MIKRO-DASK

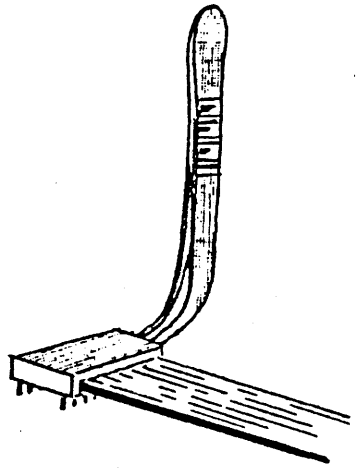
GUNNAR
LUND ©

INDHOLDSFORTEGNELSE

MIKRO-DASK

Idet følgende skal du arbejde med nogle elektroniske moduler, der kan forbindes med hinanden på forskellige måder for til sidst at udgøre en primitiv datamaskine: MIKRO-DASK. Gennem dette arbejde skulle du få et godt indblik i, hvorledes en datamaskine kan være opbygget, og hvad der sker inde i den.

Det er helt nødvendigt, at du arbejder omhyggeligt - især, når du forbinder ledningerne. HVIS DU BYTTER OM PÅ +5V OG 0V VED STRØMFORSYNINGERNE, ØDELÆGGES ET ELLER FLERE MODULER SANDSYNLIGVIS. Brug derfor altid røde ledninger ved +5V og sorte ved 0V. Når du skal frigøre fladkablerne, SKAL DU BRUGE EN PINCET:



TALSYSTEMER

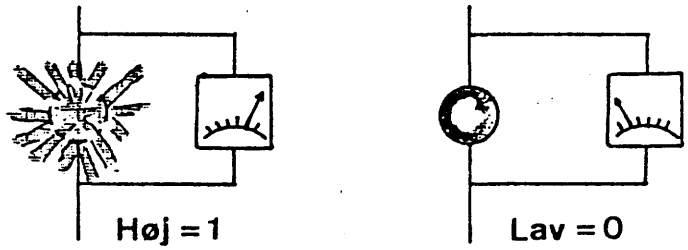
Til de kommende øvelser skal du bruge et tastatur, der ser således ud:

Som du kan se, indeholder det bl.a. 16 taster, der er afmærket med tal fra 0 til 9 og med bogstaver fra A til F. Desuden er der en enkelt seperat tast, som du ikke skal bruge endnu og en skydeomskifter, som du skal sørge for er skubbet over mod højre (OFF).

Forbind tastaturet til en strømforsyning (ikke over 5V !!). Brug en rød ledning ved +5V og en sort ved 0V. Prøv at taste nogle tal mellem 0 og 9 og læg mærke til displayets højre del. Her udlæses de tal, du taster ind. Når et nyt tal indtastes, skubbes det gamle over i den venstre del af displayet.

Bemærk nu de 2 x 4 lysdioder foran displayet. De viser i virkeligheden også de indtastede tal, blot ikke på den sædvanlige måde, men ved hjælp af to-talsystemet (det binære talsystem).

Hver af de 8 lysdioder svarer til det, der kaldes en bit. En sådan bit kan enten være høj eller lav. Hvis bit-en er høj, betyder det, at der er spænding over lysdioden, der så vil lyse. Vi siger også, at bit-en har talværdien 1. Hvis bit-en er lav, er der ingen spænding over lysdioden. Den vil derfor ikke lyse, og vi siger, at talværdien er 0.



Lysdioden længst mod højre sidder ved bit nr. 0, mens lysdioden længst mod venstre sidder ved bit nr. 7.

OPGAVE 1: Find nu ud af, hvilket tal, du skal taste, for at det kun er lysdioden ved bit nr. 0, der lyser. Prøv på tilsvarende måde at finde ud af, hvilke tal, der får lysdioden ved bit 1, 2 og 3 til at lyse.

TI-TALSYSTEMET

For at du bedre kan forstå to-talsystemt, skal vi først lige kigge lidt på ti-talsystemet, som du jo plejer at bruge. Dette talsystem har jo 10 forskellige cifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 og systemets grundtal hedder 10.

Her er nogle potenser af 10 - læg godt mærke til den første:

$10^0 = 1$
$10^1 = 10$
$10^2 = 100$
$10^3 = 1000$

Kig så på tallet 5783. Her er der som bekendt $3 \cdot 10^0$, altså 3 enere. Der er $8 \cdot 10^1$, altså 8 tiere, og der er $7 \cdot 10^2$, altså 7 hundreder. Endelig er der $5 \cdot 10^3$ - 5 tusindere.

TO-TALSYSTEMET (DET BINÆRE TALSYSTEM)

I to-talsystmet anvendes kun to forskellige cifre: 0 og 1. To-talsystemet anvendes i datamaskiner, hvilket hænger sammen med, at dets cifre kan beskrives elektrisk: hvis der er spænding på, svarer det til cifret 1, og hvis der ingen spænding er, svarer det til cifret 0.

Vi ser nu på nogle potenser af tallet 2: (bemærk den første - sammenlign med 10^0 herover.)

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$

Disse potenser skulle gerne svare til det, du før fandt ud af med tastaturet angående de forskellige bits. De forskellige lysdioder vil altså lyse ved disse talværdier:

bit 0 hvis talværdien er lig med 1
bit 1 hvis talværdien er lig med 2
bit 2 hvis talværdien er lig med 4
bit 3 hvis talværdien er lig med 8

Prøv nu at taste tallet 7. Her vil de tre lysdioder til højre lyse. Dette svarer til dette bitmønster: slukket, lys, lys, lys. Med

cifre kan vi skrive: 0 1 1 1. Netop på denne måde skrives tallet 7 ved hjælp af to-talsystemet.

Vi kan kontrollere tallet med et lille regnestykke:

```
1   ved bit 0 giver 1
1   ved bit 1 giver 2
1   ved bit 2 giver 4
0   ved bit 3 giver 0
                   ---
I alt               7
```

Prøv nu at starte med at taste 0, gå videre med 1 og slut ved 9. Regn hver gang efter, om det resultat, som lysdioderne viser, stemmer med det indtastede tal. Sammenlign med skemaet her:

bit 0:	0	1	0	1	0	1	0	1	0	1
bit 1:	0	0	1	1	0	0	1	1	0	0
bit 2:	0	0	0	0	1	1	1	1	0	0
bit 3:	0	0	0	0	0	0	0	0	1	1
talværdi:	0	1	2	3	4	5	6	7	8	9

SEKSTENTAL-SYSTEMET (DET HEXADECIMALE TALSYSTEM)

Du har måske lagt mærke til, at vi indtil nu ikke har udnyttet alle talmuligheder for vores 4 bits. Hvis alle 4 bits nemlig er høje, vil vi få dette tal:

```
1   ved bit 0 giver 1
1   ved bit 1 giver 2
1   ved bit 2 giver 4
1   ved bit 3 giver 8
                   ----
I alt              15
```

Netop for at kunne udnytte alle mulighederne ved de 4 bits, anvender man sekstental-systemet (det hexadecimale talsystem = hex). Her skal man bruge 16 cifre - nemlig: 0, 1, 2, 3, 4, 5, 6, 7, 8,

9, A, B, C, D, E, F. De første cifre svarer til dem, vi kender fra ti-talsystemet. Bogstaverne fra A til F bruges for at gøre det muligt at skrive talværdierne til og med 15 med blot et ciffer. Bemærk at ligesom 9 er det største ciffer i ti-talsystemt, er F (=15) det største ciffer i seksten-talsystemt.

Prøv nu tasterne, der er mærket fra A til F, og regn hver gang efter, om det er de rigtige lysdioder, der lyser. Bemærk også at displayet kan gengive de nævnte bogstaver. (B og D gengives dog som små bogstaver.) Find også ud af forskellen på, hvordan displayet skriver 6 og B. Sammenlign hele tiden med skemaet herunder.

bit	0:	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
bit	1:	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
bit	2:	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
bit	3:	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
talværdi:		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Indtast tallet 10. Nu betyder tallet ikke ti som sædvanligt - vi læser det "en nul" og skriver 10(hex), for at vise, at det ikke er et normalt tal i ti-talsystemet, men et tal fra sekstental-systemet.

Læg nu mærke til, at lysdioden ved bit 4 lyser. Det svarer til 2^4, hvilket er lig med 16. 10(hex) betyder altså, at vi har 0 enere og 1 sekstener.

Tast så 20(hex). Nu lyser lysdioden ved bit 5, hvilket svarer til $2^5 = 32$. 20(hex) betyder altså, at vi har 0 enere, men 2 sekstenere. Afprøv andre tilsvarende tal og regn hver gang efter ud fra lysdioderne, så du bliver sikker i systemet.

OPGAVE 2: Det højeste tal, du kan skrive, er FF(hex). Det betyder, at der er 15 enere og 15 sekstenere:

$$15 + 15 \cdot 16 = 15 + 240 = 255.$$

Prøv at finde ud af, hvorledes tallet 256 skulle skrives i sekstental-systemet.

OPGAVE 3: I ottetal-systemet skal bruges cifrene: 0, 1, 2, 3, 4, 5, 6 og 7. Hvad betyder 10 i ottetal-systemet?

Udblik

Et tastatur på en rigtig datamaskine afgiver også bitmønstre, der sendes ind i maskinen. Her giver et tryk på en tast imidlertid 8 bits på en gang. Det vil sige, at der afgives et bitmønster, der har en talværdi, der kan gå op på 255.

Hvert bogstav og hvert tal på tastaturet har sit eget private bitmønster (talværdi) - hvilket er beskrevet i den såkaldte ASCII-KODE (udtales aski-kode). Her er et udpluk af denne:

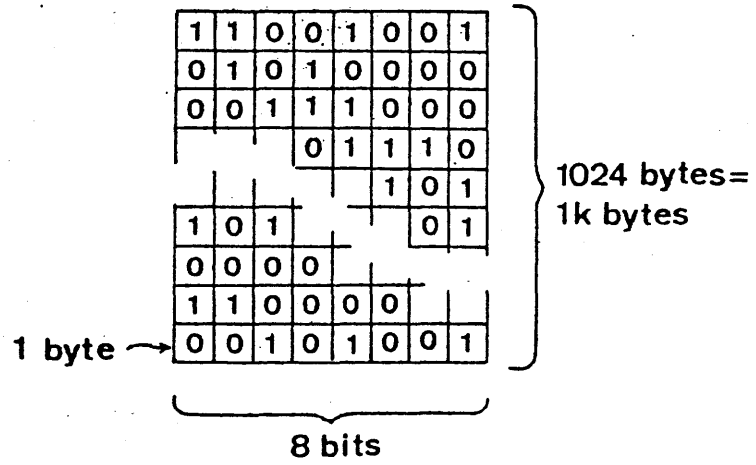
decimal	hex	bitmønster	tegn
58	3A	0011 1010	:
59	3B	0011 1011	;
71	47	0100 0111	G
72	48	0100 1000	H
103	67	0110 0111	g
104	68	0110 1000	h

OPGAVE 4: Kig på ASCII-værdien for G og g (den decimale værdi). Sammenlign med H og h. Hvilken forskel er der i talværdierne mellem de store og de små bogstaver?
Se også på bitmønstrene. Hvilken bit er her forskellig ved et stort og et lille bogstav?

BITS OG BYTES

Som du har set, kan tastaturet aflevere 8 bits. 8 bits kaldes også 1 byte. Mange computere arbejder med 8 bits ad gangen - enkelte med 16 eller 32.

Bit nr. 10 betyder 2^{10} = 1024. Da dette tal er meget tæt på 1000, bruger man ordet kilo om 1024. Hvis en hukommelse er på 1k bytes, betyder det, at den indeholder 1024 tal (words), som hver er 1 byte = 8 bits lange.



ADRESSERING MED 16 BITS

I den færdige MIKRO-DASK får vi behov for at kunne lave tal, der består af 16 bits. Dette kan vi her klare ved hjælp af et særligt mellemled, der skal forbindes til tastaturet, som vist herunder.

Bemærk at begge apparater skal forbindes ved +5V (rød ledning), ved 0V (sort ledning), at de skal forbindes med et bredt kabel med 16 ledere og at "En.1" på de to apparater skal forbindes.

Tast nu et tal på tastaturet og tryk herefter på "ADR.2" på mellemleddet. Nu skal det indtastede tal vises i displayet til venstre på mellemleddet. Tryk så på "ADR.1" - nu skal tallet også vises på det midterste display. Displayet til højre på mellemleddet skal hele tiden vise FF, og det skal ikke bruges endnu.
Prøv at skrive forskellige tal i displayene på mellemleddet, så du er helt klar over funktionen.

Nu har du set, at det er muligt at skrive tal, der består af 2·8 bits = 16 bits (4 cifre med hver 4 bits).

Sørg nu for, at displayene viser: 0008. Tryk på "OP" og bemærk, hvad der sker. Prøv et par gange og tænk over, hvorfor displayene efter et par tryk viser 000A 000B 000C osv. Prøv så at trykke nogle gange på "NED".

OPGAVE 5: Hvis du har tallet 10000 fra ti-talsystemet, men kun kan se de 4 sidste cifre, vil det se således ud: 0000. Hvis du her trækker 1 fra, vil du få tallet 9999. Prøv nu på tilsvarende måde at få displayene til at vise 0000 og tryk på "NED". Bemærk hvad displayene nu viser og prøv at forklare tallet.

De to display, du nu har studeret, skal i det følgende bruges til at finde en 16 bit adresse (herom senere) i en hukommelse. Displayet til højre skal vise os de data (herom senere), der findes i adressen. Prøv at trykke på "DATA" og bemærk, at tastaturets tal vises lige så længe, som du holder knappen nede.

Konklusion

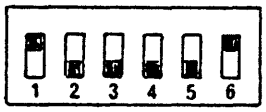
Du har nu set, at tastaturet kan sende tal over i mellemleddet. Ved hjælp af "ADR.2" kan vi føre tallet ind i displayet til venstre, og ved hjælp af "ADR.1" kan vi føre det ind i det midterste display. De to display viser tilsammen et 16 bit tal, der kan bruges til at udpege adresser.

Ved hjælp af "DATA" kan displayets tal føres ind i lystallene til højre, som en 8 bit værdi.

HUKOMMELSESKREDSE

Nu skal du føje yderligere et modul til din opstilling - nemlig et memory-modul (hukommelses-modul). Det hele skal se således ud:

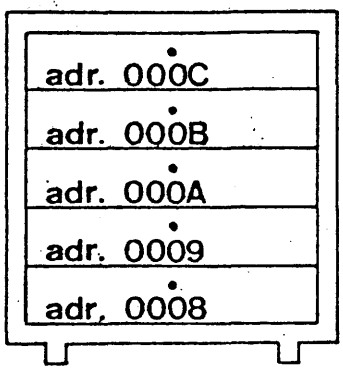
Sørg for at de små kontakter på modulet er indstillet som vist på denne tegning:



Memory-modulet skal forbindes til mellemleddet med to fladkabler, hvor "DATA" forbindes til "DATA" og "ADRESSE" forbindes til "ADRESSE".
Endelig skal modulet have +5V og 0V (rød og sort) og R/W på mellemledet skal forbindes til R/W på memory-modulet (blå).

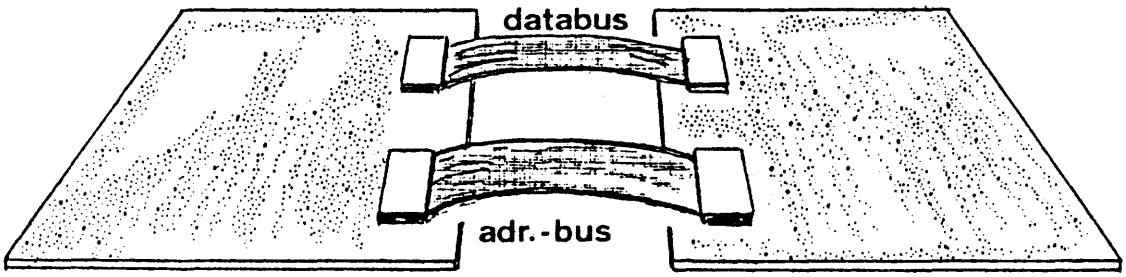
Memory-modulet indeholder bl.a. 3 hukommelseskredse: en EPROM-kreds til venstre og to RAM-kredse i midten og til højre. Bemærk, at de på printet kaldes "memory-1", "memory-2" og "memory-3".

Vi vil starte med at undersøge RAM-kredsen i midten (memory-2). En RAM-kreds er en hukommelses-kreds, hvor brugeren kan indlægge data i bestemte "skuffer" som i en kommode. Senere kan man udvælge sig den samme skuffe igen og her genfinde dataene.



Skufferne har et bestemt nummer - her kaldet en adresse. Adressen består af et 16 bit tal. Dette tal kan jo gå fra 0000 til FFFF. Nu rummer hukommelses- kredsene her imidlertid ikke skuffer nok til alle de mulige 16 bit adresser. RAM-kredsen ved "memory-2" har således adresser mellem 0000 og 07FF.

Sørg nu for, at der på adresse-displayene på mellemleddet står 0000. Nu føres denne adresse gennem kablet foroven over til hukommelses-modulet. Derfor kaldes kablet en adressebus.



Bemærk at når du nu henvender dig til adresse 0000, vil lysdioden ud for den midterste hukommelseskreds lyse som tegn på, at

adressen findes i denne kreds. Data-displayet til højre på mellemleddet vil fortælle hvilke data, der ligger i adresse 0000 (det fortæller, hvad indholdet i den udpegede skuffe er). Tallet her er tilfældigt.

Du kan imidlertid lægge et nyt tal ind: tast f.eks. 00 på tastaturet, tryk på "DATA" og bemærk, at displayene nu viser 0000 00. Det betyder, at der ved adresse 0000 ligger tallet 00. Tryk på "OP", så adressen bliver 0001 og læg tallet 11 ind: tast 11 og tryk på "DATA". Når du trykker på "DATA" laves et signal, som giver RAM-kredsen besked om, at der skal skrives, ikke læses i den.

De data, du har lagt i adresserne, er sendt igennem det nederste kabel, der kaldes en "databus" - se tegningen herover. Denne bus indeholder af praktiske grunde også 16 ledere, men det er kun de 8, der bruges.

Prøv nu at indtaste nedenstående rækkefølge af adresser og data:

adresser:	0006	data:	66
	0005		55
	0004		44
	0003		33
	0002		22
	0001		11
	0000		00

Når du er færdig, finder du adresse 0000 og bladrer op igennem adresserne. Nu skal de indlæste data kunne genfindes. Prøv at fjerne +5V ledningen ved memory-modulet i et stykke tid og sæt den så på igen. Undersøg nu, om dine data er bevaret.

I en datamaskine findes der også RAM-kredse. De bruges som arbejdslager: her gemmes programmer og data. Hvis strømmen imidlertid afbrydes, vil alle data forsvinde. Hvis man derfor skal gemme noget, må det f.eks. ske med en diskette.

Prøv nu igen at lægge data ind i RAM-kredsen - og prøv atter, om du kan genfinde dem. Du kan f.eks. lægge 2-tabellen ind fra adresse 0000, 3- tabellen fra adresse 0010, 4- tabellen fra adresse 0020.

OPGAVE 6: Prøv at spekulere over, hvor mange tabeller, du på den måde kan have i en 2 k bytes RAM (husk du må komme op til 07FF i adresse).

Find nu adresse 07FF og bemærk, at lysdioden på printet ved RAM-kredsen i midten stadig lyser. Gå så en op i adresse - til 0800. Nu slukkes lysdioden, hvilket fortæller, at 0800 ligger uden for kredsens adresserum.

Vi vil nu undersøge EPROM-kredsen (memory-1). Den ligger i adresserummet fra F800 til FFFF. Tast adressen F7FF og bemærk, at lysdioden ved EPROM-kredsen ikke lyser. Kør så en op i adressen med "OP" og læg mærke til, hvad der sker.

Indtast EPROM-ens øverste adresse - FFFF og bemærk, at dataindholdet her er 00. Gå en nedad i adresse (FFFE) og aflæs også her dataindholdet (D0). Prøv om du kan lægge nye data ind på samme måde, som du kunne ved RAM-en.

Kør nu videre baglæns i adressen og aflæs adresse FFFD og FFFC. Her skal dataene være henholdsvis F8 og 00. Så kommer 00, B0, FF, FF. Det hele ser altså således ud:

adr.:	data:
FFFF	00
FFFE	D0
FFFD	F8
FFFC	00
FFFB	00
FFFA	B0
FFF9	FF
FFF8	FF

Afbryd strømmen til datamaskinen i et stykke tid. Undersøg så, om dataene ved de fire adresser stadig er de samme.

Konklusion

En ROM- kreds er en humkommelseskreds, hvorfra man kun kan læse (Read Only Memory). Her er det fabrikanten af kredsen, der har bestemt dataindholdet. En PROM- kreds derimod kan programmeres een gang af brugeren ved en slags "brænding". Dataindholdet kan nu ikke slettes. Ved en EPROM kan også foretages en "brænding" af data, men indholdet kan slettes igen ved belysning med ultraviolet lys. Derfor findes der i EPROM-kredsen en lille rund rude, der normalt er dækket til, så den ikke bliver belyst.

I datamaskiner bruges de forskellige ROM-kredse til at gemme programmer og forskellige oplysninger, der ikke må forsvinde, når

strømmen slukkes. Man kan således sige, at et stort ROM-lager gør, at datamaskinen kan en hel række ting, mens et stort RAM-lager gør, at den kan "lære" en masse nyt.

I den EPROM-kreds, du undersøgte, ligger der et styresystem, der skal få datamaskinen til at virke. En lille del af dette styresystem ligger mellem FFFA og FFFF. Andre dele af kredsens lager benyttes ikke. Dataindholdet i en ubenyttet celle er FF - hvilket netop var tilfældet for adresse FFF8 og FFF9.

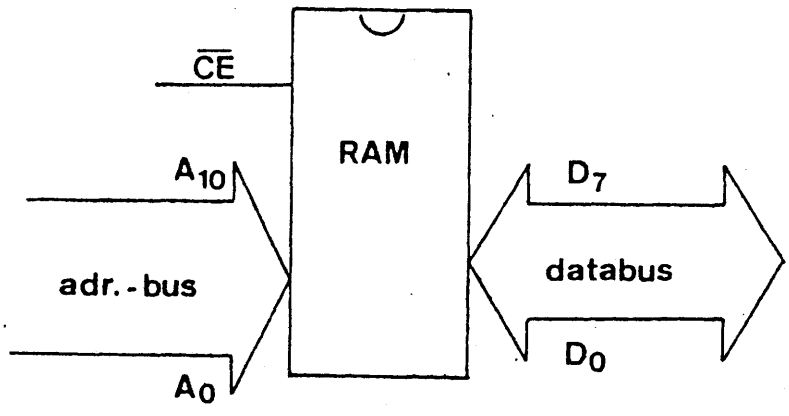
OPGAVE 7: Undersøg andre dele af styresystemet - se beskrivelsen af det bag i bogen - side **.

~~HUKOMMELSESRUM~~ Adresserum

Hvis du har 3 pladser, kan du i ti-talsystemet skrive $10^3 = 1000$ forskellige tal. Du kan nemlig skrive fra 0 og op til 999.

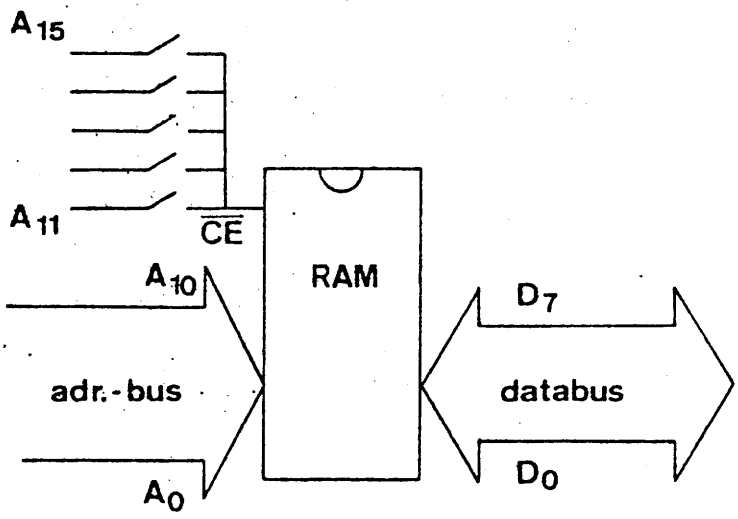
Vi har tidligere nævnt, at der findes 16 bits (fra nummer 0 til og nummer 15) i adressebussen. Det vil sige, at der findes $2^{16} = 65536$ forskellige adresser. Adresserne går fra 0000 og til FFFF. Vi har også tidligere nævnt, at $2^{10} = 1024$ kaldes 1k. Da 65536:1024 = 64, vil det sige, at et hukommelsesrum med 16 bits er 64 k stort. Da hver sæt data i adresserne er på 8 bits = 1 byte, er hukommelsen altså på 64 k bytes.

Herunder kan du se en tegning af den RAM-kreds, der sidder i memory-modulet. Bemærk at der findes 11 ben, der skal forbindes til adressebussen - de går fra adr.0 til adr.10. Det vil sige, at kredsens adresserum er på 2^{11} = 2048 bits eller 2 k bytes.



Når der nu i alt er 64 k bytes til rådighed og en hukommelseskreds kan rumme 2 k, må der altså være plads til i alt 32 stk 2 k bytes hukommelseskredse. Ved hjælp af de elektriske forbindelser er RAM-en i midten (memory-2) som nævnt lagt fra 0000 til 07FF, mens EPROM-en (memory-1) er lagt fra F800 til FFFF.

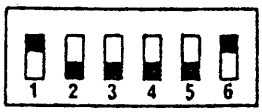
Adresserummet for den sidste RAM-kreds (memory-3) kan fastlægges ved hjælp af 5 af de 6 vippeomskiftere foroven på memory-modulet. (Bemærk at 2^5 netop giver de 32 muligheder). Princippet fremgår af denne tegning:



Fidusen er, at du nu kan fastlægge hvilke af adr. 11 til adr. 15, der skal være høje og hvilke, der skal være lave for at hukommelseskredsen kan adresseres.

De første 5 omskiftere bruges til at vælge adresserum, mens nr. 6 bruges til at give kredsen mulighed for at modtage et skrive-signal. Dette skal bruges ved en RAM-kreds, hvor omskifteren derfor skal vippes opdad, mens den ved en EPROM skal vippes nedad.

Stil vippeomskifterne således, at 1 og 6 er ON og resten er OFF:



bit: 11 12 13 14 15

Nu er kontakterne stillet således, at bit nr. 11 skal være høj, og bit nr. 12, 13, 14 og 15 skal være lave, for at kredsen kan adresseres. ON betyder altså 1, mens OFF betyder 0. Hvad angår adr.0 til adr.10, er der jo frit spil (- det er her de 2 k ligger). Nu er det laveste tal 0800(hex) - se skemaet herunder:

Bit nr.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vardi bin.	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
Vardi hex.	0				8				0				0			

Bemærk, at bit 12 - 15, som det er forlangt, er 0, mens bit 11 er 1. Bit 0 - 10 har her alle fået den laveste værdi - 0. Nu ser vi så situationen, hvor disse bits alle bliver 1. Husk at bit 11 til 15 stadig ligger fast:

Bit nr.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vardi bin.	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
Vardi hex.	0				F				F				F			

Prøv nu ved hjælp af tastatur og mellemled at finde adresse 0800 og bemærk, om den rigtige lysdiode lyser. Gå et trin baglæns i adressen og se, om lysdioden fortsat lyser. Indtast så den høje adresse: 0FFF og se, om lysdioden lyser. Gå yderligere et trin opad og se om lysdioden fortsat lyser.

Herunder ser du et skema, der viser alle de 32 muligheder for valg af adresserum. Den mulighed, du lige har prøvet, er afmærket med en stjerne - kig selv efter, så du finder ud af, hvordan skemaet skal forstås. Prøv også andre af mulighederne - MEN UNDGÅ, AT DE TO RAM-KREDSE KOMMER TIL AT LIGGE I DET SAMME ADRESSERUM (de bliver varme og kan ødelægges) - du må altså ikke vippe alle kontakter ned på en gang.

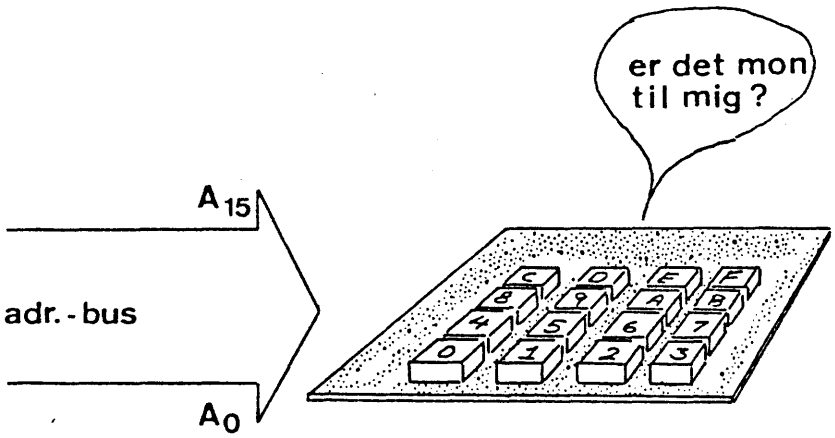
Bit 15	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Bit 14	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
Bit 13	0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
Bit 12	0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
Bit 11	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
Adr. rum:	0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 A A B B C C D D E E F F 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 I I I I I I I I I I I I I I I I I I I I I I I I I I I I I I I I 0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 A A B B C C D D E E F F 7 F 7 F 7 F 7 F 7 F 7 F 7 F 7 F 7 F 7 F 7 F 7 F 7 F 7 F 7 F 7 F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F F *

Udblik

Du har nu studeret det adresserum, man har mulighed for at lave ved at bruge adresser på 16 bits. Hvis man ønsker en datamaskine med en større hukommelse end 64k, er man normalt nødt til at bruge flere adressebits. I en Piccoline er adressebussen på 20 bits. Det giver 2 = 1048576 forskellige adresser. Målt i kilobytes bliver adresserummet derfor 1048576:1024 = 1024k. En standard Piccoline indeholder dog kun 256k.

ADRESSEDEKODEREN

Du har nu set, at de forskellige hukommelsesceller i memory-kredsene kan udpeges ved hjælp af en adresse. Ofte har man imidlertid behov for også at kunne udpege (adressere) forskellige hjælpeenheder (porte, tastaturer m.m.). Hvis dette skal kunne lade sig gøre, må de pågældende enheder kunne "forstå" en 16 bit adresse, og derved afgøre om det er dem, der bliver "ringet op" eller ej.

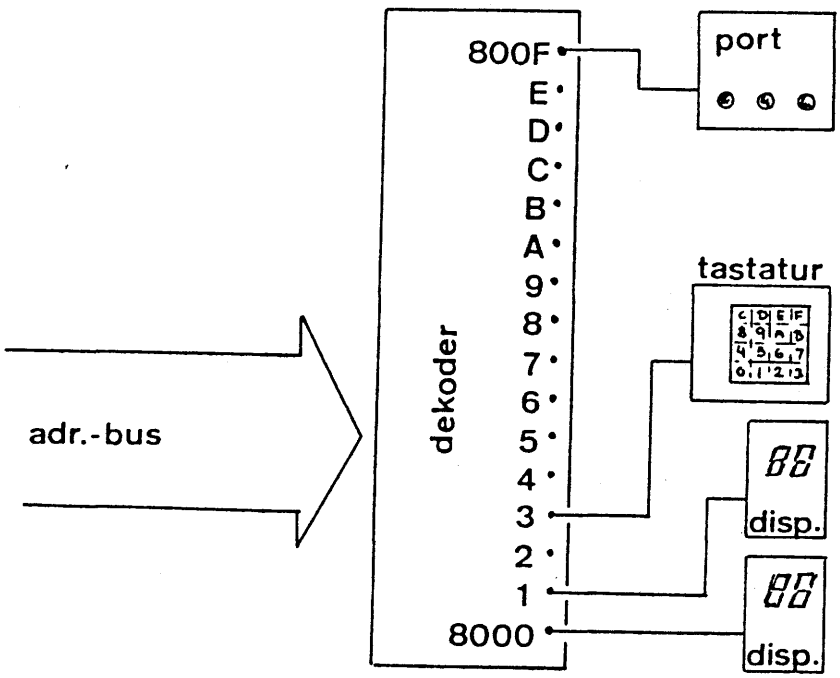


Vi vil imidlertid her vælge en anden mulighed: vi skal bruge en såkaldt adressedekoder, der kan forstå en 16 bit adresse og herefter udpege det apparat, der skal kontaktes.

Fjern derfor memorymodulet og forbind i stedet for adressedekoderen til mellemleddet og til tastaturet:

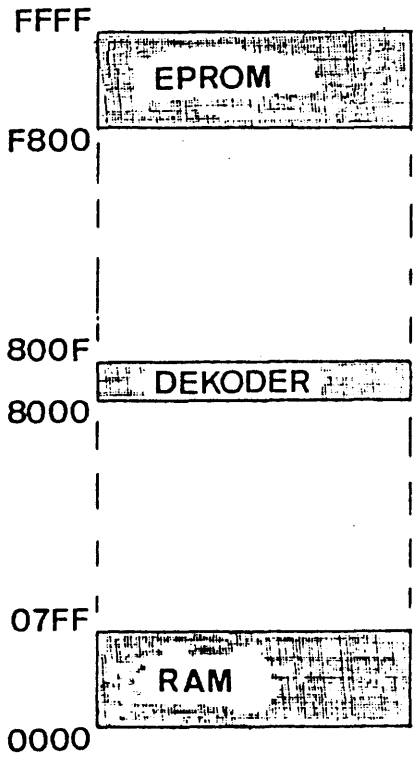
Husk som vist at forbinde en adressebus, en ledning (rød) til +5V og en ledning (sort) til 0V. Indlæs adresse 7FFF og bemærk om nogle af adressedekoderens lysdioder lyser. Gå så et trin op i adresse og bemærk, hvad der sker. Fortsæt med at gå opad i adresserummet så du ser, at der hele tiden kun er een lysdiode, der lyser.

Forestil dig nu, at der forbindes forskellige apparater til de 16 udgange (ved lysdioderne) på adressedekoderen. Nu vil disse apparater kunne kaldes ved hjælp af en adresse mellem 8000 og 800F. Det vil med andre ord sige, at 16 apparater kan være fælles om et apparat (adressedekoderen), der kan forstå adresser og herefter udpege den, der skal i funktion.



MEMORY MAP

Ud fra de forsøg, vi har lavet, kan vi nu vise hvorledes de 64 k, vi har til rådighed, er fordelt mellem de forskellige enheder. Dog er memory-3 ikke taget med, da den jo kan placeres mange steder i adresserummet ved hjælp af omskifterne.



MIKRO-DASK-PROGRAMMER

I det følgende skal du bruge en MIKRO-DASK, der er samlet:

CPU: betyder central-proces-unit og danner "hjernen" i datamaskinen. Alt hvad der sker styres af CPU-en. Modulet indeholder et krystal, der giver 1.000.000 clock-impulser pr. sekund.

Port: bruges til at maskinen kan komme i forbindelse med omverdenen - ved styring og måling.

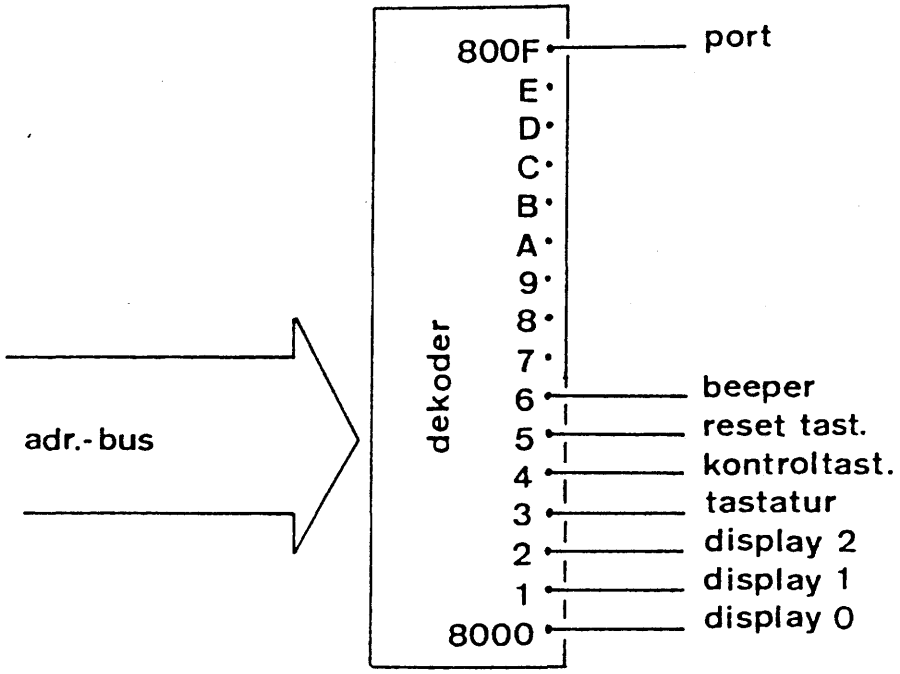
Tastatur: kan afgive bitmønstre, som kan hentes af CPU-en og evt. lagres i humkommelsen.

Display: display 1 og 2 udlæser normalt adressen, mens display 0 udlæser de tilhørende data.

Memory: indeholder en EPROM med styresystem og en RAM, hvor brugerprogrammer og forskellige data kan opbevares.

Dekoder: modtager adresser fra bussen og omsætter dem til signaler, der kan forstås af displayene, port, tastatur, og hvad man ellers ønsker at forbinde.

Du har tidligere undersøgt adressedekoderen. Den kan jo modtage en adresse på 16 bits og her undersøge, om adressen har en værdi mellem 8000 og 800F. Her er en oversigt over de forskellige moduler, der er forbundet til adressedekoderen:



I EPROM-en ligger der nogle programmer, som du skal afprøve. Det sker ved, at du efter at have trykket på "reset" finder programmets start-adresse. Tast derfor FE og fortæl, at det er en adresse-byte ved at holde "kontrol" nede og samtidig trykke på 1. Tast så: 00 og igen "kontrol"+1. Nu bør displayene vise:

FE00 A9

Det betyder, at du har fundet adresse FE00, og at dataindholdet her er A9. Du er nu ved programmets start-adresse. Start det

derfor ved at trykke "kontrol"+3, hvilket svarer til "run". Nu bør beeperen tage fat. Du kan standse programmet igen ved at trykke på "reset".

Nu vil du se, at adresse og data fra før er bevaret, så du kan starte programmet igen ved blot at trykke "kontrol"+3.

Prøv nu at finde et nyt program:

tryk på "reset",
tast FE og "kontrol"+1
tast 20 og "kontrol"+1

Nu bør displayene vise:

FE20 A9

Start igen programmet med "kontrol"+3 og bemærk lysdioderne på porten. Tryk på "reset" og finde selv programmet, der ligger ved FE40 og få det til at køre. Nu bør der ske noget ved displayene.

På adresse FE60, FE80 og FE90 ligger der flere småprogrammer, som du kan afprøve.

Du skal nu selv indtaste et lille program. Det kan det helt enkle, at det får diplay 1 til at udlæse de tal, der indtastes på tastaturet. Start med at trykke på "Reset"-knappen på CPU-modulet. tast så:

02 og fortæl, at tallet er en del af en adresse ved at trykke på "Kontrol" og 1 samtidigt.

00 tryk igen på "Kontrol" og 1, da det er en adresse-byte.

Nu bør adresse-displayene vise 0200, mens det vil være tilfældigt, hvad data-displayene viser. Tast nu:

AD tryk nu "Kontrol" og 2, for at fortælle, at det er data.

Nu bør der så i displayene stå: 0200 AD. Det næste, der skal indtastes er: 0201 03. Det betyder, at ved adresse 0201 skal dataene være 03. Det indtaster du således:

02 "Kontrol" + 1

01 "Kontrol" + 1

03 "Kontrol" + 2

Indtast nu på tilsvarende måde disse tal:

adr.	data
0202	80
0203	8D
0204	01
0205	80
0206	4C
0207	00
0208	02

Nu skal du afprøve programmet. Det gør du ved at finde dets startadresse: 0200 og derefter trykke "Run". Det sidste gør du med "Kontrol"+3. Når du nu indtaster et tal, skal det både vises i tastaturets display og i display 1.

Du vil sikkert synes, at dette her var en besværlig måde at indtaste et program på. Der findes da også nogle kontroltaster, der gør livet lidt lettere for brugeren. Tryk på reset og bemærk at displayene nu viser programmets startadresse. Du kan nu blade frem gennem programmet ved hjælp af "kontrol"+6. Prøv det og bemærk, at programmet i sin helhed ser således ud:

adr.	data
0200	AD *
0201	03
0202	80
0203	8D *
0204	01
0205	80
0206	4C *
0207	00
0208	02

Prøv også at blade baglæns ved hjælp af "kontrol"+7! Du kan nu fremover udnytte "kontrol"+6 til at blade frem til den næste adresse, når du er ved at indtaste et program.

Bemærk nu de data, der er markeret med en stjerne. Her er der tale om de såkladte operationskoder (op-kode). Det er særlige bitmønstre, som CPU-en genkender. Således betyder:

AD: Hent data fra den kommende adresse og læg det ind i lageret (accumulator) i CPU-en.

8D: Læg det tal, som er i CPU-ens lager (accumulator) hen i den kommende adresse.

4C: Spring hen til den kommende adresse.

De bytes, der kommer efter op-koden, kaldes operanden. Vi vil nu skrive programmet på en måde, så det tydeligere fremgår, hvilke op-oder og hvilke adresser (operander), der hører sammen:

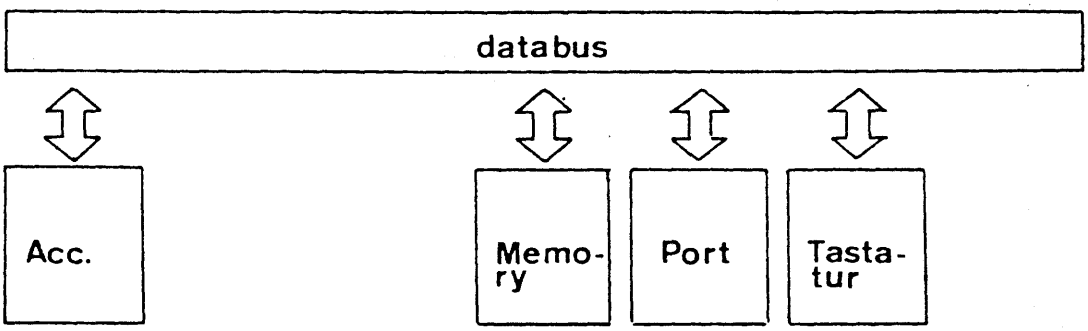
```
0200  AD 0380  Hent tal fra adresse 8003 ind i accumulator.
0203  8D 0180  Læg indholdet af accumulator ud på adresse 8001.
0206  4C 0002  Spring op til adresse 0200 og begynd forfra.
```

Bemærk nu, at processoren skal have adresserne ind baglæns: 8003 skrives som 0380 og 0200 skrives som 0002.

Læg også mærke til, at tastaturet har adresse 8003, og at det midterste display har adresse 8001. Det vil med andre ord sige, at programmet henter tal fra tastaturet og lægger dem ud på display 1 - hvilket vel må siges at være en lidt fattig præstation.

Til gengæld skal du bemærke, at CPU-en kører programmet igennem op mod 100.000 gange på et sekund. Den står med andre ord aldrig stille, men er hele tiden igang med at undersøge tastaturet og bagefter føre tallene ud på displayet.

Herunder kan du se en princip-tegning over opstillingen. Accumulatoren (inde i CPU-en) står i forbindelse med både port og tastatur ved hjælp af databussen. Det er CPU-en, der bestemmer, hvorfra og hvortil dataene sendes.



Vi vil nu føje to linier ind før 4C 0002, så tallet også skrives på display 0 og 2. Vi er derfor nødt til at ændre programmet fra adresse 0206 og fremad. Det skal altså se således ud:

```
0200    AD *
0201    03
0202    80

0203    8D *
0204    01
0205    80

0206    8D *
0207    00
0208    80

0209    8D *
020A    02
020B    80

020C    4C *
020D    00
020E    02
```

Ret programmet på følgende måde:

```
tryk på "reset"
tast 02 og "kontrol"+1 = adresse
tast 06 og "kontrol"+1 = adresse

tast 8D og "kontrol"+2 = data 8D ved adresse 0206
"kontrol"+6            = en frem
tast 00 og "kontrol"+2 = data 00 ved adresse 0207
"kontrol"+6            = en frem
tast 80 og "kontrol"+2 = data 80 ved adresse 0208
```

Fortsæt på tilsvarende måde, så programmet ser ud som vist ovenfor. Husk at du skal starte programmet fra adresse 0200.

OPGAVE 8: Porten har normalt adresse 800F (afhængig af hvordan ledningen fra dekoderen er forbundet). Indføj nu en linie, der også fører et indtastet tal ud på porten. Start igen programmet fra adresse 0200 og sammenlign bitmønstrene ved portens lysdioder med bitmønstrene ved tastaturets lysdioder.

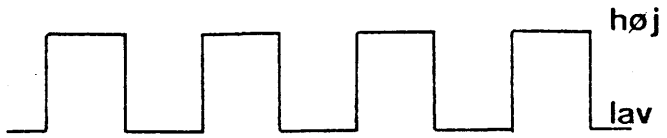
Når vi vil skrive det sidste program, så det tydeligt kan ses, hvad der er op-kode og hvad der er operander, ser det således ud - studer det grundigt, thi fremover vil alle programmer blive vist på denne måde:

```
0200  AD 0380  Hent tal fra adresse 8003 ind i accumulator.
0203  8D 0180  Læg indholdet af accumulator ud på adresse 8001.
0206  8D 0080  Læg indholdet af accumulator ud på adresse 8000.
0209  8D 0280  Læg indholdet af accumulator ud på adresse 8002.
020C  8D 0F80  Læg indholdet af accumulator ud på adresse 800F.
020F  4C 0002  Spring op til adresse 0200 og begynd forfra.
```

STYRING MED PORTEN

Du skal nu bruge et modul med en såkaldt stepmotor. Det ser således ud:

En stepmotor er en motor, der ikke kører, blot man sætter strøm til den. Ud over strøm skal den nemlig have nogle såkaldte trigge-impulser. Det betyder blot, at dens trigge-indgang skal have en spænding, der hele tiden skifter mellem at være høj og lav. Det kan den f.eks. få fra en firkantkurve:



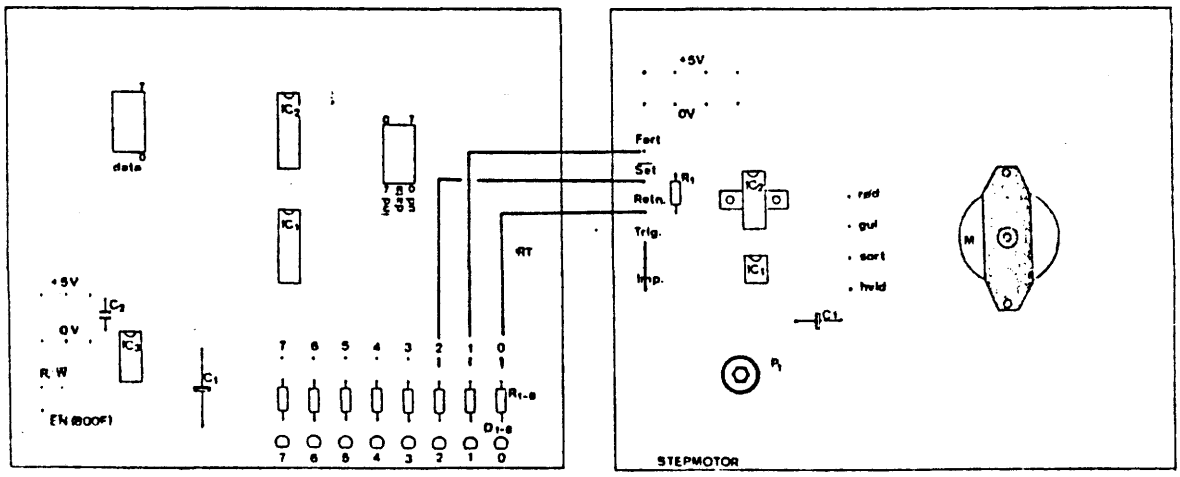
På printet er der derfor en elektronisk enhed, der laver disse firkantkurver. Forbind derfor "AMV-ud" med "Trig" ved stepmotoren. Sørg også for at "Set" er forbundet til 0V. Når du nu sætter strøm til (forbind til +5V og 0V på datamaskinen), skulle motoren gerne køre.

OPGAVE 9: Find ud af, hvad der sker, når "Set" gøres høj (= uden forbindelse eller forbundet til +5V).

OPGAVE 10: Find ud af, hvad der sker, når "Set" er lav og "Fart" skiftevis gøres høj og lav.

OPGAVE 11: Find til sidst ud af, hvad der sker, når du skiftevis gør "Retn" høj og lav.

Forbind nu stepmotorens styreindgange "Set", "Fart" og "Retn" til udporten som vist på denne tegning:



Du kan enten bruge programmet fra side 26, eller du kan blot nøjes med dette program:

```
0200  AD 0380  Hent tal fra adresse 8003 ind i accumulator.
0203  8D 0F80  Læg indholdet af accumulator ud på adresse 800F.
0206  4C 0002  Spring op til adresse 0200 og begynd forfra.
```

Hvis du nu trykker på en tast mellem 0 og 4, skal motorens andre kørselsmåde. Skift mellem at trykke 0 og 1 og læg mærke til bit 0 på porten. Nu skifter du altså mellem at gøre "Retn" høj og lav - det skal nu får motoren til at skifte retning ved hvert tryk.

Prøv på tilsvarende måde at undersøge virkningen af de andre tal mellem 0 og 4. Sammenlign så med skemaet her og se, om du kan få det til at passe med dine forsøg.

Bit 2 Set	Bit 1 Fart	Bit 0 Retn	Tal fra tastatur	Kørsels- måde
0	0	0	0	Frem/max.
0	0	1	1	Bak/max.
0	1	0	2	Frem/min.
0	1	1	3	Bak/min.
1	1/0	1/0	4-7	Stop

Konklusion:
==========

Du har nu set, at et maskinkodeprogram kan lægges ind i RAM-kredsen og at det indeholder forskellige bitmønstre, som vi udtrykker som hex-tal. Endelig har du set, at hver instruktion indledes med en operationskode (op-kode), som er en bestemt kode, CPU-en genkender. Efter operationskoden kommer en adresse - en operand som operationskoden knytter sig til:

adr.	op-kode	operand
0200	8D	0380

FUNKTIONSTASTER

Inden du nu atter indtaster et program, skal du lige lære en lille fidus at kende. Tryk på "Reset" og indtast en eller anden adresse, f.eks. 0400. Læg f.eks. FF ind som data, idet du trykker på "Kon-

trol"+4 og ikke "Kontrol"+2 som normalt. Nu kan du se, at styreprogrammet får maskinen til automatisk at gå frem til den næste adresse: 0401. Indlæs igen nogle data med "Kontrol"+4 og se fænomenet gentage sig.

Hvis du nu ønsker at blade hurtigt frem eller tilbage kan du bruge "Kontrol" sammen med 8, 9, A eller B.

Bemærk at der ingen huller må være i et program: maskinen løber programmets adresser igennem en ad gangen (sker ved hjælp af en indbygget programtæller), og den forventer hele tiden at finde en op-kode med tilhørende operand. Hvis den møder en hukommelsescelle, hvor der står noget tilfældigt, vil den forlade det egentlige program og lave noget helt tilfældigt.

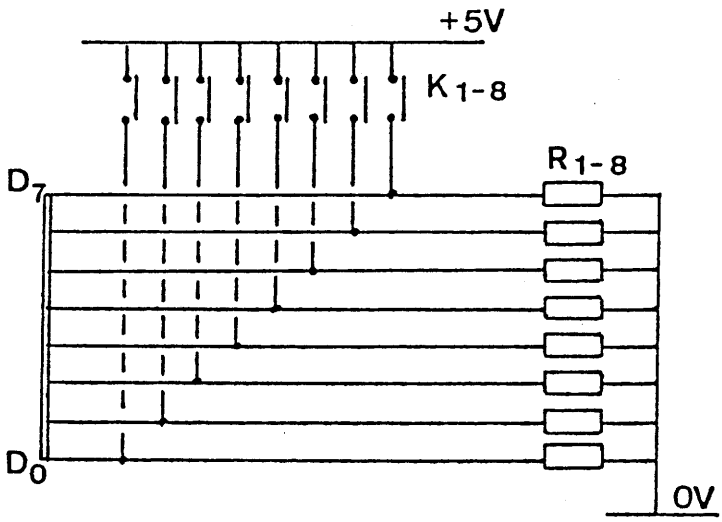
For at afhjælpe dette problem er det ved hjælp af styresystemet gjort muligt at indsætte en byte ved, at de øvrige skubbes opad. På tilsvarende måde kan en byte fjernes. Det sker ved hjælp af "kontrol" sammen med D eller C. Herunder er betydningen for de forskellige kontroltaster vist:

slet b. C	inds. b. D	E	F
side op 8	sidened 9	+10 A	−10 B
data+1 4	data−1 5	+1 6	−1 7
 0	adress 1	data 2	run 3

Hvis du selv vil prøve at lave nogle programmer, skal du i almindelighed undgå adresseområdet fra 00A0 til 01EF, da dette område bruges af processoren selv. Her vil indsættelse og sletning af bytes ved hjælp af kontroltastaturet heller ikke virke!!

MÅLING MED PORTEN

Du skal nu bruge en portkontakt. Den indeholder 8 digitaster, som hver er forbundet til en dataledning. Diagrammet ser således ud:



Modulet skal forbindes til databussen på porten, og det skal have +5V og 0V. Nu skal du indtaste et program, der henter bitmønstre, som sendes ind i porten og bagefter viser de tilsvarende tal på et (eller flere) display. Programmet kan f.eks. se således ud - tryk på "Reset", inden du begynder at taste det ind:

```
0300  AD 0F80  Hent tal fra adresse 800F ind i accumulator.
0303  8D 0180  Læg indholdet af accumulator ud på adresse 8001.
0306  4C 0003  Spring op til adresse 0300 og begynd forfra.
```

Husk at portens adresse er 800F, mens adressen på display 1 er 8001, og at disse adresser skal "baglæns ind". Start programmet fra adresse 0300 og tryk på nogle ringetryk, mens du holder øje med display 1.

OPGAVE 12: Hvilket bitmønster får du, når du ikke trykker på nogle ringetryk? Hvilket bitmønster får du, når du trykker på dem allesammen på en gang? Hvilken værdi får de enkelte bits altså, når ringetrykket ikke trykkes ned?

Udblik

På en rigtig datamaskine findes der også forskellige porte - bl.a. til printerstik. Som du har set, kan en sådan port både udsende og modtage data. Det er klart, at der fra datamaskinen udsendes data til printeren. Dataene vil her svare til ASCII-koden (side 6) og printeren vil finde og skrive de tegn, der svarer til koden.

Men der går også data fra printeren til datamaskinen: f.eks. fortæller printeren ved forskellige signaler, at den er færdig, at der ikke er mere papir, eller at der er blevet trykket på "Deselect".

På tilsvarende måde kan en datamaskine via en eller flere porte styre forskellige apparater - f.eks. robotter, idet den her både afgiver og modtager data.

STYRING AF LYSKURV

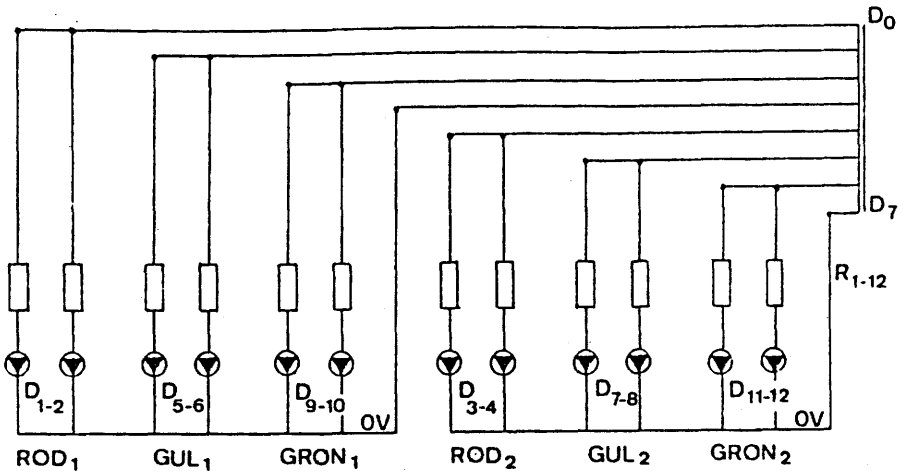
Tilslut lyskurven, tryk på "Reset" og indtast det program, du tidligere har brugt til at skrive til porten med:

```
0200  AD 0380  Hent tal fra adresse 8003 ind i accumulator.
0203  8D 0F80  Læg indholdet af accumulator ud på adresse 800F.
0206  4C 0002  Spring op til adresse 0200 og begynd forfra.
```

Sørg for, at du står ved adresse 0200 og tryk på "Run". Tryk herefter på forskellige taster på tastaturet. Nu skal lyskurven reagere ved, at der er forskellige lysdioder, der lyser.

Lysdioderne er forbundet parvis, således at lys, der hører til den samme trafikretning, følges ad. Lysdioderne er forbundet til de forskellige bits som vist på næste side.

Bemærk at hvis bit 3 er lav (= 0) giver den 0V til det ene sæt lysdioder. På samme måde kan bit 7 give 0V til det andet sæt lysdioder. Hvis du taster 88 (eller derover), vil både bit 3 og bit 7 blive høje, og alle lysdioder vil slukkes. Modsat vil tallet 77 få alle lysdioder til at lyse.



OPGAVE 13: Forestil dig nu, at du skal styre et lysreguleret kryds ved hjælp af tastaturet. Du skal derfor sørge for, at der er grønt i den ene trafik-retning, mens der er rødt i den modsatte osv. Find derfor ud af, hvilke tal du skal taste ind i de forskellige situationer - stød dig til skemaerne herunder:

Trafikretning 2	Trafikretning 1	Talværdi
rødt	grønt	41
gult	gult	??
grønt	rødt	??
gult	gult	??

TRAFIKRETNING 2				TRAFIKRETNING 1				Hex tal	Lys
Bit 7 ON/OFF	Bit 6 grøn	Bit 5 gul	Bit 4 rød	Bit 3 ON/OFF	Bit 2 grøn	Bit 1 gul	Bit 0 rød		
0	0	0	1	0	0	0	1	11	rød/rød
0	0	1	0	0	0	1	0	22	gul/gul
0	1	0	0	0	1	0	0	44	grøn/grøn
1	1/0	1/0	1/0	1	1/0	1/0	1/0	88-FF	slukket

AUTOMATISK STYRING AF LYSKURVEN

Vi vil nu prøve at lave et program, der selv kan styre lyskurven. For at det kan lade sig gøre, bliver vi nødt til at lære nogle nye operationskoder at kende. Tryk på "Reset", indtast dette program og se, om det virker:

```
Rød/grøn:
  0200  A914    LDA  Indlæs tallet 14(hex) i accumulatoren.
  0202  8D0F80  STA  Skriv indholdet af accumulatoren i adresse
                     800F (porten og dermed lyskurven).

  0205  2020FA  JSR  Spring til pause-program på adresse FA20.

Gul/gul:
  0208  A922    LDA  Indlæs tallet 22(hex) i accumalatoren.
  020A  8D0F80  STA  Skriv indholdet af accumulatoren i adresse
                     800F (porten og dermed lyskurven).

  020D  2020FA  JSR  Spring til pause-program på adresse FA20.

Grøn/rød:
  0210  A941    LDA  Indlæs tallet 41(hex) i accumalatoren.
  0212  8D0F80  STA  Skriv indholdet af accumulatoren i adresse
                     800F (porten og dermed lyskurven).

  0215  2020FA  JSR  Spring til pause-program på adresse FA20.

Gul/gul:
  0218  A922    LDA  Indlæs tallet 22(hex) i accumalatoren.
  021A  8D0F80  STA  Skriv indholdet af accumulatoren i adresse
                     800F (porten og dermed lyskurven).

  021D  2020FA  JSR  Spring til pause-program på adresse FA20.
  0220  4C0002  JMP  Spring til adresse 0200 (toppen af program-
                     met)
```

Bemærk at programmet springer til en pause, der ligger i EPROM-en. Pauselængden her er afhængig af indholdet i adresse 00A1 og 00A3. Hvis der begge steder står FF, vil pausen vare længst muligt, mens 00 vil give den korteste pause. Læg selv nogle passende værdier ind i adresserne.

Bemærk forkortelsen LDA, som her svarer til op-koden A9. LDA er

en forkortelse for "load accumulator with memory" og sådanne forkortelser bruges til at beskrive de forskellige instruktioner, idet de er lettere at huske end talkoderne. Her er en oversigt over de almindeligste af disse forkortelser (mnemonics):

ADC add memory to accumulator with carry: læg memory og evt. mente sammen med accumulator.

CLC clear carry flag: slet menten (før addition)

CLD clear decimal mode: gå væk fra at regne i ti-talsystem

CMP compare memory and accumulator: sammenlign memory og accumulator

BNE branch on result not zero: gå til en forgrening, hvis resultatet ikke er nul

DEC decrement memory by one: træk 1 fra en celle i hukommelsen

DEX decrement index x by one: træk 1 fra indholdet i x- register

DEY decrement index y by one: træk 1 fra indholdet i y- register

INC increment memory by one: læg 1 til en celle i hukommelsen

INX increment index x by one: læg 1 til indholdet i x- register

INY increment index y by one: læg 1 til indholdet i y- register

JMP jump to new location: spring til en ny adresse

JSR jump to new location saving return address: spring til et underprogram og husk den adresse, du kom fra

LDA load accumulator with memory: indlæs et tal fra hukommelsen til accumulatoren

ROL rotate one bit left: flyt bit-ene en plads mod venstre

ROR rotate one bit right: flyt bit-ene en plads mod højre

SBC subtract memory from accumulator with borrow: træk hukommelse fra accumulator, lån evt.

SEC set carry flag: set menten (før subtraktion)

SED set decimal mode: regn i ti-talsystem (vedr. ADC og SBC)

STA store accumolator in memory: skriv indholdet af accumulatoren ud i en hukommelsesadresse

Der findes flere instruktioner end de nævnte - se evt. den komplette oversigt bag i bogen (side 112). Prøv nu at kigge på lyskurveprogrammet igen og oversæt de forskellige forkortelser for instruktioner. Prøv også at se, om du kan forstå programmet.

Bemærk, at det består af forskellige blokke, der næsten er ens: der indlæses et tal i accumulatoren og dette tal skrives så til porten. Mellem hver blok springes der til et underprogram der sørger for, at der holdes en passende pause.

OPGAVE 14: En rigtig lyskurv skifter lidt anderledes end denne her gør: i det øjeblik, det bliver rødt i trafikretning 1, vil der både være gult og rødt i trafikretning 2 - se beskrivelserne herunder og lav dit program om, så lyskurven skifter på tilsvarende måde.

Trafikretning 2	Trafikretning 1
rødt	grønt
rødt	gult
rødt	rødt
rødt+gult	rødt
grønt	rødt
gult	rødt
rødt	rødt
rødt	rødt+gult

- forfra -

OPGAVE 15: Læg mærke til lysdioderne på porten, mens lyskurven kører. Fjern lyskurven (pincet!!) og prøv at lave dit program om, så det får lysdioderne til at køre som i et løbelys: først skal lysdioden ved bit 0 lyse, dernæst lysdioden ved bit 1 og så videre op til bit 7. Herefter skal programmet starte forfra.

Du skal starte med at gøre dig klart, hvilke talværdier, der får de enkelte lysdioder til at lyse.

OM AT GANGE OG DIVIDERE MED 2

Formentlig synes du, at løsningen til opgave 15 var lidt besværlig, lidt Gøg og Gokke-agtig. Der skulle jo blot hele tiden udlæses et tal, der var 2 gange så stort som det tidligere.

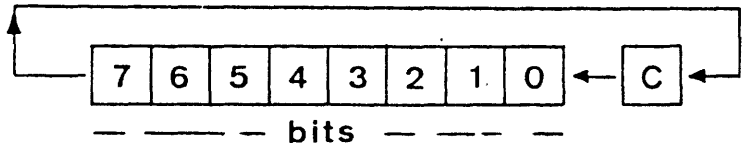
Programmet kan da også fikses op ved hjælp af nogle nye instruktioner. Prøv først at indtaste det og se, om det virker, som det skal:

```
0300  A901    LDA  Indlæs tallet 01(hex) i accumalatoren.
0302  18      CLC  Slet eventuel mente
0303  8D0F80  STA  Skriv accumulatorindholdet i adresse 800F
0306  2020FA  JSR  Spring til pause-program på adresse FA20
0309  2A      ROL  Ryk til venstre: gang tallet med 2
030A  4C0303  JMP  Spring til adresse 0303
```

Der er to nye instruktioner i programmet: ROL og CLC - find dem i oversigten på side 34. ROL får bitværdierne til at rykke en plads mod venstre, hvilket svarer til at tallet bliver ganget med 2.

CLC er en instruktion, der bruges i forskellige sammenhænge, hvor man ønsker at være sikker på, at CPU-en ikke slæber rundt på en mente. Hvis CPU-en indeholder en mente, er der et såkaldt flag, der er høj. Menten skal slettes før CPU-en møder instruktioner, der bruger mente. Det gælder f.eks. for addition, hvor der så opnås, at der kun bruges de menter, der kommer fra selve regnestykket.

ROL- funktionen kan beskrives således:



Bemærk at når bit 7 har været høj, bliver mente-flaget høj, og denne mente føres nu ind i bit 0.

OPGAVE 16: Du kan dividere et binært tal med 2 ved at skubbe bitene en plads mod højre. Den op-kode, der laver dette hedder 6A (ROR). Find ud af hvilket tal accumulatoren nu skal indeholde til at begynde med og lav så dit program om, så løbelyset løber den anden vej.

OPGAVE 17: Ved at lade accumulatoren indeholde forskellige begyndelsesværdier, kan du få forskellige bitmønstre til at bevæge sig. Eksperimenter med dette.

OPGAVE 18: Prøv også at indføje et beep efter hvert gennemløb. Det gør du ved at kalde beep-eren ved hjælp af adresse 8006. ~~Det er ligegyldigt, hvilken op-kode, du bruger.~~ Prøv således at indføje dette før JMP:

```
8D0680  STA  Skriv til beep-er (adr. 8006)
```

VI TÆLLER OP OG NED

Umiddelbart har MIKRO-DASK en regnekapacitet, der er langt mindre end den, der kendes fra en lommeregner. Hvis du vil have den til at regne, må du først selv lære den det. Vi starter med et lille program, der kan tælle opad (i hex. tal!!!):

```
0500  EE0000  INC  Forøg indholdet i adresse 0000 med 1
0503  AD0000  LDA  Indlæs indholdet fra adresse 0000 til acc.
0506  8D0080  STA  Skriv indholdet af accumulatoren på display 0
0509  2020FA  JSR  Spring til pause-program ved FA00
050C  4C0005  JMP  Spring til toppen af programmet: 0500
```

Husk at pauselængden styres af indholdet i 00A1 og 00A3, hvor FF giver den længste pause.

Læg mærke til den nye instruktion: INC (increment memory by one = forøg indholdet i en hukommelsescelle (her celle 0000) med en).

De to øverste linier kan forenkles lidt. Der findes nemlig til de fleste instruktioner flere op-koder, som bruges til hver sit formål - se dette skema:

Instrukt.	Talværdi (Immidiate)	Komplet adr. (Absolute)	Side 0 (Zero page)
LDA	A9	AD	A5
STA	-	8D	85
INC	-	EE	E6
DEC	-	CE	C6

Vi tager LDA som eksempel. Ved den første søjle findes op-koden A9. Her forventes der en talværdi som operand (det der kommer efter op-koden). I i midten forventes en komplet adresse mellem 0000 og FFFF.
Ved den sidste søjle, forventer CPU-en at finde en adresse mellem 0000 og 00FF. Bag i bogen (side III) findes et komplet skema over samtlige instruktioner og deres op-koder. Prøv om du her kan finde de nævnte op-koder.

Adresserummet fra 0000 til 00FF kaldes også side 0 (Zero page), idet man opfatter det som værende delt op i (256) sider fra 00 til FF hver indeholdende (256) linier fra 00 til FF:

adresse		side	linie
00 00	=	00	00
00 AF	=	00	AF
02 01	=	02	01

De to første cifre angiver altså sidenummeret, mens de to sidste angiver linienummeret.

Side 0 (00) bruges ofte til på en hurtig og nem måde at gemme forskellige data. Derfor er der altså forskellige op-coder, der bruges til denne adressering. Her ser du programmet fra før, idet der dog er ændret i det, så der ved INC og LDA er benyttet side 0 adressering - operanden består kun af en byte (00), idet det er underforstået, at det drejer sig om side 0:

```
0500  E600    INC  Forøg indholdet i adr.00 (side 0) med 1
0502  A500    LDA  Indlæs indholdet fra adr.00 (side 0) til acc.
0504  8D0080  STA  Skriv indholdet af accumulatoren på display 0
0507  2020FA  JSR  Spring til pause-program ved FA20
050A  4C0005  JMP  Spring til toppen af programmet: 0500
```

OPGAVE 19: Forbind atter stepmotor-modulet til +5V og 0V. Sørg for at gøre "Set" lav og forbind "Trig" med bit 0 ved porten. Lav nu dit program om, så det både skriver ud

på display 0 og på porten. Nu vil motoren trigges ved, at bit 0 hele tiden skifter mellem høj og lav.
Find ud af hvordan motoren kører, når du forbinder "Trig" med bit 1, bit 2, bit 3 o.s.v. Sørg for at lægge passende pauseværdier ind i 00A1 og 00A3.

OPGAVE 20: Det modsatte af INC er DEC (decrement memory by one = formindsk indholdet i en hukommelsescelle med en). Få maskinen til at tælle baglæns ved at bruge denne instruktion - find den rigtige op-kode i skemaet side 38 eller i det komplette skema bag i bogen.

NB: Hvis du stadig har step-motoren forbundet, vil den ikke have ædret omdrejningsretning. Hvorfor ikke?

OPGAVE 21: Fjern stepmotoren og sørg for, at programmet fra opgave 20 kører nogenlunde langsomt. Få så beep-eren til at hyle hver gang, der tælles en ned (hver gang programmet er kørt igennem).

Vi vil nu prøve at få maskinen til at tælle opad i ti-talsystemet. I processorens instruktionssæt findes der en op-kode, der ordner den sag: F8 SED (set decimal mode). Funktionen virker imidlertid kun i forbindelse med addition (ADC) og subtraktion (SBC), der går via accumulatoren.

Du skal derfor lave et program, der ser således ud:

```
0600  F8      SED  Sæt decimal-mode. (Kør i ti-talsystemet)
0601  18      CLC  Slet menten, så den ikke bliver talt med
0602  6901    ADC  Læg tallet 01 sammen med accumulator.
0604  8D0080  STA  Skriv indholdet af accumulatoren på display 0
0607  2020FA  JSR  Spring til pause-program ved FA20
060A  4C0206  JMP  Spring til 0602
```

Her er et skema, der viser lidt om, hvilke op-koder, der passer til ADC og SBC i forbindelse med forskellige adresseringsformer:

instrukt.	talværdi (Immidiate)	komplet adr. (Absolute)	side 0 (Zero page)
ADC	69	6D	65
SBC	E9	ED	E5

Prøv atter at finde de tilsvarende op-koder i det komplette skema bag i bogen (side III).

Husk at en byte er på 8 bits - f.eks.: 1001 1011 eller i hex: 9B. Find ADC i skemaet. Her står der under "Absolute": 6D 4 3. Det betyder, at op-koden hedder 6D, at instruktionen varer 6 clock-perioder, og at hele instruktionen (op-kode + operand) fylder 3 bytes. Heraf fylder op-koden naturligvis den ene byte, mens operanden fylder 2 bytes, hvilket jo svarer til en komplet adresse (f.eks. 0200).

Hvis du stadigvæk holder dig til ADC, vil du se, at der under "Zero page" står: 65 3 2. Her varer instruktionen altså 3 clock-perioder, og den fylder 2 bytes: 1 byte til op-koden og 1 byte til operanden. Denne byte bruges jo til angivelse af, hvilken linie på side 0, det drejer sig om.

Find også CLC (clear carry flag) samt SED (set decimal mode) og bemærk at op-koderne for disse instruktioner står i en søjle med betegnelsen: "Implied".
Denne form for adressering går simpelthen ud på, at der ikke forventes en operand efter op-koden. Op-koden kan med andre ord det hele selv. Når der ved SED står: F8 2 1, betyder det altså, at instruktionen varer to perioder, og at det hele fylder 1 byte (op-koden selv).

OPGAVE 22: Bemærk at man altid sletter menten (18 CLC - clear carry flag), før man indleder et program med ADC. Modsat skal man i et program, der bruger SBC (subtract memory from accumulator with borrow) altid først sætte menten med op-koden 38 (SEC = set carry flag).
Prøv ud fra dette at ændre programmet, så det ved hjælp af SBC tæller baglæns i ti-talsystemet.

BETINGELSER

Noget af det, der kendetegner enhver datamaskine er, at den kan gøre een ting, hvis en betingelse er opfyldt og en anden ting, hvis betingelsen ikke er opfyldt. Dette kan måske i nogle sammenhænge forveksles med intelligens, men vi skal her se, at det dybest set drejer sig om, at datamaskinen kan sammenligne to tal og herefter enten fortsætte i programmet, eller springe et andet sted hen i det.

Vi vil studere fænomenet ud fra en venteløkke, der afventer, at der i forbindelse med kontroltasten bliver tastet et tal, der er forskellig fra 0. Indtast dette program:

```
0300  A900    LDA  Indlæs tallet 00 i acc. = nulstilling
0302  8D0580  STA  Nulstil kontroltastaturet
0305  CD0480  CMP  Sammenlign kontroltastatur med 00 fra acc.
0308  F0FB    BNE  Gå tilbage til 0305, hvis de er ens
030A  8D0080  STA  Skriv indholdet af acc. på display 0
030D  8D0180  STA  Skriv indholdet af acc. på display 1
0310  8D0280  STA  Skriv indholdet af acc. på display 2
0313  8D0680  STA  Skriv til beep-er
0316  4C1603  JMP  Spring til 0316 - løb på stedet
```

Bemærk at der i programmet bruges to nye adresser for tastaturet. Adresse 8004 bruges, når tastaturet skal aflæses i forbindelse med kontroltasten. Fordelen er her, at kontrol-tastaturet kan nulstilles ved hjælp af adresse 8005. Det sker simpelthen ved at CPU-en henvender sig til denne adresse enten for at læse eller for at skrive.

Programmet afventer som nævnt brug af kontroltasten sammen med et tal. Du kan enten holde "kontrol" nedtrykket, eller du kan skyde omskifteren hen mod venstre til ON - det har samme funktion. Samtidig hermed skal du så trykke på en tast. Hvis denne er forskellig frá 0, skal accumulatoren udskrives på alle 3 display - det vil sige, at der kommer til at stå 0 alle steder.

Programmet afbrydes med reset og kan så startes pånу - prøv det flere gange. Husk for øvrigt, at omskifteren igen skal stå på OFF, når du ønsker at indtaste eller rette et program.

Programmet er godt til at vise funktionen af de forskellige lysdioder. Inden du starter det vil både lysdioden ved memory 1 (EPROM-en) og memory 2 (RAM-en) lyse. Det skyldes, at maskinen kører i styreprogrammet, der jo ligger i EPROM-en. Styreprogrammet skriver imidlertid nogle tal i RAM-en. Derfor lyser lysdioden også her.

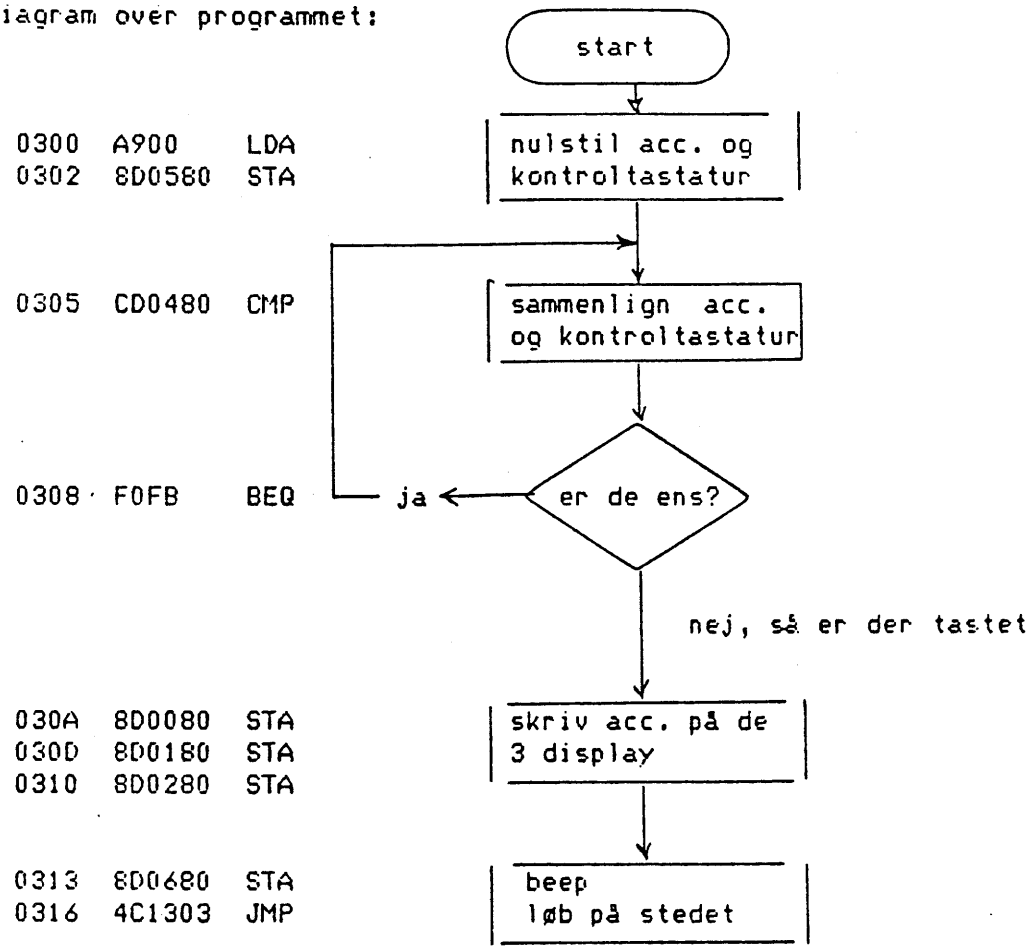
Hvis du ser godt efter, kan du se, at lysdioderne ved 8000-8002 på dekoderen lyser. Det skyldes, at styresystemet hele tiden er ved at skrive tal ud på displayene. Også lysdioden ved 8004 på dekoderen lyser. Det skyldes, at styresystemet hele tiden aflæser, om der har været trykket på en kontroltast.

Start nu programmet og bemærk, at det på memory-modulet nu kun er lysdioden ved RAM-kredsen, der lyser - processoren kører i det indtastede program. På dekoderen lyser lysdioden ved 8004 stadig: dit program aflæser jo hele tiden kontroltastaturet.

Når du trykker på en kontroltast, løber porcessoren på stedet - stadig i RAM, og der sker ingen henvendelser til andre adresser. Kontroller dekoderens og memory-modulets lysdioder.
Når du fremover skal finde fejl i dine programmer, kan du kontrol-

lere, om de rigtige lysdioder lyser. Du skal dog være klar over, at man kun kan se dem lyse, hvis programmet på en eller andet måde løber i ring. Hvis programmet kun en enkelt gang henvender sig til en adresse på dekoderen, vil lysudsendelsen være så kortvarig, at den slet ikke kan ses.

Diagram over programmet:



Prøv om du kan finde de forskellige op-koder i instruktionssættet bag i bogen. Det "intelligente" i programmet fremkommer ud fra op-koderne CMP (compare memory and accumulator = sammenlign memory og accumulator) og BEQ (branch om result zero = vælg forgreningen, hvis tallene er ens).

BEQ hører til en gruppe af instruktioner, der kaldes branch-instruktioner (forgrenings-instruktioner). De bruges altså i forbindelse med forgreninger i et program. I instruktionssættet finder du opkoden for BEQ og de andre branch-instruktioner (BCC BMI BNE BPL BVC BVS) i en søjle med betegnelsen "Relative".

Det betyder, at operanden udgør en relativ adresse. Det er ikke en

bestemt adresse, men et tal (off-set værdi), der skal lægges til den adresse, man står ved. Tallet findes på denne måde:

-7	-6	-5	-4	-3	-2	-1	0	+1	+2	+3	+4
F9	FA	FB	FC	FD	FE	FF	00	01	02	03	04
					DO	XX	**				

tæl herfra
her skal off-set skrives
her står branch op-koden

Den off-set værdi, du ønsker at finde, skal skrives ved XX. Pladsen lige efter er den, vi tæller ud fra (00). Hvis du altså ønsker at springe 4 bytes fremad i programmet, skal der stå 04 i stedet for XX.

Det er lidt mere besværligt, hvis vi ønsker at bevæge os bagud. CPU-en kan nemlig ikke regne med fortegn, så den forstår f.eks. ikke betegnelsen -3. Af tabellen kan du se, at der i stedet for skal skrives FD. I dette tal vil bit 7 være høj (bitmønster: 1111 1101). Bit 7 bruges her som fortegn: "1" betyder "-", mens "0" betyder "+". Det længste spring bagud bliver derfor et tal med dette bitmønster: 1000 0000 = 80(hex) = 128(ti), mens det længste spring fremad bliver et tal med dette bitmønster: 0111 1111 = 7F(hex) = 127(ti).

Det er ikke afgørende, om du forstår den sidste forklaring, blot du er i stand til at finde den rigtige off-set. Bag i bogen findes der en fuldstændig tabel til hjælp til dette. (Side 112).

OPGAVE 23: kontroller off-set værdien i disse to linier fra det sidste program:

```
0305  CD0480   CMP  Sammenlign kontr. med 00 fra acc.
0308  F0FB     BNE  Gå tilbage til 0305, hvis de er ens
```

ADDITION 1

Sørg for, at omskifteren på tastaturet står på OFF og indtast dette program:

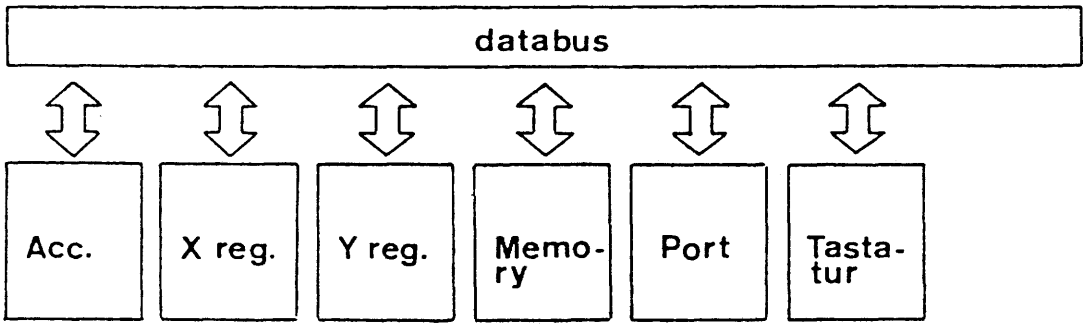
```
0200  F8       SED  Sæt decimal-mode. (Kør i ti-talsystemet)
0201  A200     LDX  Indlæs tallet 00 i x-reg. = nulstilling
0203  A900     LDA  Indlæs tallet 00 i acc. = nulstilling
```

```
0205  8D0580  STA  Nulstil kontroltastaturet
0208  8D0080  STA  Skriv indholdet af acc. på display 0
020B  EC0480  CPX  Sammenlign kontroltst. med 00 fra x-reg.
020E  F0FB    BEQ  Gå tilbage til 020C, hvis de er ens
0210  18      CLC  Slet menten, så den ikke bliver talt med
0211  6D0480  ADC  Læg acc. sammen med kontroltastatur
0214  4C0502  JMP  Spring til 0205
```

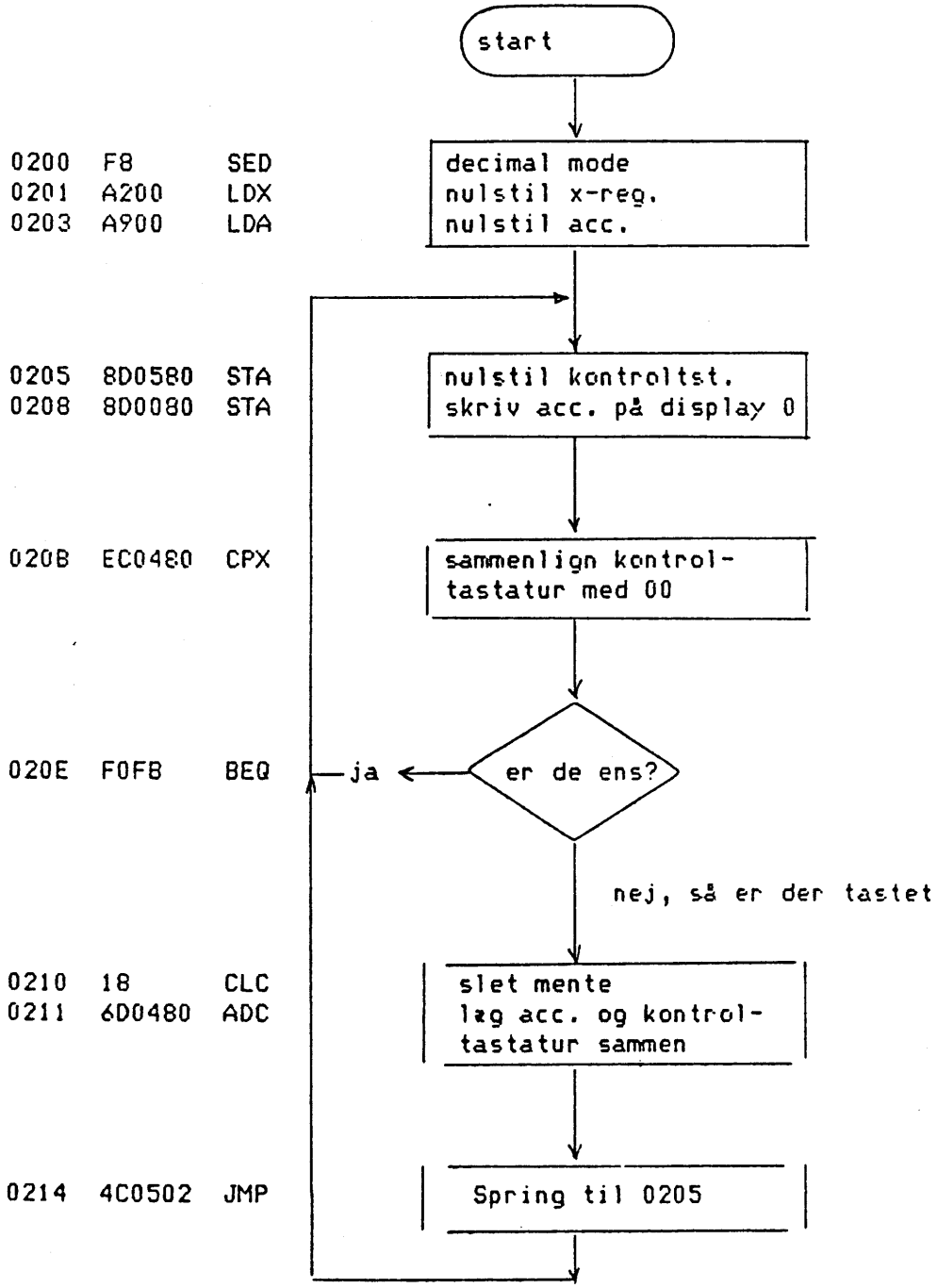
Start programmet og afprøv det: hold "kontrol" nede eller skyd omskifteren forneden på tastaturet hen på ON. Når du nu også trykker på et ciffer, skal summen af det nye og det gamle ciffer vises i display 0.

Programmet arbejder jo i ti-talsystem, så du må kun bruge tasterne mellem 0 og 9.

En nyhed i programmet er brugen af det såkaldte x-register. Foruden dette findes der et y-register. Det er registre, der med fordel kan benyttes til visse formål, hvis accumulatoren er optaget af noget andet.



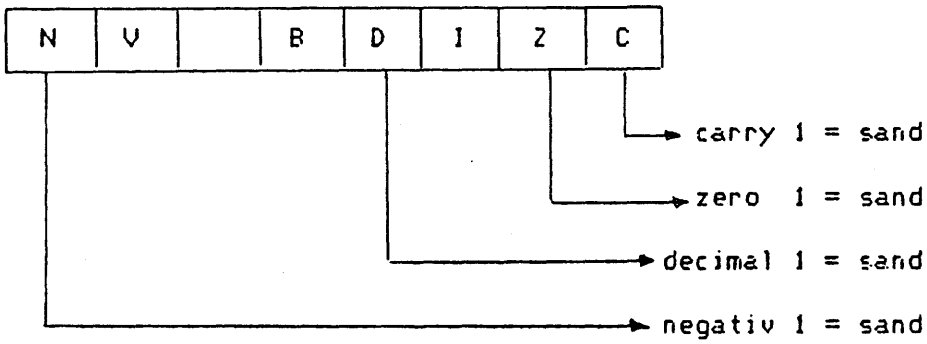
I programmet her rummer accumulatoren jo hele tiden facit, så for ikke at forstyrre dette, lader vi x-registeret indeholdet tallet 00, som bruges til at sammenligne med kontroltastaturets indhold. Dette sker i "venteløkken" ved adresse 020C-020F. Denne løkke afventer, at der indtastes noget på kontroltastaturet, idet tastaturet så ikke længere indeholder tallet 00. På næste side er et diagram over programmet:



OPGAVE 24: Som nævnt skal SBC (subtract memory from accumulator with borrow) altid indledes med SEC (set carry flag). Lav programmet om, så tallene fra kontroltastaturet trækkes fra det tidligere facit.

STATUSREGISTER OG BRANCH

Vi skal nu se nærmere på, hvordan man opstiller betingelser i forbindelse med branch. For at du kan forstå, hvorledes det går til, skal du vide lidt om det såkaldte statusregister. Det består af 8 bits, der hver for sig kaldes flags. Det hænger sammen med at de enkelte bits kan fortælle os noget om CPU-ens tilstand ved at "hejse flaget" (bitværdien er 1) eller ved at "fjerne flaget" (bitværdien er 0).



Du vil i første omgang kun få brug for en del af statusregistrets bits, så vi vil nøjes med at beskæftige os med dem:

1. Mente-flaget (Carry = C) har værdien 1, hvis der ligger en mente, som bliver talt med ved addition. Flaget kan sættes og fjernes med instruktionerne SEC og CLC. Det skal altid fjernes før addition og sættes før subtraktion.

2. Nul-flaget (Zero = Z) bruges i forbindelse med sammenligning af tal i x-register, y-register eller accumulator med andre tal (f.eks. fra memory). Hvis tallene er ens, bliver flaget 1, hvis de er forskellige, bliver det 0.
   Betegnelsen for flaget er lidt forvirrende: ved en sammenligning af de to tal sker der en subtraktion. Hvis de to tal er lige store, bliver facit 0, og nulflaget sættes til 1. Facit på 0 skrives imidlertid ikke i accumulatoren - den bevarer sit oprindelige indhold!!!

3. Decimal-flaget (Decimal mode) viser, om CPU-en arbejder i titalsystem eller i hex-tal. Flaget sættes og fjernes med SED og CLD. Decimal-mode virker kun på ADC og SBC instruktionerne.

4. Negativ-flaget (Negative = N) sættes, hvis bit 7 er høj, idet

tallet så i nogle sammenhænge opfattes som negativt (bit 7 = 1 svarer til "-", mens bit 7 = 0 svarer til "+").

Vi har tidligere benyttes os af statusregistret i forbindelse med betingelser. Studer f.eks. denne programstump:

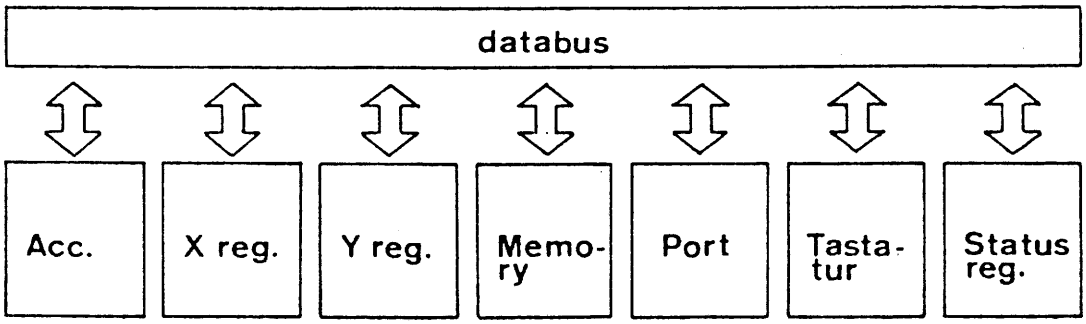
```
020C  EC0480  CPX  Sammenlign kontroltst. med 00 fra x-reg.
020F  F0FB    BEQ  Gå tilbage til 020C, hvis de er ens
```

I den øverste linie sammenlignes kontroltastaturets tal med tallet 00, der ligger i x-registret. Hvis disse to tal er ens, vil nulflaget sættes til 1. I den nederste linie undersøges dette flag med BEQ. Denne instruktion får CPU-en til at vælge forgreningen, hvis Z = 1 (branch on Z=1). Sammenlign med instruktionssættet bag i bogen.
Hvis Z ikke er = 1 (de sammenlignede tal er forskellige), vil processoren fortsætte lige ud gennem programmet - den vil ikke "branche".

Du kan nu selv prøve at studere de andre branch-instruktioner (CMP CPX COY) ud fra instruktionssættet bag i bogen. Her er en oversigt over, hvordan de påvirker de forskellige flag:

	N	Z	C
Acc., x eller y < memory	1	0	0
Acc., x eller y = memory	0	1	1
Acc., x eller y > memory	0	0	1

Vi kan nu tegne statusregistret med i vores diagram af datamaskinen:



ADDITION 2

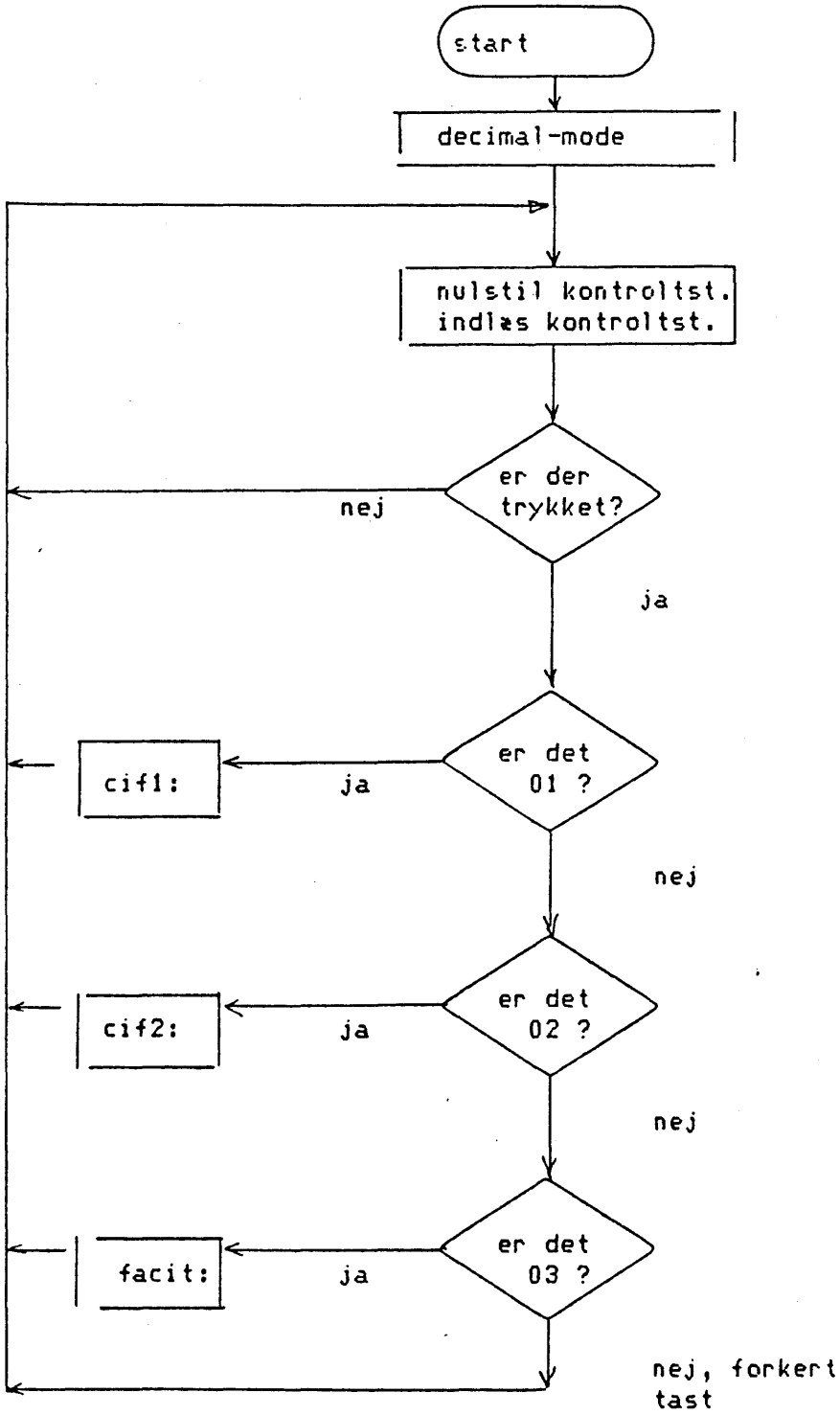
Indtast nu dette program:

```
  0300  F8      SED  Decimal-mode. Kør i ti-talsystem

forfra:
  0301  AD0580  LDA  Nulstil kontroltastaturet
  0304  AD0480  LDA  Indlæs kontroltast til acc.
  0307  C900    CMP  Sammenlign acc. med 00
  0309  F0F9    BEQ  Gå tilbage til 0304, hvis de er ens
  030B  C901    CMP  Sammenlign acc. med tallet 01
  030D  F00B    BEQ  Gå til cif1: hvis de er ens
  030F  C902    CMP  Sammenlign acc. med tallet 02
  0311  F012    BEQ  Gå til cif2: hvis de er ens
  0313  C903    CMP  Sammenlign acc. med tallet 03
  0315  F019    BEQ  Gå til facit: hvis de er ens
  0317  4C0103  JMP  Det var en forkert tast - spring til forfra:

cif1:
  031A  AD0380  LDA  Indlæs ciffer 1 fra tastaturet
  031D  8D0280  STA  Skriv acc. (ciffer 1) på display 2
  0320  8501    STA  Skriv acc. (ciffer 1) på adresse 0001
  0322  4C0103  JMP  Spring til forfra:

cif2:
  0325  AD0380  LDA  Indlæs ciffer 2 fra tastaturet
  0328  8D0180  STA  Skriv acc. (ciffer 2) på display 1
  032B  8502    STA  Skriv acc. (ciffer 2) på adresse 0002
  032D  4C0103  JMP  Spring til forfra:

facit:
  0330  18      CLC  Nulstil mente
  0331  A501    LDA  Indlæs ciffer 1 fra adr. 0001
  0333  6502    ADC  Læg ciffer 1 og ciffer 2 fra adr. 0002 sammen
  0335  8D0080  STA  Skriv acc. (facit) på display 0
  0338  4C0103  JMP  Spring til forfra:
```

Programmet skal anvendes således: start det ved 0300, indtast ciffer 1 på tastaturet og skriv dette ciffer i display 2 (og i adresse 0001) ved hjælp af "kontrol"+1. Indtast herefter ciffer 2 og skriv det i display 1 (og i adresse 0002) ved hjælp af "kontrol"+2. Nu fås facit på display 0 ved hjælp af "kontrol"+3.

Programmet viser, at kontroltasterne nu har fået en anden

betydning. I den første programdel afventes der et tryk på en kontroltast. Dernæst undersøges det hvilken tast, det drejer sig om. Det sker ved at der spørges, om den har talværdien 01 - 02 eller 03.



Hvis talværdien er 01, er der blevet trykket på "kontrol"+1, og der hoppes til det programafsnit, der har fået betegnelsen "cif1:". En sådan betegnelse kaldes en label. Labels bruges for at give bedre oversigt over programmet.

På tilsvarende måde bevirker "kontrol"+2 at der springes til "cif2:", mens "kontrol"+3 bevirker et spring til "facit:". Prøv selv at kontrollere off-set værdierne i forbindelse med BEQ-instruktionerne. Husk at pladsen efter det sted, hvor off-set værdien står, skal have nummer 00.

OPGAVE 25: Lav programmet om, så det trækker fra. Husk at sætte menten.

INDEX-ADRESSERING

Programmet her skriver 00 ved adresse 0000-000F. Indtast det og afprøv det.

```
0200  A900    LDA  Indlæs tallet 00 i acc.
0202  A200    LDX  Indlæs tallet 00 i x-reg.
0204  9500    STA  Skriv acc. ved adresse 0000+X
0206  E8      INX  Forøg x-reg. med 1
0207  E010    CPX  Sammenlign x-reg. med tallet 10 (er der skrevet på alle pladser?)
0209  D0F9    BNE  Gå til 0204 hvis de er forskellige
020B  8D0680  STA  Skriv til beep-er
020E  4C00F8  JMP  Spring til styresystemet
```

Undersøg om programmet har skrevet 00 i adresse 0000-000F. Kig så nærmere på denne linie:

```
0204  9500    STA  Skriv acc. ved adresse 0000+X
```

Der er her brugt en adresseringsform, der kaldes "Z. page, X" = Side 0, X. Find den benyttede op-kode i instruktionssættet og bemærk, at der også findes en "Z. page, Y" - adressering.

Endvidere findes der "ABS. X" og "ABS. Y". Også her adresseres med x- og y-registret, men her er det muligt at benytte hele adresserummet og ikke kun side 0. Under et kaldes de 4 adresseringsformer for index-adressering.

OPGAVE 26: Lav programmet om, så det skriver AA ved adresse 0600-

060F. Du skal altså bruge index-adressering, der henvender sig til hele adresserummet.

ADDITION 3

Ved hjælp af programmet herunder kan du skrive forskellige tal ved adresse 0001-000F. Indtast det og afprøv det således: skriv et tal på tastaturet og tryk på "kontrol"+1 for at lægge tallet i adresse 0001. Med "kontrol"+2 kan du lægge det på adresse 0002 og så fremdeles. Stands programmet og undersøg, om de indtastede data ligger ved de pågældende adresser.

```
0300  AD0580  LDA  Nulstil kontroltastaturet
0303  AD0480  LDA  Indlæs kontroltast i acc.
0306  C900    CMP  Sammenlign acc. med 00
0308  F0F9    BEQ  Gå tilbage til 0303, hvis de er ens

030A  AA      TAX  Skriv acc. i x-reg.
030B  AD0380  LDA  Indlæs ciffer fra tastaturet
030E  9500    STA  Skriv acc. (ciffer) på adr. 00+X
0311  8D0080  STA  Skriv acc. (ciffer) på display 0
0313  4C0003  JMP  Spring til 0300
```

Øverste linier danner en venteløkke, der afventer indtastning ved hjælp af en kontroltast. Værdien af den aktuelle kontroltast indlæses i accumulatoren og føres videre ud i x-registret ved hjælp af denne operation:

```
030A  AA      TAX  Skriv acc. i x-reg.
```

Herefter aflæses tastaturet, og i linien herunder føres den aflæste værdi ind i en adresse, der er lig 0000 + indholdet af x-registret:

```
030E  9500    STA  Skriv acc. (ciffer) på adr. 00+X
```

Hvis der altså tastes på "kontrol"+5, vil x-registret komme til at indeholde tallet 05, og der vil blive skrevet i adresse 0005. Der er altså atter tale om index-adressering.

På næste side ses ved "forfra:" næsten det samme program, blot er der indføjet en mulighed for et spring til "facit:". Dette vil ske

ved "kontrol"+7, og der vil blive udskrevet et facit.

Programmet bruges således: læg 2 seks-cifrede tal ind i adresse 0001-0006 ved at benytte "kontrol" samt tast 1-6. Herunder er vist hvorledes tallene 423528 og 360755 kan indlægges:

```
                    42  35  28
kontrol              1   2   3

                    36  07  55
kontrol              4   5   6
```

Summen af de to tal fås herefter ved "konrol"+7.

```
  0300  F8       SED  Decimal-mode. Kør i ti-talsystem

forfra:
  0301  AD0580   LDA  Nulstil kontroltastaturet
  0304  AD0480   LDA  Indlæs kontroltast i acc.
  0307  C900     CMP  Sammenlign acc. med 00
  0309  F0F9     BEQ  Gå tilbage til 0304, hvis de er ens
  030B  C907     CMP  sammenlign acc. med 07
  030D  F00C     BEQ  Gå til facit:
  030F  AA       TAX  Skriv acc. i x-reg.
  0310  AD0380   LDA  Indlæs ciffer fra tastaturet
  0313  9500     STA  Skriv acc. (ciffer) på adr. 00+X
  0316  8D0080   STA  Skriv acc. (ciffer) på display 0
  0318  4C0103   JMP  Spring til forfra:

facit:
  031B  18       CLC  Slet mente
  031C  A503     LDA  Indlæs tal fra adr. 0003
  031E  6506     ADC  Læg tal fra adresse 0006 til
  0320  8D0080   STA  Skriv delfacit 1 på display 0
  0323  A502     LDA  Indlæs tal fra adr. 0002
  0325  6505     ADC  Læg tal fra adresse 0005 til
  0327  8D0180   STA  Skriv delfacit 2 på display 1
  032A  A501     LDA  Indlæs tal fra adr. 0001
  032C  6504     ADC  Læg tal fra adresse 0004 til
  032E  8D0280   STA  Skriv delfacit 3 på display 2
  0331  4C0103   JMP  Spring til forfra:
```

NB: programmet findes i en lignende udgave i EPROM-en ved adresse FC00.

Hvis du har indtastet tallene 568930 og 234512, vil de forskellige cifre ligge således i hukommelsen:

adresse	0001	0002	0003
	56	89	30
adresse	0004	0005	0006
	23	45	12

I "facit:" indledes med at slette menten. Herefter lægges tallene fra 0003 og 0006 sammen. Herved kan der opstå en mente, der nu ikke slettes, men bruges ved den kommende addition: adresse 0002 og 0005. På tilsvarende måde lægges adresse 0001 sammen med 0004. Efter hver sammenlægning udskrives det opnåede delfacit på et display.

Eksemplet viser, hvorledes en datamaskine kan behandle store tal, selv om den kun kan behandle en byte ad gangen. Der er således ikke noget i vejen for at udvide programmet til flere cifre, blot vil der mangle display til udskrivning.

MIKRO-DASK SOM UR

I EPROM1-en ligger der et program, der kan få MIKRO-DASK til at fungere som et ur. Find det ved F940 og prøv det.

Nu skal display 0 vise sekunder, mens display 1 viser minutter og display viser timer. Programmet ser således ud:

```
initiering:
  F940  A900    LDA  Indlæs tallet 00
  F942  8550    STA  Nulstil sekunder
  F944  8551    STA  Nulstil minutter
  F946  8552    STA  Nulstil timer
  F948  A9F9    STA  Indlæs tallet FF
  F94A  85A1    STA  Indstil kort_pause
  F94C  A9F4    LDA  Indlæs tallet F5
  F94E  85A3    STA  Indstil mellem_pause

ur:
  F950  2060F9  JSR  Spring til sekunder:
  F953  20B0F9  JSR  Spring til udlæs:
  F956  2020FA  JSR  Spring til mellem_pause
  F959  4C50F9  JMP  Spring til ur:

sekunder:
  F960  F8      SED  Sæt decimal mode
  F961  18      CLC  Slet menten
  F962  A550    LDA  Indlæs tal fra adr. 0050 = sekunder
  F964  6901    ADC  Læg 1 til sekunder
```

```
  F966  8550    STA  Skriv sekunder på adr. 0050
  F968  C960    CMP  Sammenlign med tallet 60
  F96A  D003    BNE  Spring 3 frem hvis de er forskellige
  F96C  2070F9  JSR  Spring til minutter:
  F96F  60      RTS  Retuner igen

minutter:
  F970  A900    LDA  Indlæs tallet 00
  F972  8550    STA  Nulstil sekunder (adr. 0050)
  F974  A551    LDA  Indlæs tal fra adr. 0051 = minutter
  F976  18      CLC  Slet menten
  F977  6901    ADC  Læg 1 til minutter
  F979  8551    STA  Skriv minutter i adr. 0051
  F97B  C960    CMP  Sammenlign med tallet 60
  F97D  D003    BNE  Spring 3 frem hvis de er forskellige
  F97F  2090F9  JSR  Spring til timer:
  F982  60      RTS  Retuner igen

timer:
  F990  A900    LDA  Indlæs tallet 00
  F992  8551    STA  Nulstil minutter
  F994  A552    LDA  Indlæs timer fra adr. 0052
  F996  18      CLC  Slet menten
  F997  6901    ADC  Læg 1 til timer
  F999  C924    CMP  Sammenlign med tallet 24
  F99B  D002    BNE  Spring 2 frem hvis de er forskellige
  F99D  A900    LDA  Nulstil timer
  F99F  8552    STA  Skriv timer i adr. 0052
  F9A1  60      RTS  Retuner igen

udlæs:
  F9B0  A550    LDA  Indlæs sekunder
  F9B2  8D0080  STA  Skriv sekunder på display 0
  F9B5  A551    LDA  Indlæs minutter
  F9B7  8D0180  STA  Skriv minutter på display 1
  F9BA  A552    LDA  Indlæs timer
  F9BC  8D0280  STA  Skriv timer på display 2
  F9BF  60      RTS  Retuner igen
```

Programmet sørger selv for at indlæse nogle passende pause-værdier, så det går nogenlunde nøjagtigt. Det sker i "initiering:" ved at der i 00A1 indlæses tallet FF og i 00A3 indlæses F5.

Start uret og find ud af, om sekunderne går på 00 efter 59. Det vil imidlertid tage en del tid at finde ud af, om det samme gælder for minutterne, og om timerne viser 00 efter 23.

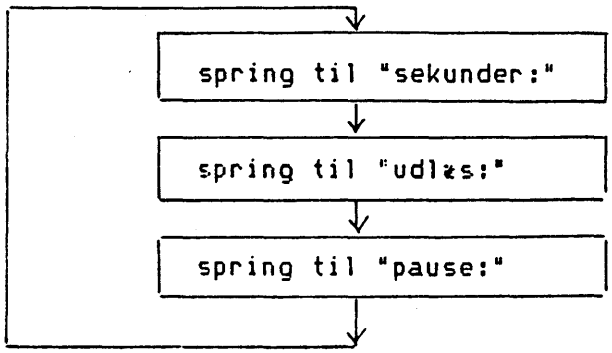
Du kan imidlertid pille ved pause-værdierne, så uret kommer til at gå for stærkt. Så skal det imidlertid ikke startes fra F940 men

fra F950 ellers nytter det hele ingenting.

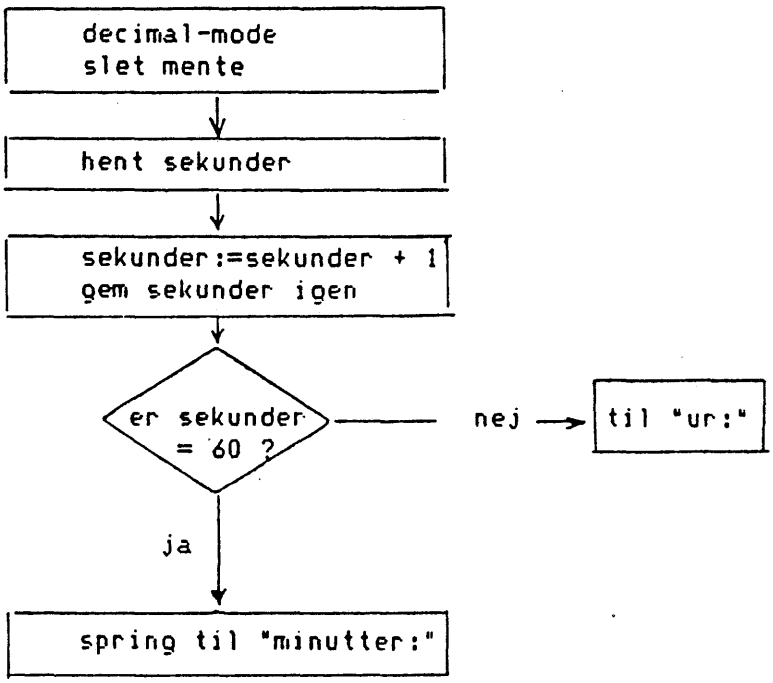
Prøv derfor at skrive 01 i adresse 00A3 og start uret fra F950. Kontroller minutterne.

Skriv så 01 i adresse 00A1 og start igen uret fra F950. Kontroller nu om timerne skifter rigtigt.

Bemærk nu programdelen "ur:". Den ser således ud på diagramform:



Her ser du et diagram over "sekunder:" - prøv om du selv kan lave et lignende diagram over "minutter:" og "timer:"



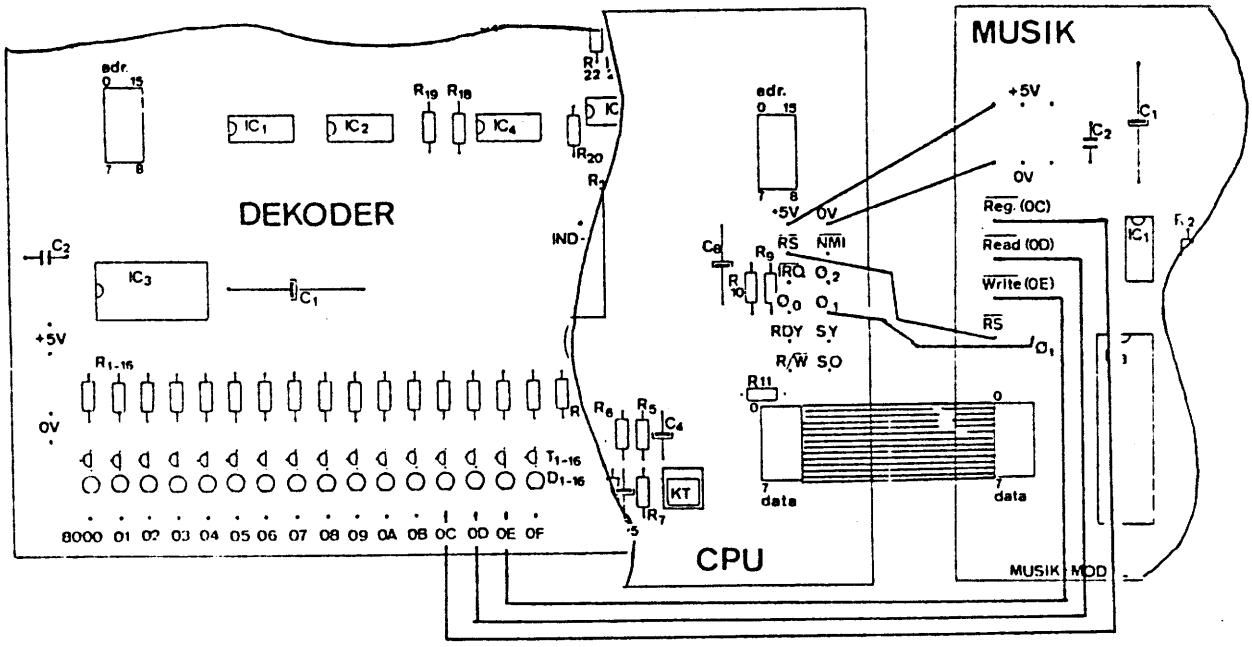
MUSIK

Du skal nu bruge MIKRO-DASK til at styre en musik-kreds. Du skal derfor bruge et musikmodul, der bl.a. indeholder en sådan kreds samt en forstærker. Forstærkeren kan trække en højttaler, som derfor også skal forbindes.

Musikmodulet skal forbindes til databussen ved CPU-modulet. Endelig skal det forbindes til: +5V, 0V, "Ø1" og "Reset" på CPU-modulet. Ved den sidste forbindelse opnås, at også musikkredsen resettes ved et tryk på "Reset". Fra "Ø1"modtager den clock-signaler med en frekvens på 1.000.000Hz, som afgives af CPU-modulets krystal. Det er denne frekvens, der af musikkredsen omsættes til hørebare toner.

Til sidst skal musikmodulets styreindgange forbindes således til adressedekoderen:

"Register"	forbindes til	800C
"Read"	forbindes til	800D
"Write"	forbindes til	800E



Indtast så dette program og prøv om det virker. Det skal få musikkredsen til at afgive en tone på 440Hz (A):

```
mixer:
  0300  A907    LDA  Indlæs tallet 07 i acc.
  0302  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 7
  0305  A9FE    LDA  Indlæs tallet FE (vælg tone ved kanal A)
  0307  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 7

volumen:
  030A  A908    LDA  Indlæs tallet 08 i acc.
  030C  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 8
  030F  A90F    LDA  Indlæs tallet 0F (max volumen)
  0311  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 8

grov_tone:
  0314  A901    LDA  Indlæs tallet 01 i acc.
  0316  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 1
  0319  A900    LDA  Indlæs tallet 00 (= 440Hz, grov tone)
  031B  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 1

fin_tone:
  031E  A900    LDA  Indlæs tallet 00 i acc.
  0320  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0
  0323  A98E    LDA  Indlæs tallet 8E (= 440Hz, fin tone)
  0325  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0

  0328  8D0680  STA  Skriv til beep-er
  032B  4C00FF  JMP  Spring til styresystem
```

Bemærk at tonen fortsætter, selv om programmet forlængst er kørt igennem. Du resetter musikkredsen samtidig med, at du resetter datamaskinen.

De 4 labels i programmet dækker over 4 forskellige registre i musikkredsen. Hver register har en betydning for den lyd, kredsen afgiver. Kredsen indeholder i øvrigt i alt 16 registre. Prøv i det følgende at sammenholde forklaringerne af musik-kredsen med oversigten bag i bogen (side 113).

"Mixer:"

Register 07 har indflydelse på en indbygget mixer. Kredsen kan nemlig afgive toner fra 3 generatorer (A-B-C). Disse toner kan fortsætte videre gennem kanal A-B-C, hvis mixeren ellers tillader det. Endelig kan mixeren sørger for, at der sendes støj fra en indbygget støjgenerator til kanal A-B-C. En kanal åbnes ved, at den tilsvarende bit gøres lav. Tallet FE åbner således til kanal A.

"Volumen:"

Register 08 bestemmer lydstyrken for generator A. Tallet 0F giver den højeste værdi.

"Grov_tone, fin_tone:"

Register 01 bestemmer sammen med register 00 frekvensen for generator A. Det er register 01, der laver den grove indstilling (normalt med tallet 00), mens register 00 laver den fine indstilling. Her giver forskellige talværdier hver sin frekvens - se oversigten herunder og bemærk, at der ved 440Hz netop står 8E:

Tone	Frekvens Hz	Register 00 fin	Register 01 grov
h	247	1111 1101 = FD	00
c	262	1110 1111 = EF	00
cis	277	1110 0010 = E2	00
d	294	1101 0101 = D5	00
dis	311	1100 1001 = C9	00
e	330	1011 1101 = BD	00
f	349	1011 0011 = B3	00
fis	370	1010 1001 = A9	00
g	392	1001 1111 = 9F	00
gis	415	1001 0111 = 97	00
a'	440	1000 1110 = 8E	00
ais	466	1000 0110 = 86	00
h	494	0111 1111 = 7F	00
c	523	0111 1000 = 78	00
cis	554	0111 0001 = 71	00
d	587	0110 1010 = 6A	00
dis	622	0110 0100 = 64	00
e	659	0101 1111 = 5F	00

Når de forskellige talværdier skal indlæses i registrene, skal det foregå efter denne opskrift:

1. Fortæl, at du vil finde et register ved at bruge adresse 800C
2. Hvilket register? - skriv et tal mellem 00 og 0F
3. Fortæl, at du vil skrive i registret ved at bruge adresse 800E
4. Hvilke data vil du skrive?

Vi kan studere metoden ud fra "fin_tone:" fra programmet ovenfor:

```
031E  A900    LDA
0320  8D0C80  STA
```

henvendelse til register 00 (fin_tone)

```
0323  A98E    LDA
0325  8D0E80  STA
```

tallet 8E skrives i register 00

Det er for øvrigt også muligt at læse fra et register. Hvis man ønsker dette, skal man bruge adresse 800D.

OPGAVE 27: Lav programmet om, så det afgiver en tone med en frekvens på 587Hz (d). Prøv også andre frekvenser.

FLERE TONER

Lav nu dit program om, så det ser ud som vist herunder. Læg mærke til, at der nu er to afdelinger med "fin_tone:" - blot med forskellige frekvenser. Endelig er der indlagt 2 pauser.

```
mixer:
  0300  A907    LDA  Indlæs tallet 07 i acc.
  0302  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 7
  0305  A9FE    LDA  Indlæs tallet FE (vælg tone ved kanal A)
  0317  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 7

volumen:
  030A  A908    LDA  Indlæs tallet 08 i acc.
  030C  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 8
  030F  A90F    LDA  Indlæs tallet 0F (max volumen)
  0311  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 8

grov_tone:
  0314  A901    LDA  Indlæs tallet 01 i acc.
  0316  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 1
  0319  A900    LDA  Indlæs tallet 00 (grov tone)
  031B  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 1

fin_tone1:
  031E  A900    LDA  Indlæs tallet 00 i acc.
  0320  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0
  0323  A98E    LDA  Indlæs tallet 8E (= 440Hz, fin tone)
  0325  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0
  0328  2020FA  JSR  Spring til pause

fin_tone2:
  032B  A900    LDA  Indlæs tallet 00 i acc.
  032D  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0
  0330  A9BD    LDA  Indlæs tallet BD (= 349Hz, fin tone)
  0332  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0
  0335  2020FA  JSR  Spring til pause
  0338  4C1E03  JMP  Spring til "fin_tone1:"
```

Du kan selv indtaste passende pause-værdier.

Som du kan se, er den eneste forskel på "fin_tone1:" og fin_tone2:", at der bruges to forskellige talværdier. Det vil derfor være oplagt at "fikse" programmet lidt op, således, at der ikke indlæses et fast tal for frekvensen, men i stedet for sker en henvendelse til forskellige adresser med forskellige frekvensværdier.

I programmet herunder er "mixer:", "volumen:" og "grov_tone:" ikke ændret - det er derimod "fin_tone:"

```
mixer:
  0300  A907    LDA  Indlæs tallet 07 i acc.
  0302  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 7
  0305  A9FE    LDA  Indlæs tallet FE (vælg tone ved kanal A)
  0317  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 7

volumen:
  030A  A908    LDA  Indlæs tallet 08 i acc.
  030C  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 8
  030F  A90F    LDA  Indlæs tallet 0F (max volumen)
  0311  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 8

grov_tone:
  0314  A901    LDA  Indlæs tallet 01 i acc.
  0316  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 1
  0319  A900    LDA  Indlæs tallet 00 (grov tone)
  031B  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 1

fin_tone:
  031E  A200    LDX  Indlæs tallet 00 i x-reg. = nulstilling
  0320  A900    LDA  Indlæs tallet 00 i acc.
  0322  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0

  0325  B500    LDA  Indlæs frekvenstal fra adr. 0000+X
  0327  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0
  032A  2020FA  JSR  Spring til pause

  032D  A900    LDA  Indlæs tallet 00 (giver pause i lyden)
  032F  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0
  0332  2020FA  JSR  Spring til pause

  0335  E00F    CPX  Sammenlign X med F (F gennemløb?)
  0337  F0E5    BEQ  Hvis X = F gå til 031E og nulstil X
  0339  E8      INX  Forøg X med 1, så der kontaktes en ny adr.
  033A  4C2003  JMP  Spring til 0320 og lav et nyt gennemløb
```

Som du kan se, er "fin_tone:" nu opbygget af 4 dele. Den øverste

del nulstiller X, der bruges til at finde de forskellige data med, og den sørger for at finde register 0 (fin_tone registret).

Den næste afdeling indlæser tal fra fra adresse 0000+X (index-adressering). Springet til pausen gør, at tonen holdes et stykke tid.

Den tredje afdeling lægger tallet 00 ind i fin_tone registret. Det gør, at der ingen lyd frembringes. Springet til pausen gør nu, at der ikke kommer lyd i et stykke tid: pause mellem tonerne.

Den sidste afdeling sammenligner X med et fast tal (her F) for at undersøge, om tonerækken er blevet gennemløbet. Hvis dette ikke er tilfældet, forøges X med 1.

Nu mangler du at indlægge nogle frekvenstal i adresse 0000 til 000F. Der er "frit spil" - brug skemaet fra side 58.

OPGAVE 28: Indtast disse toner i adresse 0000-000F. Sammenlign tallene med skemaet fra side 58:

D5-9F-9F-9F-A9-8E-8E-8E-78-8E-A9-D5-BD-A9-9F-00 (PAUSE)

Hvis du vil prøve at indlægge flere eller færre toner, skal du i denne linie ændre det tal, som X sammenlignes med, så det passer med antallet af toner:

```
  0335  E00F    CPX  Sammenlign X med F (F gennemløb?)
```

MIKRO-DASK SOM KLAVER

Her er det igen den sidste del af programmet, der er ændret. Ændringerne starter ved adresse 031E ved "grov_tone:", men for overskuelighedens skyld vises hele programmet.

```
mixer:
  0300  A907    LDA  Indlæs tallet 07 i acc.
  0302  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 7
  0305  A9FE    LDA  Indlæs tallet FE (vælg tone ved kanal A)
  0317  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 7

volumen:
  030A  A908    LDA  Indlæs tallet 08 i acc.
  030C  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 8
```

```
  030F  A90F    LDA  Indlæs tallet 0F (max volumen)
  0311  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 8

grov_tone:
  0314  A901    LDA  Indlæs tallet 01 i acc.
  0316  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 1
  0319  A900    LDA  Indlæs tallet 00 (grov tone)
  031B  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 1
  031E  4C0002  JMP  Spring til "taster:"
```

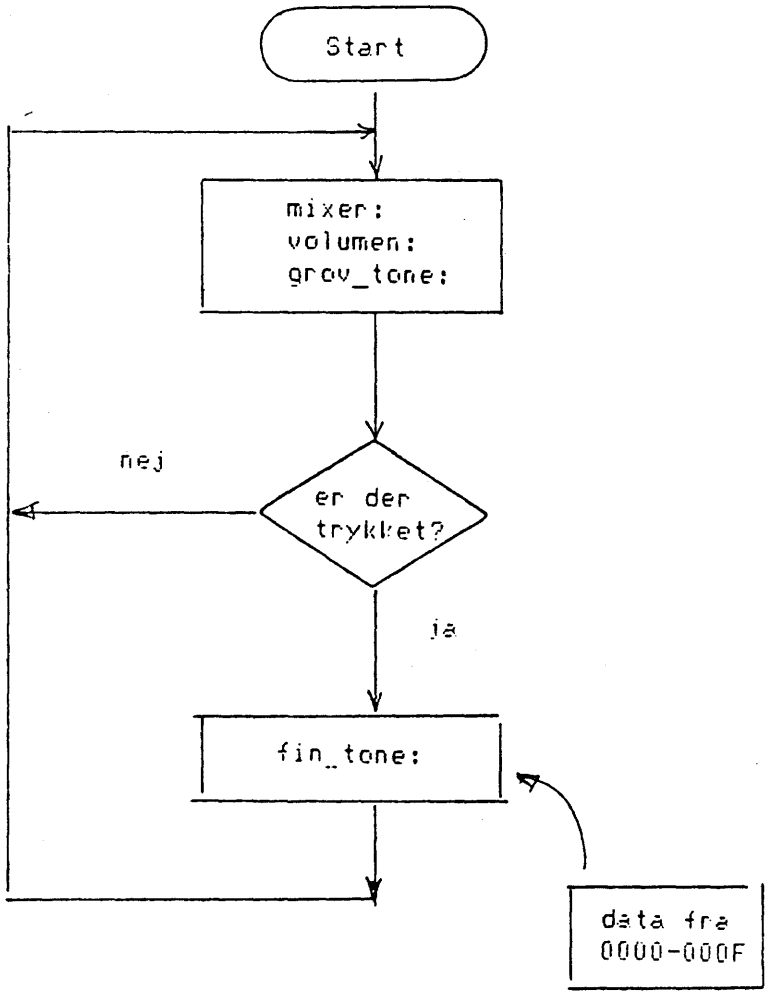
Bemærk springet i adresser!!

```
fin_tone:
  0330  A900    LDA  Indlæs tallet 00 i acc.
  0332  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0
  0335  B500    LDA  Indlæs tal fra adresse 0000+X
  0337  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0
  033A  4C0002  JMP  Spring til "taster:"

taster:
  0200  AD0580  LDA  Nulstil kontroltastaturet
  0203  AE0480  LDA  Indlæs kontroltastaturet i x-reg.
  0206  E000    CPX  Sammenlign x-reg. med 00
  0208  F0F9    BEQ  Gå tilbage til 0203, hvis de er ens
  020A  4C3003  JMP  Gå til "fin_tone:"
```

Nu taster du værdierne for en del af toneskalaen ind i adresse 0000-000F (sammenlign igen med skemaet på side 58):

00-EF-E2-D5-C9-BD-B3-A9-9F-97-8E-86-7F-78-71-6A

Programmet startes fra adresse 0300. Nu vil et tryk på forskellige kontroltaster give forskellige toner: tast 5 henter således værdien fra adresse 0005, tast F henter værdien fra 000F osv. På diagramform ser det ud som vist på næste side:



VIBRATO

Dit klaver har nu den fejl, at det holder den tone, der er anslået, indtil der anslås en ny. Dette kan vi rette ved hjælp af den såkaldte "envelope-generator". Det er en indretning i musikkredsen, der kan påvirke tonernes styrke.

Det er derfor nødvendigt endnu engang at ændre programmet. For overskuelighedens skyld vises programmet igen i sin helhed, men "mixer:", "volumen:", "grov_tone:" og "fin-tone" er uforandret med undtagelse af adresse 0310 - hvor der skal stå:

```
  030F  A91F    LDA  Indlæs tallet 1F (envelope kontrollerer vol)
```

Sammenlign ændringen med skemat over musik-kredsens registre bag i bogen (side 113). I adresse 0000-000F skal fortsat ligge talværdier for de forskellige frekvenser - se side 62.

```
mixer:
  0300  A907    LDA  Indlæs tallet 07 i acc.
  0302  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 7
  0305  A9FE    LDA  Indlæs tallet FE (vælg tone ved kanal A)
  0317  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 7

volumen:
  030A  A908    LDA  Indlæs tallet 08 i acc.
  030C  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 8
  030F  A91F    LDA  Indlæs tallet 1F (envelope kontrollerer vol)
  0311  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 8

grov_tone:
  0314  A901    LDA  Indlæs tallet 01 i acc.
  0316  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 1
  0319  A900    LDA  Indlæs tallet 00 (grov tone)
  031B  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 1
  031E  4C0002  JMP  Spring til "taster:"
```

Bemærk springet i adresser!

```
fin_tone:
  0330  A900    LDA  Indlæs tallet 00 i acc.
  0332  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0
  0335  B500    LDA  Indlæs tal fra adresse 0000+X
  0337  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0

envelope_tid:
  033A  A90B    LDA  Indlæs tallet 0B i acc.
  033C  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0B
  033F  A985    LDA  Indlæs tallet 85 (fin_indst. 2 sec.)
  0341  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0B

  0344  A90C    LDA  Indlæs tallet 0C i acc.
  0346  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0C
  0349  A91E    LDA  Indlæs tallet 1E (grov_indst. 2 sec.)
  034B  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0C

envelope_form:
  034E  A90D    LDA  Indlæs tallet 0D i acc.
  0350  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0D
  0353  A909    LDA  Indlæs tallet 09 (tonen dør efter 1 periode)
  0355  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0C
  0358  4C0002  JMP  Spring til "taster:"
```

Programdelen "taster:" fra side 62 skal stadig være bevaret. Start nu igen programmet fra 0300 og spil atter på de forskellige kontroltaster.

For at du bedre kan forstå den kommende forklaring på envelope-generatoren, skal du lige prøve at ændre et par værdier i "envelope_tid:". Det er nemlig her det bestemmes, hvor længe tonen skal være om at dø hen - periodetiden.

Værdierne skal ind af to omgange: både en fin-indstilling (register 0B) og en grov-indstilling (register 0C). Ved at indlægge 98 som grov_indst (adresse 034A) og 96 som fin_indst (adresse 0340), får du en periodetid på 10 sec. Sammenlign igen med register-oversigten på side 113.

Når du har gjort det, kan du atter afprøve "klaveret". Herefter prøver du at gå den anden vej: læg 03 ind i adresse 034A (grov_indst) og 0D ind i adresse 0340 (fin_indst). Nu er periodetiden 0,2 sec. Prøv atter at spille.

BEMÆRK: For at volumen kan påvirkes af envelop-generatoren, skal register 08 have bit 4 gjort høj. det gjorde vi ved hjælp af tallet 1F i denne linie:

```
030F  A91F    LDA  Indlæs tallet 1F (envelope kontrollerer vol)
```

Her er et skema, der viser sammenhængen mellem grov_indst, fin_indst og periodetiden:

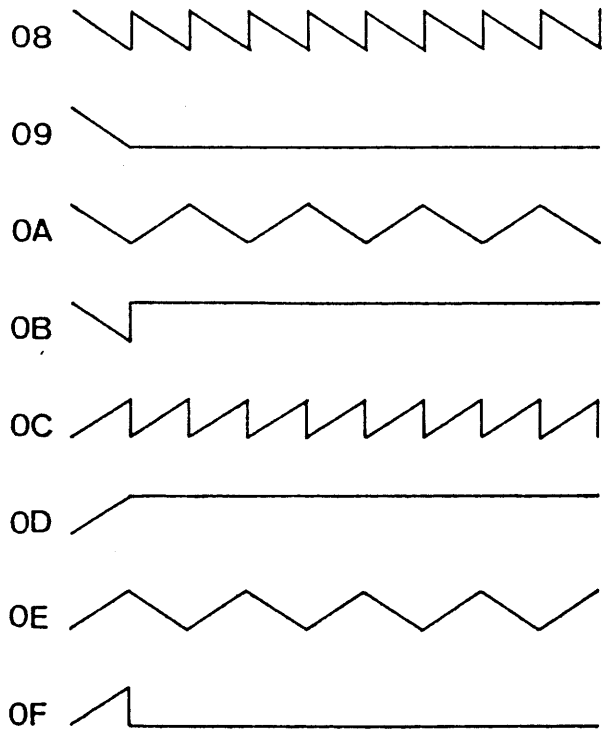
Periodetid	Grov_indst adr. 034A	Fin_indst adr. 0340
10 s	98	96
5 s	4C	4B
2 s	1E	85
1 s	0F	42
0,5 s	07	A1
0,2 s	03	0D
0,1 s	01	87
0,05 s	00	C3
0,02 s	00	4E
0,01 s	00	27

Du har nu set, at register 0B og 0C bestemmer periodetiden for

envelope-generatoren. I register 0D har man mulighed for at bestemme den kurveform, som envelope-generatoren skal påvirke musikkredsens volumen med. Du har i dette register indlagt værdien 09 ("envelope_form"):

```
0353  A909    LDA  Indlæs tallet 09 (tonen dør efter 1 periode)
```

Der er andre muligheder for kurveformer - her er en oversigt:



OPGAVE 29: Eksperimenter med både periodetid og kurveform ud fra de viste tabeller.

STØJ

Musikkredsen kan også afgive støj. Det sker ved hjælp af register 06 - se igen oversigten på side 113. Den letteste måde at afprøve effekten på, er ved at fortsætte med programmet fra side 64 og her ændre den øverste linie af "fin_tone:"

```
fin_tone:
  0330  A900     LDA  Indlæs tallet 00 i acc.
```

så den kommer til at se således ud:

```
støj:
  0330  A906     LDA  Indlæs tallet 06 i acc.
```

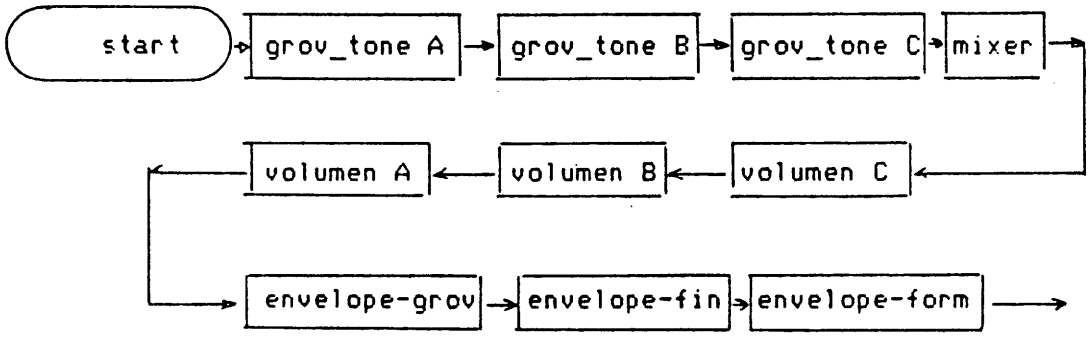
Endelig skal der ske en ændring ved "mixer:", så det er støjkanalen, der når frem. Det sker ved at gøre bit 3 lav og bit 0-1-2 høje med tallet 07. Man gør altså en kanal aktiv ved at gøre den lav:

```
  0305  A907     LDA  Indlæs tallet 07 (vælg støj ved kanal A)
```

OPGAVE 30: Start programmet fra adresse 0300 og prøv at ændre på "envelope_tid:", "envelope_form" og evt. også på indholdet i adresse 0000-000F. Værdierne her skal teoretisk ligge mellem 03 og 15.

ET EL-ORGEL

Vi skal nu prøve at bruge alle 3 kanaler på en gang og samtidig "forme" lyden ved hjælp af "envelope-generatoren". Hvis vi skulle lave programmet efter det hidtidige princip, ville det komme til at bestå af ti blokke, der var næsten ens:



- og desuden tre afdelinger til fin_tone, samt en programdel til taster.
Vi vælger derfor at lave programmet om, så de ti første ting kan klares af den samme programdel, der så køres igennem flere gange

med forskellige variabler. Denne programdel vil vil kalde "initiering:" (= nulstilling, klargøring).

Det sker ved, at programmet henter de værdier, som skal læses ind i de forskellige registre, fra adresse 0050 til 005D. Endelig er det værdien af X, der styrer, hvilket register, der skal skrives værdier i. X er 0 ved første gennemløb, men vokser efter hvert gennemløb med 1, for at der kan ske henvendelse til et nyt register.

I virkeligheden snyder vi lidt, idet der også skrives i "fin_tone 1-2-3:". Det gør imidlertid ikke noget, idet de indlagte værdier overskrives senere. Når alle registre er gjort klar, er X vokset til værdien 0D, og initieringen er færdig.

Bemærk at både i "initiering:" og "fin_tone 1-2-3" bruges indeksadressering: der hentes data fra den nævnte adresse + værdien af X.

NB: I EPROM-en findes der et tilsvarende program ved adresse FD00. De forskellige data, som er vist på næste side skal dog indtastes, før programmet kan bruges.

```
initiering:
  0300  A200    LDX  Indlæs tallet 00 i x-reg. = nulstilling
  0302  8E0C80  STX  Skriv x-reg på adr. 800C - find register
  0305  B550    LDA  Indlæs tal fra adresse 0050+X
  0307  8D0E80  STA  Skriv acc. på adr. 800E - data til register
  030A  E00D    CPX  Sammenlign x-reg med 0D (alle registre??)
  030C  F004    BEQ  Hvis ja, så hop til 0312
  030E  E8      INX  Forøg x-reg med 1, før nyt gennemløb
  030F  4C0203  JMP  Spring til 0302 og lav et nyt gennemløb
  0312  4C0002  JMP  Initering færdig, spring til "taster:"

Spring i adresse!!!

fin_tone1:
  0340  A900    LDA  Indlæs tallet 00 i acc.
  0342  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 0
  0345  B510    LDA  Indlæs tal fra adresse 0010+X
  0347  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 0

fin_tone2:
  034A  A902    LDA  Indlæs tallet 02 i acc.
  034C  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 2
  034F  B520    LDA  Indlæs tal fra adresse 0020+X
  0351  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 2
```

-fortsættes-

```
fin_tone3:
  0354  A904    LDA  Indlæs tallet 04 i acc.
  0356  8D0C80  STA  Skriv acc. på adr. 800C - find reg. 4
  0359  B530    LDA  Indlæs tal fra adresse 0030+X
  035B  8D0E80  STA  Skriv acc. på adr. 800E - data til reg. 4
  035E  A20B    LDX  Indlæs tallet 0B i x-reg. (envelope!!)
  0360  4C2003  JMP  Spring til initiering adr. 0302

taster:
  0200  AD0580  LDA  Nulstil kontroltastaturet
  0203  AE0480  LDA  Indlæs kontroltastaturet i x-reg.
  0206  E000    CPX  Sammenlign x-reg. med 00
  0208  F0F9    BEQ  Gå tilbage til 0203, hvis de er ens
  020A  4C4003  JMP  Gå til "fin_tone1:"

Tone-data til "fin_tone1":
0010-001F:    00-EF-E2-D5-C9-BD-B3-A9-9F-97-8E-86-7F-78-71-6A

Tone-data til "fin_tone2":
0020-002F:    00-78-71-6A-64-5F-5A-54-50-4B-47-43-3F-3C-38-35

Tone-data til "fin_tone3":
0030-003F:    00-3C-38-35-32-2F-2C-2A-28-26-24-22-20-1E-1C-1B
```

Dataene er lavet således, at "fin_tone2:" hele tiden ligger en oktav over "fin_tone1:", mens "fin_tone3:" ligger 2 oktaver over "fin_tone1:". Tonerne vil være placeret således på tasterne:

h	c	cis	d
g	gis	a	ais
dis	e	f	fis
	c	cis	d

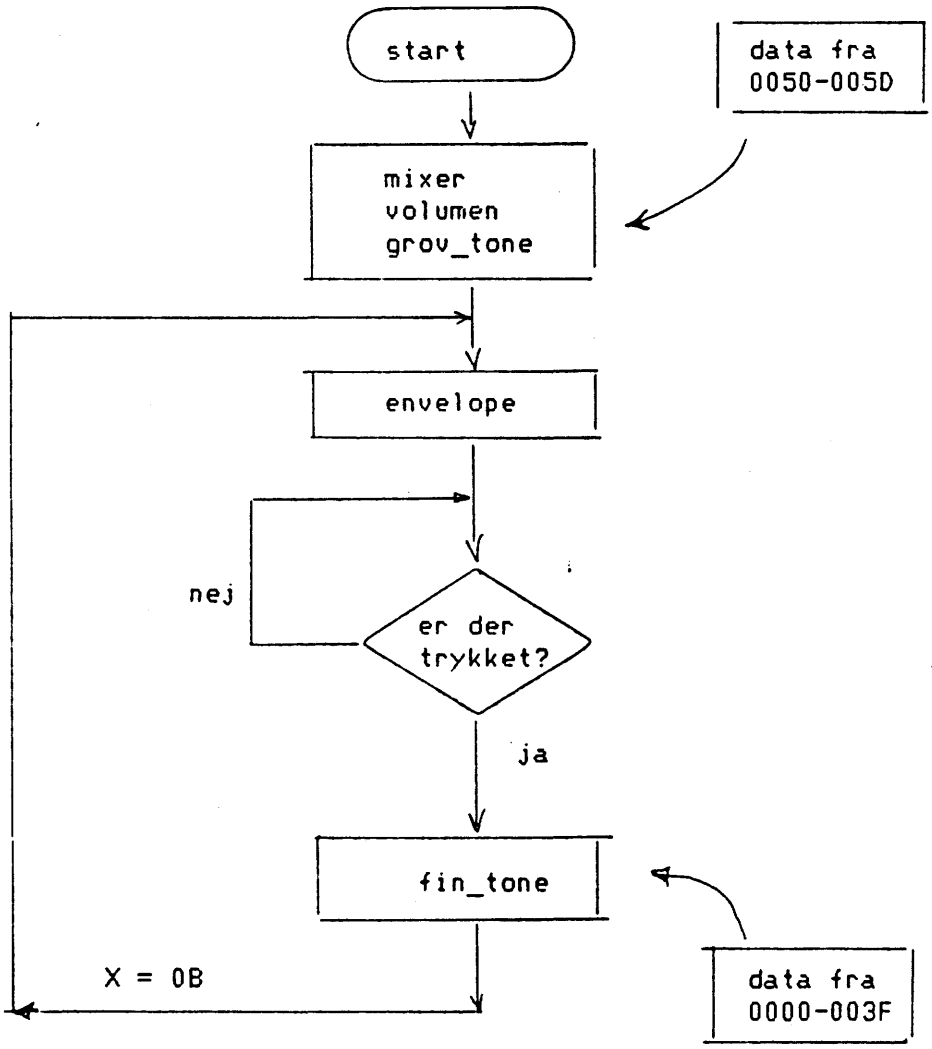
```
Data til "initiering":
  0050  00    reg. 00 fin_tone  kanal A: 00 - dummy
  0051  00    reg. 01 grov_tone kanal A: 00
  0052  00    reg. 02 fin_tone  kanal B: 00 - dummy
  0053  00    reg. 03 grov_tone kanal B: 00
  0054  00    reg. 04 fin_tone  kanal C: 00 - dummy
  0055  00    reg. 05 grov_tone kanal C: 00
  0056  15    reg. 06 støj             : 15 - ca. 3000Hz
  0057  F8    reg. 07 mixer            : toner ved A-B-C
```

-fortsættes-

```
0058  1F    reg. 08 volumen kanal A  : envelope
0059  1F    reg. 09 volumen kanal B  : envelope
005A  1F    reg. 0A volumen kanal C  : envelope
005B  A1    reg. 0B envelope_tid fin : 0,5 sec.
005C  07    reg. 0C envelope_tid grov: 0,5 sec.
005D  09    reg. 0D envelope_form    : dør ud
```

"Dummy" betyder, at der blot indlæses en værdi, for at der skal være en. Dens indhold betyder ingenting, idet den senere overskrives.

Bemærk at der efter "fin_tone3:" springes til "initiering:" samtidig med, at X har fået værdien 0B. Det skyldes, at "envelope" skal initieres efter, at toneregistrene har fået indlæst værdier, ellers vil envelopen ikke påvirke tonerne. "Initiering:" køres altså igennem igen fra X = 0B til X = 0D. Her er i øvrigt et blokdiagram over programmet - studer dette.



OPGAVE 31: Du har nu et program, hvor det er let at indlægge nye data, så du kan se deres virkning i de forskellige registre.
Find således ud af, hvordan du ændrer envelope_tid og envelope_form. Brug tal og kurveformer fra side 113.

OPGAVE 32: Lav om på data til mixeren, så du kun bruger støjkanalen. Kombiner dette med envelope-generatoren, så du kan lave "eksplosioner", "nytårsskyderi" osv.

OPGAVE 33: Find ud af, hvordan du i register 08-09-0A kan slå "envelope" fra og i stedet for få en fast volumen, hvis størrelse, du selv bestemmer.

INTERRUPT

Interrupt betyder afbrydelse. Her betyder ordet, at CPU-en kan afbrydes i et arbejde, den er i færd med, når der dukker noget op, som er vigtigere. På den måde kan CPU-en være i færd med flere ting ad gangen.

Hos 6502-processoren er der to slags interrupt: NMI (non-maskable interrupt = betingelsesløs interrupt) og IRQ (interrupt request = forespørgsel om interrupt). Vi starter med at undersøge den første - indtast derfor dette lille interrupt-program:

```
00B0  8500    STA  Gem acc. på adr. 0000
00B2  A501    LDA  Indlæs tal fra adresse 0001
00B4  8D0080  STA  Skriv acc. på display 0
00B7  8D0680  STA  Skriv til adr. 8006 - beep
00BA  E601    INC  Forøg adr. 0001 med 1
00BC  2020FA  JSR  Spring til pause
00BF  A500    LDA  Indlæs tal fra adresse 0000 igen
00C1  40      RTI  Returner fra interrupt
```

Programmet indleder med at gemme indholdet af accumulatoren (i adresse 0000). Herefter hentes et tal fra adresse 0000. Dette tal skrives ud på display 0, der laves et beep, og tallet i adresse 0001 forøges med 1. Efter en pause indlæses den gamle værdi igen i accumulatoren, og der returneres fra interrupt.

Afprøv nu interruptprogrammet på denne måde: sæt en ledning på ved et 0V-spyd og berør "NMI"-spyddet med en anden ende. Nu skal interrupt-programmet få display 0 til at vise et tal, og der skal samtidig komme et beep.

Dette vil sandsynligvis ske flere gange efter hinanden. Det skyldes, at man ikke kan røre ved "NMI"-spyddet, uden at der kommer flere impulser. Disse impulser vil blive husket, og der vil komme et nyt interrupt, når det gamle er færdigt.

Find nu adresse FFFB og adresse FFFA. De ligger jo i EPROM-en og hører med til styresystemet. Indholdet her skal se således ud:

```
FFFB  00
FFFA  B0
```

Når "NMI" modtager en impuls, vil CPU-en søge til netop disse to

adresser (sådan er den fremstillet). Her får den så at vide, hvor den kan hente et interrupt-program. Derfor startede vi vort interrupt program ved adresse 00B0. Man kunne også på denne adresse lægge en JMP, så CPU-en sprang et helt andet sted hen.

Du skal nu have et program til at køre, men det skal hele tiden afbrydes af interrupt-programmet (skal bevares). Programmet kan f.eks. se således ud:

```
0200  AD0380  LDA  Hent indhold fra tastaturet og læg det i acc.
0203  8D0080  STA  Skriv acc. på display 0
0206  4C0002  JMP  Spring til 0200 og kør programmet igen
```

I stedet for at du selv laver interrupt, skal du få en såkaldt AMV til at give impulser til "NMI". Det gør du ved at forbinde "AMV-UD" på dekoderen med "NMI". AMV-en vil nu give en impuls ~~for ca. hver 2. sekund~~. Her skal der så komme et interrupt, der både kan høres og ses (tallet fra adresse 00D1 forøges med 1). Hvis du vil, kan du evt finde denne adresse og indlæse tallet 00, før du går igang.

Find så adresse 0200 og start her det program, der hele tiden henter tal fra tastaturet og skriver dem på display 0. Nu skal dette program altså køre og hele tiden blive afbrudt af interrupt-programmet.

OPGAVE 34: Lav interrupt-programmet om, så det tæller decimalt. Fjern ledningen fra NMI, mens du laver programmet.

OPGAVE 35: Lav nogle andre interrupt-programmer, som regelmæssigt griber ind i et "almindeligt" program. Det kunne f.eks. være et program, der får stepmotoren til at køre lidt eller får lyskrydset til at skifte.

Udblik

Du har nu set, at et signal på "NMI"-spyddet kan afbryde CPU-en i et forløb, så den begynder at køre et særligt interrupt-program. I en rigtig datamaskine bruges interrupt meget hyppigt. Når man taster noget på tastaturet, skabes et interrupt-signal og proccesoren går ud for at betjene tastaturet. På tilsvarende måde

betjenes skærm, porte og den slags ydre enheder af interrupt. Når der kan være flere enheder, der har skabt interruptet, skal CPU-en naturligvis starte med at finde ud af, hvem der er "synderen".

Dette kan ske på flere måder. En af dem går ud på, at en særlig kontrolbit, der står i forbindelse med den ydre enhed, bliver gjort høj. CPU-en kan herefter afsøge enhederne en ad gangen for at se, om den pågældende bit er høj. Hvis der er flere interrups på en gang, vil de blive betjent, efterhånden som der bliver tid til det.

Et andet problem, som du måske har spekuleret over, er hvordan processoren bærer sig ad med at finde rigtigt tilbage i det oprindelige program igen.

I processoren findes en register, der hele tiden indeholder adressen på den del af programmet, der er ved at blive kørt - programtælleren. Når der kommer et interrupt (eller der springes til en sub-rutine), bliver programtællerens indhold gemt i et særligt afsnit af RAM-hukommelsen, som kaldes stacken. Til at holde rede på stacken findes en stack-pointer, der til at begynde med peger på adresse 01EF.

Når den første byte er gemt, tælles stack-pointeren nedad, så den peger på 01EE. Her vil så den næste byte blive lagt. Antag at programmet er nået til adresse 0204, da interruptet (eller springet til subrutine) kom. Stacken vil da se således ud:

```
01EF  02
01EE  04
01ED  ??
```

Spørgsmålstegnene skal vise, at stack-pointeren nu peger på adresse 01ED som det sted, hvor den næste oplysning skal gemmes, hvis der atter kommer et interrupt eller et spring til subrutine. Det er også muligt at få indholdet af accumulator og statusregister gemt i stacken (PHA, PLP).

Efter sådanne forskellige operationer kan stacken f.eks. se således ud, idet stack-pointeren peger på 01EB:

```
01EF  02
01EE  04
01ED  AD
01EC  82
01EB  ??
```

Når stacken tømmes, sker det på den måde, at det der kom sidst ind, kommer først ud (last in first out = LIFO). Her kommer altså "82" ud først, så "AD" så "04" og til sidst "02".

Ud over programtællerens indhold gemmes ved interrupt også automatisk statusregistret, som det ser ud i det aktuelle øjeblik.

INTERRUPT-REQUEST

Processoren indeholder mulighed for en anden type interrupt (interrupt-request = forespørgsel om interrupt). Det går ud på, at hvis IRQ-spyddet på CPU-modulet modtager et signal, kan der påbegyndes et interrupt-program, hvis IRQ-flaget tillader det. Det er et flag, der hører med i statusregistret (se side 112). Så længe flaget er sat (=1), bliver interruptet udsat. Når maskinen startes op, bliver IRQ-flaget sat automatisk, så det skal slettes, hvis der skal kunne laves interrupt. Det sker ved hjælp af 58 = CLI (clear interrupt disable bit).

Når interrupt-signalet kommer, henvender CPU-en sig til adresse FFFE og FFFF, hvor den henter adressen på et interrupt-program. I vores tilfælde står der ved disse adresser:

```
FFFF  00
FFFE  D0
```

Interrupt-programmet skal derfor ligge ved 00D0, eller der skal her ligge en jump (4C) som henviser til en ny adresse. Du kan bruge det første interrupt-program igen:

```
00D0  8500    STA  Gem acc. på adr. 0000
00D2  A501    LDA  Indlæs tal fra adresse 0001
00D4  8D0080  STA  Skriv acc. på display 0
00D7  8D0680  STA  Skriv til adr. 8006 - beep
00DA  E601    INC  Forøg adr. 0001 med 1
00DC  2020FA  JSR  Spring til pause
00DF  A500    LDA  Indlæs tal fra adresse 0000 igen
00E1  40      RTI  Returner fra interrupt
```

Forbind nu AMV-UD til IRQ-spyddet. Nu vil der på grund af interrupt-flaget ikke komme interrupt. Lav derfor et program, der indledes med CLI (58), så der tillades interrupt. Du kan f.eks. bruge dette program igen:

```
0200  58      CLI  Fjern interrupt-flag
```

```
  0201  AD0380  LDA  Hent indhold fra tastaturet og læg det i acc.
  0204  8D0080  STA  Skriv acc. på display 0
  0207  4C0102  JMP  Spring til 0201 og kør programmet igen
```

Når programmet kører, skal det med jævne mellemrum blive afbrudt af interrupt-programmet.

OPGAVE 36 Det modsatte af CLI er SEI (set interrupt-disable status) = sæt interrupt-flaget. Find op-koden for denne instruktion og lav programmet om, så der nu ikke længere kan laves interrupt.

Du har nu set, at programmøren kan bestemme, om IRQ kan tillades. Han kan derfor lave nogle programafsnit, hvor interrupt tillades og andre, hvor interrupt ikke tillades.

STYRESYSTEMET

I enhver datamaskine skal der være et styresystem, som får elektronikken til at opføre sig intelligent. Hvis du fjerner EPROM-en fra memory 1, vil MIKRO-DASK således ikke virke. I EPROM-en ligger der nemlig et styresystem, der sørger for at aflæse tastaturet (både det almindelige og kontrol-tastaturet). Endelig sørger styresystemet for at skrive adresse og data ud på displayene.

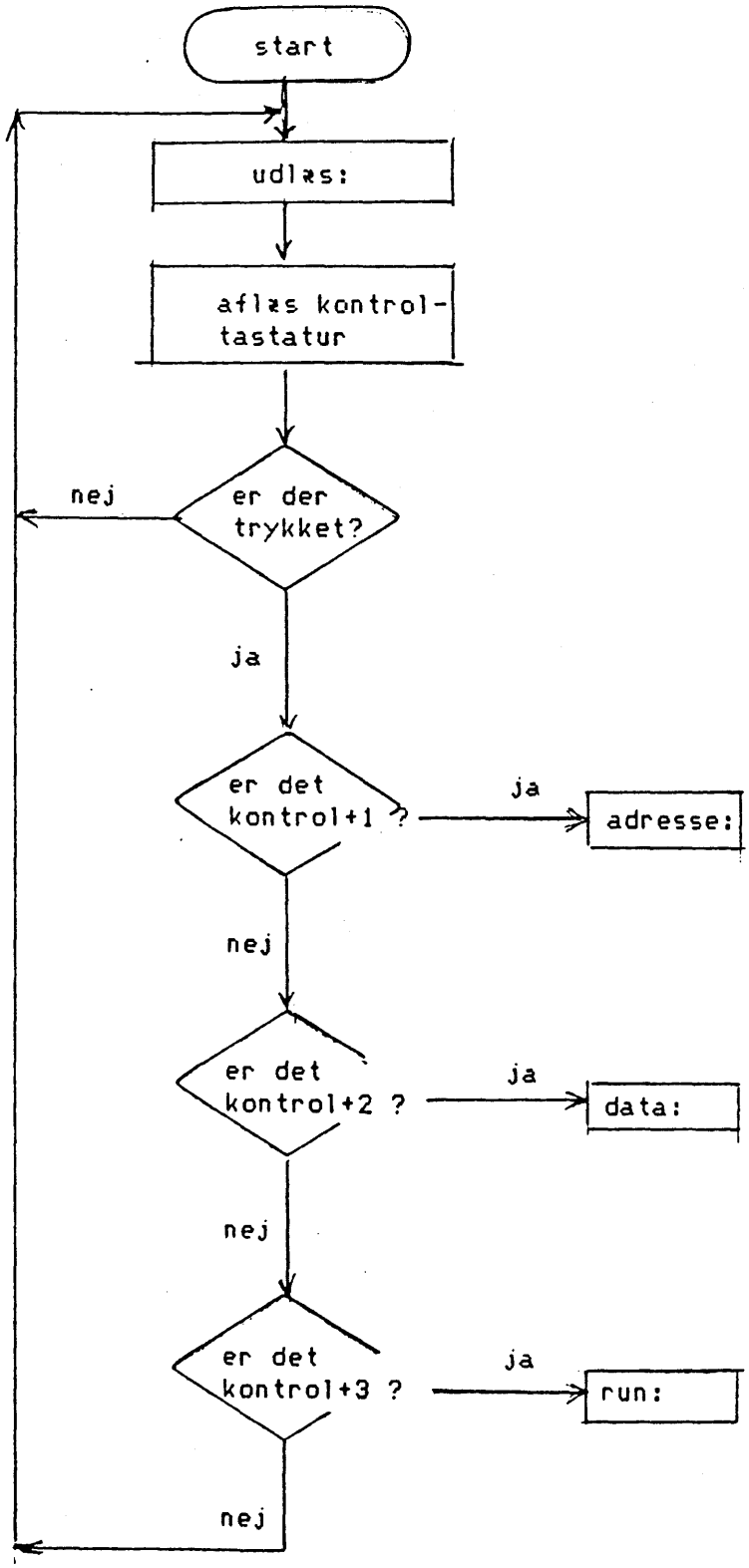
For at du kan en fornemmelse af, hvordan styresystemet er opbygget, skal du afprøve dette simple styreprogram:

```
aflæs_kontrol:
  0200  8D0580  STA  Skriv til 8005 - nulstil kontroltastatur
  0203  207002  JSR  Spring til udlæs:
  0206  AD0480  LDA  Indlæs tal fra adr. 8004 - aflæs kontroltast.
  0209  C900    CMP  Sammenlign acc. med tallet 00
  020B  F0F6    BEQ  Gå tilbage til aflæs_kontrol hvis de er ens
                     (så er der ikke trykket)
  020D  4C1002  JMP  Spring til kontrol_1:


kontrol_1:
  0210  C901    CMP  Sammenlign acc. med tallet 01
  0212  D006    BNE  Gå til kontrol_2: hvis de er forskellige
  0214  204002  JSR  Spring til adresse:
  0217  4C0002  JMP  Spring til aflæs_kontrol:
```

```
kontrol_2:
  0220  C902    CMP  Sammenlign acc. med tallet 02
  0222  D006    BNE  Gå til kontrol_3: hvis de er forskellige
  0224  205002  JSR  Spring til data:
  0227  4C0002  JMP  Spring til aflæs_kontrol:

kontrol_3:
  0230  C903    CMP  Sammenlign acc. med tallet 03
  0232  D003    BNE  Gå 3 frem hvis de er forskellige
  0234  206002  JSR  Spring til run:
  0237  4C0002  JMP  Spring til aflæs_kontrol:

adresse:
  0240  A590    LDA  Indlæs tal fra adr. 0090    - flyt tidligere
  0242  8591    STA  Skriv tallet på adr. 0091   - adr. byte
  0244  AD0380  LDA  Indlæs ny adr. byte fra tastaturet
  0247  8590    STA  Skriv adr. byte i adr. 0090
  0249  60      RTS  Retur til hovedprogram

data:
  0250  AD0380  STA  Indlæs tal fra tastaturet
  0253  A000    LDY  Indlæs tallet 00 i y-reg.
  0255  9190    STA  Skriv acc. i adressen der er i 0090 og 0091
  0257  60      RTS  Retur til hovedprogram

run:
  0260  6C9000  JMP  Spring til adresse der er i 0090 og 0091

udlæs:
  0270  A000    LDY  Indlæs tallet 00 i y-reg.
  0272  B190    LDA  Indlæs tal fra adressen der er i 0090 og 0091
  0274  8D0080  STA  Skriv acc. på display 0  = data
  0277  A590    LDA  Indlæs tal fra adresse 0090
  0279  8D0180  STA  Skriv acc. på display 1  = adresse L
  027C  A591    LDA  Indlæs tal fra adresse 0091
  027E  8D0280  STA  Skriv acc. på display 2 = adresse H
  0281  60      RTS  Retur til hovedprogram
```

Start programmet fra adresse 0200. Bemærk at det nu kun er kontrol 1-2-3, der virker. Når du skal blade frem i adresserne, skal du hver gang indtaste den komplette adresse ved hjælp af "kontrol"+1.

Diagrammet på næste side forklarer, hvorledes styreprogrammet er opbygget:


start
udlæs:
aflæs kontrol-
tastatur
er der
trykket?
nej
ja
er det
kontrol+1 ?
ja
adresse:
nej
er det
kontrol+2 ?
ja
data:
nej
er det
kontrol+3 ?
ja
run:
nej

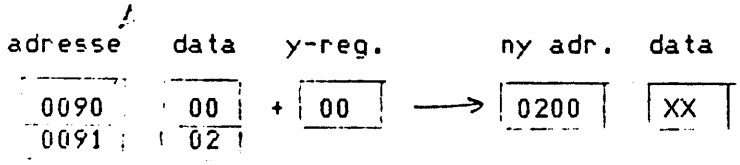
Programmet bruger adresse 0090 til at gemme den lave adresse-byte, mens 0091 bruges til den høje adresse-byte.

I "adresse:" sker der det, at det tal, der står i adresse 0090 flyttes over i 0091. Den tidligere lav-byte gøres altså til høj-byte. Som ny lav-byte bruges det tal, der står i tastaturet.

I "data:" bruges en særlig form for adressering: (IND),Y - indirekte y adressering. CPU-en henvender sig til adresse 0090 og finder her en lav-byte på en ny adresse. Høj-byte findes i adresse 0091. Den fundne adresse lægges sammen med værdien af y-reg. (her 0). I den adresse, der herved fremkommer skrives en databyte, der hentes fra tastaturet.

Eksemplet her viser princippet:

op-kode og operand: 9190



Data-byten XX skrives altså her ved adresse 0200. Data skrives med andre ord i den adresse, der hele tiden gemmes i adresse 0090 og 0091.

I "run:" springes der til den adresse, der er indeholdt i 0090 og 0091. Hvis man derfor sørger for, at 0090 og 0091 indeholder adressen for bruger-programmets begyndelse, vil springet altså få programmet til at køre.

I "udlæs:" hentes data fra adressen, der er gemt i 0090 og 0091 og vises på display 0. Endelig skrives værdien af 0090 på display 1 og værdien af 0091 på display 2.

Udblik

Styresystemet hos MIKRO-DASK indeholder flere muligheder, end dem, der er forklaret her. Der kan, som du har set, bruges flere kontroltaster, og der er derfor flere underprogrammer. Hvis du har lyst, kan du kan selv studere styresystemet - det er vist bag i bogen (side 86-90).

I en rigtig datamaskine findes også et styresystem. Her har systemet langt mere komplicerede opgaver at tage sig af: læse eller skrive på disketten, skrive tegn på printeren, filbehandling m.m.

Som eksempler på rigtige styresystemer kan nævnes: CP/M, CCP/M-86, MS-DOS og UNIX.

En rigtig datamaskine kan som regel også programmeres i maskinsprog, som du har prøvet. Her bruger man dog i almindelighed et assembler-sprog. Det går i store træk ud på, at man ikke arbejder med selve op-koderne, men derimod med betegnelserne for dem (mnemonics) - se eksemplet her:

```
Mnemonic
  LDA       XX          ; Tal til acc.
  JSR       display     ; Til display
  LDA       XX          ; Nyt tal til acc.
  JSR       display     ; Til display
```

I assembler-sprog bruges der labels i stedet for faste adresser, og der kan indføjes kommentarer (efter semikolon). Når assemblerprogrammet er færdigt, kan det oversættes til maskinkode af et assembler-program.

En anden type sprog er de såkaldte højniveau-sprog som: Basic, Comal80 og Pascal. Her kan programmeringen ske i et sprog, der minder lidt om det menneskelige. Bagefter sker der så en oversættelse til maskinsprog, for det er jo det eneste, som CPU-en forstår.

I Comal80 og Basic sker denne oversættelse linie for linie, mens programmet kører. I Pascal skal hele programmet oversættes til maskinkode, før det kan køres. Derfor arbejder et Pascal-program væsentlig hurtigere end et Comal80-program. Til gengæld har Comal80 langt bedre fejlmeddelelser, idet hver linie kontrolleres under selve programmeringen.

Højniveau-sprog udnytter styresystemet til at varetage betejeningen af maskinenes forskellige enheder: tastatur, skærm, diskettestation m.m.

EPROM-BRÆNDING

Du har mange gange oplevet, at et program, der ligger i RAM-lageret forsvinder, når du slukker for strømmen. Anderledes ser det ud med programmer, der ligger i EPROM-en. Det er således muligt at overføre et program fra RAM-kredsen til en EPROM. Det sker ved, at

programmets adresser blades igennem, idet der hver gang sker en overførsel af de tilsvarende data.

Når både en adresse med tilhørende data er overført, skal EPROM-en kortvarigt (ca. 50msek.) have en spænding på 25V ind på ben 21 (Vpp). Herefter blades frem til den næste adresse med tilhørende data, og der gives igen en spændingspuls på 25V.

Der er imidlertid mange problemer forbundet med denne brænding, hvilket sammen med den høje pris for EPROM-er gør det helt nødvendigt at man arbejder med den yderste omhu og omtanke. I EPROM-en der sidder i memory 1 ligger der ved adresse FB00 et brændeprogram, der kan bruges sammen med brænde-modulet.

Før programmet køres, skal der indlægges forskellige variabler i disse adresser:

Variabler:

0000	"fra" L-byte
0001	"fra" H-byte
0002	"til" L-byte
0003	"til" H-byte
0004	antal bytes (01-FF)

Endelig skal brændemodulet forbindes således til adressedekoderen:

8006	beep
8007	A1E
8008	A2E
8009	DE
800A	BE

Desuden skal modulet have +5V, 0V og det skal forbindes med databussen ved CPU-modulet.

Brændeprogrammet ser ud som vist her:

```
initiering:
  FB00  A970    LDA  Indlæs tallet 70 i acc.
  FB02  85A1    STA  Skriv tallet 70 i adr. 00A1 (kort pause)
  FB04  85A3    STA  Skriv tallet 70 i adr. 00A3 (mellem. pause)
  FB06  A000    LDY  Indlæs tallet 00 i y-reg. = nulstilling

adresse_høj:
  FB08  A503    LDA  Indlæs tal fra adr. 0003 ("til"  H-byte)
  FB0A  8D0880  STA  Skriv acc. på adr. 8008 (A2E = 0003)
  FB0D  8D0280  STA  Skriv acc. på display 2
```

```
adresse_lav:
  FB10  A502    LDA  Indlæs tal fra adr. 0002 ("til" L-byte)
  FB12  8D0780  STA  Skriv acc. på adr. 8007 (A1E = 0002)
  FB15  8D0180  STA  Skriv acc. på display 1

data:
  FB18  B100    LDA  Indlæs tal hentet i "fra"- adresse
  FB1A  8D0980  STA  Skriv acc. på adr. 8009 (DE)
  FB1D  2000FA  JSR  Spring til kort_pause
  FB20  8D0A80  STA  Skriv til adr. 800A = brænd data
  FB23  8D0080  STA  Skriv acc. på display 0
  FB26  2020FA  JSR  Spring til mellem_pause

hvad_mu:
  FB29  C404    CPY  Sammenlign y-reg. med adr. 0004 (antal bytes)
  FB2B  F006    BEQ  Gå til ud: hvis ens (= færdig)
  FB2D  E602    INC  Forøg adr. 0002 med 1 (= næste "til" celle)
  FB2F  C8      INY  Forøg Y med 1
  FB30  38      SEC  Sæt carry-flag
  FB31  B0DD    BCS  Spring til adresse_lav: hvis der er mente

ud:
  FB33  8D0680  STA  Skriv til adr. 8006 = beep
  FB36  B0FE    BCS  Løb på stedet
```

Du kan selv prøve at gennemgå programmet. I "data:" bruges (IND),Y adressering (B100). Det virker således, at der i adresse 0000 og 0001 er gemt den adresse, hvorfra den første byte skal hentes - først skal værdien af Y dog lægges til. Første gang er Y lig med 0, men da Y vokser med 1 efter hvert gennemløb, blades adresserne herved igennem.

ADRESSEBEREGNING

Et af problemerne ved brænding af en EPROM er at få programmet til at indeholde de rigtige adresser og at få det til at ligge det rigtige sted i EPROM-en. Tænk f.eks. på, at instruktioner med JMP (4C) kun vil virke rigtigt, når programmet ligger et bestemt sted i adresserummet. Vi kan skelne mellem 3 adresser: afprøvnings-adresse, brugsadresse og brændeadresse.

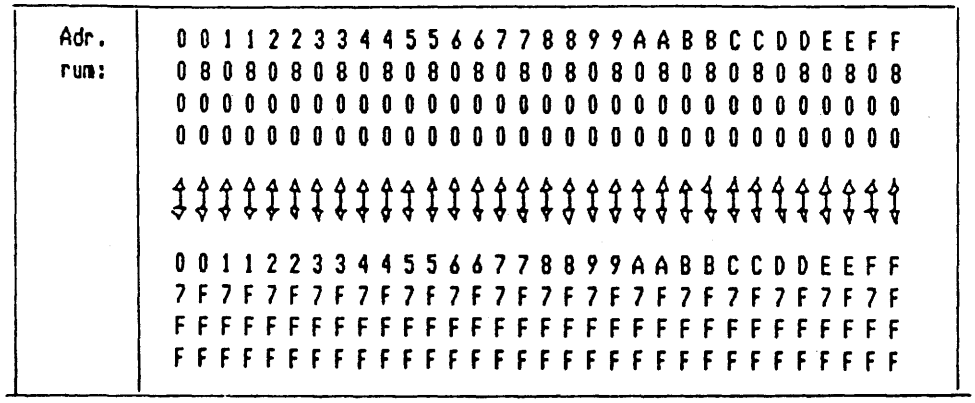
AFPRØVNINGSADRESSE: adresse som programmet indeholder, mens det ligger til afprøvning i RAM-en

BRUGSADRESSE: adresse som passer til det sted i hukommelsesrummet, hvor programmet skal ligge. Afhænger bl.a. af indstillingen af

adresserumsvælgeren på memory-modulet.

BRÆNDEADRESSE: den adresse, som programmet skal lægges ved under brændingen.

Som du tidligere har set, kan der laves disse 2K adresserum ud af det samlede 64K adresserum:



Hvis programmet skal ligge i starten af et af de 32 adresseområder, skal dets brændeadresse være 0000. Hvis det skal ligge 1 side højere oppe, skal dets brændeadresse være 0100.

Hvis brugsadressen f.eks. skal være DB00, skal EPROM-en jo placeres i det adresseområde, der går fra D800 til DFFF. Under brændingen vil D8 skulle svare til side 00, mens D9 skal svare til side 01, DA til side 02 og DB til side 03. Her er nogle eksempler på forskellige brugsadresser og de tilhørende brændeadresser. Du kan selv beregne nogle andre, når du har forstået princippet:

BRUGSADRESSE:			BRÆNDEADRESSE:
1000	D800	F800	0000
1100	D900	F900	0100
1200	DA00	FA00	0200
1300	DB00	FB00	0300
1400	DC00	FC00	0400
1500	DD00	FD00	0500
1600	DE00	FE00	0600
1700	DF00	FF00	0700

BRÆNDING:

1) Sørg for at programmet, der skal brændes er godt afprøvet. Sørg også for at have det skrevet ned på papir. Afmærk alle de steder, hvor der bruges JMP og noter, hvilke brugsadresser, der skal anvendes. Sørg for at ændre disse før brændingen.

2) Beregn brændeadressen og tæl op, hvor mange bytes, der skal overføres (start med at kalde den første byte 0). Hvis programmet begynder ved begyndelsen af en side, vil tallet simpelthen svare til adressen på programmets sidste byte (husk de allersidste!!).

   Ved et program, der begynder ved 0200 og ender som vist herunder, skal du altså sætte antallet til 2F:

   022D 4C0002

3) Indtast "fra" og "til" adresser samt antal bytes i adresse 0000-0004.

4) Find selve brændeprogrammet ved adresse FB00 og lad det køre. Kontroller om displayene hele tiden viser "til" adressen samt data-byten.

5) Mål evt. efter med et voltmeter om soklen i brænde-modulet indeholder den rigtige adresse og de rigtige data på programmets sidste byte.

6) Nu er indholdet i adresse 0020 blevet talt op med et tal, der svarer til antallet af brændinger. Ret dette tal.

7) Sæt EPROM-en i soklen på brænde-modulet.

8) Sørg for at lysdioden ved brænde-strømforsyningen IKKE LYSER (tryk på ringetrykket) og forbind så 25V-ledningen til det tilsvarende spyd på brænde-modulet.

9) Find igen brændeprogrammet ved FB00, tryk på det andet ringetryk ved strømforsyningen, så lysdioden lyser.

10) Tryk nu "run" og følg med i om displayene kører rigtigt. Sluk lysdioden på strømforsyningen, tag EPROM-en væk og afprøv den.

Vedrørende punkt 8: det er mange EPROM-er, der er gået i døden som følge af prel fra en ledning med 25V. Derfor må ledningen IKKE INDEHOLDE SPÆNDING, MENS DEN SÆTTES PÅ. I øvrigt indeholder strømforsyningen en strømbegrænser, der slår fra, hvis strømforbruget overstiger 50mA.

Hvis man fjerner +5V og 0V- ledningerne fra brænde-modulet, mens databussen stadig er monteret, vil datamaskinen ikke virke!!

DOKUMENTATION AF SOFTWARE

INDDELING AF EPROM:

```
F800 - F934  styresystem
F940 - F9BF  ur-program
FA00 - FA56  pauser (FA00 lille, FA20 mellem, FA40 lang pause)
FB00 - FB2F  brændeprogram
FC00 -       regneprogrammer
FD00 - FD62  musik- klaver
FE00 - FE20  beeper
FE20 - FE40  beep og løb
FE40 - FE60  tæl op på alle display
FFFA   L-byte for NMI-vektor           (her: B0)
FFFB   H-byte for NMI-vektor           (her: 00)
FFFC   L-byte for reset-vektor         (her: 00)
FFFD   H-byte for reset-vektor         (her: F8)
FFFE   L-byte for IRQ- og BRK- vektor  (her: D0)
FFFF   H-byte for IRQ- og BRK- vektor  (her: 00)
```

INDDELING AF RAM:

```
0000 - 009F  frit til zero page adressering
00A0 - 00AF  pauseparametre m.m.   00A1: kort,   00A3: mellem,
                                   00A5: lang pause
00B0 - 00CF  til NMI- program
00D0 - 00EF  til IRQ- program
00FA - 00FB  styresystem: flyt/slet byte
00FE - 00FF  styresystem: gemmer aktuel adresse
0100 - 01EF  stack
01F0 - 07FF  frit til absolut adressering
```

STYRESYSTEM

```
stack:
  F800  A2EF    LDX  Indlæs tallet EF i x-reg.
  F802  9A      TXS  Indstil stack-pointer på 01EF

aflæs_kontrol:
  F803  8D0580  STA  Skriv til 8005 - nulstil kontroltastatur
```

```
  F806  20BDF8  JSR  Spring til udlæs:
  F809  AD0480  LDA  Indlæs tal fra adr. 8004 - aflæs kontroltast.
  F80C  C900    CMP  Sammenlign acc. med tallet 00
  F80E  F0F6    BEQ  Gå tilbage til  aflæs_kontrol  hvis de er ens
                     (så er der ikke trykket)

kontrol_1:
  F810  C901    CMP  Sammenlign acc. med tallet 01
  F812  D006    BNE  Gå til kontrol_2: hvis de er forskellige
  F814  2098F8  JSR  Spring til adresse:
  F817  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_2:
  F81A  C902    CMP  Sammenlign acc. med tallet 02
  F81C  D006    BNE  Gå til kontrol_3: hvis de er forskellige
  F81E  20A2F8  JSR  Spring til data:
  F821  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_3:
  F824  C903    CMP  Sammenlign acc. med tallet 03
  F826  D006    BNE  Gå til kontrol_4: hvis de er forskellige
  F828  20AAF8  JSR  Spring til run:
  F82B  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_4:
  F82E  C904    CMP  Sammenlign acc. med tallet 04
  F830  D009    BNE  Gå til kontrol_5: hvis de er forskellige
  F832  20A2F8  JSR  Spring til data:
  F835  20ADF8  JSR  Spring til en_frem:
  F838  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_5:
  F83B  C905    CMP  Sammenlign acc. med tallet 05
  F83D  D009    BNE  Gå til kontrol_6: hvis de er forskellige
  F83F  20A2F8  JSR  Spring til data:
  F842  20B4F8  JSR  Spring til en_bak:
  F845  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_6:
  F848  C906    CMP  Sammenlign acc. med tallet 06
  F84A  D006    BNE  Gå til kontrol_7: hvis de er forskellige
  F84C  20ADF8  JSR  Spring til en_frem:
  F84F  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_7:
  F852  C907    CMP  Sammenlign acc. med tallet 07
  F854  D006    BNE  Gå til kontrol_8: hvis de er forskellige
  F856  20B4F8  JSR  Spring til en_bak:
  F859  4C03F8  JMP  Spring til aflæs_kontrol:
```

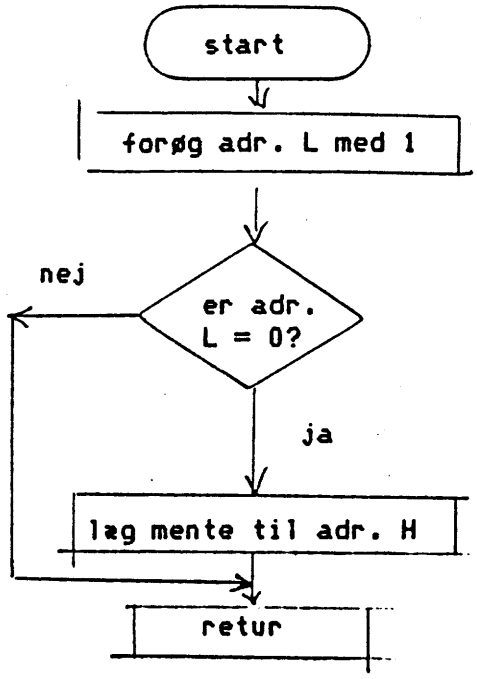
```
kontrol_8:
  F85C  C908    CMP  Sammenlign acc. med tallet 08
  F85E  D006    BNE  Gå til kontrol_9: hvis de er forskellige
  F860  20CFF8  JSR  Spring til side_op
  F863  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_9:
  F866  C909    CMP  Sammenlign acc. med tallet 09
  F868  D006    BNE  Gå til kontrol_A: hvis de er forskellige
  F86A  20D7F8  JSR  Spring til side_ned:
  F86D  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_A:
  F870  C90A    CMP  Sammenlign acc. med tallet 0A
  F872  D006    BNE  Gå til kontrol_B: hvis de er forskellige
  F874  20DFF8  JSR  Spring til 10_frem:
  F877  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_B:
  F87A  C90B    CMP  Sammenlign acc. med tallet 0B
  F87C  D006    BNE  Gå til kontrol_C: hvis de er forskellige
  F87E  20EDF8  JSR  Spring til 10_bak:
  F881  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_C:
  F884  C90C    CMP  Sammenlign acc. med tallet 0C
  F886  D006    BNE  Gå til kontrol_D: hvis de er forskellige
  F888  20FBF8  JSR  Spring til slet_byte:
  F88B  4C03F8  JMP  Spring til aflæs_kontrol:

kontrol_D:
  F88E  C90D    CMP  Sammenlign acc. med tallet 0D
  F890  D003    BNE  Gå til adr. F895 hvis de er forskellige
  F892  2017F9  JSR  Spring til indsæt_byte
  F895  4C03F8  JMP  Spring til aflæs_kontrol:

adresse:
  F898  A5FE    LDA  Indlæs tal fra adr. 00FE    - flyt tidligere
  F89A  85FF    STA  Skriv tallet på adr. 00FF   - adr. byte
  F89C  AD0380  LDA  Indlæs ny adr. byte fra tastaturet
  F89F  85FE    STA  Skriv adr. byte i adr. 00FE
  F8A1  60      RTS  Retur til hovedprogram

data:
  F8A2  AD0380  STA  Indlæs tal fra tastaturet
  F8A5  A000    LDY  Indlæs tallet 00 i y-reg.
  F8A7  91FE    STA  Skriv acc. i adressen der er i 00FE og 00FF
  F8A9  60      RTS  Retur til hovedprogram
```

```
run:
  F8AA  6CFE00  JMP  Spring til adresse der er i 00FE og 00FF

en_frem:
  F8AD  E6FE    INC  Forøg adr. 00FE med 1
  F8AF  D002    BNE  Spring til F8B3 hvis indh. ikke bliver 00
  F8B1  E6FF    INC  Forøg adr. 00FF med 1 = mente
  F8B3  60      RTS  Retur til hovedprogram

en_bak:
  F8B4  A5FE    LDA  Indlæs tal fra adr. FE
  F8B6  D002    BNE  Spring til F8BA hvis indh. ikke bliver 00
  F8B8  C6FF    DEC  Træk 1 fra adr. FF
  F8BA  C6FE    DEC  Træk 1 fra adr. FE
  F8BC  60      RTS  Retur til hovedprogram

udlæs:
  F8BD  A000    LDY  Indlæs tallet 00 i y-reg.
  F8BF  B1FE    LDA  Indlæs tal fra adressen der er i 00FE og 00FF
  F8C1  8D0080  STA  Skriv acc. på display 0  = data
  F8C4  A5FE    LDA  Indlæs tal fra adresse 00FE
  F8C6  8D0180  STA  Skriv acc. på display 1  = adresse L
  F8C9  A5FF    LDA  Indlæs tal fra adresse 00FF
  F8CB  8D0280  STA  Skriv acc. på display 2 = adresse H
  F8CE  60      RTS  Retur til hovedprogram

side_op:
  F8CF  20ADF8  JSR  Spring til en_frem:
  F8D2  A5FE    LDA  Indlæs tal fra adr. 00FE
  F8D4  D0F9    BNE. Spring til F8CF hvis acc. ikke er 00
  F8D6  60      RTS  Retur til hovedprogram

side_ned:
  F8D7  20B4F8  JSR  Spring til en_bak:
  F8DA  A5FE    LDA  Indlæs tal fra adr. 00FE
  F8DC  D0F9    BNE  Spring til F8D7 hvis acc. ikke er 00
  F8DE  60      RTS  Retur til hovedprogram

10_frem:
  F8DF  A200    LDX  Indlæs tallet 00 i x-reg.
  F8E1  20ADF8  JSR  Spring til en_frem:
  F8E4  E8      INX  Forøg X med 1
  F8E5  E010    CPX  Sammenling X med tallet 10
  F8E7  F003    BEQ  Spring til F8EC hvis ens
  F8E9  4CE1F8  JMP  Spring til F8E1 - nyt gennemløb
  F8EC  60      RTS  Retur til hovedprogram

10_bak:
  F8ED  A200    LDX  Indlæs tallet 00 i x-reg.
  F8EF  20B4F8  JSR  Spring til en_bak:
```

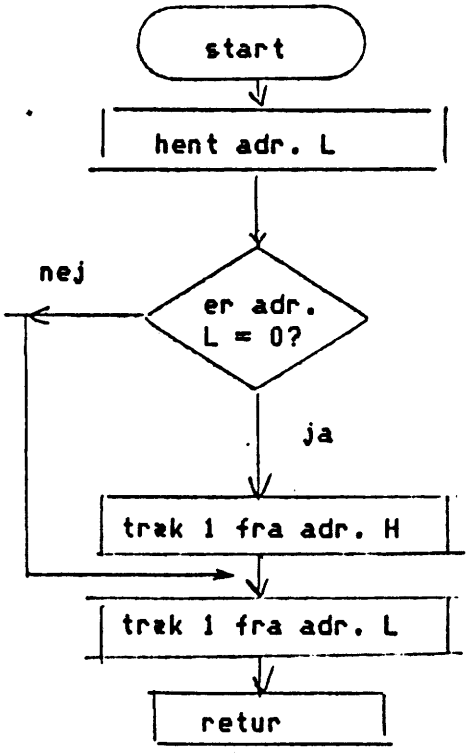
```
  F8F2  E8      INX  Forøg X med 1
  F8F3  E010    CPX  Sammenling X med tallet 10
  F8F5  F003    BEQ  Spring til F8EC hvis ens
  F8F7  4CEFF8  JMP  Spring til F8EF - nyt gennemløb
  F8FA  60      RTS  Retur til hovedprogram

slet_byte:
  F8FB  A5FE    LDA  Indlæs tal fra adr. 00FE - adresse L
  F8FD  85FA    STA  Skriv acc. på adr. 00FA  - ny adr. L
  F8FF  A5FF    LDA  Indlæs tal fra adr. 00FF - adresse H
  F901  85FB    STA  Skriv acc. på adr. 00FB  - ny adr. H
  F903  A5FA    LDA  Indlæs tal fra adr. 00FA
  F905  D001    BNE  Spring til F907 hvis forsk. fra 00
  F907  60      RTS  Retur til hovedprogram
  F908  E6FA    INC  Forøg ny adr. L med 1
  F90A  A000    LDY  Indlæs tallet 00 i y-reg.
  F90C  B1FA    LDA  Indlæs tal fra adresse i 00FA og 00FB = data
  F90E  C6FA    DEC  Formindsk adr. L med 1
  F910  91FA    STA  Skriv data i adr. angivet i 00FA og 00FB
  F912  E6FA    INC  Forøg ny adr. med 1
  F914  4C03F9  JMP  Spring til F903 = nyt gennemløb

indsæt_byte:
  F917  A9FF    LDA  Indlæs tallet FF i acc.
  F919  85FA    STA  Skriv acc. på adr. 00FA  - ny adr. L
  F91B  A5FF    LDA  Indlæs tal fra adr. 00FF - adresse H
  F91D  85FB    STA  Skriv acc. på adr. 00FB  - ny adr. H
  F91F  A5FA    LDA  Indlæs tal fra adr. 00FA
  F921  C5FE    CMP  Sammenlign med adr. 00FE
  F923  D001    BNE  Spring til F926 hvis forskellige
  F925  60      RTS  Retur til hovedprogram
  F926  C6FA    DEC  Formindsk ny adr. L med 1
  F928  A000    LDY  Indlæs tallet 00 i y-reg.
  F92A  B1FA    LDA  Indlæs tal fra adresse i 00FA og 00FB = data
  F92C  E6FA    INC  Forøg adr. L med 1
  F92E  91FA    STA  Skriv data i adr. angivet i 00FA og 00FB
  F930  C6FA    DEC  Formindsk ny adr. med 1
  F932  4C1FF9  JMP  Spring til F91F = nyt gennemløb
```

Styresystemet er i hovedsagen opbygget som vist på side 78. Her skal dog vises nogle rutediagrammer over nogle af subrutinerne:

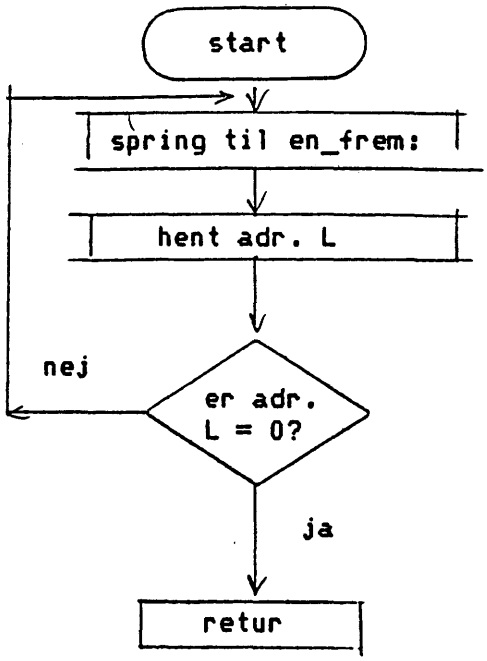
en_frem:



en_bak:

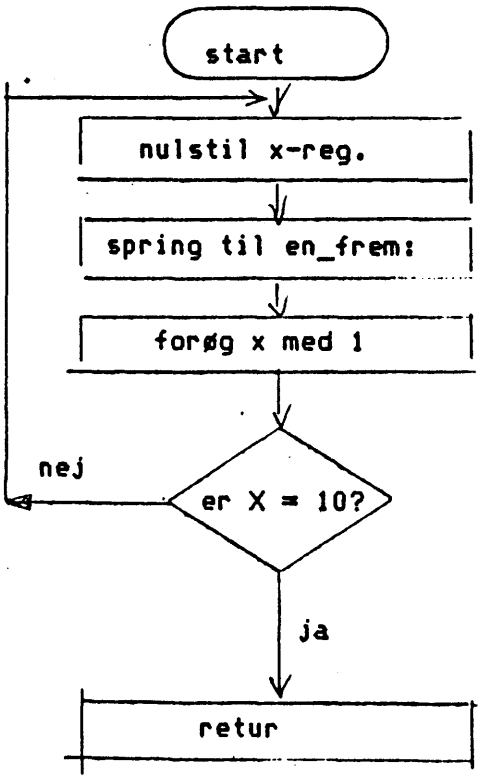


side_op:

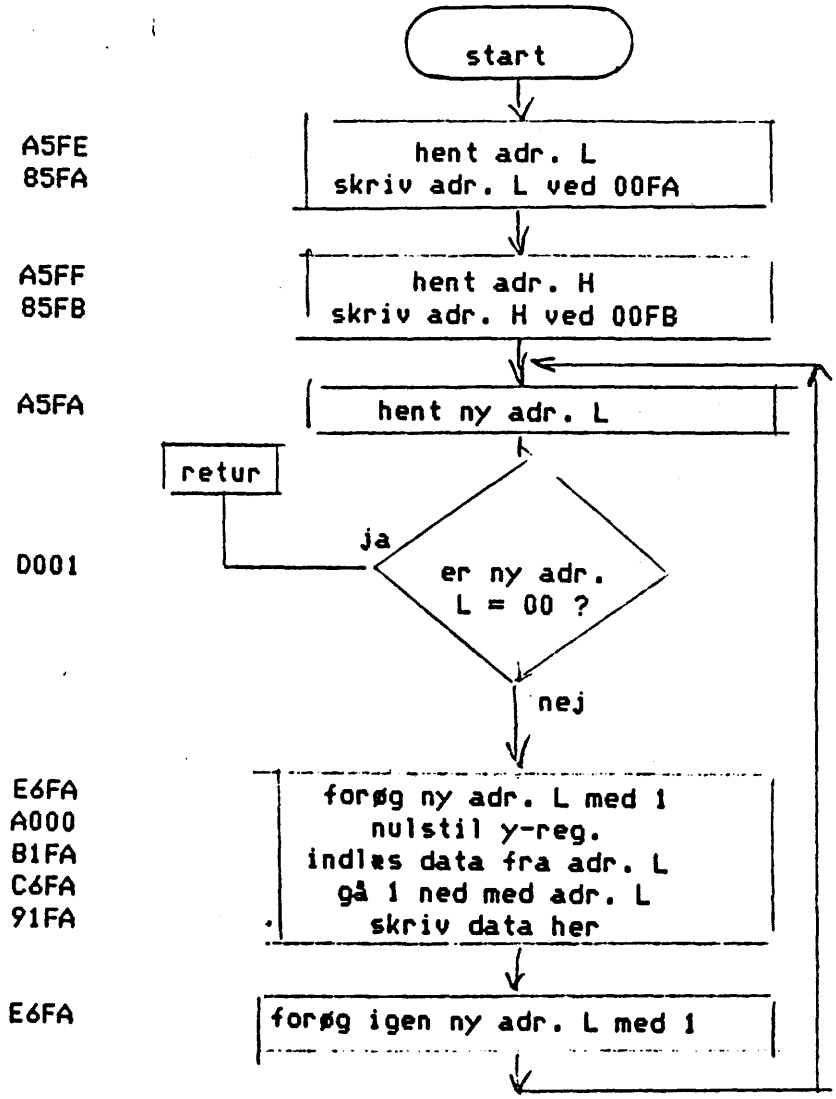


"side_ned" er opbygget helt tilsvarende, blot springes der til "en_bak:" i stedet for til "en_frem:"

10_frem:



slet_byte:

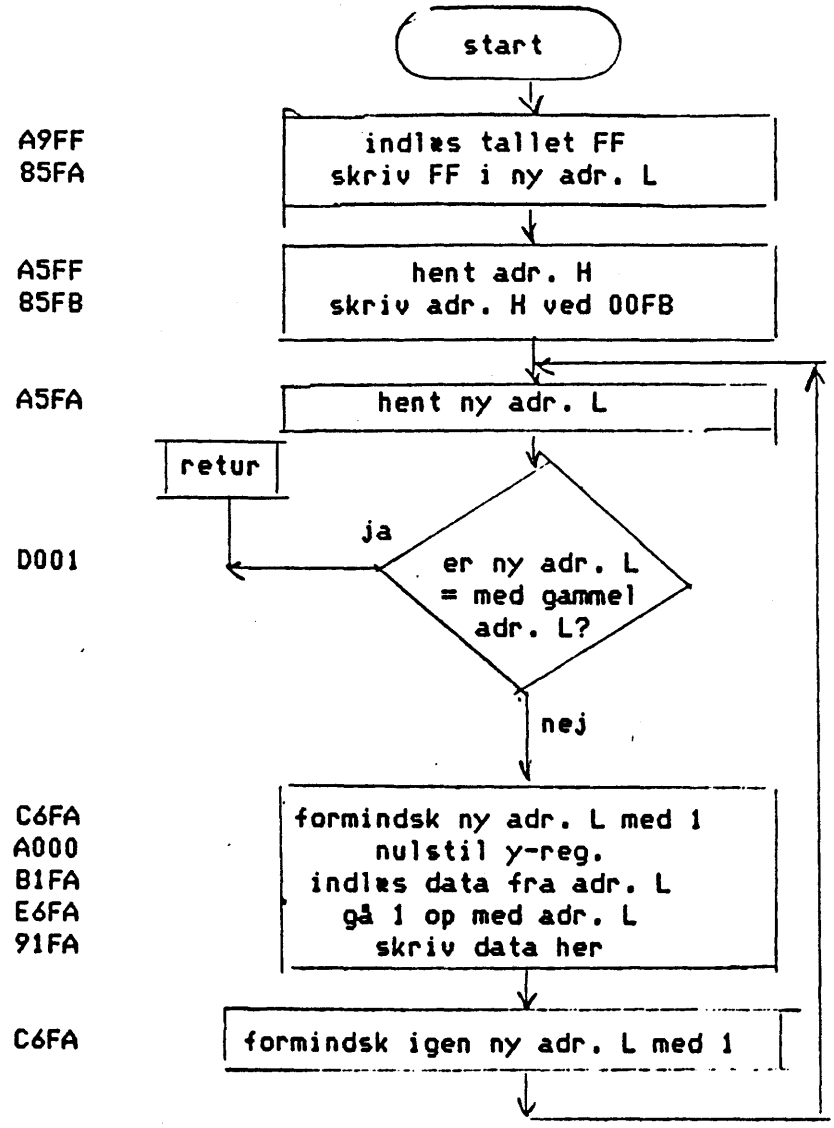


Da "slet_byte" bryder om på en hel side, kan funktionen ikke anvendes på side 00 (her er styresystemets data gemt) eller på side 01 (her ligger stacken). Funktionen virker heller ikke hvis adresse-L er 00.

Programmet virker således, at den byte, der slettes, erstattes med den efterfølgende, indtil siden er kørt helt op (adr. L = 00).

I "indsæt_byte:" på næste side, sker der det modsatte: der fjernes en byte fra toppen af siden, byten nedenunder rykkes en plads op osv. Således fortsættes, til den aktuelle adresse er nået.

indsæt-byte:



PAUSE-PROGRAMMER:

kort_pause: (varighed ved adr. 00A1)

```
FA00  48      PHA  Gem acc. i stack
FA01  08      PHP  Gem statusregister i stack
FA02  A900    LDA  Indlæs tallet 00
FA04  85A0    STA  Skriv acc. i adr. 00A0
FA06  A5A0    LDA  Indlæs tal fra adr. 00A0 i acc.
FA08  C5A1    CMP  Sammenlign med adr. 00A1
FA0A  F005    BEQ  Spring til FA11 hvis ens
FA0C  E6A0    INC  Forøg adr. 00A0 med 1
```

```
  FA0E  4C06FA  JMP  Spring til FA06 og lav et nyt gennemløb
  FA11  28      PLP  Hent statusregister fra stack
  FA12  68      PLA  Hent accumulator fra stack
  FA13  60      RTS  Retur


mellem_pause:   (variabel ved adr. 00A3)
  FA20  48      PHA  Gem acc. i stack
  FA21  08      PHP  Gem statusregister i stack
  FA22  A900    LDA  Indlæs tallet 00
  FA24  85A2    STA  Skriv acc. i adr. 00A2
  FA26  A5A2    LDA  Indlæs tal fra adr. 00A2 i acc.
  FA28  C5A3    CMP  Sammenlign med adr. 00A3
  FA2A  F008    BEQ  Spring til FA34 hvis ens
  FA2C  E6A2    INC  Forøg adr. 00A2 med 1
  FA2E  2000FA  JSR  Spring til kort_pause:
  FA31  4C26FA  JMP  Spring til FA26 og lav et nyt gennemløb
  FA34  28      PLP  Hent statusregister fra stack
  FA35  68      PLA  Hent accumulator fra stack
  FA36  60      RTS  Retur


lang_pause:     (variabel ved adr. 00A5)
  FA40  48      PHA  Gem acc. i stack
  FA41  08      PHP  Gem statusregister i stack
  FA42  A900    LDA  Indlæs tallet 00
  FA44  85A4    STA  Skriv acc. i adr. 00A4
  FA46  A5A4    LDA  Indlæs tal fra adr. 00A4 i acc.
  FA48  C5A5    CMP  Sammenlign med adr. 00A5
  FA4A  F008    BEQ  Spring til FA54 hvis ens
  FA4C  E6A4    INC  Forøg adr. 00A4 med 1
  FA4E  2020FA  JMP  Spring til mellem_pause:
  FA51  4C46FA  JMP  Spring til FA46 og lav et nyt gennemløb
  FA54  28      PLP  Hent statusregister fra stack
  FA55  68      PLA  Hent accumulator fra stack
  FA56  60      RTS  Retur



DEMOPROGRAMMER som ikke er vist i teksten:


Beep med variabel længde:
  FE00  A960    LDA  Indlæs tallet 60 i acc.
  FE02  85A1    STA  Skriv acc. på adr. 00A1 (kort pause)
  FE04  8D0680  LDA  Skriv til adr. 8006 = beep
  FE07  2020FA  JSR  Spring til pause
  FE0A  E6A3    INC  Forøg adr. 00A3 med 1 ( mellem pause)
```

```
  FE0C  E6A3    INC  Forøg adr. 00A3 med 1 ( mellem pause)
  FE0E  E6A3    INC  Forøg adr. 00A3 med 1 ( mellem pause)
  FE10  E6A3    INC  Forøg adr. 00A3 med 1 ( mellem pause)
  FE12  38      SEC  Sæt carry-flag
  FE13  B0EF    BCS  Spring op og begynd forfra
```

Løbelys på porten med beep:

```
  FE20  A970    LDA  Indlæs tallet 70
  FE22  85A1    STA  Skriv acc. i adr. A1 = kort_pause
  FE24  85A3    STA  Skriv acc. i adr. A3 = mellem_pause
  FE26  18      CLC  Slet menten
  FE27  8D0F80  STA  Skriv acc. i adr. 800F = porten
  FE2A  2020FA  JSR  Spring til pause
  FE2D  2A      ROL  Roter til venstre
  FE2E  8D0680  STA  Skriv til 8006 = beep
  FE31  D0F4    BNE  Spring tilbage til FEA7
```

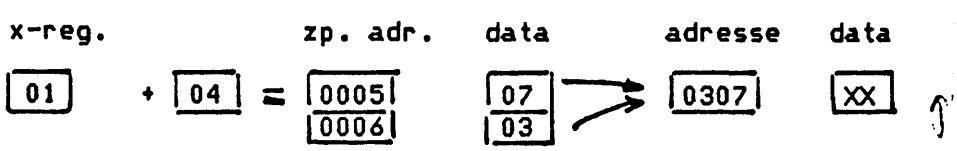
Tæller op på alle display:

```
  FE40  F8      SED  Sæt decimal mode
  FE41  A970    LDA  Indlæs tallet 70
  FE43  85A1    STA  Skriv acc. i adr. A1 = kort_pause
  FE45  85A3    STA  Skriv acc. i adr. A3 = mellem_pause
  FE47  A900    LDA  Indlæs tallet 00 (startværdi)
  FE49  18      CLC  Slet menten
  FE4A  6901    ADC  Læg 1 til acc.
  FE4C  8D0080  STA  Skriv til display 0
  FE4F  8D0180  STA  Skriv til display 1
  FE52  8D0280  STA  Skriv til display 2
  FE55  2020FA  JSR  Spring til pause
  FE58  18      CLC  Slet menten
  FE59  90EF    BCC  Spring til FEC8
```

ADRESSERINGSFORMER

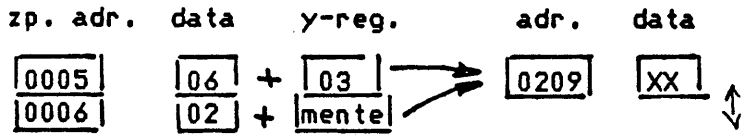
IMMIDIATE	Tager den kommende byte som talkonstant A902: LDA med tallet 02
ABSOLUTE	Henviser til en fuldstændig adresse AD0380: LDA med tal fra adresse 8003

ZERO PAGE	Den høje adresse-byte er 00 (side 09 A550: LDA med tal fra adresse 0050
ACCUM	Forandringer i accumulatoren 2A: ROL - flyt bits en plads til venstre i acc.
IMPLIED	Kun op-kode, operanden skal ikke bruges E8: INX forøg x-reg. med 1
(IND.X)	Man henter en to-byte adresse ved hjælp af 1 byte. Mange adresser kan blades igennem ved at X forøges. Kun på zero page. A104: LDA med data fra den adresse, der findes ved 0004+X og den efterfølgende celle.



Man kan blade ved at forøge X med 2. Mente fra x-reg. tælles ikke med.

(IND),Y	Man henter en to-byte adresse ved hjælp af 1 byte. Mange adresser kan blades igennem ved at Y forøges. Kun på zero page. B105: LDA med data fra den adresse, der findes ved 0005 og 0006. Hent data fra den adresser, der fremkommer, ved at Y lægges til det fundne tal.



Man kan blade ved at forøge Y med 1. Mente fra y-reg. tælles med i adresse-H.

Z PAGE X

X lægges til den foreslåede adresse. Ved at forøge X kan der blades i adresserne.
9500: STA på zero page adrees + X.

x-reg.		zp. adr.		data
01	+ 00	= 0001		XX

Z PAGE Y

Svarer til ovenstående, men er ikke så alsidig.

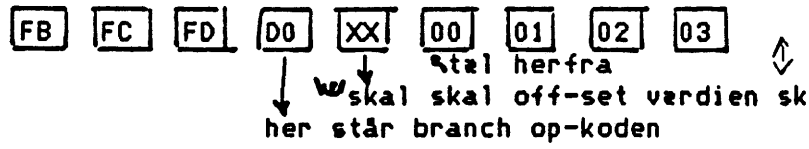
ABS.X = ABS.Y

Svarer til ovenstående men kan bruges i hele adresserummet.
990006: STA på den absolutte adresse, der fås ved 0600 + X

x.reg.		adr.	data
03	+ 00	= 0603	XX

RELATIVE

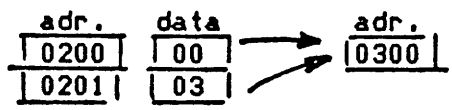
Bruges i forbindelse med branc-instruktioner.
9002: BCC spring 2 frem, hvis der ike er mente
D0FC: BNE spring 4 tilbage, hvis tallene er forskellige.



Se også skemaet på side 112.

INDIRECT

Indirekte adressering i forbindelse med JMP.
6C0002: JMP til den adresse, der findes i 0200 og 0201.



SAMLING AF MIKRO-DASK) dokumentation af hardware

MIKRO-DASK består af følgende moduler: CPU-modul, adressedekoder, memory-modul, port-modul, tastatur, display-modul. Disse moduler skal monteres på en aluminiumsplade:

Før det beskrives, hvorledes maskinen samles, skal de forskellige udtryk vedrørende kontrolsignaler forklares. Det er CPU-en, der hele tiden afgør, hvad der skal ske i maskinen. Til det formål afgiver den bl.a. et signal fra det spyd, der er mærket R/$\overline{W}$. Dette betyder read/write (læs/skriv).

Bemærk, at der er en streg over $\overline{W}$. Det betyder, at $\overline{W}$ er aktiv lav, hvilket vil sige, at CPU-en giver besked om, at den agter at skrive (f.eks. til en hukommelse) ved, at den gør punktet lavt (0V). Hvis punktet derimod er højt, betyder det, at CPU-en agter at læse fra hukommelsesenheden. På memory-modulet, på display-modulet og på port-modulet sidder der et tilsvarende R/$\overline{W}$-spyd.

Forbind nu de forskellige kontrolledninger (=kontrolbussen) således:

1. "R/W" på CPU-modulet forbindes med "R/$\overline{W}$" på display-modulet (blå ledninger), på memory-modulet og på port-modulet. "Ø2" på CPU-modulet afgiver en clock-signal, som skal forbindes til "Ø2" på dekoderen.

2. Herefter skal de forskellige enheder have en adresseledning (blå) fra adressedekoderen:

adresse	enhed
8000	Display 0
8001	Display 1
8002	Display 2
8003	Enable 1 på tastatur
8004	Enable 2 på tastatur
8005	Reset på tastatur
8006	Ind beeper
800F	Enable (En) på port-modulet

3. Forbind nu alle data- og adresse-busser og giv alle moduler såvel +5V (røde ledninger) som 0V (sorte ledninger).

4. [illegible]

FORKLARING AF DIAGRAMMER

CPU-MODUL

IC1 er processoren - 6502. Dens adressebus sidder på ben 9 til 25, mens databussen sidder på ben 26-33. Adresse-signaler går kun ud fra processoren, mens datasignaler går både ud af og ind i den.

IC2+3 hedder 74373. De indeholder hver 8 latch-kredsløb, der dog her blot anvendes som buffere for adresseledningerne. IC4 er en 74245, der kan lede signaler begge veje: når ben 1 er lav, går signalerne ud på databussen, og når ben 1 er høj, går signalerne ind i processoren. Dette styres af processorens R/$\overline{W}$-signal.

Clock-kredsløbet er krystalstyret og afgiver en frekvens på 1MHz. Reset kredsløbet bevirker, at processoren kan resettes med et tryk på KT og når der tilsluttes spænding.

KOMPONENTER CPU

R1: 33K	R9-10: 1,5K	C2: 680pF	IC1: 6502
R2: 2,2K	R11: 3,3K	C3: 470pF	IC2: 74373
R3: 33K	R12: 220 ohm	C4: 10uF	IC3: 74373
R4: 470 ohm	T1: BC547	C5: 10uF	IC4: 74245
R5: 1K	T1: BC547	C6: 47nF	IC5: 74132
R6: 1K	KT: digitast	C7: 47nF	1 stk 40 ben tulip.
R7: 100 ohm	K1: 1MHz kryst.	C8: 220uF	4 stk 16 ben sokl.
R8: 3,3K	C1: 180pF	C9: 22pF trim. kondens.	

evt.: 1 stk 14 ben tulipansokler, 3 stk 20 ben tulipansokler

ADRESSEDEKODER

IC3 er en 74154. Når ben 18 og 19 er lave, vil den omsætte en binær talværdi, der sendes ind på ben 20-23 til en dekodet værdi, der får et af benene mellem 1 og 17 til at gå lav.

Hvis den binære værdi er 0 0 0 0, vil ben 1 gå lav
Hvis den binære værdi er 0 0 0 1, vil ben 2 gå lav
.....
Hvis den binære værdi er 1 1 1 1, vil ben 17 gå lav

Ben 19 er lav, hvis A15 er høj. Ben 18 er lav, hvis A4-A14 er lave (samtidig med at "Ø2" er høj). Dette gør, at dekoderen træder i funktion ved disse adresseværdier mellem 8000 og 800F:

A15-A14-A13-A12	A11-A10-A9-A8	A7-A6-A5-A4	A3- A2- A1- A0
1 0 0 0	0 0 0 0	0 0 0 0	1/0 1/0 1/0 1/0
8	0	0	0-F

KOMPONENTER DEKODER

C1: 470uF R17: 150 ohm T1-16: BC557 IC3: 74154
C2: 100nF R18-19: 1K IC1-2: 7405 IC4: 74132
R1-16: 10K D1-16: lysdioder 1 stk 24 ben tulipansokkel

evt.: 3 stk 14 ben tulipansokler, 1 stk 14 ben alm. sokkel

BEEPER, AMV

IC1 er en 74132, hvor to gates er koblet som en monostabil multivibrator. Hvis IND går lav, vil ben 6 også gå lav, og piezo-lydgiveren får et spændingsfald over sig, hvilket får den til at afgive lyd.
De to andre gates er sammen med en transistor koblet som en langsom AMV, der med mellemrum på ca 2 sek. afgiver et lavt signal. Transistoren gør det muligt at få en særlig langsom impulsgivning med en forholdsvis lille kondensator.

KOMPONENTER BEEPER, AMV

IC5: 74132 R23: 100 ohm T17: BC547 C4: 470uF
R20-21: 680 ohm R24: 330 ohm C3: 47uF Kr1: piezo
R22: 33K D17: 1N4148 1 stk 16 ben sokkel

MEMORY-MODUL

IC5 bliver kun aktiv, hvis ben 18 er lav. Dette vil kun ske, hvis A11-A15 alle er høje. Hvis A0-A10 alle er lave, kan IC5 adresseres med F800. Hvis også A0-A10 er høje, adresseres IC5 med FFFF.

IC6 bliver også kun aktiv, hvis ben 18 er lav. Dette vil kun ske, hvis A15-A11 er alle er lave. Adresse A0-A10 kan være enten lave eller høje, hvilket giver kredsen adresserummet fra 0000 til 0700.

Ved IC7 sidder der 5 stk EXCLUSIVE-OR gates. En udgang er her kun høj, hvis enten kontakten er afbrudt eller adressebit-en er høj. De steder, hvor kontakten afbrydes, skal den tilsvarende adressebit være lav, for at ben 18 på IC7 går lav. Hvis alle kontakter derfor står åbne, skal A11-A15 være lave og det valgte adresserum er fra 0000 til 0700. Hvis kontakten ved A12 sluttes, skal A12 være høj, for at kredsen adresseres. Det giver et adresserum fra 1000 til 1FFF

R/$\overline{W}$-signalet sendes ind i nogle buffere fra IC4 og fortsætter til RAM-kredsen. Hvis der i memory 3 indsættes en RAM-kreds, skal K6 sluttes for at den kan modtage signalet.

De tre transistorer trækker lysdioder, der angiver, om den pågældende memory-kreds bliver adresseret.

KOMPONENTER MEMORY-MODUL

R1-4: 1,5K	C2: 220uF	IC1-2: 7414	IC6: 6116 RAM
R5-7: 10K	D1-15: AA119	IC3-4: 7486	IC7: 6116/2716
R8-10: 330 ohm	- evt. 1N4148	IC5: 2716 EPROM	KT: mikroswitch
C1: 220uF	D16-18: lysd.	1 stk 24 ben sok.	3 stk 16 ben sk

evt.: 4 stk 14 ben sokler

Punkter, der er afmærket med "X" er beregnet til andre typer hukommelseskredse. Teoretisk set skal D1-5 være AA119, men 1N4148 plejer også at virke. IC5 skal indeholde styresystemet.

UDLÆSNINGSMODUL - DISPLAY

I kredsløbet indgår 3 OR-gates. De er hver især forbundet til "Enable" på displayene. Når dette punkt går lav, vil bitmønsteret fra databussen overføres til IC1+2 i displayene (9368). Disse kredse

virker både som latch og dekoder. Det vil sige, at bitmønstret vil blive stående i IC1+2 til næste gang, der kommer en impuls på "Enable".
For at latchningen kan lykkes, skal R/$\overline{W}$ være lav samtidig med, at display-enable (D0E - D2E) er lav. Det vil sige, at der samtidigt skal komme et R/$\overline{W}$-signal fra CPU-en og et enable-signal fra dekoderen.

KOMPONENTER UDLÆSNINGSMODUL

IC1: 74132 — 3 stk. konnektorer, 12-polede (Commodore)
2 stk 16 ben tulip. — evt.: 1 stk 14 ben sokkel

KOMPONENTER DISPLAY

IC1-2: 9368 — DS1-2: D350PKH, display med fælles katode
3 stk. isol. lus — 2 stk 16 ben tulipansokler

Der skal i alt bruges 4 stk. display.

TASTATUR

Hjertet i tastaturet udgøres af IC1, IC3 og IC5. IC1 (74132) indeholder en NAND-gate, der sender impulser ind i IC3, der er en binær tæller (7493). Tællerens bitmønster sendes ind i IC4 (74154), hvor det dekodes. Det betyder, at der her hele tiden vil være et af 16 mulige ben, der er lav - afhængig af hvilket bitmønster tælleren viser. Hvis der trykkes på en af tastaturets knapper (0-F), vil IC1 holde op med at sende impulser ind i tælleren.

Nu vil tællerens bitmønster latches ind i IC5 (74193), der modtager en latch-impuls på ben 11. Herved er de fire laveste bits blevet dannet. Næste gang, der trykkes på en tast, vil dette bitmønster overføres til IC6 og her danne de fire høje bits. IC5 vil således hele tiden indeholde de nydannede bits, mens IC6 vil indeholde de gamle. Fordelingen opnås ved, at C11 forsinker latch-impulsen til IC5.

Bitmønsteret er nu rykket frem til IC9 (74373), hvor udgangene er af tri-state typen. Det betyder, at de kan være lave - høje eller højimpedansede. Den sidste tilstand betyder, at de slet ikke påvirker databussen - det virker som om forbindelsen hertil afbrydes.

Når IC9 fra dekoderen modtager en enable-impuls på "En.1", vil kredsen forlade højimpedanstilstanden og sende sit indhold ud på

databussen.

Så mangler vi at få forklaret kontrol-tastaturet. Dette aktiveres, hvis der trykkes på den sepearte digitast (KT1), eller hvis omskifteren står på ON. Herved sendes bitmønstret fra tælleren (IC3) over i IC7 (74193), der får en latch-impuls på ben 11. Hvis "En.2" modtager en impuls fra dekoderen, vil IC8 (73373) forlade sin højimpedanstilstand og sende bitmønstret ud på databussen. Bemærk at kontrol-tastaturets bitmønster altid er 0 0 0 0 for de fire høje bits. Kontroltastaturet kan resettes ved hjælp af en impuls fra dekoderen til "Res".

KOMPONENTER TASTATUR

R1-6: 1,5K	C4-7: 100nF	IC1: 74132	IC8-9: 74373
R7: 10K	C8: 2,2uF	IC2: 7402	16 hvide digitst.
R8: 470 ohm	C9: 1000uF	IC3: 7493	1 sort digitast
R9-16: 330 ohm	C10: 100nF	IC4: 74154	1 miniomskifter
R17: 1,5K	C11: 10uF	IC5-7: 74193	1 printkonnektor
C1-3: 10nF	D1-8: lysdioder	1 stk konnekt.	1 stk 16 ben tulip

evt.: 5 stk 16 ben sokler 1 stk 20 ben sokler
1 stk 16 ben tulipansokkel 1 stk 20 ben tulipansokkel

PORT

Porten indeholder to stk. 74373 (IC1-2). IC1 danner udporten. Her er ben 1 hele tiden lav, hvorved tristateudgangene aldrig bliver højimpedansede. Hvis der både kommer en lav impuls på "En." fra dekoderen og en lav impuls på "R/W", vil kredsen latches (via ben 11) og den sender bitmønsteret ud af porten.

IC2 vender modsat af IC1. Her er ben 11 vedvarende høj, så den er altid klart til at latche. Normalt vil dens udgange, der vender ud på databussen være højimpedansede. Hvis "R/$\overline{W}$" imidlertid bliver høj samtidig med, at "En." er lav, vil kredsens bitmønster blive sendt ud på databussen - der læses fra kredsen.

KOMPONENTLISTE - PORT

R1-8: 470 ohm	IC1-2: 74373	C1: 220uF	C2: 100nF
D1-8: lysdioder	IC3: 7402	1 stk 16 ben sk.	1 stk 16 tulip.

evt.: 1 stk 14 ben tulip, 2 stk 20 ben tulipansokler

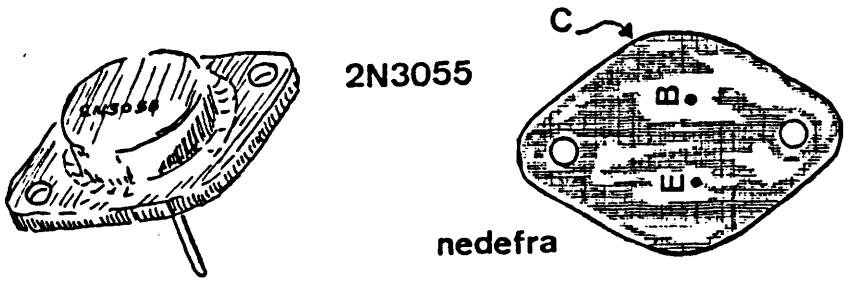
STRØMFORSYNING

Vekselstrømmen fra transformatoren ensrettes af D1-4 og udglattes af C1. IC1 er en 7805, der afgiver en stabiliseret spænding på 5V. Denne spænding "løftes lidt op" af D5-6 og sendes ind i T1 (2N3055), der er koblet som emitterfølger. Det vil sige, at den afgiver (næsten) den samme spænding, som den får ind på basis.

D7 kortslutter eventuelle indgående transienter og C3 fjerner eventuel støj.

KOMPONENTER - STRØMFORSYNING

R1: 22 ohm/5W	D5-7: 1N4007	IC1: 7805	C2: 100nF
D1-4: 1N5404	T1: 2N3055	C1: 2200uF	C3: 2,2uF
Trafo 10V/3A 25V/100mA: DT 8013-1		Køleplade: KL 102	



STRØMFORSYNING - BRÆNDING

Vekselspændingen fra transformatoren ensrettes af D1-4. Den udglattes af C1 og stabiliseres til 25V af IC1 og D5. IC1 hedder 78L24 og den afgiver en stabiliseret spænding på 24V. Denne spænding "løftes" 0,7V af D5. T1 er koblet som emitterfølger, og vil, hvis den leder, afgive ca. 25V (det får den ind på basis).

Hvis den afgivne strøm bliver større end 70mA (her går de 20mA til lysdioden D7), vil der over R8 (= 10 ohm) lægge sig omkring 0,7V - hvilket får T5 til at lede.

Herved slår flip/floppen, der består af T2 og T3, om (basis trækkes lav af T4), og emitterfølgeren (T1) vil spærre. Flip/floppen kan bringes til at slå om igen, hvis basis på T3 trækkes lav ved hjælp af K2.

Zenerdioden D8 vil sammen med spændingsfaldet over lysdioden be-

virke, at denne først vil lyse, når udgangsspændingen er på omkring 20V.

KOMPONENTER - BRÆNDE-STRØMFORSYNING

R1: 10K	R5: 100K	R9: 220 ohm	D8: 18V zenerdiode
R2: 100K	R6: 100K	C1: 220uF	T1-4: BC547
R3: 10K	R7: 1K	D1-6: 1N4148	T5: BC557
R4: 1,5K	R8: 10 ohm	D7: lysdiode	IC1: 78L24

EPROM-BRÆNDER

Modulet indeholder 5 stk. 74193 (IC1-5), der er op/ned-tællere. De kan "loades" med et bitmønster, hvis ben 11 gøres høj. Dette bitmønster kan så tælles op eller ned, afhængig af, om der tilføres impulser på ben 5 eller 4. IC1-2 bruges til at omdanne bitmønstret fra databussen til de 8 lave adressebits. Dette sker, når "A1" får en lav impuls.

På tilsvarende måde kan databussens indhold danne de næste 3 adressebits, hvis "A2" gøres lav. Herved er der i alt dannet 11 adressebits, hvilket er det, som EPROM-en 2716 kan rumme (2^{11} = 2048 = 2k-bytes).

Endelig kan databussens bitmønster overføres som data, hvis "DE" gøres lav. Når der således tilført de rigtige adressebits og de rigtige databits, er EPROM-en klar til at blive brændt. Dette sker ved, at der tilføres en kortvarig (50msek.) høj-impuls til ben 18, mens ben 21 har en spænding på 25V.
IC6 danner en monostabil multivibrator, der afgiver den krævede høje impuls, når "BE" går lav.

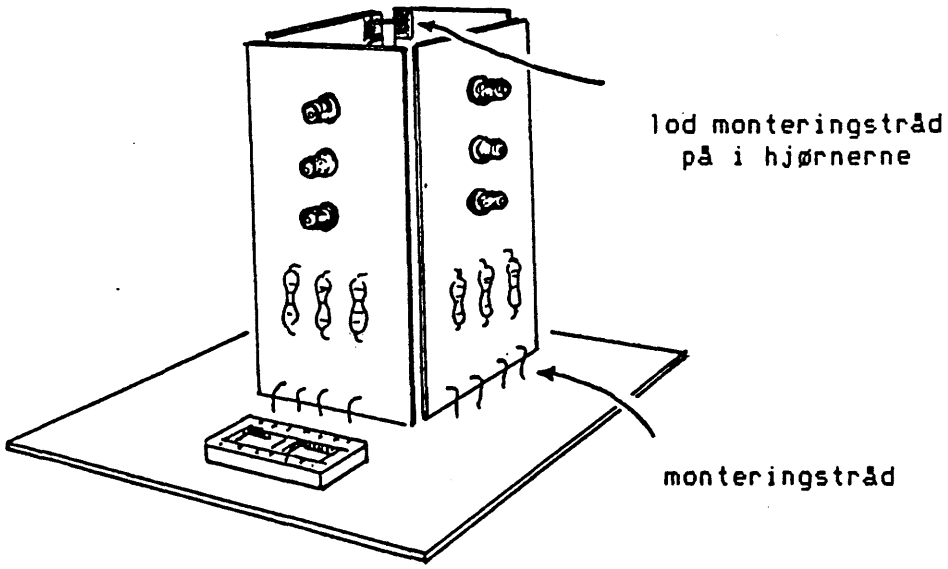
KOMPONENTER BRÆNDER

R1: 680 ohm	C2: 10uF	IC1-5: 74193	IC6: 74132
C1: 22uF	1 stk 24 ben tup	1 stk 16 ben tup	

evt.: 4 stk 16 ben tulipansokler 1 stk 14 ben tulipansokkel

LYSKURV

Lyskurven er delt op i 2 parallelle dele. Lysdioder, der hører til trafikretning 2, får 0V, når bit 3 er lav, mens lysdioder, der hører til trafikretning 1 får 0V, når bit 7 er lav. For at en lysdiode kan lyse, kræves der yderligere, at dens styrebit er høj.



KOMPONENTER - LYSKURV

R1-2: 330 ohm D1-4: røde lysd. D5-8: gule lysdioder
D9-12: grønne lysd. 1 stk 16 ben sokkel

PORTKONTAKT

Samtlige bits er normalt trukket lave ved hjælp af R1-8. En bit kan imidlertid trækkes høj ved, at den tilhørende kontakt aktiveres.

KOMPONENTER - PORTKONTAKT

K1-8: digitaster (0-7) R1-8: 2,2K 1 stk 16 ben sokkel

MELLEMLED

Mellemleddet indeholder 4 stk 74193 (op/ned-tælllere). Indgangene til disse forbindes ved hjælp af bus-kablet til tastaturet før IC9. Det vil sige, at der på indgangene hele tiden vil stå det bitmønster, der stammer fra indtastningen. Hvis "Adr.1" aktiveres vil bitmønstret latches og overføres som adressebit 0-3. Hvis "Adr.2" aktiveres, vil bitmønstret imidlertid overføres som bit 4-8.

Mellemleddets data-del forbindes ved hjælp af bus-kablet til punkterne efter IC9. Normalt vil ledningerne her være højimpedansede, hvilket gør, at det er en evt. tilkoblet memory-kreds, der bestemmer dataindholdet. Hvis "En.1" og "R/$\overline{W}$" forbindes til de tilsvarende punkter på tastatur og memory, vil et tryk på "Data" imidlertid bevirke, at tastaturets data sendes ud på bussen og skrives i en evt. RAM-kreds.

IC5 danner to flip/flops, der kan bruges til op- og nedtælling af adresserne.

KOMPONENTER - MELLEMLED

R1: 680 ohm	IC1-4: 74193	C1: 22uF	C3: 100nF
R2: 100 ohm	IC5: 7400	C2: 220uF	5 stk. digitaster
R3-4: 1,5K	3 stk 16 ben tulipansokler		

evt.: 5 stk 16 ben tulipansokler

MUSIK-MODUL

Ben 27 og 29 på IC3 (musik-kredsen) har betydning for hvilke funktioner, der udføres. Benene kan enten være høje eller lave og mulighederne fremgår af dette skema:

ben 27	ben 29	Funktion
0	0	inaktiv
0	1	læs fra kredsen (Read)
1	0	skriv til kredsen (Write)
1	1	udvælg register (Reg.)

Kredsløbet, der er opbygget med inverterne fra IC1 og IC2, har til formål at udvælge netop en af de tre sidste funktioner. Hvis "Reg." er lav, vil både ben 27 og 29 være høje. Hvis "Write" gøres lav, vil ben 27 være høj og ben 29 lav. På tilsvarende måde vil ben 27 være lav og ben 29 høj, hvis "Read" gøres lav - sammenlign med skemaet. Denne udvælgelse kan nu ske ved hjælp af adressedekoderen.

Musik-kredsen er forbundet til databussen, så de forskellige data kan føres frem og tilbage. Kredsen laver sine toner ud fra krystal-frekvensen, som den modtager fra CPU-modulet via Ø2. Den kan resettes ved hjælp af "Reset". Kredsens musik-signaler forstærkes op af en forstærker, der er opbygget omkring IC4.

KOMPONONTER - MUSIKMODUL

IC1: 7404	R1: 2,2K	C1: 470uF	C6: 47nF
IC2: 7405	R2: 2,2K	C2: 100nF	C7: 220uF
IC3: AY-3-8910A	R3: 4,7K	C3: 330pF	P1: 1K drejep.
IC4: 386	R4: 470 ohm	C4: 2,2uF	8 ohm højttaler
1 stk drejeknap	R5: 10 ohm	C5: 10uF	1 stk 40 ben tup
1 stk 16 ben sok.			

evt.: 2 stk 14 ben tulipansokler, 1 stk 8 ben tulipansokkel

STEPMOTOR

Stepmotoren er karakteristisk ved, at den kun kan køre, hvis den modtager impulser. Hver impuls vil få akslen til at "steppe" lidt frem. IC2 indeholder styre-elektronik til stepmotoren. IC1 er koblet som en AMV, hvis frekvens kan ændres med P1. Impulserne kan sendes ind i triggerindgangen ("Trig.") på IC2.

KOMPONENTER - STEPMOTOR

IC1: 555	R1: 15K	C1: 2,2uF
IC2: SAA1042	P1: 10K drejep.	M: stepmotor 6V, SAIA UBD 13 403
1 stk drejekn. 1 stk 16 ben sok.		

evt.: 1 stk 8 ben sokkel

På 16 ben-soklen klippes de 4 midterste ben af.

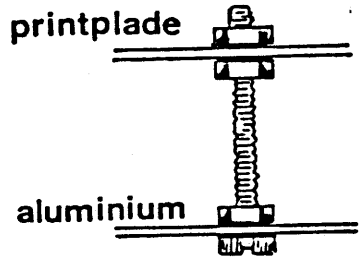
DESUDEN SKAL BRUGES

ca. 100 stk runde spyd med fatninger
2m fladkabel med 16 ledere
4m rød ledning
4m sort ledning
6m blå ledning
16 stk BERG 16-polede klem-stik til fladkabel
50 stk 3mm bolte 30mm
150 stk 3mm møtrikker
4 stk 2mm bolte med møtrikker
1 stk kantbøjet aluminiumsbund (35 x 105 cm, 1mm tyk)

MONTERINGSVEJLEDNING

Hullerne ved IC-erne bores med et 0,8mm bor. Huller til de andre komponenter bores med 1mm bor. Huller til D1-4 på strømforsyning og til printspyd bores med 1,3mm.

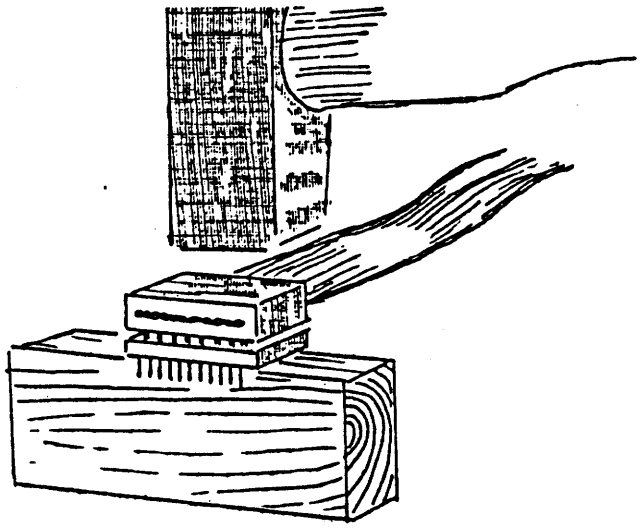
Være opmærksom på, at nogle komponentben skal loddes fast på begge sider, idet de bruges til at skabe forbindelse mellem printbaner på over- og undersiden. Dette kan også være tilfældet ved IC-ben. Hvis der her bruges sokler, er det nødvendigt med tulipansokler, idet almindelige sokler ikke kan loddes på begge sider af printet. Enkelte steder, især ved tastaturet, skal der laves forbindelser mellem over- og underside med små stumper monteringstråd.
De 7 faste moduler monteres på aluminiumsbunden ved hjælp af bolte og møtrikker efter dette system:



Transformatoren fastgøres med de fire 2mm bolte. De løse moduler forsynes også med ben af bolte.

Der skal bruges omkring 10 røde og 10 sorte ledninger til forbindelser mellem +5V og mellem 0V. Ledningerne kan have forskellige længder (25-45cm). På tilsvarende måde laves der ca. 15 blå "signalledninger".

Fladkablet skal bruges til fremstilling af data- og adressebusser - studer billedet af den færdige MIKRO-DASK på side 20. Afmålingen foretages, når modulerne er monteret på aluminiumsbunden. Klemstikkene samles ved lette hammerslag, hvor stikket beskyttes af træklodser:



R6500 INSTRUCTION SET

INSTRUCTIONS		IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM			IMPLIED			(IND, X)		
MNEMONIC	OPERATION	OP	n	d	OP	n	d	OP	n	d	OP	n	d	OP	n	d	OP	n	d
ADC	A + M + C → A (4) (1)	69	2	2	6D	4	3	65	3	2							61	6	2
AND	A∧M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2
ASL	C ← 7 … 0 ← 0				0E	6	3	06	5	2	0A	2	1						
BCC	BRANCH ON C = 0 (2)																		
BCS	BRANCH ON C = 1 (2)																		
BEQ	BRANCH ON Z = 1 (2)																		
BIT	A∧M				2C	4	3	24	3	2									
BMI	BRANCH ON N = 1 (2)																		
BNE	BRANCH ON Z = 0 (2)																		
BPL	BRANCH ON N = 0 (2)																		
BRK	BREAK (See Fig. 1)													00	7	1			
BVC	BRANCH ON V = 0 (2)																		
BVS	BRANCH ON V = 1 (2)																		
CLC	0 → C													18	2	1			
CLD	0 → D													D8	2	1			
CLI	0 → I													58	2	1			
CLV	0 → V													B8	2	1			
CMP	A − M	C9	2	2	CD	4	3	C5	3	2							C1	6	2
CPX	X − M	E0	2	2	EC	4	3	E4	3	2									
CPY	Y − M	C0	2	2	CC	4	3	C4	3	2									
DEC	M − 1 → M				CE	6	3	C6	5	2									
DEX	X − 1 → X													CA	2	1			
DEY	Y − 1 → Y													88	2	1			
EOR	A ⊻ M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2
INC	M + 1 → M				EE	6	3	E6	5	2									
INX	X + 1 → X													E8	2	1			
INY	Y + 1 → Y													C8	2	1			
JMP	JUMP TO NEW LOC				4C	3	3												
JSR	JUMP SUB (See Fig. 2)				20	6	3												
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2									
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2									
LSR	0 → 7 … 0 → C				4E	6	3	46	5	2	4A	2	1						
NOP	NO OPERATION													EA	2	1			
ORA	A ∨ M → A	09	2	2	0D	4	3	05	3	2							01	6	2
PHA	A → Ms S − 1 → S													48	3	1			
PHP	P → Ms S − 1 → S													08	3	1			
PLA	S + 1 → S Ms → A													68	4	1			
PLP	S + 1 → S Ms → P													28	4	1			
ROL	7 … 0 ← C				2E	6	3	26	5	2	2A	2	1						
ROR	C → 7 … 0				6E	6	3	66	5	2	6A	2	1						
RTI	RTRN INT (See Fig. 1)													40	6	1			
RTS	RTRN SUB (See Fig. 2)													60	6	1			
SBC	A − M − C → A (1)	E9	2	2	ED	4	3	E5	3	2							E1	6	2
SEC	1 → C													38	2	1			
SED	1 → D													F8	2	1			
SEI	1 → I													78	2	1			
STA	A → M				8D	4	3	85	3	2							81	6	2
STX	X → M				8E	4	3	86	3	2									
STY	Y → M				8C	4	3	84	3	2									
TAX	A → X													AA	2	1			
TAY	A → Y													A8	2	1			
TSX	S → X													BA	2	1			
TXA	X → A													8A	2	1			
TXS	X → S													9A	2	1			
TYA	Y → A													98	2	1			

(IND), Y			Z PAGE, X			ABS X			ABS, Y			RELATIVE			INDIRECT			Z PAGE, Y			PROCESSOR STATUS CODES	
OP	n	d	OP	n	d	OP	n	d	OP	n	d	OP	n	d	OP	n	d	OP	n	d	7 6 5 4 3 2 1 0 N V · B D I Z C	MNEMONIC
71	5	2	75	4	2	7D	4	3	79	4	3										N V · · · · Z C	ADC
31	5	2	35	4	2	3D	4	3	39	4	3										N · · · · · Z ·	AND
			16	6	2	1E	7	3													N · · · · · Z C	ASL
												90	2	2							· · · · · · · ·	BCC
												B0	2	2							· · · · · · · ·	BCS
												F0	2	2							· · · · · · · ·	BEQ
																					M7 M6 · · · · Z ·	BIT
												30	2	2							· · · · · · · ·	BMI
												D0	2	2							· · · · · · · ·	BNE
												10	2	2							· · · · · · · ·	BPL
																					· · · 1 · · · ·	BRK
												50	2	2							· · · · · · · ·	BVC
												70	2	2							· · · · · · · ·	BVS
																					· · · · · · · 0	CLC
																					· · · · 0 · · ·	CLD
																					· · · · 0 · · ·	CLI
																					· 0 · · · · · ·	CLV
D1	5	2	D5	4	2	DD	4	3	D9	4	3										N · · · · · Z C	CMP
																					N · · · · · Z C	CPX
																					N · · · · · Z C	CPY
			D6	6	2	DE	7	3													N · · · · · Z ·	DEC
																					N · · · · · Z ·	DEX
																					N · · · · · Z ·	DEY
51	5	2	55	4	2	5D	4	3	59	4	3										N · · · · · Z ·	EOR
			F6	6	2	FE	7	3													N · · · · · Z ·	INC
																					N · · · · · Z ·	INX
																					N · · · · · Z ·	INY
															6C	5	3				· · · · · · · ·	JMP
																					· · · · · · · ·	JSR
B1	5	2	B5	4	2	BD	4	3	B9	4	3										N · · · · · Z ·	LDA
									BE	4	3							B6	4	2	N · · · · · Z ·	LDX
			B4	4	2	BC	4	3													N · · · · · Z ·	LDY
			56	6	2	5E	7	3													0 · · · · · Z C	LSR
																					· · · · · · · ·	NOP
11	5	2	15	4	2	1D	4	3	19	4	3										N · · · · · Z ·	ORA
																					· · · · · · · ·	PHA
																					· · · · · · · ·	PHP
																					N · · · · · Z ·	PLA
																					(RESTORED)	PLP
			36	6	2	3E	7	3													N · · · · · Z C	ROL
			76	6	2	7E	7	3													N · · · · · Z C	ROR
																					(RESTORED)	RTI
																					· · · · · · · ·	RTS
F1	5	2	F5	4	2	FD	4	3	F9	4	3										N V · · · · Z C	SBC
																					· · · · · · · 1	SEC
																					· · · · 1 · · ·	SED
																					· · · · 1 · · ·	SEI
91	6	2	95	4	2	9D	5	3	99	5	3										· · · · · · · ·	STA
																		96	4	2	· · · · · · · ·	STX
			94	4	2																· · · · · · · ·	STY
																					N · · · · · Z ·	TAX
																					N · · · · · Z ·	TAY
																					N · · · · · Z ·	TSX
																					N · · · · · Z ·	TXA
																					· · · · · · · ·	TXS
																					N · · · · · Z ·	TYA

(1) ADD 1 to "N" IF PAGE BOUNDARY IS CROSSED

(2) ADD 1 TO "N" IF BRANCH OCCURS TO SAME PAGE
ADD 2 TO "N" IF BRANCH OCCURS TO DIFFERENT PAGE

(3) CARRY NOT = BORROW

(4) IF IN DECIMAL MODE, Z FLAG IS INVALID
ACCUMULATOR MUST BE CHECKED FOR ZERO RESULT

X	INDEX X	+	ADD	M	MEMORY BIT 7
Y	INDEX Y	-	SUBTRACT	M6	MEMORY BIT 6
A	ACCUMULATOR	∧	AND	n	NO CYCLES
M	MEMORY PER EFFECTIVE ADDRESS	∨	OR	#	NO BYTES
Ms	MEMORY PER STACK POINTER	⊻	EXCLUSIVE OR		

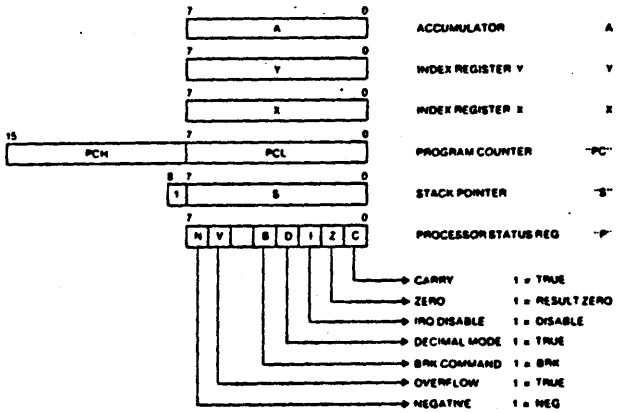
FORWARD RELATIVE BRANCH TABLE

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3		48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4		64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5		80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6		96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7		112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

BACKWARD RELATIVE BRANCH TABLE

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8		128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9		112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A		96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B		80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C		64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D		48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E		32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F		16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

PROCESSOR PROGRAMMING MODEL



MACHINE INSTRUCTIONS

ADC	Add Memory to Accumulator with Carry
AND	AND Memory with Accumulator
ASL	Shift Left One Bit (Memory or Accumulator)
BCC	Branch on Carry Clear
BCS	Branch on Carry Set
BEQ	Branch on Result Zero
BIT	Test Bits in Memory with Accumulator
BMI	Branch on Result Minus
BNE	Branch on Result Not Zero
BPL	Branch on Result Plus
BRK	Force Break
BVC	Branch on Overflow Clear
BVS	Branch on Overflow Set
CLC	Clear Carry Flag
CLD	Clear Decimal Mode
CLI	Clear Interrupt Disable Bit
CLV	Clear Overflow Flag
CMP	Compare Memory and Accumulator
CPX	Compare Memory and Index X
CPY	Compare Memory and Index Y
DEC	Decrement Memory by One
DEX	Decrement Index X by One
DEY	Decrement Index Y by One
EOR	Exclusive-OR Memory with Accumulator
INC	Increment Memory by One
INX	Increment Index X by One
INY	Increment Index Y by One
JMP	Jump to New Location
JSR	Jump to New Location Saving Return Address
LDA	Load Accumulator with Memory
LDX	Load Index X with Memory
LDY	Load Index Y with Memory
LSR	Shift Right One Bit (Memory or Accumulator)
NOP	No Operation
ORA	OR Memory with Accumulator
PHA	Push Accumulator on Stack
PHP	Push Processor Status on Stack
PLA	Pull Accumulator from Stack
PLP	Pull Processor Status from Stack
ROL	Rotate One Bit Left (Memory or Accumulator)
ROR	Rotate One Bit Right (Memory or Accumulator)
RTI	Return from Interrupt
RTS	Return from Subroutine
SBC	Subtract Memory from Accumulator with Borrow
SEC	Set Carry Flag
SED	Set Decimal Mode
SEI	Set Interrupt Disable Status
STA	Store Accumulator in Memory
STX	Store Index X in Memory
STY	Store Index Y in Memory
TAX	Transfer Accumulator to Index X
TAY	Transfer Accumulator to Index Y
TSX	Transfer Stack Pointer to Index X
TXA	Transfer Index X to Accumulator
TXS	Transfer Index X to Stack Pointer
TYA	Transfer Index Y to Accumulator

COMPARE INSTRUCTION RESULTS

Condition	N	Z	C
A, X, or Y < Memory	1*	0	0
A, X, or Y = Memory	0	1	1
A, X, or Y > Memory	0*	0	1

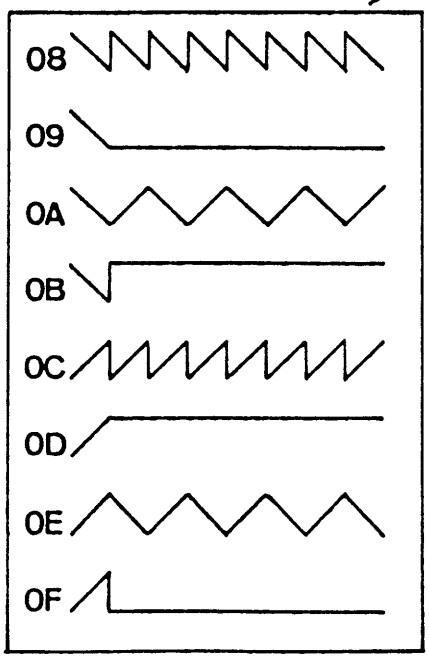
*N is valid only for 2's complement compare.

MUSIK-MODUL

h :	FD
c :	EF
cis:	E2
d :	D5
dis:	C9
e :	BD
f :	B3
fis:	A9
g :	9F
gis:	97
a :	8E
ais:	86
h :	7F
c :	78
cis:	71
d :	6A
dis:	64
e :	5F

REG. \ BIT	7	6	5	4	3	2	1	0
00 TONE A	fin tone							
01 TONE A	grov tone							
02 TONE B	fin tone							
03 TONE B	grov tone							
04 TONE C	fin tone							
05 TONE C	grov tone							
06 STØJ	frekvens							
07 MIXER	støj				tone			
	C			B	A	C	B	A
08 VOL. A	m=0: i3-io			m	i3	i2	i1	io
09 VOL. B	m=1: envel.			m	i3	i2	i1	io
0A VOL. C				m	i3	i2	i1	io
0B ENVELOPE TID	fin							
0C ENVELOPE TID	grov							
0D ENV. FORM								
0E								
0F								

En aktiv kanal skal have bit-en lav (0)



Periodetid	Grov_indst	Fin_indst
10 s	98	96
5 s	4C	4B
2 s	1E	85
1 s	0F	42
0,5 s	07	A1
0,2 s	03	0D
0,1 s	01	87
0,05 s	00	C3
0,02 s	00	4E
0,01 s	00	27

FACITLISTE

OPGAVE 1: bit 0 = 1, bit 1 = 2, bit 2 = 4, bit 3 = 8.

OPGAVE 2: 256 skulle skrives som 100 i hex. Det betyder jo, at der er 0 af værdien 1, 0 af værdien 16, 1 af værdien 16*16.

OPGAVE 3: 10 i otte-talsystemet svarer til 8 i ti-talsystemet.

OPGAVE 4: Der er en forskel på 32 (dec) = 20 (hex). Det er derfor bit 5 (2 = 32), der er forskellig - studer bitmønstret og husk, at bit-en helt til højre har nummer 0.

OPGAVE 5: Før der trykkes på "NED" viser de 5 cifre 10000. Efter der trykkes, viser de 0FFFF. Det er kun de fire sidste tal, der ses.

OPGAVE 6: Der afsættes 16 bytes (10 i hex) til hver tabel. Derfor kan der være 2048:16=128 tabeller. Hvis der som normalt blev afsat 10 bytes til hver tabel, kunne der være 204 tabeller. Tabellerne kan ikke indeholde tal, der er større end 255 (det højeste tal, der kan skrives med 8 bits).

OPGAVE 7: Prøv om du kan finde det rigtige indhold ved de forskellige adresser. Når der f.eks. forneden på side 86 står:

F800 A2EF betyder det, at der her findes to bytes (A2 og EF), som hører sammen. Det skal så forstås således:

F800 A2
F801 EF

OPGAVE 8: Den sidste del af programmet skal se således ud:

0209 8D
020A 02
020B 80

```
020C   8D
020D   0F
020E   80
020F   4C
0210   00
0211   02
```

OPGAVE 9: Når "SET" er høj, vil motoren ikke køre.

OPGAVE 10: Når "Fart" er høj, køres med halv fart.

OPGAVE 11: "Retn" afgør hvilken vej motoren kører rundt: når den er høj køres med uret.

OPGAVE 12: Når der ikke trykkes på en digitast, vil den tilhørende bit holdes lav ved hjælp af modstandene (R1-8). Hvis man ikke trykker, vil man derfor få: 0000 0000. Hvis man trykker på alle digitaster, fås: 1111 1111. Ved et tryk på en digitast, gøres den tilsvarende bit altså høj.

OPGAVE 13: Lyskurven kan styres således:

```
rødt/grønt     41
gult/gult      22
grønt/rødt     14
gult/gult      22
```

OPGAVE 14: Lyskurven kan styres således:

```
rødt/grønt     41
rødt/gult      42
rødt/rødt      22
rødt+gult/rødt 64
grønt/rødt     14
gult/rødt      24
rødt/rødt      44
rødt/rødt+gult 46
```

OPGAVE 15: Udlæs hex-tallene her til porten (med pause imellem):

```
00 - 01 - 02 - 04 - 08 - 10 - 20 - 40 - 80   (hex)
00 - 01 - 02 - 04 - 08 - 16 - 32 - 64 - 128  (dec)
```

OPGAVE 16: Indsæt op-koden 6A ved adresse 0309. Starttallet skal være 80 (hex), svarende til 128 (dec), hvilket får lysdioden ved bit 7 til at lyse.

OPGAVE 17: Prøv f.eks. at indlæse 77 i adresse 0301.

OPGAVE 18: Programmet skal se således ud:

```
0300  A980    LDA  Indlæs tallet 80(hex) i accum.
0302  18      CLC  Slet eventuel mente
0303  8D0F80  STA  Skriv accum. i adresse 800F
0306  2020FA  JSR  Spring til pause-program0
0309  6A      ROL  Ryk til venstre: gang tallet med 2
030A  8D0680  STA  Skriv til beep-er på adr 8006
030D  4C0303  JMP  Spring til adresse 0303
```

OPGAVE 19: Programmet skal se således ud:

```
0500  E600    INC  Forøg indholdet i adr. 00 med 1
0502  A500    LDA  Indlæs indholdet fra adr. 00
0504  8D0080  STA  Skriv acc. på display 0
0507  8D0F80  STA  Skriv acc. på porten
050A  2020FA  JSR  Spring til pause
050D  4C0005  JMP  Forfra
```

I adr. 00A1 og 00A3 indlægges f.eks. 10 som pauseværdier. Motoren vil køre hurtigst ved bit 0, med halv fart ved bit 1, kvart fart ved bit 2 osv.

OPGAVE 20: I adresse 0500 erstattes E6 med C6. Motorens omdrejningsretning afgøres af "Retn". Porten giver jo kun triggeimpulser til motoren, og de er ikke blevet ændret.

OPGAVE 21: Programmet skal nu se således ud:

```
0500  E600    INC  Forøg indholdet i adr. 00 med 1
0502  A500    LDA  Indlæs indholdet fra adr. 00
0504  8D0080  STA  Skriv acc. på display 0
0507  8D0680  STA  Skriv til beep-er
050A  2020FA  JSR  Spring til pause
050D  4C0005  JMP  Forfra
```

Adr. 00A1 kan indeholde FF og 00A3 kan indeholde 55.

OPGAVE 22: Programmet skal nu se således ud:

```
0600  F8      SED  Sæt decimal-mode. (ti-talsystem)
0601  38      CLC  Sæt menten før SEC
0602  E901    SEC  Træk tallet 01 fra accumulator.
0604  8D0080  STA  Skriv accumulatoren på display 0
0607  2020FA  JSR  Spring til pause-program ved FA20
060A  4C0206  JMP  Spring til 0602
```

OPGAVE 23: Den byte, der står efter off-set byten (FB) har nummer 0. Herfra tælles baglæns til byten ved adresse 0305 - der er 5 trin. det svarer til FB.

-5	-4	-3	-2	-1	0	off-set værdi
CD	04	80	F0	FB	8D	promgrambytes

OPGAVE 24: Adresse 0210 skal være 38 og adresse 0211 skal være ED.

OPGAVE 25: Adresse 0330 skal være 38 og adresse 0333 skal være E5.

OPGAVE 26: I stedet for 95 i adresse 0204 skal bruges 9D. Denne op-kode skal efterfølges af 2 bytes, så programmet bliver 1 byte længere. Derfor skal off-set værdien F9 i adresse 020Å ændres til F8 (samtidig kommer den til at ligge i adresse 020B). Endelig ændres det tal, der skal skrives fra 00 til AA - det sker i adresse 0201. Så ser programmet således ud:

```
0200  A9AA    LDA  Indlæs tallet AA i acc.
0202  A200    LDX  Indlæs tallet 00 i x-reg.
0204  9D0006  STA  Skriv acc. ved adresse 0600+X
0207  E8      INX  Forøg x-reg. med 1
0208  E010    CPX  Sammenlign x-reg. med tallet 10
020A  D0F8    BNE  Gå til 0204 hvis de er forskellige
020C  8D0680  STA  Skriv til beep-er
020F  4C00F8  JMP  Spring til styresystemet
```

OPGAVE 27: Adresse 0324 skal ændres til 6A, som er den værdi, der giver tonen d - se skemaet side 58.

OPGAVE 28: Opstillingen skal nu spille de første toner af "Tyv, ja tyv, det skal du hedde".

OPGAVE 29: Der er 80 kombinationsmuligheder ud fra skemaerne, så klem du bare på.

OPGAVE 30: "Envelope_tid" ændres i adresse 0340 (fin) og i adresse 034A (grov), mens "envelope_form" ændres i adresse 0354. Der er atter (mindst) 80 muligheder.

OPGAVE 31: Hvis du i adresse 0057 skriver FE i stedet for F8, får du kun toner ud i kanal A. Så er det bedst at skrue ned for volumen i kanal B og C. Det gøres ved at give adresse 0059 og 005A værdien 00.

OPGAVE 32: Hvis du vil lave støj i kanal A, skal adresse 0057 have værdien F7. Et godt brag fås ved, at vælge 09 som "envelope_form" (adresse 005D = 09) og "envelope_tid" som 2 sekunder (0058 = 85 og 0059 = 1E).

OPGAVE 33: Her skal bit 4 (m) være lav. Volumen varieres derfor med tal mellem 00 og 0F.

OPGAVE 34: Husk når du retter programmet, at "slet byte" og "indsæt byte" ikke virker her på side 0 - du skal derfor skrive oveni. SED (F8) får CPU-en til at køre i decimal mode og CLD (D8) får den til at holde op igen. Imidlertid virker decimal mode kun på ADC og SBC - ikke på INC. Det er en god skik at efterlade CPU-en i samme stand, som da man overtog den - derfor indføjes også CLD:

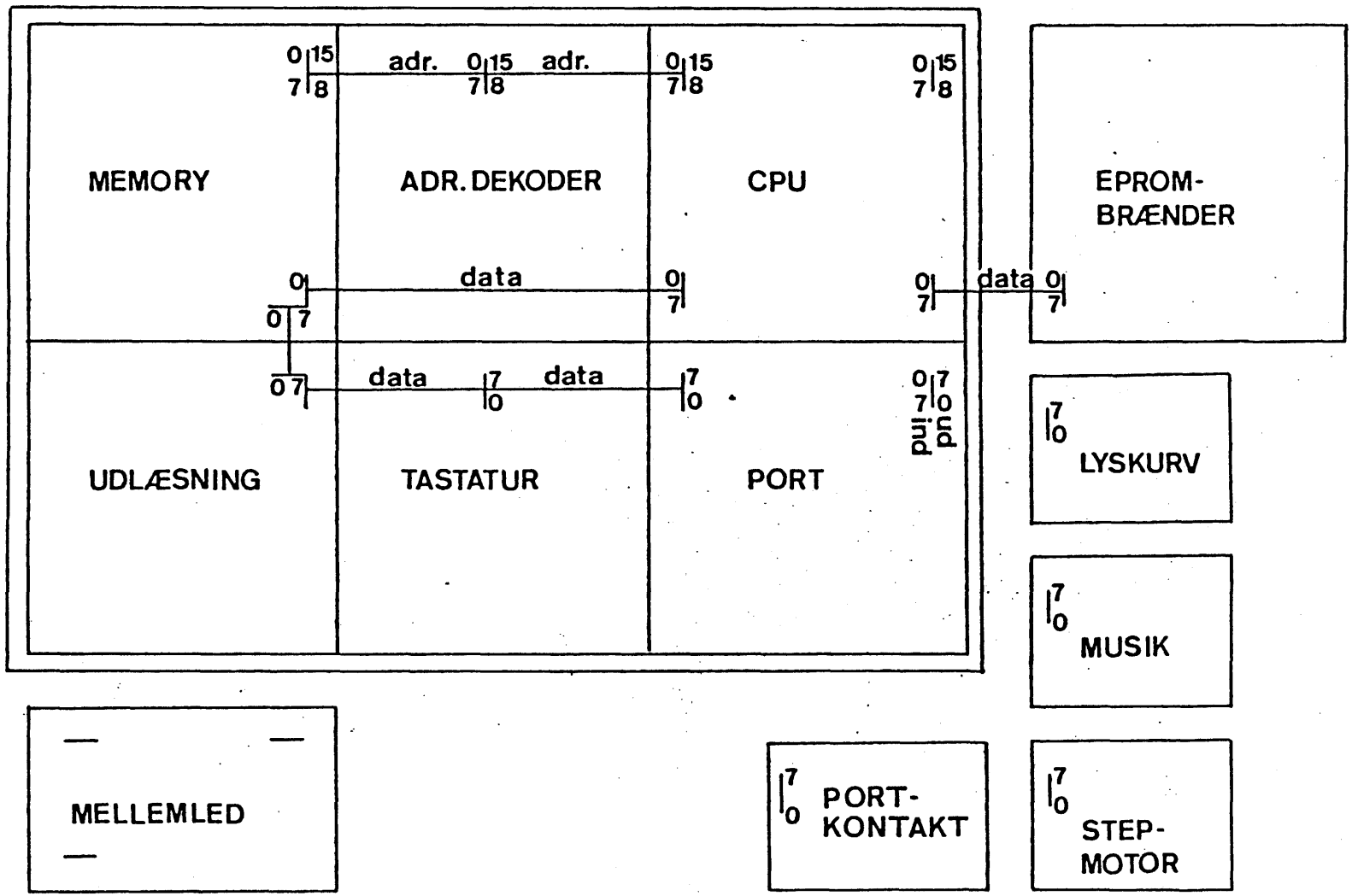
```
00B0  8500    STA  Gem acc. på adr. 0000
00B2  F8      SED  Kør i decimal mode
00B3  A501    LDA  Indlæs tal fra adresse 0001
00B5  8D0080  STA  Skriv acc. på display 0
00B8  8D0680  STA  Skriv til adr. 8006 - beep
00BB  18      CLC  Slet menten
00BC  6901    ADC  Læg tallet 1 til acc.
00BE  8501    STA  Gem acc. på adr. 0001
00C0  D8      CLD  Fjern decimal mode
00C1  2020FA  JSR  Spring til pause
00C4  A500    LDA  Indlæs tal fra adresse 0000 igen
00C6  40      RTI  Returner fra interrupt
```

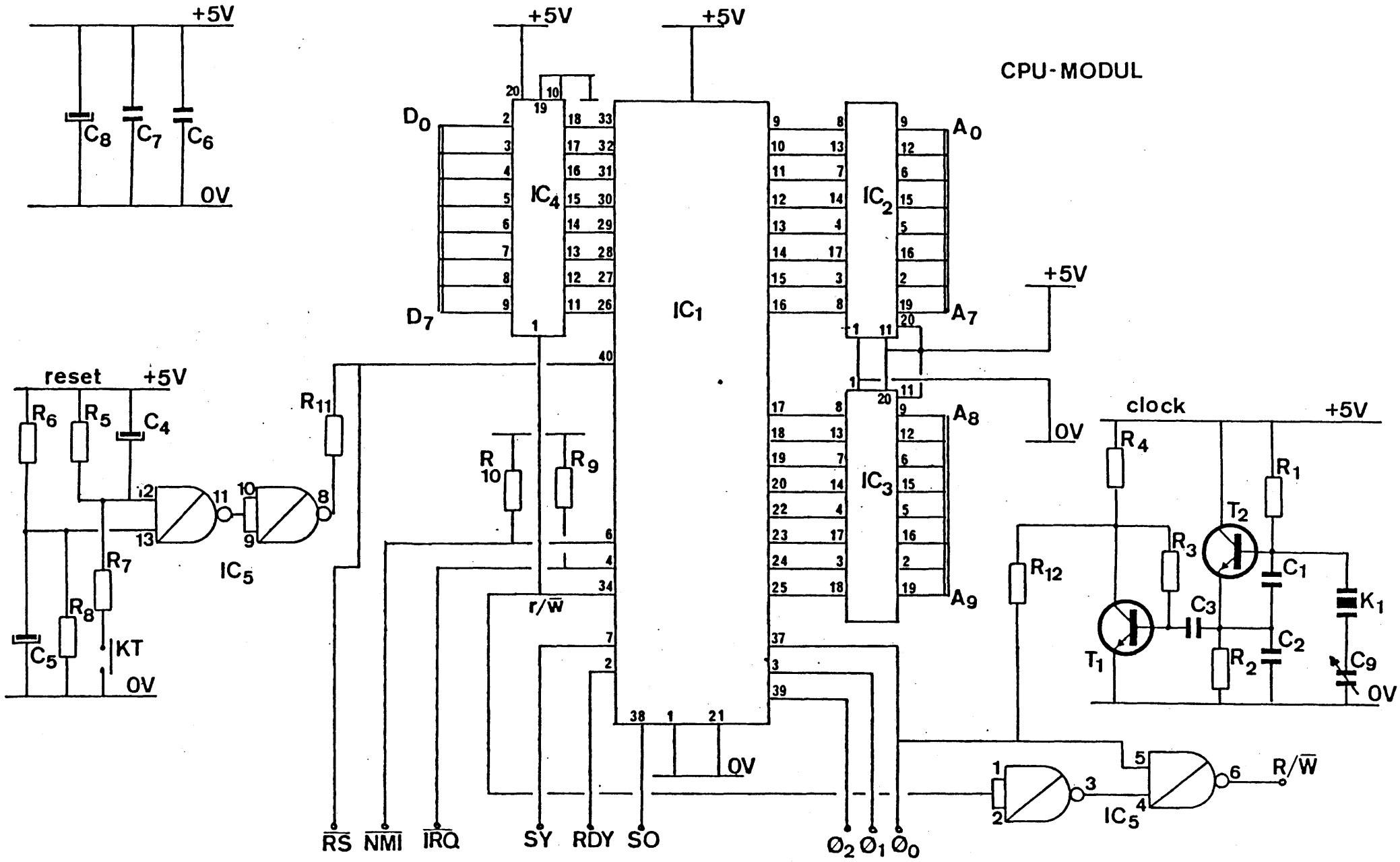
OPGAVE 35: Forbind stepmotoren som vist på side 27 og lav adresse 00B6 i programmet om, så linien hedder:

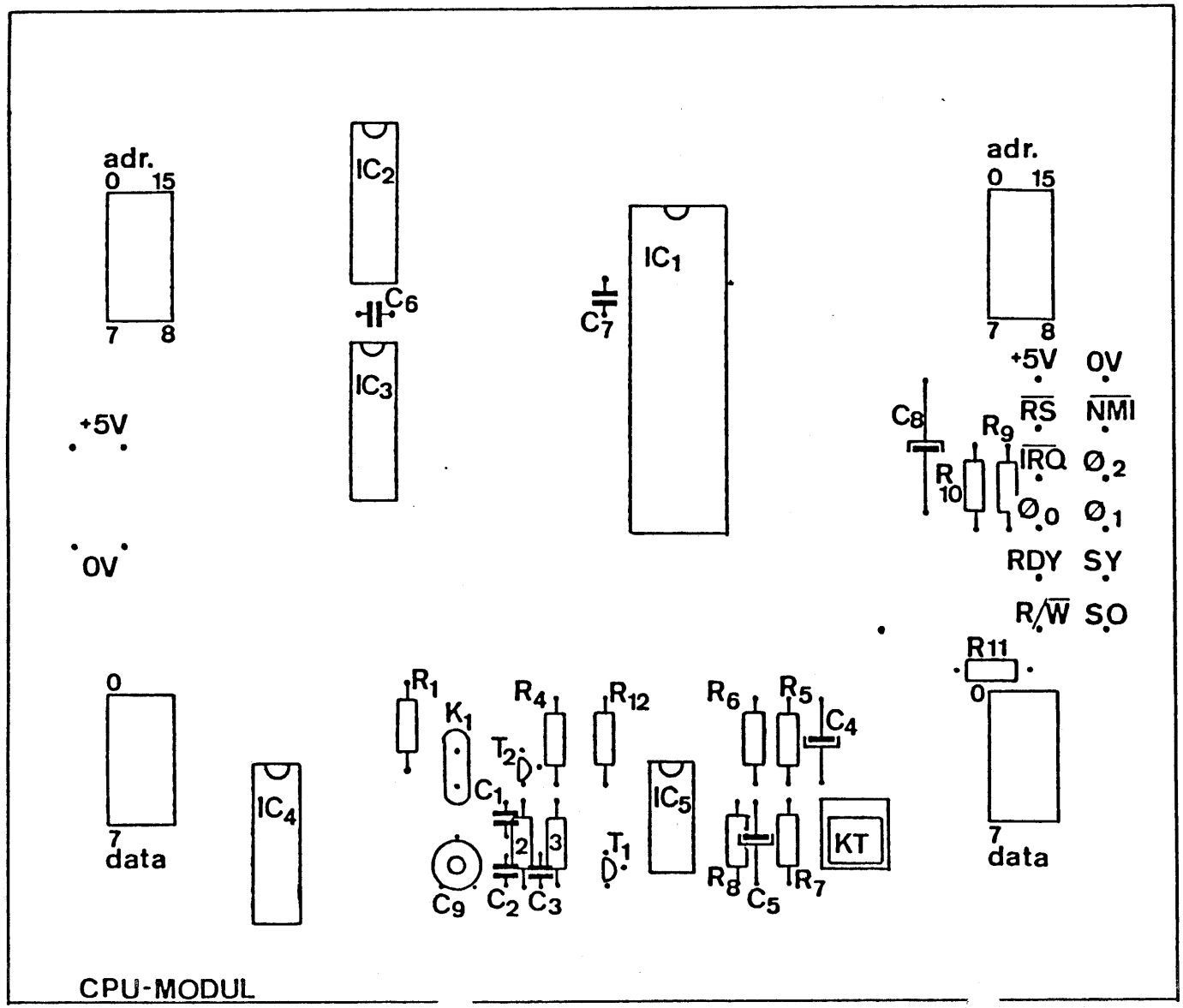
```
00B5  8D0F80  STA  Skriv til porten
```

Nu vil stepmotoren ændre kørselsmåde, når der kommer et interrupt. Der er selvfølgelig mange andre muligheder for programmer.

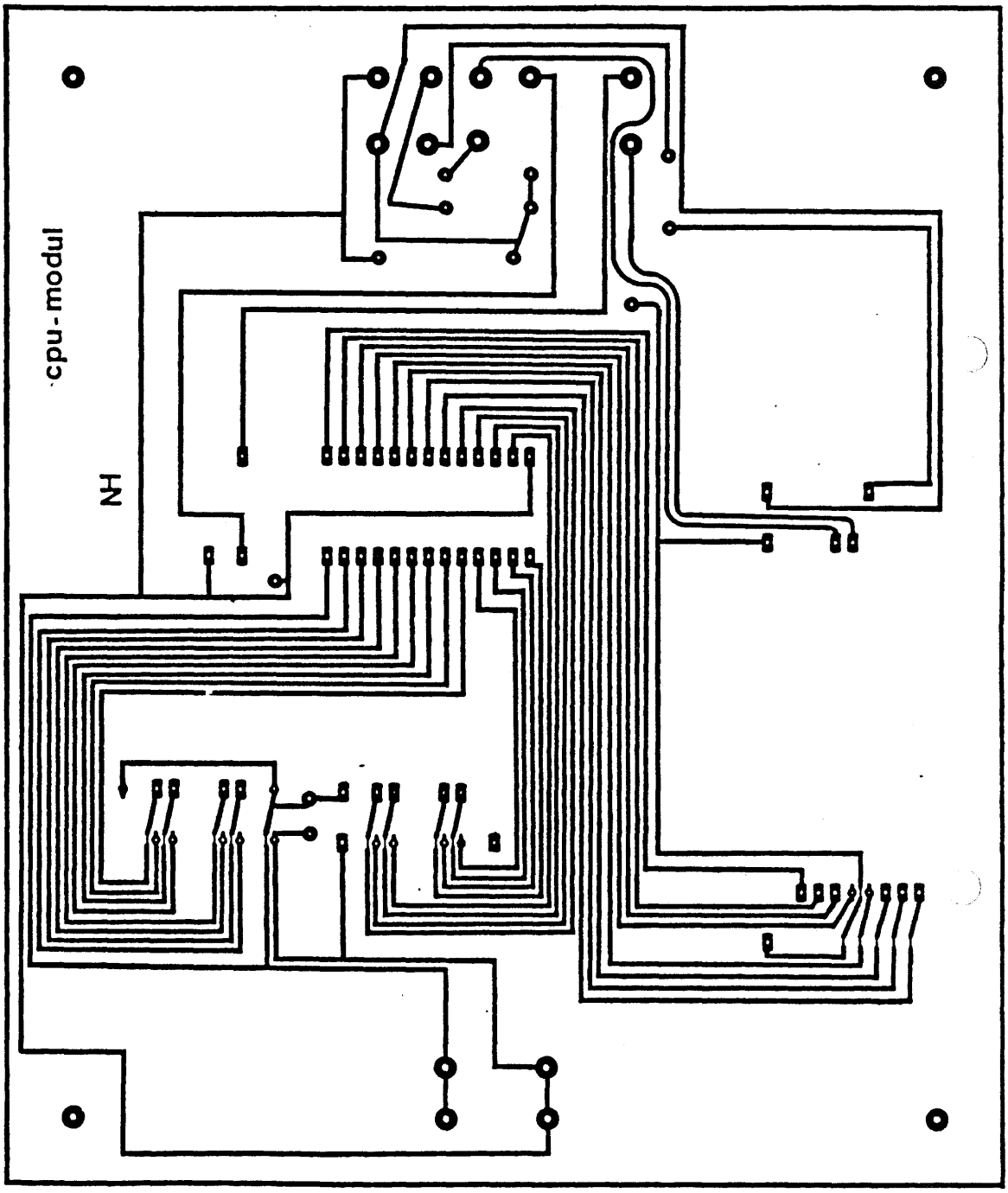
OPGAVE 36: Adresse 0200: 58 skal ændres til 78, som er op-koden for SEI.

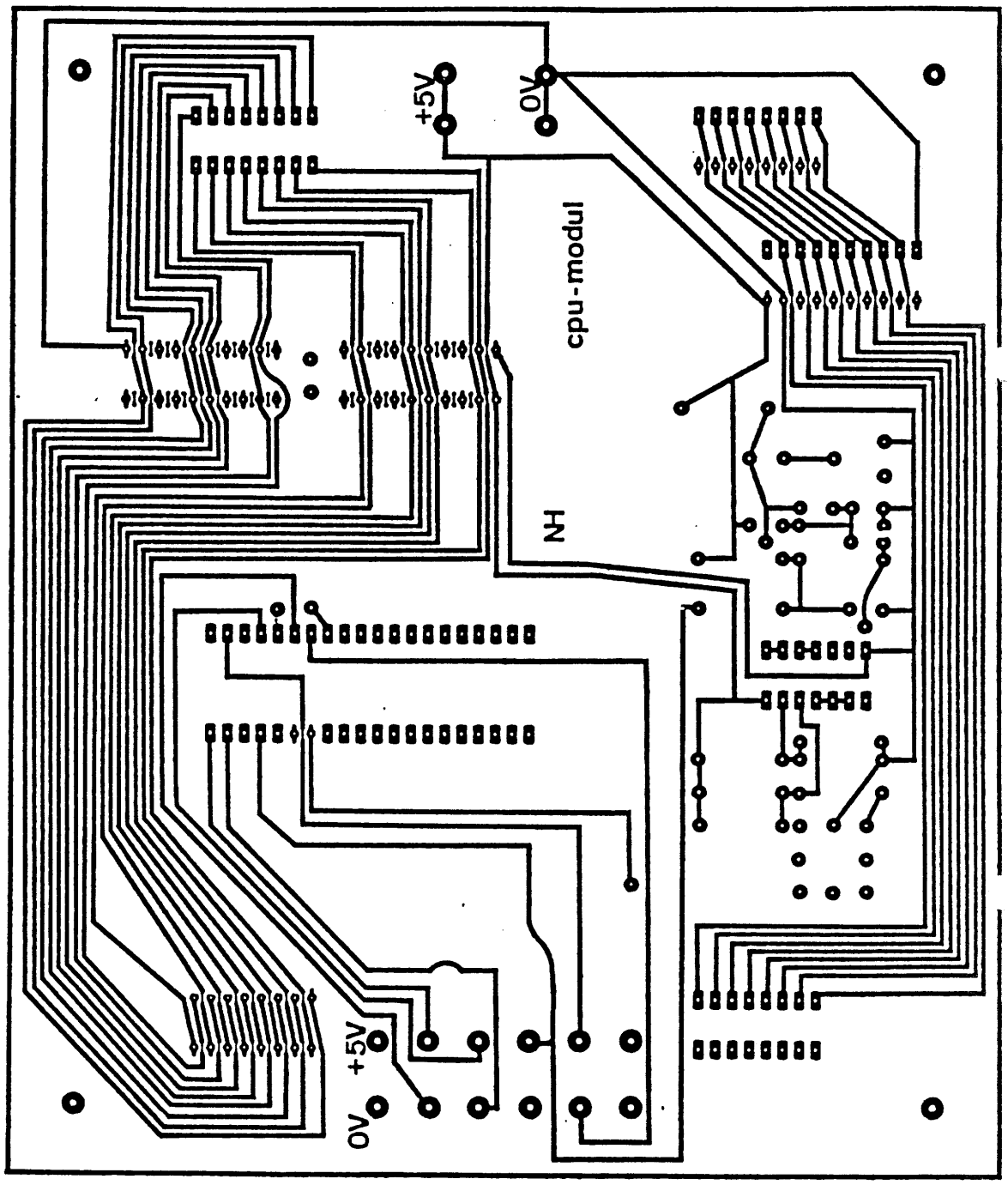

MEMORY
ADR. DEKODER
CPU
adr.
adr.
data
data
data
UDLÆSNING
TASTATUR
PORT
EPROM-
BRÆNDER
LYSKURV
MUSIK
STEP-
MOTOR
PORT-
KONTAKT
MELLEMLED


CPU-MODUL
+5V
C8
C7
C6
0V
D0
D7
IC4
IC1
IC2
IC3
A0
A7
A8
A9
reset
R6
R5
C4
R11
R10
R9
R7
R8
C5
KT
IC5
r/w
clock
R4
R1
T2
R3
C1
R12
C3
C2
K1
T1
R2
C9
RS
NMI
IRQ
SY
RDY
SO
Ø2 Ø1 Ø0
R/W

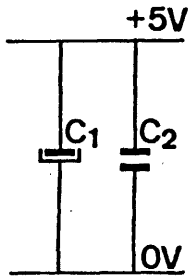

adr.
0 15
7 8
IC2
C6
IC3
IC1
C7
adr.
0 15
7 8
+5V
0V
C8
R9
R10
RS
NMI
IRQ
Ø2
Ø0
Ø1
RDY
SY
R/W
SO
R11
0
7
data
+5V
0V
0
7
data
IC4
R1
K1
C1
T2
R4
R12
IC5
T1
C9
C2
C3
2
3
R6
R5
C4
R8
C5
R7
KT

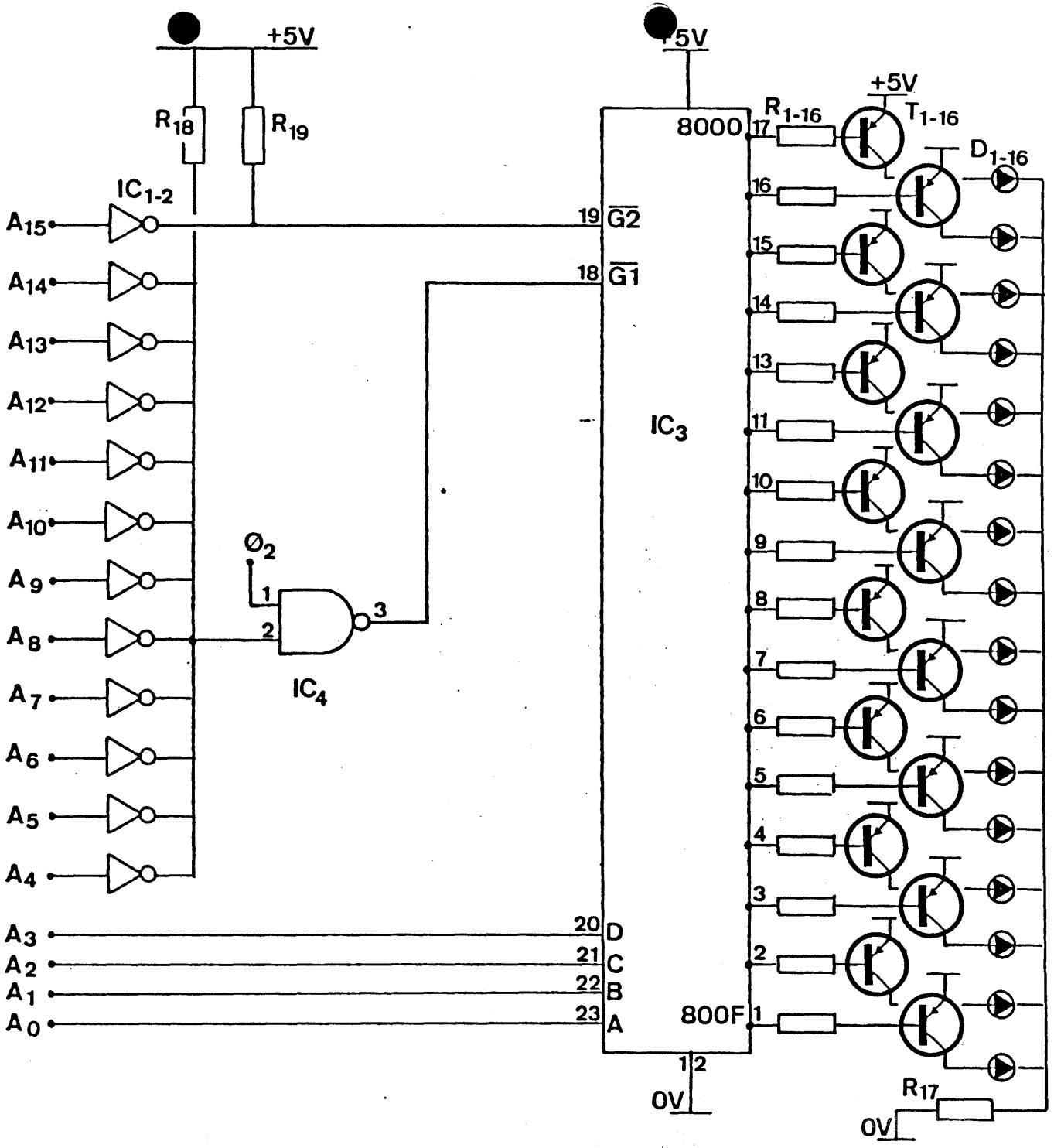
CPU-MODUL

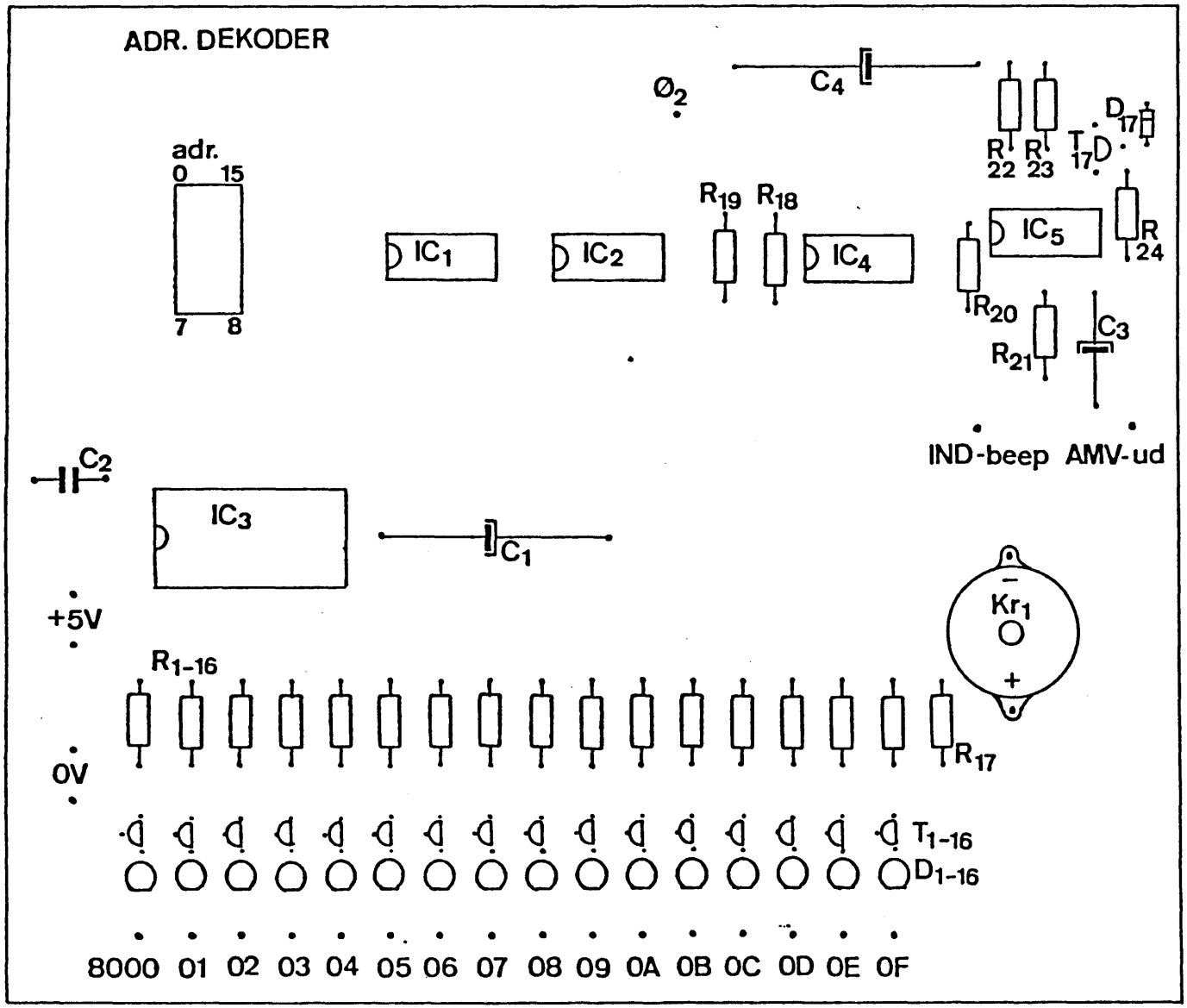

cpu-modul
NH

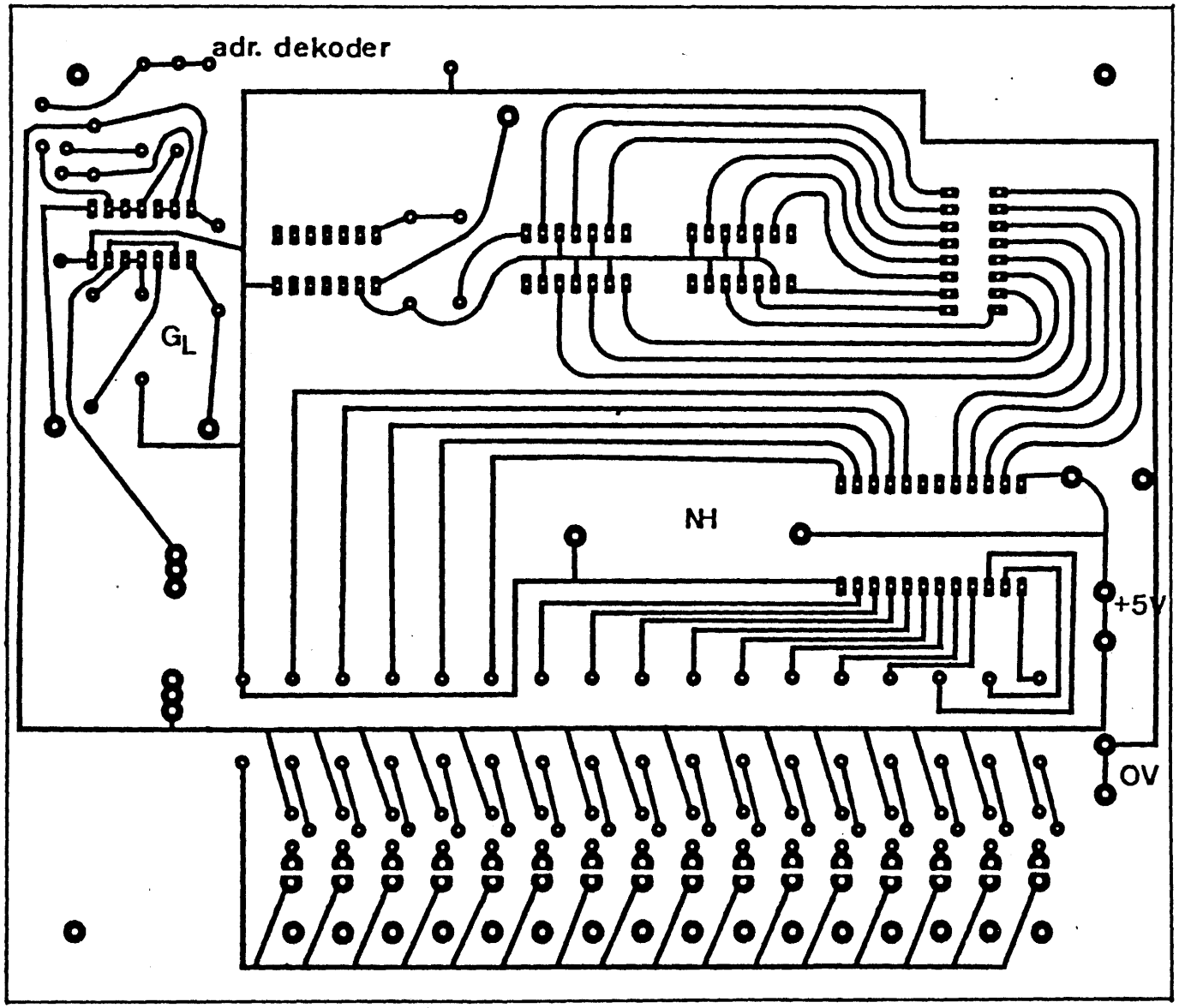

+5V
0V
cpu-modul
NH
0V
+5V

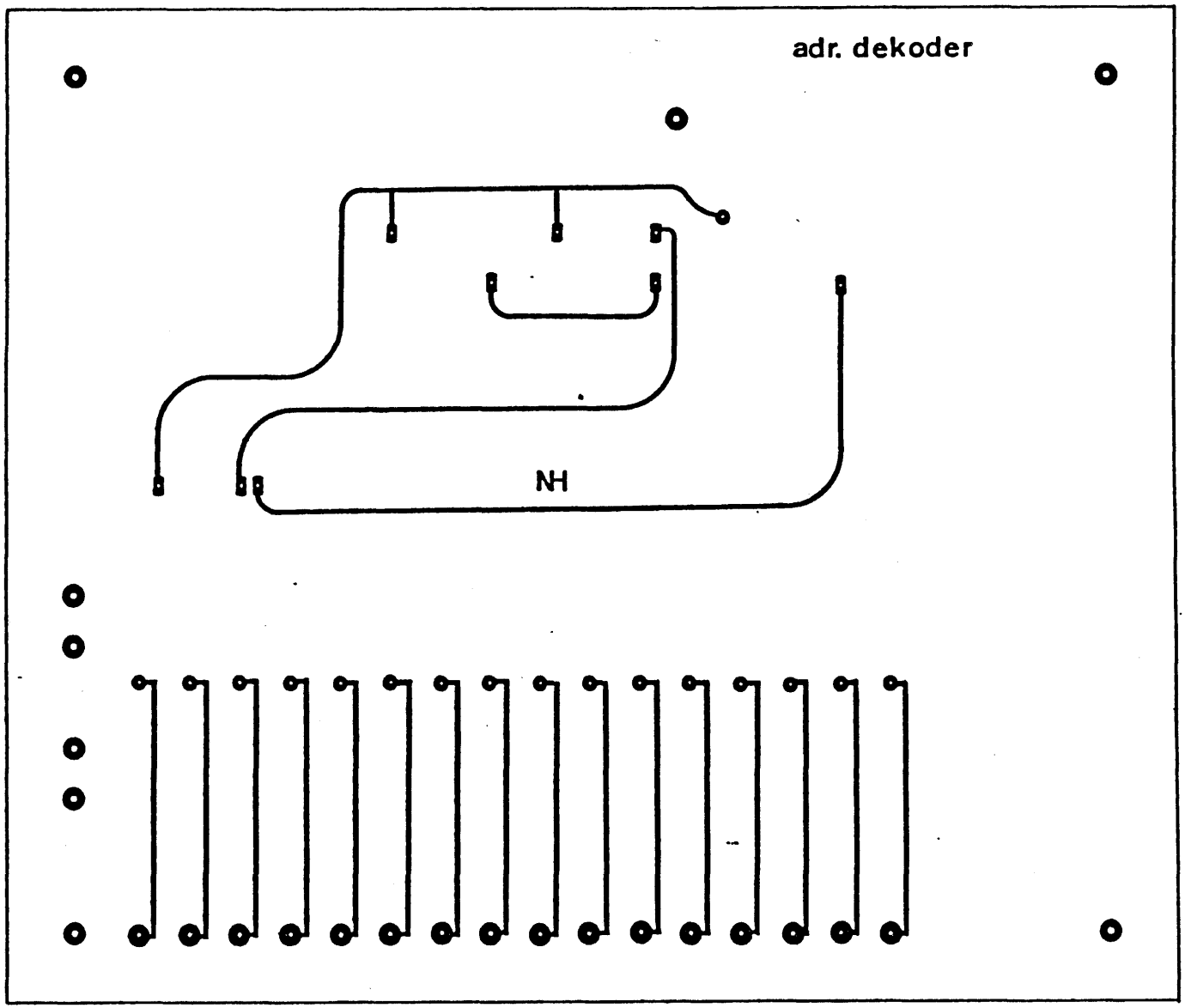
ADRESSEDEKODER



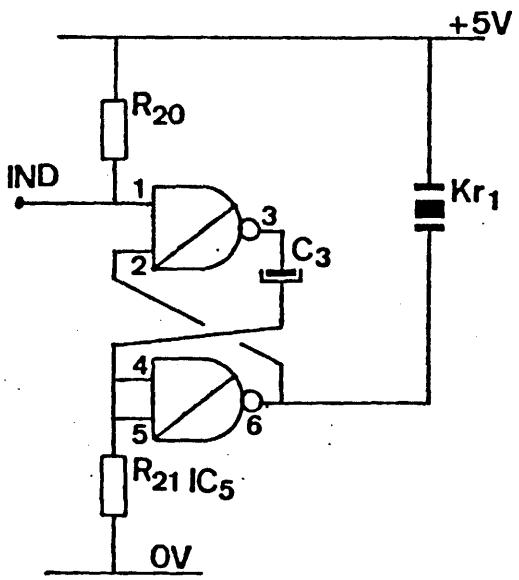



ADR. DEKODER
Ø2
C4
D17
R22
R23
T17
adr.
0 15
7 8
IC1
IC2
R19
R18
IC4
IC5
R24
R20
R21
C3
IND-beep
AMV-ud
C2
IC3
C1
+5V
Kr1
R1-16
0V
R17
T1-16
D1-16
8000 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

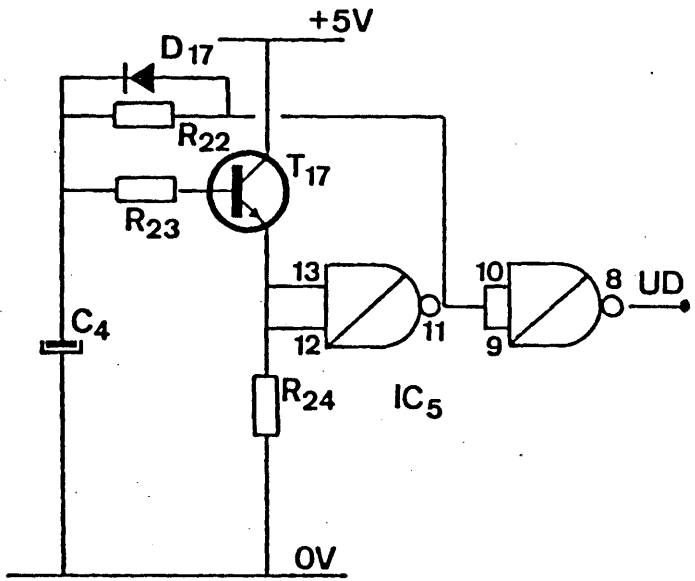

adr. dekoder
GL
NH
+5V
OV


adr. dekoder
NH

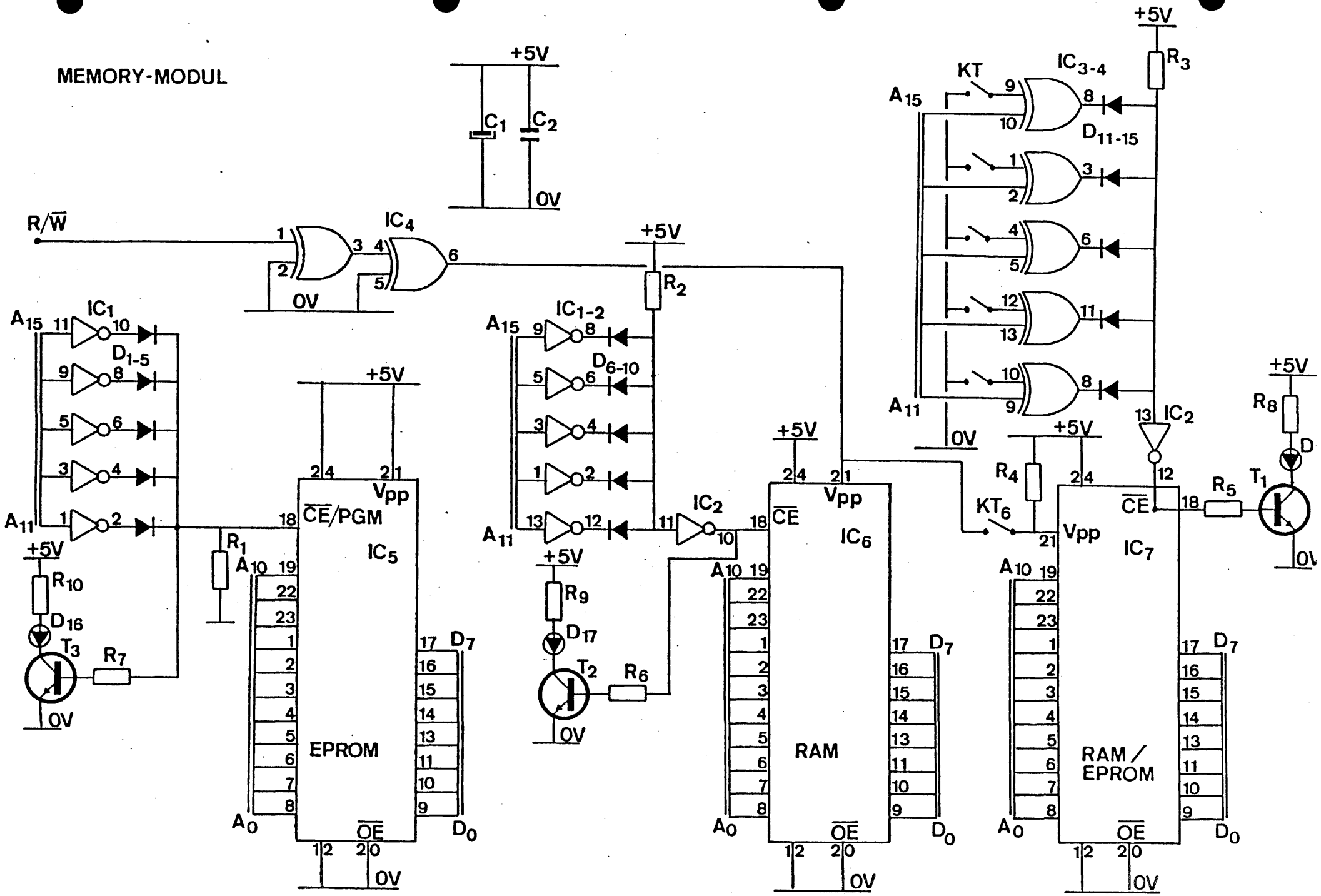
BEEPER

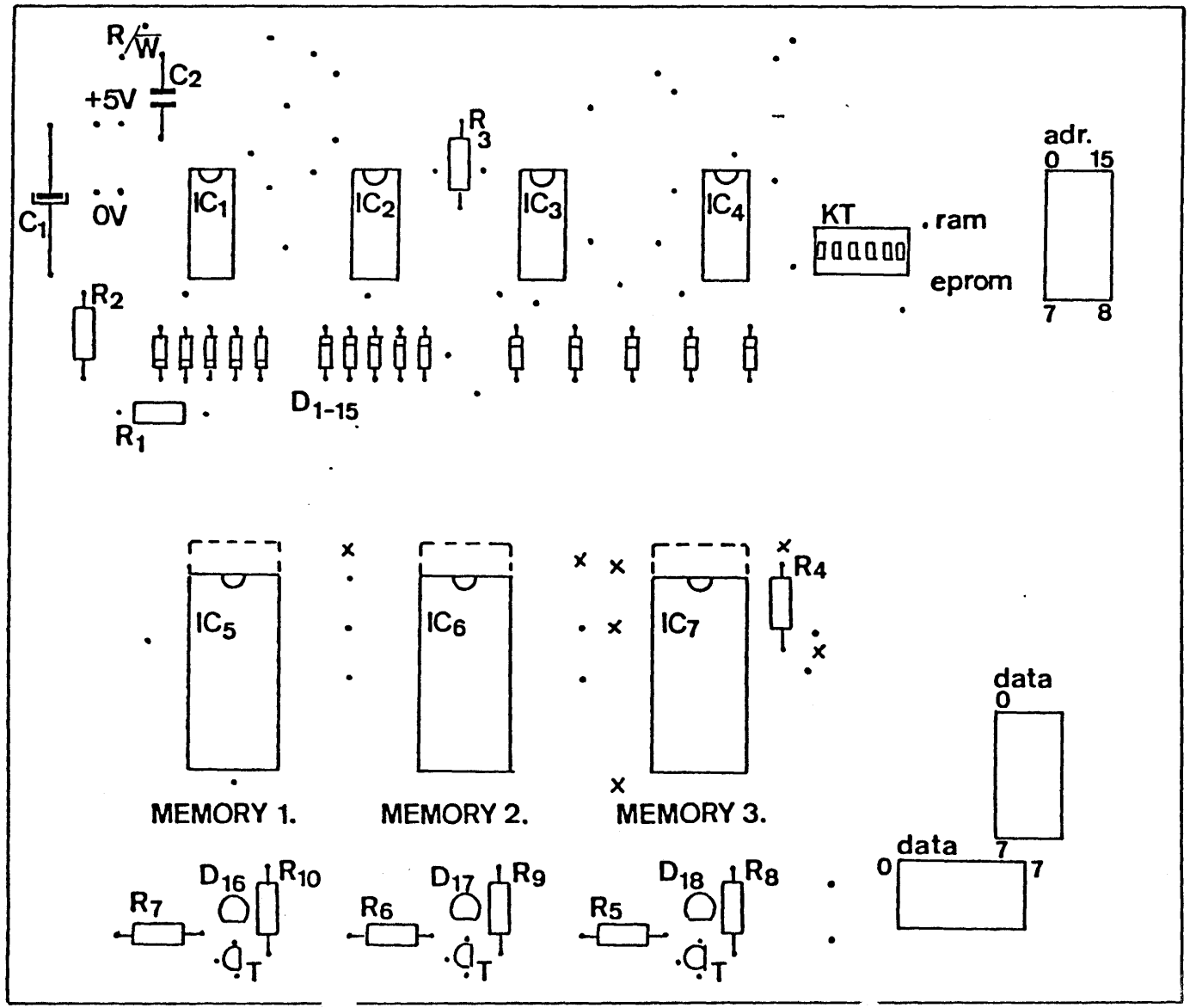

+5V
R20
IND
1
3
C3
2
Kr1
4
5
6
R21 IC5
0V

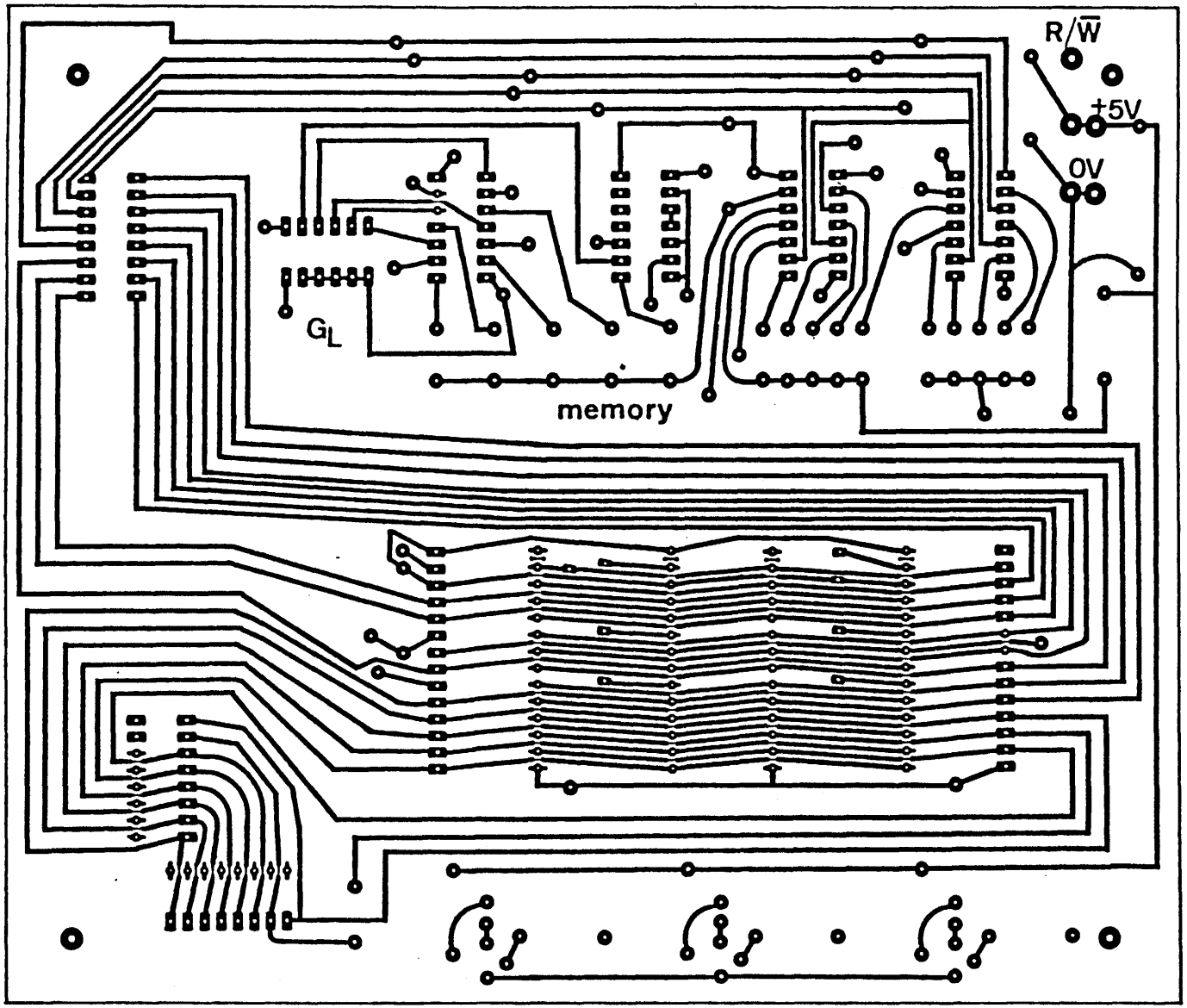
AMV

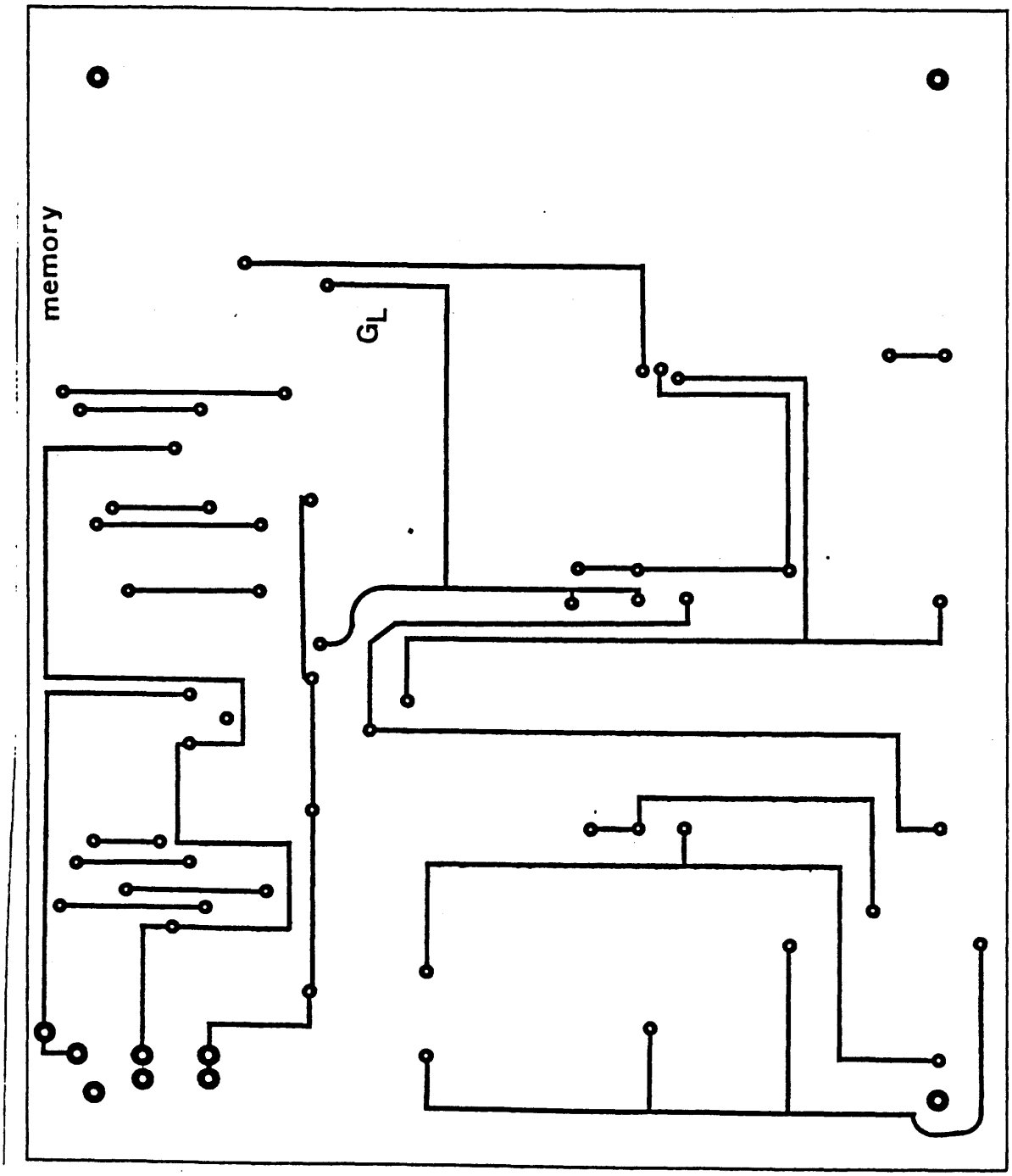

+5V
D17
R22
T17
R23
13
10
8
UD
C4
12
11
9
R24
IC5
0V

MEMORY-MODUL

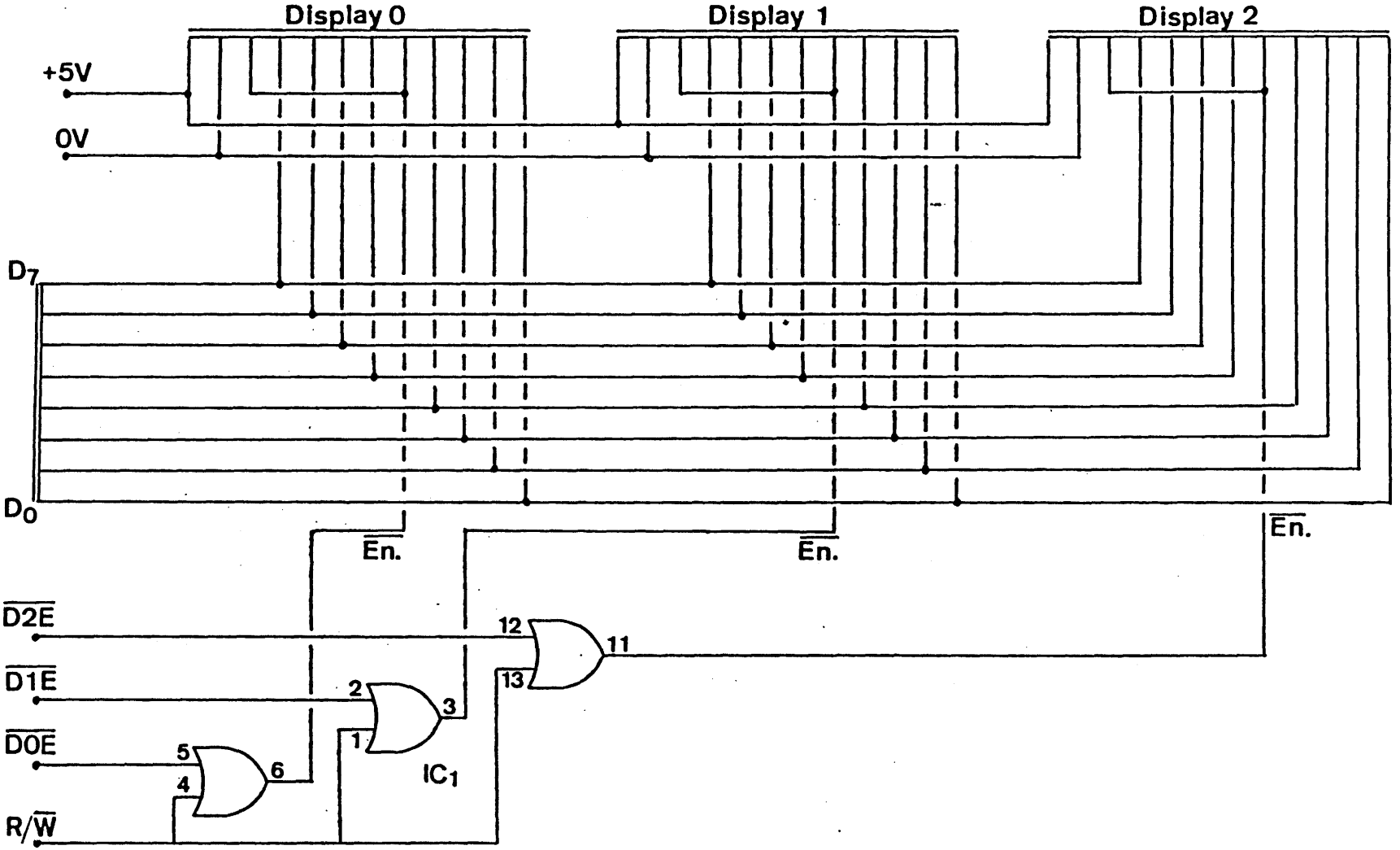


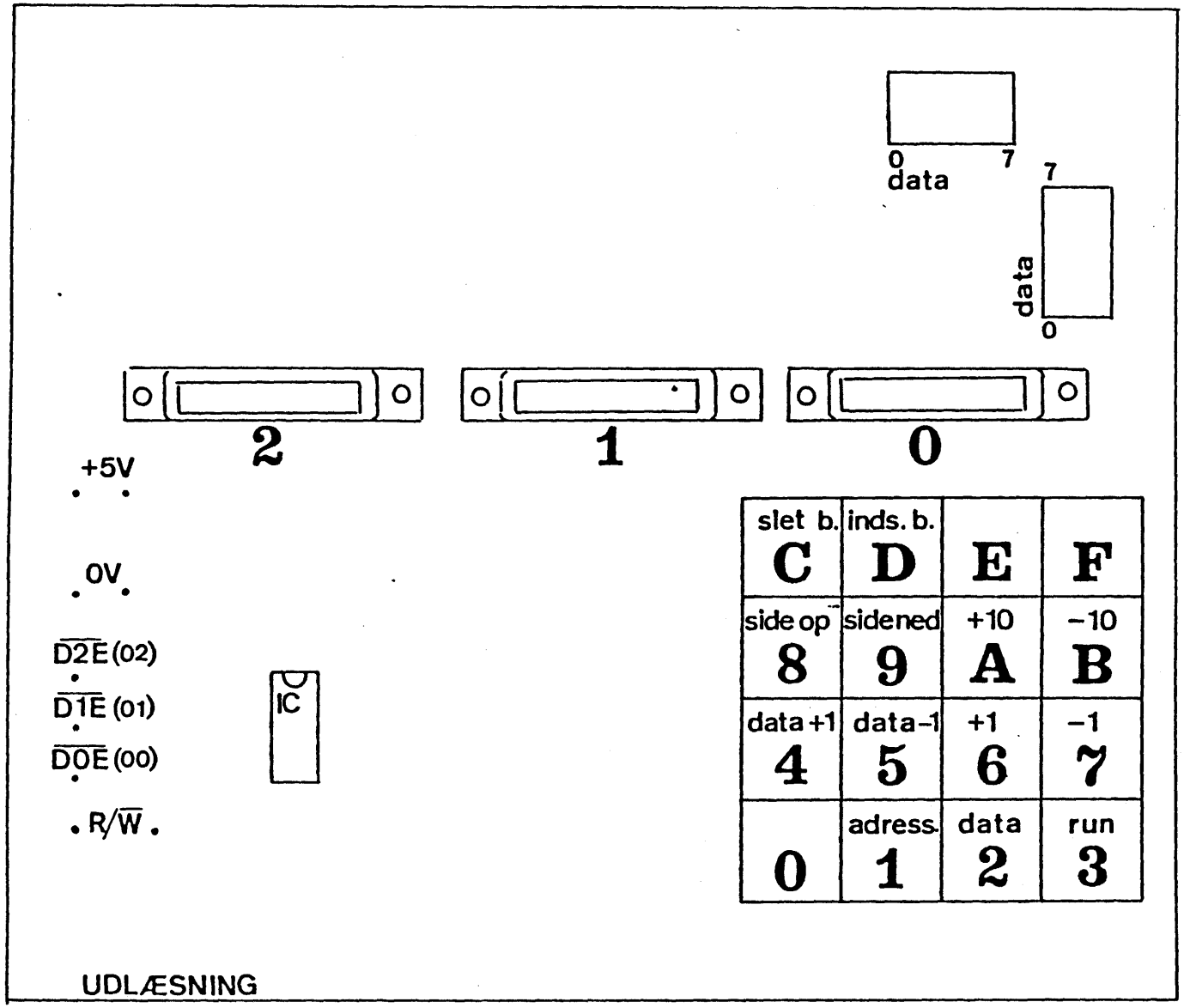

R/W
C2
+5V
0V
C1
IC1
IC2
R3
IC3
IC4
KT
ram
eprom
adr.
0 15
7 8
R2
R1
D1-15
IC5
IC6
IC7
R4
MEMORY 1.
MEMORY 2.
MEMORY 3.
data
0
data
7
0 7
D16
R10
R7
D17
R9
R6
D18
R8
R5
T
T
T


R/W
+5V
0V
GL
memory

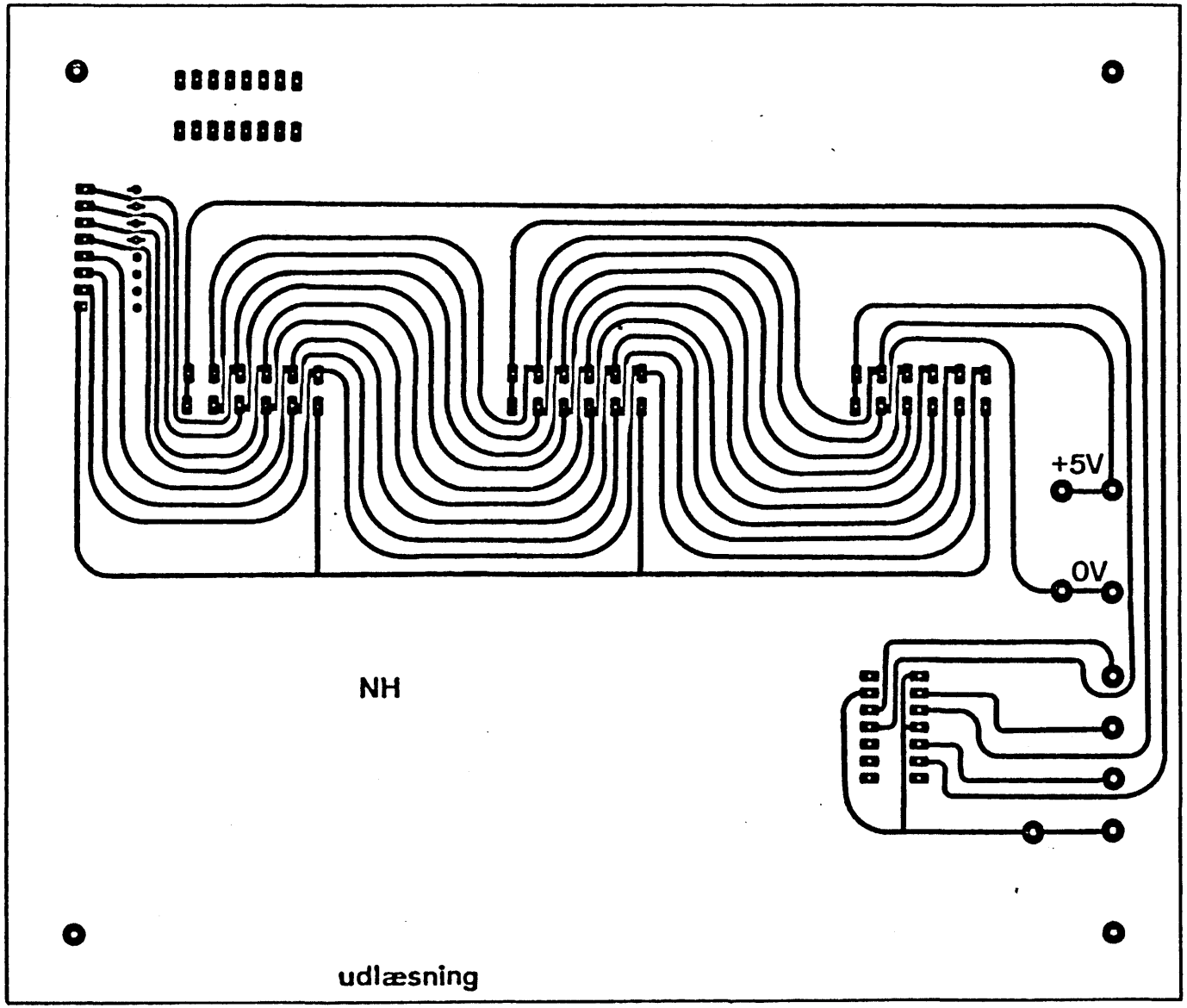

memory
G_L

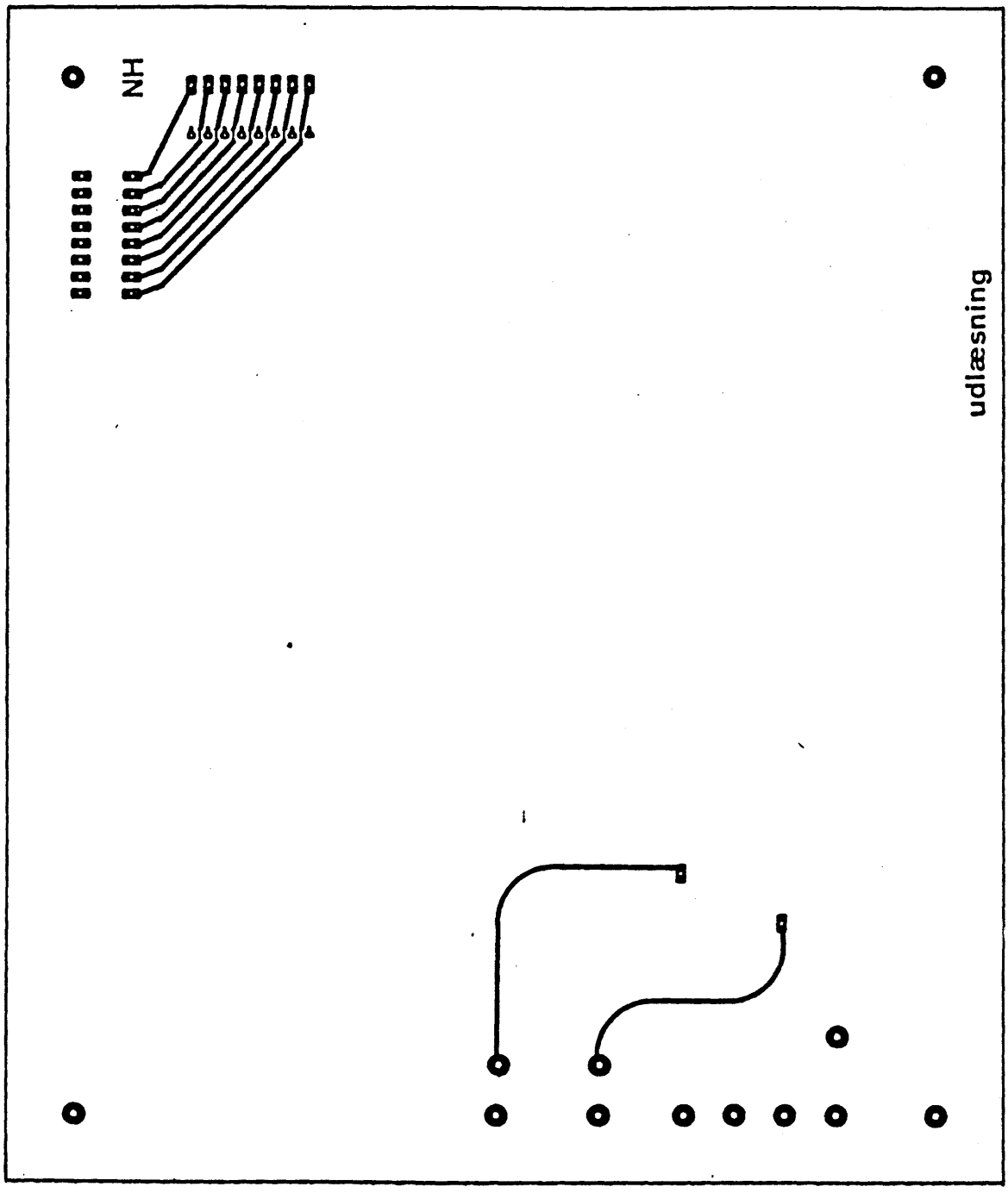
UDLÆSNINGSMODUL



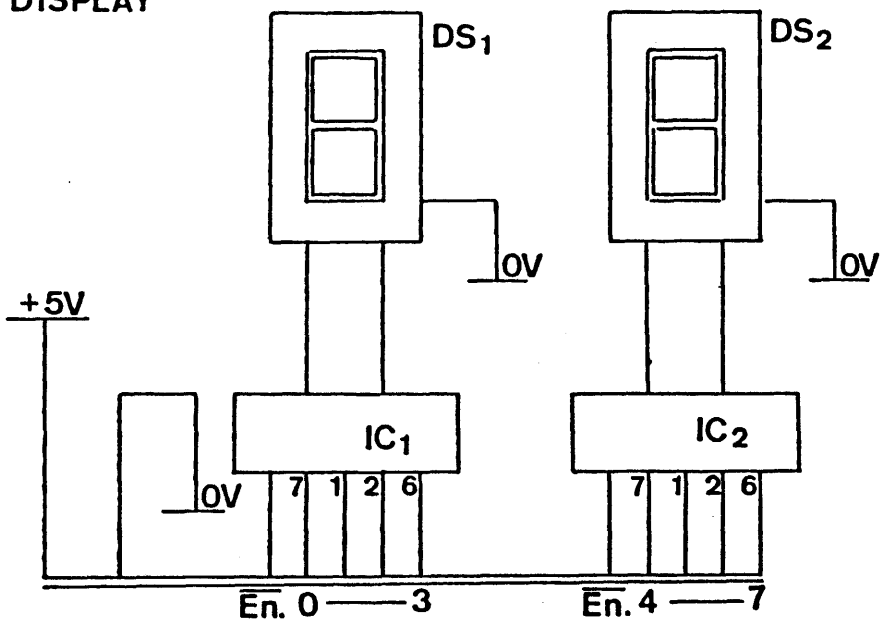

0
7
data
7
data
0
2
1
0
+5V
0V
D2E (02)
D1E (01)
D0E (00)
R/W
IC
slet b.
C
inds. b.
D
E
F
side op
8
sidened
9
+10
A
−10
B
data +1
4
data −1
5
+1
6
−1
7
0
adress
1
data
2
run
3

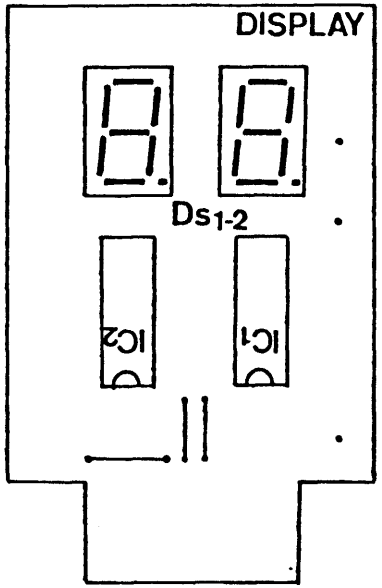
UDLÆSNING

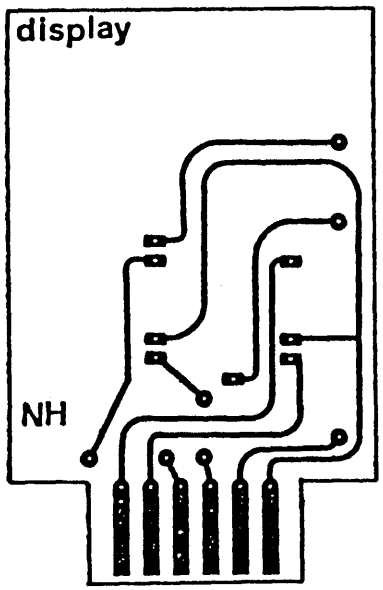

+5V
0V
NH
udlæsning

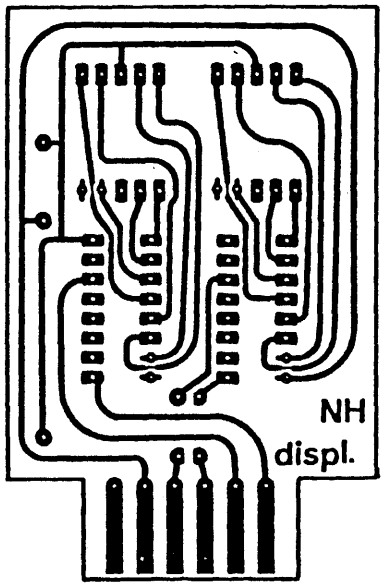

NH
udlæsning

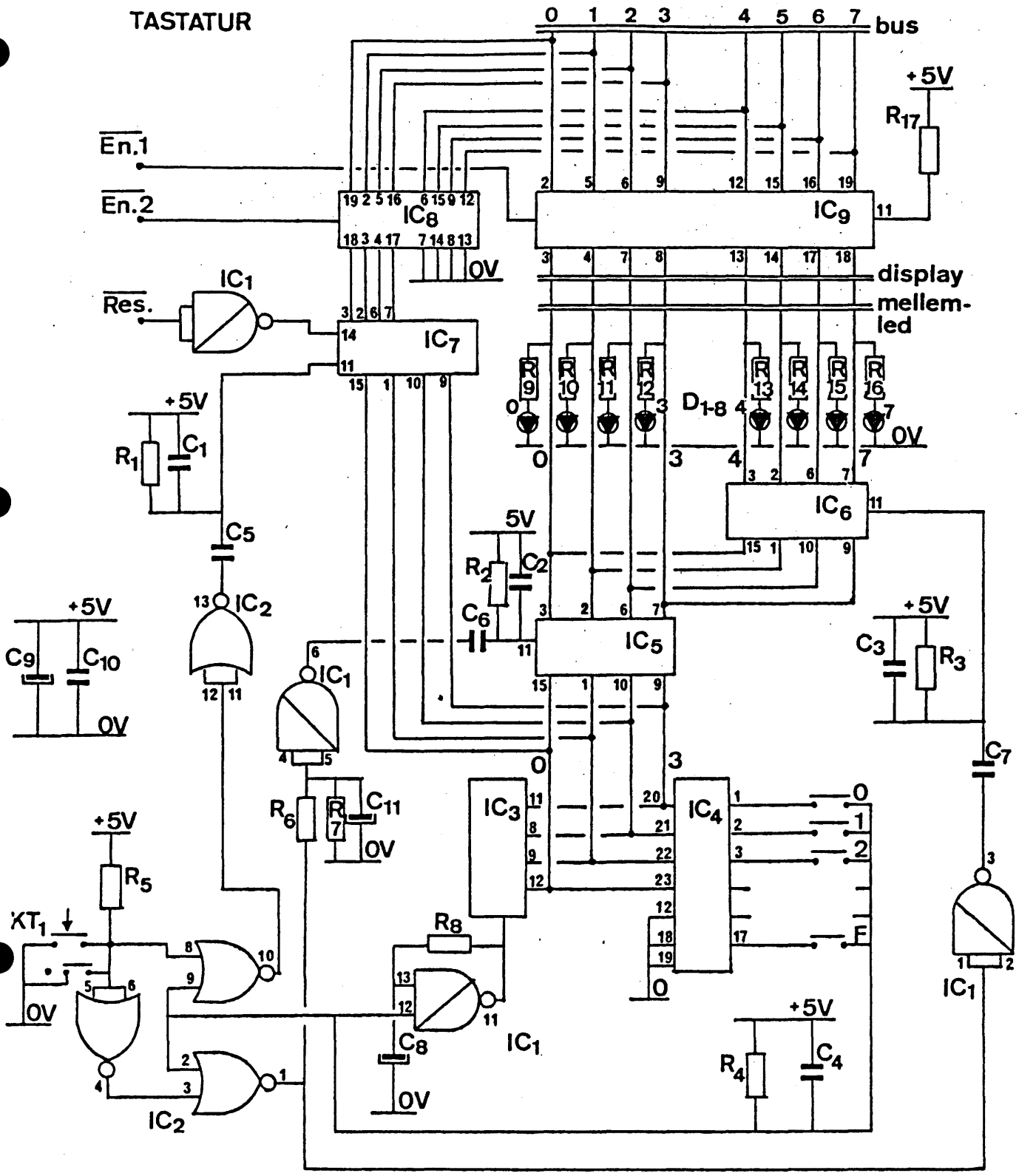
DISPLAY

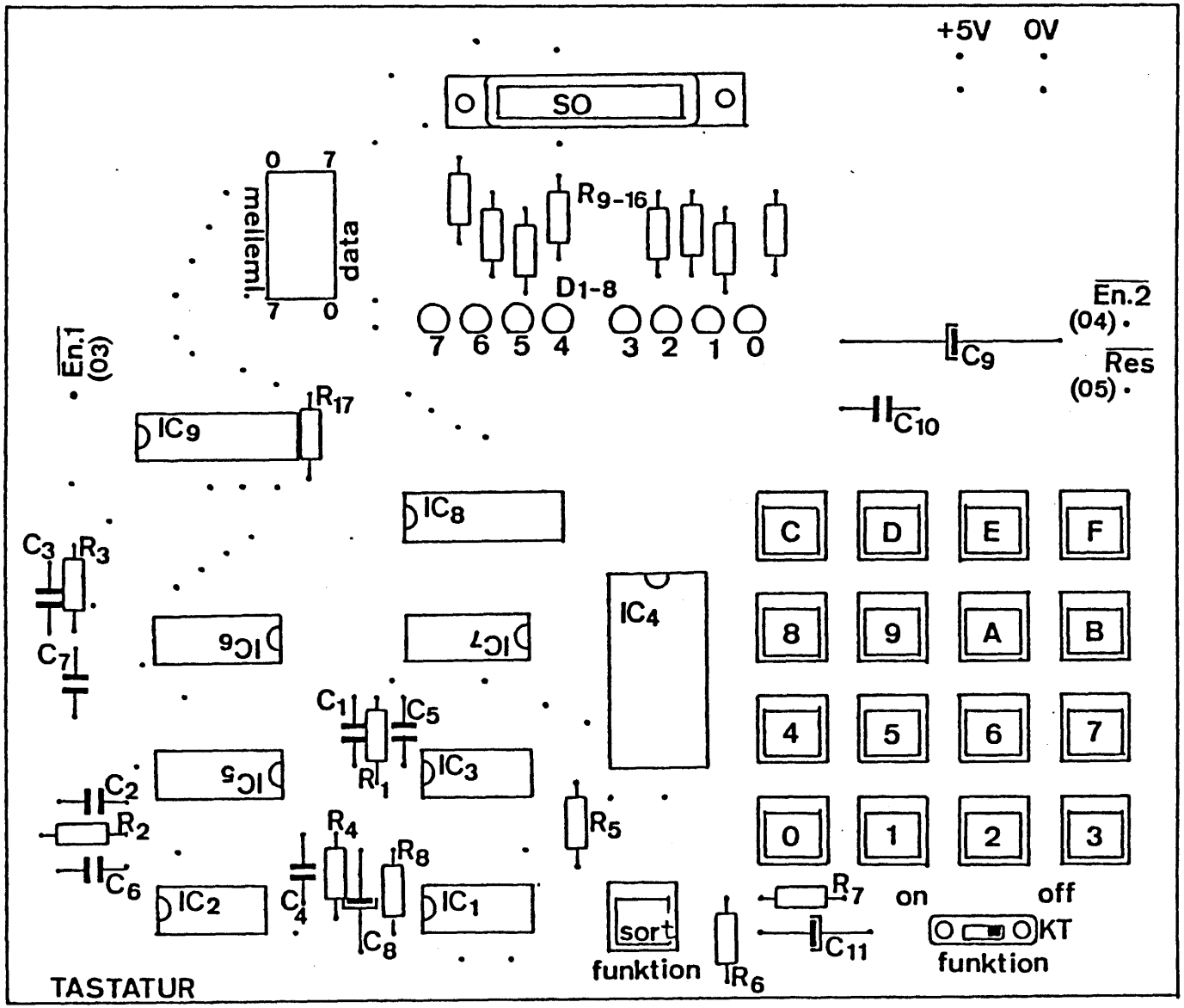


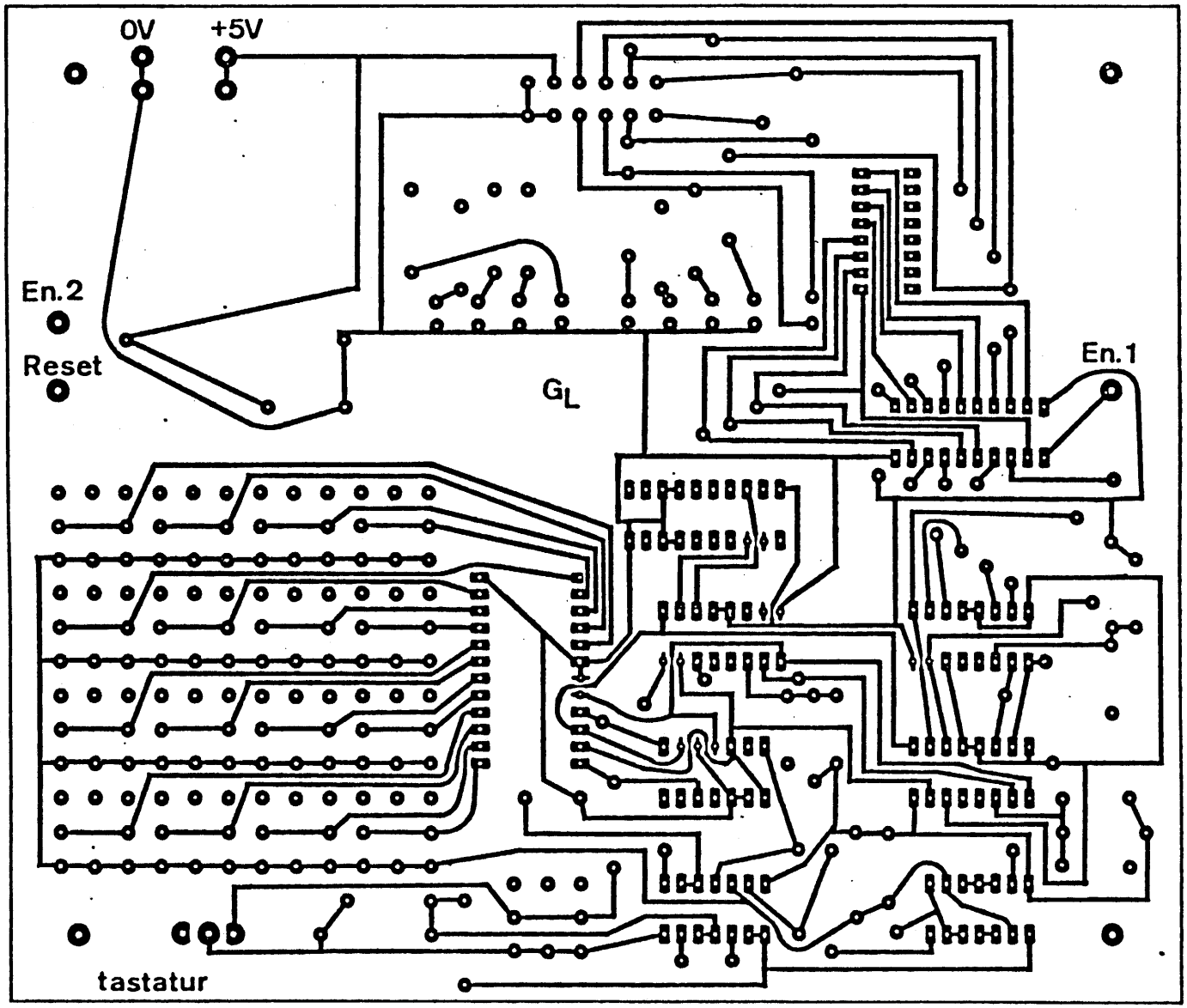

DISPLAY
Ds1-2
IC2
IC1

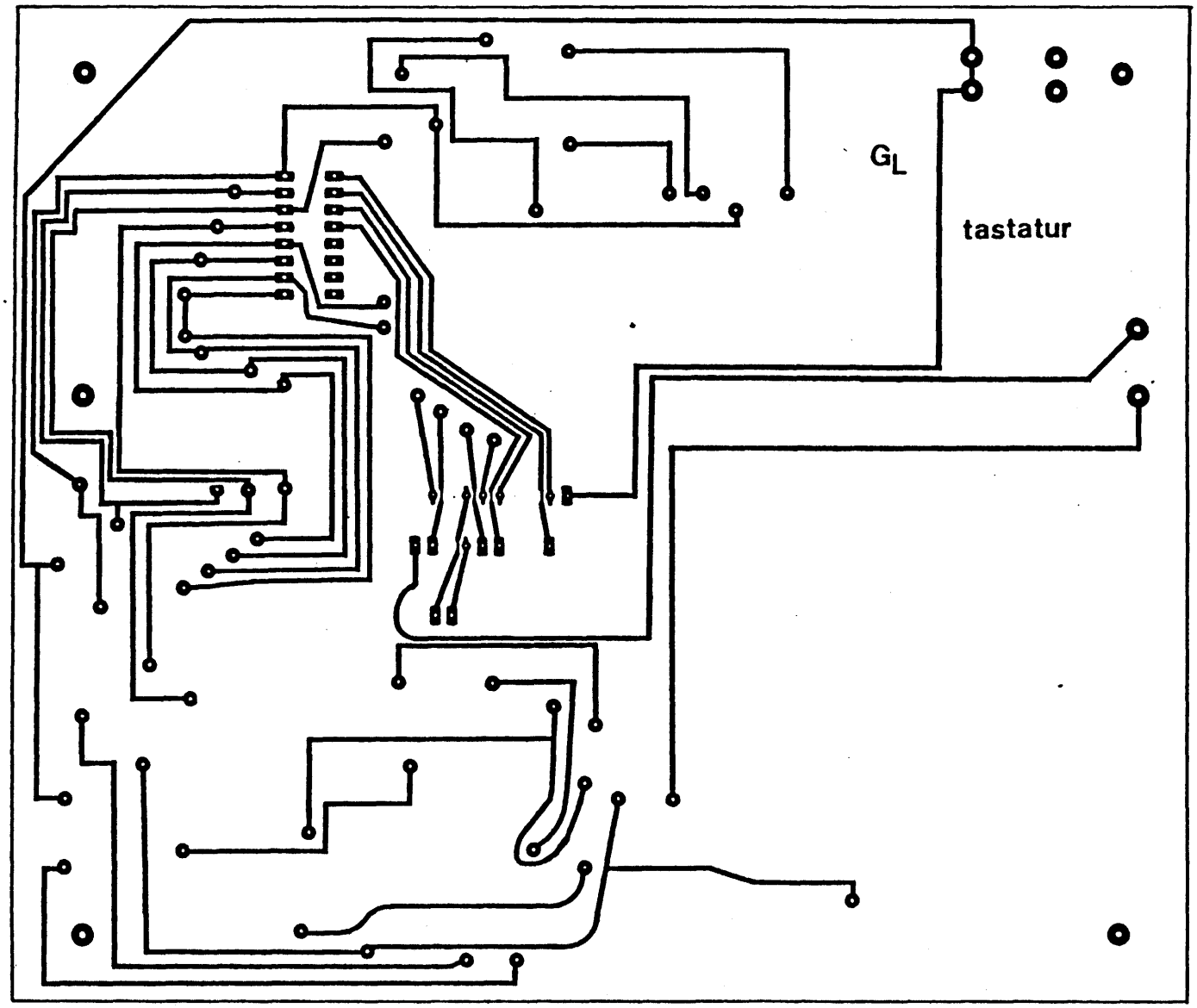

display
NH

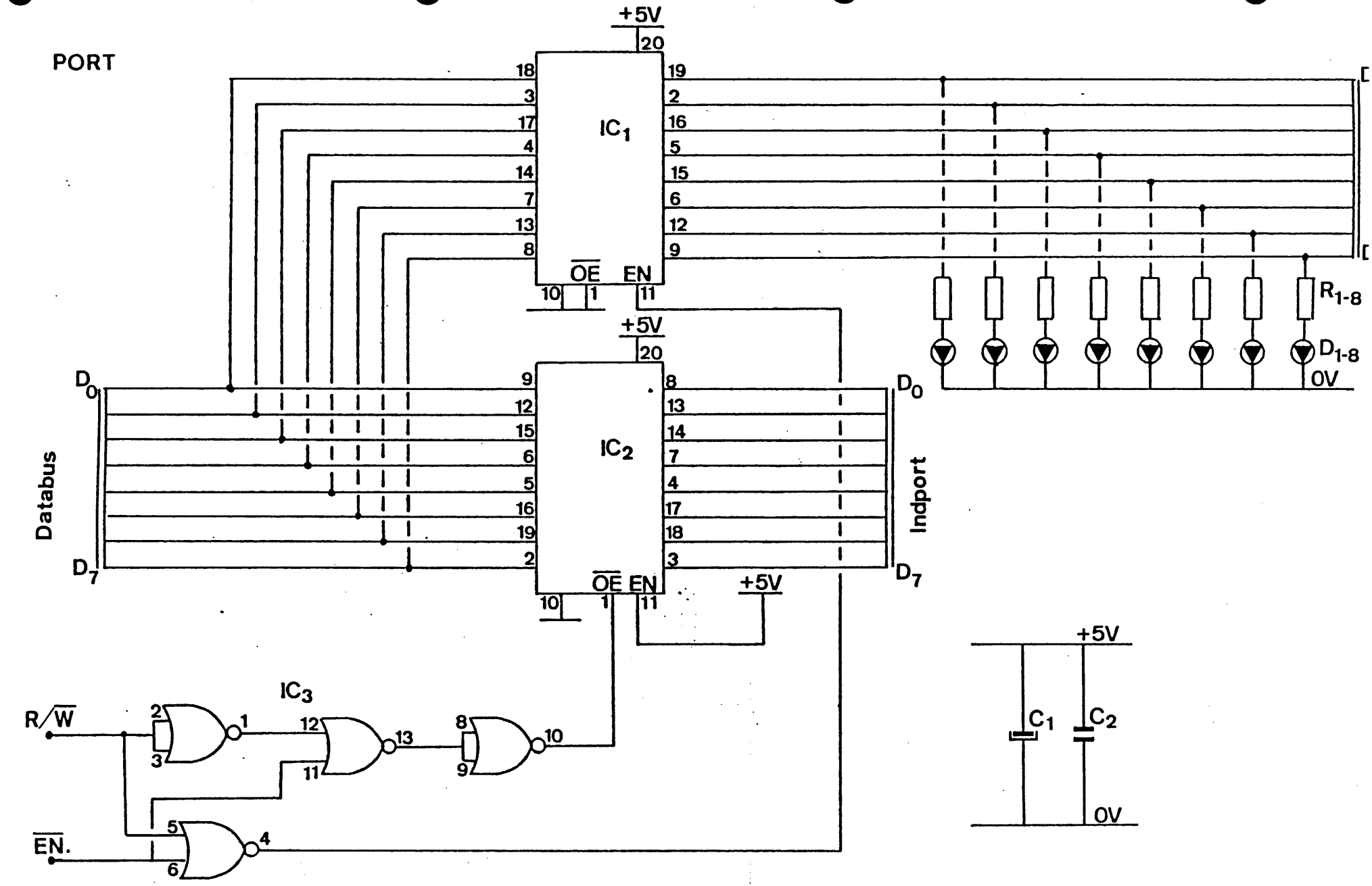

NH
displ.

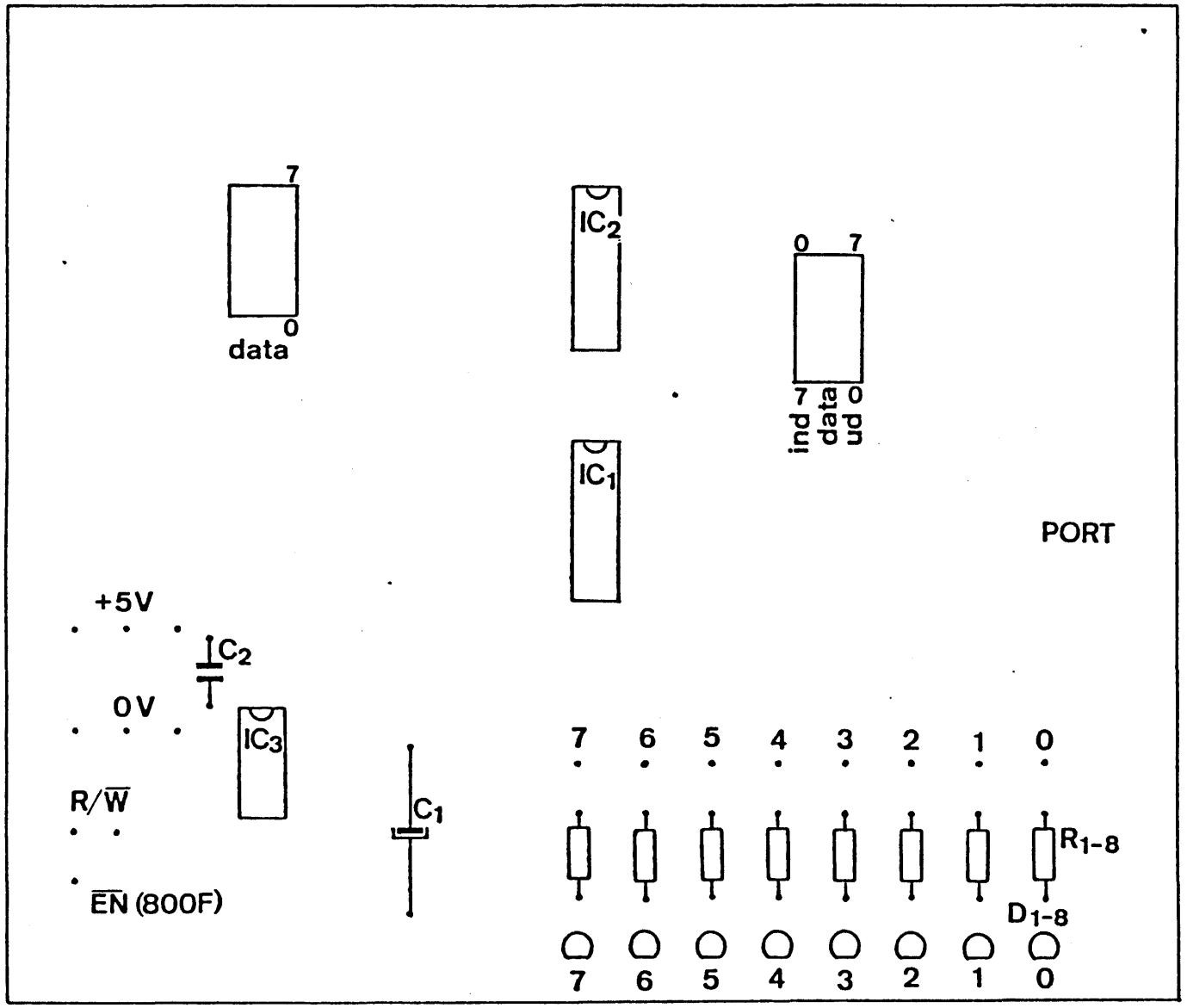

TASTATUR
0 1 2 3 4 5 6 7 bus
+5V
R17
En.1
En.2
IC8
IC9
0V
IC1
Res.
IC7
display
mellem-
led
R9 R10 R11 R12 R13 R14 R15 R16
D1-8
IC6
+5V
R1
C1
C5
5V
R2
C2
C6
IC5
+5V
C9
C10
0V
IC2
IC1
C3
R3
C7
R6
R7
C11
0V
IC3
IC4
KT1
R5
R8
IC1
C8
0V
+5V
R4
C4
IC2

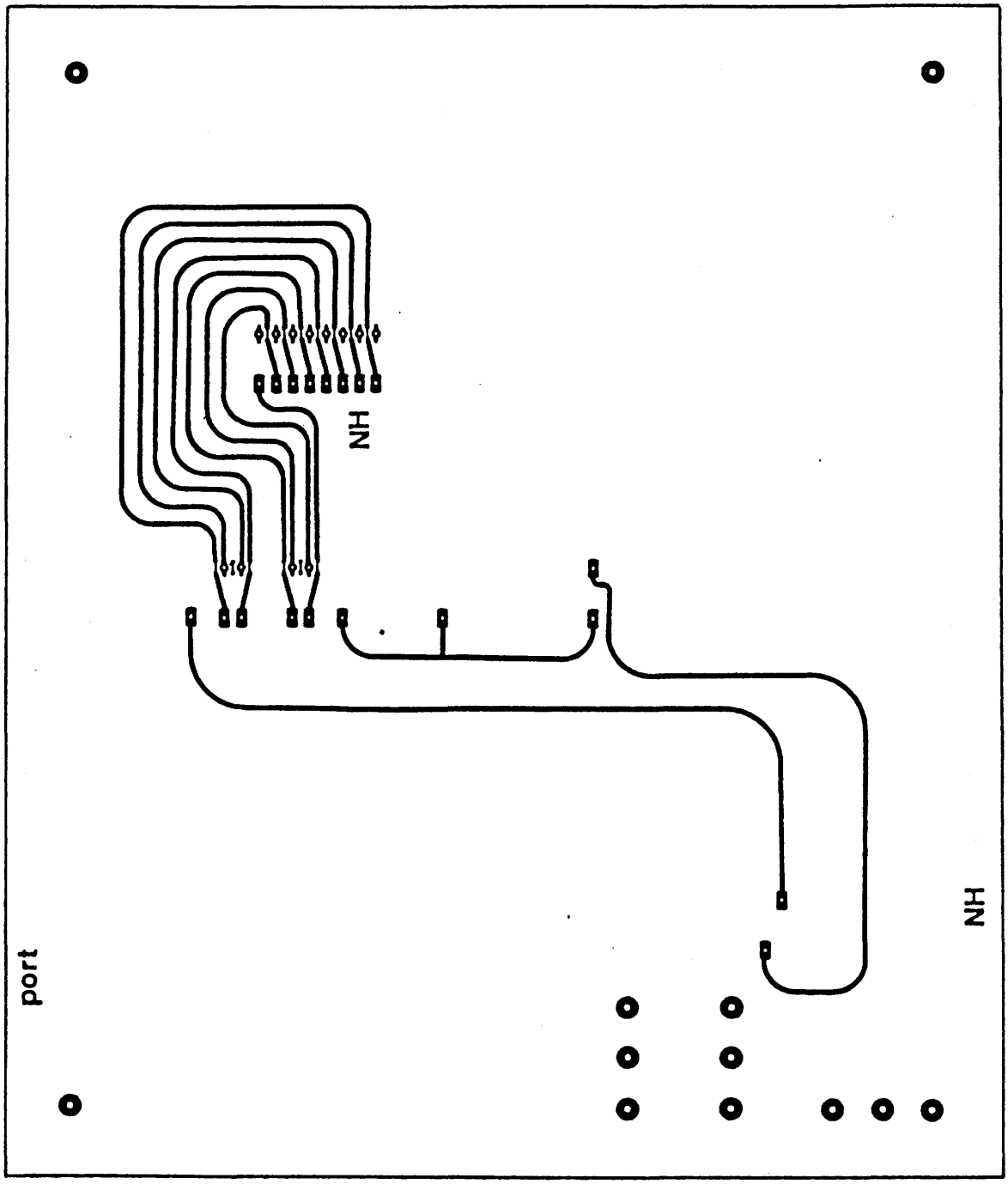

+5V
0V
SO
0
7
mellem.
data
7
0
R9-16
D1-8
7 6 5 4 3 2 1 0
En.2
(04)
Res
(05)
C9
C10
En.1
(03)
R17
IC9
IC8
C3
R3
C7
IC6
IC7
IC4
C1
C5
R1
IC3
C2
R2
C6
IC5
R4
R8
C4
C8
IC2
IC1
R5
sort
funktion
R6
R7
C11
on
off
KT
funktion
C
D
E
F
8
9
A
B
4
5
6
7
0
1
2
3
TASTATUR

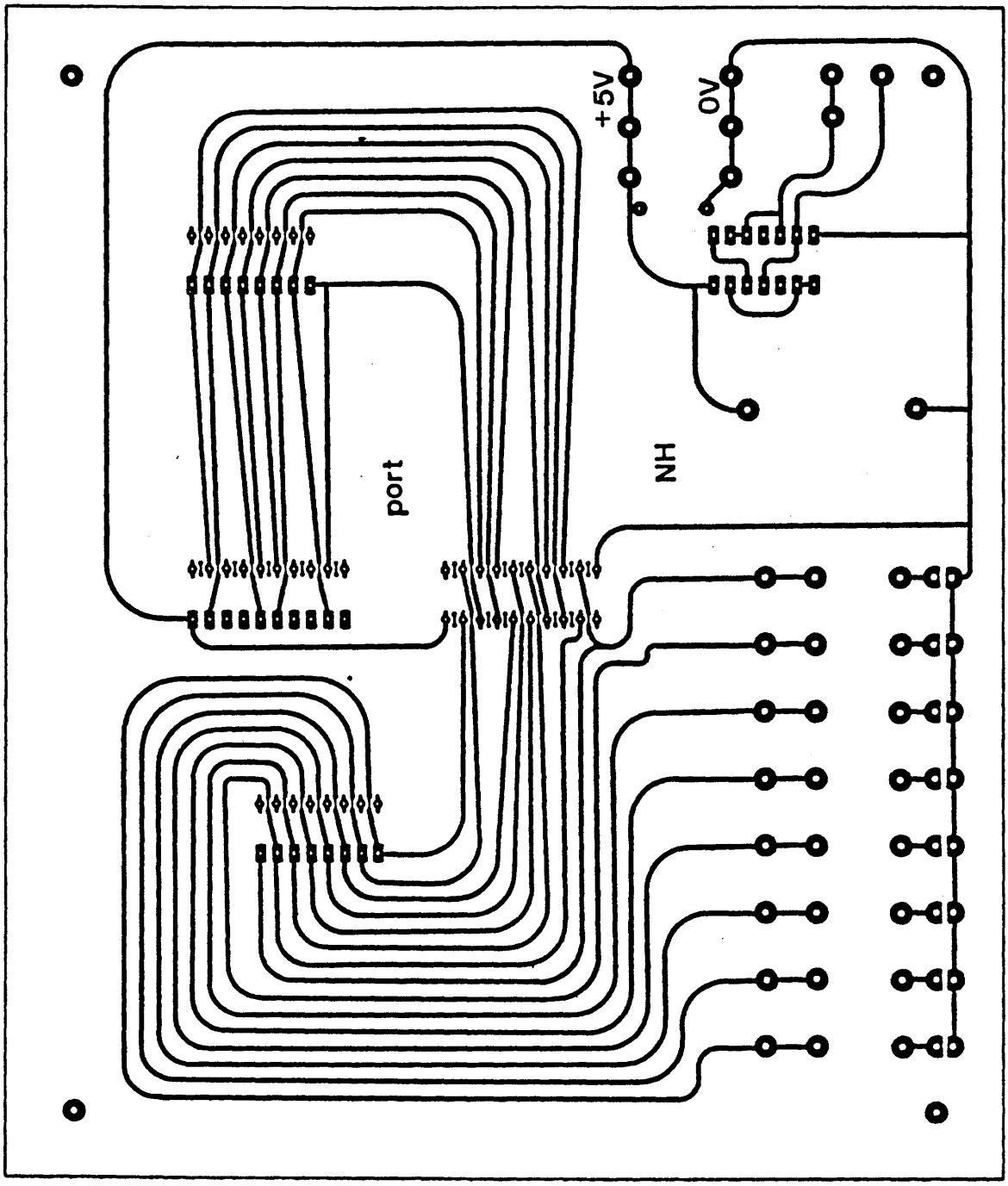

0V
+5V
En.2
Reset
G_L
En.1
tastatur


G_L
tastatur

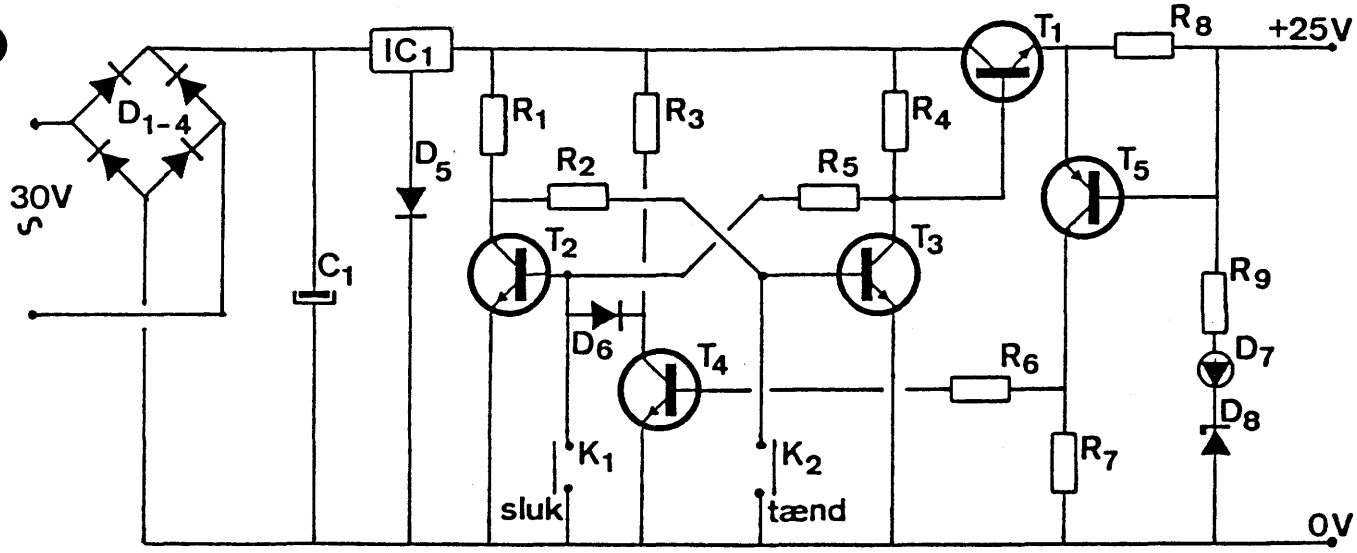

PORT
+5V
IC1
IC2
IC3
OE
EN
R/W
EN.
Databus
Indport
D0
D7
R1-8
D1-8
0V
C1
C2


7
0
data
IC2
0 7
7 data 0
ind ud
IC1
PORT
+5V
C2
0V
IC3
R/W
EN (800F)
C1
7 6 5 4 3 2 1 0
R1-8
D1-8
7 6 5 4 3 2 1 0

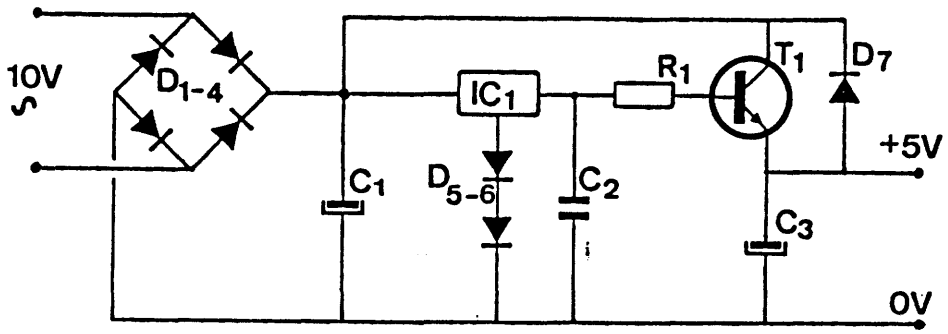

NH
port
NH

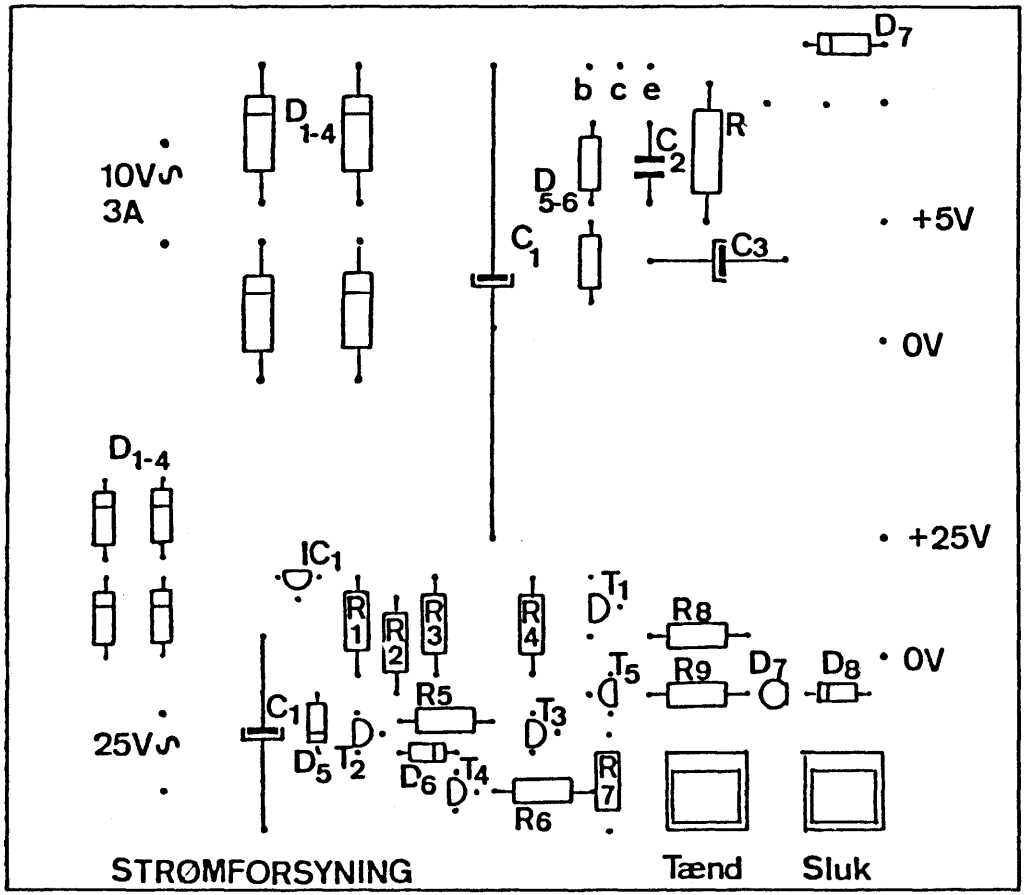

+5V
0V
port
NH

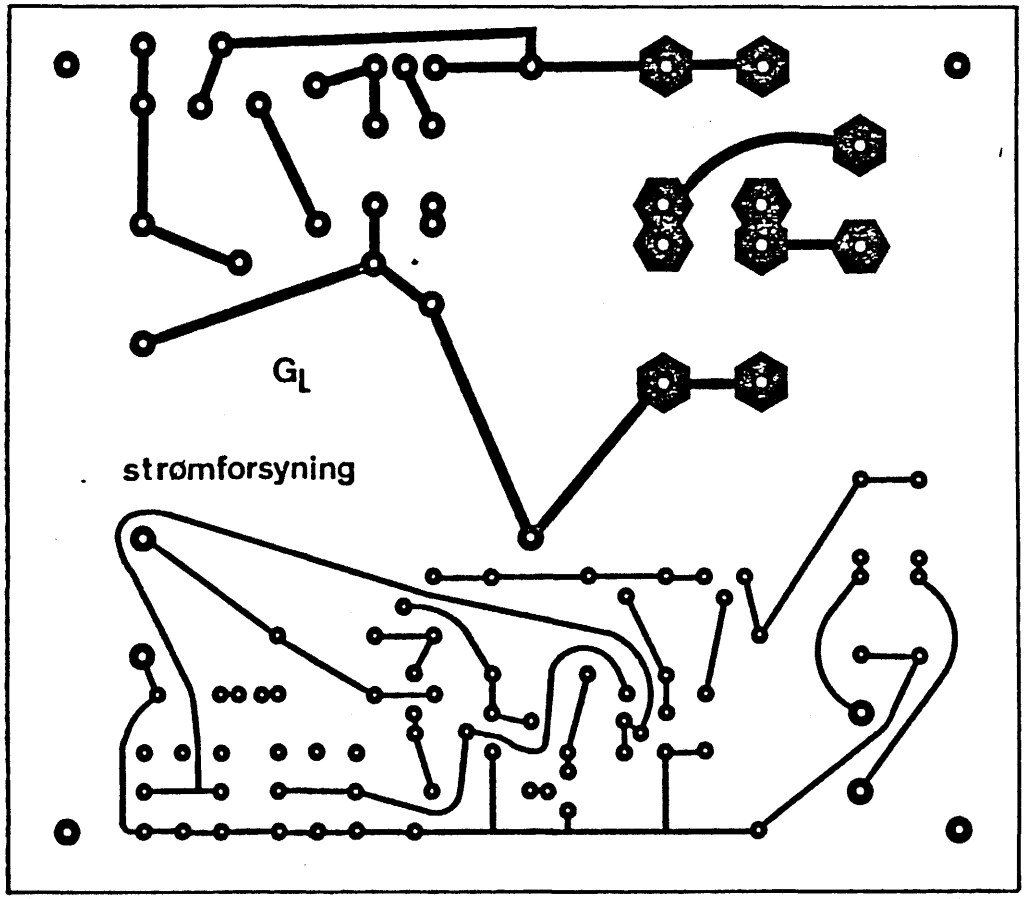
STRØMFORSYNING - BRÆNDER



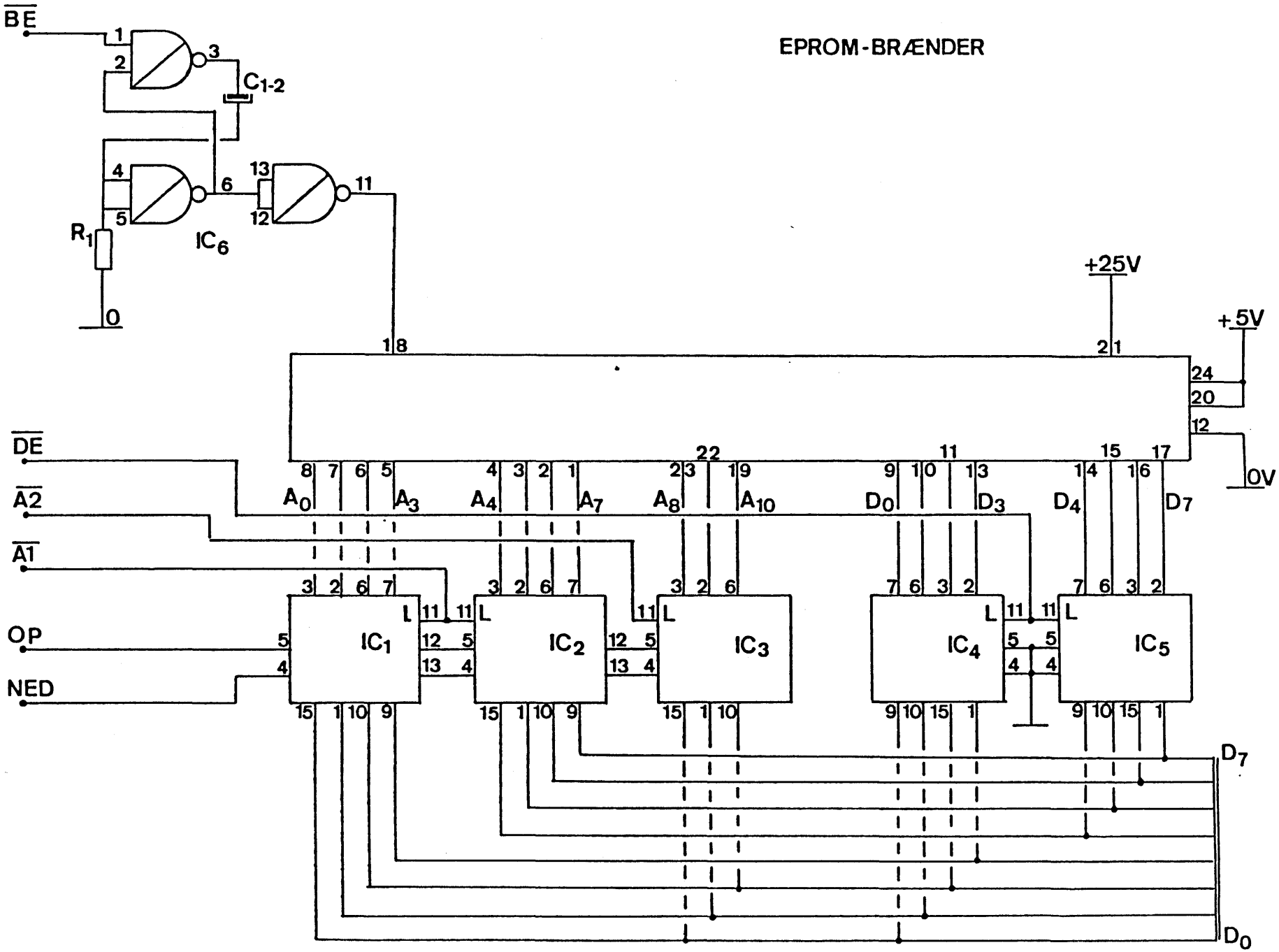
STRØMFORSYNING

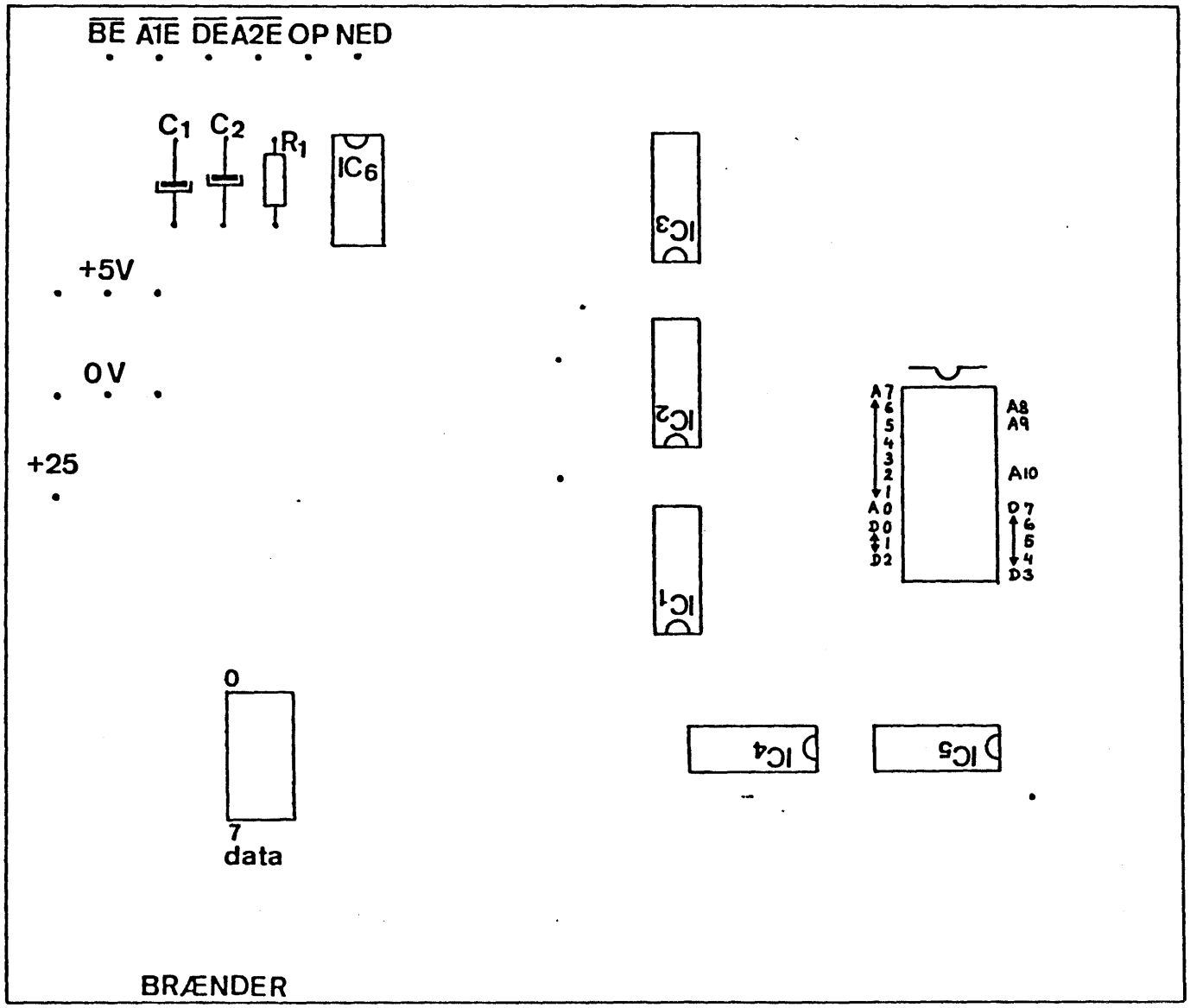


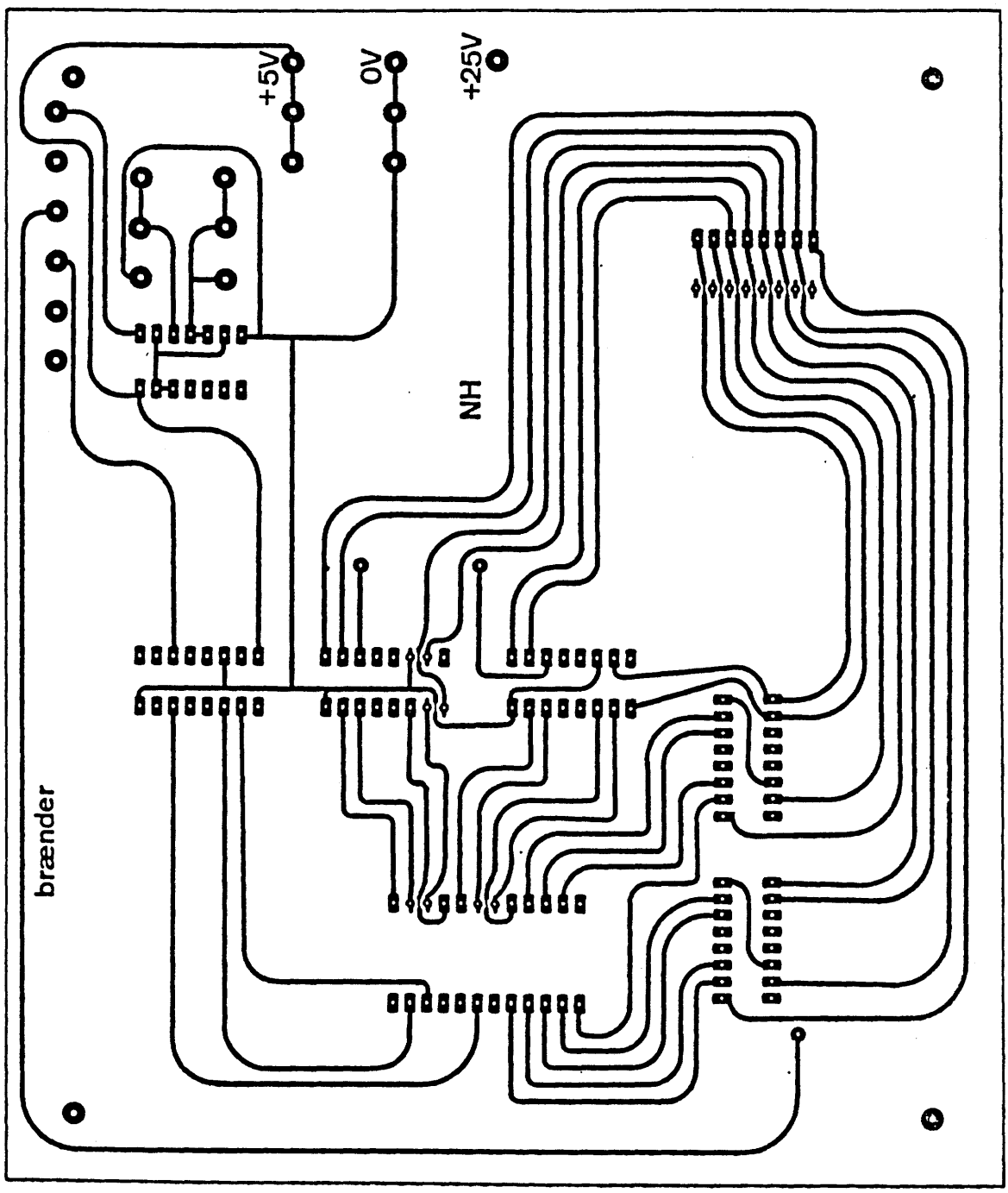

D7
b c e
D1-4
R
C2
10V~
3A
D5-6
+5V
C1
C3
0V
D1-4
+25V
IC1
T1
R8
R1
R2
R3
R4
T5
R9
D7
D8
0V
C1
R5
T3
25V~
D5
T2
D6
T4
R7
R6
STRØMFORSYNING
Tænd
Sluk

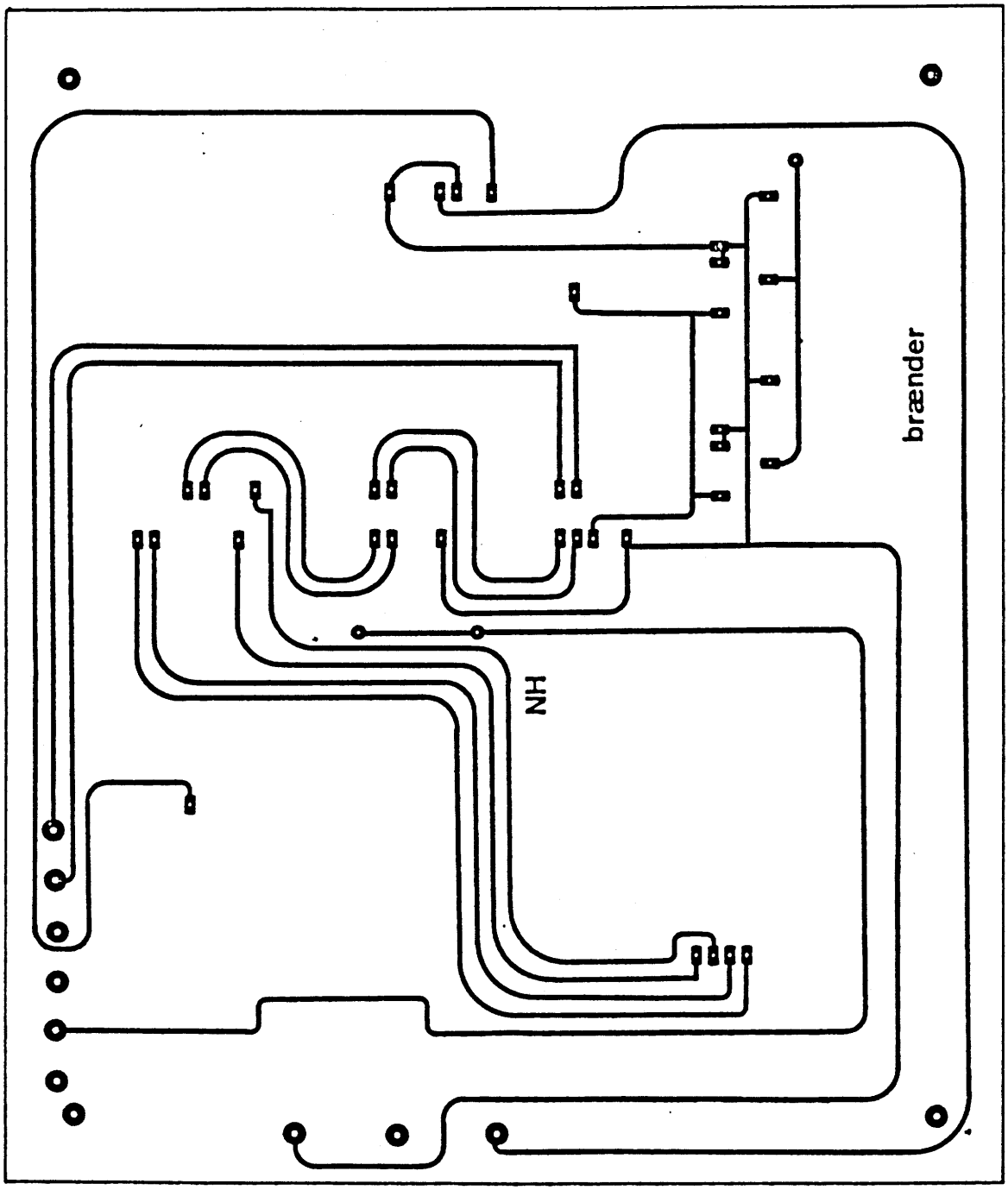

G_L
strømforsyning

EPROM-BRÆNDER

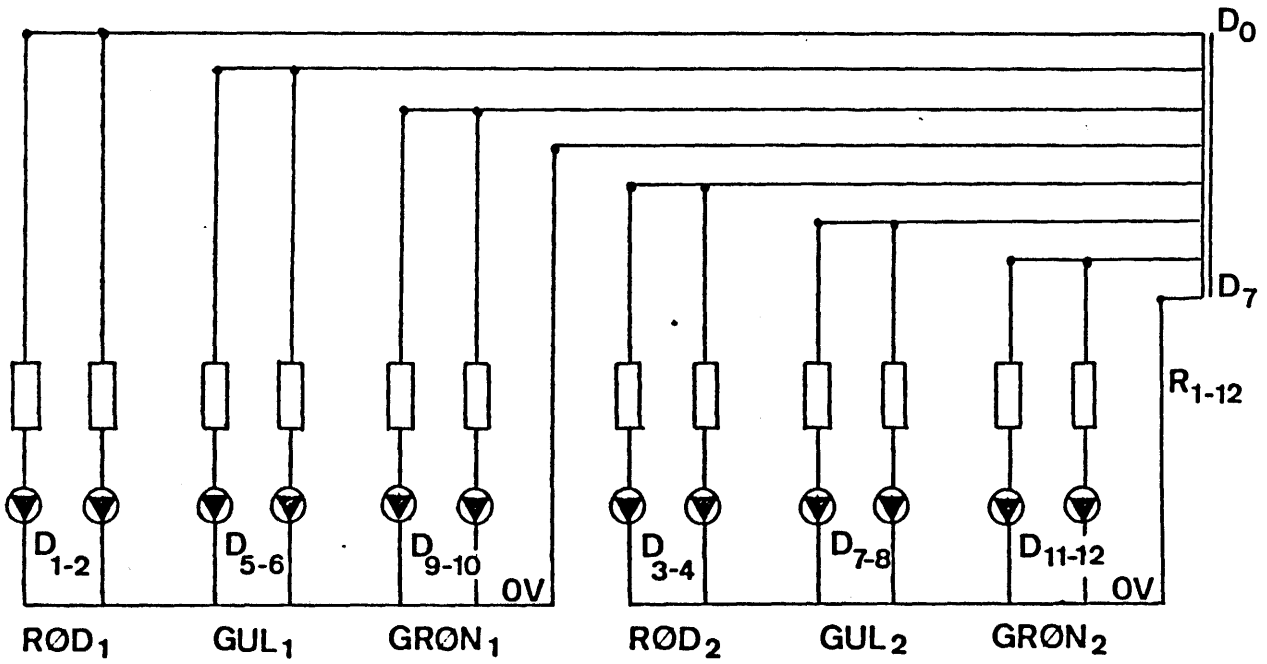


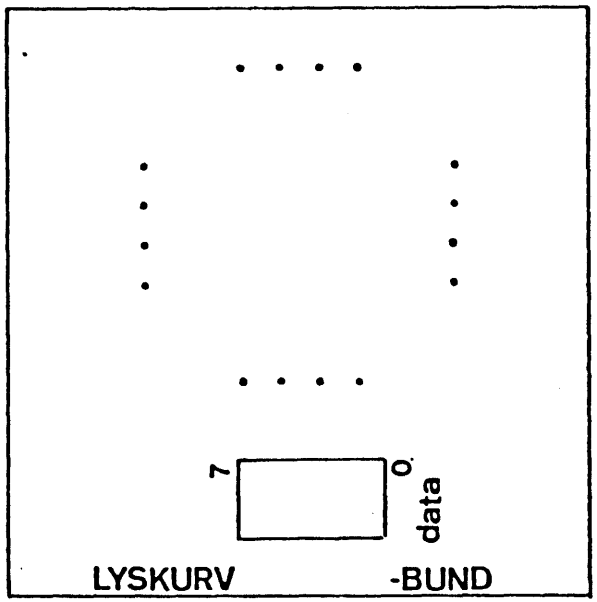

BE A1E DE A2E OP NED
C1
C2
R1
IC6
+5V
0V
+25
IC3
IC2
IC1
IC4
IC5
A7 6 5 4 3 2 1 A0 D0 1 D2
A8 A9 A10 D7 6 5 4 D3
0
7
data
BRÆNDER

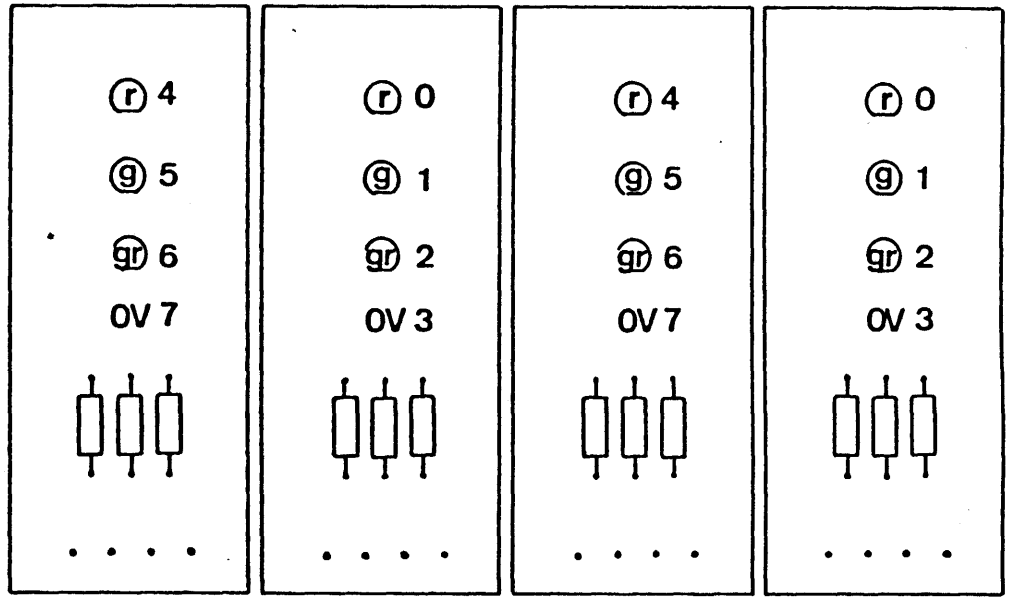

+5V
0V
+25V
NH
brænder

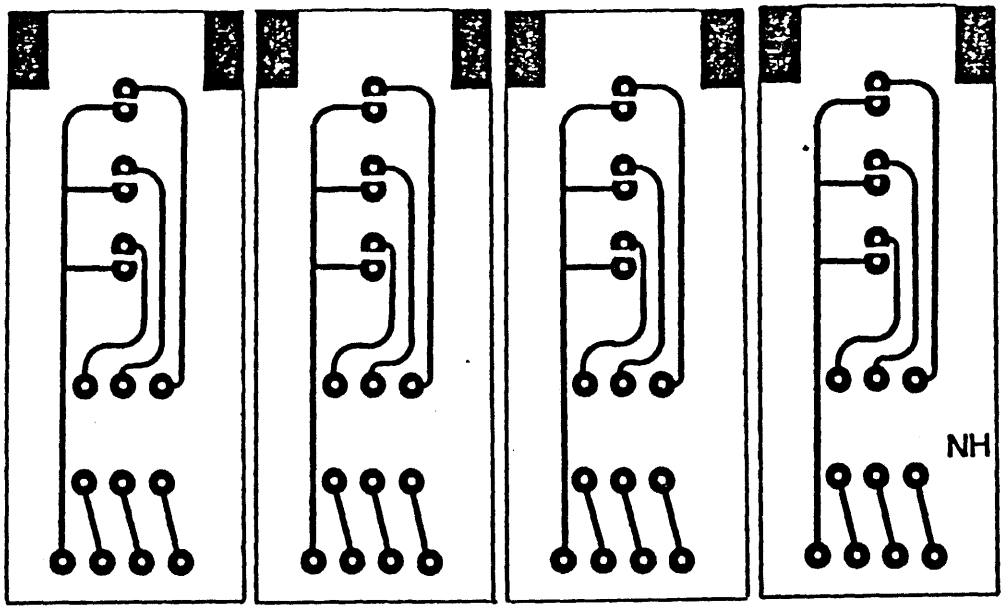

brænder
NH

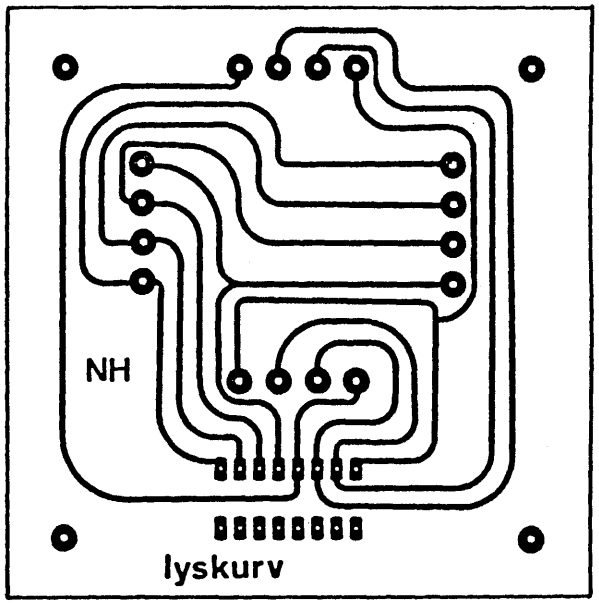
LYSKURV


D0
D7
R1-12
D1-2
D5-6
D9-10
0V
D3-4
D7-8
D11-12
0V
RØD1
GUL1
GRØN1
RØD2
GUL2
GRØN2

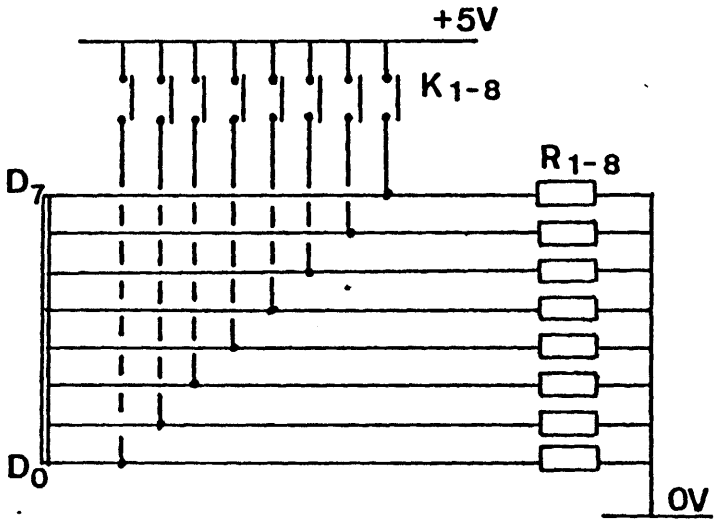

7
0
data
LYSKURV
-BUND

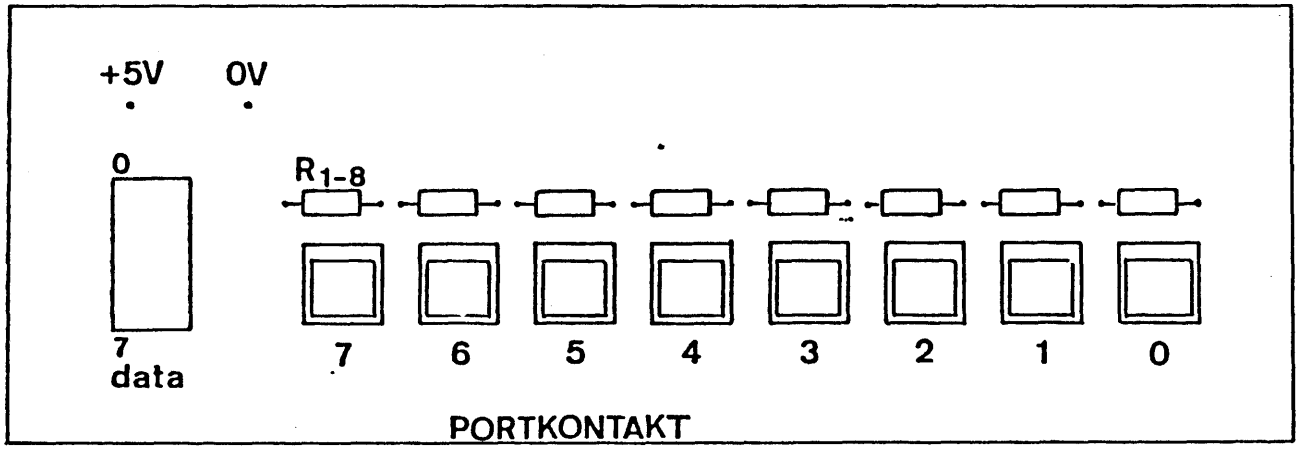

r 4
g 5
gr 6
0V 7
r 0
g 1
gr 2
0V 3
r 4
g 5
gr 6
0V 7
r 0
g 1
gr 2
0V 3

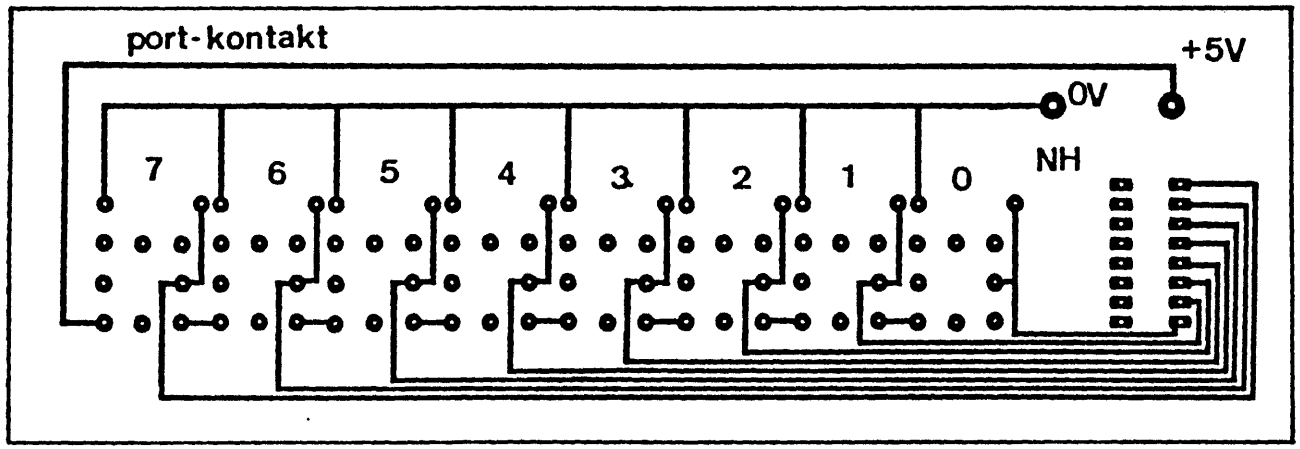

NH

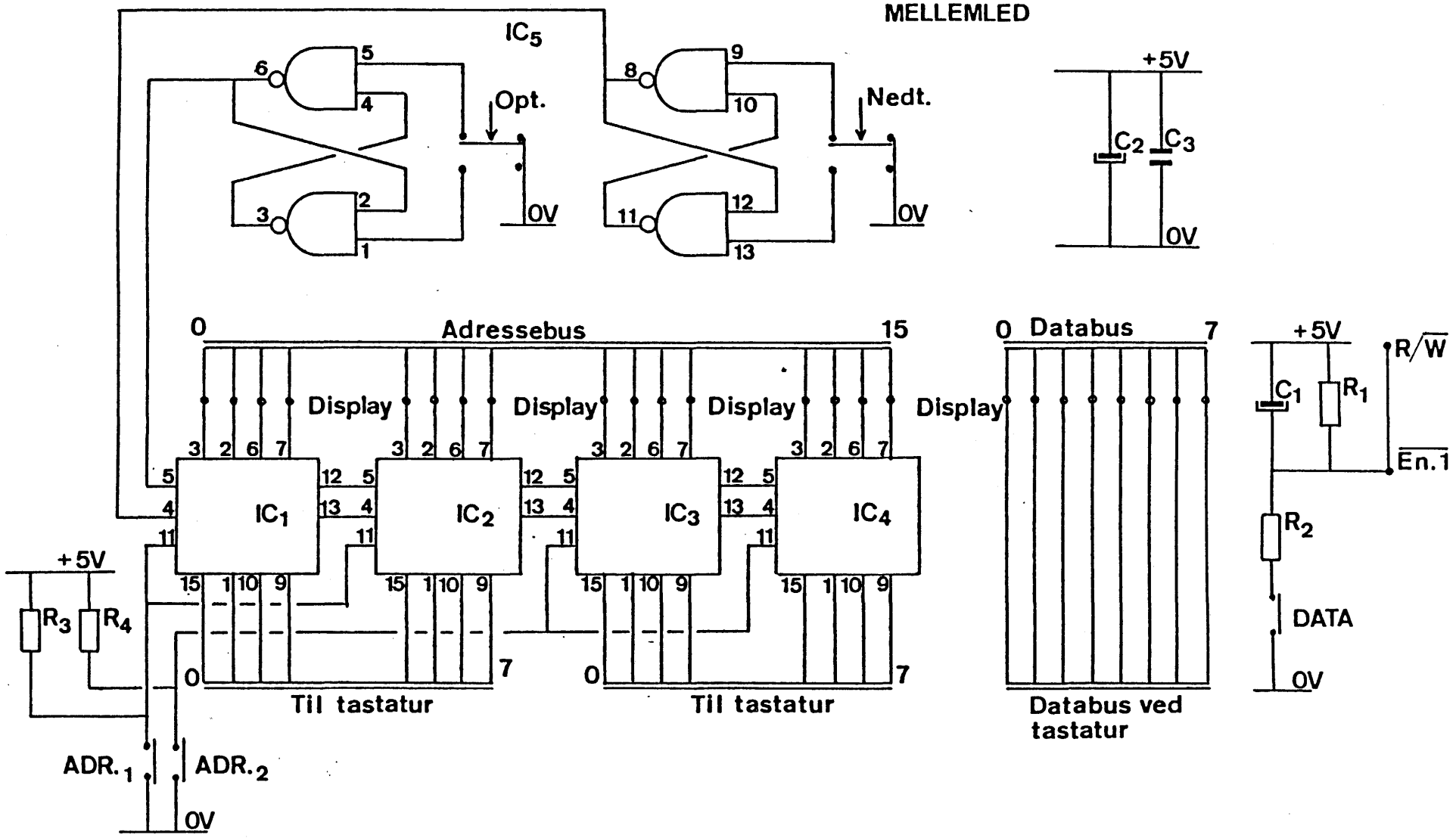

NH
lyskurv

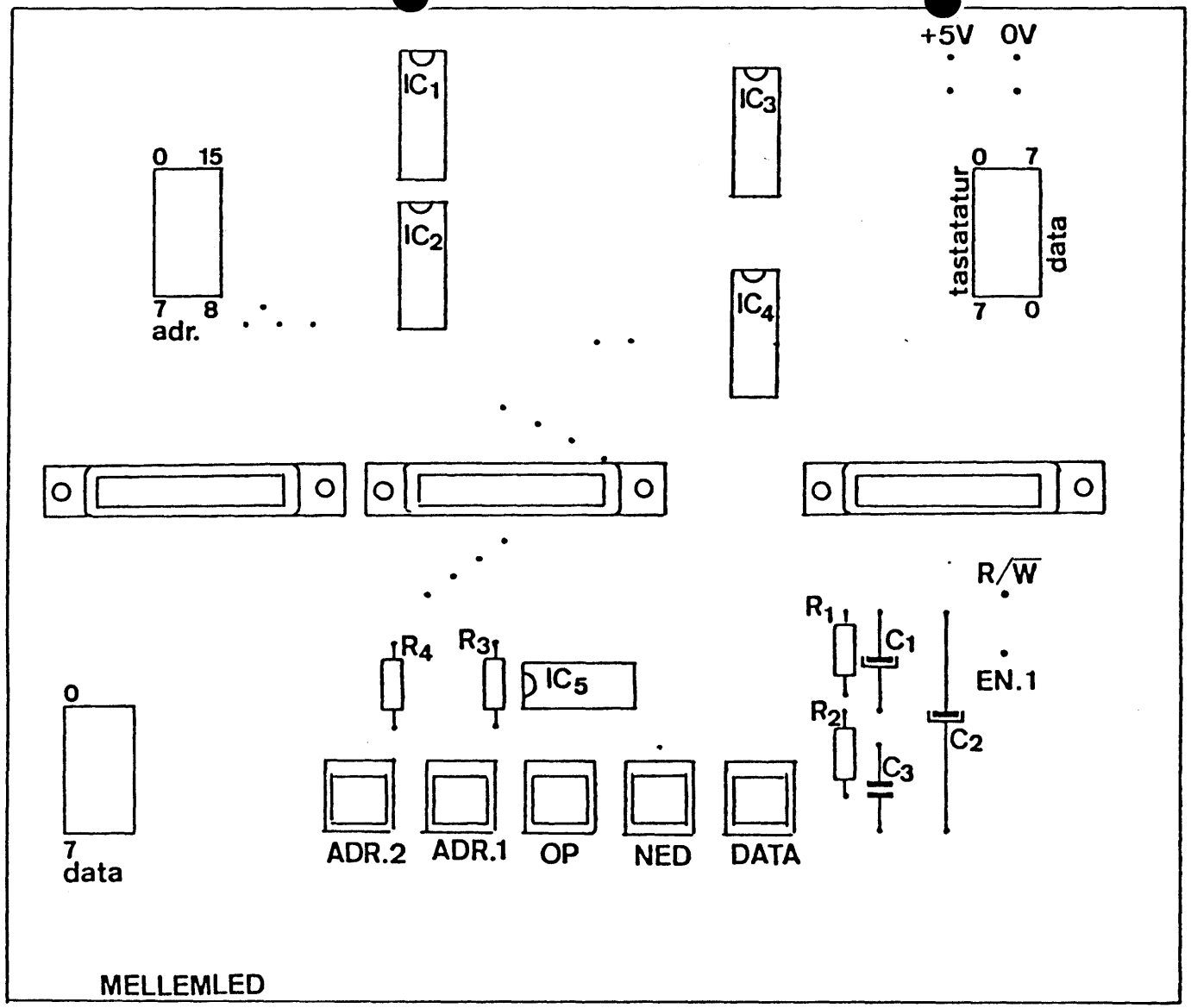
PORTKONTAKT

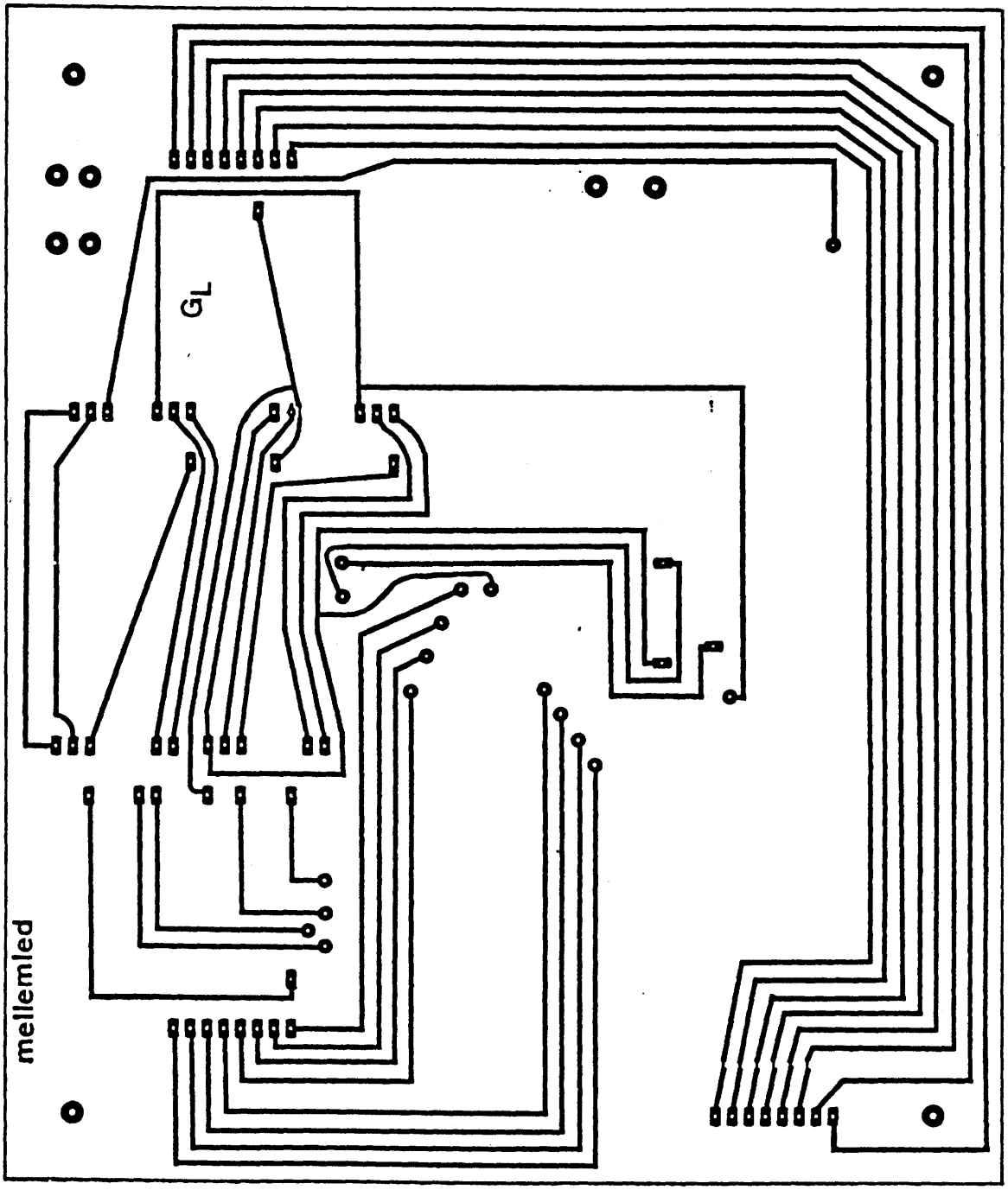


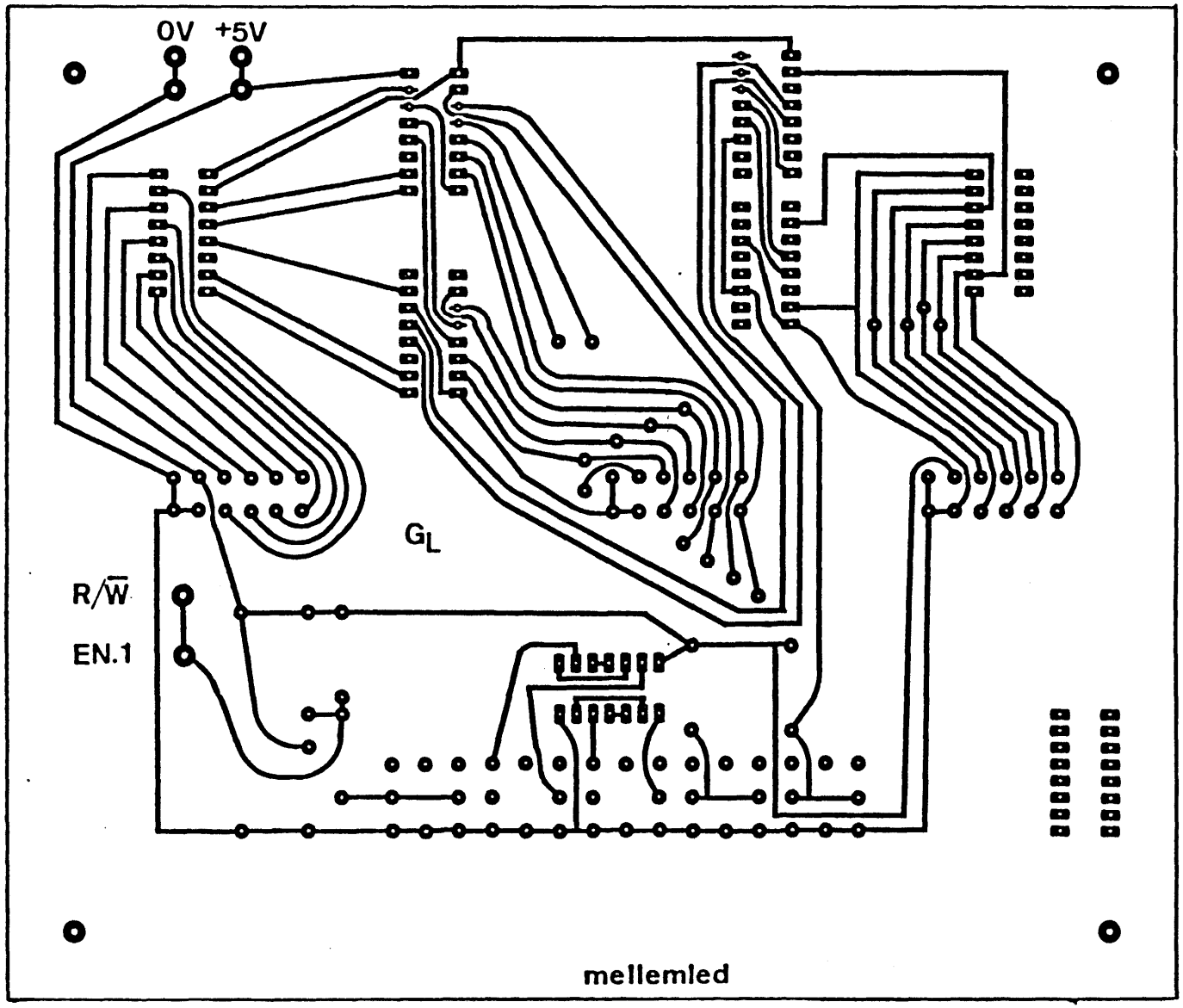

+5V
0V
0
7
data
R1-8
7
6
5
4
3
2
1
0
PORTKONTAKT


port-kontakt
+5V
0V
NH
7
6
5
4
3
2
1
0

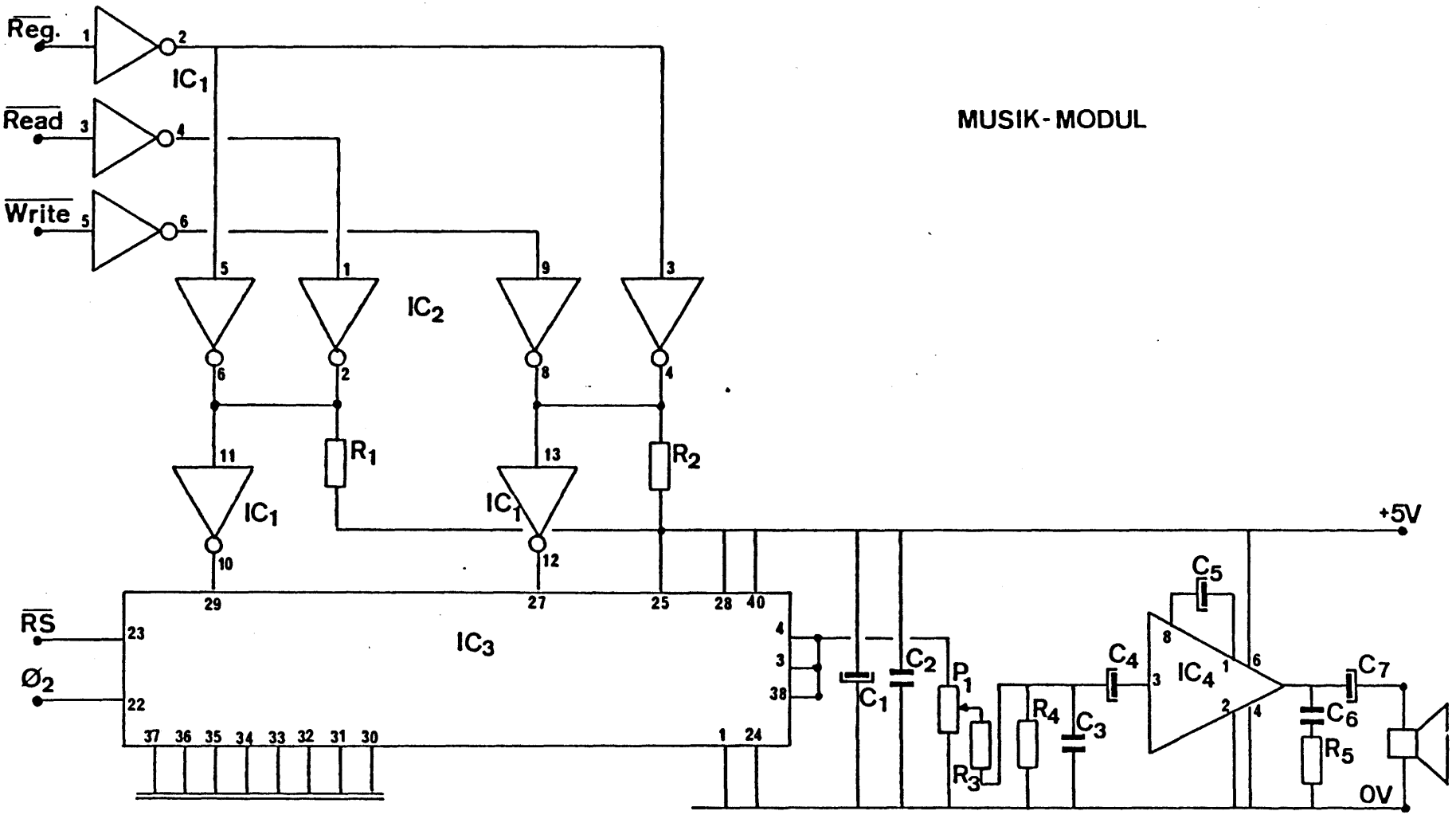

MELLEMLED
IC5
Opt.
Nedt.
0V
+5V
C2
C3
0
Adressebus
15
Display
IC1
IC2
IC3
IC4
Til tastatur
0 Databus 7
Databus ved tastatur
R/W
C1
R1
En.1
R2
DATA
R3
R4
ADR.1
ADR.2

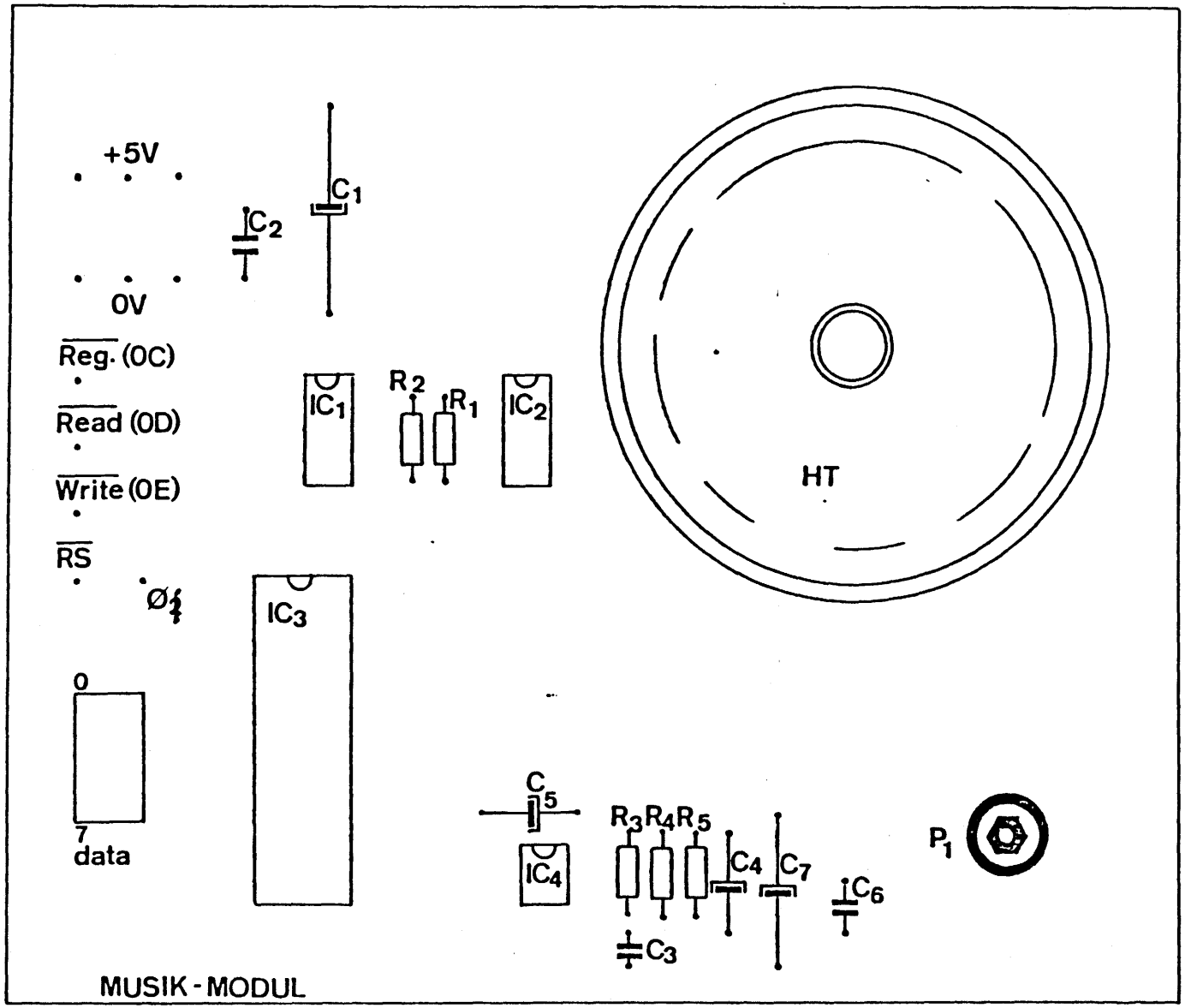

+5V
0V
IC1
IC2
IC3
IC4
0
15
7
8
adr.
0
7
tastatatur
data
7
0
R/W
R1
C1
R2
C2
C3
EN.1
R4
R3
IC5
0
7
data
ADR.2
ADR.1
OP
NED
DATA
MELLEMLED

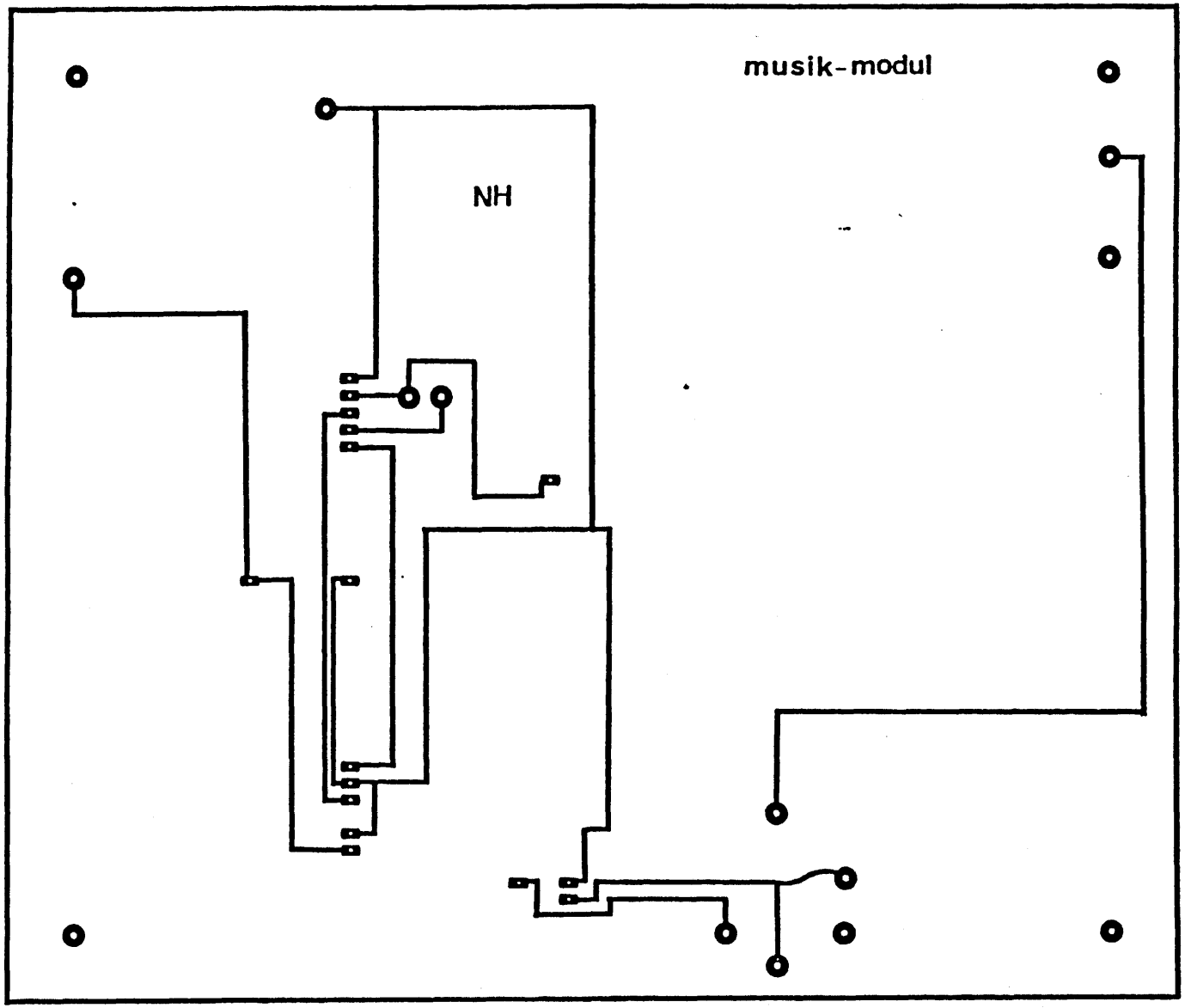

GL
mellemled

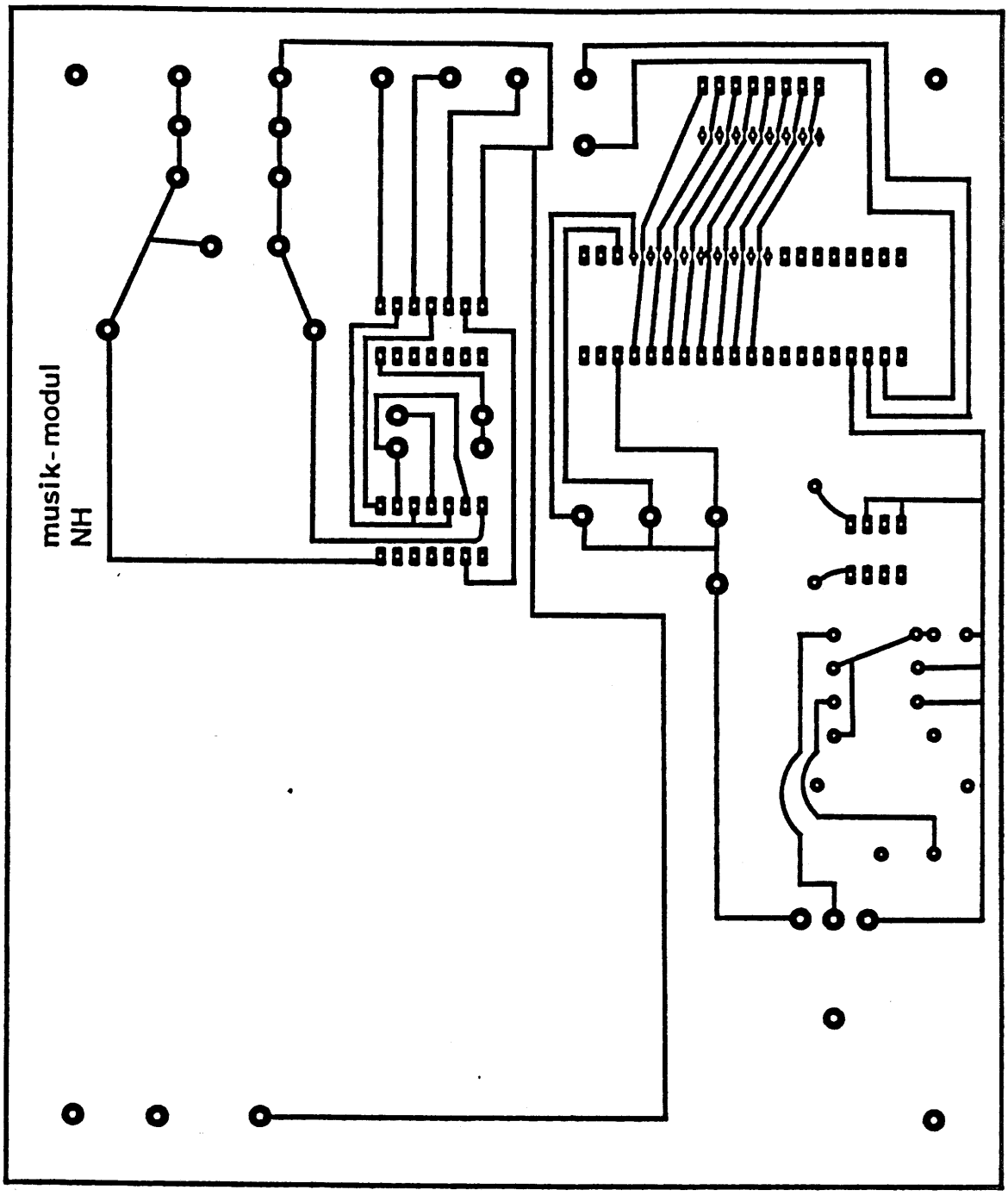

0V
+5V
G_L
R/$\overline{W}$
EN.1
mellemled

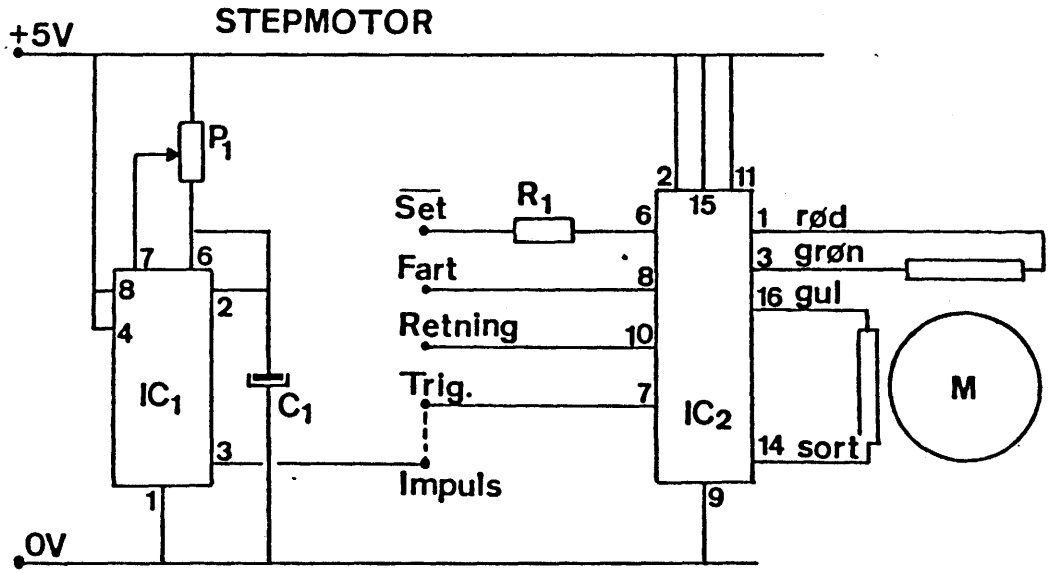
MUSIK-MODUL

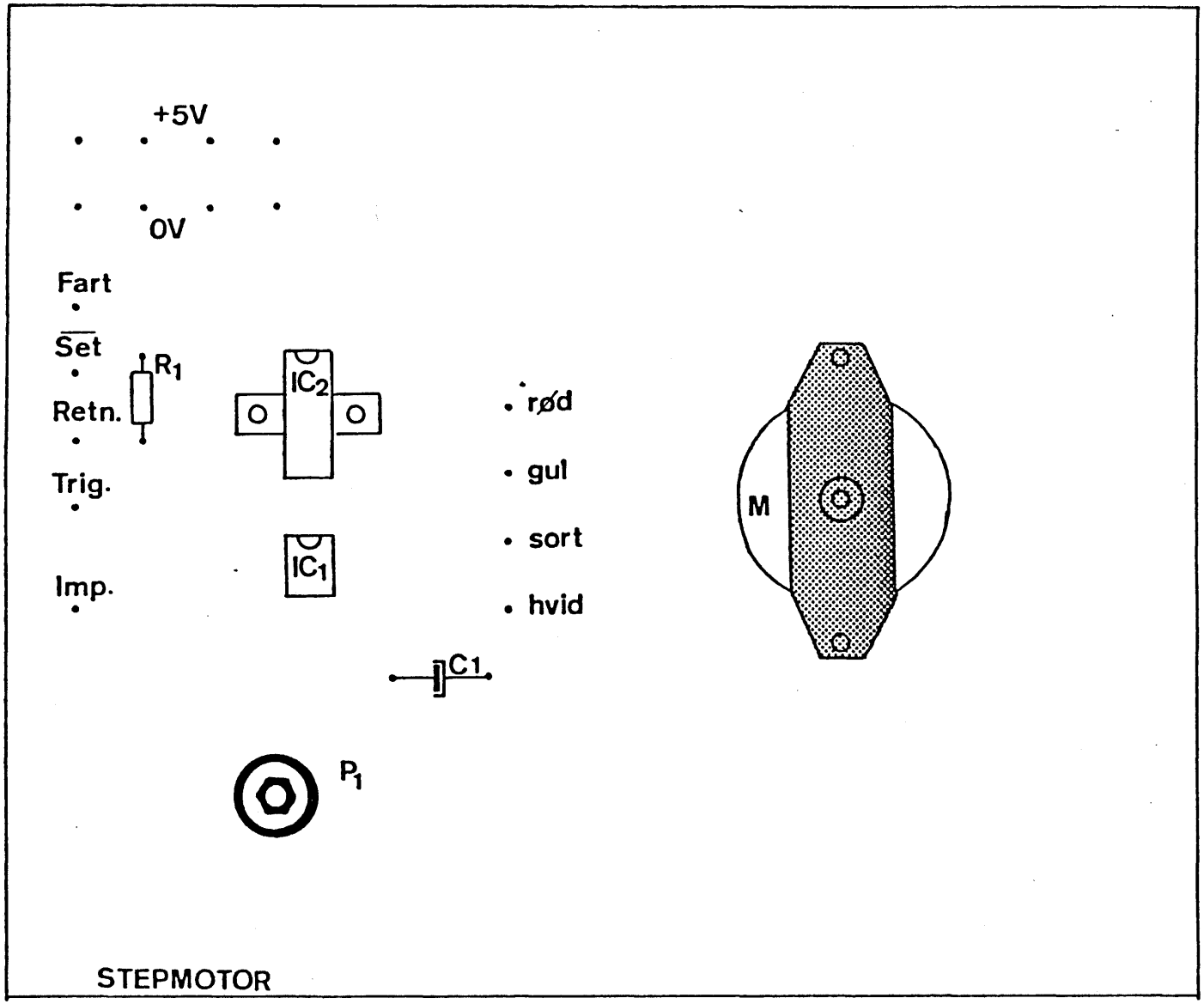

Reg.
Read
Write
IC1
IC2
R1
R2
IC1
IC1
+5V
RS
Ø2
IC3
C1
C2
P1
R3
R4
C3
C4
IC4
C5
C6
C7
R5
0V

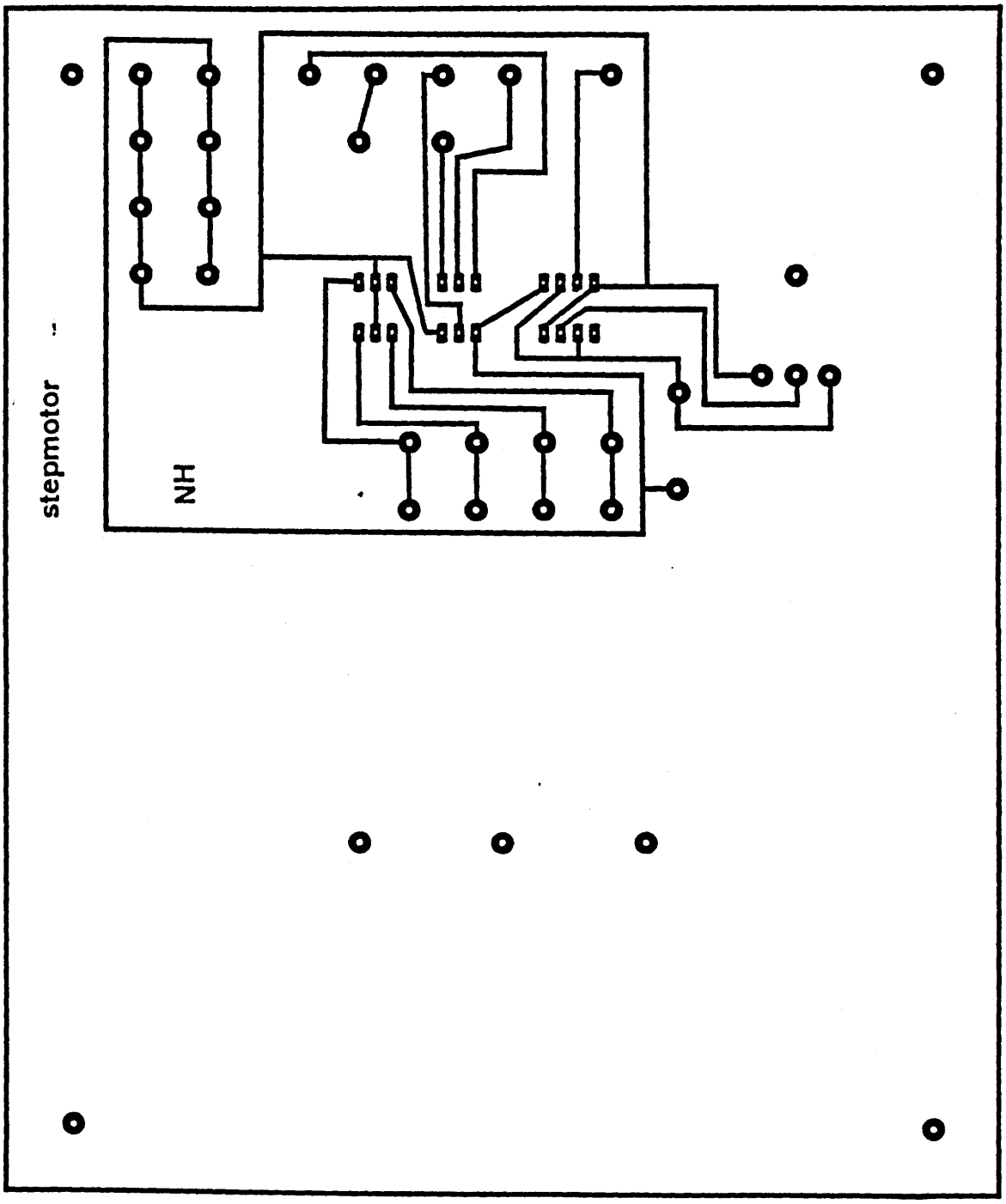

+5V
0V
Reg. (0C)
Read (0D)
Write (0E)
RS
Ø2
0
7
data
C1
C2
IC1
R2
R1
IC2
IC3
HT
C5
IC4
R3
R4
R5
C4
C7
C6
C3
P1
MUSIK - MODUL


musik-modul
NH


musik-modul
NH


STEPMOTOR
+5V
0V
P1
IC1
C1
R1
Set
Fart
Retning
Trig.
Impuls
IC2
M
rød
grøn
gul
sort
1
2
3
4
6
7
8
9
10
11
14
15
16


+5V
0V
Fart
Set
Retn.
Trig.
Imp.
R1
IC2
IC1
C1
P1
rød
gul
sort
hvid
M
STEPMOTOR


stepmotor
NH

REGISTER OVER TEKNISKE UDTRYK