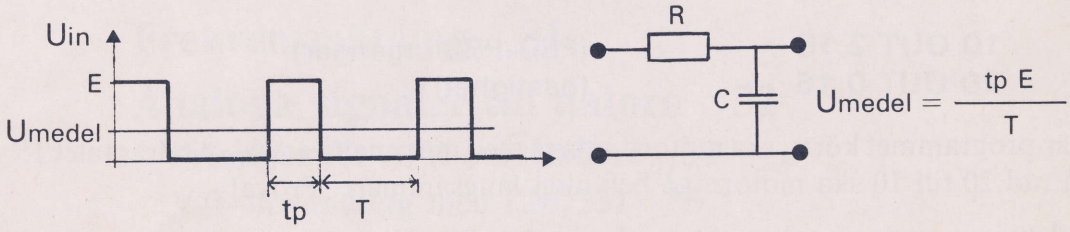


Fig. 3.10 D/A-omvandling med pulsbreddsmodulering

Med en dator till förfogande går det lätt att generera pulståget och de enda yttre komponenter som behövs är ett motstånd och en kondensator. Nackdelen med metoden är, att det tar förhållandevis lång tid att ändra utsignalen. Med ett basic-program är det svårt att generera högre frekvens än någon Hz, om pulsbredden ska gå att variera i 256 steg (8 bitar). RC-nätets tidskonstant bör vara åtminstone 20 - 30 gånger större än periodtiden, varför slutresultatet blir en omvandlare där det kan ta upp till en minut att ändra utsignal från min till max! Som vi ska se senare, kan ett program i *maskinkod* lösa problemet med den långsamma reaktionstiden.

Styrning av DC-motor

I det här avsnittet ska vi titta på hur en enkel 4-bitars D/A-omvandlare med ett effektsteg för styrning av en liten likströmsmotor kan byggas. En 4-bitars D/A-omvandlare kan ge $2^4=16$ nivåer, och upplösningen blir 1/16 d v s 5%. Det förhållandevis låga noggrannhetskravet gör att vi kan använda ett resistansnät, matat direkt från UT-porten, inga switchar behövs alltså! Å andra sidan måste motstånden vara tillräckligt högohmiga, så att inte UT-porten belastas för mycket. Motstånd med resistans högre än 100 kΩ är lämpliga.

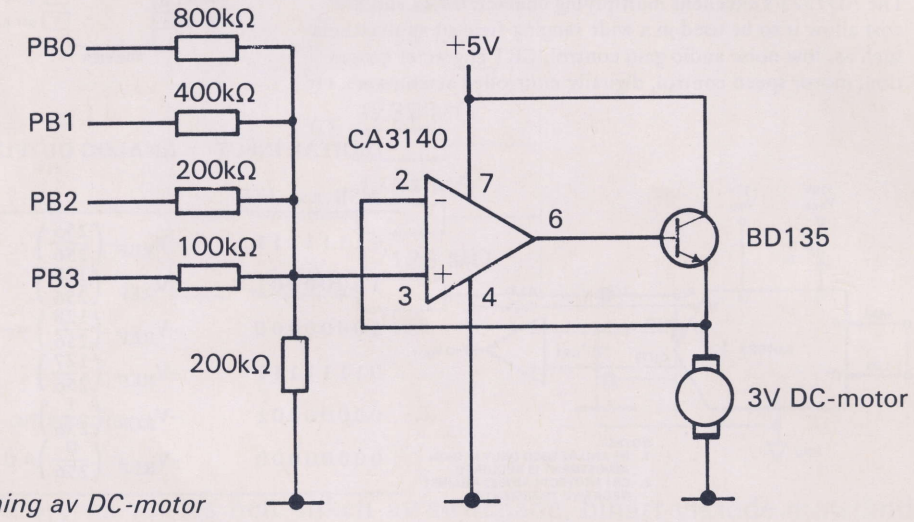


Fig. 3.11 Styrning av DC-motor

Spänningen får *inte* tas från datorn p g a att strömuttaget är begränsat till 100 mA. En liten DC-motor ("leksaksmotor", t ex Clas Ohlssons nr 22-2249) drar åtminstone 0,5 A. Strömförsörj istället motorn från ett 4,5V ficklampsbatteri eller ett externt 5V-aggregat.

Operationsförstärkaren och effekttransistorn fungerar som effektförstärkare, med spänningsförstärkning 1. 200 kΩ-motståndet till jord tjänar till att belasta själva D/A-omvandlaren, så att maximal insignal till operationförstärkaren blir ca 3V. För att köra motorn behövs ett Basic-program med endast två instruktioner.

```
10 OUT 2,15        (PB0–PB3 utgångar)
20 OUT 0,15        (hastighet)
```

När programmet körts, ska motorn snurra med maximalt varvtal. Ändras talet 15 på rad 20 till 10 ska motorn gå betydligt långsammare. Prova!

Vill du styra motorn enligt någon sekvens, kan "trafiksignalprogrammet", som vi studerade i ett tidigare avsnitt användas.

4. Att använda datorn för mätning

Antalsräkning med fotocell

En av de enklaste formerna av "mätning" är onekligen *antalsräkning*. Ur datorns synpunkt är antalsräkning ekvivalent med att hålla reda på hur många gånger insignalen till en inport har växlat från t ex "0" till "1".

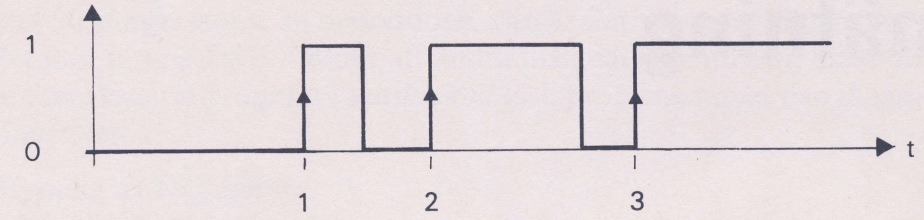


Fig. 4.1 Antalsräkning

I det här avsnittet ska vi titta på hur du kan koppla in en fotocell till användarporten och använda den för att räkna hur många gånger en ljusstråle bryts. Tillämpningarna är många, speciellt inom området processautomatisering. Beroende på ljuskällans intensitet, kan gränssnittet ha olika utseende. Med en stark, välfokuserad ljuskälla, riktad mot fotocellen, behövs bara en givare, en *fototransistor*, inkopplad till t ex PB6.

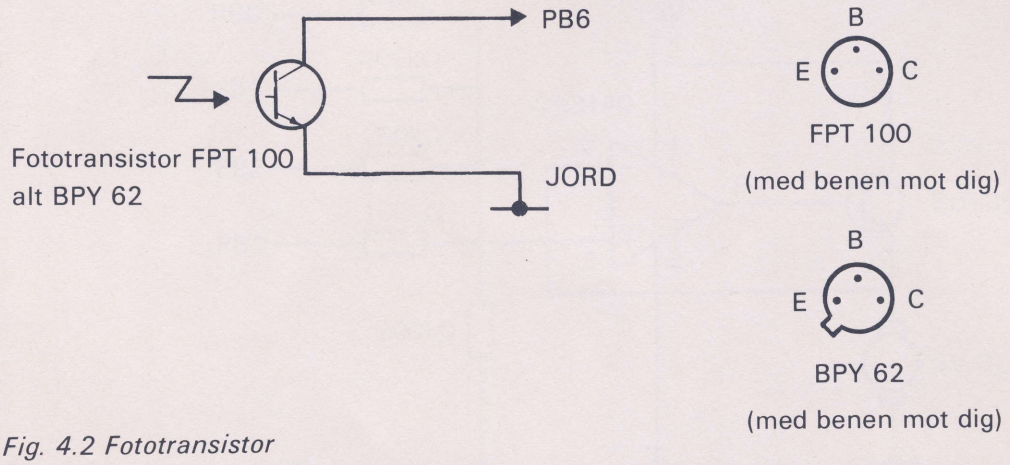


Fig. 4.2 Fototransistor

När ljus träffar fototransistorns fönster, blir PB6 "0" och när ljusstrålen bryts blir PB6 "1". Med lägre ljusintensitet behöver signalen från fototransistorn förstärkas innan den kopplas in till porten. Även här blir slutresultatet "0" för belyst fototransistor, "1" för skuggad, fig. 4.3.

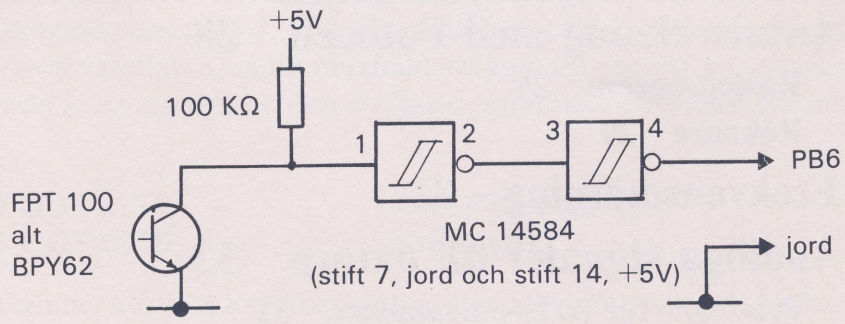


Fig. 4.3. Fototransistor med förstärkare

Basicprogram

Det är nu dags att ta en titt på ett program för antalsräkning:

```
10 OUT 2,191
20 X=0
30 A=INP(0) AND 64
40 IF A=0 THEN 30
50 X=X+1
60 PRINT X
70 A=INP(0) AND 64
80 IF A=64 THEN 70
90 GOTO 30
```

En analys av programmet kan kanske vara på sin plats:

Rad 10: Gör PB6 till ingång, övriga PB till utgångar.

Rad 20: X används som räknevariabel, d v s X ska ökas med ett varje gång PB6 blir en "etta".

Rad 30: INP(0) mäter status hos alla PB, AND 64 maskar bort allt utom PB6. Om PB6=0 blir A=0, om PB6=1 blir A=64.

Rad 40: Om A=0 betyder det att PB6=0, d v s ljusstrålen har ännu inte brutits. Räkna *inte* upp variabeln X, utan gå tillbaka och mät på nytt.

Rad 50 A var ≠ 0 (d v s 64) varför ljusstrålen måste ha brutits. Räkna därför upp X.

Rad 60: Skriv värdet av X på skärmen.

Rad 70: Samma som rad 30!

Rad 80: Om A=64 betyder det att PB6=1 d v s ljusstrålen är fortfarande bruten. Gå därför tillbaka till 70 och mät på nytt, ända till dess att ljuset kommer igen.

Rad 90: Starta en ny räknecykel.

Eftersom varje Basickommando tar ca 1 ms för datorn att utföra, tar varje räknecykel storleksordningen 10 ms. Maximal räknehastighet begränsas därför till ca 100 pulser per sekund. Har du krav på större snabbhet återstår att använda antingen mera "maskinvara" (räknare) eller att programmera i maskinkod. Vi återkommer till maskinkod i kapitel 5 – låt oss istället undersöka maskinvarualternativet!

Räknare

Redan i ett tidigare avsnitt nämndes att VIA-kretsen innehåller mycket mera än bara en port. Bl a innehåller den en 16-bitars binärräknare, som kan användas för att räkna antalet pulser på PB6. I fabrikantens datablad kallas denna räknare "Timer 2", (det finns en till!). Timer 2 är uppdelad i två 8-bitarshalvor. Eftersom vi arbetar med en 8-bitarsdator transporteras ju data som 8-bitarstal på bussen. Du kan själv nollställa räknaren eller förinställa den på önskat utgångsvärde och du kan givetvis också avläsa den. Som vanligt används OUT- och INP-kommandon för detta.

Timer 2 kan användas i två olika moder – i den ena som timer, i den andra som räknare. Du väljer önskad mod genom att skriva ett tal i ett s k kontrollregister.

De 8 minst signifikanta bitarna i räknaren i Timer 2 har adressen 8. De 8 mest signifikanta har adressen 9. Kontrollregistret har adressen 11 och det är bit 5 i kontrollregistret som används för val av mod. Bit 5 ="0" väljer timermod, bit 5 ="1" väljer räknarmod, fig 4.4.

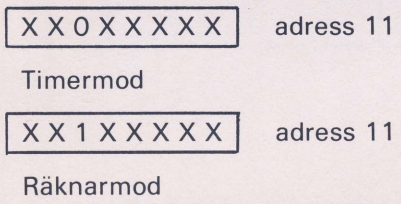


Fig. 4.4 Kontrollregister i VIA

En liten hake är, att Timer 2 räknar baklänges. Fabrikanten har nämligen tänkt sig att den ska användas för räkning av ett förutbestämt antal (som man från början laddar räknaren med). När räknaren räknats ned till noll genererar VIA-kretsen en signal till processorn, som talar om att den är klar, så att lämplig åtgärd kan vidtas. Laddas räknaren från början med 65535, innehåller den efter första räknepulsen 65534, efter nästa 65553 o s v. Genom att subtrahera räknarinnehållet från 65535 kan vi alltså beräkna antalet pulser.

För att räkna pulser med TIMER 2 måste följande punkter åtgärdas:

1 Ettställ bit 5 i kontrollregistret (adress 11)

2 Gör PB6 till ingång genom att skriva en nolla i bit 6 i datariktningsregistret (adress 2).

3 Förinställ de 8 minst signifikanta bitarna, d v s skriv 255 i adress 8

4 Förinställ de 8 mest signifikanta bitarna, d v s skriv 255 i adress 9.

Räknaren startas automatiskt när punkt 4 utförs.

När räknaren räknats ned till noll, slår den över och börjar på nytt, precis som vägmätaren i en bil. Detta kan utnyttjas för att räkna större antal än 65535.

Program:

```
10 OUT 11, INP (11) OR 32
20 OUT 2, 191
30 OUT 8, 255
40 OUT 9, 255
50 A=INP (8)
60 B=INP(9)
70 PRINT 65535-(256*B + A)
80 GOTO 50
```

Även detta program tarvar en förklaring:

Rad 10: INP(11) ger som resultat det tal som för tillfället står i kontrollregistret, OR 32 medför att alla bitar i talet förblir oförändrade utom bit 5 (vikt 32) som alltid ettställs. Resultatet lagras i kontrollregistret. Denna operation väljer räknarmoden. Orsaken till att vi inte bara skrev en "etta" i bit 5 är att vissa av de övriga bitarna kan användas för andra ändamål.

Rad 20: Gör PB6 till ingång.

Rad 30: Förinställer minst signifikanta räknehalvan till $11111111_2=255_{10}$.

Rad 40: Förinställer mest signifikanta räknehalvan till 255_{10} och startar räknaren.

Rad 50: Läser minst signifikanta räknarhalvan och placerar resultatet i A.

Rad 60: Läser mest signifikanta räknarhalvan och placerar resultatet i B.

Rad 70: (256★B + A) kombinerar de två 8-bitarstalen till ett sextonbitars, genom att ge mest signifikanta talet vikten 256. 65535 minskat med detta tal korrigerar för "baklängesräkningen". Resultatet skrivs på skärmen.

Rad 80 Gå tillbaka och läs räknaren igen.

Fördelen med Timer 2 är att räknehastigheten kan vara så pass hög som 1 MHz, jämfört med bara 100 Hz om man väljer att räkna med ett program. Tiotusen gånger snabbare är väl inte så illa?

Frekvensmätning

För att kunna mäta frekvens, behöver vi en *tidbas,* som medger räkning av antal pulser under en sekund (ger resultat i Hz). Vi har tidigare genererat tidsintervall med FOR...NEXT-slingor, men detta ger dålig precision – speciellt för korta tider. Vi vet inte exakt hur lång tid Basictolken fordrar för att översätta programmet, bara att det tycks röra sig om några ms per rad. Bättre är att använda en timer i VIA-kretsen eller den inbygda realtidsklockan i datorn. Realtidsklockan kan avläsas med Basickommandon.

I följande program utnyttjas PB6 som ingång och Timer 2 för räkning, precis som i föregående avsnitt:

```
10 OUT 11, INP (11) OR 32
20 OUT 2, 191
30 OUT 8, 255
40 T = TIME
50 IF T = TIME THEN 50
60 OUT 9, 255
70 IF TIME<>T +50 THEN 70
80 PRINT 65535 - (256★INP(9)+INP (8))
90 GOTO 30
```

Kommentarer till ovanstående program:

Rad 10 Sätter bit 5 i kontrollregistret "1" för att välja räknarmoden hos Timer 2.

Rad 20 Gör PB6 till ingång.

Rad 30 Förinställer minst signifikanta räknarhalvan till 255.

Rad 40 Läser av realtidsklockan (i 50-dels sek) och placerar resultatet i T.

Rad 50 Datorn väntar här tills nästa gång TIME ändras och går sedan vidare. Klockan läses ju av "i flykten" och ger en osäkerhet på 1/50 sek, om vi inte väntar till ett omslag.

Rad 60 Förinställer mest signifikanta räknarhalvan till 255 och startar räknaren.

Rad 70 Datorn väntar här,men fortsätter att räkna pulser på PB6 tills totalt 1 sekund har förflutit från det att rad 50 påbörjades. T + 50 ger 1 sekund mera än T, eftersom datorn mäter femtiondels sekunder.

Rad 80 Skriver frekvensen på skärmen.

Rad 90 Går tillbaka och gör en ny mätning.

Den högsta frekvens som kan mätas med det här programmet är 65 kHz, eftersom Timer 2 bara kan räkna till 65535. Datorn mäter inte under exakt en sekund, vi har ett fel på några millisekunder p g a att Basictolken fordrar viss tid att tolka raderna 50, 60 och 70. Det är omöjligt att eliminera detta fel, så länge vi skriver programmen i Basic. Detta språk är egentligen inte avsett för sådana här tillämpningar, utan när kännedom om exakta tider är väsentlig, programmerar man lämpligen i maskinkod, men mer om detta i Kapitel 5.

Hur som helst – det fel på några promille, som Basicprogrammeringen orsakar i ovanstående frekvensmätningsprogram går ofta att tolerera. Vi får anledning att utnyttja det här programmet längre fram i detta kapitel.

Analoga signaler till datorn

Insignaler till datorn från olika givare föreligger oftast i analog form. Vanligtvis fordras någon form av signalbehandling, t ex förstärkning eller filtrering, innan den analoga signalen är användbar. Med den elektronik som står till buds idag (t ex operationsförstärkare) är det därför ganska naturligt att utgå från att de flesta analoga signaler som ska "databehandlas" utgörs av mer eller mindre konstanta spänningar. Innan datorn kan ta hand om signalerna, måste de digitaliseras. Detta görs med en A/D-omvandlare, (analog till digital-omvandlare). Utsignalen från A/D-omvandlaren består av "ettor" och "nollor", som kan tas om hand av en inport på datorn.

Principer för A/D-omvandlare

De A/D-omvandlare som används för datainsamling till datorer, fungerar i huvudsak enligt någon av följande principer: integrerande, successiv approximation och spänning/frekvensomvandling. Samtliga dessa typer av A/D-omvandlare går att köpa färdiga som IC-kretsar, dock måste några motstånd och kondensatorer ofta tillfogas innan A/D-omvandlaren kan fungera.

Den integrerande A/D-ovandlaren

En integrerande A/D-omvandlare är egentligen en spännings/tidomvandlare, där tiden mäts med en räknare och en inbyggd klocka. Den uppmätta tiden är proportionell mot den "okända" spänningen. Denna typ av omvandlare integrerar insignalen med hjälp av en operationsförstärkarkoppling och jämför den med en referensspänning. Det tar förhållandevis lång tid att göra en omvandling, ca 1/10 sek, men fördelen är att eventuella störningar på insignalen medelvärdesbildas, varför deras inverkan minskas. Speciellt viktigt är detta om man har långa, oskärmade ledningar som kan plocka upp 50 Hz-brum från nätet. Inverkan av sådana periodiska störningar kan helt elimineras, om integrationstiden väljs till en multipel av nätets periodtid, d v s till 1/20 sek, 1/10 sek o s v. De flesta digitalvoltmetrar innehåller en integrerande A/D-omvandlare.

Den successivt approximerande A/D-omvandlaren

En A/D-omvandlare av typ successiv approximation utnyttjar en D/A-omvandlare, styrd av digital elektronik, och en komparator. Komparatorn jämför den "okända" insignalen med spänningen från D/A-omvandlaren och den digitala elektroniken styrs av utsignalen från komparatorn. Slutresultatet blir att utsignalen från D/A-omvandlaren blir lika med den okända spänningen. Den digitala signalen till D/A-omvandlaren är då lika med den sökta digitala ekvivalenten till den okända analoga spänningen.

En sådan här omvandling kan ta mycket kort tid. Mellan 1 μs och 100 μs är vanliga omvandlingstider för 8-bitars omvandlare. Det är viktigt att tänka på att störningar kan få stor inverkan vid denna typ av omvandling, eftersom ingen medelvärdesbildning sker. Lösningen är filtrering och bandbegränsning av insignalen.

A/D-omvandlare av typ V/f

Den tredje typen av A/D-omvandlare, V/f-omvandlaren, eller "spännings/frekvensomvandlaren", är egentligen en spänningsstyrd oscillator, där utsignalen är ett pulståg vars frekvens är proportionell mot en pålagd, "okänd", spänning. Genom att datorn får mäta pulstågets frekvens, erhålls ett mått på spänningen. V/f-omvandlare fordrar förhållandevis lång omvandlingstid, 1/10–1 sek, men fördelarna är många. Bl a är den medelvärdesbildande till sin natur, varför störningar inte får så stor inverkan. Vidare fordras bara en enda ledare för den digitala utsignalen. Detta gör att det är mycket lätt att galvaniskt isolera omvandlaren från datorn med optokopplare. V/f-omvandlare är dessutom billiga! I nästa avsnitt ska vi titta närmare på en V/f-omvandlare i en IC-krets, LM 331.

V/f-omvandling med LM 331

Med LM 331 kan du koppla upp en mycket linjär och noggrann V/f-omvandlare på kopplingdäcket. En del motstånd och kondensatorer behövs dock.

För nedan visade krets gäller sambandet

$$f = \frac{U_{in} \times R_S}{2{,}09 \times R_L \times R_T \times C_T}$$

f är utsignalens frekvens i Hz, U_{in} är insignalen i volt. R_S, R_L och R_T mäts i ohm och C_T i F. Använd 1% metallfilmmotstånd för R_S, R_L och R_T och plastfoliekonden-

Vid ett experiment med kopplingen i fig. 4.7 uppmättes följande samband mellan temperatur och frekvens:

Temperatur °C	Frekvens Hz
+25	67
+45	104
+65	152
+85	209

Med hjälp av t ex "minsta-kvadrat-metoden" går det att till dessa mätpunkter anpassa en matematisk funktion, som beskriver sambandet. Kalibreringskurvan visas i fig. 4.8.

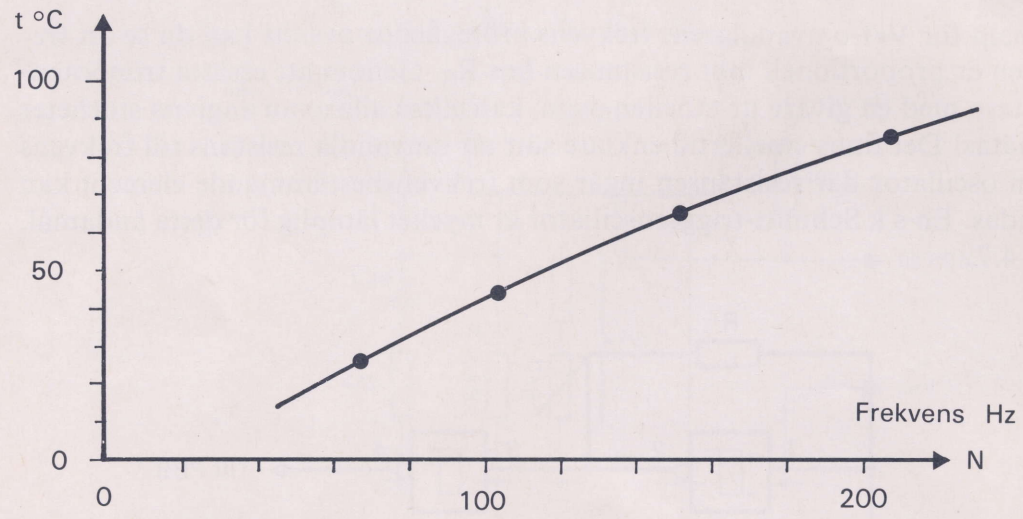


Fig. 4.8 Kalibreringskurva

Minsta-kvadrat-metoden är ganska besvärlig och en enkel grafisk anpassning kan vara tillräcklig. Kurvan kan approximeras med två räta linjer, fig. 4.9.

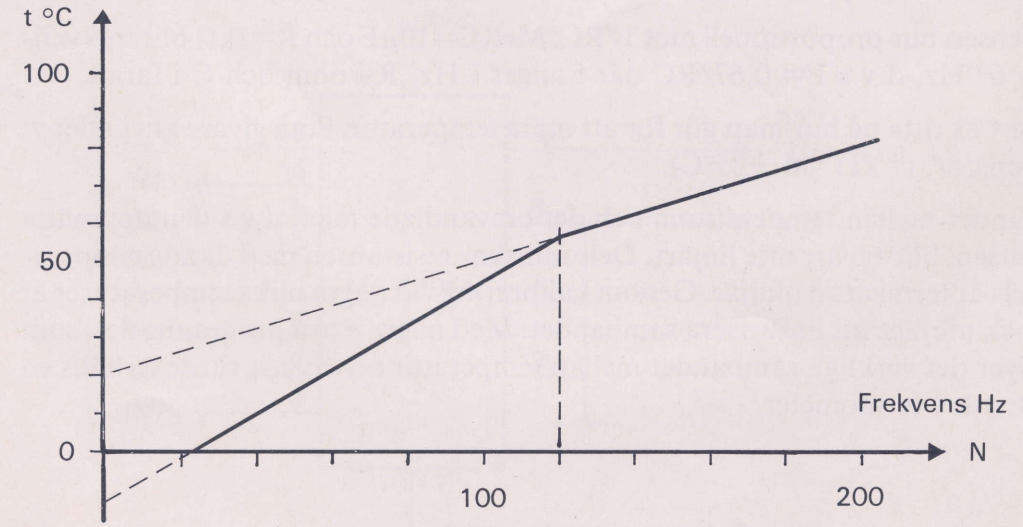


Fig. 4.9 Kalibreringskurvan approximeras med två räta linjer.

Ur de två räta linjerna erhålls (räta linjens ekvation)

$t = -15 + N \star 0{,}60$ $\quad N<120$
$t = 20 + N \star 0{,}30$ $\quad N>120$

Program för mätning och linearisering:

```
10 OUT 11, INP(11) OR 32
20 OUT 2, 191
30 OUT 8, 255
40 T = TIME
50 IF T = TIME THEN 50
60 OUT 9, 255
70 IF TIME <> T + 50 THEN 70
80 N = 65535 - (256*INP(9) + INP (8))
90 IF N < 120 THEN PRINT -15 + N * 0.6 : GOTO 30
100 PRINT 20 + N * 0.3
110 GOTO 30
```

Raderna 10-80 utför mätningen (se sid 31) och raderna 90 och 100 lineariserar.

5. Maskinkod

Vad är maskinkod?

Redan i kapitel 1 nämndes att operativsystemet och Basictolken i SV-318 och SV-328 är lagrade i läsminnen i form av åttabitars binära tal. Båda dessa program är *maskinkodsprogram,* d v s de är skrivna i en kod som mikroprocessorn direkt kan läsa och utföra. När du kör ett Basic-program på datorn ser Basictolken till att ditt Basicprogram översätts till en lämplig kombination av maskinkoder.

Varför nu allt detta besvär med översättning? Vore det inte enklare om varje datoranvändare lärde sig skriva program i maskinkod, istället för Basic? Det finns dock några nackdelar. För det första är maskinkoden beroende av vilken processor som används. Maskinkoderna för mikroprocessorn Z80, som sitter i SV-318 och SV-328 och t ex mikroprocessorn 6502 som sitter i VIC 20 har inte ringaste likhet med varandra. Däremot är den Basic-dialekt som används i VIC 20 nästan identisk med SV318/SV328-Basic. Att använda ett s k högnivåspråk som Basic innebär alltså en stor fördel ur standardiseringssynpunkt. Ett Basicprogram kan köras på i stort sett vilken dator som helst, utan större modifieringar. För det andra är maskinkodrepertoaren synnerligen begränsad. När man som novis studerar instruktionslistan, som beskriver maskinkoderna hos en mikroprocessor, reser sig förmodligen nackhåren i pur förskräckelse. Det visar sig att mikroprocessorn nästan inte kan göra någon nytta alls. Den klarar bara av särdeles triviala uppgifter i stil med "addera talet ett till ett 8-bitars binärt tal", "subtrahera ett", "addera två 8-bitarstal", (detta är bravurnumret), "skifta ett 8-bitarstal ett steg åt vänster", "flytta ut ett 8-bitarstal från mikroprocessorn till en viss adress i minnet" o s v. Ingen multiplikation eller division! Inga trigonometriska funktioner!

Vid närmare analys visar det sig dock, att man, genom kombination av koder ur instruktionslistan, kan göra alla tänkbara operationer. Multiplikation kan t ex utföras genom en serie skiftningar och additioner (så som man gör när man multiplicerar "för hand" med papper och penna), division likaså. Den arbetsinsats som krävs för att skriva ett maskinkodsprogram, som multiplicerar t ex två 9-siffriga decimaltal är dock jämförelsevis enorm, även en van maskinkodsprogrammerare skulle få jobba åtskilliga timmar för att få ett sådant program att fungera! Under datorvetenskapens barndom (1940-talet) skrevs i princip alla datorprogram i maskinkod, av lärde män i vita rockar. Så småningom växte olika högnivåspråk fram (ALGOL, COBOL, FORTRAN, BASIC etc), vilket medförde att datorer äntligen kunde programmeras av icke-specialister.

Vid det här laget börjar du säkert undra varför det finns ett helt kapitel om maskinkod i den här boken. Lugn, förklaringen kommer här!

Maskinkod och Basic

Under vissa betingelser kan det faktiskt vara befogat att skriva små korta maskinkodsprogram istället för ett (förmodligen ännu kortare) Basicprogram. Den huvudsakliga anledningen är krav på snabbhet vid programkörningen och då speciellt vid kommunikation med yttre enheter. Det här är en situation som ofta uppstår vid styrning och mätning med hjälp av datorn. Som du vet vid det här laget tar det ca 1 ms för Basictolken att översätta och utföra en rad i ett Basicprogram. Vill du t ex generera ett pulståg på en port med hjälp av POKE-kommandon, blir frekvensen därmed begränsad till några hundra Hz, på sin höjd. Ett enkelt maskinkodsprogram kan däremot generera ett pulståg i 50 kHz-området, om så önskas.

Vid datainsamling från en snabb A/D-omvandlare kan ett nytt mätvärde föreligga var tjugonde mikrosekund. Ett maskinkodsprogram för SV-318 och SV-328 klarar med god marginal av att samla in alla dessa data och placera dem i en tabell i minnet. Ett basicprogram däremot, skulle bara hinna med vart femhundrade mätvärde, resten skulle gå till spillo. En kombination av maskinkod och Basic blir i det närmaste oslagbar! Använd maskinkod för de tidskänsliga avsnitten och Basic för de beräkningsintensiva.

Det tar förvisso rätt lång tid att bli en fullfjädrad maskinkodsexpert, men de grundläggande begreppen är inte så besvärliga att förstå. I de följande avsnitten ska vi titta närmare på hur man gör.

Att programmera i maskinkod på SV-318 och SV-328

Instruktionslistan för mikroprocessorn Z80 i din SV-318 eller SV-328 omfattar i huvudsak 87 olika grundoperationer. Flera av dem kan utnyttja olika varianter, s k adresseringsmetoder, som vi ska titta närmare på om en stund. Totalt utnyttjas 276 olika koder för att beskriva vilken operation som ska utföras. Varje sådan kod kallas *operationskod,* eller kortare *op-kod,* och utgörs av ett eller ibland två 8-bitars binära tal, d v s decimaltal mellan 0 och 255. Instruktionslistan för Z80 med alla dessa koder hittar du i Appendix B. För en komplett detaljerad instruktionslista, som ingående beskriver varje op-kod, hänvisas du dock till fabrikantens datablad.

En god maskinkodsprogrammerare måste känna till alla små intrikata detaljer hos varenda en av de 87 grundoperationerna och deras varianter och detta tar förstås sin tid att bemästra. Om vi begränsar oss till styrning och mätning med datorn och dessutom utnyttjar Basictolken för alla beräkningar, krymper antalet nödvändiga instruktioner till en handfull enkla koder!

Ackumulatorn

De två viktigaste instruktionerna är "Ladda ackumulatorn" och "Lagra ackumulatorn". Ackumulatorn är ett åtta-bitars register inne i processorn, som är inblandat i nästan alla operationer. Om t ex två tal ska adderas måste det ena talet från början finnas i ackumulatorn. Resultatet hamnar också i ackumulatorn. Om du vill lagra information i en minnescell (eller en utport) måste informationen flyttas dit från ackumulatorn.

Memokoder

I instruktionslistan återfinner du en förkortning för varje instruktion, bestående av två, tre eller fyra bokstäver och förstås motsvarande op-kod. Förkortningen kallas *memokod* eller *symbolisk kod* (ibland mnemonic) och är avsedd att vara ett stöd för programmerarens minne, när han/hon skriver program. Förkortningarna är ganska listigt valda, speciellt om man har engelska som modersmål. Memokoden för "Ladda" är LD (från *LoaD),* för "Addera" ADD, "Subtrahera" SUB o s v.

Att anropa maskinkodsprogram

Det finns en BASIC-funktion, som gör det möjligt att anropa maskinkodsprogram från ett Basic-program, USR N(I). N är ett tal mellan 0 och 9 och anger vilken av upp till tio maskinkodsprogram som ska köras. Argumentet I är ett uttryck eller en variabel. Det är nämligen möjligt att skicka variabler från ett Basicprogram till maskinkodsprogrammet, utföra beräkningar och få tillbaka nya variabelvärden till Basic-programmet igen. Innehållet i ackumulatorn A specificerar typ av argument I. A=2 är koden för heltal, A=3 sträng, A=4 flyttal med enkel precision och A=8 flyttal med dubbel precision. I det här kapitlet ska vi inte utnyttja denna facilitet, men vi måste ändå tillordna I något värde, t ex I=2.

Det maskinkodsprogram, som ska anropas med USR N(I), måste finnas inskrivet i minnet i konsekutiva adresser (d v s på varandra följande adresser). Dessutom måste vi definiera maskinspråksprogrammets startadress med Basic-instruktionen DEFUSRN. Vidare är det ett krav att maskinkodsprogrammet måste sluta med en speciell instruktion, memokod RET (opkod 201), för att datorn ska hitta tillbaka till Basic igen.

Låt oss nu titta på hur det går till att ladda ackumulatorn med ett tal. Z80 innehåller inte mindre än 22 register, varav ackumulatorn utgör ett. För att du ska kunna hämta nödvändig information ur instruktionslistan i appendix B, måste vi först klargöra hur listan är uppställd.

Instruktionslistan

Instruktionslistan ser ganska avskräckande ut vid första anblicken. Den är dock ganska välstrukturerad, med elva underrubriker och med lite övning går det att hitta rätt i den utan alltför mycket möda.

Följande avdelningar återfinns i listan:

Engelsk rubrik	Betydelse
8-Bit Load Group	Laddning av 8-bitars register.
16-Bit Load Group	Laddning av 16-bitars register.
Exchange, Block Transfer Block Search Groups	Utbyte av data mellan register, överföring av datablock, sökning av datablock.
8-Bit Arithmetic and Logical Groups	8-bitars aritmetiska och logiska funktioner.
General-Purpose Arithmetic and CPU Control Groups	Speciella aritmetiska funktioner och styrkommandon för CPU:n.
16-Bit Arithmetic Group	16-bitars aritmetiska funktioner.
Rotate and Shift Group	Rotation och skiftning av registerinnehåll.
Bit Set, Reset and Test Group	Ettställning, nollställning och test av enskilda bitar i register.
Jump Group	Hopp i program.
Call and Return Group	Subrutinhantering.
Input and Output Group	In- och utmatning via portar.

Ladda ackumulatorn

Om vi vill ladda ackumulatorn med ett visst tal, är det följaktligen ingen större idé att leta någon annanstans än under "8-Bit Load Group"! Antag att vi vill ladda ackumulatorn med talet 128. Eftersom talet 128 gott och väl ryms i ett 8-bitarsregister, (ända upp till 255 går in i ett 8-bitarsregister), tittar vi närmare på "8-Bit Load Group" och hittar där 21 olika LD-operationer: (se sid 58 i Appendix B).

8-Bit Load Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
LD r, r'	r ← r'	•	•	X	X	•	•	•	01 r r'		1	1	4	r, r' Reg.
LD r, n	r ← n	•	•	X	X	•	•	•	00 r 110		2	2	7	000 B
									← n →					001 C
LD r, (HL)	r ← (HL)	•	•	X	X	•	•	•	01 r 110		1	2	7	010 D
LD r, (IX+d)	r ← (IX+d)	•	•	X	X	•	•	•	11 011 101	DD	3	5	19	011 E
									01 r 101					100 H
									← d →					101 L
LD r, (IY+d)	r ← (IY+d)	•	•	X	X	•	•	•	11 111 101	FD	3	5	19	111 A
									01 r 110					
									← d →					
LD (HL), r	(HL) ← r	•	•	X	X	•	•	•	01 110 r		1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	X	X	•	•	•	11 011 101	DD	3	5	19	
									01 110 r					
									← d →					
LD (IY+d), r	(IY+d) ← r	•	•	X	X	•	•	•	11 111 101	FD	3	5	19	
									01 110 r					
									← d →					
LD (HL), n	(HL) ← n	•	•	X	X	•	•	•	00 110 110	36	2	3	10	
									← n →					
LD (IX+d), n	(IX+d) ← n	•	•	X	X	•	•	•	11 011 101	DD	4	5	19	
									00 110 110	36				
									← d →					
									← n →					
LD (IY+d), n	(IY+d) ← n	•	•	X	X	•	•	•	11 111 101	FD	4	5	19	
									00 110 110	36				
									← d →					
									← n →					
LD A, (BC)	A ← (BC)	•	•	X	X	•	•	•	00 001 010	0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	X	X	•	•	•	00 011 010	1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	X	X	•	•	•	00 111 010	3A	3	4	13	
									← n →					
									← n →					
LD (BC), A	(BC) ← A	•	•	X	X	•	•	•	00 000 010	02	1	2	7	
LD (DE), A	(DE) ← A	•	•	X	X	•	•	•	00 010 010	12	1	2	7	
LD (nn), A	(nn) ← A	•	•	X	X	•	•	•	00 110 010	32	3	4	13	
									← n →					
									← n →					
LD A, I	A ← I	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
									01 010 111	57				
LD A, R	A ← R	↕	↕	X	0	X	IFF	0	•	11 101 101	ED	2	2	9	
									01 011 111	5F				
LD I, A	I ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
									01 000 111	47				
LD R, A	R ← A	•	•	X	•	X	•	•	•	11 101 101	ED	2	2	9	
									01 001 111	4F				

NOTES: r, r' means any of the registers A, B, C, D, E, H, L.
IFF the content of the interrupt enable flip-flop, (IFF) is copied into the P/V flag.
*For an explanation of flag notation and symbols for mnemonic tables, see Symbolic Notation section following tables.

Vilken ska vi välja? Ja, först är det nog nödvändigt att vi tittar på vad de olika förkortningarna betyder. Låt oss försöka dechiffrera t ex första raden:

8-Bit Load Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
LD r, r'	r ← r'	•	•	X	X	•	•	•	01 r r'		1	1	4	r, r' Reg.
														000 B
														001 C
														010 D
														011 E
														100 H
														101 L
														111 A

Vad memokoden "LD r,r'" egentligen står för förklaras under "Symbolic Operation" med det kryptiska r←r´. r och r´ är allmänna beteckningar för något av registren A,B, C, D, E, H och L, står det i fotnoten till "8-Bit load Group". Pilen betyder "kopieras till". r←r´ betyder alltså att innehållet i ett register kopieras till ett annat. Om vi t ex skulle vilja kopiera över innehållet i register B till ackumulatorn (d v s till register A) skulle memokoden skrivas "LD A,B", d v s "ladda A från B":

Tittar vi vidare på "LD r,r´" återfinns under "Flags" uppgift om hur de sex flaggorna i processorns F-register påverkas av instruktionen. Flaggorna är av intres-

se bl a vid villkorliga hopp, något som vi inte alls ska gå in på här. Hur som helst, en punkt under respektive flagga innebär att flaggan i fråga inte påverkas alls av instruktionen. Tydligen påverkar inte "LD r,r´"någon flagga!

Under nästa rubrik "Opcode" (operationskod på svenska) går att utläsa hur instruktionskoden ser ut i binär form. Eftersom Z80 är en 8-bitarsprocessor "tuggar" den ju i sig information en "byte" d v s 8 bitar, åt gången. En instruktion i Z80-kod kan bestå av en, två, tre eller fyra "byte", beroende på vilken instruktion det handlar om. Detta framgår av nästa rubrik, "No of Bytes". "LD r,r´" består tydligen av bara en enda byte, som enligt listan ska vara "01 r r ´". Det ser ju knappast binärt ut vid första anblicken, men det är faktiskt inte så konstigt som det ser ut! Enligt "Comments" är r och r´ förkortningar för trebitars koder. I LD A,B motsvaras A av koden 111 och B av 000 och följaktligen är operationskoden för LD A,B: 01111000. Om koden 01111000 skrivs in i datorns minne och processorn kan fås att utföra denna instruktion, kommer innehållet i register B att kopieras över till register A. Enkelt, inte sant! Den enda riktiga haken är förstås att vi människor, till skillnad från datorer, inte är speciellt duktiga på att jobba med binära tal. Efter att du har studerat åttabitars binära tal i två timmar, har de förmodligen en viss tendens att se likadana ut, det är lätt att göra fel och trötthet kommer smygande. Det är mycket behändigare att översätta varje åttabitarstal till motsvarande decimaltal och i appendix C hittar du en tabell för detta ändamål. Enligt tabellen motsvaras 01111000 av decimaltalet 120. "No of T-states" anger slutligen hur lång tid instruktionen tar uttryckt i klockperioder.

Enda återstående problemet är förstås att vi inte alls hade tänkt kopiera innehållet från ett register till ackumulatorn, vi vill ju ladda ackumulatorn med ett visst åttabitars tal! Nästa rad i tabellen innehåller lösningen på problemet:

8-Bit Load Group	Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
	LD r, r'	r ← r'	•	•	X	•	X	•	•	•	01 r r'		1	1	4	r, r' Reg.
	LD r, n	r ← n	•	•	X	•	X	•	•	•	00 r 110		2	2	7	000 B
											← n →					001 C
																010 D
																011 E
																100 H
																101 L
																111 A

Under "Mnemonic" läser vi "LD r,n". I förklaringen längst bak i appendix B kan du läsa att "n" är ett 8-bitarstal, med decimal motsvarighet mellan noll och 255. Vill vi ladda ackumulatorn (d v s register A) med talet 128 skrivs memokoden följaktligen LD A,128. Totalt består instruktionen av två "byte", som ska skrivas i två på varandra följande minnesadresser. Första "byten" är 00111110, vilket motsvarar det decimala talet 62 (se appendix C). Andra "byten" är talet 128.

Utmatning

Låt oss nu anta att vi vill mata ut innehållet i ackumulatorn (talet 128) på användarporten, port B, adress 0. Vi förutsätter att port B gjorts till utgångar enligt kapitel 2.

Avsnittet "Input and Output Group" i instruktionslistan verkar rimligt att titta i, inte sant? Den första OUT-instruktionen vi träffar på är "OUT (n),A". (n) innebär att datorn tolkar talet n som adressen till den port, där ackumulatorinnehållet ska matas ut. I vårt fall är adressen 0 och instruktionen skrivs "OUT (0),A". Instruktionen består av två byte, den första är 11010011, d v s decimalt 211, och den andra av 0.

Input and	OUT (n), A	(n) ← A	•	•	X	•	X	•	↕	X	11 010 011 ← n →	D3	2	3	11	n to A_0 ~ A_7 Acc. to A_8 ~ A_{15}
Output Group	OUT (C), r	(C) ← r	•	•	X	•	X	•	↕	X	11 101 101 01 r 001	ED	2	3	12	C to A_0 ~ A_7 B to A_8 ~ A_{15}

Vårt program går nu nästan att köra på datorn, men en viktig detalj fattas ännu. Enligt vad som tidigare sagts måste maskinkodsprogram, som anropas från Basic, avslutas med instruktionen "RET" (som f ö återfinns under "Call and Return Group" i instruktionslistan). RET består av en enda byte, koden är 11001001, d v s decimalt 201.

Programmet

Memokoden är bara av intresse för programmeraren och givetvis ur dokumentationssynpunkt. Maskinkoderna, som ska skrivas in i minnet, i på varandra följande adresser, blir 62, 128, 211, 0, 201. Ofta skriver man maskinkoderna i anslutning till memokoderna för att underlätta kontroll:

```
62, 128          LD A, 128
211, 0           OUT (0), A
201              RET
```

Programmet kommer att ta 7 + 11 + 10 = 28 klockperioder att utföra. I SV-318 och SV-328 är klockfrekvensen 3,58 MHz, d v s varje klockperiod är 1/3,58= 0,279 μs. Programmet tar alltså 28x0,279 = 7,82 μs att utföra.

Att mata in program i minnet

Det är lämpligt att placera maskinkodsprogram med början i adress 64398. Vårt lilla program kan matas in i minnet med fem POKE-kommandon:

```
POKE 64398, 62
POKE 64399, 128
POKE 64400, 211
POKE 64401, 0
POKE 64402, 201
```

Observera att koderna *måste* placeras i på varandra följande adresser! Enklare är dock att använda en slinga, som matar in koderna från en DATA-sats, speciellt vid lite längre program:

```
20 FOR I = 64398 TO 64402
30 READ A
40 POKE I, A
50 NEXT I
60 DATA 62, 128, 211, 0, 201
```

Provkör detta inmatningsprogram och titta sedan i minnet med PRINT PEEK (64398) o s v, och kontrollera att de önskade koderna verkligen står i de avsedda adresserna.

Datorn utför programmet

Det är nu dags för sammanfogning av alla bitar. För att kunna testa programmet behöver du koppla in lysdioder till användarporten (PB0—PB7), som vi studerat tidigare. Vårt Basic-program måste ställa om porten till en utport med åtta utgångar med OUT 2,255.

Vidare måste maskinkodsprogrammet skrivas in, t ex med slingan i föregående avsnitt. Slutligen ska maskinkodsprogrammets startadress anges med DEFUSR N och programmet startas med X=USR N(I).

```
10 OUT 2, 255
20 FOR I = 64398 TO 64402
30 READ A
40 POKE I,A
50 NEXT I
60 DATA 62, 128, 211, 0, 201
70 DEFUSR 1 = 64398
80 X = USR 1(2)
```

Provkör programmet! Om allt är i sin ordning ska lysdioden, ansluten till PB7 tändas. (Vi laddade ju ackumulatorn med $128_{10}=10000000_2$). Prova gärna med att ladda ackumulatorn med något annat tal, genom att byta ut talet 128 i DATA-satsen på rad 60.

Generering av pulståg i maskinkod

Föregående program matade ut informationen till port B "blixtsnabbt" men tyvärr finns det ingen möjlighet för oss att kontrollera sanningshalten i detta påstående. Om du har tillgång till ett oscilloskop, kan vi dock modifiera programmet, så att ett pulståg genereras. Då kommer snabbheten fram! Vi ska modifiera maskinkodsprogrammet på följande sätt:

```
LD A, 128
OUT (0), A      Matar ut "1" på PB7
LD A, 0
OUT (0), A      Matar ut "0" på PB7
JP 64398
```

JP-instruktionen åstadkommer ett hopp tillbaka till början av maskinkodsprogrammet, varför resultatet blir ett pulståg på PB7. JP är en ny instruktion för oss och återfinns under "Jump Group":

Jump Group	JP nn	PC ← nn	•	•	X	•	X	•	•	•	11 000 011 ← n → ← n → C3	3	3	10
	JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	X	•	X	•	•	•	11 cc 010 ← n → ← n →	3	3	10

cc	Condition
000	NZ non-zero
001	Z zero
010	NC non-carry
011	C carry
100	PO parity odd
101	PE parity even
110	P sign positive
111	M sign negative

JP nn är tre "byte" lång och första "byten" ska tydligen vara 11000011, d v s 195 decimalt. Resterande två "byte" ska utgöras av adressen dit hoppet ska ske, d v s 64398. Nu är det tyvärr lite krångligt att dela upp adressen i två "byte". Så här kan man göra:

1 Översätt adressen 64398 till motsvarande binära tal. Efter en del räknande får du då fram det binära talet 1111101110001110.

2 Dela upp detta tal i två åttabitarshalvor, d v s 11111011 och 1000 1110.

3 Översätt dessa två binära 8-bitarstal till decimaltal igen. 1111 1011 blir 251, 1000 1110 blir 142.

4 *Skifta ordning* på de två halvorna (detta är en liten besynnerlighet hos Z80-processorn). Första talet ska alltså vara 142, andra 251.

Den kompletta instruktionen JP 64398 omfattar alltså tre tal, lagrade i tre på varandra följande minnesadresser: först opkoden 195, sedan talet 142 och sist talet 251.

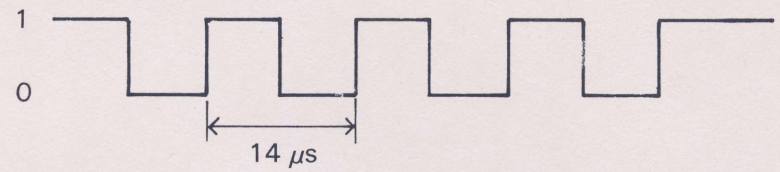


Fig. 5.1 Pulståget på PB7 kan beskådas med ett oscilloscop.

Periodtiden blir 14 μs! I Basic, med OUT-instruktioner, skulle det faktiskt ha tagit ca 17 ms, d v s mer än tusen gånger långsammare! Innan vi provkör måste dock programmet översättas till maskinkod, eller *assembleras,* som man ofta säger:

```
62, 128          LD A, 128
211, 0           OUT(0),A
62, 0            LD A,0
211, 0           OUT(0),A
195, 142, 251    JP 64398
```

Adressen 64398 i JP-instruktionen blir 142, 251, i uppdelad form som vi redan räknat tidigare. Inmatningsprogrammet måste ändras lite i raderna 20 och 60.

```
20 FOR I = 64398 TO 64408
60 DATA 62, 128, 211, 0, 62, 0, 211, 0, 195, 142, 251
```

Det färdiga programmet ser alltså ut så här

```
10 OUT 2, 255
20 FOR I = 64398 TO 64408
30 READ A
40 POKE I,A
50 NEXT I
60 DATA 62, 128, 211, 0, 62, 0, 211, 0, 195, 142, 251
70 DEFUSR 1 = 64398
80 X = USR 1(2)
```

Men stopp ett tag! Vårt maskinkodsprogram slutar ju inte med RET, som kravet var! Nej, eftersom programmet är en oändlig slinga kan vi inte avsluta det och resultatet blir att datorn inte hoppar tillbaka till Basic-tolken. Detta för med sig att pulståget inte kan stoppas. Enda sättet att få hejd på maskinen är faktiskt att slå av nätspänningen varvid ditt mödosamt inmatade program givetvis försvinner.

Sens moral:

1 Spela in ditt program på band eller flexskiva först.
2 Avsluta dina maskinkodsprogram med RET, om du vill komma tillbaka till Basictolken.

Trots alla besvär är du nog övertygad om snabbheten hos maskinkodsprogram vid det här laget, inte sant?

6. Några projekt

Några Projekt

I det här kapitlet ska vi studera ett par lite större uppgifter, där du själv får bygga vidare på den information som ges och där enbart din egen kreativitet sätter en gräns! Uppgifterna kombinerar både hård- och mjukvara (så är det ju normalt i verkliga livet). När du jobbat igenom dessa projekt kan du verkligen med fog känna att du behärskar din dator när det gäller mätning och styrning.

Datalogger

En datalogger är ett instrument som läser och lagrar mätvärden från en eller flera givare vid bestämda tidpunkter. Lagringen kan t ex ske genom att mätvärdena skrivs ut av en printer på en pappersremsa. Alternativt kan mätvärdena temporärt lagras i minnet på en dator för senare överföring till band eller flexskiva. Det är nämligen mycket behändigt att ha mätdata samlade i en datafil om vidare behandling av mätvärdena ska göras, såsom statistiska beräkningar, trendanalys etc.

I det här avsnittet får du bygga upp en datalogger kring en analog multiplexer och en V/f-omvandlare, enligt vad som beskrivits i ett tidigare avsnitt. Dataloggern ska användas för temperaturmätning i upp till åtta mätpunkter. Som givare använder vi vanliga kiseldioder, 1N4148. Framspänningsfallet över en diod minskar nämligen linjärt med temperaturen, med ca 2 mV/°C, över ett stort temperaturintervall (-40 °C - +150 °C). Framspänningsfallet över en kiseldiod varierar lite från diod till diod. Om man vill, kan man förstås "matcha" dioder genom att välja ut dioder med samma framspänningsfall. Med en dator till förfogande är detta onödigt, eventuella avvikelser mellan olika dioder går att ta hänsyn till programmässigt, via en kalibreringsprocedur.

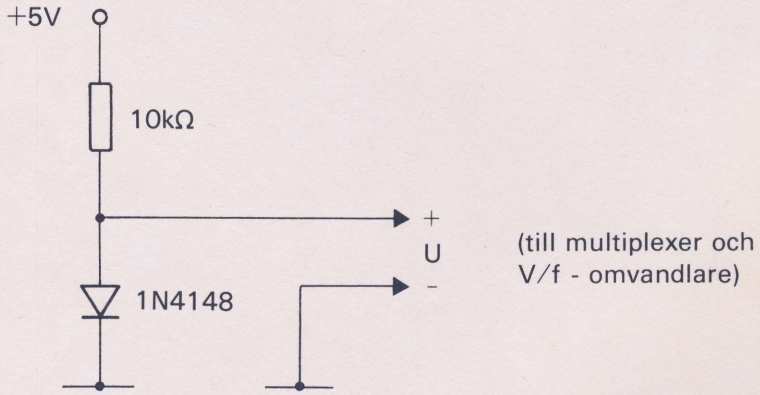


Fig. 6.1 Dioden kan anslutas på detta sätt.

Strömmen genom dioden blir ca 450 μA och spänningen över den ca 0,55 V vid +20°C. Värms dioden upp till + 100°C blir spänningen ca 0,39 V, alltså en ganska liten ändring. Dessutom minskar spänningen då temperaturen ökar, det skulle onekligen vara trevligare om den ökade med temperaturen! Med en bryggkoppling och en differentialförstärkare kan vi slå två flugor i en smäll, dels förstärka spänningsändringen till anständig nivå (ca 5 ggr) dels få en spänning som ökar med temperaturen, figur 6.2.

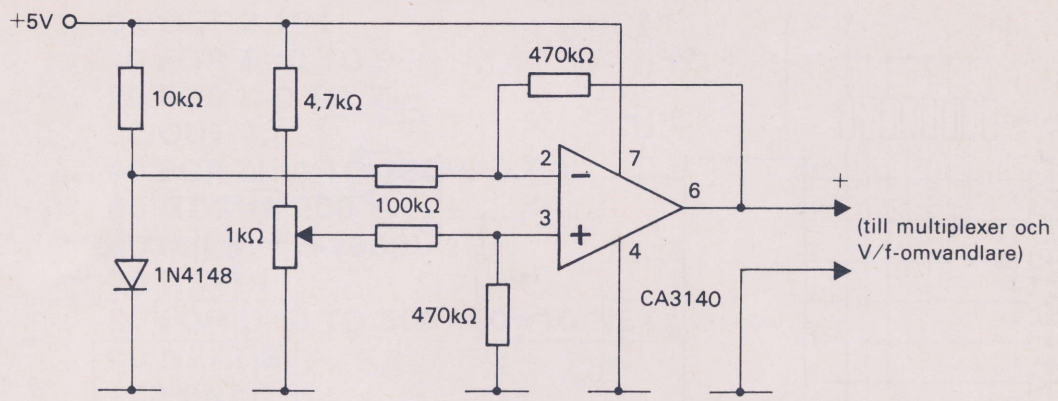


Fig. 6.2. Förstärkare till temperaturgivare

Det är lämpligt att isolera dioden med t ex krympslang, om den ska kalibreras genom att doppas i vattenbad av känd temperatur, eftersom vatten kan ha ganska god elektrisk ledningsförmåga. Vid ett kalibreringsförsök med ovanstående koppling sänktes dioden ned i smältande is (0°C) och 1 kΩ-potentiometern justerades, så att 0°C motsvarade U=+1,0 V. Vid +23°C erhölls då U=+1,23 V och vid +71°C U =+1,71 V, d v s ett linjärt samband mellan utspänning och temperatur. Graf över sambandet mellan utspänning och temperatur återfinnes i figur 6.3. Detta är ett utmärkt spänningsintervall för V/f-omvandlaren vi studerade i ett tidigare avsnitt.

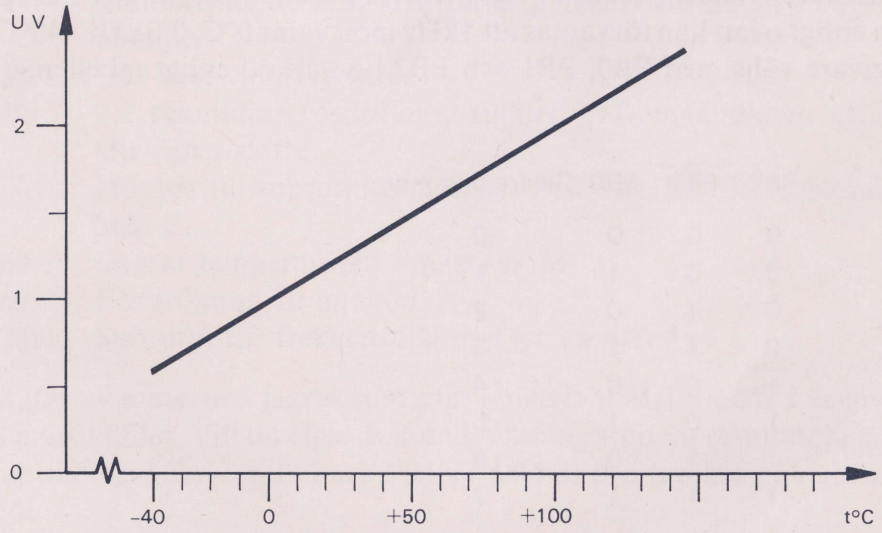


Fig. 6.3 Samband mellan utspänning och temperatur

Komplett utbyggd, med alla åtta givarna, multiplexer och V/f-omvandlaren, ser datainsamlingsenhetens kopplingsschema ut som visas i figur 6.4.

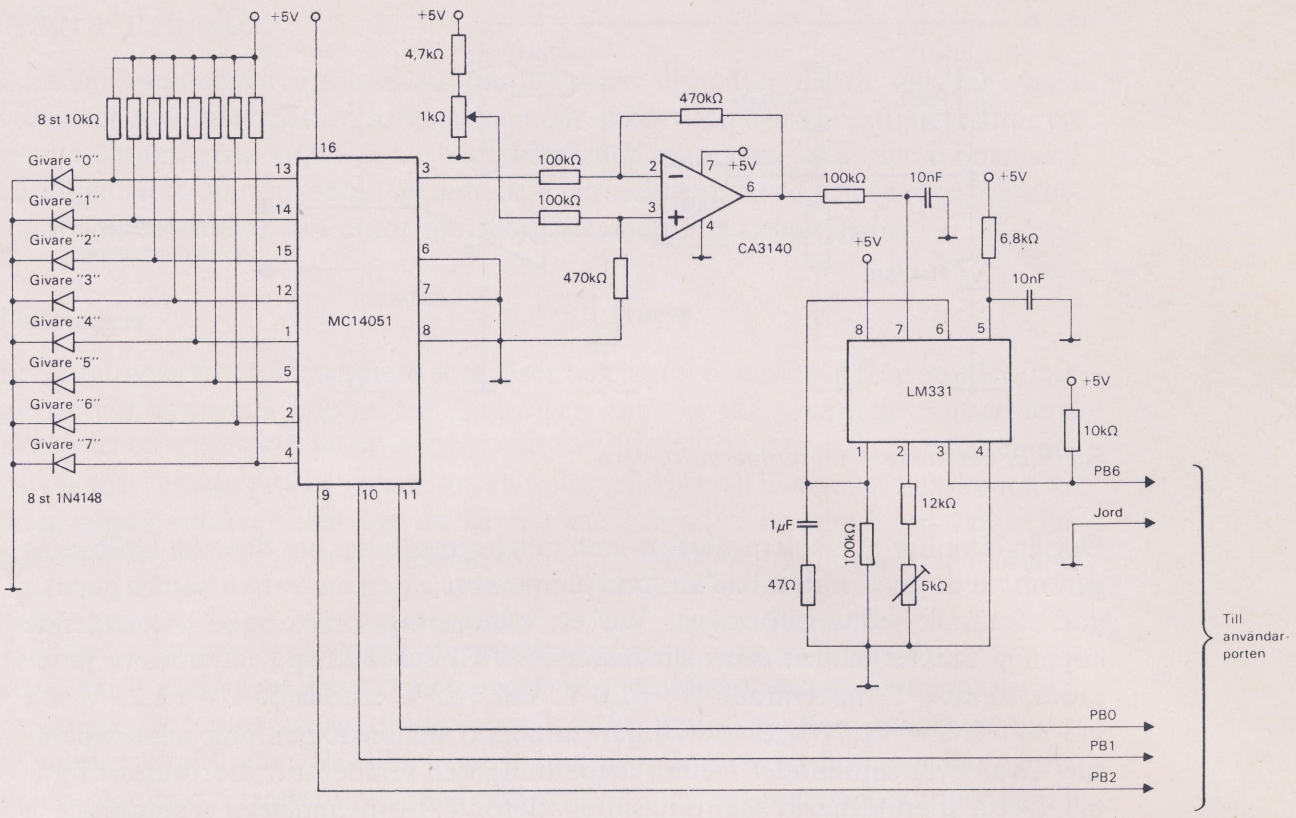


Fig. 6.4 8-kanalers datalogger

V/f-omvandlarens känslighet är 1 kHz/V varför det efter justering av 1 kΩ potentiometern enligt ovan kan förväntas att 1kHz motsvarar 0°C, 0,6 kHz –40°C o s v. Önskad givare väljs med PB0, PB1 och PB2 i binärkod enligt tabellen:

PB2	PB1	PB0	Givare nummer
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Då är vi nästan klara att titta på ett programexempel, men först en liten notis. När multiplexern kopplar in en ny givare, tar det ca 0,2 sekunder för V/f-omvandlaren att ställa in sig på den nya frekvensen. Det är alltså nödvändigt att lägga in fördröjningar i programmet för att ta hänsyn till detta.

Ett program för SV-318 eller SV-328 som lagrar 10 mätvärden från var och en av de 8 givarna var tionde minut, följer här. Vi utnyttjar frekvensmätningsprogrammet från kapitel 4 som subrutin.

```
5  OUT 2,191
10 FOR S=0 TO 9
20 FOR I=0 TO 7
30 OUT 0,I
40 FOR N=0 TO 100:NEXT N
50 GOSUB 200
60 M(I,S)=(F-1000)/10
70 NEXT I
80 FOR U=0 TO 500*60*10:NEXT U
90 NEXT S
100 END
200 OUT 11,INP(11) OR 32
210 OUT 2, 191
220 OUT 8, 255
230 T = TIME
240 IF T = TIME THEN 240
250 OUT 9, 255
260 IF TIME <>T + 50 THEN 260
270 F = 65535-(256*INP(9)+INP(8))
280 RETURN
```

Rad 10: Avgränsar tillsammans med rad 90 en FOR..NEXT-slinga, som åstadkommer mätning vid 10 tillfällen. Variabeln S anger vilket mättillfälle som är för handen.

Rad 20: Avgränsar tillsammans med rad 70 en FOR..NEXT-slinga, som väljer ut en av de åtta givarna åt gången. Variabeln I anger givarens nummer.

Rad 30: Här sker val av givare.

Rad 40: 0,2 sekunders fördröjning tillåter V/f-omvandlaren att svänga in efter givarskifte.

Rad 50: Hoppar till subrutin som mäter frekvens. Resultatet hamnar i variabeln F.

Rad 60: Lagrar temperaturen i matrisen M.

Rad 80: Fördröjning 10 minuter.

Rad 200: Subrutin för frekvensmätning (se kapitel 4).

Resultatet av mätserien lagras som sagt i matrisen M(I,S), där I är givarnumret och S mättillfället. Vill du efter avslutad mätning studera resultatet i mätpunkt 5 vid mättillfälle 3 skriver du bara PRINT M(5,3) och trycker på return, så får du besked!

Några förslag till vidareutveckling:

1 Generera utskrift på skärmen av varje mätserie när en sådan är klar.
2 Plotta resultatet på skärmen, (lämpligen gör du flera mätningar).
3 Arrangera så att du kan välja ut bara några av givarna, t ex 3, 4 och 7! Valet ska kunna ändras på ett enkelt sätt.
4 Bygg ut programmet så att efter avslutad mätning alla mätvärden lagras på kassett eller flexskiva.
5 Skriv ett program som behandlar mätvärden, lagrade på band eller flexskiva och t ex beräknar medelvärdet för varje givare.
6 Hur kan flera givare kopplas in?

7 Det går givetvis att lagra andra mätvärden än temperaturer med en datalogger. En intressant vidareutveckling är att göra en liten väderstation. Åtminstone vindriktning och vindhastighet borde vara lätta att mäta, (vindflöjel + potentiometer eller ev. optiska givare, propeller med optisk givare eller ev. magnet + reedrelä). Luftfuktighet och lufttryck är kanske svårare, men inte omöjligt.

Temperaturreglering

En temperaturregulator är en anordning som håller önskad temperatur i t ex en process eller i ett rum. I det här avsnittet ska datorn få fungera som ON/OFF-regulator för en ugn, d v s den håller önskad temperatur i ugnen genom att slå till eller från värmen i ugnen. Temperaturen mäts med en termistor och mätvärdena talar om för datorn om värmen ska slås till eller från. Det finns förstås mera raffinerade metoder för reglering, men trots allt är det här en vanlig typ.

Som ugn använder vi helt enkelt en pappkartong med en vanlig 220 V, 60 W glödlampa som värmeelement. För att datorn ska kunna slå till och från värmeelementet fordras ett relä, lämpligen ett halvledarrelä av den typ, som användes i kapitel 3 figur 6.5. sid 22.

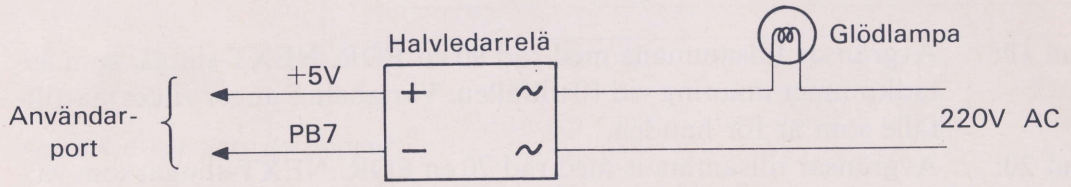


Fig. 6.5 Kopplingsschema för temperaturreglering

Observera att PB7 = "1" *släcker* lampan, PB7 = "0" *tänder* den. För att mäta temperaturen inne i "ugnen" använder vi en termistor, kopplad till en Schmitttriggeroscillator enligt kapitel 4.

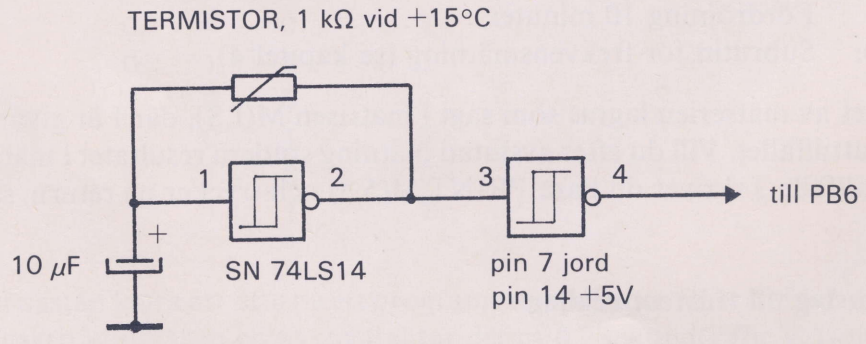


Fig. 6.6 Temperaturen mäts med en termistor

Fig. 6.5 och 6.6 visar all hårdvara som behövs! Var försiktig med 220V-anslutningarna – se till att allt är välisolerat, (och rätt kopplat), innan du kopplar in nätspänning. Varken du själv eller datorn mår bra av att komma i kontakt med nätspänningen.

För temperaturmätningen använder du lämpligen programmet i kapitel 4, som även lineariserar termistorn. Programmet anropas som subrutin, rad 100. Ett program som håller önskad temperatur, *börvärdet,* +50°C, kan sålunda se ut så här:

```
10 OUT 2, 128
20 OUT 0, 128
30 BV=50
40 GOSUB 100
50 IF X>BV THEN 20
60 OUT 0, 0
70 GOTO 40
100 OUT 11, INP(11) OR 32
110 OUT 2, 191
120 OUT 8, 255
130 T=TIME
140 IF T=TIME THEN 140
150 OUT 9, 255
160 IF TIME<>T+50 THEN 160
170 N=65535-(256*INP(9)+INP(8))
180 IF F<120 THEN X=-15+N*0.6 : GOTO 200
190 X=20+N*0.3
200 PRINT X
210 RETURN
```

Kommentarer till ovanstående program:

Rad 10 Ställer PB7 på användarporten som utgång.

Rad 20 Ställer PB7 = "1", d v s slår från värmeelementet.

Rad 30: Börvärdet +50°C väljs.

Rad 40: Hoppa till subrutin för temperaturmätning. Resultatet hamnar i variabeln X.

Rad 50: Datorn testar om temperaturen X är för hög. I så fall sker återhopp till rad 20, där värmeelementet slås från.

Rad 60: Om temperaturen är för låg, sätts PB7 = "0", d v s värmeelementet slås till.

Rad 70: Hoppa tillbaka till 40 och mät temperaturen igen.

Rad 100: Mäter och skriver temperaturen (subrutin).

Observera, att uttrycken för linearisering förmodligen måste ändras, så att de stämmer för just din termistor, se kapitel 4. Provkör programmet med några olika börvärden och kontrollera att din temperaturregulator verkligen fungerar!

Förslag till vidareutveckling: En fördel med datorn, jämfört med en enkel ON/OFF-regulator (som strängt taget kan byggas med en operationsförstärkare), är att man med lätthet kan låta börvärdet ändras med tiden, enlig en i förväg uppgjord temperaturprofil. Ändra programmet så att temperaturen +40°C hålls under 2 minuter, 250°C under 3 minuter! Tips: använd en DATA-sats för att specificera temperatur och tid och utnyttja realtidsklockan för tidmätning. En ytterligare utveckling kan vara att du ändrar programmet, så att det kan hantera *två* olika ugnar samtidigt!

Styrning av stegmotor

För bara några år sedan gick det åt ganska mycket elektronik för att styra en stegmotor. Idag finns det IC-kretsar, som genererar de flerfassignaler en stegmotor måste styras med och som dessutom innehåller nödvändiga drivsteg för direktanslutning till motorlindningarna.

I det här avsnittet ska vi styra en liten stegmotor från Philips, 9904 112 07005, via en drivkrets SAA1027, också den från Philips. Motorn och drivkretsen fordrar +12V matningsspänning, som du måste använda ett separat nätaggregat för. Drivkretsen har två digitala ingångar för styrning av motorn, den ena ingången styr rotationsriktningen och den andra stegar motorn, ett steg per puls. Båda styrsignalerna är i princip enkla att generera från datorn. Enda svårigheten är att drivkretsen SAA1027 fordrar +12V för "1" och 0 V för "0", medan datorn bara genererar +5V för "1". Detta är dock inget stort problem, en TTL-krets, SN7407, mellan datorn och drivkretsen sköter om nivåskiftningen, figur 6.7

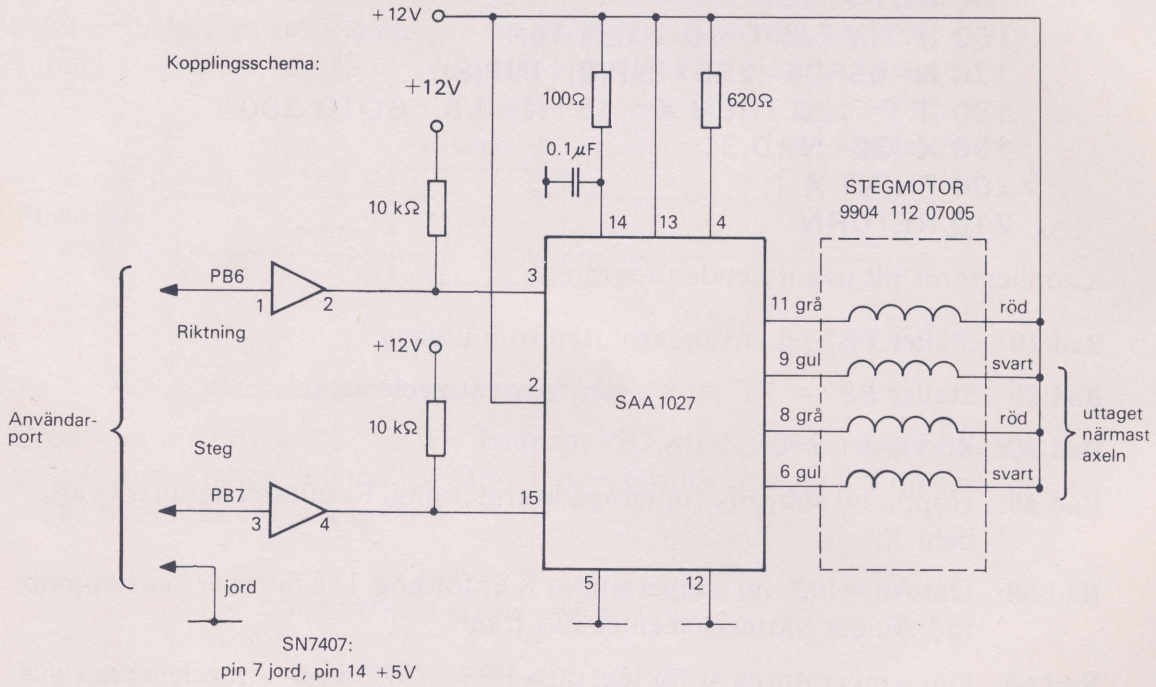


Fig. 6.7 Drivelektronik för stegmotor

Ett program, som låter datorn mata N pulser, i riktningen R, med konstant pulsfrekvens, kan se ut som visas här nedan.

```
10 OUT 2, 192
20 N = 45
30 R = 64
40 FOR X = 1 TO N
50 OUT 0, 128 + R
60 FOR T = 0 TO 50 : NEXT T
70 OUT 0, R
80 FOR T = 0 TO 50 : NEXT T
90 NEXT X
```

Kommentarer till ovanstående program:

Rad 10: Ställer PB6 och PB7 som utgångar på användarporten.

Rad 20: N är antalet steg – här har valts 45 steg.

Rad 30: R är riktningen. R = 0 ger rotation i en riktning R = 64 i andra, (PB6 har vikten 64!).

Rad 40: Avgränsar tillsammans med rad 90 en FOR...NEXT-slinga, som matar ut N pulser.

Rad 50: Matar ut "1" på PB7 och "1" eller "0" på PB6, beroende på R.

Rad 60: Väntar 0,1 sekund.

Rad 70: Matar ut "0" på PB7 och "1" eller "0" på PB6, beroende på R.

Rad 80: Väntar 0,1 sekund.

Periodtiden bestäms i huvudsak av rad 60 och 80, den är ca 0,2 sekunder. Om du vill kan den göras kortare, men även utan rad 60 och 80 är den ca 15 ms, vilket är det snabbaste vi kan åstadkomma i Basic. Motorn klarar högre pulsfrekvens, så här har vi en typisk tillämpning för maskinkod för den intresserade.

Förslag till vidareutveckling:

1 Låt datorn styra motorn från data i en tabell, d v s antal pulser, riktning, antal pulser, riktning, o s v.

2 Utvidga tabellen till att omfatta även pulsfrekvens.

3 Gör ett program som accelererar motorn enligt en rampfunktion!

Appendix A

Register i mikroprocessorn Z80

Mikroprocessorn Z80 har totalt 22 olika register. Fyra av dessa är 16-bitarsregister och övriga är 8-bitarsregister, men kan i flera fall kombineras till 16-bitarsregister.

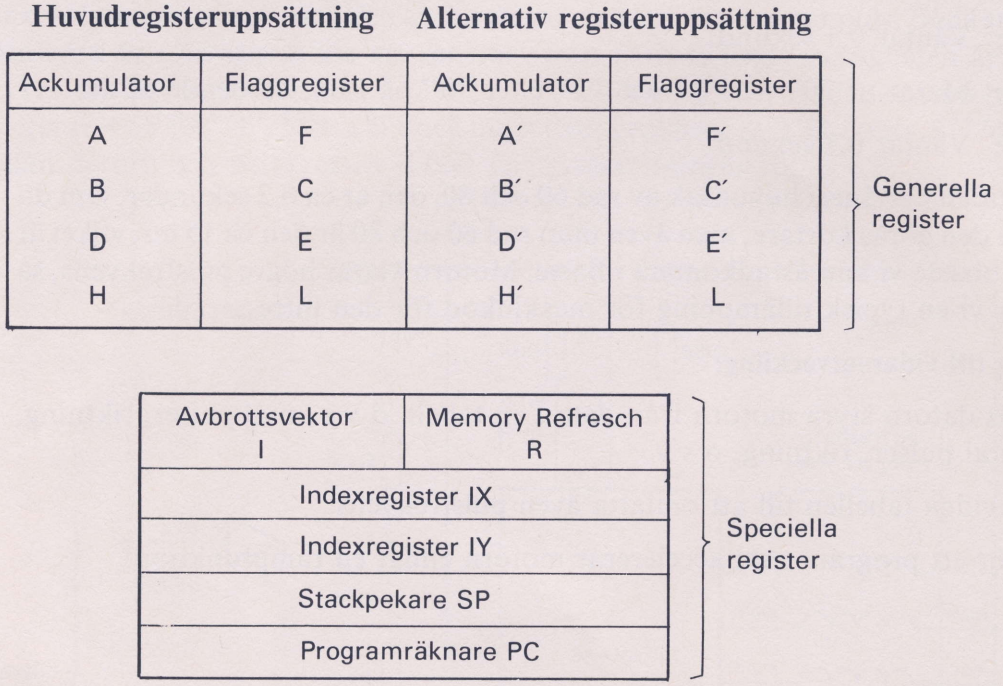


Fig. A. 1 Registerkonfiguration i Z80

Ackumulator och flaggregister

Z80 innehåller två stycken ackumulatorer som påverkar varsitt flaggregister. Ackumulatorn är processorns viktigaste register där t ex alla aritmetiska beräkningar utförs. Val av vilken ackumulator som ska användas sker med en särskild instruktion.

Generella register

Det finns två uppsättningar med generella register, vart och ett bestående av 6 st 8-bitars register. Den ena uppsättningen benämns B, C, D, E, H och L och den andra B′, C′, D′, E′, H′ och L′. Vilken uppsättning som ska användas bestäms på samma sätt som vid val av ackumulator. Registren kan kopplas samman parvis och bildar då två uppsättningar med tre 16-bitarsregister. De betecknas då BC, DE och HL respektive BC′, DE′ och HL′. Möjligheten att bilda 16-bitarsregister gör att de kan användas som adressregister.

Speciella register

Avbrottsvektor (I). Används för att definiera de åtta mest signifikanta bitarna vid avbrott där den periferikrets som genererar avbrottet definierar de åtta minst signifikanta bitarna. Därmed åstadkoms en komplett 16-bitsadress till avbrottsrutinen. Detta gör att avbrottsrutinen kan placeras på godtycklig plats i minnet.

Memory Refresh Register (R). Används för att generera "refresh" av dynamiska RAM.

Programräknare (PC)

Programräknaren är ett 16-bitars register som innehåller adressen till nästa instruktion som ska hämtas från minnet. Programräknaren ökas automatiskt när instruktionen hämtats. Vid hoppinstruktioner laddas programräknaren automatiskt med nya adressen.

Stackpekaren (SP)

Stackpekaren är ett 16-bitarsregister som innehåller adressen till stacken. Stacken kan placeras var som helst i datorns RAM. Stacken används för tillfällig lagring av data och adresser.

Indexregister (IX och IY)

Det finns två av varandra oberoende 16-bitars indexregister i Z80. De används i samband med indexerad adressering. Indexregistret kan peka ut en adressarea, t ex en tabell.

Appendix B

Instruktionslista för mikroprocessor Z80

Instruction Set

The Z80 microprocessor has one of the most powerful and versatile instruction sets available in any 8-bit microprocessor. It includes such unique operations as a block move for fast, efficient data transfers within memory or between memory and I/O. It also allows operations on any bit in any location in memory.

The following is a summary of the Z80 instruction set and shows the assembly language mnemonic, the operation, the flag status, and gives comments on each instruction. The *Z80 CPU Technical Manual* (03-0029-01) and *Assembly Language Programming Manual* (03-0002-01) contain significantly more details for programming use.

The instructions are divided into the following categories:

- ☐ 8-bit loads
- ☐ 16-bit loads
- ☐ Exchanges, block transfers, and searches
- ☐ 8-bit arithmetic and logic operations
- ☐ General-purpose arithmetic and CPU control
- ☐ 16-bit arithmetic operations
- ☐ Rotates and shifts
- ☐ Bit set, reset, and test operations
- ☐ Jumps
- ☐ Calls, returns, and restarts
- ☐ Input and output operations

A variety of addressing modes are implemented to permit efficient and fast data transfer between various registers, memory locations, and input/output devices. These addressing modes include:

- ☐ Immediate
- ☐ Immediate extended
- ☐ Modified page zero
- ☐ Relative
- ☐ Extended
- ☐ Indexed
- ☐ Register
- ☐ Register indirect
- ☐ Implied
- ☐ Bit

8-Bit Load Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
LD r, r'	r ← r'	•	•	X	X	•	•	01 r r'		1	1	4	r, r' Reg.
LD r, n	r ← n	•	•	X	X	•	•	00 r 110		2	2	7	000 B
								← n →					001 C
LD r, (HL)	r ← (HL)	•	•	X	X	•	•	01 r 110		1	2	7	010 D
LD r, (IX+d)	r ← (IX+d)	•	•	X	X	•	•	11 011 101	DD	3	5	19	011 E
								01 r 101					100 H
								← d →					101 L
LD r, (IY+d)	r ← (IY+d)	•	•	X	X	•	•	11 111 101	FD	3	5	19	111 A
								01 r 110					
								← d →					
LD (HL), r	(HL) ← r	•	•	X	X	•	•	01 110 r		1	2	7	
LD (IX+d), r	(IX+d) ← r	•	•	X	X	•	•	11 011 101	DD	3	5	19	
								01 110 r					
								← d →					
LD (IY+d), r	(IY+d) ← r	•	•	X	X	•	•	11 111 101	FD	3	5	19	
								01 110 r					
								← d →					
LD (HL), n	(HL) ← n	•	•	X	X	•	•	00 110 110	36	2	3	10	
								← n →					
LD (IX+d), n	(IX+d) ← n	•	•	X	X	•	•	11 011 101	DD	4	5	19	
								00 110 110	36				
								← d →					
								← n →					
LD (IY+d), n	(IY+d) ← n	•	•	X	X	•	•	11 111 101	FD	4	5	19	
								00 110 110	36				
								← d →					
								← n →					
LD A, (BC)	A ← (BC)	•	•	X	X	•	•	00 001 010	0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	X	X	•	•	00 011 010	1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	X	X	•	•	00 111 010	3A	3	4	13	
								← n →					
								← n →					
LD (BC), A	(BC) ← A	•	•	X	X	•	•	00 000 010	02	1	2	7	
LD (DE), A	(DE) ← A	•	•	X	X	•	•	00 010 010	12	1	2	7	
LD (nn), A	(nn) ← A	•	•	X	X	•	•	00 110 010	32	3	4	13	
								← n →					
								← n →					
LD A, I	A ← I	↕	↕	X	0	X	IFF	0 •	11 101 101	ED	2	2	9
									01 010 111	57			
LD A, R	A ← R	↕	↕	X	0	X	IFF	0 •	11 101 101	ED	2	2	9
									01 011 111	5F			
LD I, A	I ← A	•	•	X	X	•	•	• •	11 101 101	ED	2	2	9
									01 000 111	47			
LD R, A	R ← A	•	•	X	X	•	•	• •	11 101 101	ED	2	2	9
									01 001 111	4F			

NOTES: r, r' means any of the registers A, B, C, D, E, H, L.
IFF the content of the interrupt enable flip-flop, (IFF) is copied into the P/V flag.
For an explanation of flag notation and symbols for mnemonic tables, see Symbolic Notation section following tables.

16-Bit Load Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
LD dd, nn	dd ← nn	•	•	X	•	X	•	•	•	00 dd0 001 ← n → ← n →	3	3	10	dd Pair 00 BC 01 DE 10 HL 11 SP
LD IX, nn	IX ← nn	•	•	X	•	X	•	•	•	11 011 101 DD 00 100 001 21 ← n → ← n →	4	4	14	
LD IY, nn	IY ← nn	•	•	X	•	X	•	•	•	11 111 101 FD 00 100 001 21 ← n → ← n →	4	4	14	
LD HL, (nn)	H ← (nn + 1) L ← (nn)	•	•	X	•	X	•	•	•	00 101 010 2A ← n → ← n →	3	5	16	
LD dd, (nn)	dd_H ← (nn + 1) dd_L ← (nn)	•	•	X	•	X	•	•	•	11 101 101 ED 01 dd1 011 ← n → ← n →	4	6	20	
LD IX, (nn)	IX_H ← (nn + 1) IX_L ← (nn)	•	•	X	•	X	•	•	•	11 011 101 DD 00 101 010 2A ← n → ← n →	4	6	20	
LD IY, (nn)	IY_H ← (nn + 1) IY_L ← (nn)	•	•	X	•	X	•	•	•	11 111 101 FD 00 101 010 2A ← n → ← n →	4	6	20	
LD (nn), HL	(nn + 1) ← H (nn) ← L	•	•	X	•	X	•	•	•	00 100 010 22 ← n → ← n →	3	5	16	
LD (nn), dd	(nn + 1) ← dd_H (nn) ← dd_L	•	•	X	•	X	•	•	•	11 101 101 ED 01 dd0 011 ← n → ← n →	4	6	20	
LD (nn), IX	(nn + 1) ← IX_H (nn) ← IX_L	•	•	X	•	X	•	•	•	11 011 101 DD 00 100 010 22 ← n → ← n →	4	6	20	
LD (nn), IY	(nn + 1) ← IY_H (nn) ← IY_L	•	•	X	•	X	•	•	•	11 111 101 FD 00 100 010 22 ← n → ← n →	4	6	20	
LD SP, HL	SP ← HL	•	•	X	•	X	•	•	•	11 111 001 F9	1	1	6	
LD SP, IX	SP ← IX	•	•	X	•	X	•	•	•	11 011 101 DD 11 111 001 F9	2	2	10	
LD SP, IY	SP ← IY	•	•	X	•	X	•	•	•	11 111 101 FD 11 111 001 F9	2	2	10	
PUSH qq	(SP − 2) ← qq_L (SP − 1) ← qq_H SP ← SP − 2	•	•	X	•	X	•	•	•	11 qq0 101	1	3	11	qq Pair 00 BC 01 DE 10 HL 11 AF
PUSH IX	(SP − 2) ← IX_L (SP − 1) ← IX_H SP ← SP − 2	•	•	X	•	X	•	•	•	11 011 101 DD 11 100 101 E5	2	4	15	
PUSH IY	(SP − 2) ← IY_L (SP − 1) ← IY_H SP ← SP − 2	•	•	X	•	X	•	•	•	11 111 101 FD 11 100 101 E5	2	4	15	
POP qq	qq_H ← (SP + 1) qq_L ← (SP) SP ← SP + 2	•	•	X	•	X	•	•	•	11 qq0 001	1	3	10	
POP IX	IX_H ← (SP + 1) IX_L ← (SP) SP ← SP + 2	•	•	X	•	X	•	•	•	11 011 101 DD 11 100 001 E1	2	4	14	
POP IY	IY_H ← (SP + 1) IY_L ← (SP) SP ← SP + 2	•	•	X	•	X	•	•	•	11 111 101 FD 11 100 001 E1	2	4	14	

NOTES: dd is any of the register pairs BC, DE, HL, SP.
qq is any of the register pairs AF, BC, DE, HL.
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively,
e.g., BC_L = C, AF_H = A.

Exchange, Block Transfer, Block Search Groups

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
EX DE, HL	DE ↔ HL	•	•	X	•	X	•	•	•	11 101 011 EB	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	X	•	X	•	•	•	00 001 000 08	1	1	4	
EXX	BC ↔ BC' DE ↔ DE' HL ↔ HL'	•	•	X	•	X	•	•	•	11 011 001 D9	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔ (SP + 1) L ↔ (SP)	•	•	X	•	X	•	•	•	11 100 011 E3	1	5	19	
EX (SP), IX	IX_H ↔ (SP + 1) IX_L ↔ (SP)	•	•	X	•	X	•	•	•	11 011 101 DD 11 100 011 E3	2	6	23	
EX (SP), IY	IY_H ↔ (SP + 1) IY_L ↔ (SP)	•	•	X	•	X	•	•	•	11 111 101 FD 11 100 011 E3	2	6	23	
LDI	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC − 1	•	•	X	① 0	X	① 1	0	•	11 101 101 ED 10 100 000 A0	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC − 1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 ED 10 110 000 B0	2 2	5 4	21 16	If BC ≠ 0 If BC = 0

NOTE: ① P/V flag is 0 if the result of BC − 1 = 0, otherwise P/V = 1.

Exchange, Block Transfer, Block Search Groups (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
LDD	(DE) ← (HL) DE ← DE − 1 HL ← HL − 1 BC ← BC − 1	•	•	X 0	X ①	↕	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE) ← (HL) DE ← DE − 1 HL ← HL − 1 BC ← BC − 1 Repeat until BC = 0	•	•	X 0	X 0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A − (HL) HL ← HL + 1 BC ← BC − 1	↕	↕ ②	X	↕ X	↕ ①	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A − (HL) HL ← HL + 1 BC ← BC − 1 Repeat until A = (HL) or BC = 0	↕	↕ ②	X	↕ X	↕ ①	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A − (HL) HL ← HL − 1 BC ← BC − 1	↕	↕ ②	X	↕ X	↕ ①	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A − (HL) HL ← HL − 1 BC ← BC − 1 Repeat until A = (HL) or BC = 0	↕	↕ ②	X	↕ X	↕ ①	1	•	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

NOTES: ① P/V flag is 0 if the result of BC − 1 = 0, otherwise P/V = 1.
② Z flag is 1 if A = (HL), otherwise Z = 0.

8-Bit Arithmetic and Logical Group

ADD A, r	A ← A + r	↕	↕	X	↕	X	V	0	↕	10 000 r	1	1	4	r Reg.
ADD A, n	A ← A + n	↕	↕	X	↕	X	V	0	↕	11 000 110 ← n →	2	2	7	000 B 001 C 010 D
ADD A, (HL)	A ← A + (HL)	↕	↕	X	↕	X	V	0	↕	10 000 110	1	2	7	011 E
ADD A, (IX + d)	A ← A + (IX + d)	↕	↕	X	↕	X	V	0	↕	11 011 101 10 000 110 ← d →	DD 3	5	19	100 H 101 L 111 A
ADD A, (IY + d)	A ← A + (IY + d)	↕	↕	X	↕	X	V	0	↕	11 111 101 10 000 110 ← d →	FD 3	5	19	
ADC A, s	A ← A + s + CY	↕	↕	X	↕	X	V	0	↕	001				s is any of r, n, (HL), (IX + d), (IY + d) as shown for ADD instruction. The indicated bits replace the 000 in the ADD set above.
SUB s	A ← A − s	↕	↕	X	↕	X	V	1	↕	010				
SBC A, s	A ← A − s − CY	↕	↕	X	↕	X	V	1	↕	011				
AND s	A ← A ∧ s	↕	↕	X	1	X	P	0	0	100				
OR s	A ← A ∨ s	↕	↕	X	0	X	P	0	0	110				
XOR s	A ← A ⊕ s	↕	↕	X	0	X	P	0	0	101				
CP s	A − s	↕	↕	X	↕	X	V	1	↕	111				
INC r	r ← r + 1	↕	↕	X	↕	X	V	0	•	00 r 100	1	1	4	
INC (HL)	(HL) ← (HL) + 1	↕	↕	X	↕	X	V	0	•	00 110 100	1	3	11	
INC (IX + d)	(IX + d) ← (IX + d) + 1	↕	↕	X	↕	X	V	0	•	11 011 101 00 110 100 ← d →	DD 3	6	23	
INC (IY + d)	(IY + d) ← (IY + d) + 1	↕	↕	X	↕	X	V	0	•	11 111 101 00 110 100 ← d →	FD 3	6	23	
DEC m	m ← m − 1	↕	↕	X	↕	X	V	1	•	101				m is any of r, (HL), (IX + d), (IY + d) as shown for INC. DEC same format and states as INC. Replace 100 with 101 in opcode.

General-Purpose Arithmetic and CPU Control Groups

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands.	↕	↕	X	↕	X	P	•	↕	00 100 111 27	1	1	4	Decimal adjust accumulator.
CPL	$A \leftarrow \bar{A}$	•	•	X	1	X	•	1	•	00 101 111 2F	1	1	4	Complement accumulator (one's complement).
NEG	$A \leftarrow 0 - A$	↕	↕	X	↕	X	V	1	↕	11 101 101 ED 01 000 100 44	2	2	8	Negate acc. (two's complement).
CCF	$CY \leftarrow \overline{CY}$	•	•	X	X	X	•	0	↕	00 111 111 3F	1	1	4	Complement carry flag.
SCF	CY ← 1	•	•	X	0	X	•	0	1	00 110 111 37	1	1	4	Set carry flag.
NOP	No operation	•	•	X	•	X	•	•	•	00 000 000 00	1	1	4	
HALT	CPU halted	•	•	X	•	X	•	•	•	01 110 110 76	1	1	4	
DI ★	IFF ← 0	•	•	X	•	X	•	•	•	11 110 011 F3	1	1	4	
EI ★	IFF ← 1	•	•	X	•	X	•	•	•	11 111 011 FB	1	1	4	
IM 0	Set interrupt mode 0	•	•	X	•	X	•	•	•	11 101 101 ED 01 000 110 46	2	2	8	
IM 1	Set interrupt mode 1	•	•	X	•	X	•	•	•	11 101 101 ED 01 010 110 56	2	2	8	
IM 2	Set interrupt mode 2	•	•	X	•	X	•	•	•	11 101 101 ED 01 011 110 5E	2	2	8	

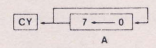
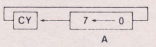
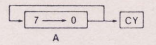
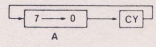
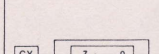
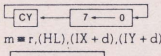
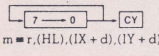
NOTES: IFF indicates the interrupt enable flip-flop.
CY indicates the carry flip-flop.
★ indicates interrupts are not sampled at the end of EI or DI.

16-Bit Arithmetic Group

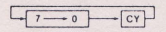
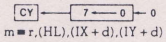
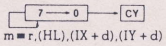
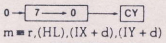
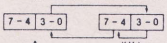
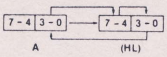
ADD HL, ss	HL ← HL + ss	•	•	X	X	X	•	0	↕	00 ss1 001	1	3	11	ss Reg. 00 BC 01 DE 10 HL 11 SP
ADC HL, ss	HL ← HL + ss + CY	↕	↕	X	X	X	V	0	↕	11 101 101 ED 01 ss1 010	2	4	15	
SBC HL, ss	HL ← HL − ss − CY	↕	↕	X	X	X	V	1	↕	11 101 101 ED 01 ss0 010	2	4	15	
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	↕	11 011 101 DD 01 pp1 001	2	4	15	pp Reg. 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	↕	11 111 101 FD 00 rr1 001	2	4	15	rr Reg. 00 BC 01 DE 10 IY 11 SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00 ss0 011	1	1	6	
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11 011 101 DD 00 100 011 23	2	2	10	
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11 111 101 FD 00 100 011 23	2	2	10	
DEC ss	ss ← ss − 1	•	•	X	•	X	•	•	•	00 ss1 011	1	1	6	
DEC IX	IX ← IX − 1	•	•	X	•	X	•	•	•	11 011 101 DD 00 101 011 2B	2	2	10	
DEC IY	IY ← IY − 1	•	•	X	•	X	•	•	•	11 111 101 FD 00 101 011 2B	2	2	10	

NOTES: ss is any of the register pairs BC, DE, HL, SP.
pp is any of the register pairs BC, DE, IX, SP.
rr is any of the register pairs BC, DE, IY, SP.

Rotate and Shift Group

RLCA		•	•	X	0	X	•	0	↕	00 000 111 07	1	1	4	Rotate left circular accumulator.
RLA		•	•	X	0	X	•	0	↕	00 010 111 17	1	1	4	Rotate left accumulator.
RRCA		•	•	X	0	X	•	0	↕	00 001 111 0F	1	1	4	Rotate right circular accumulator.
RRA		•	•	X	0	X	•	0	↕	00 011 111 1F	1	1	4	Rotate right accumulator.
RLC r		↕	↕	X	0	X	P	0	↕	11 001 011 CB 00 000 r	2	2	8	Rotate left circular register r.
RLC (HL)		↕	↕	X	0	X	P	0	↕	11 001 011 CB 00 000 110	2	4	15	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX + d)	 r,(HL),(IX + d),(IY + d)	↕	↕	X	0	X	P	0	↕	11 011 101 DD 11 001 011 CB ← d → 00 000 110	4	6	23	
RLC (IY + d)		↕	↕	X	0	X	P	0	↕	11 111 101 FD 11 001 011 CB ← d → 00 000 110	4	6	23	
RL m	 m = r,(HL),(IX + d),(IY + d)	↕	↕	X	0	X	P	0	↕	010				Instruction format and states are as shown for RLC's. To form new opcode replace 000 or RLC's with shown code.
RRC m	 m = r,(HL),(IX + d),(IY + d)	↕	↕	X	0	X	P	0	↕	001				

Rotate and Shift Group (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
RR m	 m ≡ r,(HL),(IX + d),(IY + d)	↕	↕	X	0	X	P	0	↕	011					
SLA m	 m ≡ r,(HL),(IX + d),(IY + d)	↕	↕	X	0	X	P	0	↕	100					
SRA m	 m ≡ r,(HL),(IX + d),(IY + d)	↕	↕	X	0	X	P	0	↕	101					
SRL m	 m ≡ r,(HL),(IX + d),(IY + d)	↕	↕	X	0	X	P	0	↕	111					
RLD		↕	↕	X	0	X	P	0	•	11 101 101 01 101 111	ED 6F	2	5	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected.
RRD		↕	↕	X	0	X	P	0	•	11 101 101 01 100 111	ED 67	2	5	18	

Bit Set, Reset and Test Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
BIT b, r	$Z \leftarrow \overline{r_b}$	X	↕	X	1	X	X	0	•	11 001 011 01 b r	CB	2	2	8	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A b Bit Tested 000 0 001 1 010 2 011 3 100 4 101 5 110 6 111 7
BIT b, (HL)	$Z \leftarrow \overline{(HL)_b}$	X	↕	X	1	X	X	0	•	11 001 011 01 b 110	CB	2	3	12	
BIT b, (IX + d)$_b$	$Z \leftarrow \overline{(IX+d)_b}$	X	↕	X	1	X	X	0	•	11 011 101 11 001 011 ← d → 01 b 110	DD CB	4	5	20	
BIT b, (IY + d)$_b$	$Z \leftarrow \overline{(IY+d)_b}$	X	↕	X	1	X	X	0	•	11 111 101 11 001 011 ← d → 01 b 110	FD CB	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 11 b r	CB	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 001 011 11 b 110	CB	2	4	15	
SET b, (IX + d)	$(IX+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 011 101 11 001 011 ← d → 11 b 110	DD CB	4	6	23	
SET b, (IY + d)	$(IY+d)_b \leftarrow 1$	•	•	X	•	X	•	•	•	11 111 101 11 001 011 ← d → 11 b 110	FD CB	4	6	23	
RES b, m	$m_b \leftarrow 0$ m ≡ r, (HL), (IX + d), (IY + d)	•	•	X	•	X	•	•	•	10					To form new opcode replace 11 of SET b, s with 10. Flags and time states for SET instruction.

NOTES: The notation m_b indicates bit b (0 to 7) or location m.

Jump Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
JP nn	PC ← nn	•	•	X	•	X	•	•	•	11 000 011 ← n → ← n →	C3	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	X	•	X	•	•	•	11 cc 010 ← n → ← n →		3	3	10	cc Condition 000 NZ non-zero 001 Z zero 010 NC non-carry 011 C carry 100 PO parity odd 101 PE parity even 110 P sign positive 111 M sign negative
JR e	PC ← PC + e	•	•	X	•	X	•	•	•	00 011 000 ← e − 2 →	18	2	3	12	
JR C, e	If C = 0, continue	•	•	X	•	X	•	•	•	00 111 000 ← e − 2 →	38	2	2	7	If condition not met.
	If C = 1, PC ← PC + e											2	3	12	If condition is met.
JR NC, e	If C = 1, continue	•	•	X	•	X	•	•	•	00 110 000 ← e − 2 →	30	2	2	7	If condition not met.
	If C = 0, PC ← PC + e											2	3	12	If condition is met.
JP Z, e	If Z = 0 continue	•	•	X	•	X	•	•	•	00 101 000 ← e − 2 →	28	2	2	7	If condition not met.
	If Z = 1, PC ← PC + e											2	3	12	If condition is met.
JR NZ, e	If Z = 1, continue	•	•	X	•	X	•	•	•	00 100 000 ← e − 2 →	20	2	2	7	If condition not met.
	If Z = 0, PC ← PC + e											2	3	12	If condition is met.
JP (HL)	PC ← HL	•	•	X	•	X	•	•	•	11 101 001	E9	1	1	4	
JP (IX)	PC ← IX	•	•	X	•	X	•	•	•	11 011 101 11 101 001	DD E9	2	2	8	

Jump Group (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
JP (IY)	PC ← IY	•	•	X	•	X	•	•	•	11 111 101 FD 11 101 001 E9	2	2	8	
DJNZ, e	B ← B – 1 If B = 0, continue	•	•	X	•	X	•	•	•	00 010 000 10 ← e – 2 →	2	2	8	If B = 0.
	If B ≠ 0, PC ← PC + e										2	3	13	If B ≠ 0.

NOTES: e represents the extension in the relative addressing mode.
e is a signed two's complement number in the range < – 126, 129 >.
e – 2 in the opcode provides an effective address of pc + e as PC is incremented by 2 prior to the addition of e.

Call and Return Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
CALL nn	(SP – 1) ← PC_H (SP – 2) ← PC_L PC ← nn	•	•	X	•	X	•	•	•	11 001 101 CD ← n → ← n →	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	X	•	X	•	•	•	11 cc 100 ← n → ← n →	3	3	10	If cc is false.
											3	5	17	If cc is true.
RET	PC_L ← (SP) PC_H ← (SP + 1)	•	•	X	•	X	•	•	•	11 001 001 C9	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	X	•	X	•	•	•	11 cc 000	1	1	5	If cc is false.
											1	3	11	If cc is true.
RETI	Return from interrupt	•	•	X	•	X	•	•	•	11 101 101 ED 01 001 101 4D	2	4	14	
RETN[1]	Return from non-maskable interrupt	•	•	X	•	X	•	•	•	11 101 101 ED 01 000 101 45	2	4	14	
RST p	(SP – 1) ← PC_H (SP – 2) ← PC_L PC_H ← 0 PC_L ← p	•	•	X	•	X	•	•	•	11 t 111	1	3	11	

cc	Condition
000	NZ non-zero
001	Z zero
010	NC non-carry
011	C carry
100	PO parity odd
101	PE parity even
110	P sign positive
111	M sign negative

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

NOTE: [1]RETN loads IFF_2 → IFF_1

Input and Output Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
IN A, (n)	A ← (n)	•	•	X	•	X	•	•	•	11 011 011 DB ← n →	2	3	11	n to A_0 ~ A_7 Acc. to A_8 ~ A_{15}
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	↕	↕	X	↕	X	P	0	•	11 101 101 ED 01 r 000	2	3	12	C to A_0 ~ A_7 B to A_8 ~ A_{15}
INI	(HL) ← (C) B ← B – 1 HL ← HL + 1	X	↕ ①	X	X	X	X	↕	X	11 101 101 ED 10 100 010 A2	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
INIR	(HL) ← (C) B ← B – 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	↕	X	11 101 101 ED 10 110 010 B2	2	5 (If B ≠ 0)	21	C to A_0 ~ A_7 B to A_8 ~ A_{15}
											2	4 (If B = 0)	16	
IND	(HL) ← (C) B ← B – 1 HL ← HL – 1	X	↕ ①	X	X	X	X	↕	X	11 101 101 ED 10 101 010 AA	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
INDR	(HL) ← (C) B ← B – 1 HL ← HL – 1 Repeat until B = 0	X	1	X	X	X	X	↕	X	11 101 101 ED 10 111 010 BA	2	5 (If B ≠ 0)	21	C to A_0 ~ A_7 B to A_8 ~ A_{15}
											2	4 (If B = 0)	16	
OUT (n), A	(n) ← A	•	•	X	•	X	•	↕	X	11 010 011 D3 ← n →	2	3	11	n to A_0 ~ A_7 Acc. to A_8 ~ A_{15}
OUT (C), r	(C) ← r	•	•	X	•	X	•	↕	X	11 101 101 ED 01 r 001	2	3	12	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OUTI	(C) ← (HL) B ← B – 1 HL ← HL + 1	X	↕ ①	X	X	X	X	↕	X	11 101 101 ED 10 100 011 A3	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OTIR	(C) ← (HL) B ← B – 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	X	X	↕	X	11 101 101 ED 10 110 011 B3	2	5 (If B ≠ 0)	21	C to A_0 ~ A_7 B to A_8 ~ A_{15}
											2	4 (If B = 0)	16	
OUTD	(C) ← (HL) B ← B – 1 HL ← HL – 1	X	↕ ①	X	X	X	X	↕	X	11 101 101 ED 10 101 011 AB	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}

NOTE: ① If the result of B – 1 is zero the Z flag is set, otherwise it is reset.
② N Flag is 1 if data bit is 1, othewise N Flag is 0.

Input and Output Group (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	No.of Bytes	No.of M Cycles	No.of T States	Comments
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	X	1	X	11 101 101 ED 10 111 011	2 2	5 (If B≠0) 4 (If B = 0)	21 16	C to A_0 ~ A_7 B to A_8 ~ A_{15}

Summary of Flag Operation

Instruction	D_7 S	Z	H	P/V	N	D_0 C	Comments
ADD A, s; ADC A, s	↕	↕	X	↕	X	V	0	↕	8-bit add or add with carry.
SUB s; SBC A, s; CP s; NEG	↕	↕	X	↕	X	V	1	↕	8-bit subtract, subtract with carry, compare and negate accumulator.
AND s	↕	↕	X	1	X	P	0	0	Logical operations.
OR s, XOR s	↕	↕	X	0	X	P	0	0	
INC s	↕	↕	X	↕	X	V	0	•	8-bit increment.
DEC s	↕	↕	X	↕	X	V	1	•	8-bit decrement.
ADD DD, ss	•	•	X	X	X	•	0	↕	16-bit add.
ADC HL, ss	↕	↕	X	X	X	V	0	↕	16-bit add with carry.
SBC HL, ss	↕	↕	X	X	X	V	1	↕	16-bit subtract with carry.
RLA, RLCA, RRA; RRCA	•	•	X	0	X	•	0	↕	Rotate accumulator.
RL m; RLC m; RR m; RRC m; SLA m; SRA m; SRL m	↕	↕	X	0	X	P	0	↕	Rotate and shift locations.
RLD; RRD	↕	↕	X	0	X	P	0	•	Rotate digit left and right.
DAA	↕	↕	X	↕	X	P	•	↕	Decimal adjust accumulator.
CPL	•	•	X	1	X	•	1	•	Complement accumulator.
SCF	•	•	X	0	X	•	0	1	Set carry.
CCF	•	•	X	X	X	•	0	↕	Complement carry.
IN r (C)	↕	↕	X	0	X	P	0	•	Input register indirect.
INI, IND, OUTI; OUTD	X	↕	X	X	X	X	1	•	Block input and output. Z = 0 if B ≠ 0 otherwise Z = 0.
INIR; INDR; OTIR; OTDR	X	1	X	X	X	X	1	•	
LDI; LDD	X	X	X	0	X	↕	0	•	Block transfer instructions. P/V = 1 if BC ≠ 0, otherwise P/V = 0.
LDIR; LDDR	X	X	X	0	X	0	0	•	
CPI; CPIR; CPD; CPDR	X	↕	X	X	X	↕	1	•	Block search instructions. Z = 1 if A = (HL), otherwise Z = 0. P/V = 1 if BC ≠ 0, otherwise P/V = 0.
LD A, I, LD A, R	↕	↕	X	0	X	IFF	0	•	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag.
BIT b, s	X	↕	X	1	X	X	0	•	The state of bit b of location s is copied into the Z flag.

Symbolic Notation

Symbol	Operation
S	Sign flag. S = 1 if the MSB of the result is 1.
Z	Zero flag. Z = 1 if the result of the operation is 0.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V = 1 if the result of the operation is even, P/V = 0 if result is odd. If P/V holds overflow, P/V = 1 if the result of the operation produced an overflow.
H	Half-carry flag. H = 1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator.
N	Add/Subtract flag. N = 1 if the previous operation was a subtract.
H & N	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
C	Carry/Link flag. C = 1 if the operation produced a carry from the MSB of the operand or result.

Symbol	Operation
↕	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care."
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range < 0, 255 >.
nn	16-bit value in range < 0, 65535 >.

Appendix C

Tabell över sambandet mellan decimala och binära tal

Decimalt	Binärt
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	1 0000
17	1 0001
18	1 0010
19	1 0011
20	1 0100
21	1 0101
22	1 0110
23	1 0111
24	1 1000
25	1 1001
26	1 1010
27	1 1011
28	1 1100
29	1 1101
30	1 1110
31	1 1111
32	10 0000
33	10 0001
34	10 0010
35	10 0011
36	10 0100
37	10 0101
38	10 0110
39	10 0111
40	10 1000
41	10 1001
42	10 1010
43	10 1011
44	10 1100
45	10 1101
46	10 1110
47	10 1111
48	11 0000
49	11 0001
50	11 0010
51	11 0011
52	11 0100
53	11 0101
54	11 0110
55	11 0111
56	11 1000
57	11 1001
58	11 1010
59	11 1011
60	11 1100
61	11 1101
62	11 1110
63	11 1111
64	100 0000
65	100 0001
66	100 0010
67	100 0011
68	100 0100
69	100 0101
70	100 0110
71	100 0111
72	100 1000
73	100 1001
74	100 1010
75	100 1011
76	100 1100
77	100 1101
78	100 1110
79	100 1111
80	101 0000
81	101 0001
82	101 0010
83	101 0011
84	101 0100
85	101 0101
86	101 0110
87	101 0111
88	101 1000
89	101 1001
90	101 1010
91	101 1011
92	101 1100
93	101 1101
94	101 1110
95	101 1111
96	110 0000
97	110 0001
98	110 0010
99	110 0011
100	110 0100
101	110 0101
102	110 0110

Decimalt	Binärt
103	110 0111
104	110 1000
105	110 1001
106	110 1010
107	110 1011
108	110 1100
109	110 1101
110	110 1110
111	110 1111
112	111 0000
113	111 0001
114	111 0010
115	111 0011
116	111 0100
117	111 0101
118	111 0110
119	111 0111
120	111 1000
121	111 1001
122	111 1010
123	111 1011
124	111 1100
125	111 1101
126	111 1110
127	111 1111
128	1000 0000
129	1000 0001
130	1000 0010
131	1000 0011
132	1000 0100
133	1000 0101
134	1000 0110
135	1000 0111
136	1000 1000
137	1000 1001
138	1000 1010
139	1000 1011
140	1000 1100
141	1000 1101
142	1000 1110
143	1000 1111
144	1001 0000
145	1001 0001
146	1001 0010
147	1001 0011
148	1001 0100
149	1001 0101
150	1001 0110
151	1001 0111
152	1001 1000
153	1001 1001
154	1001 1010

Decimalt	Binärt
155	1001 1011
156	1001 1100
157	1001 1101
158	1001 1110
159	1001 1111
160	1010 0000
161	1010 0001
162	1010 0010
163	1010 0011
164	1010 0100
165	1010 0101
166	1010 0110
167	1010 0111
168	1010 1000
169	1010 1001
170	1010 1010
171	1010 1011
172	1010 1100
173	1010 1101
174	1010 1110
175	1010 1111
176	1011 0000
177	1011 0001
178	1011 0010
179	1011 0011
180	1011 0100
181	1011 0101
182	1011 0110
183	1011 0111
184	1011 1000
185	1011 1001
186	1011 1010
187	1011 1011
188	1011 1100
189	1011 1101
190	1011 1110
191	1011 1111
192	1100 0000
193	1100 0001
194	1100 0010
195	1100 0011
196	1100 0100
197	1100 0101
198	1100 0110
199	1100 0111
200	1100 1000
201	1100 1001
202	1100 1010
203	1100 1011
204	1100 1100
205	1100 1101
206	1100 1110

Decimalt	Binärt
207	1100 1111
208	1101 0000
209	1101 0001
210	1101 0010
211	1101 0011
212	1101 0100
213	1101 0101
214	1101 0110
215	1101 0111
216	1101 1000
217	1101 1001
218	1101 1010
219	1101 1011
220	1101 1100
221	1101 1101
222	1101 1110
223	1101 1111
224	1110 0000
225	1110 0001
226	1110 0010
227	1110 0011
228	1110 0100
229	1110 0101
230	1110 0110
231	1110 0111
232	1110 1000
233	1110 1001
234	1110 1010
235	1110 1011
236	1110 1100
237	1110 1101
238	1110 1110
239	1110 1111
240	1111 0000
241	1111 0001
242	1111 0010
243	1111 0011
244	1111 0100
245	1111 0101
246	1111 0110
247	1111 0111
248	1111 1000
249	1111 1001
250	1111 1010
251	1111 1011
252	1111 1100
253	1111 1101
254	1111 1110
255	1111 1111

Ett viktigt användningsområde för datorer är för kontroll och styrning av processer och för automatisk insamling av mätvärden.

Denna bok är avsedd att ge grundläggande kunskaper om mätning och styrning med datorer och att använda Spectravideo SV-318 och SV-328.

Du får lära dig hur din Spectravideodator fungerar och hur användarporten kan utnyttjas för att mäta och styra. Du får även lära dig att programmera din Spectravideodator med maskinkod. Boken innehåller en hel mängd små praktiska experiment och avslutas med tre projekt där du kan bygga en datalogger och utrustning för temperaturreglering och styrning av en stegmotor.

För att kunna tillgodogöra sig innehållet i boken krävs grundläggande kunskaper i BASIC-programmering och för några experiment viss kännedom om filhantering. Dessutom är det bra med kännedom om digitalteknikens grunder.

Liber

ISBN 91-40-20697-1