

# TI kursus

## DATAMASKINETEKNIK, GRUNDLÆGGENDE.

40/50 15110.62  
1978/79

31. juli - 11. august 1978.

Mandag den 31.07.:

Kl. 8.00 - 11.50	Indledning.	
- 12.30 - 16.20	Datamaskinens hovedenheder Talsystemer.	Kap. 2. (s.1-25). Kap. 1.

Tirsdag den 01.08.:

Kl. 8.00 - 11.50	Datamaskinens hovedenheder	Kap. 2. (s.26-32).
	Datamaskinens virkemåde	Kap. 3. (s.12-8).
Kl. 12.30 - 16.20	Datamaskinens virkemåde	Kap. 3. (s.1-8).
	Diagrammering	Kap. 6. (s.1-12).

Onsdag den 02.08.:

Kl. 8.00 - 11.50	Datamaskinens virkemåde	Kap. 3. (s. 8-12).
	Ydre enheder	Kap. 5.
Kl. 12.30 - 16.20	Ydre enheder	Kap. 5.

Torsdag den 03.08.

Kl. 8.00 - 11.50	Programmering i maskinsprog	Kap. 7. (s. 1-8 + 15-16).
- 12.30 - 16.20	Diagrammering	Kap. 6. (s.13-27).

Fredag den 04.08.:

Kl. 8.00 - 11.50	Programmering i maskinsprog	Kap. 7. (s.8-14 + 22-24).
- 12.30 - 16.20	Programmering i maskinsprog	Kap. 7. (s.8-14 + 22-24).

Mandag den 06.08.

Kl. 8.00 - 11.50	Datamaskinens virkemåde	Kap. 3. (s.13-23).
- 12.30 - 16.20	Datamaskinens virkemåde	Kap. 3. (s.13-23).

Tirsdag den 08.08.

Kl. 8.00 - 11.50	Programmering i maskinsprog	Kap. 7. (s.17-22).
- 12.30 - 16.20	Programmering i maskinsprog	Kap. 7. (s.17-22).

Onsdag den 09.08.:

Kl. 8.00 - 11.50	Datamaskinens ind/udlæse- system	Kap. 4.
Kl. 12.30 - 16.20	Datamaskinens ind/udlæse- system	Kap. 4.

Torsdag den 10.08.

Kl. 8.00 - 11.50	Programmering i maskinsprog	Kap. 7. (øvelser).
	Programmering i symbolsk maskinsprog	Kap. 8. (s. 1-5 ).
Kl. 12.30 - 16.20	Programmering i symbolsk maskinsprog	Kap. 8 (s. 9-17).

# TI kursus

DATAMASKINETEKNIK, GRUNDLÆGGENDE.

40/50 15110.62

31. juli - 11. august 1978.

1978/79

side 2.

Fredag den 11.08.:

Kl. 8.00 - 11.50

Programmering i symbolsk  
maskinsprog

Kap. 8. (s.6-8).

Kl. 12.30 - 15.30

Programmering i symbolsk  
maskinsprog

Kap. 8. (øvelser).

Kl. 15.30 -

Afslutning.

VARIGHED:

80 timer.

UNDERVISNINGSTED:

Teknologisk Institut  
Gregersensvej  
2630 Tåstrup, Indg. 8.

LOKALE:

2073

LÆRERE:

Mogens Nimand Thomsen T.I.  
Leif Bjerregaard T.I.



## F O R O R D

Hensigten med dette læremiddel "Datamaskineteknik" er at give deltagerne på efteruddannelseskursus i "Datamaskineteknik, grundlæggende" de bedst mulige betingelser for at tilegne sig stoffet, dels ved gennemgang af de enkelte teorilektioner, dels gennem løsning af de praktiske opgaver.

Kursus er opdelt i otte fagområder, og vi har fundet det hensigtsmæssigt at opdele læremidlet i et tilsvarende antal afsnit, hvert svarende til et af de otte fagområder.

Læremidlet kan yderligere suppleres med tre afsnit - "Datamaskineteknik, supplement" - omhandlende indholdet i efteruddannelseskursus "Kontormaskineelektronik trin 5, datamaskiner".

Selve udarbejdelsen af kursus er foregået i nært samarbejde mellem repræsentanter fra erhvervsliv og uddannelsesinstitutioner. Undervisningsplanen og læremidlet er udarbejdet af medarbejdere fra afdelingen for automatiseringsteknik på Teknologisk Institut i København samt fra Institutet for Elektroniske Systemer på Ålborg Universitetscenter, koordineret af ingeniør Otto Højbjerg Jeppesen og lektor, civilingeniør Allan Dresling. Den forlagsmæssige del af arbejdet med læremidlet er varetaget af Jernindustriens Forlag, og Direktoratet for arbejdsmarkedsuddannelserne har støttet projektet økonomisk.

Udvalget vil gerne her benytte lejligheden til at takke erhvervs- og uddannelsesrepræsentanter samt Jernindustriens Forlag for den store indsats, der er ydet i forbindelse med udarbejdelsen. Ligeledes vil udvalget takke Direktoratet for arbejdsmarkedsuddannelserne for positiv indstilling til projektet og den økonomiske støtte, som muliggjorde dets realisering.

Det er udvalgets håb, at der med kurset er tilført endnu et nyttigt led i den række af efteruddannelseskursus, som dækker området elektronik og kontormaskineelektronik.

København, december 1977

Metalindustriens Efteruddannelsesudvalg

## INDHOLD

### 1. TALSYSYSTEMER

1.1	Talnotationssystemer	1.1
1.2	Binær aritmetik	1.17
1.3	Talrepræsentation i datamaskinen	1.25
1.4	Opgaver	1.30

### 2. DATAMASKINENS HOVEDENHEDER

2.1	Generel blokmodel	2.1
2.2	Lagerenhed	2.4
2.3	Regneenhed	2.10
2.4	Ind/udlæseenhed	2.18
2.5	Styreenhed	2.22
2.6	Alternativ beskrivelse af datamask	2.26
2.7	Spec. af register- struktur	2.27
2.8	Øvelsesopg. og spørgsmål	2.30

### 3. DATAMASKINENS VIRKEMÅDE

3.1	Princip for program- styring	3.1
3.2	Maskinordrer	3.3
3.3	Ordretyper	3.5
3.4	Ordresekvens	3.8
3.5	Eksekvering af maskinordre	3.8
3.6	Trinvis udførelse	3.10
3.7	Eksekvering af maskinordre v.h.a. mikro-program	3.15
3.8	Principopbygning af mikroprogrammerbar styreenhed	3.17
3.9	Øvelsesspørgsmål	3.22

### 4. DATAMASKINENS IND/UDLÆSESYSTEM

4.1	Relevante begreber	4.1
4.2	Interfacekortets opgaver	4.5
4.3	Programstyret I/O	4.9
4.4	Interrupt	4.13
4.5	DMA	4.23

<u>5.</u>	<u>YDRE ENHEDER</u>	
5.1	Relevante begreber	5.1
5.2	Baggrundslager	5.5
5.3	Ind/udlæseenheder	5.14
5.4	Procesdata	5.22
<u>6.</u>	<u>DIAGRAMMERING</u>	
6.1	Funktionsdiagrammer	6.1
6.2	Systemdiagram	6.13
6.3.	Blokdiagram	6.21
<u>7.</u>	<u>PROGRAMMERING I MASKINSPROG</u>	
7.1	Maskininstruktioner	7.1
7.2	Elementær programmering	7.15
<u>8.</u>	<u>PROGRAMMERING I SYMBOLSK MASKINSPROG</u>	
8.1	Instruktioner i symbolsk maskinsprog.	8.1
8.2	Særlige faciliteter	8.9
<u>12.</u>	<u>STIKORDSREGISTER</u>	12.1

## 1. TALSYSTEMER

Talangivelser, således som de kendes fra hverdagen, er baseret på visse principper i deres opbygning.

I et flercifret tal er tallets størrelse således bestemt af de enkelte cifres placering, eller position, i tallet og ikke af deres placering i forhold til hinanden. På fig. 1.a. er vist et eksempel på et 4-cifret tal.

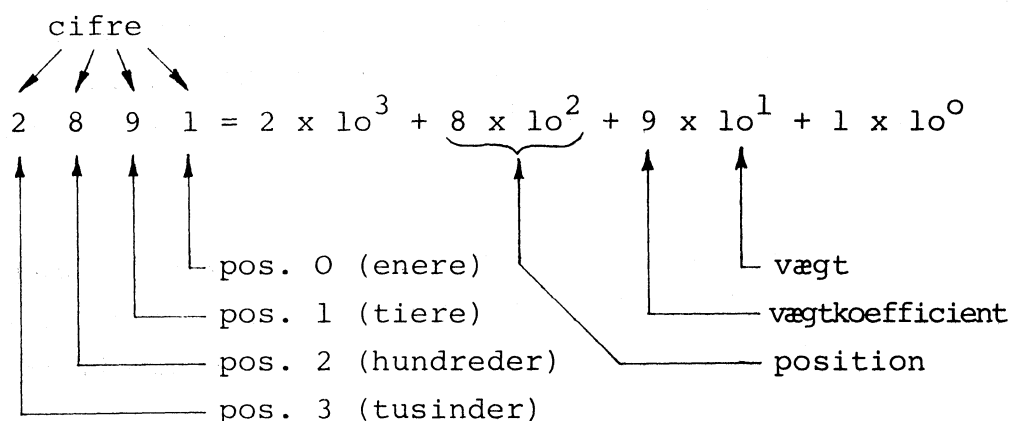


Fig. 1.a.

Positionen er cifferets nummer i cifrerrækken, regnet bagfra, begyndende med nummer 0.

Vægtkoefficienten er cifferets værdi: 0, 1, 2, ..., 8 eller 9.

Vægten er den værdi, der fremkommer, når tallet løses op til den potens, som positionen angiver. For trediebagerste ciffer altså  $10^2 = 100$ . For bagerste ciffer  $10^0 = 1$ .

Et tal opbygget efter dette princip tilhører et positionstalsystem.

## 1.1. Talnotationssystemer

Af vægtkoefficienten i fig. 1.a. ses, at det omtalte talsystem råder over 10 forskellige cifre, og systemet kaldes derfor det decimale talsystem. I datateknikken bruges i udstrakt grad talsystemet med andre antal cifre end netop 10, og man definerer derfor:

Et talsystems base eller grundtal er lig antallet af forskellige cifre, som talsystemet arbejder med.

På steder, hvor der kan opstå tvivl om et tals base, angives denne ved et suffix til tallet. F.eks.

$2891_{10}$ ,  $4711_8$ ,  $4711_{10}$ . Basen selv angives altid som et decimalt tal.

I det følgende omtales tre talsystemer med et antal cifre h.h.v. lig 2 (det binære talsystem) 8 (det oktale talsystem) og 16 (det hexadecimale talsystem).

### 1.1.1. Det binære talsystem

Dette talsystem er uden sammenligning det vigtigste i al EDB-teknik. Det skyldes, at en mængde elektroniske og elektriske komponenter naturligt kan indtage to forskellige stillinger, som defineres som h.h.v. 0 og 1.

Et relæ kan være trukket eller strømløst, en kontakt kan være åben eller sluttet, en transistor kan trække strøm eller ikke trække strøm, en ferritkerne (som er ringformet) kan være magnetiseret den ene eller den anden vej, en lampe kan lyse eller være slukket o.s.v.

Det binære talsystem har altså to forskellige cifre, 0 og 1, og opbygningen af et binært tal sker på sædvanlig måde gældende for et positionstalsystem se fig. 1.1.1.a.

$$\begin{aligned} 11010_2 &= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 16 + 8 + 0 + 2 + 0 \\ &= 26_{10} \end{aligned}$$

Fig. 1.1.1.a.

Figuren angiver endvidere hvorledes omregningen fra det binære - til det decimale talsystem foretages.

Ønskes en talangivelse derimod omskrevet fra decimal til binær notation kan følgende procedure bruges:

Det decimale tal divideres successivt med 2. Hver gang dette sker fremkommer et nyt decimalt tal samt en rest som er enten 0 eller 1.

Restcifrene noteres op, og divisionen med 2 fortsætter på det decimale tal, indtil det er bragt ned på 0. Rækken af restcifre, først udregnede ciffer læst sidst, vil da angive det tilsvarende binære tal.

#### Eksempel:

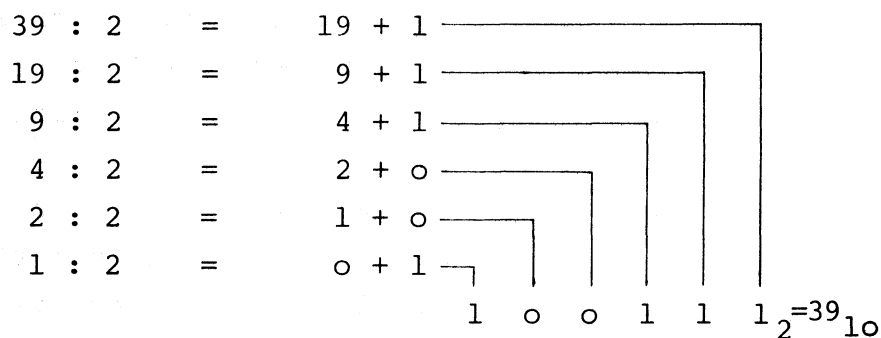
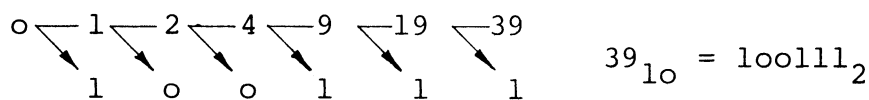


Fig. 1.1.1.b.

Omregningen kan skematiseres:



## 1.1.2. Det hexadecimale talsystem

Dette talsystem har grundtallet 16 og arbejder derfor med 16 forskellige cifre.

En standard har efterhånden udviklet sig, hvorefter disse 16 cifre angives med de sædvanlige 10 cifre samt bogstaverne A, B, C, D, E, F. Herved opnås bl.a. at hexadecimale tal kan skrives af alle sædvanlige EDB-udskrivningsapparater.

Decimal:    0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

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

Fig. 1.1.2.a.

For sammenligning af talsystemer iøvrigt, se fig. 1.1.4.a. side 10.

Fig. 1.1.2.b. viser et eksempel på et hexadecimalt tal og udregning af dets decimale værdi:

$$\begin{aligned} 6A3D_{16} &= 6 \times 16^3 + 10 \times 16^2 + 3 \times 16^1 + 13 \times 16^0 \\ &= 6 \times 4096 + 10 \times 256 + 3 \times 16 + 13 \times 1 \\ &= 27197_{10} \end{aligned}$$

Fig. 1.1.2.b.



### 1.1.3. Det oktale talsystem

Dette talsystem arbejder med 8 forskellige cifre, som f.eks. kan benævnes: 0, 1, 2, 3, 4, 5, 6, 7.

Værdien af et oktalt tal kan udtrykkes decimalt, idet man foretager en omskrivning med følgende eksempel som model:

$$2472_8 = 2 \times 8^3 + 4 \times 8^2 + 7 \times 8^1 + 2 \times 8^0$$

$$= 2 \times 512 + 4 \times 64 + 7 \times 8 + 2 \times 1$$

$$= 1338_{10}$$

Fig. 1.1.3.a.

Sammenhængen mellem talrækken i dette system og det decimale system kan ses på fig. 1.1.4.b.

#### 1.1.4. Omregning af tal med forskellig base

Eksistensberettigelsen af det binære talsystem ligger som tidligere omtalt i det faktum, at mange komponenter i EDB-teknikken kan antage to stabile tilstande.

Men lige så velegnet en række på f.eks. 16 bit er som datarepræsentation i en maskine, lige så uoverskuelig er den samme række at arbejde med for programmører, teknikere og andre, som arbejder omkring maskinen.

Af mnemotekniske grunde laver man derfor tit en omskrivning af det binære tal til enten et oktalt eller et hexadecimalt tal. Omskrivningen sker ved at inddele de binære cifre bagfra i grupper på 3, hvilket giver et oktalt tal, eller på 4, hvilket giver et hexadecimalt tal.

Ved omskrivning til et oktalt tal benytter man sig af en tabel over de oktale cifres binære ækvivalent:

Binær:	ooo	ool	olo	oll	loo	lol	llo	lll
Oktal:	0	1	2	3	4	5	6	7

Ved omskrivning af f.eks. det binære tal

100111010101011

foretages en skematiseret omskrivning:

(oo)1	ool	llo	lol	ool	oll
1	1	6	5	1	3 <sub>8</sub>

100111010101011 = 116513<sub>8</sub>

Den samme omskrivning kan anvendes fra et binært tal til et hexadecimalt tal. Inddelingen sker i grupper på 4, regnet bagfra, og den tilhørende binær-hexadecimale ciffertabel ser således ud:

Binær: 0000 0001 0010 0011 0100 0101 0110 0111

Hexadec.: 0 1 2 3 4 5 6 7

Binær: 1000 1001 1010 1011 1100 1101 1110 1111

Hexadec.: 8 9 A B C D E F

Det foranstående eksempel loolllololoololl, bliver i hexadecimal notation til:

lool 1101 0100 1011

9 D 4 B<sub>16</sub>

Således at man har:

$$\text{loolllololoololl}_2 = 116513_8 = 9D4B_{16} (=40167_{10})$$

En omsætning fra hexadecimal til oktal notation eller omvendt sker nemmest ved at gå omvejen over den binære notation. Som eksempel vises konverteringen af 9A3<sub>16</sub>:

$$\begin{array}{ccccccc} & 9 & & A & & 3 & \\ \hline 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ \hline & 4 & & 6 & & 4 & & 3 & \end{array} \quad 9A3_{16} = 4643_8$$

Brugen af oktal og hexadecimal notation ved talangivelser i forbindelse med EDB-arbejde er til en vis grad betinget af den historiske udvikling.

Tidligere var den oktale notation meget benyttet på grund af dens store lighed med den decimale. En dominans af maskiner med ordlængder på 4, 8 16 24 og 32 bit har imidlertid tilskyndet til en mere udbredt anvendelse af den hexadecimal notation trods de praktiske besværligheder, det giver for mennesker at skulle arbejde med 6 nye cifre.

BINÆR	OKTAL	HEX	DEC.
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15
10000	20	10	16
10001	21	11	17
10010	22	12	18
10011	23	13	19
10100	24	14	20
10101	25	15	21
10110	26	16	22
10111	27	17	23
11000	30	18	24
11001	31	19	25
11010	32	1A	26
11011	33	1B	27
11100	34	1C	28
11101	35	1D	29
11110	36	1E	30
11111	37	1F	31
100000	40	20	32
100001	41	21	33
100010	42	22	34
100011	43	23	35
100100	44	24	36
100101	45	25	37
100110	46	26	38
100111	47	27	39
101000	50	28	40
101001	51	29	41
101010	52	2A	42
101011	53	2B	43
101100	54	2C	44
101101	55	2D	45
101110	56	2E	46
101111	57	2F	47

Fig.1.1.4.a.

### 1.1.5.Brøktal

Den i fig. 1.a. viste fremstilling af et decimalt heltal kan udvides til også at omfatte brøktal, som består af summen af et heltal og en given brøkdel af en talenhed. Dette sker ved at introducere et såkaldt brøktegn (her et sædvanligt decimalpunktum) umiddelbart til højre for cifferet med vægten  $10^0$ , og derefter fortsætte med cifre, som da får vægten  $10^{-1}$ ,  $10^{-2}$ ,  $10^{-3}$ .....:

$$43.27 = 4 \times 10^1 + \underbrace{3 \times 10^0}_{\text{brøktegn}} + 2 \times 10^{-1} + 7 \times 10^{-2}$$

vægt  
vægtkoefficient  
position

Fig. 1.1.5.a.

Definitionen af vægt, position og vægtkoefficient forbliver uændret, og definitionen af et binært brøktal sker på tilsvarende måde:

$$\begin{aligned} 110.01_2 &= 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} \\ &= 4 + 2 + 0 + 0 + 0,25 = 6.25_{10} \end{aligned}$$

Man kan matematisk vise nogle regler om sammenhængen mellem decimale og binære brøker:

1. Visse decimalbrøker kan ikke omskrives til binær form således, at værdien forbliver eksakt den samme. F.eks. 0.1, 0.2, 0.3, 0.4. Men nok 0.5, 0.75.
2. Enhver binær brøk kan eksakt omskrives til en decimalbrøk.
3. Ved omskrivning af en binær brøk vil den decimale brøk få lige så mange betydende cifre som den binære.

Ved omskrivning fra decimalbrøk til binær brøk kan man omskrive heltalsdelen og den rene brøkdel hver for sig, idet tallet jo består af summen af disse to dele uafhængigt af talsystemet. Ved omskrivning af den rene brøkdel kan følgende procedure anvendes:

Brøkdelen, som har 0 foran brøktegnet, ganges med 2 og cifferet foran brøktegnet (0 eller 1) noteres ned som det første ciffer efter brøktegnet i den tilsvarende binære brøk.

Hvis cifferet er 1 trækkes det fra, således at der fremkommer en ny brøkdel. Denne ganges igen med 2 og cifferet foran brøktegnet noteres ned som det næste ciffer i den binære brøk.

Således fortsættes indtil brøkdelen enten bliver 0 (og der således er foretaget en eksakt omskrivning) eller binærbrøken er angivet med det ønskede antal cifre, eller binærbrøken bliver periodisk (i begge tilfælde er konverteringen ikke eksakt).

Proceduren er vist på fig 1.1.5.b. Det decimale brøktal  $t = 0,675$  skal konverteres til binær form:

0.675	Oprindeligt brøkdel.
<u>1</u> .35	Gang med 2 og gem 1.
0.35	Træk 1 fra.
<u>0</u> .7	Gang med 2 og gem 0. Intet 1-tal at trække fra.
<u>1</u> .4	Gang med 2 og gem 1.
0.4	Træk 1 fra
<u>0</u> .8	Gang med 2 og gem 0.
<u>1</u> .6	Gang med 2 og gem 1.
0.6	Træk 1 fra.
<u>1</u> .2	Gang med 2 og gem 1.
0.2	Træk 1 fra
<u>0</u> .4	Gang med 2 og gem 0. Tallet er nu det samme som i linie 6, og de følgende binære cifre vil repetere sig med en periode på 4.

$$t = 0.675_{10} = 0.10101010_2$$

Fig.1.1.5b.

### 1. 1. 6. Flydende tal

En almindelig forekommende måde at angive tal på, som f.eks. kendes fra visse lommeregnerne er vist i følgende eksempel:

3.2764 o8

Det komplette taludtryk som disse to tal fremstiller, er følgende:

$3.2764 \times 10^8$  som også har skriveformen:  
3.2764 E o8

Andre skriveformer for den samme talstørrelse er:

o.32764 E o9 svarende til  $0.32764 \times 10^9$   
32764.o E o4 svarende til  $32764.0 \times 10^4$

Det første af tallene kaldes mantissen, det sidste eksponenten.

Et tal angivet på denne form, hvor et ciffer på en bestemt position giver forskellig bidrag til tallets størrelse, afhængig af eksponentens størrelse, kaldes et flydende tal.

Begrebet defineres på samme måde i binær notation.

Eksempel:

11.010011 E 1011 svarende til  $11.010011_2 \times 10_2^{1011}$

hvilket omregnes til:

$1101001100000_2 (= 6752_{10})$



Af udtrykket for et flydende tal kan man ikke umiddelbart se, om selve tallet er et heltal eller det også har en brøkdelt.

Eksempel:

11.010011 E 1011 svarer til 1101001100000 (=  $6752_{10}$ )

11.010011 E 0011 svarer til 11010.011 (=  $26.375_{10}$ )

Omskrivningen af et flydende binært tal til decimalform, sker nemmest ved først at omskrive det binære tal således, at dets eksponentdel bliver 0, og derefter omskrive det fremkomne brøktal på sædvanlig måde.

#### 1.1.6.1. Flydende tal på normaliseret form

Dette begreb defineres for både decimale og binære flydende tal:

For et flydende tal på normaliseret form gælder det, at mantissen ligger mellem 0 og 1, og at første ciffer efter brøktegnet ikke er 0.

Eksempler:

$0.1101_2$  E 0100 svarende til  $13_{10}$   
 $0.1000_2$  E 0000 svarende til  $0.5_{10}$   
 $0.7623_{10}$  E 69 svarende til  $0.7623 \times 10^{69}$   
 $0.1300_{10}$  E 08 svarende til  $13000000$

Introduktionen af et flydende tal på normaliseret form sker af hensyn til regneenheden i et EDB-anlæg. Denne er konstrueret således, at optimal nøjagtighed ved regneoperationer kun opnås, hvis de indgående talstørrelser er angivet på denne form.

Omskrivningen af et decimalt tal til et normaliseret flydende binært tal sker ved først at omskrive til et sædvanligt binært tal. Derefter forskydes brøktegnet indtil de for en normalisering gældende regler er opfyldt, og den tilhørende potens af 2 tilføjes..

Eksempel:

$$t = 6.75_{10} = 110.11_2$$

Brøktegnet i det binære tal forskydes tre pladser til venstre, hvorved tallet bliver reduceret til  $\frac{1}{8}$  eller  $2^{-3}$ . Derfor tilføjes potensen  $2^3$ :

$$t = 6.75_{10} = 110.11_2 = 0.11011_2 \quad E \ 11$$

## 1. 2. BINÆR ARITMETIK

Enhver talbehandlingsopgave kan ved omskrivninger og opsplætning i delopgaver reduceres til en kombination af de fire grundlæggende regnearter: addition, subtraktion, multiplikation, division.

De principper, som regnearterne arbejder efter, stiller ikke andre krav til de indgående tal, end at de er elementer i samme positionstalsystem, og regneprocedurerne vil derfor i det væsentlige være ens for decimale og binære tal.

I det følgende vil disse fire regnearter blive gennemgået.

### 1. 2. 1. De fire regnearter

Ved addition af to binære cifre gælder følgende regler:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 10$$

Når binære tal skal adderes er det, som for decimal addition, vigtigt, at tallene er rettet ind til højre. Hvis der ved additionen af cifre med en given position fremkommer et flercifret tal, skrives bagerste ciffer mens øvrige cifre føres som mente over i kolonnen umiddelbart til venstre.

Eksempel:

$$\begin{array}{r} \text{11} \\ \text{ool1} = 3 \end{array}$$

$$\begin{array}{r} \text{ol1o} = 6 \end{array}$$

$$\begin{array}{r} \text{lool} = 9 \end{array}$$

$$\begin{array}{r} \text{11111} \\ \text{11oloololl} = 843 \end{array}$$

$$\begin{array}{r} \text{lool1ol} = 77 \end{array}$$

$$\begin{array}{r} \text{111lool1ooo} = 920 \end{array}$$

Ved multiplikation gælder følgende "lille tabel":

	o	1
o	o	o
1	o	1

og en multiplikation følger samme mønster som for decimale tal:

$$6 \times 3:$$

$$\begin{array}{r} \text{1lo} \times \text{11} \\ \hline \end{array}$$

$$\text{1lo}$$

$$\text{1lo}$$

$$\text{loolo} = 18$$

$$25 \times 11:$$

$$\begin{array}{r} \text{1lool} \times \text{lo11} \\ \hline \end{array}$$

$$\text{1lool}$$

$$\text{1lool}$$

$$\text{ooooo}$$

$$\text{1lool}$$

$$\text{looollool1} = 275$$

Ved subtraktion kan det komme på tale at "låne" en cifferenhed fra en mere betydende kolonne for at fuldføre beregningen:

$$\text{11oll} = 27$$

$$- \text{lool} = -9$$

$$\text{loolo} = 18$$

$$\text{loolollo} = 150$$

$$- \text{lo11ol} = -45$$

$$\text{11olool} = 105$$

Hvis minuenden er mindre end subtrahenden laver man den omvendte subtraktion og sætter et minus foran resultatet:

$$\begin{array}{rcl} 110110 & = & 54 \\ - 1100101 & = & -101 \\ - 0101111 & = & - 47 \end{array}$$

Ved division bruges den sædvanlige skematiserede form:

$$\begin{array}{r} \underline{10110} \overline{) 110001111} \quad \underline{10010} \\ \underline{10110} \\ 00101 \\ \underline{00000} \\ 1011 \\ \underline{0000} \\ 10111 \\ \underline{10110} \\ 11 \\ \underline{00} \\ \text{rest } 11 \end{array}$$

$$\text{d.v.s. } \frac{110001111}{10110} = 10010 \frac{11}{10110}$$

$$\text{eller omskrevet til decimal tal: } \frac{399}{22} = 18 \frac{3}{22}$$

## 1. 2. 2. Talangivelse på komplement form

Når tal skrives i det decimale talsystem er det almindeligt at skrive et plus eller et minus tegn foran som indikation af, om tallet er positivt eller negativt.

Ved repræsentation af et negativt tal skrevet binært, anvendes ofte tallets numerisk værdi skrevet på komplement form, enten som 1-komplement eller 2-komplement

## 1-komplement

For 1-komplement tal sker dannelsen af et negativt tal ved at lade de enkelte bit i repræsentationen af det tilsvarende positive tal (incl. fortegnsbitten) skifte til den modsatte værdi (kaldes også invertering, negering):

0000011	= + 3	0010110	= + 22
1111100	= - 3	1110101	= - 22
0011	= + 3	0000000	= + 0
1100	= - 3	1111111	= - 0

Begrebet komplementering skal altid ses i sammenhæng med en given ordlængde, d.v.s. et konstant antal bit, som står til rådighed for repræsentationen af et tal. Iøvrigt forekommer her ulempen med to repræsentationer for tallet 0.

## 2-komplement

For 2-komplement sker dannelsen af et negativt tal ved til det tilhørende 1-komplement, at addere + 1. En tabel over 3 bit tal i 1-komplement form ser således ud:

Decimal	-3	-2	-1	-0	+0	+1	+2	+3
1-komp.	100	101	110	111	000	001	010	011

- og for 2-komplement:

Decimal	-4	-3	-2	-1	0	1	2	3
2-komp.	100	101	110	111	000	001	010	011

Man ser, at de negative 2-komplementtal er fremkommet ved at forskyde de negative 1-komplementtal een position i retning mod 0-værdien. Herved forsvinder bekvemt den ene repræsentation for 0 og i den anden ende fremkommer en tom plads, som passende kan tilordnes værdien -4.

## Positive - negative tal

Reglerne for komplementering af talstørrelser kan bruges uændret ved komplementering af negative tal, hvorved de tilsvarende positive tal fremkommer.

Eksempel:

$$\begin{array}{rcl} 1101 & = & -3 \quad \text{i 2-komplement} \\ 0010 & & \\ + \quad \underline{1} & & \\ \hline 0011 & = & +3 \quad \text{i 2-komplement} \end{array}$$

og

$$\begin{array}{rcl} 1100 & = & -3 \quad \text{i 1-komplement} \\ 0011 & = & +3 \quad \text{i 1-komplement} \end{array}$$

I det decimale talsystem eksisterer begreber svarende til 1-og 2-komplement, her kaldes de 9- og 10-komplement.

9-komplementet til et decimalt tal fås ved at tage hvert af dets cifre og trække fra 9. F.eks. er 9-komplementet til 238 lig 761.

10-komplementet findes tilsvarende ved at addere +1 til 9-komplementet.



### 1. 2. 3. Addition og subtraktion af komplementtal

Regnearten subtraktion kan principielt omskrives til en addition:

$$\text{diff.} = a - b = a + (-b)$$

Denne omskrivning bruges altid i regneenheden i en datamaskine, når en subtraktion skal udføres. Regneenheden kan herved nøjes med at skulle udføre processerne addition og komplementering. Eftersom multiplikation og division kan omskrives til en række af additioner og subtraktioner, kan nu alle 4 regnearter udføres af dette simple regneværk.

Ved beregninger, hvor både de indgående størrelser og resultatet holder sig indenfor det til ordlængden svarende talområde (overholdelsen af dette er brugerens ansvar), vil resultatet fremkomme i et resultatregister med sædvanlig ordlængde. Undertiden kan beregninger give en eller flere ekstra bit foran registeret, se følgende eksempler.

Visse maskiner udnytter disse ekstra bit til at detektere resultater, som ligger udenfor maskinens talområde. Forskellige reaktioner herpå kan tænkes, lige fra en fejludskrift til en afbrydelse af programudførelsen.

I det følgende vises nogle eksempler. De afspejler det faktum, at praktisk taget alle EDB-anlæg regner i 2-komplement:

Af 00010000 = +16 fås

11101111

+ 1

11110000 = -16

og fx subtraktionen 22 - 16 bliver da:

00010110 = +22

+ 11110000 = -16

1 00000110 = + 6

idet menten foran mest betydende ciffer bortkastes.

Af 00010110 = +22 fås

11101001

+ 1

11101010 = -22

og subtraktionen 16 - 22 bliver da:

00010000 = +16

+ 11101010 = -22

11111010 = - 6 idet

00000110 = + 6 giver

11111001

+ 1

11111010 = - 6

### 1. 3. TALREPRÆSENTATION I DATAMASKINER

Talrepræsentation i en datamaskine er meget tæt knyttet til en fundamental størrelse i maskinen, som kaldes dens ordlængde.

Som tidligere omtalt er mange komponenter i en datamaskine indrettet til at kunne indtage to stabile tilstande, som tilordnes de binære værdier 0 og 1.

Datamaskiner må være konstrueret til at behandle tal indenfor et rimeligt stort område og med en rimelig stor nøjagtighed, og dette realiseres i praksis ved at koble et antal af disse binært arbejdende elementer sammen i serie.

Der vælges et standard antal elementer, som da forekommer som en enhed overalt i maskinen, hvor tal skal behandles eller lagres. Man taler om, at maskinen har en given ordlængde, som altså er det antal binære cifre eller bit, som maskinen behandler som en enhed.

Minidatamater har hyppigt en ordlængde på 16 bit, andre ordlængder er 24 bit (RC 4000), 32 bit (IBM maskiner), samt for mikrodatamater hyppigt 4 og 8 (og 16) bit.

### 1.3.1. Repræsentation af positive og negative heltal

Positive binære heltal repræsenteres i en datamaskine ved at de enkelte bit i datamaskinens ord tilordnes en vægt.

Er ordlængden f.eks. 16 bit får man:

bit nr.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
binær vægt	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

således, at der i et sådant ord kan repræsenteres tal i området:

$$2^{16} - 1 > \text{tal} > 0 \quad \text{eller} \quad 65535 > \text{tal} > 0$$

Skal maskinen også kunne behandle negative tal, vælger man repræsentationsformer, som forskyder dette talområde, så det ligger symmetrisk omkring 0. I det følgende skal gennemgås tre sådanne almindeligt kendte former for repræsentation af positive og negative binære tal:

1. Størrelse- og- fortegn.
2. 1-komplement.
3. 2-komplement.

Den første form kan bruges til alle talsystemer, mens de to sidste forudsætter brug af et positionstalsystem.

I kap. 1.1. og 1.2. har størrelse-ogfortegn metoden været enerådende. I datamaskineteknik realiseres den bekvemt ved at reservere en bit til angivelse af fortegnet, f.eks. 0 for positiv og 1 for negativ. Med 16 bit kan således repræsenteres tal i området:

$$\begin{aligned} -2^{15} - 1 < \text{tal} < +2^{15} - 1 \text{ eller} \\ -32767 < \text{tal} < +32767 \end{aligned}$$

For tallet 0 eksisterer to repræsentationer:

1000 0000 0000 0000 og 0000 0000 0000 0000

hvad der i regnemekanismer er en ulempe. Repræsentation af negative tal ved størrelse-ogfortegn er mest brugt udenfor egentlige EDB-anlæg, f.eks. i måleinstrumenter og cifferdisplays.

I talrepræsentationer, hvor negative tal repræsenteres ved en komplementform, reserveres også en bit (den mest betydende) til fortegnsangivelse. Herved bliver det numeriske talområde tilsvarende formindsket til det halve.

Ved repræsentation af tal i komplementform angives et positivt tal med 0 i fortegnsbitten, et negativt tal med 1.

Eksempel på talrepræsentation i en 8-bit maskine hvor negative tal angives i 2-komplement.

Fortegn →

01111111 = +127

00000000 = 0

11111111 = - 1

10000000 = -128

### 1.3.2.Repræsentation af flydende tal

Repræsentation af flydende tal i en datamaskine sker altid i normaliseret form. Regneværket foretager derfor altid som afslutning på en regneoperation en omskrivning af resultatet således, at mantissen ligger mellem 0 og 1, og at første ciffer efter brøktegnet ikke er 0.

Repræsentationsformen er lagt ind i maskinen ved dens konstruktion og er således lige så uforanderlig som f.eks. ordlængden. Den består i, at et antal bit er tilordnet mantissedelen, som er bestemmende for tallets nøjagtighed (dvs. antal betydende cifre) og et andet antal er tilordnet eksponentdelen, som er bestemmende for tallets størrelse.

Ofte reserverer man flere ord til et flydende tal, f.eks. i 8 bit maskiner - 3 ord, og i 16 bit maskiner - 2 ord pr. tal. Summen af bit i de to dele er dermed konstant, og der er derfor ved udviklingen af datamaskinen gjort mange overvejelser over, hvordan fordelingen af bit skal være i de to dele.

Som eksempel kan anføres, at i en HP 2100 minidatamat med 16 bit ordlængde er der afsat 23 bit + fortegn til mantissen og 7 bit + fortegn til eksponenten. Det giver et talområde for maskinen fra ca.  $10^{-38}$  til  $10^{+38}$  og en nøjagtighed på ca. 7 decimale cifre. Formatet er vist i fig. 1.3.2.a.

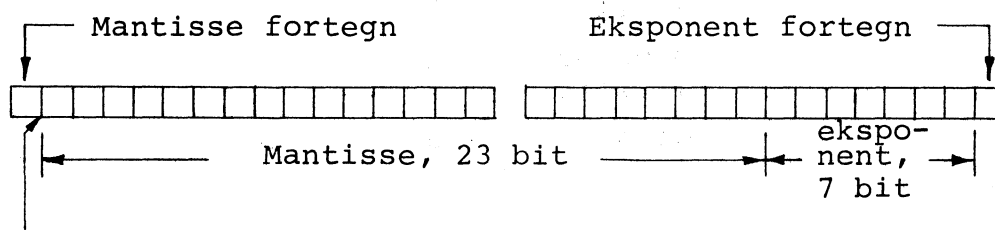


Fig. 1.3.2.a.

Da heltal og flydende tal repræsenteres på helt forskellig måde i maskinen, er det uomgængelig nødvendigt, at brugeren gør sig dette klart og tager hensyn til det i sine programmer.

Normalt er datamaskinerne udrustet med konverteringsrutiner, som brugeren kan benytte sig af, hvis han vil lade maskinen arbejde med blandede udtryk.

Visse maskiner er dog i stand til at foretage konverteringen automatisk uden at brugeren mærker det. Konverteringen skaber ingen problemer ved overgang fra heltal til flydende tal, men man må være opmærksom på, at ved konvertering den modsatte vej forsvinder eventuelle brøkcifre.

### 1.3.3.Repræsentation af tal med dobbelt præcision

Visse maskiner er udrustet med faciliteter, som tillader behandling af tal med såkaldt dobbelt præcision. Dette begreb dækker i praksis over forskellige udvidelser af maskinens talområde og præcision, og forbindelsen til udtrykket "dobbelt præcision" er ikke altid lige gennemskuelig.

For heltal gælder hyppigt, at den afsatte plads udvides fra eet ord til to. Herved forandres tallets præcision (absolutte nøjagtighed) ikke, men dets område udvides til det mangedobbelte.

For flydende tal udvides repræsentationen ligeledes med et ord eller måske to. De ekstra bit fordeles mellem mantisse og eksponent, og her kan man bedre tale om forøget nøjagtighed, samtidig med forøget talområde. Regning i dobbelt præcision kræver forøget regnetid og bør derfor begrænses til det strengt nødvendige.

# 1. 4. ØVELSER VEDRØRENDE TALSYSTEMER

## OPGAVE 1.1.

Omskrivning af tal med forskellige baser.

Udfyld skemaet med tallene omskrevet til de øvrige notationer.

binær	oktal	decimal	hexadecimal
1	1	1	1
10	2	2	2
100	4	4	4
1010	12	10	A
10000	20	16	10
11100	34	28	1C
1100101			
1100101101	1455	813	32D
010110	26	22	16
10001000	100	64	40
	377		
10101110	256	174	AE
	1000		
1000010100	1024	276	214
	1733		
1020	12	10	A
1111	17	15	F
111111	77	63	3F
		82	
110010	62	100	32
100010001000	2000	1024	400
11101010100	3524	1876	754
			10
1111	17	15	F
111001	71	55	37
10101010	252	170	AA
			100
1110100111	1647	935	3A7
1111111111	7777	4095	FFF



# 1. 4. ØVELSER VEDRØRENDE TALSYSTEMER

## OPGAVE 1.1.

Omskrivning af tal med forskellige baser.

Udfyld skemaet med tallene omskrevet til de øvrige notationer.

binær	oktal	decimal	hexadecimal
1	1	1	1
10	2	2	2
100	4	4	4
1010	12	10	A
10000	20	16	10
11100	34	28	1C
1100101	145	101	65
1100101101	1455	813	32D
10110	26	22	16
1000000	100	64	40
1111111	377	255	FF
10101110	256	174	AE
1000000000	1000	512	200
1000010100	1024	532	214
1111011011	1733	987	3DB
1010	12	10	A
1111	17	15	F
11111	77	63	3F
1010010	122	82	52
1100100	144	100	64
10000000000	2000	1024	400
11101010100	3524	1876	754
10000	20	16	10
1111	17	15	F
110111	67	55	37
10101010	252	170	AA
100000000	400	256	100
1110100111	1647	935	3A7
11111111111	7777	4095	FFF

## OPGAVE 1.2

### Addition og subtraktion af binære 2-komplement tal

Udfør følgende additioner og angiv decimale ækvivalenter:

$$\begin{array}{r} 001011 \quad 11 \\ + 010010 \quad 18 \\ \hline 011101 \quad 29 \end{array}$$

$$\begin{array}{r} 011010 \quad 26 \\ + 011011 \quad 27 \\ \hline 110101 \quad 53 \end{array}$$

$$\begin{array}{r} 00111010 \quad 58 \\ + 00101111 \quad 47 \\ \hline 01101001 \quad 105 \end{array}$$

Vis hvordan følgende beregninger sker i 8 bit dataord:

$$\begin{array}{r} 26_{10} + 11_{10} = 37_{10} \\ \begin{array}{r} 011010 \\ 001011 \\ \hline 100101 \end{array} \end{array}$$

$$\begin{array}{r} 11_{10} - 26_{10} = -15_{10} \\ \begin{array}{r} 001011 \quad 11 \\ 100110 \quad 26 \\ \hline 110001 \quad -15 \end{array} \end{array}$$

$$\begin{array}{r} -34_{10} - 53_{10} = -87_{10} \\ \begin{array}{r} 11011110 \\ 11001011 \\ \hline 10101001 \quad - \\ 01010111 \quad 87 \end{array} \end{array}$$

$$\begin{array}{r} 26_{10} - 11_{10} = 15_{10} \\ \begin{array}{r} 011010 \\ 110101 \\ \hline 001111 \end{array} \end{array}$$

$$\begin{array}{r} 90_{10} - 56_{10} = 34_{10} \\ \begin{array}{r} 01011010 \\ 11101000 \\ \hline 01000010 \end{array} \end{array}$$

$$\begin{array}{r} 96_{10} - 56_{10} = 40_{10} \\ \begin{array}{r} 01100000 \\ 11101000 \\ \hline 01001000 \end{array} \end{array}$$

### OPGAVE 1.3

Addition og subtraktion af oktale og hexadecimale  
2-komplement tal.

$$26_8 + 11_8 = \underline{\hspace{2cm}}$$

$$6A_{16} - 42_{16} = \underline{\hspace{2cm}}$$

$$160_8 - 32_8 = \underline{\hspace{2cm}}$$

$$-36_{16} - 28_{16} = \underline{\hspace{2cm}}$$

$$26_8 - 11_8 = \underline{\hspace{2cm}}$$

$$6F_{16} - 80_{16} = \underline{\hspace{2cm}}$$

$$-46_8 - 125_8 = \underline{\hspace{2cm}}$$

$$6A_{16} + 42_{16} = \underline{\hspace{2cm}}$$

## 2. Datamaskinens hovedenheder

### 2.1. Generel blokmodel

#### 2.1.1. Datamaskinesystemet

Et datamaskinesystem består af to grundelementer:

- 1) Materiellet (den fysiske, af elektroniske og mekaniske elementer opbyggede datamaskine; på engelsk benævnt "hardware")
- 2) Programmellet (et sæt af færdige brugerprogrammer, på engelsk benævnt "software").

I de følgende afsnit 2-5 skal vi nærmere beskrive materiellets principielle opbygning og funktion, men for at få indblik i hele datamaskinesystemet skal her kort nævnes de vigtigste bestanddele af programmellet:

Programmellet kan opdeles i følgende kategorier:

- a) Ladeprogrammer, som tjener til at indlæse binære programmer (maskinprogrammer) til datamaskinens lager.
- b) Oversætterprogrammer (kompilere, kompileringsprogrammer), som tjener til at oversætte et program skrevet i et kildesprog (f.eks. Algol) til maskinprogram.
- c) Retteprogrammer, til retning og afprøvning af brugerudarbejdede programmer.
- d) Subrutinebibliotek, dvs. færdige, ofte anvendte, brugerrutiner (f.eks. multiplikations- og divisionsrutiner).
- e) Diagnostikprogrammer, som tjener til at finde fejl i materiellet (elektronikken), således at fejlen kan udbedres.
- f) Driftssystem (operativ system), et program som letter den daglige drift af datamaskinesystemet.

Den foregående korte opremsning af programmelbestanddelene skulle belyse, at materiellet kun er den synlige del af det "isbjerg", som vi kalder datamaskinesystemet.

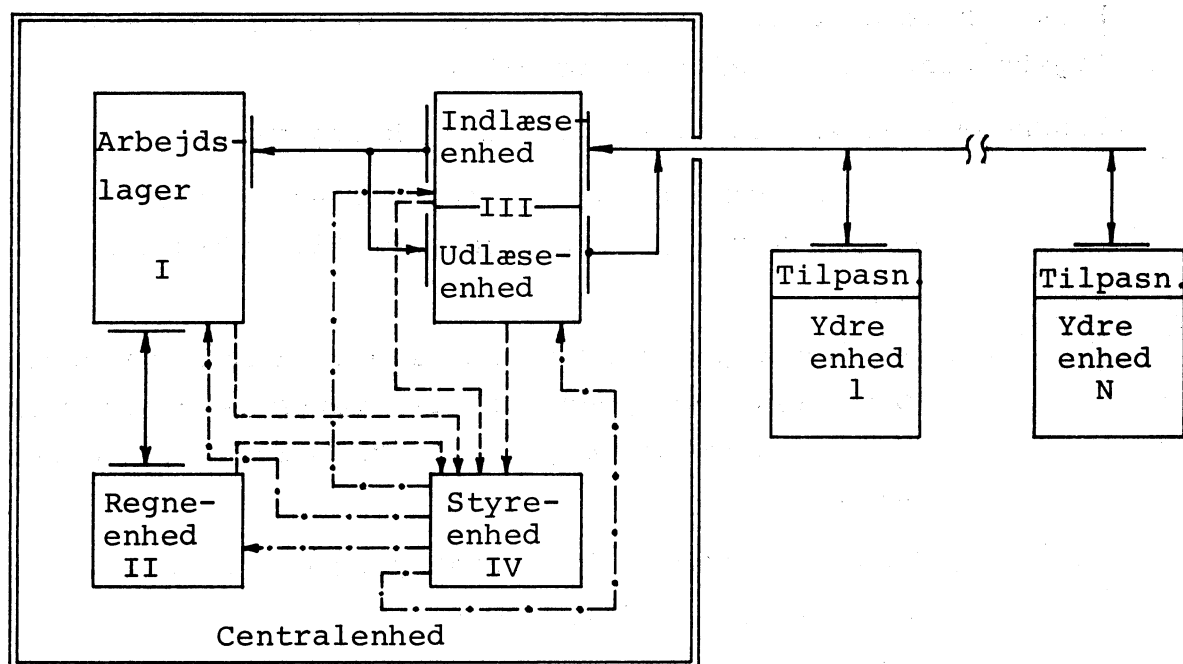
### 2.1.2. Generel blokmodel for materiellet (den fysiske datamaskine).

Maskinellet består af

- 1) centralenheden
- 2) ydre enheder.

De ydre enheder og deres funktion beskrives i kap. 4 og 5.

I det følgende koncentrerer vi os om centralenheden. Denne ses ofte repræsenteret ved et blokdiagram som vist i fig. 2.1.



Signaturer:

- | ———> | Datatransporter
- > Følelinier
- .....> Styrelinier

Fig. 2.1.

Centralenheden er repræsenteret af 4 funktionsblokke:

- I Arbejdslager
- II Regneenhed
- III Indlæse/udlæseenhed
- IV Styreenhed

De tre første blokke modsvarer datamaskinens væsentlige operationer:

- at modtage data (via indlæse enhed)
- at afsende data (via udlæse " ")
- at gemme data (i arbejdslager)
- at omforme (bearbejde) data (i regneenhed).

Den fjerde funktionsblok, styreenheden, er hjernen i systemet og sørger for kommunikationen imellem de 3 øvrige funktionsblokke.

### 2.1.3. Datatransportveje mellem funktionsblokkene.

Ovenfor har vi flere gange anvendt ordet "data" uden at klargøre, hvad der hermed menes.

Definition: Ved data forstås en formaliseret repræsentation af kendsgerninger (f.eks. tal, bogstaver, tegn) eller forestillinger på en sådan form, at den kan kommunikeres eller omformes ved en eller anden proces.

De på fig. 2.1 viste groft markerede pile (→) angiver centralenhedens datatransportveje, dvs. de kanaler over hvilke data flyttes mellem de forskellige funktionsenheder, styret af styreenheden IV.

Datatransporten mellem blokkene styres altså af styreenheden. Fysisk sker dette ved, at styreenheden over styrelinierne (-→) åbner eller lukker "portene" til funktionsenhederne.

Hvilke porte, der skal åbnes, og hvornår bestemmes af styreenheden, der via følelinierne (-→) aftaster funktionsblokkenes "tilstand".

I de følgende afsnit beskrives de enkelte funktionsblokke i fig. 2.1.

## 2.2. Lagerenheden

### 2.2.1. Principiel opbygning af funktionsblokken (systemkomponenten) "lagerenheden"

Funktionsblokken "lagerenheden" består af

- 1) selve lageret med afkoder
- 2) lagerelektronikken bestående af lageradresseregister (MAR) og lagerbufferregister (MBR)

Et principskema for lagerenheden er vist i fig. 2.2.

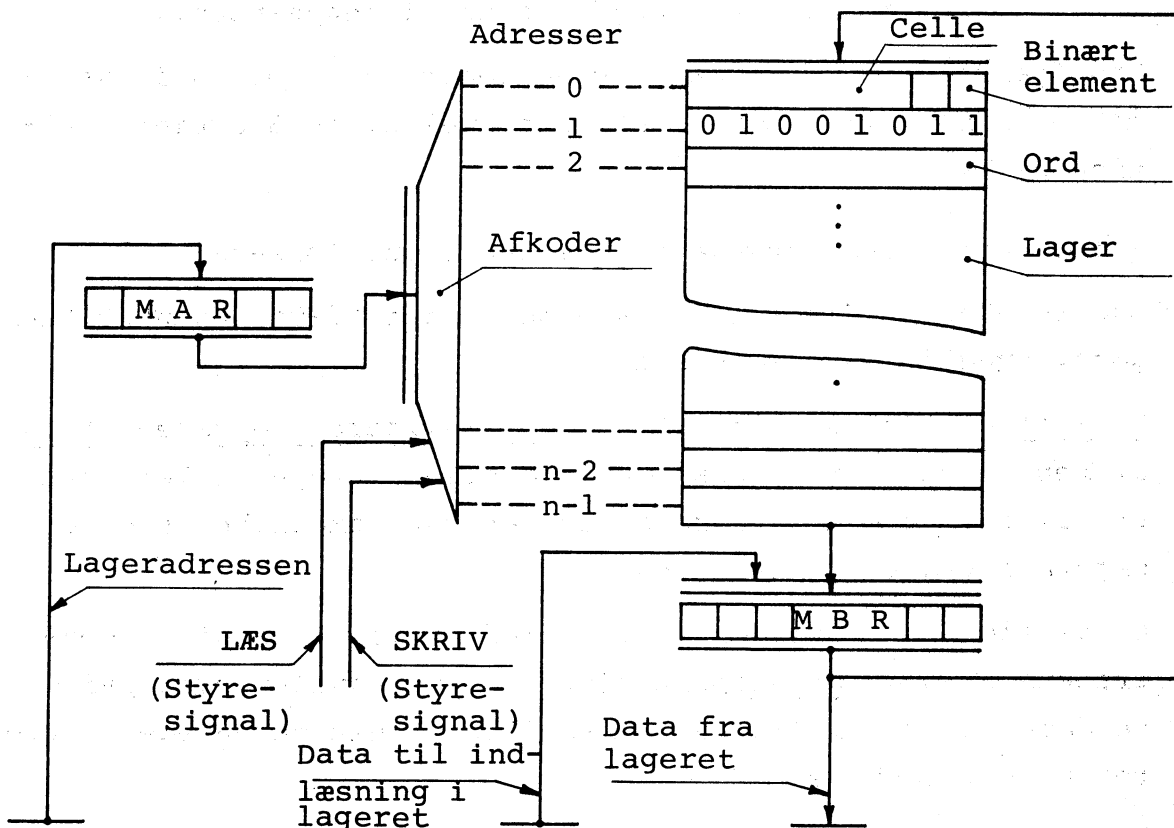


Fig. 2.2.

Selve lageret kan opfattes som en (veldig høj) bogreol. Hyl-derne benævnes celler. En celle identificeres ved en såkaldt adresse (hyldenummer). Indholdet af en celle benævnes et ord. Antal bit i et ord benævnes lagerets ordlængde. Typiske ordlængder er

- 8 bit for mikrodaturer
- 16 bit for minidaturer
- 32-64 bit for større dataturer.

Ordlængden angiver det antal bit, som på en gang kan indlæses i lageret (i den celle som adressen udpeger) eller udlæses fra lageret.

Ordlængden er en for lageret karakteristisk enhed; den anvendes ved angivelse af et arbejdslagers størrelse, idet lagerstørrelsen ofte angives i antallet af ord, som lageret kan indeholde (f. eks. 16 k ord, ordlængde 16 bit).

Hvis lagerets størrelse angives i antal ord, må ordlængden angives, for at man kan foretage sammenligninger med andre datamatsystemer, idet ordlængden jo kan variere fra fabrikat til fabrikat. Af denne grund ser man ofte lagerstørrelsen opgivet i antal 8-bit oktetter (bytes) (f.eks. 32k bytes). Der kan rummes 2 oktetter i et 16-bit ord.

Arbejdslageret bruges til at lagre

centralenhedens program

og

centralenhedens data.

Svarende hertil kan et ord repræsentere

en ordre

eller

et tegn (tal eller bogstav i dataformat).

Hvorledes datamaskinen skelner ordrer fra tegn, skal vi senere vende tilbage til.

### 2.2.2. Lagerenhedens principielle funktionsmåde

Læsning fra lageret sker som en sekvens af følgende operationer (læsecyklus):

1. Lageradressen overføres til lageradresseregistret (MAR).

Kommentar: Ved lageradressen forstås et bitmønster svarende til nummeret på den celle, hvis indhold ønskes udlæst. Udgangen af MAR afkodes af lagerafkoderen, som altså udvælger og aktiverer den celle, hvis adresse står placeret i MAR.



## 2. Styresignalet LÆS aktiveres.

Kommentar: Ved at styresignalet LÆS aktiveres, placeres det udvalgte ord (indholdet i den adresserede celle) i lagerbufferregisteret MBR efter en for lageret karakteristisk tid, tilgangs- eller access-tiden.

Skrivning til lageret sker som en sekvens af følgende operationer (skrivecyklus):

- 1: Data til indlæsning overføres til lagerbufferregisteret MBR.
- 2: Lageradressen overføres til lageradresseregisteret MAR.

Kommentar: Lageradressen angiver den celle i lageret, vi vil indskrive data i (fra MBR). Se iøvrigt kommentar under læsning 1.

- 3: Styresignalet SKRIV aktiveres.

Kommentar: Ved at styresignalet SKRIV aktiveres placeres - efter en for lageret karakteristisk tid "skrivetiden" - indholdet af MBR (det ord som skal indskrives) i den celle, som indholdet i MAR + afkoderen har udvalgt og aktiveret.

For et kernelager (se nedenfor) gælder specielt, at læsning fra lageret er destruktiv, dvs. den celle, hvorfra udlæsning sker, bliver nul-stillet. Derfor må for et kernelager en læsecyklus altid efterfølges af en skrivecyklus med uforandret adresseregister og med lagerbufferregisteret indeholdende det netop udlæste ord, for at den udlæste celle kan få genindsat sit oprindelige indhold.

Indskrivning af nye data må også foregå af en læsecyklus for at nulstille cellen, inden indskrivningen foregår.

For kernelagre er det derfor den kombinerede læse- og skrive-tid, der opgives som den for lageret karakteristiske størrelse cyklustiden.

Generelt gælder, at cyklustiden er den minimale tid, der skal gå imellem 2 successive accesser til samme lagercelle. Cyklustiden er således med til at bestemme lagerets arbejdshastighed.

### 2.2.3. Lagertyper

Lagrene kan opdeles i følgende typer:

- 1) Skrive-læselagre (RAM)
- 2) Læselagre (ROM)
- 3) Programmérbare læselagre (PROM, EPROM).

### 2.2.4. Skrive- læselagre

Et skrive-læselager er et lager, hvori der kan indskrives ord til den af adresseregisteret udvalgte celle, og hvorfra der kan udlæses ord fra den af adresseregisteret udvalgte celle, således som beskrevet i afsnit 2.2.2.

Betegnelsen RAM står for Random Access Memory, dvs. et lager, hvor hver celle - udvalgt ved sin adresse - er tilgængelig for såvel læsning som skrivning.

Fysisk kan et skrive-læselager (RAM) være opbygget som kernelager, hvor hvert binært element i lageret (se fig. 2.2.) er en elektrisk magnetisérbar ferritkerne, eller som halvlederlager, hvori de binære elementer er flip-flops opbygget ved integreret MOS-halvleder teknik.

Indtil for få år siden var kernelageret helt enerådende som arbejdslogersystemkomponent på grund af demonstreret pålidelighed og økonomi, men nu er de integrerede halvlederlagre konkurrencedygtige og anvendes mere og mere.

Kernelagerets fordele:

- informationen tabes ikke ved forsyningsspændingsbortfald, idet de små magnetkerner beholder magnetiseringen
- demonstreret høj pålidelighed

#### Kernelagerets ulemper:

- volumen og vægt; kernelageret fylder langt mere end et halvlederlager af tilsvarende størrelse
- stort effektforbrug
- relativt langsomt (cyklustid  $\approx 1\mu s$ )
- hver lageroperation kræver både en læse- og en skrivefase.

#### Halvlederlagerets fordele:

- lille volumen og vægt
- lille effektforbrug
- cyklustid (0,1 - 0,7 $\mu s$ )

#### Halvlederlagerets ulemper:

- den lagrede information fortabes ved forsynings-spændingsbortfald.

Blandt halvlederlagre skelnes mellem statiske og dynamiske halvlederlagre. De statiske halvlederlagre (RAM) består af flip-flops, hvori informationen forbliver lagret, indtil vedkommende flip-flop adresseres og "overskrives" med ny information.

De dynamiske halvlederlagre (RAM benytter - i stedet for flip-flops - interne kondensatorer til informationslagring. Da disse kondensatorers lækstrømme ikke er uendelig små, må deres afladningstab med millisekunders mellemrum udlignes. Denne proces kaldes "opfriskning" (refresh) og hertil behøves ekstra elektronik (materiel). Alligevel lønner det sig at anvende dynamiske RAM'er i større systemer, da lagercellerne er langt mindre arealkrævende i dynamiske RAM end i statiske, hvorved lageromkostningerne pr bit bliver mindre.

#### 2.2.5. Læselagre (ROM)

ROM står for Read Only Memory, dvs. et fastlager, hvis indhold ikke kan ændres ved overskrivning, men kun udlæses.

Den danske betegnelse er: læselager. Indholdet i læselagerets celler fastindlægges under fabrikationen i overensstemmelse med kundens ønske og kan siden ikke ændres, men forsvinder til gengæld heller ikke ved spændingsudfald. Moderne læselagre er opbygget som halvlederlagre.

Læselagre anvendes som permanente hurtiglagre for vitale programmer og for data, som ikke skal ændres. Endvidere anvendes læselageret som opbevaringssted for mikroprogrammer, som definerer ordrepertoiret for nogle maskiner.

#### 2.2.6. Programmérbare læselagre (PROM)

PROM (Programmable ROM) er et læselager, som man ved hjælp af et programmeringsapparat selv kan programmere, dvs. fastindlægge celleindholdet i. Programmeringen sker mest ved elektrisk gennembrænding af udvalgte diodeovergange. Når denne type læselager først er programmeret (indbrændt), kan informationen i den ikke ændres.

PROM'en har samme anvendelsesområde som ROM'en, men anvenderen (kunden) kan selv programmere sit læselager.

#### 2.2.7. Genprogrammérbare læselagre (EPROM, REEPROM)

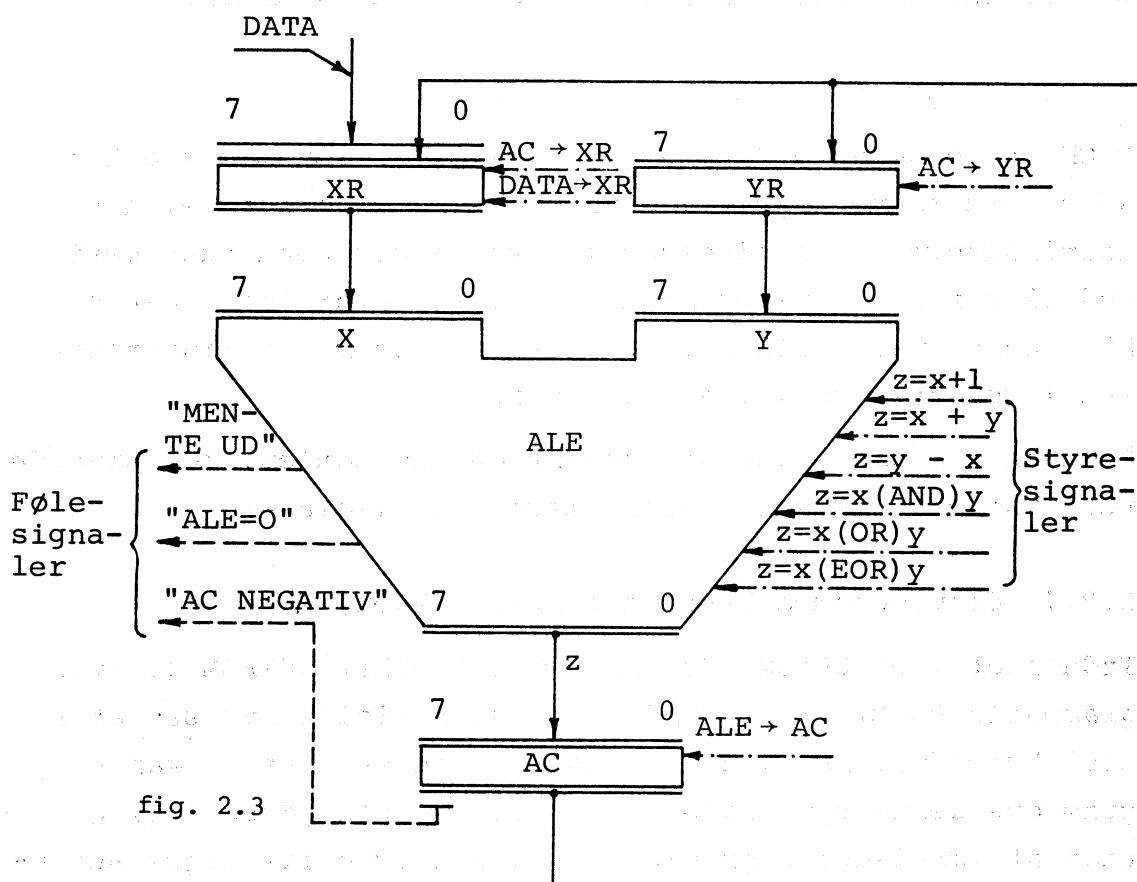
Indholdet i en EPROM (Erasable PROM) eller REEPROM (Re-programmable PROM) kan ved hjælp af ultraviolet lys samlet slettes, hvorefter man med et programmeringsapparat kan ny-programmere denne type læselager. Herved muliggøres fejlkorrektion af læselagerprogrammer, og denne type læselagre anvendes derfor ofte under udvikling af et systems læselager. Når indholdet (programmet) i læselageret er fri for fejl og fungerer efter hensigten, erstattes REEPROM'en med en ROM i det endelige system.

## 2.3. Regneenheden

### 2.3.1. Principiel opbygning af regneenhed

Den principielle opbygning af regneenheden er vist på fig. 2.3 og består af

- 1) en aritmetisk-logisk enhed (ALE)
- 2) et antal arbejdsregistre (XR,YR,AC,MR)



Den aritmetisk-logiske enhed er et kombinatorisk elektronisk netværk, som kan udføre operationer (aritmetiske  $x+y$ ,  $x+1$ ,  $y-x$  samt logiske  $x(\text{AND})y$ ,  $x(\text{OR})y$ ,  $x(\text{EOR})y$  på dataoperanderne  $x$  og  $y$ . Disse dataoperander tilføres ALE som inddata i parallel form, og resultatet af operationen leveres som ud-data i parallel form ( $z$ ).

Til opbevaring af dataoperanderne  $x$  og  $y$ , samt til midlertidig opbevaring af resultatet af en operation anvendes arbejdsregistre, XR og YR, henholdsvis AC.

Resultatregisteret (her AC) betegnes også akkumulatoren eller sumværket.

På fig. 2.3 er datatransportvejene mellem arbejdsregistre og ALE vist. Endvidere er der vist de nødvendige styresignaler til regneenheden (til såvel ALE som arbejdsregistre). For ALE er antydnet 6 (funktions)-styresignaler. Kun eet af disse kan aktiveres ad gangen. Såfremt styrelinien  $x+y$  aktiveres, vil ALE udføre en addition af indholdet i  $x$ -registeret med indholdet af  $y$ -registeret, idet indholdet i disse registre opfattes som binære tal. Sålænge styrelinien  $x+y$  er aktiveret, vil resultatet af denne operation findes tilgængelig på ALE's udgangslinier ( $z$ ). Herfra kan resultatet overføres til - og midlertidigt gemmes i - akkumulatoren AC ved aktivering af styresignalet  $ALE \rightarrow AC$  (læses ALE til AC).

Indholdet af AC kan siden på ny anvendes som operand i en senere ALE-operation, hvis det flyttes (kopieres) til XR eller YR ved hjælp af styresignalet  $AC \rightarrow XR$ , henholdsvis  $AC \rightarrow YR$ ).

På fig. 2.3 er også vist 3 følesignaler, nemlig "MENTE UD", "ALE = 0", "AC NEGATIV". Disse følesignaler kan anvendes af styreenheden (se denne i afsnit 2.5) til at foretage "valg af, hvad der dernæst skal ske" på grundlag af resultatet af den netop foretagne aritmetiske eller logiske operation.

### 2.3.2. Eksempel på gennemførelse af ALE-operation

Lad os antage, at indholdet af YR er kendt - f.eks. fra foregående operation - og lig det binære tal  $01010010_2 = 122_8 = +82_{10}$ ; dvs.  $(YR) = 122_8$  (registerlængden er som vist på fig. 2.3 lig 8 bit).

På "databussen" antages at ligge det binære tal  $00010010_2 = 022_8 = +18_{10}$ .

Vi ønsker at foretage en addition af det binære tal på "databussen" med indholdet i Y-registeret, og resultatet lagres midlertidigt i AC:

Arbejdssekvens: (algoritme)

1. DATA→XR ; kommentar: ved at aktivere styresignalet  
DATA→XR overføres den binære tilstand på data-  
talinierne til XR, hvor det binære tal lagres.  
Udgangslinierne (8 stk) fra XR tjener som ind-  
data til venstre side af ALE, dvs.  $(XR) := 022_8$   
YR indeholder den anden operand, og dets ud-  
gangslinier tjener som inddata til højre side  
af ALE,  $(YR) := 122_8$ .
2. z=x+y ; kommentar: ved at aktivere styresignalet  
z=x+y foretages i ALE en sammenknytning af de  
operander x og y, således at den binære sum  
x+y=z optræder på ALE's udgangslinier.
3. ALE→AC ; kommentar: ved at aktivere styresignalet  
ALE→AC overføres tilstanden på udgangen af  
ALE til akkumulatoren AC, som derefter inde-  
holder:  
$$\begin{array}{rcl} & 00010010_2 & (x) \\ + & 01010010_2 & (y) \\ \hline (AC) := & 01100100_2 & (z) \\ & = 144_8 = +100_{10} & \end{array}$$

### 2.3.3 Skiftefunktion.

På fig. 2.4 er den grundlæggende regneenhed udvidet med en såkaldt skiftefunktion, som meget ofte findes i praktiske datamaters regneenheder.

Udvidelsen består dels i et ekstra arbejdsregister MR, hvori uddata fra ALE kan lagres, dels i at dette nye arbejdsregister og akkumulatoren AC er opbygget og sammenkoblet som et (16 bits) skifteregister.

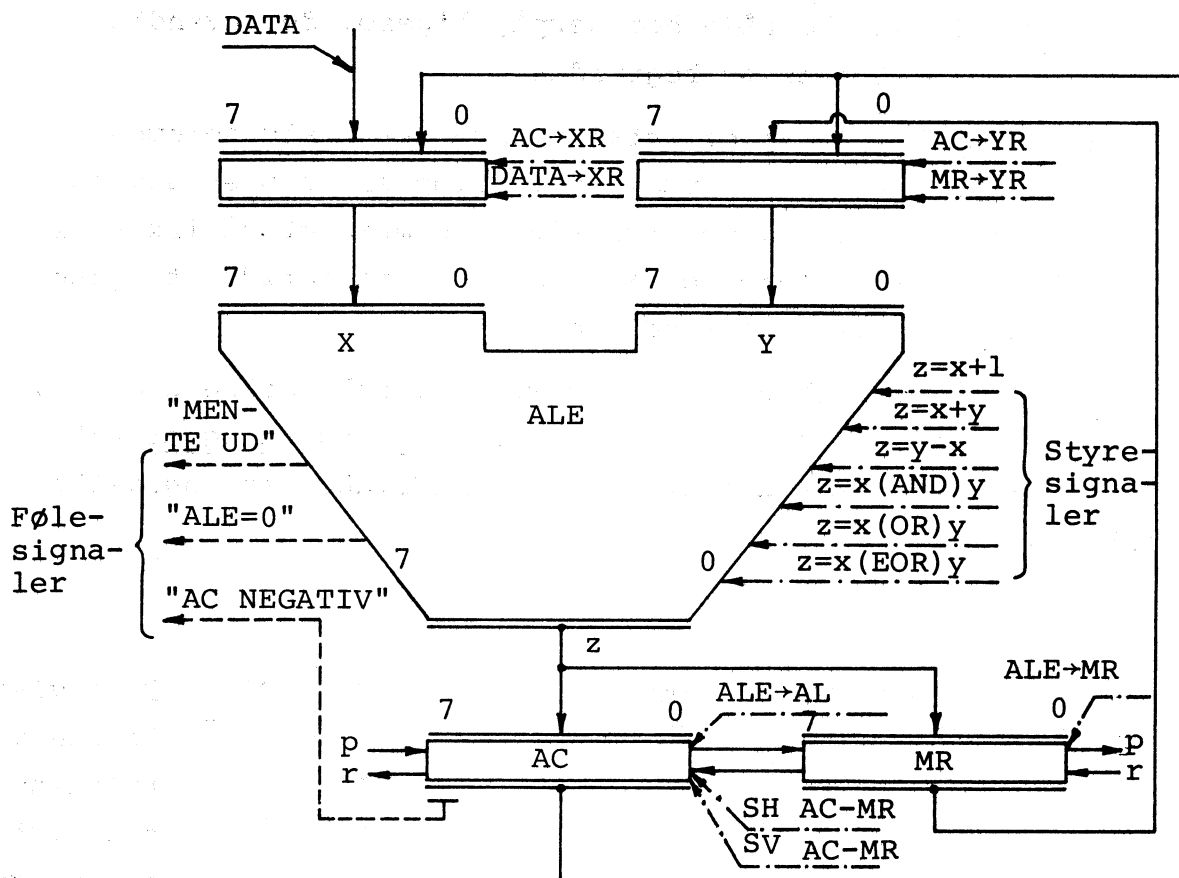


Fig. 2.4

Ved hjælp af styresignalerne SH AC-MR (læses skift AC-MR een position til højre) kan indholdet af det kombinerede AC-MR forskydes een position til højre.

Tilsvarende kan indholdet i AC-MR skiftes een position mod venstre, hvis styresignalet SV AC-MR aktiveres.

Skiftefunktionerne anvendes f.eks. ved udførelse af multiplikations- og divisionsrutiner, men behandles ikke yderligere her.

#### 2.3.4. Registre og registeroverførsler (datatransporter)

I det foregående er der flere gange anvendt betegnelsen register, arbejdsregister og datatransporter uden at gå nærmere ind på, hvad disse betegnelser eksakt står for.



Dette afsnit skulle råde bod herpå, ligesom de anvendte signaturer defineres og begrundes.

Et register består af et antal binære hukommelselementer (lagerelementer), normalt flip-flop'er, som er sammenkoblet således, at samme styresignal samtidigt påvirker alle hukommelselementerne. Man har også hele tiden tilgang til samtlige registerets udgange.

I det følgende tænkes det grundlæggende hukommelselement, hvoraf registrene opbygges, at være en forkanttrigget D-FF (D-flip-flop), hvis signatur og sandhedstabel er angivet i fig. 2.5.

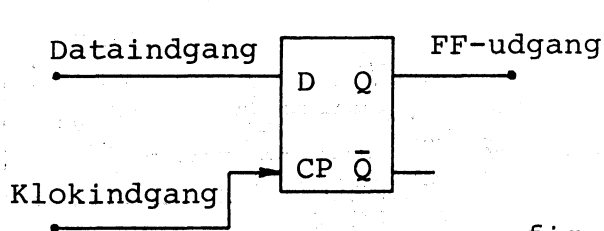


fig. 2.5

D	$Q^+$
0	0
1	1

$Q^+$  angiver FF's tilstand efter trigning af et positivt gående spændingsspring på klokindgangen CP.

Registerets længde er lig antallet af binære hukommelselementer, som registeret består af. Dette er også værdien af den informationsmængde, som kan forvares i registeret. Fig. 2.6. viser et eksempel på et register AR med længde 8, dvs. som kan indeholde "8 bit information".

Nummerering af registerpositionerne er fra højre mod venstre 0 til og med  $n-1$ , hvor  $n$  er antallet bit.

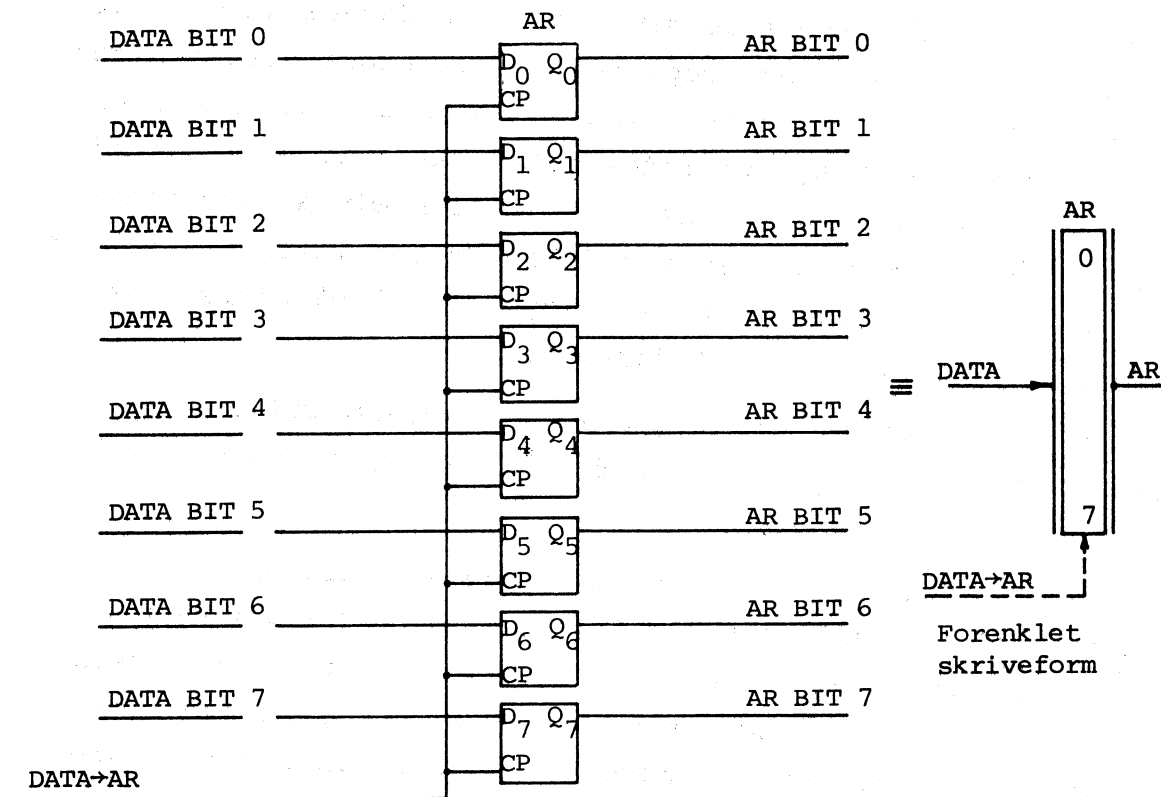


Fig. 2.6

Indholdet i et register er en bitstreng som eksempelvis kan repræsentere:

- a) et binært tal,
- b) et decimalt tal i binærkodet form,
- c) alfanumeriske data, dvs. en tegnstreng
- d) et antal binærtal, f.eks. repræsenterende kontakt-slutninger i en proces,
- e) en operationskode

Det er måden, hvorpå registeret anvendes, som afgør, hvorledes indholdet skal fortolkes.

## Registeroverførsler

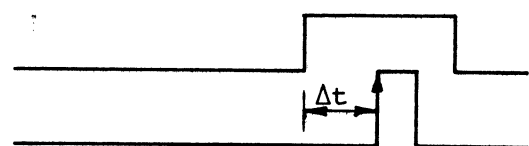
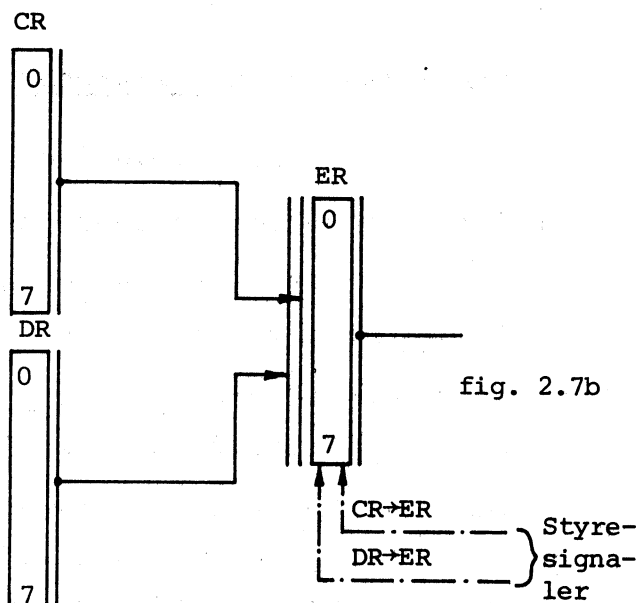
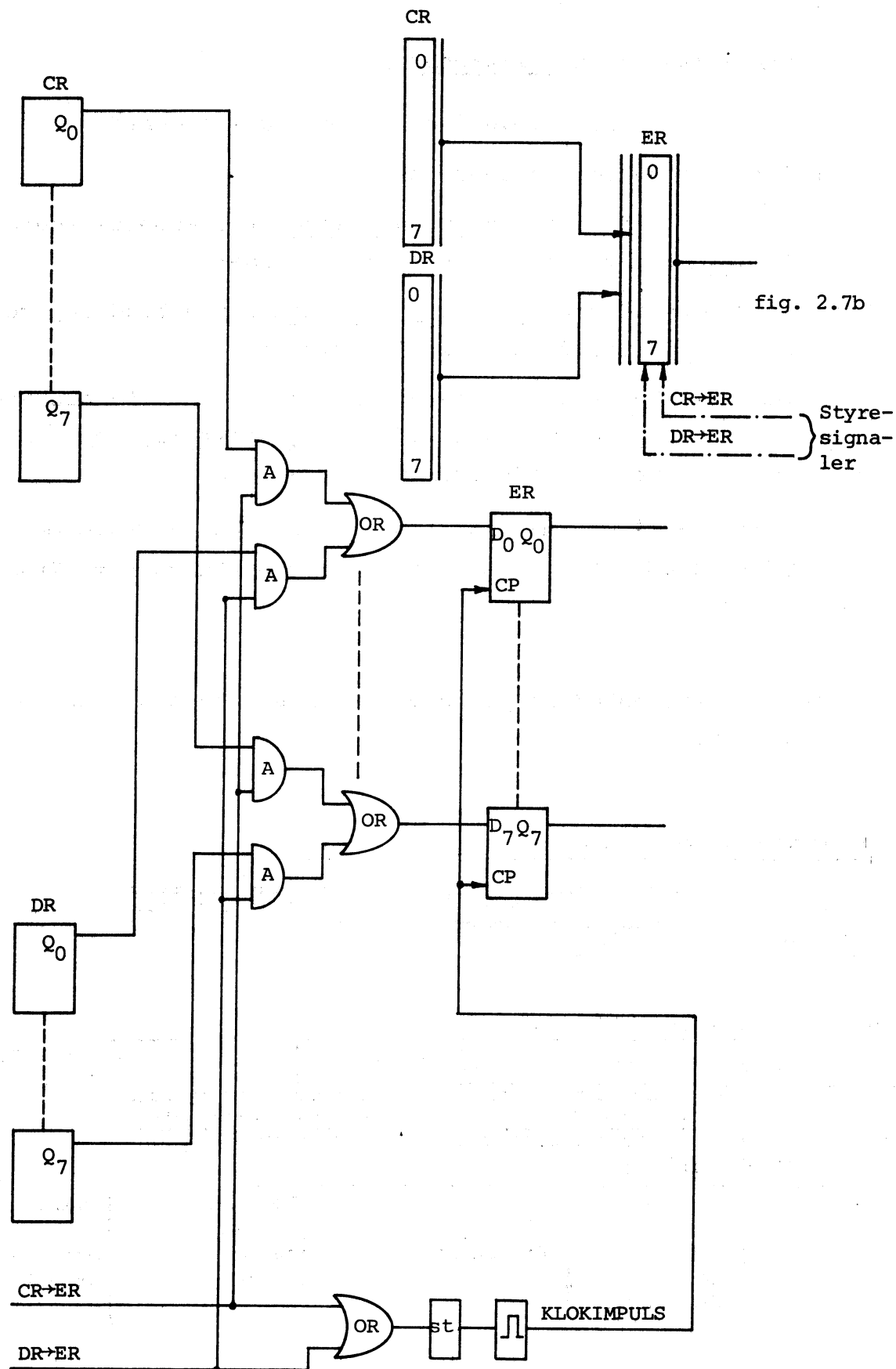
En meget ofte forekommende handling i en datamaskine er at overføre indholdet i eet register til et andet. Såfremt det modtagende register kun skal modtage data fra eet register, illustrerer fig. 2.6, hvorledes dette sker: Data fra et eller andet register overføres til AR ved hjælp af styresignalet DATA→AR, der simpelt hen "klokker den information, som står på dataindgangene, ind i A-registeret" på styresignalet positivt gående impulsforkant.

Overførslen indebærer en kopiering, eftersom indholdet i det register, hvorfra der overføres, bliver stående uforandret.

I fig. 2.6 er endvidere vist en forenklet skriveform, som anvendes i kurset: Et helt register anskueliggøres som et rektangel. Den lodrette streg til venstre for rektangelet med markeret "indkommende pil" angiver, at samtlige hukommelsespositioner overføres parallelt langs datatransportvejen ("bussen") til registeret AR.

Såfremt et register kan modtage data fra mere end eet register, kræves en vis mængde udvælgelogik foran det modtagende register. Et eksempel er vist på fig. 2.7a. Her er anskueliggjort 2 registre CR og DR, hvis indhold kan overføres til registeret ER. Ved at aktivere enten styresignalet CR→ER eller styresignalet DE→ER udvælges, om det er indholdet i CR eller DR, der overføres (kopieres) til ER. Den til "indklokning" (trigning) af modtageregisteret nødvendige positivt gående impulsforkant genereres internt i registerudvælgelogikken (f.eks. ved hjælp af en tidsforsinkelse  $\Delta t$  efterfulgt af en monostabil multivibrator). Resultat: ved at aktivere enten styresignalet CR→ER eller DE→ER overføres indholdet af enten CR eller DR til ER. (Styresignalerne CR→ER og DR→ER er udsignaler fra styreenheden, som beskrives i afsnit 2.5).

På fig. 2.7b er vist den forenkledede skriveform. Den lodrette streg til højre for CR (DR) angiver, at samtlige hukommelsespositioner i CR(DR) overføres parallelt langs datavejen frem til pilen, som tilsammen med de lodrette streger til venstre for ER symboliserer udvalgenetværket til ER.



Tidsrelation mellem styresignaler og klokimpuls.

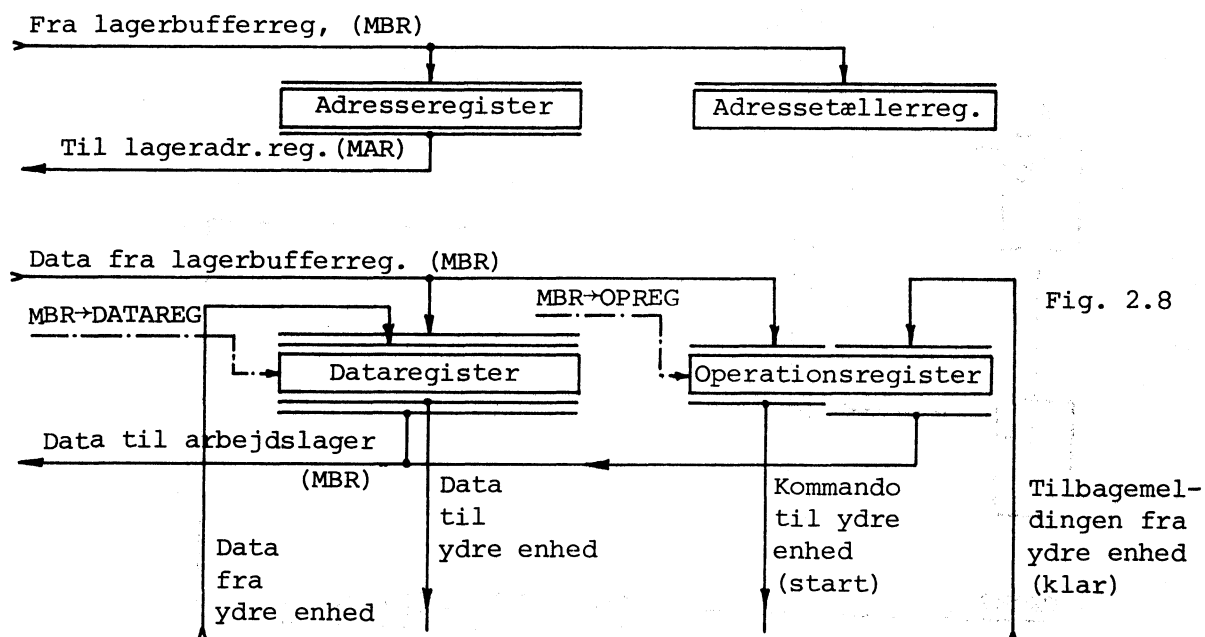
## 2.4. Ind-/udlæse enhed (I/u-enhed)

### 2.4.1. Principiel opbygning af Ind-/udlæse enhed

Ind-/udlæseenheden sørger for

- transport af data fra en ydre enhed til arbejdslageret (indlæsning fra f.eks. hulstrimmellæser)
- transport af data til en ydre enhed fra arbejdslageret (udlæsning til f. eks. linieskriver)
- modtager kommandoer (operationsordrer) fra styreenheden og sørger på grundlag heraf for igangsætning (opstart) af den udvalgte (adresserede) ydre enhed.
- sørger - på grundlag af signaler fra den adresserede ydre enhed - for afslutning af den igangværende operation.

Ind-/udlæse enhedens princip i enkleste form er vist på fig. 2.8.



Den grundlæggende Ind-/udlæse enhed består af følgende registre:

- et dataregister, som midlertidigt opbevarer det dataord, der skal kommunikeres til eller fra den ydre enhed
- et operationsregister, der opbevarer den for den ydre enhed afsette kommando samt den ydre enheds interne tilstand (parat, optaget osv.)

samt evt.

- et adresseregister, der indeholder adressen til den celle i lageret, hvor næste dataord skal lagres (eller er lagret), adresseregisteret kan inkrementeres dvs. forøges med 1
- et adressetælleregister som fra operationens begyndelse kan "sættes" til en værdi svarende til det antal dataord, der skal overføres mellem lageret og den ydre enhed; adressetælleregisteret formindskes med 1 ved hver dataoverførsel mellem lager og dataregister og tælleregisterets 0-tilstand (dvs. når samtlige dataord er overført) kan afføles.

Dertil kommer et antal styre- og følelinier til brug for styreenheden.

Ved ind/ud-ordlængden forstås længden (i antal bit) af det dataord, der overføres mellem dataregisteret og den ydre enhed (på fig. 2.8 er ind/ud-ordlængden 8 bit.)

#### 2.4.2. Ind-/udlæse-enhedens hovedopgaver er

- 1) igangsætning (opstart) af ind/ud-operationen
- 2) gennemførelse af ind/ud-operationen (typisk data-transport mellem lager og ydre enhed)
- 3) afslutning af ind/ud-operationen.

En nærmere beskrivelse af disse hovedopgaver og deres gennemførelse foretages i afsnit 4: Datamaskinens Ind-/udlæsesystem.

### 2.4.3. Dataoverføringsprincipper mellem lager og ydre enheder

Man skelner imellem 3 overføringsprincipper:

- A. Programstyret dataoverførsel
- B. Dataoverførsel med programafbrud (Interrupt)
- C. Dataoverførsel med direkte lagertilgang (direct memory access, cycle steal).

Disse dataoverføringsprincipper behandles detaljeret i afsnit 4: Datamaskinens ind-/udlæsesystem.

Her skal kun kort beskrives så meget om A.: Programstyret dataoverførsel, som er nødvendigt for forståelse af hele centralenhedens funktionsmåde:

Eksempel på programstyret dataoverførsel fra ydre enhed (indlæseenhed):

- 1) LÆS I/U-kommando til operationsregister.  
Kommentar: Et bitmønster svarende til kommandoen "LÆS I/U" overføres til operationsregisteret (MBR → OPREG). Herved foranlediger I/U-enheden, at den ydre enhed opstartes (og læser et tegn) og tilbagesender et bitmønster til I/U-enheden (tilbage-melding), der angiver, om den ydre enhed har udført den beordrede operation eller ej.
- 2) FØL-kommando til operationsregisteret.  
Kommentar: Tilbagemeldingsbitmønsteret overføres fra operationsregisteret til regneenhedens akkumulator (OPREG → AC). I AC kan programmøren nu undersøge bitmønsteret i tilbagemeldingen fra den ydre enhed og afgøre, om denne har udført den beordrede operation (LÆS) og som resultat overført det læste tegn til dataregisteret. Hvis dette er tilfældet fortsættes med 3) eller gentages 2), indtil den ydre enhed har afsluttet LÆS-operationen og placeret resultatet i dataregisteret.

### 3) GEM I/U-ord i lageret

Kommentar: Dataregisterets indhold overføres til lagerbufferregisteret (DATAREG → MBR), hvorefter lagerbufferregisterets indhold "indskrives" i den celle i arbejdslageret, der er udpeget af programøren (via GEM I/U-ordren).

Herefter kan startes forfra med 1).



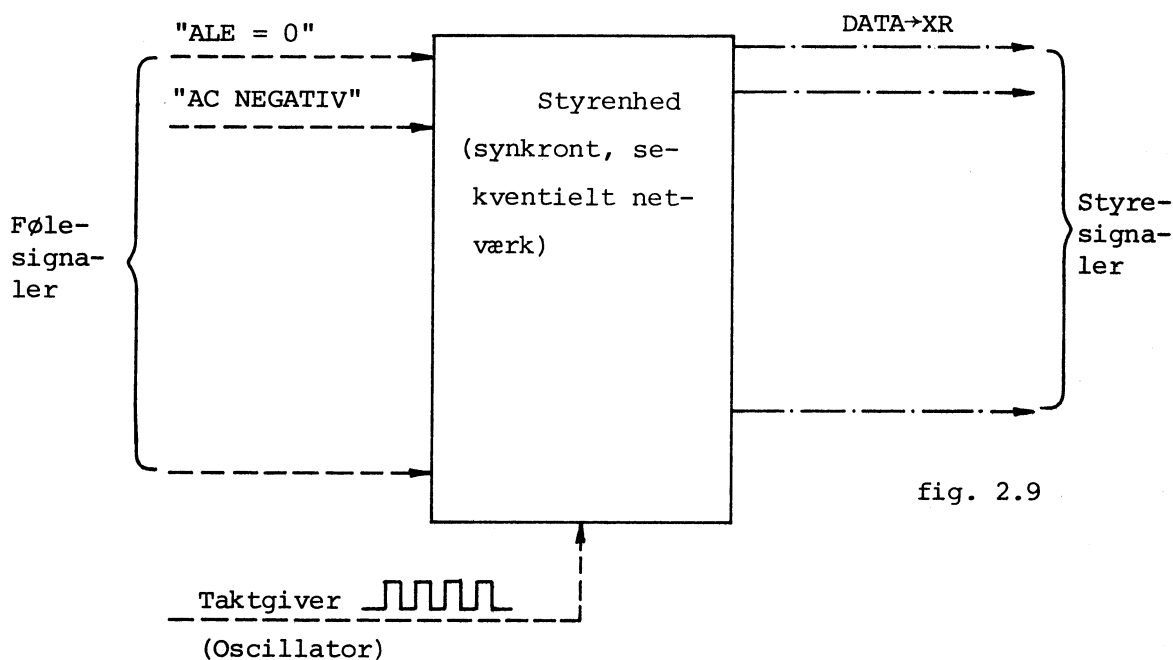
## 2.5. Styreenhed (Styreværk)

### 2.5.1. Styreenhedens opgave

Af den foregående beskrivelse af lagerenhed, regneenhed og I/U-enhed skulle styreenhedens funktion fremgå tydeligt:

- på grundlag af information (via følesignaler) fra lagerenhed, regneenhed og I/U-enhed skal styreenheden styre datastrømmen (fra register til register) i centralenheden. Den aktive styring sker ved hjælp af styresignaler, der åbner og lukker indgangsportene til de forskellige registre.

Vi kan derfor foreløbig beskrive styreenheden som i fig. 2.9:



Definitioner: Et følesignal er et logisk spændingsniveau på en fysisk følelinje.

Et følesignal kan antage 2 logiske værdier (0 eller 1); hvis følesignalet er logisk 1, betyder det, at en ganske bestemt betingelse i datamaskinen er opfyldt (f.eks. "AC NEGATIV").

Følesignaler tjener som inddata for styreenheden.

Et styresignal er et logisk spændingsniveau på en fysisk styrelinje og genereres af styreenheden.

Et styresignal kan antage 2 logiske værdier (0 el. 1).

Hvis styresignalet er logisk 1, betyder det udførelse af en ganske bestemt handling i datamaskinen (f.eks. "AC→YR", overførsel af indholdet i akkumulatoren til Y-registret (fig. 2.4.))

Styresignaler er styreenhedens uddata.

Ved datastrømmen forstås transporten af data (i parallel form) over datamaskinens datatransportveje - fysisk opbygget som et antal parallelle kobberledninger - mellem datamaskinens registre.

På fig. 2.9 er foruden følesignaler (inddata) og styresignaler (uddata) vist et yderligere indkommende signal, nemlig det såkaldte taktgiver-signal, som er en kontinuert følge af spændingsimpulsen fra en impulsgiver (oscillator). Taktgiver-signalet anvendes til synkronisering af styreenheden.

Styreenheden på fig. 2.9 kan således betragtes som et synkront sekventielt netværk, hvis inddata er følesignalerne, og hvis uddata er styresignalerne.

#### 2.5.2. Faste registre i styreenheden

Foruden den egentlige styreenhed som beskrevet på fig. 2.9 henregnes traditionelt visse bestemte registre med til den blok IV i fig. 2.1, som betegnes "styreenheden".

Disse registre er følgende:

- ordregisteret (instruction register), OR
- ordretælleren (program counter, location counter) OT
- kontrolpanelets "datakontakter" KR
- kontrolpanelets trykknapper, spec. START og STOP.

Den fuldstændige blok IV ("styreenheden") er vist på fig. 2.10.



### 2.5.3. Tidsgenerering

Det på fig. 2.9 og 2.10 viste inddatasignal "taktgiversignal" anvendes til generering "klokimpulser", som sørger for at drive centralenheden synkront, dvs. genererer styresignalerne i en kontinuert, taktmæssig strøm.

Til belysning af tidsgenerering er på fig. 2.11 vist et tidsdiagram for en lagercyklus for et destruktivt RAM-lager (f. eks. kernelager)

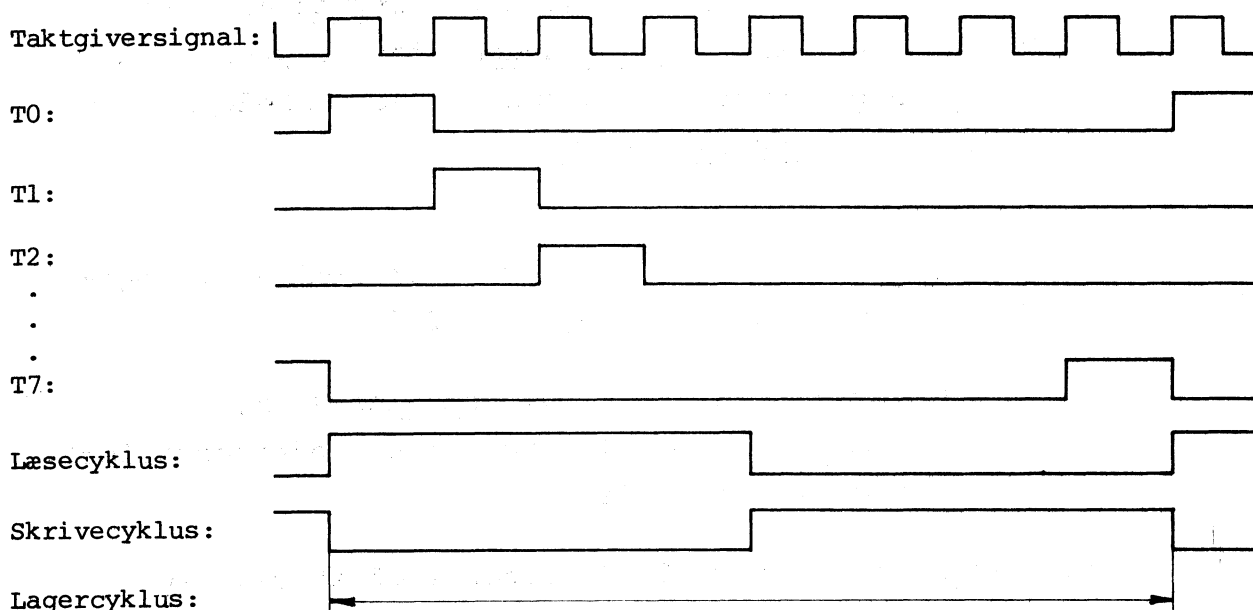


Fig. 2.11

Klokimpulserne (T0 til T7) styrer hele arbejdssekvensen i centralenheden. Antallet af klokimpulser, som er nødvendige i en lagercyklus, bestemmes af, hvor mange - i tiden adskilte - aktiviteter, som skal udføres under en cyklus (i eksemplet anvendes 8 klokimpulser, et normalt antal for en datamaskine).

Hvorledes styresignalerne genereres i styreenheden, betinget af følelinier og tidsmæssigt taktgivet af klokimpulserne, behandles senere i afsnit 3.8.

## 2.6. Alternativ beskrivelse af datamaskinen

### 2.6.1. Alternativ beskrivelse

Sammenkobles de i afsnit 2.2 til 2.5 beskrevne enheder (blok I til IV på fig. 2.1) til een enhed, datamaskinen, således som det er gjort på fig. 2.12, kan man beskrive datamaskinen alternativt som:

- en registerstruktur forbundet af et system af datatransportveje;
- datatransporten mellem registrene styres af et sekventielt netværk, styreenheden, der på grundlag af følesignaler åbner indgangene til de ønskede registre ved hjælp af styresignaler.

2.6.2. Registerstrukturen udgøres af alle de beskrevne registre afbildet i fig. 2.2, 2.3, 2.4, 2.8, 2.10.

2.6.3. Datatransportvejene består af databussystem, dvs. et net af parallelle kobberledere (eller ledningskabler), der forbinder registrene indbyrdes.

2.6.4. Styreenheden (se fig. 2.12) udgøres af et rent synkront sekventielt netværk, som kan opbygges konventionelt af logiske kredse, eller den kan opbygges omkring et ROM-lager, der mikroprogrammeres (se afsnit 3).

Styreenheden styrer datatransporten mellem registrene ved hjælp af styresignaler, der er styreenhedens uddata. Styresignalerne er styreenhedens handler, der f.eks. overfører data fra eet register til et andet ved at åbne respektive dataport.

Disse handler udføres på baggrund af betingelser, manifesteret som følesignaler, rundt omkring i registerstrukturen (f.eks. operationsdelen af ordregistret OR; mente-FF).

Alle styreenhedens funktioner taktgives af en oscillator, der leverer en kontinuert impulssekvens.

## 2.7. Specifikation af registerstrukturen

Af hensyn til afsnit 3, datamaskinens virkemåde, specificeres registerstrukturen i fig. 2.12 således, at vi i det følgende har en "konkret datamaskine" at arbejde med.

### 2.7.1. Arbejdslager

Lagerstørrelse: 4096 ord (4k-ord)  
Ordlængde: 16 bit  
Lagertype: RAM-kernelager  
Cyklustid: 1  $\mu$ s  
Lageradresse-  
register (MAR): 12 bit  
Lagerbuffer-  
register (MBR): 16 bit

### 2.7.2. Regneenhed

Akkumulatorer: (AC) 16 bit  
Operandregister: (XR,YR) 16 bit  
Regneoperationer  
i ALE ADD, SUB, XFER,  
AND, OR, EOR, INV  
Følebetingelser: Mente - FF = 1  
(AC) = 0  
(AC) NEG

### 2.7.3. Ind-/udlæseenhed

Dataregister: I/U- DATAREG 16 bit  
Operationsregister: I/U- OPREG 16 bit  
Ind/ud-ordlængde: 16 bit

### 2.7.4. Datatransportsystem: 16 bit parallel bus

### 2.7.5. Styreenhed

Ordregister: (OR) 16 bit  
Ordretæller: (OT) 12 bit  
Oscillatorfrekvens: 8 MHz

## Styresignaler:

	<u>Styresignal nr.:</u>	<u>Funktionsbeskrivelse:</u>
Lagerstyring	1	L(MAR) → MBR (LÆS)
	2	MBR → L(MAR) (SKRIV)
Registerstyring	3	MBR → XR
	4	MBR → YR
	5	MBR → OR
	6	MBR → DATAREG
	7	MBR → OPREG
	8	AC → XR
	9	AC → YR
	10	AC → MBR
	11	KR → MBR
	12	KR(0..11) → OT
	13	OR(0..11) → OT
	14	OT+1 → OT
	15	OT → MAR
	16	OR(0..11) → MAR
	17	DATAREG → AC
	18	z(ALE) → AC
ALE-styring	19	z = x+y
	20	z = y-x
	21	z = x+1
	22	z = x(AND)y
	23	z = x(OR)y
	24	z = x(EOR)y
	25	z = x
	26	z = y
Intern styring i styreenhed	27	OPKODE → STYR
	28	"(AC)=0" → STYR
	29	"MENTE-FF" → STYR
	30	"AC NEGATIV" → STYR

<u>Følesignaler:</u>	<u>Følesignal nr.:</u>	<u>Følebetingelse:</u>
	31	"(AC) = 0"
	32	"MENTE-FF = 1"
	33	"AC NEGATIV"
	34..... 40	OR( 9 .....15) (OR-OPKODE)

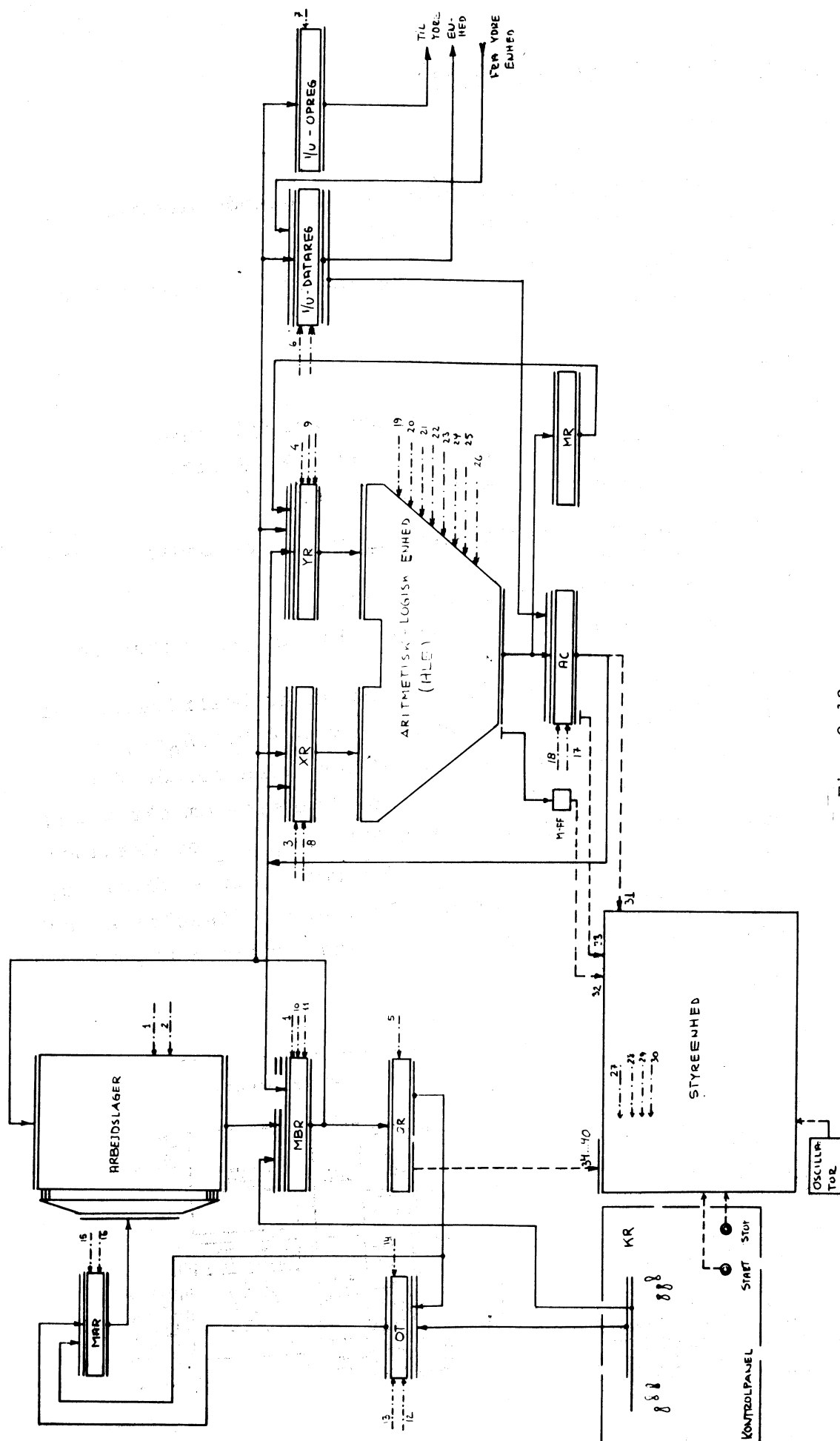


Fig. 2.12



## 2.8. Øvelsesspørgsmål og opgaver

### Generel blokmodel

G.1. Et datamaskinsystem kan opdeles i 2 grundelementer. *Hardware*  
Hvilke er disse og beskriv dem? *Software*

G.2. Nævn centralenhedens systemkomponenter og beskriv kort deres funktion.

### Lagerenheder

L.1. Flere minidatamater har et mindste arbejdslager på 8 k.ord (8.192 ord) og en ordlængde på 16 bit.  
Angiv den nødvendige længde (antal FF) på

L.1.1. lageradresseregisteret (MAR) for en sådan lagerstørrelse *13681 8K*

L.1.2. lagerbufferregisteret (MBR). *16 bit ordlængde*

L.1.3. Angiv lagerkapaciteten i 8 bit-bytes (oktetter). *16K byte*

L.2. Fig. A er hentet fra en databog og viser beskrivelsen af en statisk RAM af størrelsen 256 ord  $\times$  4 bit.  $A_0-A_7$  er adresseindgange til udvælgelse af den adresserede 4-bit celle. R/W er en styrelinie til bestemmelse om skrivning til RAM henholdsvis læsning fra RAM.  $DI_1-DI_4$  er dataindgange;  $DO_1-DO_4$  er dataudgange.  $\overline{CE}_1, \overline{CE}_2$  (chip enable) = 0,1 "udvælger" (aktiverer) RAM'en. OD (output disable) er en styrelinie, ved hjælp af hvilken dataudgangene  $DO_1-DO_4$  kan deaktiveres.

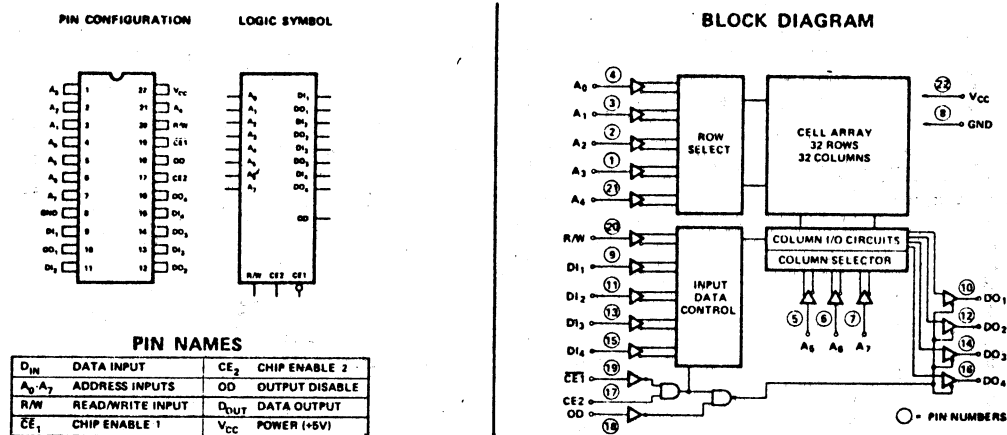


Fig. A

## FORSLAG TIL OPGAVEBESVARELSE AF ØVELSESOPGAVERNE:

### G1, G2, L1, L2, R1, S1 og S2 i afsnit 2.8.

#### G1.

~~Software~~  
Programmel: Programmer og procedurer, der er knyttet til en datamaskine for at lette brugen af denne.

~~Hardware~~  
Materiel: Den fysiske af elektroniske og mekaniske elementer opbyggede datamaskine.

#### G2.

Lager: Til opbevaring af data for senere brug.

Regneenhed: Den del af datamaskinen, hvor regneoperationer, logiske operationer og skift udføres.

Ind-/Udlæsenhed: Den del af datamaskinen der varetager kommunikationen med omverdenen.

Styreenhed: Den del af datamaskinen, der ved modtagelsen af en ordre oversætter ordren, får de relevante kredsløb til at udføre ordren og styrer rækkefølgen af og tidspunkterne for deloperationernes udførelse.

#### L1.

Ved 8K-ord fås:

L.1.1 Lageradresseregisteret (MAR) = 13 bit ( $2^{13}=8K$ )

L.1.2 Lagerbufferregisteret (MBR)= 16 bit

L.1.3 Lagerkapaciteten = 16384 byte (2x8K).

## L 2

L.2.1 4 stk RAM kan realisere 256 ord á 16 bit.

16 stk RAM totalt for 1024 ord á 16 bit.

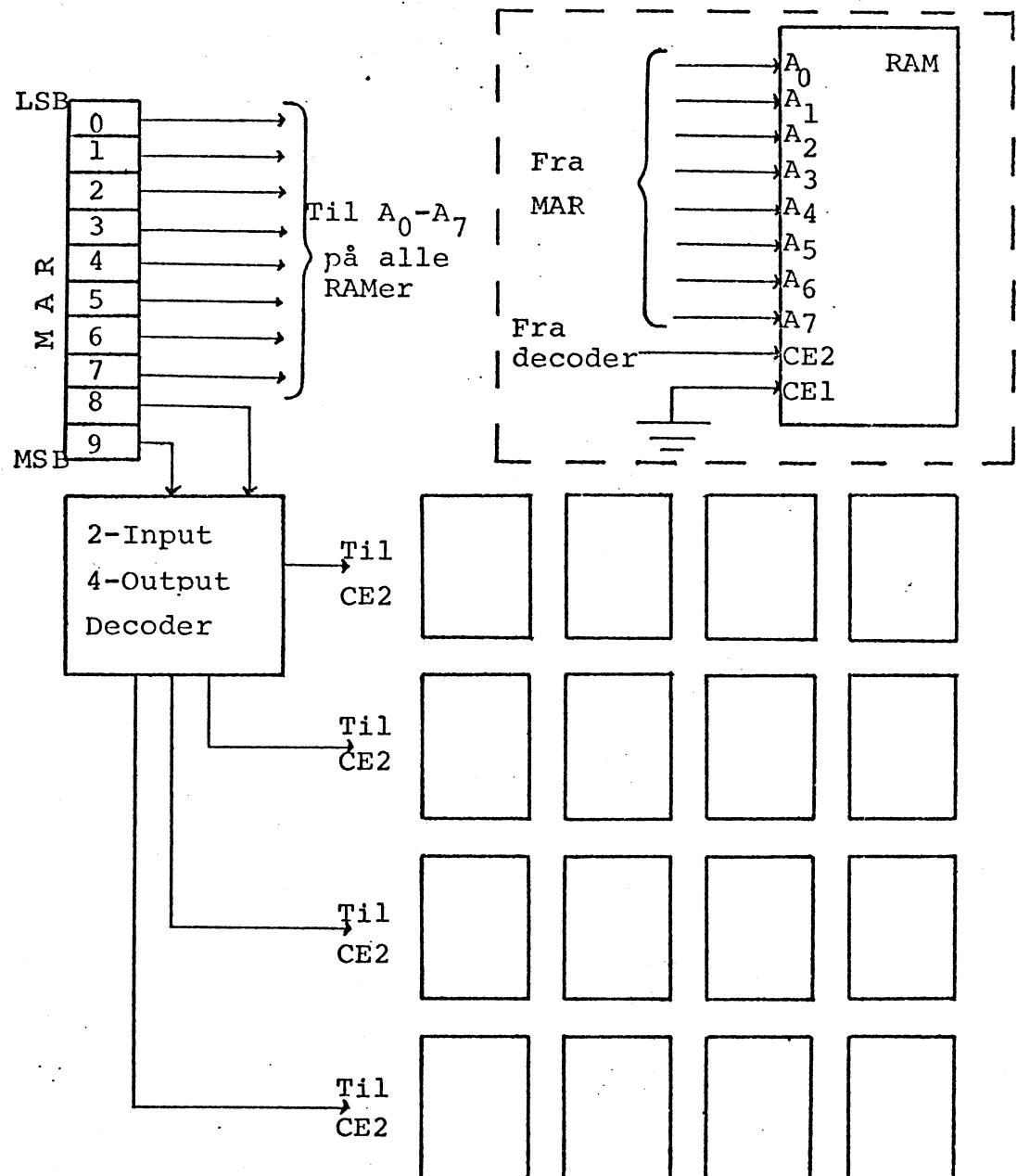
### L.2.2

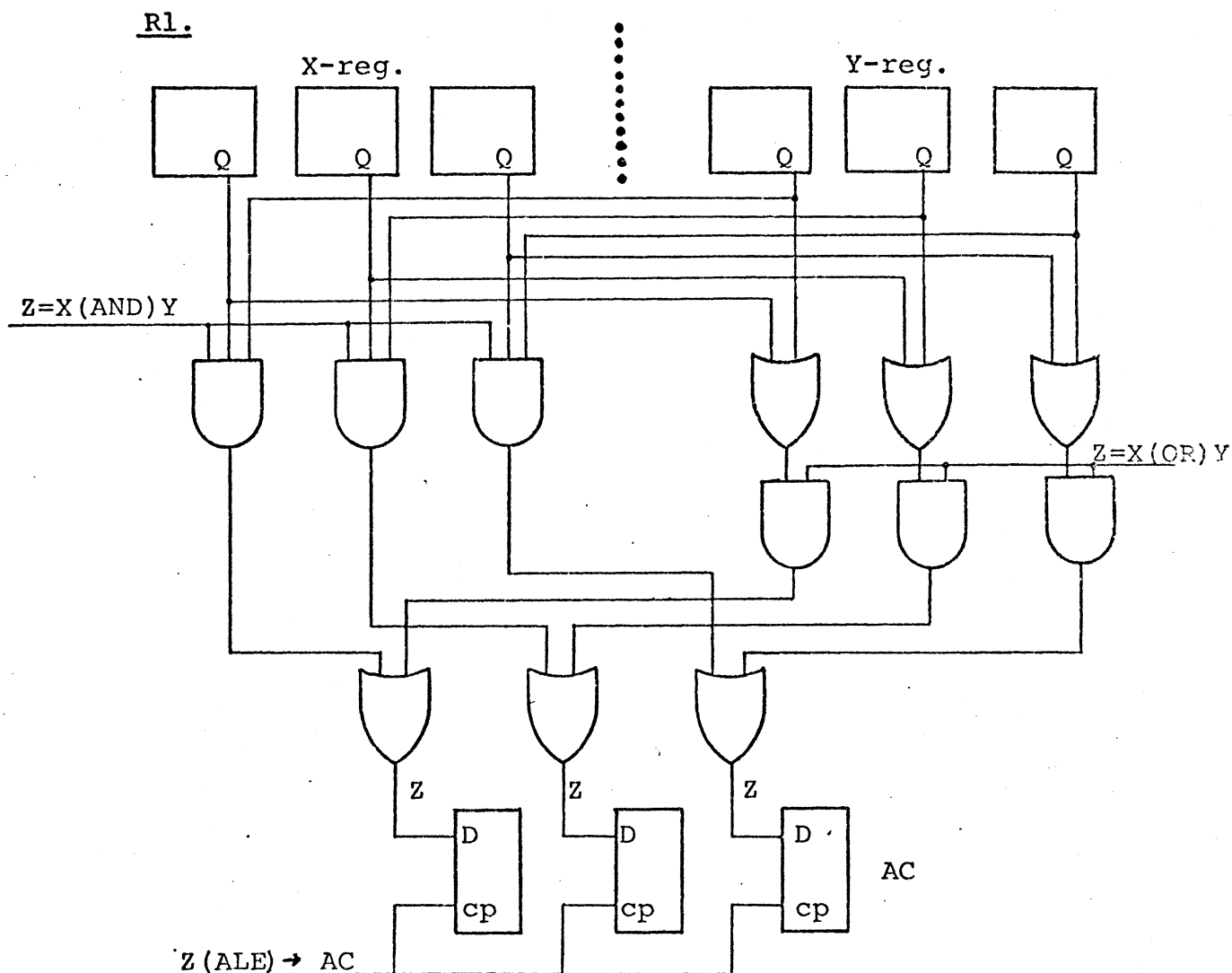
Lageradresseregisteret (MAR) = 10 bit ( $2^{10}=1K$ ).

Lageradresseregisterets 8 LSB bit skal forbindes til adresseindgangene  $A_0 - A_7$  på alle 16 RAM'er.

Lageradresseregisterets 2 MSB (bit 8 og 9) skal styre RAM'ens CE-indgange; dette kræver dog en 2-input til 4-output dekoder.

### L.2.3





S1.

Længden (antal FF) af ordretælleren (OT) bestemmes af det antal ord, der direkte skal kunne adresseres i programlageret.

S2.

Et styresignal er et logisk niveau på en styreledning.

Et følesignal er et logisk niveau på en føleledning.

- L.2.1. Hvor mange moduler skal bruges til konstruktion af et 1 k RAM lager med 16 bits ordlængde? 16
- L.2.2. Hvilken registerlængde (antal FF) er nødvendig i adresseregisteret MAR til adressering af et 1 k RAM-lager som anført i L.2.1.? 10
- L.2.3. Vis hvorledes CE-indgangene skal styres fra MAR via en 2 input - 4 output-dekoder for at adressere et 1 k RAM lager, som anført i L.2.1.

## Regneenheden

R.1. Konstruer en meget enkel 3 bits regneenhed bestående af 2 3-bits operandregistre XR og YR samt en 3 bits akkumulator AC.

Registrene opbygges af forkanttriggede D-FF.

Den aritmetisk-logiske enhed, ALE, skal på 3 bits operanderne x og y kunne udføre operationerne

$$z = x(\text{AND})y$$

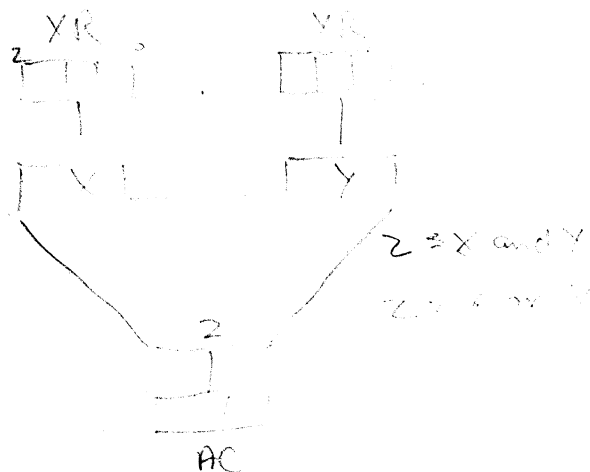
$$z = x(\text{OR})y$$

styret af styresignaler med tilsvarende betegnelser (se fig. 2.3).

## Styreenheden

S.1. Hvad bestemmer længden (antal FF) af registeret "ordretælleren" (OT)? *lagers størrelse*

S.2. Hvad forstås ved et styresignal? *output fra styreenhed*  
Hvad forstås ved et følesignal? *input til styreenhed*



### 3. Datamaskinens virkemåde

#### 3.1. Princip for programstyring

##### 3.1.1. Datamaskinens "arbejdsbeskrivelse"

Datamaskinens opbygning og virkemåde bygger på princippet: programstyring.

For at kunne løse en opgave behøver datamaskinen en detal-  
leret arbejdsbeskrivelse. Inden for matematikken (hvorunder  
datamatikken ofte henregnes) betegnes en sådan detailleret  
arbejdsbeskrivelse for en algoritme.

Definition: Ved en algoritme forstås en endelig og entydig  
følge af arbejdsanvisninger (arbejdsskridt) til løsning af  
et forelagt problem.

Et program er et eksempel på en algoritme, idet  
definition: Et maskinprogram er en endelig og entydig se-  
kvens af maskinordrer til løsning af en forelagt opgave.

Elementarprocessen (arbejdsskridtet) i et maskinprogram er  
altså maskinordren.

En datamaskine (som beskrevet i afsnit 2) er i stand til at  
udføre et fåtal forskellige maskinordrer.

Programmøren kan af dette fåtal forskellige maskinordrer sam-  
mensætte maskinprogrammer, der, når programmet udføres i ma-  
skinen (ved at maskinordrerne udføres sekventielt den ene ef-  
ter den anden) løser forskellige opgaver.

Hver enkelt maskinordre indeholder således

- oplysning om hvilken elementarproces, der skal udføres

Af ovenstående fremgår det, at et maskinprogram består af en  
sekvens af maskinordrer, der ligger placeret i celler i arbejds-  
lageret.

Programmet kan placeres (indlæses) i lageret sammen med de data,  
programmet arbejder på.

Definition: Ved data forstås en formaliseret repræsentation af kendsgerninger (f.eks. tal) på en sådan form, at den kan kommunikeres eller omformes ved en proces i datamaskinen.

### 3.1.2. Eksempel på enkel programstyring

Som eksempel på en enkel programstyring vil vi se på følgende opgave: Adder de 2 binære heltal, der er placeret i henholdsvis celle  $0110_8$  og  $0111_8$ , og placer resultatet af opgaven i celle  $0112_8$ .

Vor datamaskine er defineret i afsnit 2.7.

En betingelse for at få udført noget overhovedet er ifølge det foregående afsnit 3.1.1., at der i arbejdslageret er placeret et maskinprogram.

På fig. 3.1 er vist programmet og dets placering i arbejdslageret, idet første maskinordre er indlagt i celle  $0100_8$  og de følgende maskinordrer i de efterfølgende celler. Data, dvs. de 2 tal som maskinprogrammet arbejder med, er placeret i celle  $0110_8$  og  $0111_8$ , som krævet i opgaven. Endvidere har vi i ordretælleren (OT) placeret begyndelsesadressen ( $0100_8$ ) på programmet, nemlig nummeret på den celle, der indeholder første maskinordre.

ADRESSE (cellenr)	CELLEINDHOLD (oktale værdier)	HUSKE-FORM	KOMMENTAR
$0100_8$	0 6 0 1 1 0	HENT 110, XR	Hent indhold af $110_8$ til XR
$0101_8$	0 5 0 1 1 1	HENT 111, YR	Hent indhold af $111_8$ til YR
$0102_8$	0 1 0 0 0 0	ADDER	Adder (XR) og (YR) til AC
$0103_8$	0 4 0 1 1 2	GEM AC, 112	Gem (AC) i $112_8$
$0104_8$	0 0 0 0 0 0	STOP	Standt ordreudførelsen
$0105_8$			
$0106_8$			
$0107_8$			
$0110_8$	0 0 0 0 0 5	DATA 5	Tallet $5_{10}$
$0111_8$	0 0 0 1 0 0	DATA 100	Tallet $64_{10}$
$0112_8$			

Fig. 3.1



Når vi trykker på START-knappen på kontrolpanelet sker følgende: (se også fig. 2.12)

- 1) Ordretællerens (OT) indhold (0100) overføres til lageradresseregisteret (MAR)  
Ved hjælp af lagerelektronikken udlæses dernæst indholdet af celle 0100<sub>8</sub> til lagerbufferregisteret (MBR) (generelt udlæses til MBR indholdet af den celle, hvis adresse står i MAR).
- 2) Indholdet af lagerbufferregisteret MBR overføres til ordregisteret OR.

De første 4 bit i det ord, som ordregisteret (OR) nu indeholder (som jo netop er første maskinordre) overføres til styreenheden og opfattes af denne som en besked om, hvad der skal gøres (denne del af maskinordren, den såkaldte operationsdel er i eksemplet 06<sub>8</sub>, og fortæller altså styreenheden, at ordren er: Hent fra lager til XR).

De sidste 12 bit i ordregisteret kaldes adressedelen, og disse 12 bit udgør adressen til den celle i lageret, hvorfra operanden skal hentes. Adressedelen overføres til lageradresseregisteret MAR.

- 3) Så snart styreenheden har registreret "HVAD" der skal udføres, udføres denne ordre, dvs. i eksemplet udlæses indholdet af celle 0110 (=5) til MBR, hvorefter MBR overføres til XR.

Herved er første maskinordre udført.

- 4) Ordretællerens (OT) indhold øges med 1, og der startes ved 1) igen, og næste maskinordre udlæses osv.

### 3.2. Maskinordrer

#### 3.2.1. Maskinordrens opbygning

Maskinordren er at betragte som brugerens (programmørens) elementaroperation, dvs. "det mindst mulige arbejdsskridt" i maskinens detaljerede arbejdsanvisning.

Den generelle opbygning er vist på fig. 3.2.

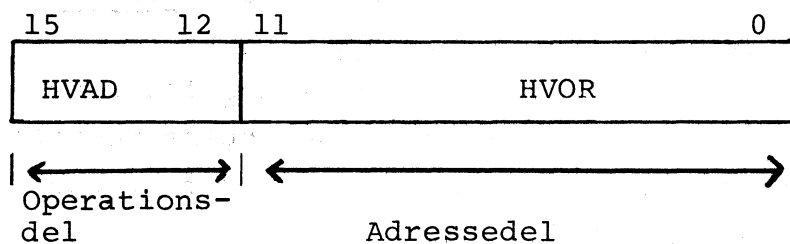


Fig. 3.2.

Maskinordren består af et eller flere lagerord. (I vort system udgøres maskinordren af eet ord på 16 bit)

Maskinordren er opdelt i 2 felter: (se fig. 3.2)

- operationsdel, der bestemmer HVAD datamaskinen skal udføre under elementarprocessen. I vor maskine består operationsdelen af de 4 mest betydende bit (bit 12-15)
- adressedel, der bestemmer HVOR operanden, der skal opereres på, findes i arbejdslageret. I vor maskine består adressedelen af de 12 mindst betydende (bit 0-11).

3.2.2. Operationsdelen (OPKODEN) udgøres af et antal bit (her 4), hvis kombination entydigt bestemmer elementaroperationen.

I eksemplet på fig. 3.3. er anvendt flg. OPKODE:

15	12	11	0	
0	1	1	0	06 HENT til XR
0	1	0	1	05 HENT til YR
0	0	0	1	01 ADDER XR og YR
0	1	0	0	04 GEM fra AC
0	0	0	0	00 STOP
OPKODE				ADRESSEDEL

Fig. 3.3

3.2.3. Adressedelen er en bitkombination, der angiver operandens placering i lageret. I vor maskine opfattes de 12 bit, der udgør adressedelen som et binært tal, der direkte bestemmer adressen (cellenummeret) på den celle, der indeholder operanden. Bit 0 er mindst betydende bit.

I vor maskine kan vi altså med de til rådighed stående 12 adressebit adressere cellerne 0 til  $4095_{10}$ .

#### 3.2.4. Adresseringsformer

Den i vor maskine anvendte adresseringsform, hvor adressedelen direkte angiver adressen (cellenummeret) på den celle, hvori operanden findes, kaldes direkte adressering. Denne adresseringsform vil blive anvendt i den følgende beskrivelse af datamaskinens virkemåde.

Angående andre adresseringsformer som indexeret adressering og indirekte adressering henvises til afsnit 7.1.

### 3.3. Ordretyper

Maskinordrerne kan opdeles i fire typer:

- 1) Lageroverførselsordrer
- 2) Regneordrer
- 3) Styreordrer
- 4) Ind/ud-ordrer

#### 3.3.1 Lageroverførselsordrer

Ved lageroverførselsordrer forstås ordrer, som flytter data fra en lagercelle til et register (f.eks. HENT) eller flytter data fra et register til en lagercelle (f.eks. GEM).

Da lageroverførselsordrer altid implicerer en lagercelle, må disse ordrer altid indeholde adressen til en lagercelle.

I vor maskine definerer vi følgende lageroverførselsordrer:

15	12	11	0	
0	1	1	0	06 HENT TIL XR FRA ADR
0	1	0	1	05 HENT TIL YR FRA ADR
0	1	0	0	04 GEM AC I ADR
0	1	1	1	07 DEK (ADR)

OP KODE

ADResse

Fig. 3.4

### 3.3.2. Regneordrer

Regneordrer er ordrer, som bearbejder data i regneenheden. Karakteristisk for regneordrer er, at de udfører grundlæggende aritmetiske-(addition, subtraktion) eller logiske (AND,OR) regneoperationer på operander.

I overensstemmelse hermed opdeles regneordrer i aritmetiske og logiske ordrer samt flytteordrer.

I vor maskine forudsættes altid, at operanderne for enhver regneordre i forvejen er placeret i operandregistrene XR og YR.

Følgende regneordrer defineres:

	15	12	11	9	8	0						
Aritmetiske ordrer	0	0	0	1	0	0	0	010	ADD	XR	og	YR
	0	0	0	1	0	0	1	011	SUB	XR	FRA	YR
	0	0	0	1	0	1	0	012	ADD	1	TIL	XR
Logiske ordrer	0	0	0	1	1	0	0	014	AND	XR	og	YR
	0	0	0	1	1	0	1	015	OR	XR	og	YR
	0	0	0	1	1	1	0	016	EOR	XR	og	YR
	0	0	0	1	1	1	1	017	INV	XR		
Flytte- ordrer	0	0	1	0	0	0	0	020	FLYT	AC	TIL	XR
	0	0	1	0	0	0	1	021	FLYT	AC	TIL	YR
	0	0	1	0	0	1	0	022	FLYT	XR	TIL	AC
	0	0	1	0	0	1	1	023	FLYT	YR	TIL	AC

Fig. 3.5

Da disse ordrer ikke kræver lageradressering, kan adressedelen af maskinordren anvendes til underspecifikation af regneordren (bit 12-15 =01 angiver, at det er en regneor-

dre, bit 9-11 hvilken speciel regneordre, det drejer sig om).

### 3.3.3. Styreordrer

Styreordrerne tjener til styring af programforløbet. Normalt findes ordrene i et program i på-hinanden følgende celler i arbejdslageret og eksekveres successivt af styreenheden. Hvis denne udførelsesrækkefølge skal brydes, sker det ved hjælp af en hopordre, som udgør en af programmets ordrer, og som meddeler styreenheden, hvor denne skal finde næstfølgende maskinordre.

Specielle styreordrer: STOP-ordren, som standser den trin-vise ordreudførelse (ved at blokere klokgeneratorens takt-impulser).

Styreordrer kan inddeles i:

- Ubetingede hopordrer
- Betingede hopordrer
- Specielle styreordrer.

Ubetingede hopordrer, er ordrer, som i deres adressedel angiver, hvor den næste maskinordre står i lageret; ved udførelse af ordren ændres udførselssekvensen ubetinget.

Betingede hopordrer er ordrer, som i deres adressedel angiver, hvor næste maskinordre findes, hvis en bestemt betingelse er opfyldt; hvis betingelsen ikke er opfyldt, fortsættes med udførelsen af den ordre, som står i den normale næstfølgende celle (d.v.s. i den normale, stigende adresserækkefølge).

Betingede hopordrer gør det muligt at lave programløkker.

Følgende styreordrer defineres:

15	12	11	0		
1	1	0	0		14 HOP TIL ADR
1	1	0	1		15 HOP TIL ADR HVIS (AC)=0
1	1	1	0		16 HOP TIL ADR HVIS AC NEG
1	1	1	1		17 HOP TIL ADR HVIS M-FF=1
0	0	0	0		00 STOP
OP KODE				ADR	

FIG. 3.6.

### 3.3.4. Ind/ud-ordrer

Ind/ud-ordrer foranlediger, at data overføres mellem centralenhedens registre og de ydre enheder, samt starter og stopper ydre enheder.

Vi definerer følgende Ind/ud-ordrer:

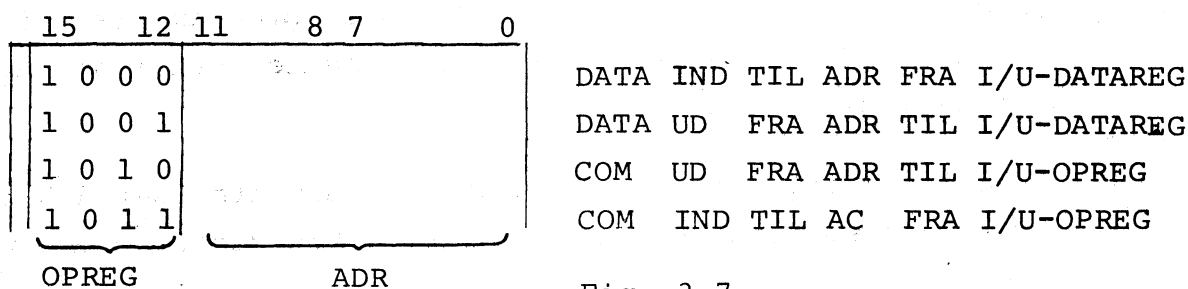


Fig. 3.7

### 3.4. Ordresekvenser (Programmer)

Øvelse: På grundlag af de i afsnit 3.3 definerede maskinordrer skal der konstrueres et program, der ombytter placeringen af 2 heltal A og B placeret i henholdsvis celle  $144_{10}$  og  $145_{10}$ .

Programmet placeres i lageret med første ordre i celle  $80_{10}$ .

### 3.5. Eksekvering af maskinordre

#### 3.5.1. Hvorledes udføres en maskinordre? (Eksekveringsalgoritme)

Udførelsen af en maskinordre sker i et antal arbejdsstrin, som kan beskrives som følger:

- 1) Maskinordren hentes i lageret fra den celle, som lagerregisteret (MAR) angiver og placeres i ordregisteret (OR)  
Ordretælleren (OT) forøges med een, således at OT herefter indeholder adressen på NÆSTE KONSEKUTIVE MASKINORDRE.
- 2) Maskinordren fortolkes, dvs. at styreenheden på grundlag af maskinordrens operationsdel (OPKODEN) bestemmer HVAD der skal ske.

## Forslag til besvarelse af øvelsesopgave 3.4: Ordresekvenser

Bestemmelse af de oktale adresser for programstart samt for data A og B:

Programstart  $80_{10} = 120_8$   
A-adresse  $144_{10} = 220_8$   
B-adresse  $145_{10} = 221_8$

### Maskinprogram:

Adresse (celle nr)	Celleindhold (oktale værdier)	Kommentar
0120 <sub>8</sub>	060220 <sub>8</sub>	Hent indhold af 220 <sub>8</sub> til XR
0121 <sub>8</sub>	050221 <sub>8</sub>	Hent indhold af 221 <sub>8</sub> til YR
0122 <sub>8</sub>	022000 <sub>8</sub>	Flyt (XR) til AC
0123 <sub>8</sub>	040221 <sub>8</sub>	Gem (AC) i celle 221 <sub>8</sub>
0124 <sub>8</sub>	023000 <sub>8</sub>	Flyt (YR) til AC
0125 <sub>8</sub>	040220 <sub>8</sub>	Gem (AC) i celle 220 <sub>8</sub>
0126 <sub>8</sub>	000000 <sub>8</sub>	STOP
0220 <sub>8</sub>	000144 <sub>8</sub>	$A = 144_8 = 100_{10}$
0221 <sub>8</sub>	000012 <sub>8</sub>	$B = 12_8 = 10_{10}$

- 3) Operandadressen beregnes, dvs. at styreenheden på grundlag af maskinordrens adressedel bestemmer, HVOR i arbejdslageret operanden ligger placeret (dvs. bestemmer nummeret på den celle, HVORI operanden (data) ligger placeret).

Efter udførelse af arbejdsstrin 1)-3) ved styreenheden HVAD, den skal udføre, og HVOR operanden står.

- 4) Operanden hentes fra den lagercelle, der er beregnet under 3) og placeres i lagerbufferregistret (MBR).
- 5) Maskinordren udføres på operanden.
- 6) Næste ordreadresse beregnes, dvs. at styreenheden på grundlag af arbejdsstrin 2)-5) kan bestemme, HVOR i lageret NÆSTE MASKINORDRE ligger placeret.

Herefter fortsættes med udførelsen af NÆSTE MASKINORDRE fra 1).

Eksempel: Lad os antage, at celle nr  $100_8$  har følgende indhold:

15	12	11													0
0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0

og at indholdet i lageradresseregisteret (MAR) er  $0100_8$ .

- 1) Maskinordren  $060400_8$  hentes i celle  $100_8$  og placeres i operationsregisteret (OR); (OT) forøges med 1.
- 2) Operationskoden  $06_8$  afkodes af styreenheden, som nu ved, at ordren er HENT FRA ADR til XR.
- 3) Operandadressen  $0400_8$  i maskinordrens adressedel angiver direkte operandadressen og overføres til MAR.
- 4) Indholdet af celle  $0400_8$  (operandadressen) udlæses til lagerbufferregistret (MBR).
- 5) Indholdet i MBR overføres til arbejdsregisteret XR.
- 6) Da maskinordren ikke var en hopordre, findes NÆSTE MASKINORDRE i celle  $101_8$ . Denne værdi er allerede anbragt i OT (Se 1).



Det er karakteristisk for udførelsen af maskinordren, at den består af 2 faser:

Første fase bestående af arbejdstrin 1) - 3), hvor maskinordren hentes fra lageret til ordreregistret og fortolkes af styreenheden.

Anden fase bestående af arbejdstrin 4) - 6), hvor maskinordren udføres på operanden eller operanderne (eksekveres) af styreenheden.

Denne opdeling af maskinordrens udførelsessekvens i 2 faser er så karakteristisk og generel, at de 2 faser har fået standardiserede navne:

I-fase - Hente fase-(Første fase)

E-fase - Udførelses-fase (Anden fase).

#### 3.5.2. I-fase (Hente fase, Instruktionsfase, I-FETCH)

I-fasen (bestående af arbejdstrin 1) - 3)) udgøres af de arbejdstrin i udførelsessekvensen for en maskinordre, hvor maskinordren udlæses fra lageret til ordreregistret, og hvor derefter maskinordrens operationsdel og adressedel fortolkes af styreenheden.

#### 3.5.3. E-fase (Udførelses fase, eksekveringsfase, EXECUTE)

E-fasen (bestående af arbejdstrin 4) - 6)) udgøres af de arbejdstrin i udførelsessekvensen for en maskinordre, hvor den i foregående I-fase udlæste og fortolkede maskinordre eksekveres (udføres) af styreenheden.

### 3.6 Trinvis udførelse.

Til beskrivelse af hvorledes I- og E-faserne i detaljer udføres, refereres i det følgende til registerstrukturen fig. 2.12 samt specifikationen af styre-og følesignalerne i afsnit 2.7.5.

#### 3.6.1 Trinvis udførelse af I-fase.

Som beskrevet i afsnit 3.5.2. består I-fasen af følgende arbejdstrin:

- 1) Maskinordren hentes fra lageret (adresseret af MAR) til ordregisteret (OR); (OT) forøges med 1.
- 2) Maskinordren fortolkes på grundlag af maskinordrens ordredel (OPKODEN)
- 3) Operandadressen beregnes på grundlag af adressedelen i maskinordren.

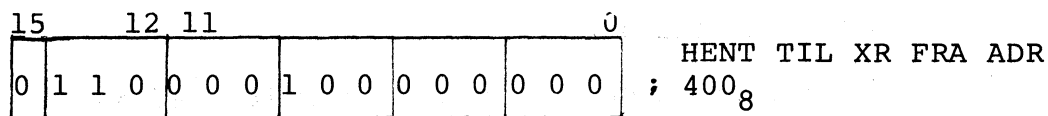
Disse arbejdsstrin kan i detaljer udføres ved følgende sekvens af registre til registeroperationer styret af styreenheden:

Operationer:	Aktiveret styrelinie nr.:	Kommentar:
1) { OT → MAR	15 ;	Indhold af OT overføres til MAR (OT indeholder adressen til næste ordre, se afsnit 2.5.2.)
L(MAR) → MBR	1 ;	Læs indhold af cellen, adresseret af MAR, til MBR
MBR → L(MAR)	2 ;	Tilbageskrivning af MBR til lagercellen adresseret af MAR
MBR → OR	5 ;	(MBR) til OR; OR bit 0...11 indeholder operandadressen, OR bit 12...15 indeholder OPKODEN)
OT+1 → OT	14 ;	Forøg OT med 1, således at NÆSTE ORDREADRESSE herefter står parat i OT.
2) { OPKODE → STYR	27 ;	Operationsdelen af maskinordren overføres fra OR til styreenheden, som afkoder OPKODEN.
	;	Ang. 3) beregning af operandadressen, er dette ikke nødvendigt i "vor simple maskine", idet operandadressen <u>direkte</u> står parat i OR bit 0...11.

### 3.6.2. Trinvis udførelse af E-fasen.

Medens I-fasen er generel og fuldstændig identisk for alle maskinordrer, er E-fasen speciel for hver enkelt maskinordre.

Lad os som eksempel betragte den trinvis udførelse af E-fasen for maskinordren



Som beskrevet i afsnit 3.5.1. består E-fasen af følgende arbejdsstrin:

- 4) Operanden hentes til MBR fra den lagercelle, som er angivet i operationsregisterets bit 0-11 (OR bit 0...11)
- 5) Maskinordren udføres på operanden
- 6) Næste ordreadresse udregnes i tilfælde af betinget hopordre.

Disse arbejdsstrin kan i detalje udføres ved følgende sekvens af register til registeroperationer styret af styreenheden:

Operationer:		Aktiveret styrelinie nr.:	Kommentar:
4) {	OR(0...11) → MAR	16 ;	Indhold af OR bit 0-11 overføres til MAR
	L(MAR) → MBR	1 ;	Læs indhold af cellen adresseret af MAR til MBR
	MBR → L(MAR)	2 ;	Tilbageskrivning af MBR til lagercellen adresseret af MAR
5) {	MBR → XR	3 ;	(MBR) til XR
6) {	HOP → I-fase	; "Styresignal" der tvinger styreenheden til at udføre en ny I-fase-sekvens.	

### 3.6.3. Mikroprogram

I fig.3.8 er den fuldstændige sekvens af register til registeroperationer (I-fase + E-fase), som er nødvendige for udførelsen af maskinordren "HENT TIL XR FRA ADR" gentaget :

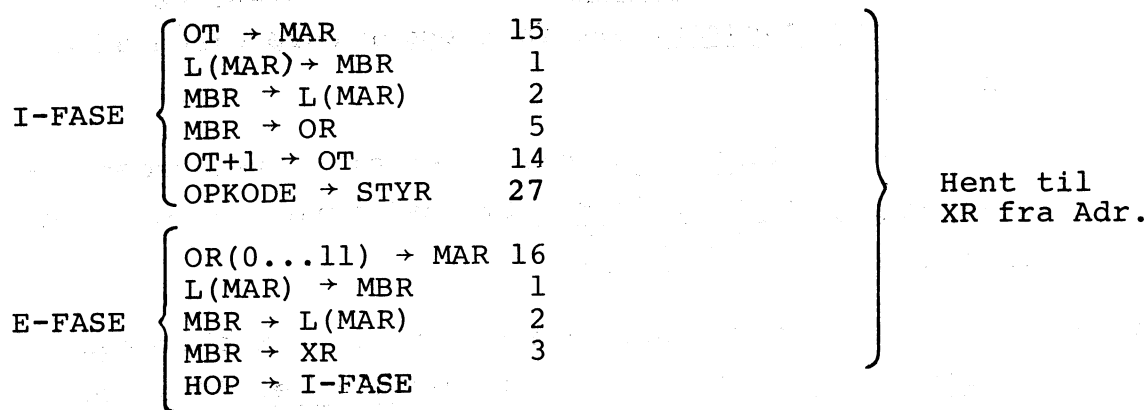


fig. 3.8

I fig.3.9 er til sammenligning vist den fuldstændige sekvens af register til registeroperationer (I-fase + E-fase), som er nødvendig for udførelsen af maskinordren ADDER (XR) OG (YR):

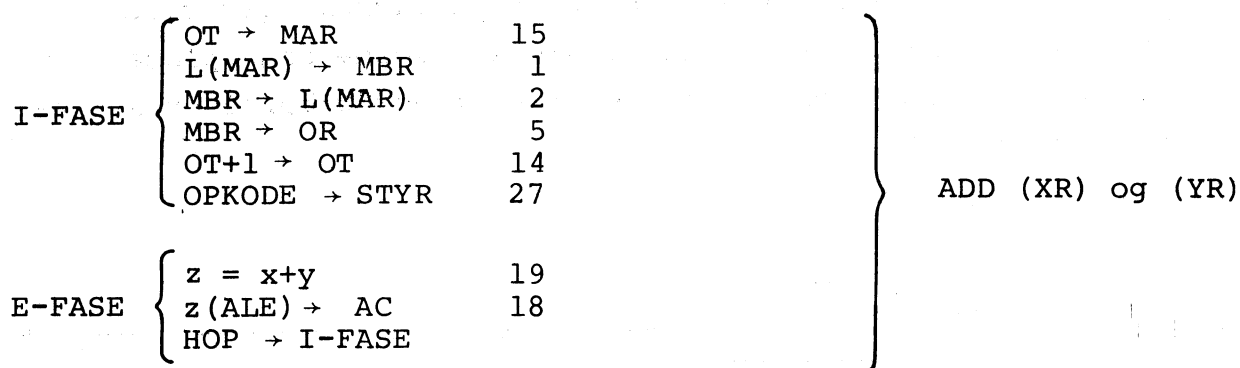


fig. 3.9

Vi ser, at I-faserne er identiske i de to eksempler, medens E-faserne er forskellige svarende til de 2 forskellige maskinordrer; men fælles for eksemplerne for de 2 valgte maskinordrer er, at de hver udføres som endelige og entydige sekvenser af styreoperationer.

Denne (understregede) formulering svarer ganske præcis til definitionen på et program (se afsnit 3.1.1.), og man kalder fig. 3.8 og 3.9 for mikroprogrammer.

Betegnelsen "mikro"-program refererer til, at programmet er endnu dybere knyttet til maskinens maskineldede, og de enkelte arbejdsskridt - mikroordrerne - endnu mere enkle og grundlæggende (nemlig register til registeroverførsler) end for maskinprogrammet og de tilhørende maskinordrer.

Definition: Ved et mikroprogram forstås en endelig og entydig sekvens af mikroordrer, der fastlægger udførelsen af en maskinordre.

Definition: Ved en mikroordre forstås et af styreenheden udførbart grundlæggende arbejds-skridt, realiseret som en enkelt register til registeroperation.

Enhver af datamaskinens maskinordrer kan altså realiseres som et mikroprogram af grundlæggende mikroordrer (dvs. register til registeroverførsler).

#### 3.6.4. Tidsdiagrammer for udførelse af maskinordre.

I fig.3.10 er vist et tidsdiagram for udførelsen af maskinordren HENT TIL XR FRA ADR, dvs. et tidsdiagram, der viser de enkelte mikroordres (styresignalers) tidsmæssige placering inden for det mikroprogram, der fastlægger udførelsen af maskinordren HENT.

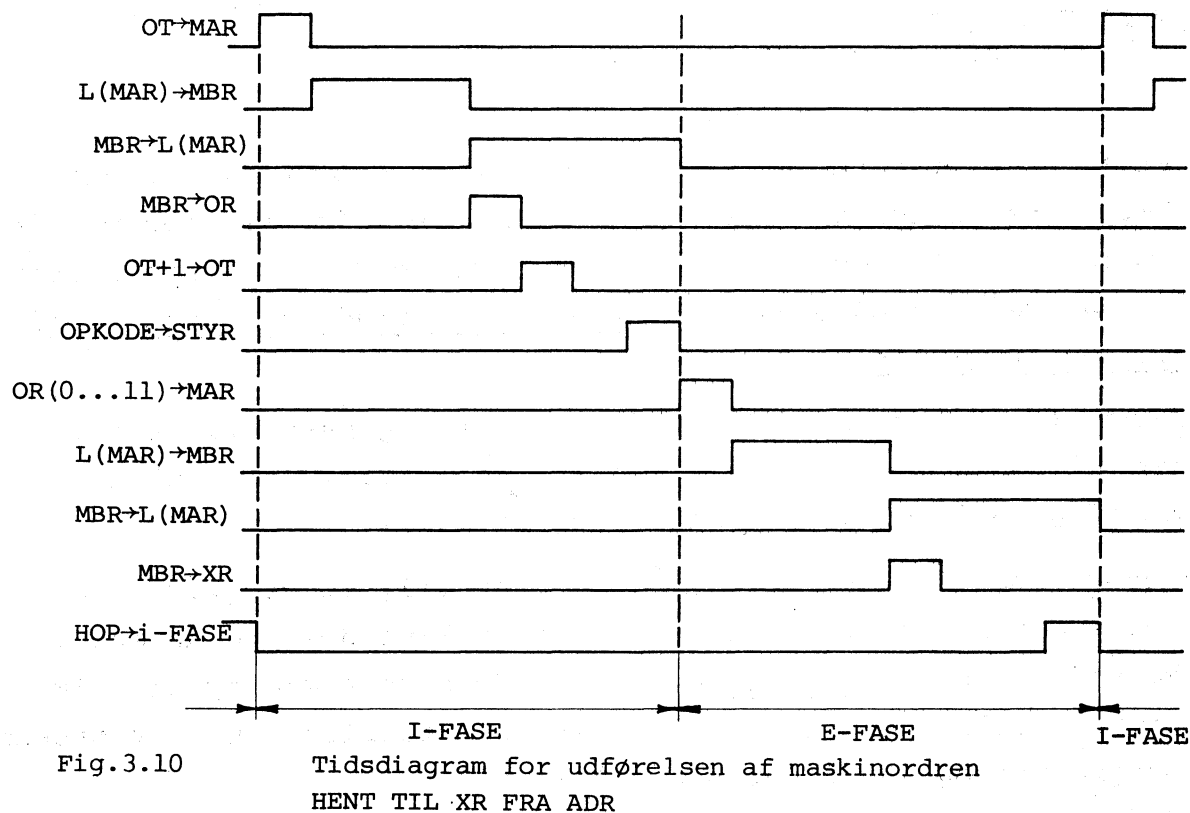


Fig.3.10

Tidsdiagram for udførelsen af maskinordren  
HENT TIL XR FRA ADR

### 3.7. Eksekvering af maskinordre ved hjælp af mikroprogram

3.7.1. Konstruktion af mikroprogram til eksekvering af maskinordren: SUB(traher)(XR) FRA (YR). (Øvelse)

Fetch 1 Fase

OT → MAR 15  
L(MAR) → MBR 1  
MBR → L(MAR) 2  
MBR → OR 5  
OT + 1 → OT 14  
OR CODE → STYR 27

E Fase EXECUTE

Z = Y - X 20  
Z(ALE) → AC 18  
Hop til 1 Fase

3.7.2. Konstruktion af mikroprogram til eksekvering af MANUEL START (Øvelse).

Når et maskinprogram skal startes manuelt, foregår det ofte på følgende måde:

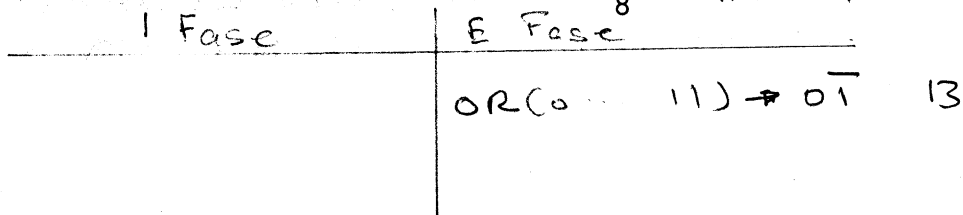
Et sæt datakontakter (KR) på frontpanelet sættes op i et bitmønster svarende til programmets startadresse i arbejdslageret. Derefter trykkes på START-knappen, og programmet starter i den nævnte adresse.

Lav et mikroprogram, som udfører MANUEL START, når operatøren trykker på START-knappen.

1 Fase

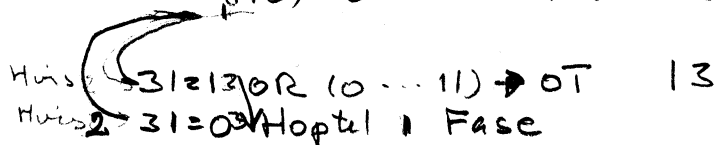
E Fase  
VR(0...11) → OT 12  
Hop til 1 Fase

- 3.7.3. Konstruktion af mikroprogram til eksekvering af maskinordren: HOP TIL ADR  $1200_8$ . (Øvelse)



- 3.7.4. Konstruktion af mikroprogram til eksekvering af maskinordren: HOP TIL ADR  $1200_8$ , HVIS (AC) = 0

(Øvelse)  $1^{st} (AC) = 0 \rightarrow$  STYR 28



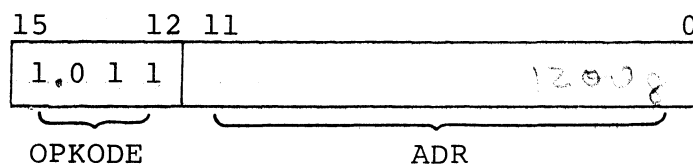
- 3.7.5. Den i afsnit 2,7 definerede datamaskine arbejder med direkte adressering, hvor adressen ved lageroverførselsordrer og hopordrer er angivet ved bit 0...11 i maskinordrens adressedel (fig. 3.2, 3.4, 3.6). Med direkte adressering er lagerstørrelsen således begrænset til  $2^{12} = 4096$  ord á 16 bit.

Hvis et større lager skal adresseres, kunne man tænke sig at anvende indirekte adressering.

Overvej hvilke maskinelle udvidelser, der er nødvendige i fig. 2.12 for at muliggøre adressering af et lager på 64 k-ord á 16 bit, samt konstruér et mikroprogram til eksekvering af maskinordren:

HOP (Indirekte) TIL ADR  $1200_8$  (Øvelse)

Definition af HOP(I) TIL ADR:



OPKODE =  $13_8$

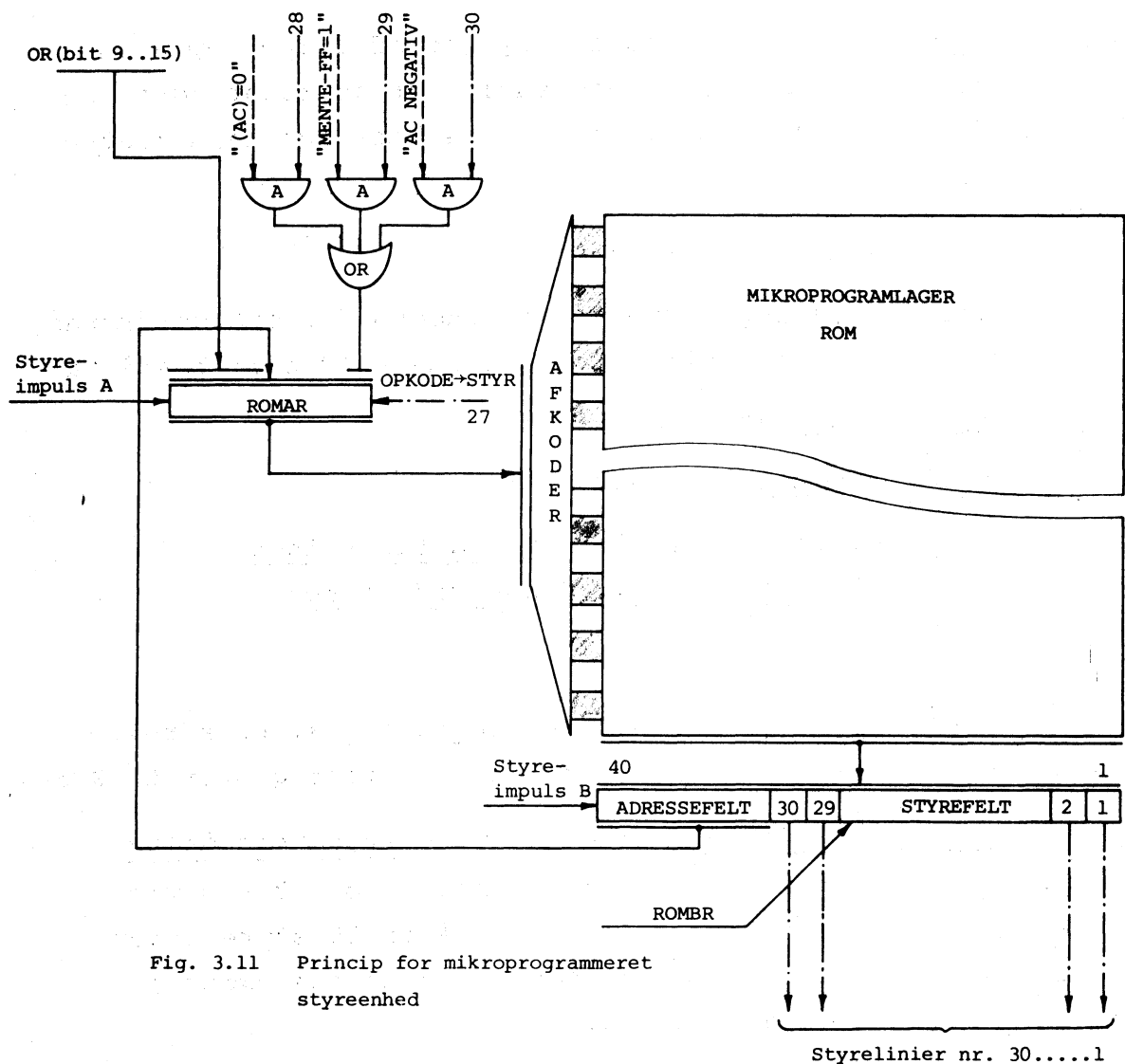
OR(0 ... 11)  $\rightarrow$  MAR 16  
L(MAR)  $\rightarrow$  MBR 1  
MBR  $\rightarrow$  L(MAR) 2  
MBR  $\rightarrow$  OT 13  
Hop til I fase

udvid OT  
udvid MAR  
BUS fra MBR  
til OT  
nyt styresignal

### 3.8. Principopbygning af mikroprogrammeret styreenhed

### 3.8.1. Grundlæggende princip for mikroprogrammeret styreenhed.

Det grundlæggende princip for en mikroprogrammeret styreenhed er vist på fig.3.11.



Styreenheden består grundlæggende af

et mikroprogramlager (der ofte er udformet som et læselager (ROM) med tilhørende adresseringselektronik (afkoder)

et databufferregister (ROMBR), hvortil indholdet af ROM-cellerne kan udlæses.



et lageradresseregister, hvis indhold adresserer læselageret (ROM).

Mikroprogramlageret (ROM) består af et antal celler (f.eks. 1024) af en vis ordlængde B bit, (f.eks. 40).

Databufferregisteret er opbygget af flip-flop's og har samme længde som læselagerets ordlængde dvs. B bit.

Adresseregisteret er ligeledes opbygget af flip-flop's og har en sådan længde, som er nødvendig for at kunne adressere hele læselageret (dvs. 10 bit, hvis læselagerkapaciteten er 1024 ord)

### 3.8.2. Beskrivelse af mikroordre.

Lagercellerne i mikroprogramlageret indeholder mikroordrerne, een mikroordre i hver celle. En mikroordre (B bit) består af

et styrefelt af bredde S bit

et adressefelt af bredde A bit

hvor S = antallet af styrelinier i datamaskinen

og A = antallet af bits i adresseregisteret

(B = S+A)

Ved hjælp af styreimpuls B udlæses indholdet af en adresse-ret celle (dvs. en mikroordre) til databufferregisteret (ROMBR).

De S flip-flop's i databufferregisterets styrefelt driver hver sin styrelinie (bit 1 - styrelinie 1, bit 2 - styrelinie 2 osv.), således at tilstanden på de 30 styrelinier, der udgør styreenhedens udsignaler, direkte bestemmes af bit 0-30 i styrefeltet i ROMBR.

Indholdet i adressefeltet i ROMBR angiver adressen (celle-nummeret) til næste mikroordre, og adressefeltet kan derfor kopieres til adresseregisteret ved hjælp af styreimpuls A.

Indsignalerne til styreenheden udgøres af følesignalerne OR (bit 11-15) samt følesignalerne "(AC) = 0", "Mente-FF =1", "(AC)NEG".

### 3.8.3. Principiel virkemåde

Enheden fungerer efter følgende princip:

Til tidspunkt  $t_n$  overføres en ny mikroprogramlageradresse (adr N) til ROMAR ved hjælp af styreimpuls A.

Indholdet i ROMAR adresserer celle N i mikroprogramlageret.

Til tidspunkt  $t_{n+1}$  udlæses indholdet af celle N til databufferregisteret ROMBR ved hjælp af styreimpuls B.

Herved sættes ROMBR-styrefeltets flip-flop's (og hermed datamaskinens styrelinier) op i overensstemmelse med den netop udlæste mikroordre, og ROMBR-adressefeltet angiver adressen til næste mikroordre.

Til tidspunkt  $t_{n+2}$  overføres denne nye adresse til ROMAR osv.

Fig.3.12 viser styreimpulsernes og et vilkårligt styresignals indbyrdes tidsafhængighed.

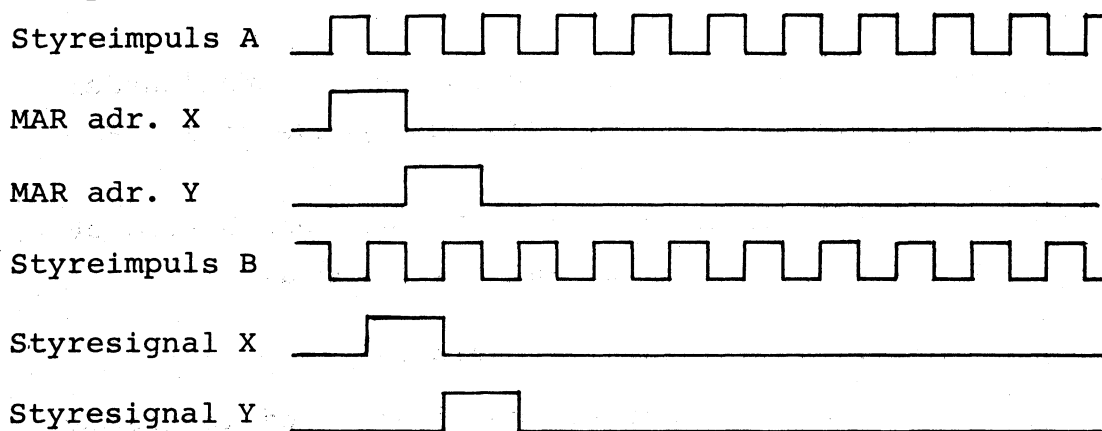


Fig. 3.12

Normalt vil adressefeltet indeholde cellenummeret på næste konsekutive celle i mikroprogramlageret, men principielt kan adressefeltet indeholde en vilkårlig adresse, således at ubetingede hop i mikroprogrammet let kan udføres.

I mikroprogramlageret har datamaskinfabrikanten indlagt de (faste) mikroprogrammer, som fastlægger maskinens maskinordrer. Mikroprogrammet er således normalt uforanderligt og helt uden for brugerens (programmørens) rækkevidde.

#### 3.8.4. Ubetingede mikrohopordrers udførelse.

Ubetingede hop i mikroprogrammerne kan som nævnt foretages ved at specificere hopadressen i mikroordrens adressefelt, idet adressefeltet efter mikroordrens udlæsning til ROMBR siden overføres til ROMAR som næste adresse.

Et typisk eksempel på ubetinget hop i mikroprogrammet har vi ved afslutningen af hver E-fase, idet alle E-fase-mikroprogrammer afsluttes med mikroordren HOP → I-FASE.

Dette ubetingede hop realiseres enkelt ved i næstsidste mikroordre i hver E-fase at specificere adressefeltets indhold til det bitmønster, som angiver adressen (cellenummeret) til den celle i ROM-lageret, hvor I-FASENs mikroprogram starter (se eks. i 3.8.6.)

En speciel type ubetingede mikroprogramhop sker som sidste arbejdsstrin i den trinvis udførelse af I-fasen (se afsn. 3.6.1.), idet sidste mikroordre i mikroprogrammet for I-fasen er ordren OPKODE → STYR (27)

Herved overføres indholdet af ordregisterets operationsdel (OR bit 11...15) direkte til ROMAR (de resterende bits i ROMAR nulstilles).

Afhængig af OPKODEN i ordregisteret OR foretages herved et ubetinget hop til det korrekte E-fase mikroprogram.

#### 3.8.5. Betingede mikrohopordrers udførelse.

Betingede hop i mikroprogrammet kan realiseres ved at fastlægge ROMAR's mindst betydende bit ved hjælp af et AND-OR-netværk som antydnet i fig. 3.11.

Adressen til næste mikroordre bestemmes for de (A-1) mest betydende bits vedkommende af adressefeltet i ROMBR og for den mindst betydende bits vedkommende af tilstanden på den følelinie, som udgør hopbetingelsen. Den relevante følelinie udvælges ved hjælp af styrelinierne i styrefeltet i ROMBR (styrelinie 28 eller 29 eller 30 på fig. 3.11).

Hvis hopbetingelsen er opfyldt, tvinges mindst betydende bit til 1; hvis hopbetingelsen ikke er opfyldt, tvinges mindst betydende bit i ROMAR til 0. De øvrige bits i adressen til næste mikroordre fastlægges af ROMBR-adressefeltets indhold.

### 3.8.6. Eksempel på realisation af maskinordren

HENT TIL XR FRA ADR i mikroprogrammeret styreenhed.

Det indkodede bitmønster, der er vist på fig. 3.13, modsvarende mikroprogrammet fig. 3.8 og det hertil hørende tidsdiagram fig. 3.10.

Mikroprogramlager (ROM)		
Mikroprogramlager-adresse	Adressefelt	Styrefelt
	40	31 30 1
0 0 1 0	0 0 0 0 0 0 1 0 0 1	bit 15=1, resten = 0
0 0 1 1	0 0 0 0 0 0 1 0 1 0	bit 1=1, resten = 0
0 0 1 2	0 0 0 0 0 0 1 0 1 1	bit 1=1, resten = 0
0 0 1 3	0 0 0 0 0 0 1 1 0 0	bit 1=1, resten = 0
0 0 1 4	0 0 0 0 0 0 1 1 0 1	bit 2=1, bit 5=1, resten=0
0 0 1 5	0 0 0 0 0 0 1 1 1 0	bit 2=1, bit 14=1, resten=0
0 0 1 6	0 0 0 0 0 0 1 1 1 1	bit 2=1, resten = 0
0 0 1 7	0 0 0 0 0 0 0 0 0 0	bit 2=1, bit 27=1, resten=0
I-FASE		
0 6 0 0	0 1 1 0 0 0 0 0 0 1	bit 16=1, resten = 0
0 6 0 1	0 1 1 0 0 0 0 0 1 0	bit 1=1, resten = 0
0 6 0 2	0 1 1 0 0 0 0 0 1 1	bit 1=1, resten = 0
0 6 0 3	0 1 1 0 0 0 0 1 0 0	bit 1=1, resten = 0
0 6 0 4	0 1 1 0 0 0 0 1 0 1	bit 2=1, bit 3=1, resten=0
0 6 0 5	0 1 1 0 0 0 0 1 1 0	bit 2=1, resten = 0
0 6 0 6	0 1 1 0 0 0 0 1 1 1	bit 2=1, resten = 0
0 6 0 7	0 0 0 0 0 0 1 0 0 0	bit 2=1, resten = 0
E-FASE		

Fig. 3.13

Mikroprogrammet for I-fasen er placeret i mikroprogramlagerets celle  $0010_8 \dots 0017_8$  med start i celle  $0010_8$ . Mikroprogrammerne svarende til E-faserne for de forskellige maskinordrer ligger placeret i mikroprogramlageret med start på adresser, hvis mest betydende bits bestemmes af OPKODEN i operationsregisteret OR, dvs. for vort eksempel "HENT TIL XR FRA ADR" (med OPKODEN  $06_8$ ) i celle  $0600_8$ .

### 3.9.1 Øvelsesopgave 3.9.1 og 3.9.2.

Realisation (implementering) af mikroprogrammerne fra øvelsesopgave 3.7.1. og 3.7.2 i mikroprogramlager med ordlængde 40

(se fig. 3.11 og 3.13).

Adressefeltet udgøres af bit 31-40

Styrefeltet udgøres af bit 0-30

Mikroprogrammet for I-fasen er allerede placeret (se figuren nedenfor) og ligger i celle  $0010_8$  -  $0017_8$  med start i celle  $0010_8$ .

Mikroprogramlager-adresse

Mikroprogramlagerindhold

Adressefelt

Styrefelt

	40	31	30	1							
0 0 1 0 <sub>8</sub>	0	0	0	0	1	0	0	1	bit 15=1, resten = 0		
0 0 1 1	0	0	0	0	0	0	1	0	1	0	bit 1=1, resten = 0
0 0 1 2	0	0	0	0	0	0	1	0	1	1	bit 1=1, resten = 0
0 0 1 3	0	0	0	0	0	0	1	1	0	0	bit 1=1, resten = 0
0 0 1 4	0	0	0	0	0	0	1	1	0	1	bit 2=1, bit 5=1,resten=0
0 0 1 5	0	0	0	0	0	0	1	1	1	0	bit 2=1, bit 14=1,resten=0
0 0 1 6	0	0	0	0	0	0	1	1	1	1	bit 2=1, resten = 0
0 0 1 7	0	0	0	0	0	0	0	0	0	0	bit 2=1, bit 27=1,resten=0
0 0 2 0											
0 0 2 1											
0 0 2 2											
0 0 2 3 <sub>8</sub>											

I-FASE

I-FASE

E-fase svarende til øvelse 3.7.1: SUB(XR) FRA(YR)

0110  
0111

0001001000 bit 20=1 resten 0  
0000001000 bit 18=1, bit 20=1 resten 0

E-FASE (SUB)

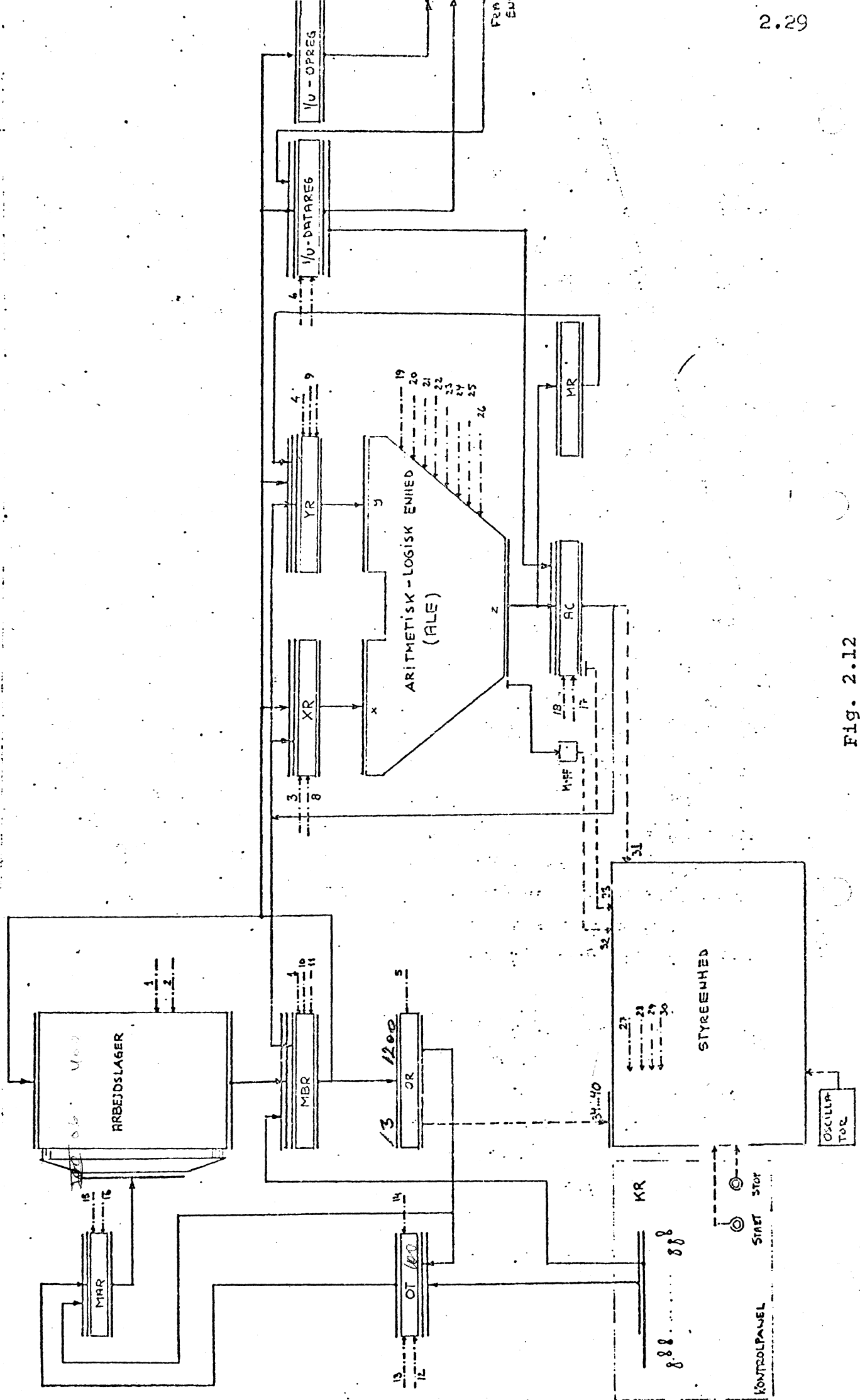
Mikroprogram svarende til øvelse 3.7.2: MANUEL START

\* 1 7 7 7<sub>8</sub> I fase  
0010

0000001000 bit 12=1 resten 0  
0000001001 bit 15=1 resten 0

MANUEL START

- \* Ved aktivering af START-knappen tænkes bitmønsteret  $1777_8$  tvunget ind i ROMAR, hvorefter styreimpulserne A og B slippes fri, startende med styreimpuls B.



Forslag til opgavebesvarelse til øvelse 3.7.1:

Konstruktion af mikroprogram til eksekvering af maskinordren

SUB(traher) (XR) FRA (YR)

Det forudsættes, at operanderne i forvejen er placeret i XR og YR.

Dette kan ske ved hjælp af maskinordrerne

HENT TIL XR FRA ADR

HENT TIL YR FRA ADR

[SUB (XR) FRA (YR)]

Følgende sekvens af mikroordrer (mikroprogram) vil udføre maskinordren SUB (XR) FRA (YR)

	Mikroordre:	Aktiveret styrelinie nr.:	Kommentar:
Hente-fase	OT → MAR	15 ;	(OT) kopieres over i MAR
	L(MAR → MBR	1 ;	Udlæs indhold af den celle, som MAR adresseret til MBR.
	MBR → L(MAR)	2 ;	Tilbageskrivning af (MBR)
	MBR → OR	5 ;	(MBR) kopieres over i OR
	OT+1 → OT	14 ;	(OT) forøges med 1
	OPKODE → STYR	27 ;	Operationskoden (OR 9-15) kopieres over til styreenheder
Udførelses-fase	$z = y - x$	20 ;	ALE udfører subtraktionen $z = y - x$
	z(ALE) → AC	18 ;	Resultatet af $z = y - x$ overføres til AC
	HOP → I-FASE	;	HOP til 1. mikroordre i I-faser

Forslag til opgavebesvarelse til øvelse 3.7.2:

Konstruktion af mikroprogram til eksekvering af MANUEL START

Når START-knappen på frontpanelet indtrykkes (dvs. når "følelinien" START aktiveres), skal styreenheden eksekvere følgende sekvens af mikroordrer (mikroprogram):

Mikroordre:	Aktiveret styrelinie nr.:	Kommentar:
<i>for datakopiering</i> KR(0...11) → OT	12	; Datakontakternes bitmønster kopieres over i OT
HOP → I-FASE		; HOP til 1. mikroordre i I-fasen



Forslag til opgavebesvarelser:Til øvelse 3.7.3: HOP TIL ADR 1200<sub>8</sub>

Udførelsesfase:

Mikroordre:	Aktiveret styrelinie nr.:	Kommentar:
OR(0...11) → OT	13 ;	Adressen til næste ordre overføres fra adressedelen af OR til ordretælleren OT.
HOP → I-FASE	;	

Til øvelse 3.7.4: HOP TIL ADR 1200<sub>8</sub> HVIS (AC) = 0

Udførelsesfase:

Mikroordre:	Aktiveret styrelinie nr.:	Kommentar:
"(AC)=0" → STYR	28 ;	Overfør tilstanden på følesignal 31 til styreenhed.
HVIS "(AC)=0"		
OR(0...11) → OT	13 ;	Hvis betingelsen "(AC)=0" er opfyldt, overføres adressen til næste ordre fra adressedelen af OR til OT.
HOP → I-FASE	;	

Til øvelse 3.7.5: HOP(I) TIL ADR 1200<sub>8</sub>

Udførelsesfase:

Mikroordre:	Aktiveret styrelinie nr.:	Kommentar:
OR(0...11) → MAR	16 ;	Adressen til den celle, som indeholder den <u>effektive adresse</u> (16 bit) overføres til MAR.
L(MAR) → MBR	1 ;	Læs effektiv adr. til MBR.
MBR → L(MAR)	2 ;	Skriv tilbage.
MBR → OT	(x) ;	Effektiv adresse overføres fra MBR til ordretæller OT.
HOP → I-FASE	;	

## Maskineltilføjelser:

- 1) OT og MAR udvides fra 12 til 16 bits længde.
- 2) Datatransportvej fra lagerbufferregister MBR til ordretæller OT.
- 3) Definition af styresignal MBR → OT, styrelinje nr. (x)

Forslag til opgavebesvarelse af øvelsesopgave 3.9.3, 3.9.4 og 3.9.5.

Realisation (implementering) af mikroprogrammerne fra øvelsesopgave 3.7.3, 3.7.4 og 3.7.5 i mikroprogramlager med ordlængde 40 (se fig. 3.11 og 3.13).

Adressefeltet udgøres af bit 31-40

Styrefeltet udgøres af bit 0-30

Mikroprogrammet i I-fasen er allerede placeret (se figuren nedenfor) og ligger i celle 0010<sub>8</sub> - 0017<sub>8</sub> med start i celle 0010<sub>8</sub>.

Mikroprogramlager-adresse

Mikroprogramlagerindhold

Adressefelt

Styrefelt

	40	31	30		1							
0 0 1 0 <sub>8</sub>	0	0	0	0	0	0	1	0	0	1	bit 15=1, resten = 0	
0 0 1 1	0	0	0	0	0	0	1	0	1	0	bit 1=1, resten = 0	
0 0 1 2	0	0	0	0	0	0	1	0	1	1	bit 1=1, resten = 0	
0 0 1 3	0	0	0	0	0	0	1	1	0	0	bit 1=1, resten = 0	
0 0 1 4	0	0	0	0	0	0	1	1	0	1	bit 2=1, bit 5=1, resten=0	
0 0 1 5	0	0	0	0	0	0	1	1	1	0	bit 2=1, bit 14=1,resten=0	
0 0 1 6	0	0	0	0	0	0	1	1	1	1	bit 2=1, resten = 0	
0 0 1 7	0	0	0	0	0	0	0	0	0	0	bit 2=1, bit 27=1,resten=0	I-FASE
1 4 0 0	0	0	0	0	0	0	1	0	0	0	bit 13=1, resten = 0	E-FASE 3 7 2
1 5 0 0	1	1	0	1	0	0	0	0	1	0	bit 28=1, resten = 0	
1 5 0 1	0	0	0	0	0	0	1	0	0	0	alle bit = 0	
1 5 0 2	0	0	0	0	0	0	1	0	0	0	bit 13=1, resten = 0	E-FASE 3 7 4
1 5 0 3	0	0	0	0	0	0	1	0	0	0		
1 3 0 0	1	0	1	1	0	0	0	0	0	1	bit 16=1, resten = 0	
1 3 0 1	1	0	1	1	0	0	0	0	1	0	bit 1=1, resten = 0	
1 3 0 2	1	0	1	1	0	0	0	0	1	1	bit 1=1, resten = 0	
1 3 0 3	1	0	1	1	0	0	0	1	0	0	bit 1=1, resten = 0	
1 3 0 4	1	0	1	1	0	0	0	1	0	1	bit 2=1, bit x=1,resten=0	
1 3 0 5	1	0	1	1	0	0	0	1	1	0	bit 2=1, resten = 0	
1 3 0 6	1	0	1	1	0	0	0	1	1	1	bit 2=1, resten = 0	
1 3 0 7	0	0	0	0	0	0	1	0	0	0	bit 2=1, resten = 0	E-FASE 3.7.5

# Forslag til opgavebesvarelse af øvelsesopgave 3.9.1 og 3.9.2

Realisation (implementering) af mikroprogrammerne fra øvelsesopgave 3.7.1. og 3.7.2 i mikroprogramlager med ordlængde 40 (se fig. 3.11 og 3.13).

Adressefeltet udgøres af bit 31-40

Styrefeltet udgøres af bit 0-30

Mikroprogrammet for I-fasen er allerede placeret (se figuren nedenfor) og ligger i celle  $0010_8 - 0017_8$  med start i celle  $0010_8$ .

Mikropro-  
gramlager-  
adresse

## Mikroprogramlagerindhold

Adressefelt

Styrefelt

	40	31	30	1	
0 0 1 0 <sub>8</sub>	0	0	0	1	bit 15=1, resten = 0
0 0 1 1	0	0	0	0	bit 1=1, resten = 0
0 0 1 2	0	0	0	0	bit 1=1, resten = 0
0 0 1 3	0	0	0	0	bit 1=1, resten = 0
0 0 1 4	0	0	0	0	bit 2=1, bit 5=1,resten=0
0 0 1 5	0	0	0	0	bit 2=1, bit 14=1,resten=0
0 0 1 6	0	0	0	0	bit 2=1, resten = 0
0 0 1 7	0	0	0	0	bit 2=1, bit 27=1,resten=0
0 0 2 0					
0 0 2 1					
0 0 2 2					
0 0 2 3 <sub>8</sub>					

## E-fase svarende til øvelse 3.7.1: SUB(XR) FRA(YR)

0 1 1 0 <sub>8</sub>	0	0	0	1	0	0	1	0	0	1	bit 20=1, resten = 0
0 1 1 1 <sub>8</sub>	0	0	0	0	0	0	0	1	0	0	bit 20=1, bit 18=1, resten=0

## Mikroprogram svarende til øvelse 3.7.2: MANUEL START

* 1 7 7 7 <sub>8</sub>	0	0	0	0	0	0	1	0	0	0	bit 12=1, resten = 0
------------------------	---	---	---	---	---	---	---	---	---	---	----------------------

Rev. 1 \* Ved aktivering af START-knappen tænkes bitmønsteret 1777<sub>8</sub> tvunget ind i ROMAR, hvorefter styreimpulserne A og B slippes fri, starten med styreimpuls B.

### Øvelsesopgave 3.9.3, 3.9.4 og 3.9.5

Realisation (implementering) af mikroprogrammerne fra øvelsesopgave 3.7.3, 3.7.4 og 3.7.5 i mikroprogramlager med ordlængde 40 (se fig. 3.11 og 3.13)

Adressefeltet udgøres af bit 31 - 40

Styrefelt udgøres af bit 0 - 30

Mikroprogrammet i I-fasen er allerede placeret (se figuren nedenfor) og ligger i celle 0010<sub>8</sub> - 0017<sub>8</sub> med start i celle 0010<sub>8</sub>.

<u>Mikroprogramlager- adresse</u>	<u>Mikroprogramlagerindhold</u>		
	Adressefelt	Styrefelt	
	40	31 30	1
0 0 1 0 <sub>8</sub>	0 0 0 0 0 0 1 0 0 1	bit 15=1, resten = 0	I-FASE
0 0 1 1	0 0 0 0 0 0 1 0 1 0	bit 1=1, resten = 0	
0 0 1 2	0 0 0 0 0 0 1 0 1 1	bit 1=1, resten = 0	
0 0 1 3	0 0 0 0 0 0 1 1 0 0	bit 1=1, resten = 0	
0 0 1 4	0 0 0 0 0 0 1 1 0 1	bit 2=1, bit 5=1, resten=0	
0 0 1 5	0 0 0 0 0 0 1 1 1 0	bit 2=1, bit 14=1, resten=0	
0 0 1 6	0 0 0 0 0 0 1 1 1 1	bit 2=1, resten = 0	
0 0 1 7	0 0 0 0 0 0 0 0 0 0	bit 2=1, bit 27=1, resten=0	
1400	E-fase svarende til øvelse 3.7.3		E-FASE 3.7.3
	0 0 0 0 0 0 1 0 0 0	bit 13=1 resten 0	
1500	E-fase svarende til øvelse 3.7.4		E-FASE 3.7.4
1502	1 1 1 0 0 0 0 0 1 0	Bit 28=1 resten 0	
1503	0 0 0 0 0 0 1 0 0 0	Bit 28=1 Bit 13=1 Bit 28=1	
	E-fase svarende til øvelse 3.7.5		E-FASE 3.7.5

#### 4. DATAMASKINENS IND/UDLÆSE-SYSTEM

##### 4.1. Relevante begreber

For at kunne kommunikere med omverdenen er det nødvendigt, at datamaskinen har et indlæse (Input) og udlæse (Output) system.

Datamaskinens omverden er de tilsluttede ydre enheder. Datamaskinens funktion er i høj grad begrænset af hvilke ydre enheder der er tilkoblet.

Programmerne skrives så datamaskinen på hensigtsmæssig vis henter data (Input) fra ydre enheder, behandler data og sender data (Output) til ydre enheder.

De ydre enheder bliver nærmere omtalt i afsnit 5.

Datatransporten til og fra de ydre enheder foregår via et I/O system (Input/Output), som består af en række interface- (grænseflade) print, et for hver tilsluttet ydre enhed.

Disse interfaceprint er stort set ens opbygget, uanset hvilken enhed der er tale om.

Dette indebærer, at de programrutiner, som styrer ind- og udlæsning, bliver næsten ens uanset hvilke ydre enheder der er tale om.

##### 4.1.1. Princip for I/O

Der er adskillige måder, på hvilken data kan overføres til/fra en datamat og dens ydre enheder; men de falder alle indenfor følgende tre kategorier:

1. Programstyret I/O (Vente på flag metoden).
2. Interrupt I/O.

### 3. Direct Memory Acces (DMA).

#### Programstyret I/O

Ved denne metode styres al datakommunikation af et brugerprogram, der eksekveres i datamaten.

Hovedkarakteristikken ved denne metode er, at datamaten programmæssigt "aftaster" om en ydre enhed er klar til dataoverførsel af nye data, for derefter at lade selve dataoverførslen ske programmæssigt. (Se afsnit 4.3.).

#### Interrupt I/O

Interrupt metoden giver den ydre enhed mulighed for at afbryde et igangværende program og tvinge datamaten til øjeblikkelig at hoppe til en programrutine, der kan betjene den ydre enhed. Selve betjeningen af den ydre enhed foregår programmæssigt. (Se afsnit 4.4.).

#### Direct Memory Acces (DMA)

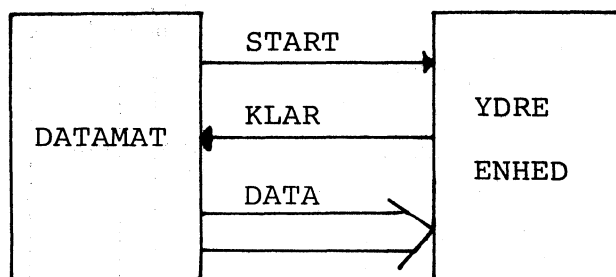
Denne form for dataoverførsel tillader, at data overføres direkte mellem datamatens lager og den ydre enhed uden at involvere datamatens styreenhed; dvs. at overførslen foregår udelukkende ved hjælp af materiellet uden medvirken af et brugerprogram. (Se afsnit 4.5.).

Ved overførsel af data mellem en hurtig enhed (datamat) og en langsom enhed (ydre enhed), foregår datakommunikationen ofte asynkront.

Det er derfor nødvendigt, at overførslen af data sker efter en klart defineret procedure, der til enhver tid giver sikkerhed for korrekt dataoverførsel.

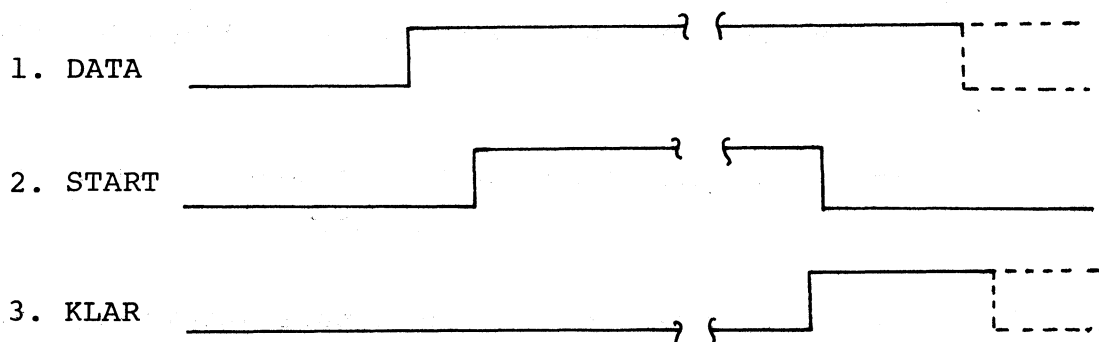
Sådanne transmissionsprocedurer kaldes ofte "handshake-procedurer".

En handshake-procedure for udlæsning af data fra en datamat til en ydre enhed er skitseret nedenfor.



1. Datamatens udlæseenhed placerer uddata på databussen.
2. Datamatens udlæseenhed generer et START signal til den ydre enhed.
3. Det er nu den ydre enheds opgave at modtage og behandle de data, der står på bussen. Når den ydre har færdigbehandlet dataene, sender den en kvittering til datamatens udlæseenhed. Denne kvittering betyder, at den ydre enhed er KLAR til at modtage nye data.

En handshake-procedure kan forløbe som skitseret på nedenstående timing-diagram.



Ved indlæsning af data kan tilsvarende handshake-procedurer etableres.



#### 4.1.2. Blokdiagram for I/O.

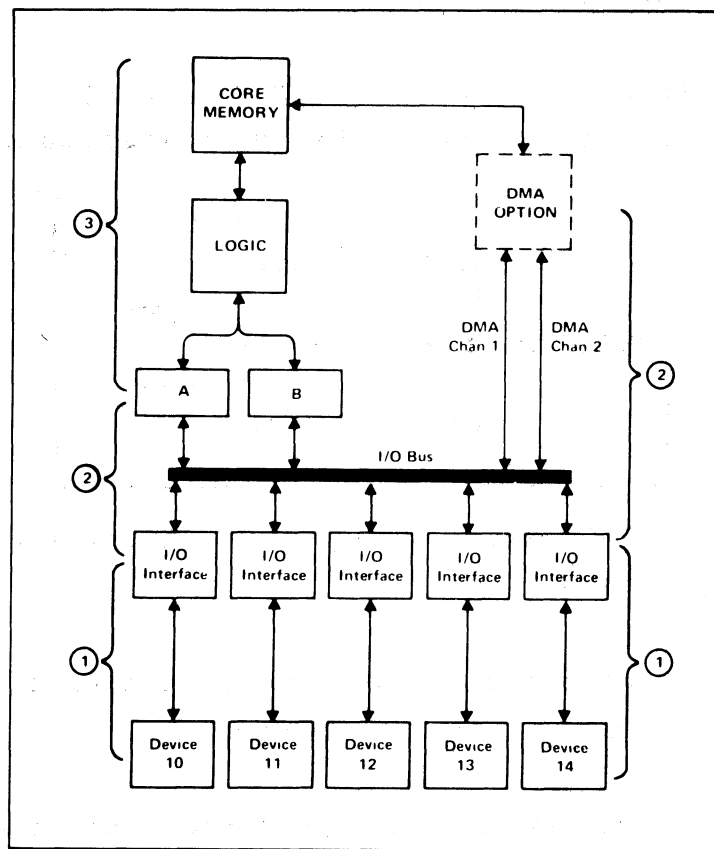


Fig. 4.1.1 (Hewlett Packard)

Fig. 4.1.1. viser hvorledes dataoverførslen foregår i 3 trin.

Ved for eksempel Input sker der følgende:

1. Data overføres parallelt fra den ydre enhed til et register på interfaceprintet. Transporten styres af den ydre enhed.
2. Interfaceenheden melder med et kontrolsignal til datamaten, at data er modtaget.

Datamaten overfører med en Inputinstruktion, data fra interfacekortets register til register A eller B.

3. Data overføres fra A eller B registeret til den udpegede lagercelle.

Ved output foregår de 3 trin i modsat rækkefølge.

Har man ydre enheder, som kan sende (Input) eller modtage (Output) data hurtigere end det tager at afvikle det ovenfor beskrevne program, må datamaten udbygges med en Direct Memory Access (DMA) kanal.

En DMA kanal kan standse datamaskinen et øjeblik, overtage kontrol over Lager Adresse Register og Lager Buffer Register, overføre et eller flere ord til eller fra lageret og igen frigive styreenheden.

#### 4. 2. Interfaceprintets opgaver

Interfaceprintets opgaver er følgende:

1. Opbevaring af et dataord, der enten er overført fra datamaten eller fra den ydre enhed.
2. Overføring af en startkommando fra datamaten til den ydre enhed.
3. Overføring af en klarmelding fra den ydre enhed til datamaten.
4. Generere afbrydesignal til datamaten når den ydre enheder klar, og samtidig forhindre enheder med lavere prioritet i at sende afbrydesignal.
5. Identificere sig selv ved et nummer.



Disse opgaver kan løses på mange forskellige måder. Hver datamaskinefabrikant har sin måde at lave "standard" interfaceprint på.

Fig. 4.2.1. viser et generelt og forenklet diagram af et Input/output system.

Sammenkoblingen mellem centralenhed og interface sker via 3 busser.

1. En 16 bit databus DO - D 15  
Udgangene fra samtlige dataregistre og CPU'ens akkumulatorer er parallelkoblet til databussen, og må derfor nødvendigvis være THREE-STATE eller åben kollektor kredse.
2. En 6 bit I/O adressebus DSO - DS 5  
(Device Select nr.) som går fra de 6 mindst betydende bit i adresseregisteret til en adresse-dekoder på hvert print.
3. En kontrolbus som overfører kommando og status signaler mellem CPU og alle interfaceprint.

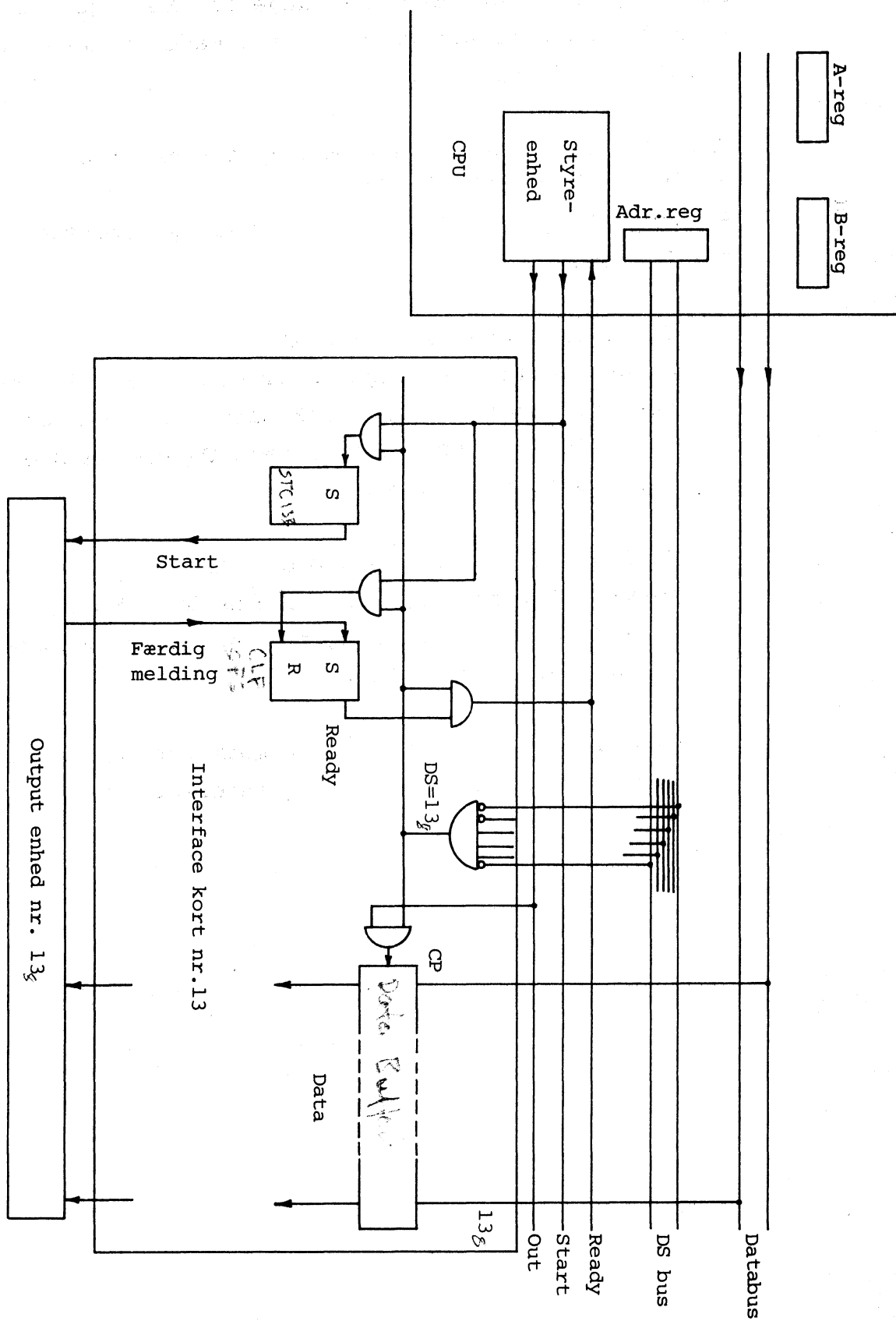


Fig. 4.3.1.

### 4.3. Programstyret I/O

#### 4.3.1. Output program

Fig. 4.3.1. viser en del af interfacen til en output enhed (f.eks. enhed nr. 13 en strimmel perforator).

Datatransporten kan foregå som følger:

1. Overfør data fra A eller B til interfacekort for enhed 13.

Dette gøres ved at A eller B reg. styrer databussen og DS-bussen adresserer enhed 13 til at give kloksignal til interfacerregisteret.

2. Start 13

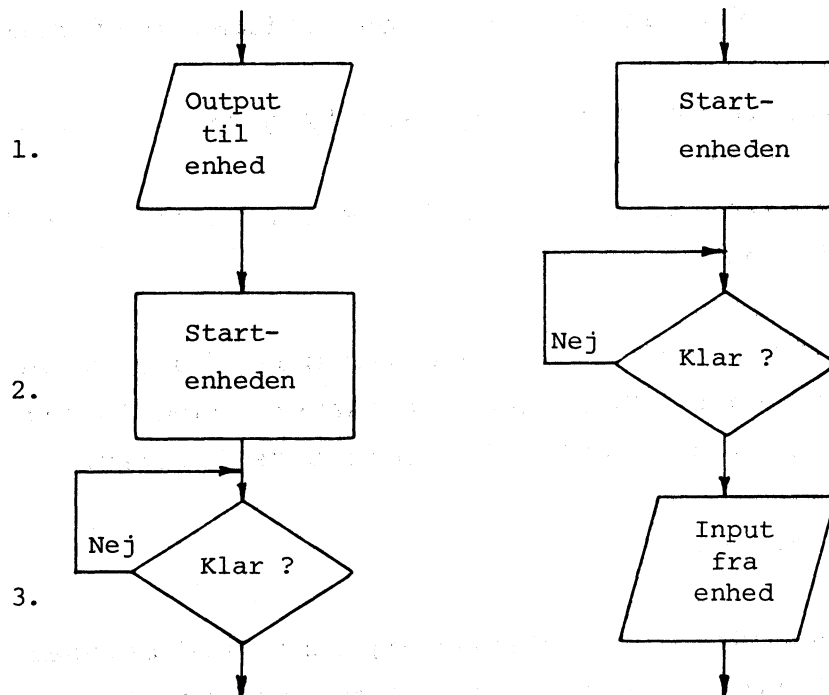
Den ydre enhed startes og dens færdigsignal nulstilles. Dette gøres ved at udsende et startsignal til alle enheder samtidig med et adresse-nr. på DS-linierne. Kun i den udvalgte enhed sættes Start F-F'en.

3. Man tester om enheden er klar til at modtage nye data ved at udsende DS-nr. og teste på READY signalet.

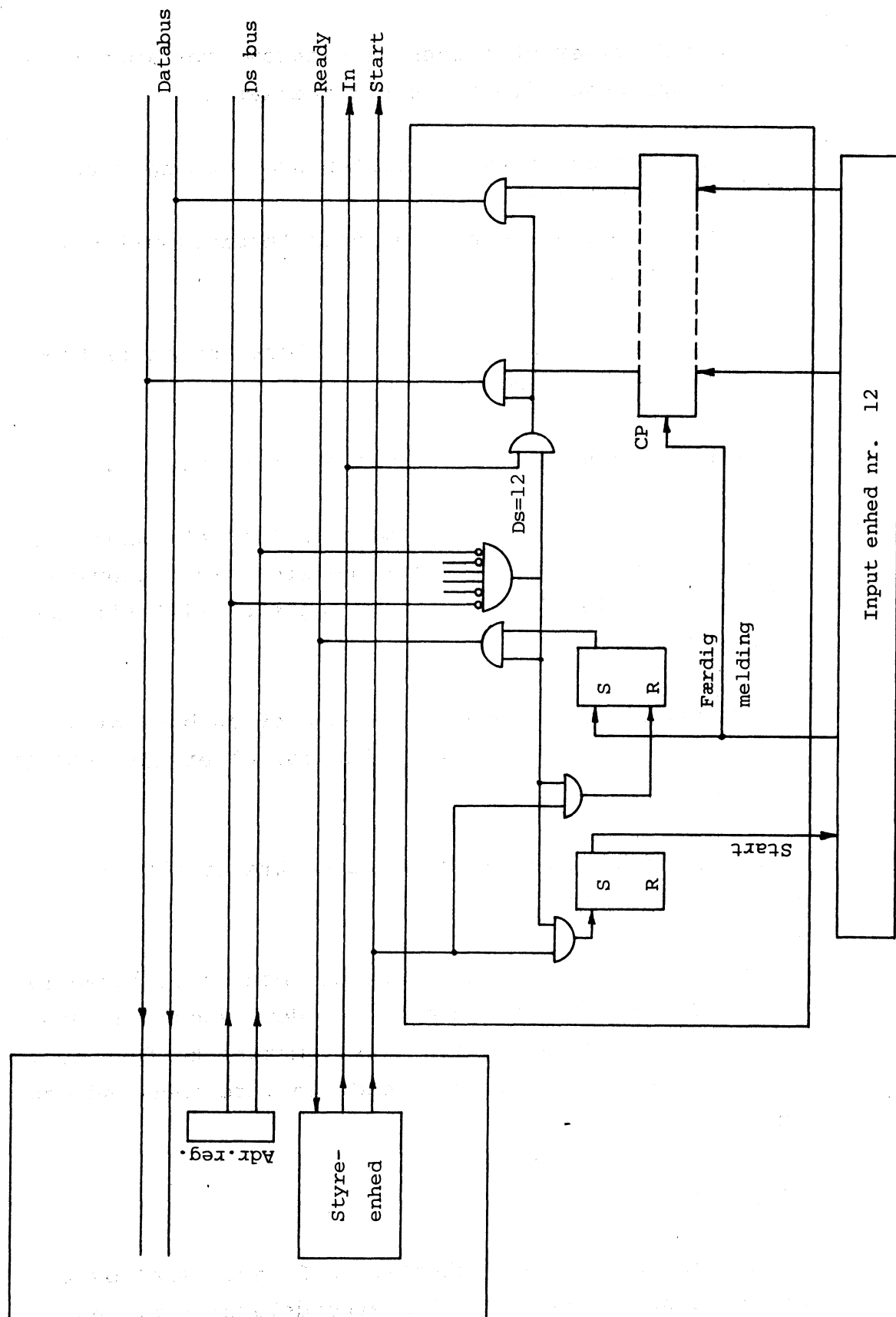
Hvis enheden ikke er klar gentages punkt 3.

Hvis enheden er klar (Motor på fuld hastighed og sidst udsendte karakter er hullet) gås videre i programmet.

Dataoverførslen kaldes "Vente på Flag" metoden og kan beskrives således i blokdiagramform:



Den viste programstump er normalt en generelt tilgængelig subrutine i operativsystemet, som kan kaldes fra brugernes programmer.



Input enhed nr. 12

Fig. 4.3.3



#### 4.3.2. Input-program

Fig. 4.3.2. viser et forenklet diagram over interface til en inputenhed f.eks. en strimmellæser.

Programstyret kan datatransporten ske, som følger:  
(Se blokdiagram side 4.10).

1. Den ydre enhed startes og dens færdigsignal nulstilles, som ved output.
2. Man tester om enheden har overført den læste karakter til dataregisteret.

Hvis enheden ikke er færdig gentages punkt 2.

3. Hvis enheden er færdig indlæses fra det valgte inputregister til A eller B register via databussen.  
(In DS = 12) åbner fra data register (12) til databussen.

Alle udgange, som parallellforbindes på en buslinie, skal være åben collektor eller THREE-STATE kredse.

Metoden er simpel at programmere og kræver ikke ret megen interfacelogik.

Ulemperne består i, at datamaskinen bruger en anseelig del af sin tid til at vente på, at den langsomme ydre enhed bliver færdig med sin elektromekaniske funktion, samt, at man kun kan arbejde med een ydre enhed ad gangen.

Spørgsmål:

En datamaskine med en cyklustid på 2  $\mu$ sec. skal overføre en større datamængde fra arbejdslageret til en "hurtig" strimmelperforator (75 tegn pr. sek.).

Hvor mange % af tiden bruges ca. til at "vente på flag"?

99.8%

#### 4. 4. Interrupt     styring fra Ydre Enheder

##### 4.4.1. Interrupt hvorfor

Interrupt er det engelske ord for afbrydelse og tilkendegiver, at de ydre enheder får mulighed for at afbryde datamaskinen i det kørende program.

Som det fremgår af det foregående regneeksempel, kan effektiviteten ved betjening af ydre enheder forøges voldsomt ved anvendelse af en bedre strategi.

##### 4.4.2. Interruptenhedens opgaver

Ved interruptstyret input er hændelsesforløbet, som vist i fig. 4.4.1.

1. Enhed nr. 12 startes v.h.a. instruktionen Set Control 12, Clear Flag.
2. Control F-F'en starter den ydre enhed.  
  
(Programafviklingen fortsætter uden at vente på færdigmelding fra den ydre enhed).
3. Karakteren er læst og overføres til interfaceprintets buffer. Samtidig sætter den ydre enhed færdig Flag.
4. Efter udførelsen af enhver instruktion, tester styreenheden om en af de ydre enheder har sat Flag. Hvis det er tilfældet hentes enhedens nr. ind i
5. styreenheden, og der startes en subrutine til betjening af den pågældende ydre enhed.
6. Inden subrutinen er færdig, bringes interfacekortet tilbage til sin begyndelsestilstand.

Til sidst returneres til det afbrudte program.

Ved interruptstyret Output er hændelsesforløbet, som vist i fig. 4.4.2.:

1. Een karakter overføres fra akkumulator A til enhed nr. 13's buffer.
2. Enheden startes v.h.a. instruktionen Set Control 13, Clear Flag, og programafviklingen fortsætter.
3. Data overføres til enheden, som starter sin elektromekaniske funktion.
4. Når enheden har udført sin funktion sætter den færdig Flag i interfaceenheden.
5. Flaget giver interrupt til datamaskinen,
6. som afbryder det kørende program og starter en subrutine for enhed nr. 13.
7. Inden subrutinen er færdig, bringes interfacekortet tilbage til sin begyndelsestilstand, og der returneres til det afbrudte program.

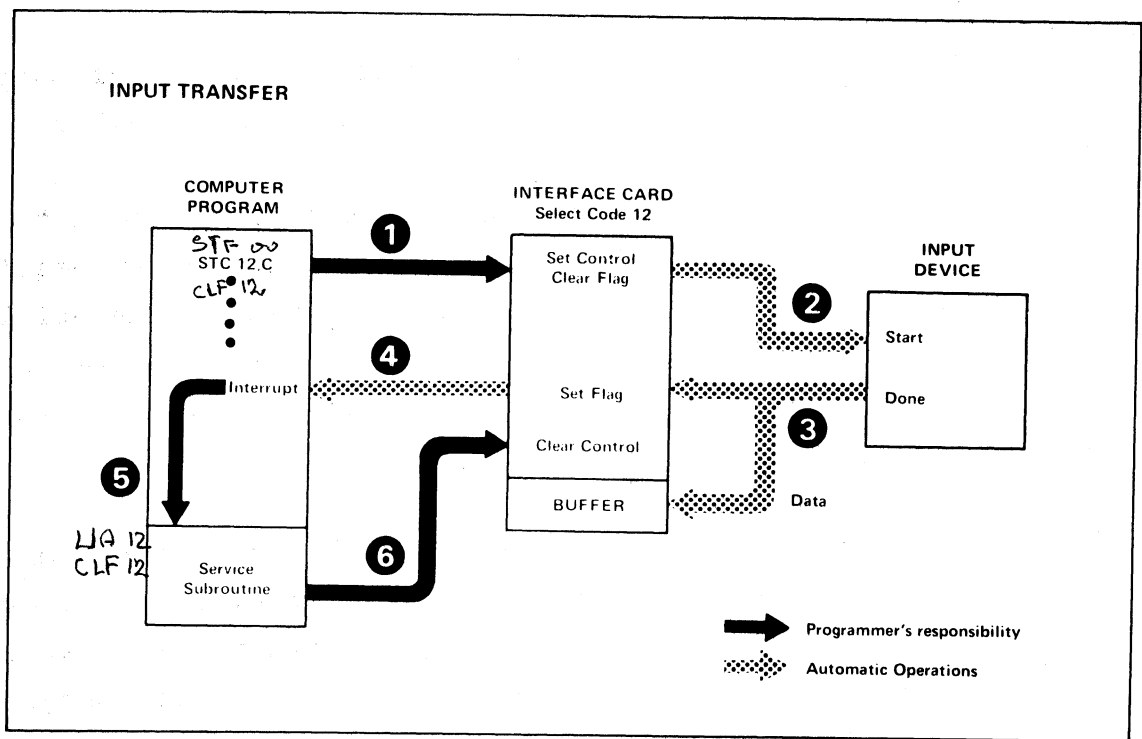


Fig. 4.4.1

(Hewlett Packard)

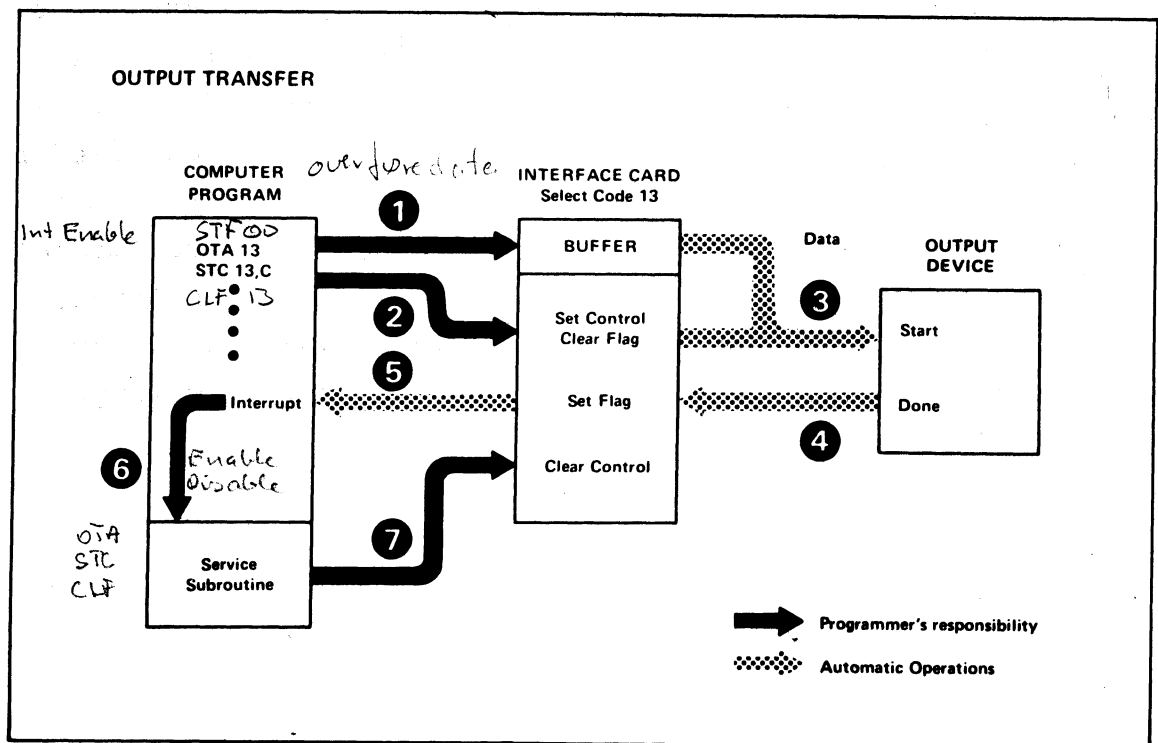


Fig. 4.4.2

(Hewlett Packard)

Det netop beskrevne hændelsesforløb for interruptstyret in- og output, er hentet fra en typisk minidatamat.

Programafbrydelsesfunktionen udføres delvis af materiellet (automatisk), og delvis af programmet.

Da interruptenheden er forskellig fra det ene datamaskinefabrikat til det andet, vil man se, at maskintyperne adskiller sig fra hinanden ved, at materiellet på nogle maskiner automatisk udfører en større eller mindre del af de funktioner, der på andre maskiner udføres af programmet.

Som eksempel på en programafbrydelsesfunktion, der kan varetages af enten programmet eller af materiellet, kan nævnes: Gemning og retablering af status for det afbrudte program. For ikke at skulle begynde forfra på en programdel, når der returneres fra en interruptrutine, er det ofte nødvendigt at gemme indholdet af datamaterns aktive registre (Akkumulator-operand- og statusregistre). Inden der returneres fra interruptrutinen, kaldes det gemte indhold tilbage i de aktive registre, og der returneres således med det korrekte indhold i de aktive registre; programudførslen af det afbrudte program kan da umiddelbart fortsættes. Af yderligere materielfunktioner, har vi f. eks. de kredsløb, der automatisk kan prioritere de interruptsignaler, der kommer samtidigt fra flere enheder.

#### 4. 4. 3. Interrupt hvordan

For at kunne arbejde med interrupt er det nødvendigt, at udbygge interfaceenhederne en smule.

Kontrolbussen udbygges med yderligere 2 signaler:

Eet output signal TEST INTERRUPT (TSTINT), som er en kort impuls, der udsendes til alle interfacekort efter udførelsen af enhver maskininstruktion.

Eet in-put signal INTERRUPT ( $\overline{\text{INT}}$ ), som "returnerer" TSTINT på inverteret form til styreenheden, hvis en Ready F-F er sat.

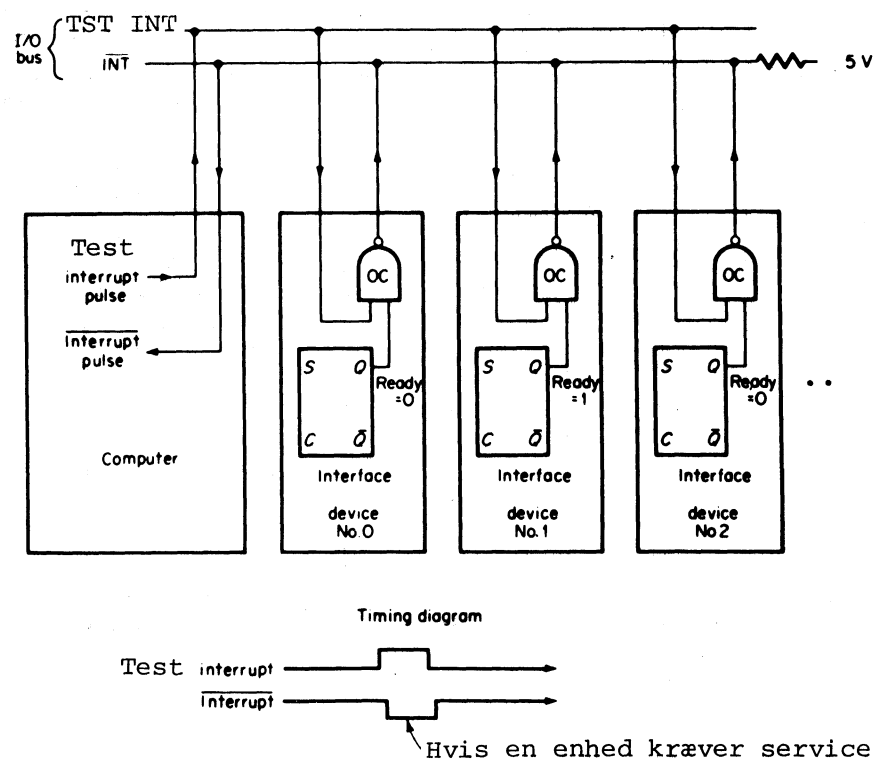


Fig. 4.4.31

På fig. 4.4.3.1. ses, hvorledes enhed nr. 1 returnerer testimpulsen via en åben collector NAND gate, fordi dens Ready Flag er sat.

Når  $\overline{\text{INT}}$  går lav, ved styreenheden, at der ønskes interrupt, men ikke hvilken enhed der er tale om.

En simpel løsning på dette problem er vist i fig. 4.4.3.2.

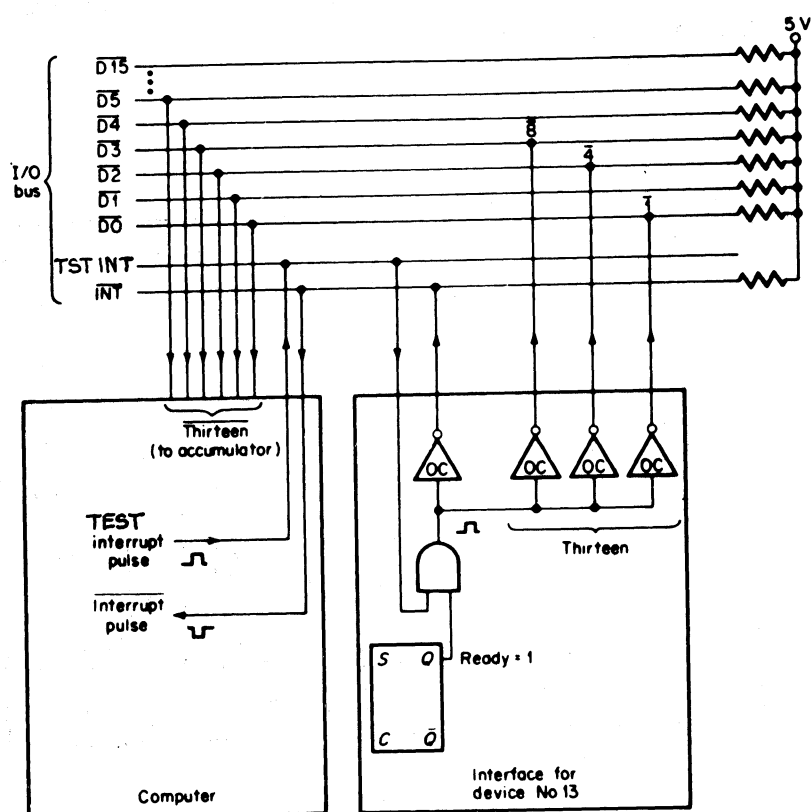


Fig. 4.4.3.2

Her er vist, hvorledes enhed nr. 13, når den genererer interrupt, sender sit eget nr. inverteret ud på databussen via et antal åben collektor invertere.

Dette nr. går via databussen ind til styreenheden og angiver startadressen for den subrutine, som ønskes udført. Hver enhed har sin egen servicesubrutine.

Problem: Hvad sker der, hvis både enhed nr. 13 og nr. 14 sætter Ready Flag indenfor samme maskincyklus?

Ja, så starter subrutinen for enhed nr. 15.

En løsning på dette problem er vist i fig. 4.4.3.3.

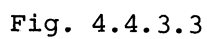
Her er systemet udvidet med en prioriteringslinie og 3 gates. Udgangen fra enhed 12 hedder Enable 13, udgangen fra enhed 13 hedder Enable 14 o.s.v. Indgangen på første enhed (enhed nr. 0) er et ettal.

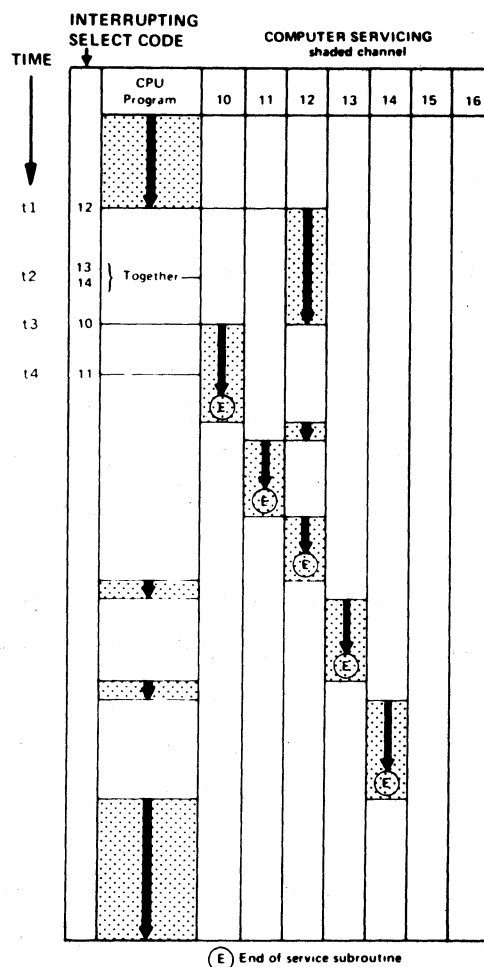
Hvis enhed 13 og 14 sætter Ready Flag samtidig, bliver Enable 14 = 0 og kun enhed 13 giver interrupt og sender sit nr. ud på databussen.

Et sådant system kaldes et prioriteringssystem. Prioriteringen er her bestemt af kortenes placering, og enheden længst til venstre har højest prioritet, og kan altid give interrupt.

Ved manglende kort kortsluttes ind- og udgang, således at prioriteringslinien altid føres videre til de resterende kort.







(Hewlett Packard)

Ovenstående tegning anskueliggør tidsforløbet når ydre enheder med forskellig prioritet giver interrupt.

De ydre enheder med Select Code 10, 11, 12, 13 og 14 er alle aktiveret på et tidligere tidspunkt, og datamaten er ved at udføre et program (CPU program), til tiden  $t_1$  giver den ydre enhed med Select Code 12 interrupt; herved sker der generelt det, at datamaten udfører en instruktion, der er placeret et fast sted i lageret; denne instruktion er i HP-datamaten placeret i den lagerplads, der har samme adresse som den ydre enheds Select Code, her altså lagerplads nr. 12. Denne lagerplads kaldes en Trap-celle og indeholder normalt en Jump to Subroutine (JSB) SERVICE instruktion.

Service NOP

.  
. .  
. .  
. .

JMP Service, I.

En subrutine starter med instruktionen No Operation (NOP) - i denne lagerplads gemmes nu returadressen, det vil i dette tilfælde sige adressen i hovedprogrammet, hvor interruptet fandt sted.

Subrutinen afsluttes med et indirekte hop (JMP SERVICE, I), og næste instruktion, der vil blive udført, er den instruktion der står i returadressen. Med andre ord: Når en subrutine er afsluttet springes tilbage til det sted, hvor man kom fra.

Fortsættes nu på figuren, kommer der til tiden  $t_2$  interrupt fra enhederne 13 og 14, disse har lavere prioritet end enhed 12, og datamaten fortsætter uanfægtet med enhed 12's servicerutine.

Til tiden  $t_3$  giver enhed 10 interrupt, denne enhed har højestprioritet, og datamaten springer nu over og udfører enhed 10's servicerutine selv om enhed 12 ikke er færdigbetjent. Til tiden  $t_4$  giver enhed 11 interrupt, men datamaten fortsætter med enhed 10's servicerutine indtil denne er færdig (markeret med et E).

Når enhed 10 er færdigbetjent springes tilbage til det sted, hvor enhed 10 gav interrupt; her til det sted datamaten nåede i enhed 12's servicerutine; men da enhed 11 i mellemtiden har givet interrupt, og har højere prioritet end enhed 12, vil datamaten straks springe over og betjene enhed 11. Når denne, der nu har højestprioritet, er færdigbetjent springes igen tilbage til enhed 12's servicerutine, der nu færdiggøres.

Enhed 12 var den første enhed, der gav interrupt, og datamaten sprang da fra et sted i hovedprogrammet. Når enhed 12 således er færdigbetjent, returneres til hovedprogrammet; men da enhed 13 også har givet interrupt og nu har højestprioritet, vil denne enhed blive betjent. Når enhed 13 er færdigbetjent, springes sluttelig via hovedprogrammet til enhed 14's servicerutine, først når denne enhed er færdigbetjent fortsættes med hovedprogrammet.

Afviklingsforløbet for interruptet er ofte forskelligt fra den ene datamaskine til den anden. Til forskel for det netop beskrevne hændelsesforløb, kan det nævnes, at nogle datamaskiner altid gemmer returadressen i en fast lagerlokation (f.eks. adresse 0). Man kan også finde datamaskiner, der kræver, at alle interruptrutiner starter på samme adresse. (Man må så programmæssigt finde ud af, hvilken ydre enhed, der anmoder om interrupt). Andre datamaskiner kan være konstrueret på en sådan måde, at interruptrutinens startadresse automatisk indlæses fra den enhed, der anmoder om interruptet.

#### 4. 5. Direct Memory Access (DMA)

Hurtige ydre enheder, som f.eks. pladelagre, kan ikke kommunikere effektivt med datamaskinen v.h.a. interrupt, da en interruptservicerutine ofte vil være af størrelsesordenen 50 - 200  $\mu$ sec.

Input/Output af en datablok via DMA kan foregå, som følger. (Se fig. 4.5.1):

1. Overfør den ønskede startadresse, hvor datablokken skal anbringes i lageret, til reg. 1.
2. Overfør bloklængden til reg. 2.
3. Start den ydre enhed og fortsæt programafviklingen.

4. Pladelageret melder, at 1. ord er læst/skrevet.
5. DMA enheden standser CPU'en.
6. Reg. 1 adresserer arbejdslageret.
7. Ordet overføres via DMA kanaler til/fra arbejds-  
lageret.
8. CPU'en får lov til at fortsætte.
9. Reg. 1 tælles en op.
10. Reg. 2 tælles en ned.  
Punkt 4-10 gentages indtil reg. 2 = 0.
11. Reg. 2 = 0 medfører, at DMA kanalen giver inter-  
rupt med høj prioritet.
12. Den tilhørende servicerutine standser DMA kanalen.

Metoden kaldes også cyklustyveri, idet DMA kanalen nu og da stjæler en lagercyklus fra CPU enheden.

Eksempel:

En datamaskine har en cyklustid på 2  $\mu$ sec. og en ord-  
længde på 16 bit. Der er tilsluttet et pladelager, som  
kan transmittere 8 bit ord (Bytes) med en hastighed af  
200.000 Bytes/sec.

Der foretages et cyklustyveri af 2  $\mu$ sec. varighed,  
hver gang der er læst 2 bytes d.v.s.  $(2 \times \frac{1}{200.000} =$   
 $10^{-5})$  hvert 10.  $\mu$ sec.

Datamaskinens effektive arbejdshastighed nedsættes  
derfor med  $(2/10) = 20\%$ , medens datatransporten fore-  
går.

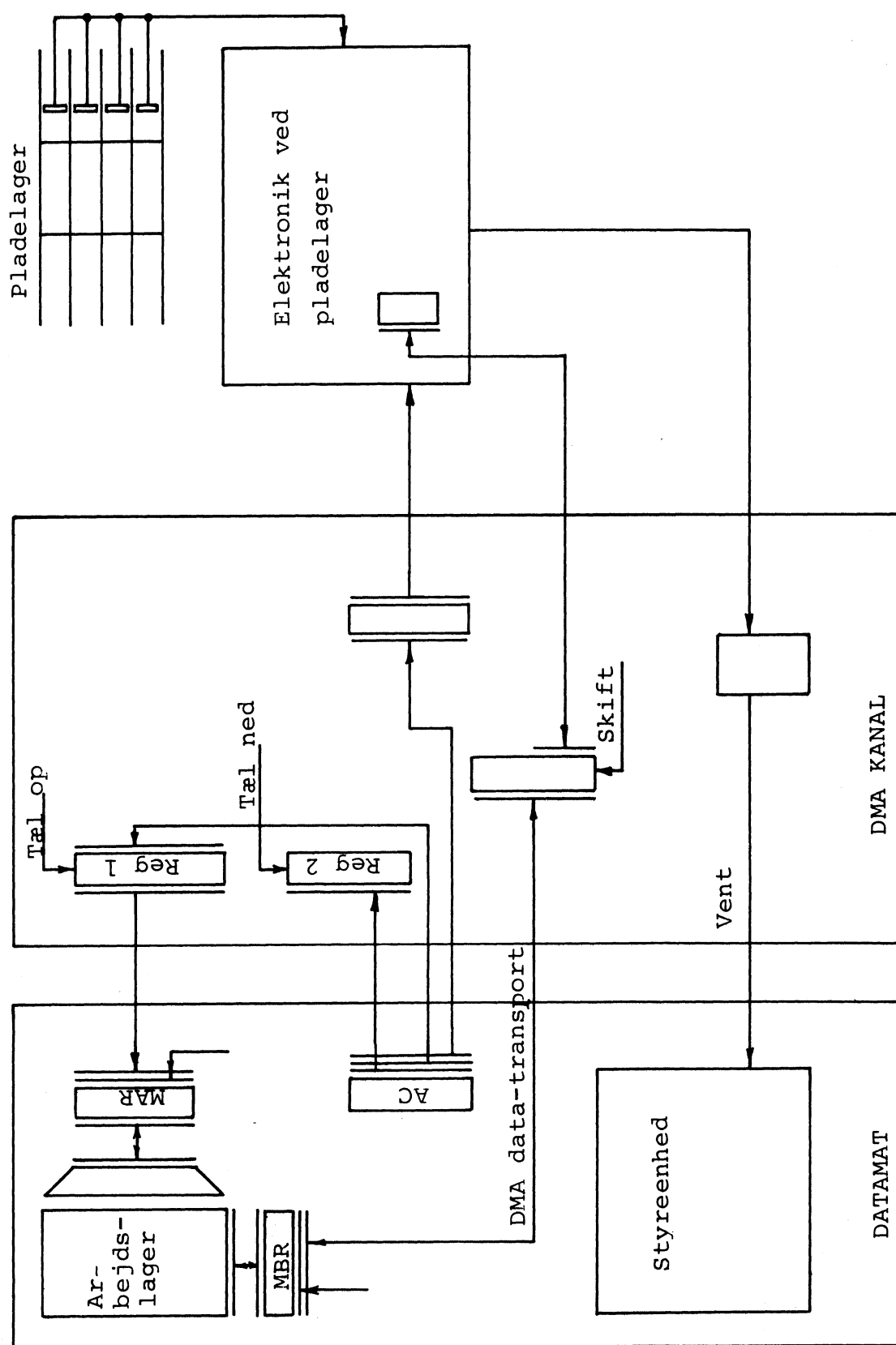


Fig 4.5.1.1. Cyklustyveri ved direkte lagertilgang.

## DATAMASKINER - ME

### Opgave 4.2

#### Input/output programmering.

Lav et program der v.h.a. vente på flag-metoden, kan indlæse en karakter fra en TTY, gemme den i lageret og udlæse den igen til terminalen (ekko).

TTY-enheden har I/O-adresse nr.  $15_8$ .

Da enhed nr.  $15_8$  (TTY-en) er både input og output enhed, må interfacekortet have besked om hvilken funktion der benyttes.

Hvis man sender et kontrolord ud til enhed  $15_8$ , som har MSB=1 opfattes det af interfacen som et kontrolord, som fortolkes således:

$140000_8$ .....tastatur = input  
 $130000_8$ .....skriver = output

Programmet placeres startende i adresse:

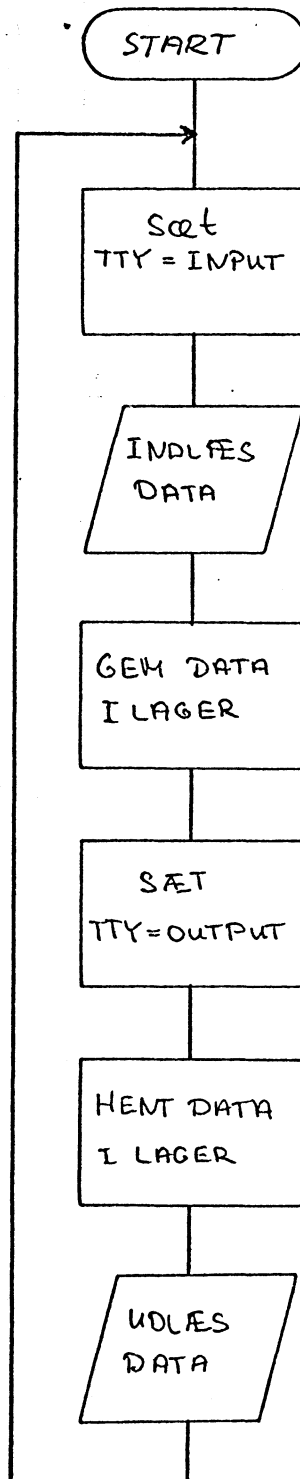
Hold 1	adresse	$100_8$
Hold 2	"	$200_8$
Hold 3	"	$300_8$
Hold 4	"	$400_8$

# DIAGRAMMERING

Udfyldt af <b>L.B</b>	Udfyldt den <b>15 08 77</b>	Opgavenr <b>4.2</b>	Bilag nr <b>ME.</b>	Sidenr
Processens navn / Nr. <b>I/O PROGRAM FOR TTY</b>				

1 2 3 4 5

A  
B  
C  
D  
E  
F  
G  
H  
I







[illegible]

AUTOMATISERINGSTEKNIK



TEKNOLOGISK INSTITUT KØBENHAVN

## 5. Ydre enheder

### 5.1 Relevante begreber

En datamaskine har til formål at behandle information. Dette gøres ved hjælp af datamaskinens lager- regne- og styreenhed.

Datamaskinesystemet må også være i stand til at udveksle data mellem sig selv og dens omgivelser. Informationerne, der skal behandles, må indlæses i datamaskinen, og resultaterne må sendes tilbage til den ydre verden. Ind- og udlæsningen sker ved hjælp af datamaskinens ind/udlæseenhed.

Datamaskinens kan kommunikere direkte med en bruger ved hjælp af kommunikationsenheder. Kommunikationen kan også ske mellem datamaskine og ydre enheder, der er i stand til at lagre store datamængder. Enheder der benyttes til at lagre data til senere behandling på den samme eller en anden datamaskine kaldes baggrundslagre.

Endelig kan kommunikation ske ved at datamaskinens indlæseenhed er forbundet direkte til mekanisk eller elektrisk udstyr, der afgiver information om den øjeblikkelige tilstand af en eller anden proces.

Datamaskinen bearbejder de målte procesdata og kommunikerer ved hjælp af udlæseenheden styre- og regulerings signaler tilbage til processen.

Man vil ofte se at behandling og kommunikation af procesdata foregår som tidstro kørsel, d.v.s. at operationerne i datamaskinen finder sted så hurtigt i forhold til forløbene i det fysiske system, at resultaterne af operationerne kan anvendes til styring af det fysiske system.

Man kan således groft opdele datamaskinens ydre enheder i to kategorier:

1. Baggrundslagre og
2. Kommunikationsenheder.

Af baggrundslagre kan nævnes:

Pladelagre  
Tromlelagre  
Magnetbåndstationer  
Kassettebåndstationer  
Floppy Disc's

Under kommunikationsenheder hører:

Terminaler  
Strimmellæsere  
Hulkortlæsere  
Strimmelhullere  
Hulkorthullere

De her nævnte enheder indgår i kommunikationen menneske - maskine.

Til kommunikation af procesdata kan nævnes:

Analog til digitalomsætter A/D  
Digital til Analogomsætter D/A  
Digitale indgange, kontakter, relær,  
termostater, alarmer m.m.  
Digitale udgange, on/off styring af  
ventiler, motorer m.m.

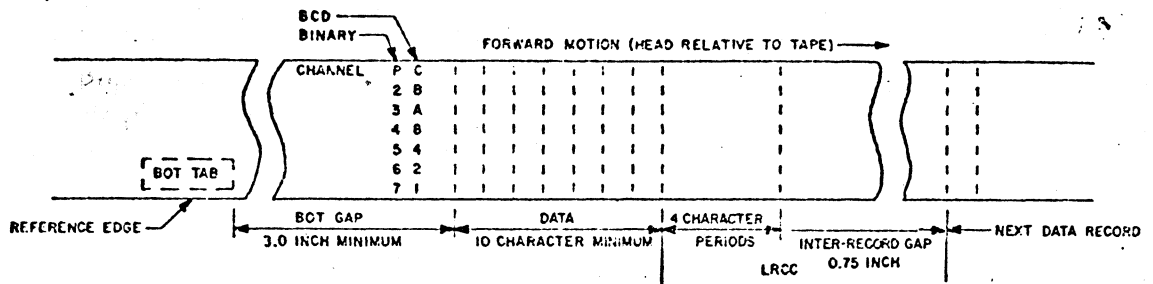
En yderligere gruppering af de ydre enheder kan foretages ved at underopdele kommunikations- og lagermedierne:

## Mechanical and Electrical Specifications, Model 8X40

Tape (Computer Grade)	
Width	0.5 inches
Thickness	1.5 mil
Tape Tension	8.0 ounces
Reel Diameter	10.5 inches maximum
Recording Mode (ANSI and IBM compatible)	NRZI
Magnetic Head	Dual Stack with Erase Head
Tape Speed, Standard	45, 37.5, 25, 18.75, 12.5 ips
Instantaneous Speed Variation	±3 percent maximum
Long-Term Speed Variation	
Forward	±1 percent maximum
Reverse	±3 percent maximum
Rewind Speed	200 ips nominal
Interchannel Displacement Error	Refer to Paragraph 6.6.11.1
Stop/Start Time at 45 ips (inversely proportional to tape speed)	8 ±0.55 msec
Stop/Start Displacement	0.19 ±0.02 inches
Beginning of Tape (BOT) and End of Tape (EOT) Detectors	Photoelectric* IBM Compatible
Weight	85 pounds
Dimensions	
Height	24.5 inches**
Width	19.0 inches
Depth (from mounting surface)	
with Multiple Transport Adapter	15.0 inches maximum
without Multiple Transport Adapter	12.5 inches maximum
Depth (total) (without MTA)	16.0 inches
Operating Temperature	2°C (35°F) to 50°C (122°F)
Non-Operating Temperature	-45°C (-50°F) to 71°C (160°F)
Operating Altitude	0 to 20,000 feet
Non-Operating Altitude	0 to 50,000 feet
Power	
Volts ac	95, 100, 110, 115, 125, 190, 200, 210, 215, 220, 225, 230, 235, 240, 250
Watts (maximum on high line)	300
Hertz	48 to 400
Mounting	19 inches, consistent with EIA requirements
Electronics	All silicon
* Approximate distance from detection area to head gap equals 1.2 inches.	
** Includes one-half-inch filler panel.	

Specifikationer på magnetbåndstation for minidatamat.

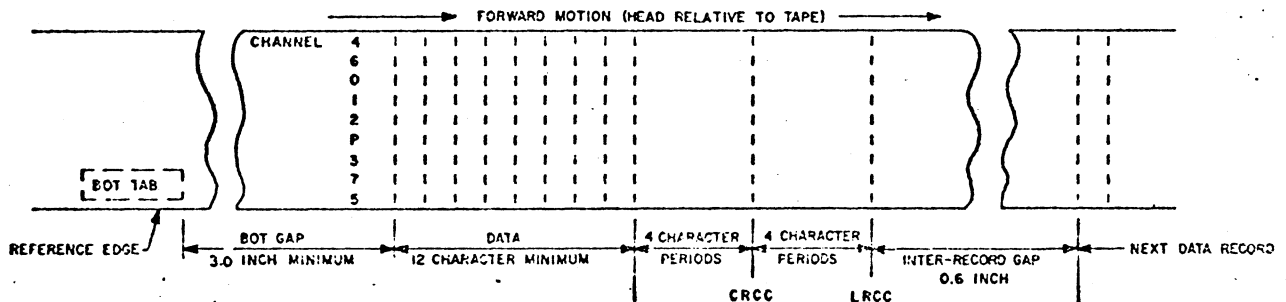
## Eksempler på standardiserede båndformater.



### NOTES

1. TAPE SHOWN WITH OXIDE SIDE UP.
2. CHANNELS 2 THROUGH 7 CONTAIN DATA BITS IN DESCENDING ORDER OF SIGNIFICANCE.
3. CHANNEL 1 (PARITY) CONTAINS ODD DATA PARITY FOR BINARY TAPES, OR EVEN PARITY FOR BCD TAPES.
4. EACH BIT OF THE LRCC IS SUCH THAT THE TOTAL NUMBER OF "1" BITS IN THAT TRACK (INCLUDING THE LRCC, IS EVEN. IT IS POSSIBLE IN THE 7-TRACK FORMAT FOR THIS CHARACTER TO BE ALL ZEROS, IN WHICH CASE A READ DATA STROBE WILL NOT BE GENERATED.
5. A FILE MARK IS A SINGLE CHARACTER RECORD HAVING "1" BITS IN CHANNELS 4, 5, 6 AND 7 FOR BOTH THE DATA CHARACTER AND THE LRCC. THIS RECORD IS SEPARATED BY 3.5 INCHES FROM THE PREVIOUS RECORD AND BY A NORMAL IRG (0.75 INCH) FROM THE FOLLOWING RECORD.
6. DATA PACKING DENSITY MAY BE 200, 556, OR 800 BITS PER INCH.

## 7-Track Format

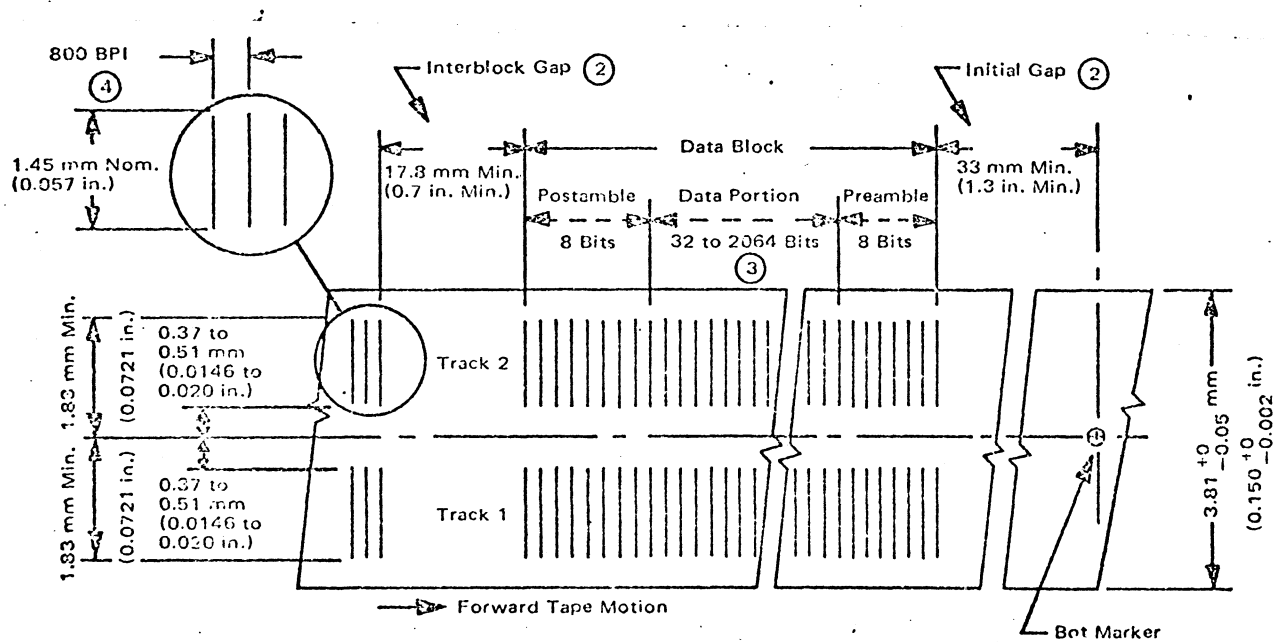
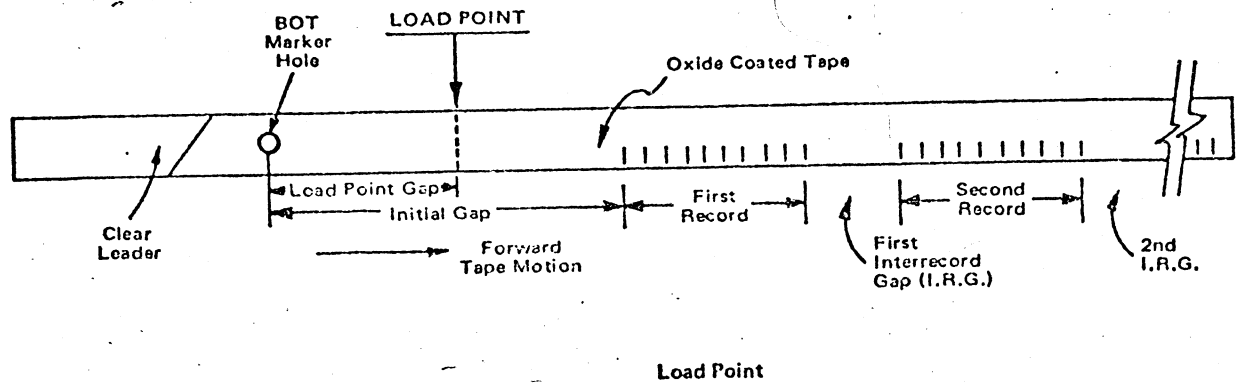


### NOTES

1. TAPE SHOWN WITH OXIDE SIDE UP.
2. CHANNELS 0 THROUGH 7 CONTAIN DATA BITS IN DESCENDING ORDER OF SIGNIFICANCE.
3. CHANNEL 0 (PARITY) ALWAYS CONTAINS ODD DATA PARITY.
4. EACH BIT OF THE LRCC IS SUCH THAT THE TOTAL NUMBER OF "1" BITS IN THAT TRACK (INCLUDING THE CRCC AND THE LRCC, IS EVEN. IN THE 7-TRACK FORMAT THE LRCC WILL NEVER BE AN ALL-ZEROS CHARACTER.
5. IT IS POSSIBLE FOR THIS CRCC CHARACTER TO BE ALL ZEROS, IN WHICH CASE A READ DATA STROBE WILL NOT BE GENERATED.
6. A FILE MARK IS A SINGLE CHARACTER RECORD HAVING "1" BITS IN CHANNELS 3, 6, AND 7 FOR BOTH THE DATA CHARACTER AND THE LRCC. THE CRCC CONTAINS ALL ZEROS. THIS RECORD IS SEPARATED BY 3.5 INCHES FROM THE PREVIOUS RECORD AND BY A NORMAL IRG (0.6 INCH) FROM THE FOLLOWING RECORD.
7. DATA PACKING DENSITY IS FIXED AT 800 BITS PER INCH.

## 9-Track Format

# Eksempler på båndformat for en kassettebåndstation.

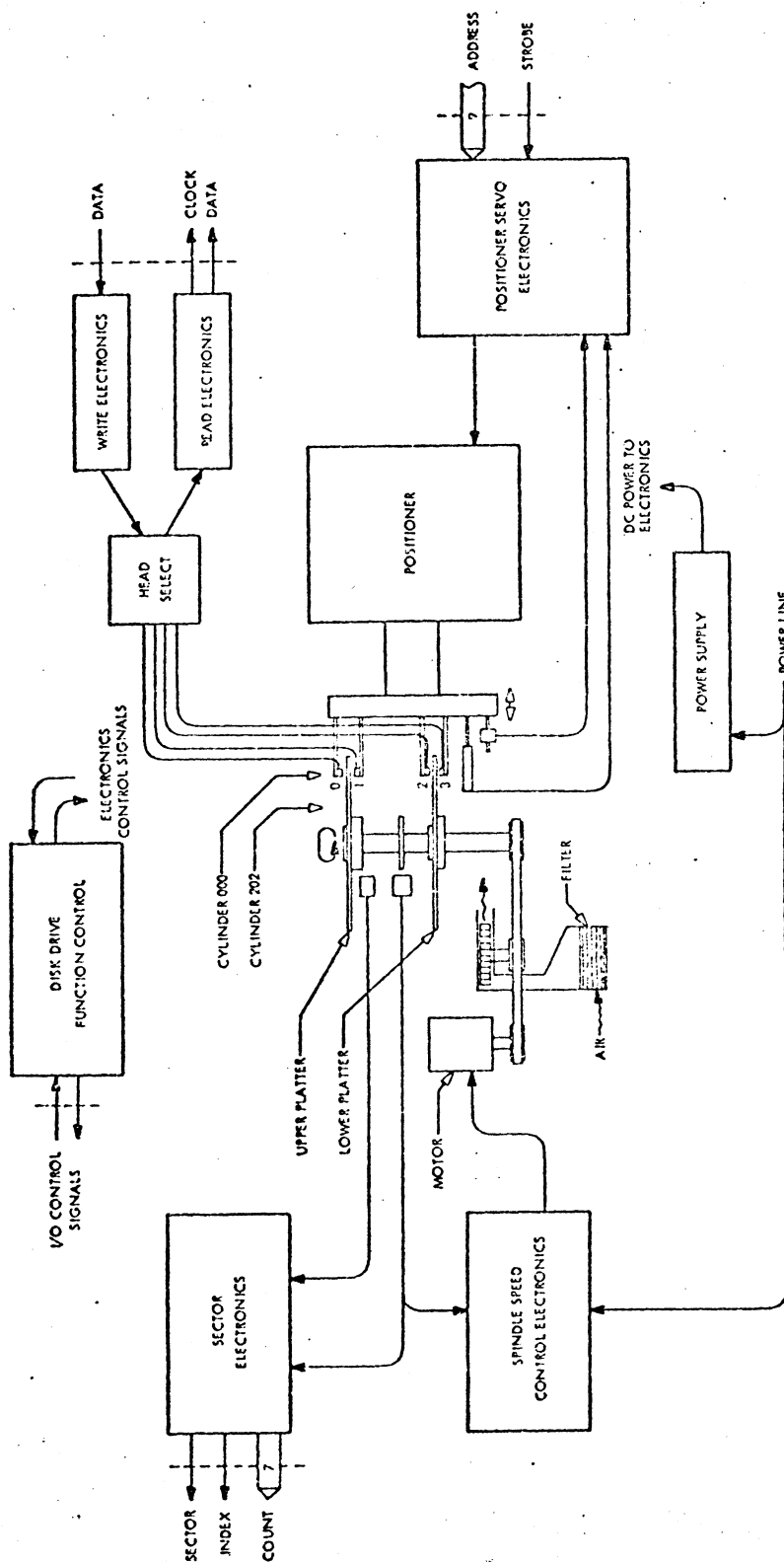


## NOTES:

- 1 Tape is shown with oxide side out.
- 2 Tape is fully saturated in the erase direction in the interblock gap and the initial gap.

- 3 The last 2 characters (16 bits) of the data portion is the Cyclic Redundancy Check (CRC).
- 4 Shown without phase flux reversals that may exist between data bits.

Recording Format 800 BPI



Functional Block Diagram

Funktions blokdiagram af Disk System med 2 plader og bevægelige hoveder.

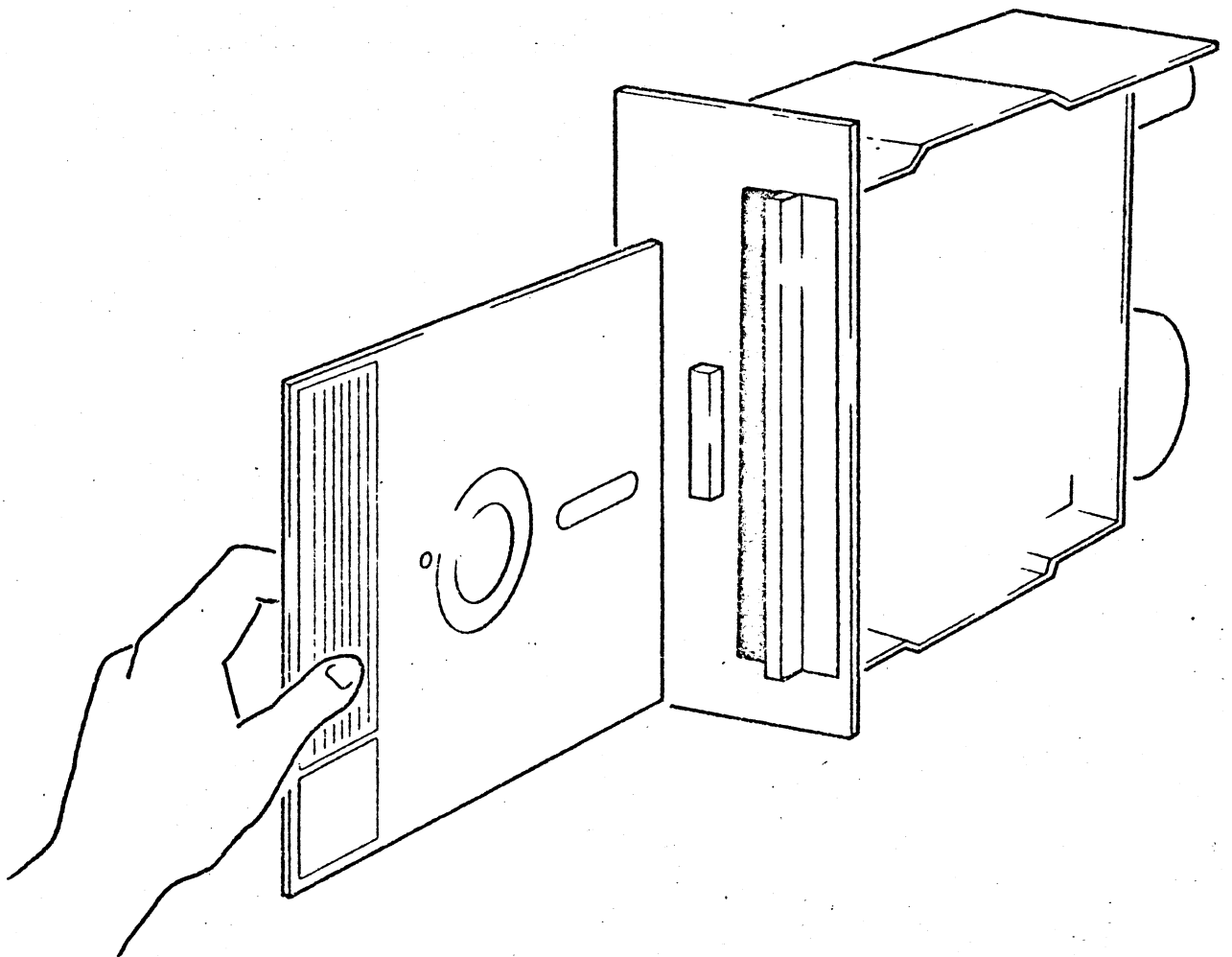


# Disk system med to plader og bevægelige læse/skrivehoveder.

## Mechanical and Electrical Specifications

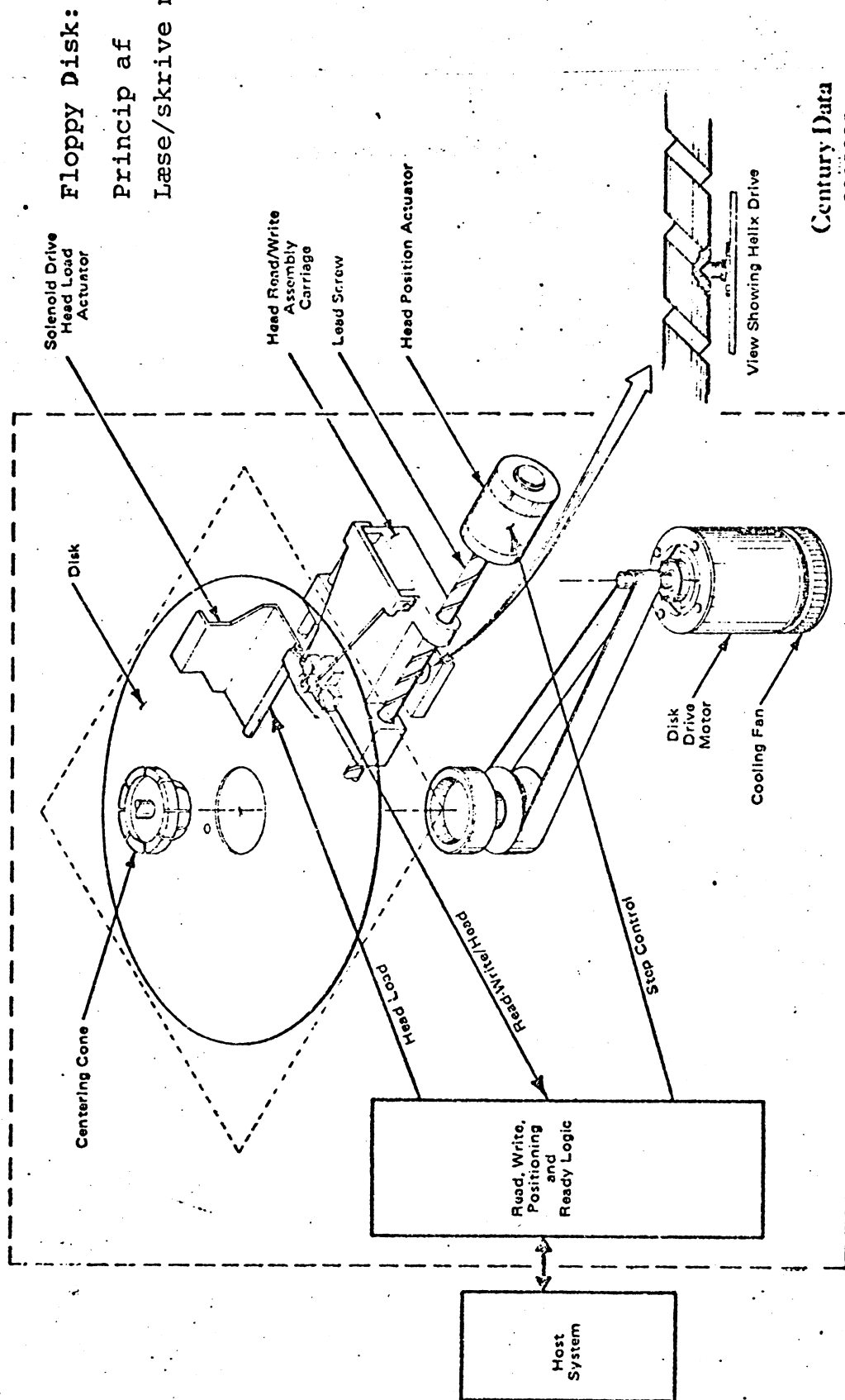
	100 tpi Models	200 tpi Models
<b>Storage Capacity (Unformatted)</b>		
Single Disk Models at 1025 bpi 1100 bpi 2200 bpi	11.6028 megabits 12.68344 megabits 25.3750 megabits	50.75 megabits
Dual Disk Models at 1025 bpi 1100 bpi 2200 bpi	23.3556 megabits 25.36688 megabits 50.750 megabits	101.5 megabits
<b>Cylinders/Tracks, Single Disk Models</b>	203 cylinders/406 tracks	406 cylinders/812 tracks
<b>Dual Disk Models</b>	203 cylinders/812 tracks	406 cylinders/1624 tracks
<b>Sectors</b>	6, 8, 10, 12, 14, 16, 18, 20, 24, 28, 30, 32, 36, 40, 42, 48, 56, 60, 64, 70, 72, 80, 84, 90, 96, 112, 120, 126, 128	6, 8, 10, 12, 14, 16, 18, 20, 24, 28, 30, 32, 36, 40, 42, 48, 56, 60, 64, 70, 72, 80, 84, 90, 96, 112, 120, 126, 128
<b>Bits per Inch/Tracks per Inch</b>	1025 bpi/100 tpi, 1100 bpi/100 tpi, or 2200 bpi/100 tpi	2200 bpi/200 tpi
<b>Data Transfer Rate</b>		
1025 bpi, 1500 rpm	0.72000 megabits per second	
1025 bpi, 2400 rpm	1.17200 megabits per second	
1100 bpi, 1500 rpm	0.78100 megabits per second	
1100 bpi, 2400 rpm	1.24560 megabits per second	
2200 bpi, 1500 rpm	1.56250 megabits per second	1.56250 megabits per second
2200 bpi, 2400 rpm	2.50000 megabits per second	2.50000 megabits per second
<b>Disk Speed</b>	1500 or 2400 rpm (±1%)	1500 or 2400 rpm (±1%)
<b>Latency Time (Average)</b>		
1500 rpm Models	20 milliseconds (±1%)	20 milliseconds (±1%)
2400 rpm Models	12.5 milliseconds (±1%)	12.5 milliseconds (±1%)
<b>Head Positioner</b>	Voice Coil Linear Motor with Optical Detent	Voice Coil Linear Motor with Optical Detent
<b>Seek Time</b>		
Adjacent Track	9 milliseconds maximum	10 milliseconds maximum
Average (One-third Stroke)	35 milliseconds maximum	40 milliseconds maximum
Maximum (Full Stroke)	60 milliseconds maximum	65 milliseconds maximum
<b>Start Time</b>	57 seconds maximum	57 seconds maximum
<b>Stop Time</b>	22 seconds maximum	22 seconds maximum
<b>Removable Media Type</b>	IBM 5440 or 2315 type Cartridge	IBM 5440 or 2315 type cartridge
<b>Read/Write Heads</b>		
Type	Ramp Loaded, Radially Aligned	Ramp Loaded, Radially Aligned
Number	2 or 4 (One Per Disk Surface)	2 or 4 (One Per Disk Surface)
Recording Mode	Double Frequency	Double Frequency
<b>Dimensions</b>		
Height	8-3/4 inches maximum	8-3/4 inches maximum
Width	19 inches	19 inches
Depth from Mounting Surface	26 inches	26 inches
Front Projection from Mtg Surface	3-1/4 inches	3-1/4 inches
Total Depth	29-1/4 inches	29-1/4 inches
<b>Mounting</b>	Standard 19-inch EIA	Standard 19-inch EIA
<b>Weight (Excluding Slides and Cartridge)</b>		
Top Loading Models	116 pounds (Including Integral Power Supply)	116 pounds (Including Integral Power Supply)
Front Loading Models	112 pounds (Including Integral Power Supply)	112 pounds (Including Integral Power Supply)
<b>Operating Temperature</b>	10°C (50°F) to 40°C (104°F)	15.6°C (60°F) to 33.0°C (100°F)
<b>Non-Operating Temperature</b>	-10°C (14°F) to 65°C (145°F)	-10°C (14°F) to 65°C (145°F)
<b>Operating Humidity</b>	5% to 85% Non-condensing	5% to 85% Non-condensing
<b>Storage Humidity</b>	to 95% Non-Condensing at 40°C to 80% Non-Condensing at 65°C	to 95% Non-Condensing at 40°C to 80% Non-Condensing at 65°C
<b>Operating Altitude</b>	0 to 7500 feet	0 to 7500 feet
<b>Non-Operating Altitude</b>	0 to 20,000 feet	0 to 20,000 feet
<b>Power</b>		
Volts ac	95, 100, 110, 115, 125, 190, 200, 210, 215, 220, 225, 230, 235, 240, 250	95, 100, 110, 115, 125, 190, 200, 210, 215, 220, 225, 230, 235, 240, 250
Watts (Maximum on High Line) (Typical)	1100 Peak (Start/Stop Cycles Only) 400	1100 Peak (Start/Stop Cycles Only) 400
Hz	48 to 52, and 58 to 62	48 to 52, and 58 to 62
<b>Electronics</b>	All Silicon	All Silicon
<b>Underwriters Laboratory</b>	Designed to Qualify for UL and CSA Approval	Designed to Qualify for UL and CSA Approval

## Floppy Disk System



Loading SA900/901

# THE CDS 140



Floppy Disk:  
Princip af  
Lase/skrive mekanik.

Century Data  
00000000

The CDS 140

Floppy Disk: Specifications.

**SHUGART ASSOCIATES**  
**SA900/901**  
**Diskette Storage Drive**

**SPECIFICATION SUMMARY**

**Performance Specifications**

Capacity (Unformatted)	Per Disk Per Track	3.1 megabits 41 kilobits
Data Transfer Rate		250 kilobits/second
Access Time	Track to Track Settling Time	10 MS 10 MS
Average Access Time		260 MS
Rotational Speed		360 RPM
Average Latency		83 MS
Recording Mode		Frequency Modulation

**Media Characteristics**

Cartridge Required	SA900 SA901	SA100 or IBM "Diskette SA101
Physical Sectors	SA900 SA901	0 32
Index		1
Tracks		77
Density	Recording Track	3200 bpi (approx. inside track) 48 TPI

**Additional Features for SA900/901**

50 Hz - 100 VAC, single phase  
60 Hz - 208/230 VAC, single phase  
50 Hz - 208/230 VAC, single phase  
Write Protect (SA901 only)  
-12 or -15V option to replace -5V input  
Chassis Slide  
10½" High Front Plate for use with  
Chassis Slide  
5.25" x 11" Front Plate  
5.25" x 10" Front Plate

**READ ERROR RATE**

1 x 10<sup>9</sup> bits read/soft error (nominal)  
1 x 10<sup>12</sup> bits read/hard error (nominal)

**SEEK ERROR RATE**

1 seek error in 10<sup>6</sup> seeks

### Kommunikationsmedier:

<u>Tidstro ind/uddata</u>	<u>Databærere</u>
Elektriske og mekaniske signaler lys, lyd	papir film magnetbånd

Bemærk at databærere er medier, der benyttes til at bære registrerede data, samtidig med at de er transportable uafhængig af datamaskine og ydre enhed.

### Lagermedier:

<u>Flygtige</u>	<u>Ikke-flygtige</u>	<u>Permanente</u>
Halvleder-komponenter	Magnetiske materialer	Magnetiske og optiske materialer, visse halvledermaterialer

Flygtige lagre defineres som lagre, hvis indhold går tabt ved strømafbrydelse.

Permanente lagre defineres som lagre, hvis indhold ikke er sletbart, men hvori de lagrede data kun kan ændres ved udskiftning af lagringsmediet.

Eksempler på de til datamedierne hørende ydre enheder kan opstilles således:

<u>Datamedier</u>	<u>Enheder</u>
Tidstro I/U-data	Dataskærme, transducere, modemer, lamper, A/D og D/A-konvertere
Databærere	Hullere, læsere, skrivemaskiner og båndstationer.
Flygtige lagre	Registre Halvleder RAM-lagre

Ikke-flygtige lagre	Kernelagre, tromlelagre, pladelagre, båndlagre og magnetkortlagre
Permanente lagre	Læselagre

Data, der lagres/bæres af de forskellige medier, kan være tegn, bogstaver, tal, måleværdier o.s.v. Disse må repræsenteres på medierne på en sådan form, at det er muligt at tolke informationen enten direkte eller indirekte (f.eks. ved hjælp af datamaskinen selv).

En sådan tolkning kan kun finde sted, hvis lagringen foregår ved at benytte koder; d.v.s. at datarepræsentationen foregår efter entydige regler.

Ved kommunikation og lagring af data benytter man sig ofte af standardisere koder, f.eks. en ren binær eller BCD-kode. Andre standardiserede koder er f.eks. Hollerith-koden for datarepræsentation på hulkort og A S C I I - koden der er en ofte benyttet kode indenfor elektronisk databehandling og datatransmission.

I forbindelse med ydre enheder, specielt baggrundslagre er der mange faktorer, der bestemmer hvilke ydre enheder, der skal vælges til et bestemt datamaskinesystem. Af disse faktorer kan bl.a. nævnes:

<u>Kapacitet</u>	(antal bit eller ord)
<u>Ventetid</u>	(tid fra start indtil overføring påbegyndes)
<u>Overføringstid</u>	(tid fra start til slut af overføring)
<u>Tilgangstid</u>	(sum af vente- og overføringstid)
<u>Pålidelighed</u>	(fejlhyppighed pr. overført bit)
<u>Pris</u>	(anskaffelsespris)
<u>Pris</u>	pris pr. bit

## 5.2 Baggrundslagre

Brugernes behov for baggrundslagre opstår, når der skal manipuleres med store datamængder. Datamængder der er så store, at man ikke har mulighed for at lagre dem i datamaskinens arbejdslager. Arbejdslagre, der normalt er kernelagre eller halvleder RAM-lagre, er forholdsvis dyre enheder, hvorfor de ud fra omkostningsmæssige betragtninger (pris pr. bit) er mindre egnede som lagerenheder for store datamængder.

Datamængder, der manipuleres ved hjælp af ydre enheder, opdeles normalt i en række filer.

En fil er en komplet samling af data organiseret til brug for en given opgave. F.eks. kunne en fil ved lagerstyring bestå af den samlede mængde fakturarer for en regnskabsperiode. En fil kan hensigtsmæssigt betragtes som sammensat af et antal poster, hvor hver post indeholder de data, der vedrører en bestemt afgrænset størrelse med relation til opgaven. I lagerstyringsopgaven kunne hver faktura udgøre en post. En post kan yderligere indstilles i felter, hvor hvert felt er den mindste mængde data, der betragtes som en enhed til den pågældende opgaves formål. Hver linie i fakturaen kunne f.eks. udgøre et felt.

Brugeren tildeler sine filer symbolske navne og har ikke egentlig behov for at kende de fysiske placeringer af dataerne. Behovet for en rimelig effektiv drift, og enhedernes kompleksitet og forskellighed medfører dog, at brugeren alligevel må kende de forskellige ydre enheders funktion og styring.

Baggrundslagrenes arbejdsopgave er, at omsætte data til og fra bevægelige, magnetiserbare medier. De generelle problematikker i den forbindelse er lange tilgangstider, lave overføringshastigheder og høje fejlfrekvenser i forhold til datamaskinens interne arbejdslagre. Generelt løses disse problemer ved at behandle data blokvis, at lade baggrundslagrene køre simultant

med centralenhedens operationer og ved at indbygge fejldekkerende og -korrigerende udstyr i enhederne.

En blok er en gruppe af ord der behandles som en samlet enhed. Antallet af ord i en blok (bloklængden) er afhængig af det bestemte udstyr, der skal lagre eller overføre blokken.

### Magnetbåndstationer

Magnetiske baggrundslagre, til hvilke magnetbåndstationer hører, er konstrueret på en sådan måde, at det informationsbærende medium indeholder et tyndt lag af magnetiserbart materiale, som ved læsning eller skrivning passerer et magnethoved.

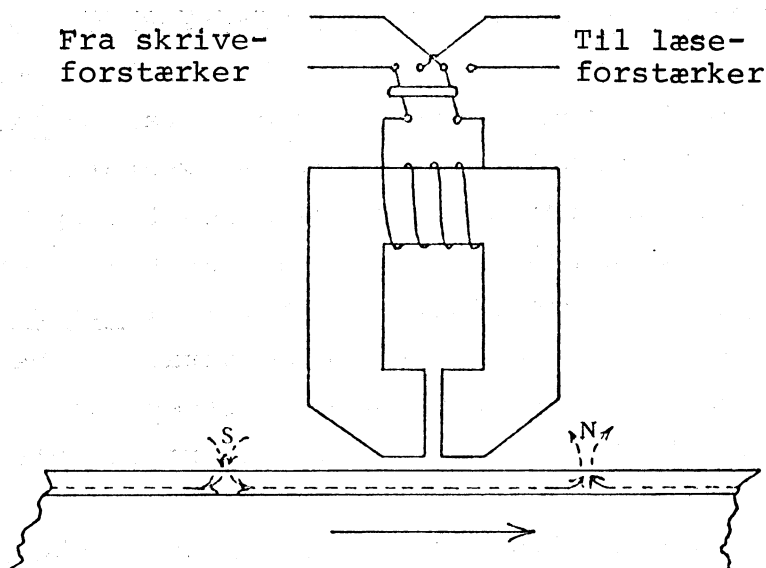


Fig. 5.1

Figur 5.1. viser et magnethoved. Hvis strømretningen i spolen pludselig ændres, opstår der en nordpol eller en sydpol i den magnetiske belægning på båndet. Det er på denne måde man skriver digital information på magnetiske lagre.

Ved læsning kobles spolen til en læseforstærker, og der induceres en spændingsimpuls, hvergang en nord- eller sydpol passerer forbi luftspalten.



Oftentimes, one has two different magnetic heads, one for reading and one for writing.

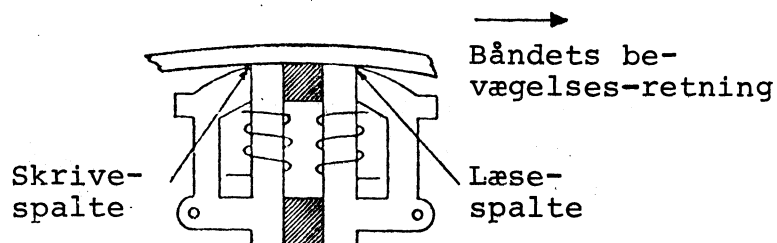


Fig. 5.2

If a read and a write head are placed close together, one can use the read head to check the parity of the written information a few milliseconds after the writing, which is of great value. This method is called "Read on Write".

Magnetic tape is  $\frac{1}{2}$ " wide and contains 7 or 9 tracks. There must therefore be 7-9 magnetic heads on the side of the tape. The bits, which form a transverse row on the tape, together constitute an alphanumeric character.

On cassette tape, one has only one track plus a synchronization track, and one character fills 8 bits after the other.

The gap in the head is ca. 0,01 mm "long" and ca. 1 mm wide. The packing density is from 200 - 1600 bits per inch.

It is very important, that the heads sit on a line perpendicular to the tape. Especially when a tape is played on a second tape station.

Båndhastigheden ligger fra 0,5 m/sek. ved langsomme maskiner, op til ca. 3 m/sek. ved hurtige maskiner.

Hvor mange karakterer pr. sek. kan overføres, når tætheden er 1600 bit/tomme og båndhastigheden er 3 m/sek.?

Informationen på båndet ligger samlet i blokke adskilt af tomme strækninger, blokfab, af ca. 2 cm.'s længde.

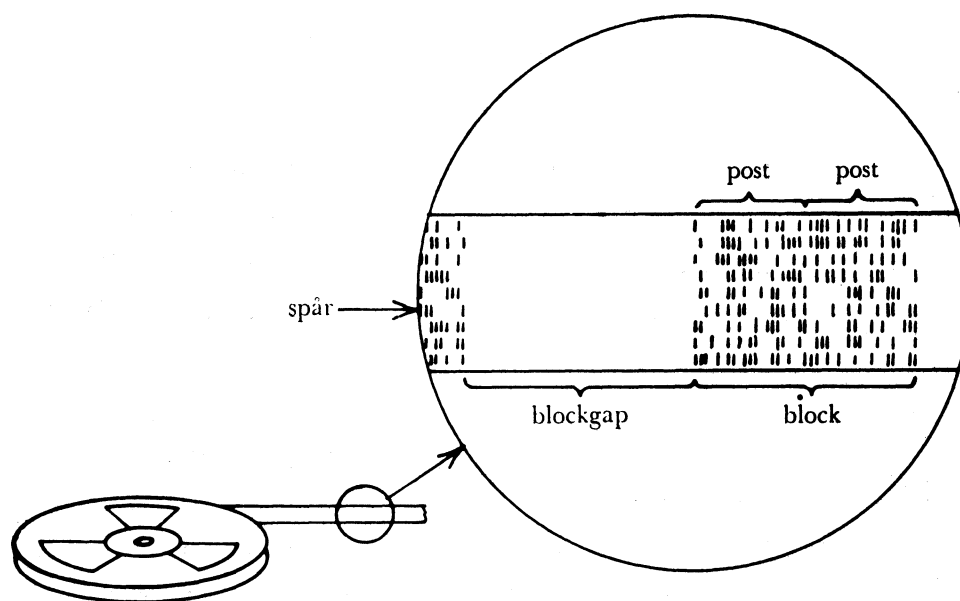


Fig. 5.3

Blokgabene danner accelerations- og bremsestrækningen, som muliggør blokvis læsning.

Tiden for accelerationen eller bremsning til eller fra fuld hastighed ligger i området 5 til 20 msek. Med en båndhastighed på 2 m/sek. og en accelerations-tid på 10 msek. bliver middelaccelerationen  $(2/0,01=)$  200 m/sek.<sup>2</sup> d.v.s. 20 gange tyngdeaccelerationen.

Den tekniske udformning af en båndstation er udviklet ud fra behovet for at "udjævne" de ryk, som opstår i båndet ved accelerationerne. Mellem magnetbåndet og de tunge spoler, som ikke kan accelerere så hurtigt, findes der udligningssløjfer. Sløjferne hænger i vacuumlommer.

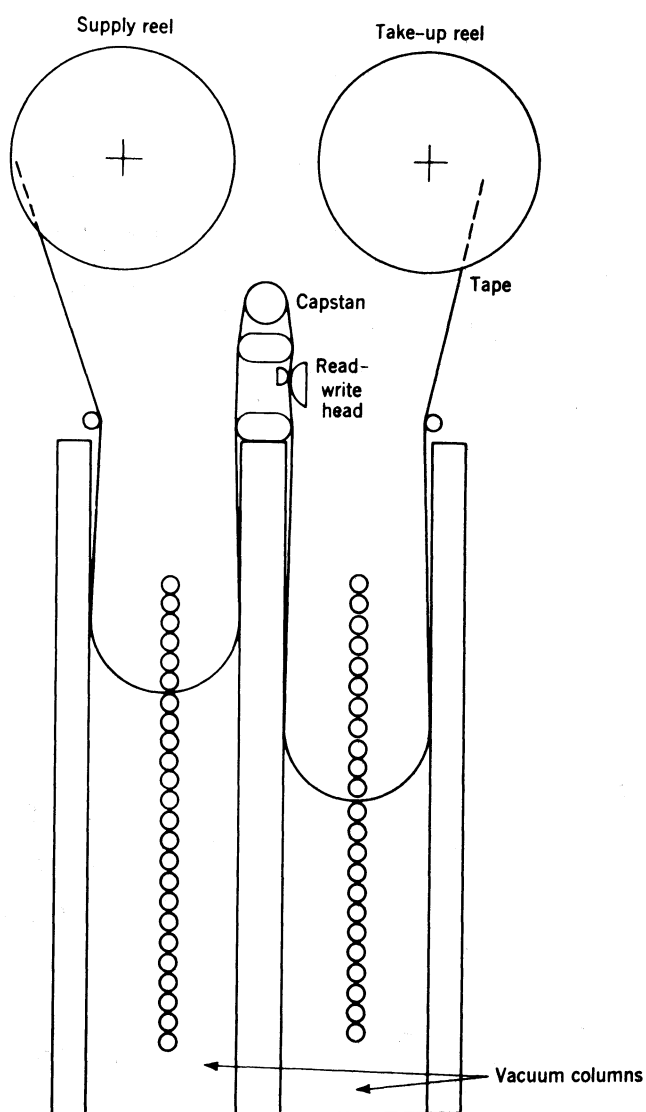


Fig. 5.4

Spolerne drives af store servomotorer, som styres af sløjfernes længde via nogle optiske følere.

Båndet trækkes forbi hovederne af en Kapstan-motor og en modtryksrulle.

I nyere maskiner erstattes alle modtryksruller af et vacuumsystem som holder båndet på plads uden at slide på den magnetiske belægning.

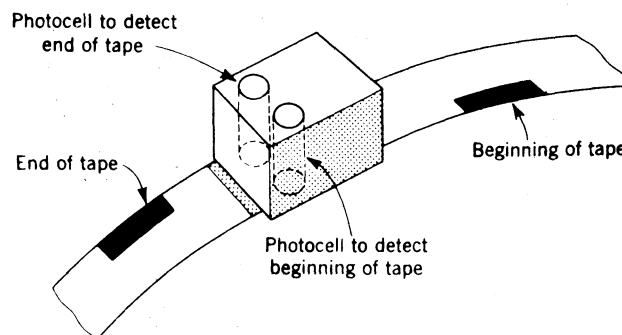


Fig. 5.5

Ved enderne af båndet er anbragt et stykke pålimet aluminiumsfolie, som via lysreflektion og et fotocellearrangement angiver begyndelse og afslutning på båndet.

En magnetbåndstation er en dyr lagerenhed, men prisen pr. bit er meget lav.

På et 200 meter langt bånd til 200,00 kr. med 1000 karakterer pr. tomme er prisen pr. bit ca. 0,0003 øre.

Kapaciteten er ubegrænset (flere spoler).

Overføringshastighed 10-340000 tegn/sek.

Tilgangstid op til flere minutter, hvis man skal læse i den anden ende af båndet.

### Trumlelager

Trumlelageret består af en roterende cylinder med magnetiserbar overflade.

Her kan man ikke lade hovederne berøre overfladen, men må have et luftgab på ca. 0,01 mm.

Hovederne er aerodynamisk udformet således, at de, trods et konstant fjedertryk ind mod tromlen, svæver på den luft som roterer sammen med tromlen. Kun ved langsom hastighed under start og stop slides hovederne.

Normalt har tromlen et magnethoved for hvert spor.

Der findes tromler med kun et hoved, som så til gengæld kan bevæges aksialt langs tromlen, det man sparrer i hoveder og læse/skriveforstærkere betales med flere bevægelige dele og meget større tilgangstid.

Sædvanligvis er der et spor med klokpulser og et spor til adressering af vinkelpositionen.

En tromle kører normalt synkront med netfrekvensen 3000 omdr./min. svarende til, at det tager 20 msek. for en omdrejning.

Middelventetiden ved læsning er derfor 10 msek.

Er det en hel "omdrejning", som skal læses eller skrives kan ventiden elimineres fuldstændig ved hjælp af en speciel adresseringsteknik.

Tromlelagre er dyre og anvendes sjældent i nyere konstruktioner. I stedet anvendes i udstrakt grad pladelagre.

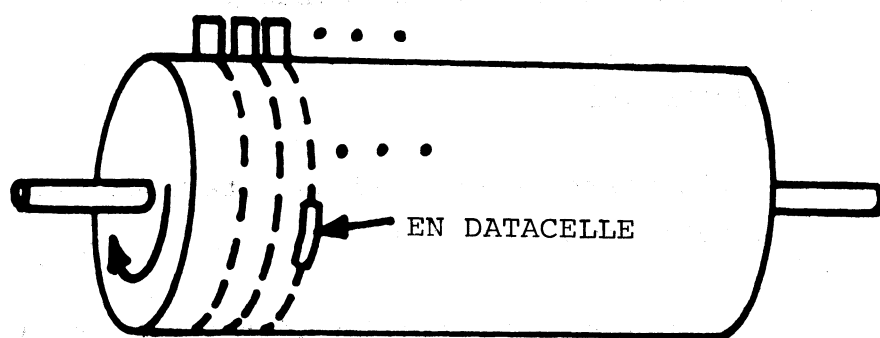


Fig. 5.6 Tromlelager

### Pladelager

Et pladelager består af en stabel plader med magnetisk belægning på overfladen. Stablen indeholder fra en til ti plader med en diameter fra ca. 25 cm. op til ca. 130 cm. Pladestablen, som kan være udskiftelig, roterer med en hastighed omkring 1200 omdr./min.

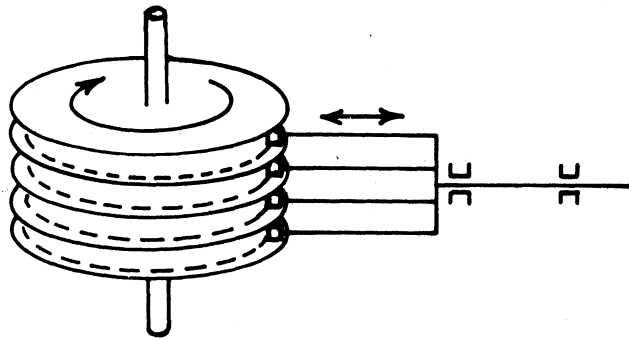


Fig. 5.7 Pladelager

Mellem pladerne er anbragt en eller flere "arme" med et læse/skrivehoved, som kan bevæges ud og ind. Ved hjælp af et fjedersystem presses læse/skrivehovederne ind imod undersiden og oversiden af pladerne, men det tynde luftlag, som roterer sammen med pladerne, forhindrer at hovederne berører pladerne. Uden denne "luftisolering" ville plader og hovederne hurtigt ødelægges.

Et moderne pladelager med 10 plader kan rumme op til 200.000.000 tegn.

Til anvendelse i forbindelse med billigere minidatamater er det ikke rentabelt at anvende et dyrt pladelager.

Der er derfor udviklet et væsentligt billigere pladesystem kaldet Floppy Disc.

En Floppy Disc enhed indeholder normalt 2 udskiftelige plader med en diameter på ca. 20 cm.

Pladerne er bøjelige og kan sendes i et brev. En enkelt plade rummer normalt 256K bytes (8 bit ord). En Floppy Disc enhed med interface koster i dag fra kr. 6.000,00 til 30.000,00.

En datamaskine med baggrundslager kræver normalt et relativt stort program til at administrere placering på lageret og datatransporterne. Dette program kaldes et operativsystem.

### 5.3. Terminaler

Terminaler kan opdeles i 2 kategorier - skrivende terminaler og dataskærme.

Nyere typer skrivende terminaler kan skrive op til 80 tegn pr. sek. og indeholder meget få bevægelige dele.

Ældre typer kan skrive 10 tegn pr. sek. og er rent mekanisk virkende med mange bevægelige dele og deraf følgende større vedligeholdelse. Årsagen til, at de stadig sælges, er at de udover tastatur og skriveenhed også indeholder strimmellæser og strimmelperforator. F.eks. Teletype.

Dataskærme er i dag billigere end skrivende terminaler. Transmissionshastigheden er valgbar, og kan vælges mellem 10 og ca. 2000 tegn pr. sek.

En dataskærm indeholder et relativt stort halvlederlager, da der jo skal gemmes et helt billede bestående af f.eks. 24 linier med 80 tegn pr. linie. Hvert tegn fylder 6 bit. Altså et lager på 11.520 bit.



Lageret er ofte opbygget som 6 cirkulerende skifteregistre.

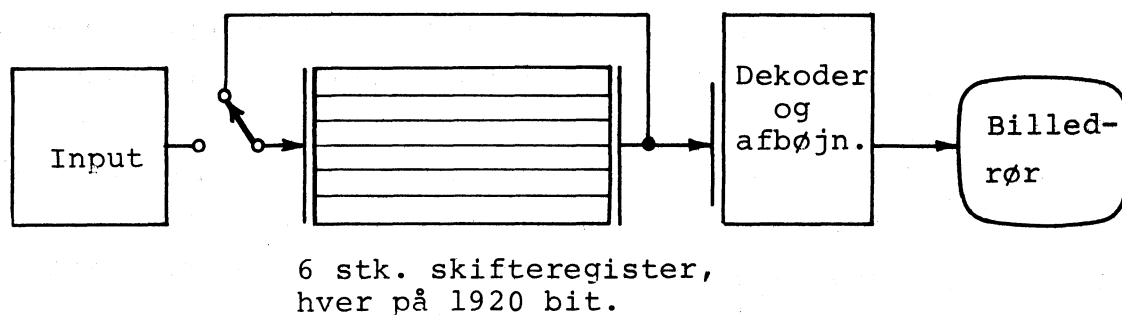


Fig. 5.8

Her ses et stærkt forenklet blokdiagram af en data-skærm.

In/outheden modtager karakterer fra tastaturet eller fra den tilkoblede datamaskine, og omsætter dem til 6 bit kode.

Den indlæste information cirkulerer til stadighed i skifteregisteret med en klokfrekvens 1920 gange større end billedfrekvensen.

Der er mulighed for at udskifte sidste karakter med en fra Input efter hvert billede.

Karaktererne fremkommer, som en 5 x 7 prik matrix på skærmen, styret af en dekoder og afbøjningsenhed.

På grund af den hurtige billedfrekvens, ser det ud som om alle karakterer står på skærmen samtidig.

Transmissionen mellem terminal og datamaskine foregår på serieform. Hastigheden måles i BAUD.

Antal BAUD er lig 1 divideret med den tid, målt i

sek., som det korteste tegnelement (0 eller 1) tager.

Ved binær transmission (2 betydende tilstande) kan man sige, at antal BAUD er lig antal bit pr. sek., men begreberne er ikke ækvivalente.

Transmissionen foregår normalt asynkront, da modtageren ikke ved, hvornår der sendes en karakter.

Først sendes en startbit, dernæst 7 informationsbærende bit, en paritetsbit (til generering af lige eller ulige paritet) og tilslut en eller 2 stopbit. Ialt 10-11 bit pr. karakter.

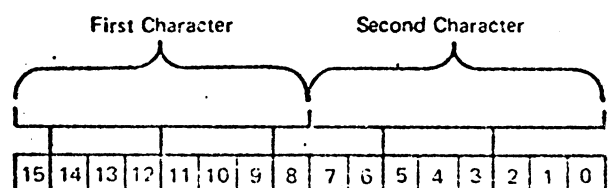
Karaktererne er fastlagt efter en standardkode, normalt ASCII kode - ASCII står for American National Standard Code for Information Interchange.

ASCII-karaktersæt. I HP 2100 "lagres" to ASCII-karakterer i hver datacelle (ord).

# CHARACTER CODES

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
A	040400	000101
B	041000	000102
C	041400	000103
D	042000	000104
E	042400	000105
F	043000	000106
G	043400	000107
H	044000	000110
I	044400	000111
J	045000	000112
K	045400	000113
L	046000	000114
M	046400	000115
N	047000	000116
O	047400	000117
P	050000	000120
Q	050400	000121
R	051000	000122
S	051400	000123
T	052000	000124
U	052400	000125
V	053000	000126
W	053400	000127
X	054000	000130
Y	054400	000131
Z	055000	000132
0	030000	000050
1	030400	000061
2	031000	000062
3	031400	000063
4	032000	000064
5	032400	000065
6	033000	000066
7	033400	000067
8	034000	000070
9	034400	000071
space	020000	000040
!	020400	000041
"	021000	000042
#	021400	000043
\$	022000	000044
%	022400	000045
&	023000	000046
'	023400	000047
(	024000	000050
)	024400	000051
*	025000	000052
+	025400	000053
,	026000	000054
-	026400	000055
.	027000	000056
/	027400	000057

ASCII Character	First Character Octal Equivalent	Second Character Octal Equivalent
:	035000	000072
;	035400	000073
<	036000	000074
=	036400	000075
>	037000	000076
?	037400	000077
@	040000	000100
[	055400	000133
\	056000	000134
]	056400	000135
^	057000	000136
_	057400	000137
ACK	036000	000174
Ⓢ	036400	000175
ESC	037000	000176
DEL	037400	000177
NULL	000000	000000
SUM	000400	000001
EOA	001000	000002
EOM	001400	000003
EOT	002000	000004
WRU	002400	000005
RU	003000	000006
BELL	003400	000007
FE <sub>0</sub>	004000	000010
HT/SK	004400	000011
LF	005000	000012
V <sub>T</sub> AB	005400	000013
FF	006000	000014
CR	006400	000015
SO	007000	000016
SI	007400	000017
DC <sub>0</sub>	010000	000020
DC <sub>1</sub>	010400	000021
DC <sub>2</sub>	011000	000022
DC <sub>3</sub>	011400	000023
DC <sub>4</sub>	012000	000024
ERR	012400	000025
SYNC	013000	000026
LEM	013400	000027
S <sub>0</sub>	014000	000030
S <sub>1</sub>	014400	000031
S <sub>2</sub>	015000	000032
S <sub>3</sub>	015400	000033
S <sub>4</sub>	016000	000034
S <sub>5</sub>	016400	000035
S <sub>6</sub>	017000	000036
S <sub>7</sub>	017400	000037



<div><div><div><div><div>b<sub>7</sub></div><div>b<sub>6</sub></div><div>b<sub>5</sub></div></div><div><div><div><div>b<sub>4</sub></div><div>b<sub>3</sub></div><div>b<sub>2</sub></div><div>b<sub>1</sub></div></div><div><div>Column</div><div>Row</div></div></div></div></div></div></div>					0 <sub>0</sub> 0	0 <sub>0</sub> 1	0 <sub>1</sub> 0	0 <sub>1</sub> 1	1 <sub>0</sub> 0	1 <sub>0</sub> 1	1 <sub>1</sub> 0	1 <sub>1</sub> 1	
					0	1	2	3	4	5	6	7	
0 0 0 0					0	NUL	DLE	SP	0	•	P	'	p
0 0 0 1					1	SOH	DC1	!	1	A	Q	o	q
0 0 1 0					2	STX	DC2	"	2	B	R	b	r
0 0 1 1					3	ETX	DC3	#	3	C	S	c	s
0 1 0 0					4	EOT	DC4	\$	4	D	T	d	t
0 1 0 1					5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0					6	ACK	SYN	&	6	F	V	f	v
0 1 1 1					7	BEL	ETB	'	7	G	W	g	w
1 0 0 0					8	BS	CAN	(	8	H	X	h	x
1 0 0 1					9	HT	EM	)	9	I	Y	i	y
1 0 1 0					10	LF	SUB	*	:	J	Z	j	z
1 0 1 1					11	VT	ESC	+	;	K	[	k	{
1 1 0 0					12	FF	FS	,	<	L	\	l	
1 1 0 1					13	CR	GS	-	=	M	]	m	}
1 1 1 0					14	SO	RS	.	>	N	~	n	~
1 1 1 1					15	SI	US	/	?	O	—	o	DEL

NUL	Null	DLE	Data Link Escape (CC)
SOH	Start of Heading (CC)	DC1	Device Control 1
STX	Start of Text (CC)	DC2	Device Control 2
ETX	End of Text (CC)	DC3	Device Control 3
EOT	End of Transmission (CC)	DC4	Device Control 4 (Stop)
ENQ	Enquiry (CC)	NAK	Negative Acknowledge (CC)
ACK	Acknowledge (CC)	SYN	Synchronous Idle (CC)
BEL	Bell (audible or attention signal)	ETB	End of Transmission Block (CC)
BS	Backspace (FE)	CAN	Cancel
HT	Horizontal Tabulation (punched card skip) (FE)	EM	End of Medium
LF	Line Feed (FE)	SUB	Substitute
VT	Vertical Tabulation (FE)	ESC	Escape
FF	Form Feed (FE)	FS	File Separator (IS)
CR	Carriage Return (FE)	GS	Group Separator (IS)
SO	Shift Out	RS	Record Separator (IS)
SI	Shift In	US	Unit Separator (IS)
		DEL	Delete

American National Standard Code for Information Interchange (ASCII 68).

Fig. 5.9

Mindst betydende bit (b1) sendes først, og mest betydende bit (b7) sendes sidst.

"1" er lav spænding - 3 til - 15 volt og

"0" er høj spænding + 3 til + 15 volt.

ABE ser således ud sendt med lige paritet og 2 stopbits:

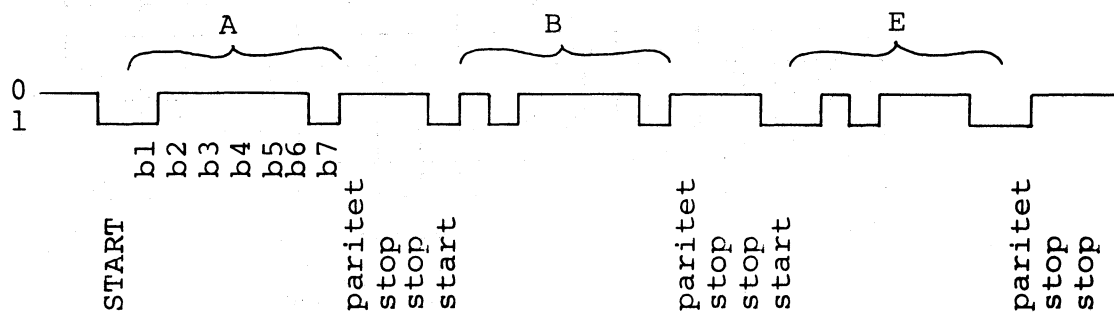


Fig. 5.10

## Strimmellæser

Papirstrimler fås til 5-6-7 og 8 huller. Mest anvendt er 8 huller på 1" brede papirstrimler.

Hulafstanden er 1/10" og mellem 3. og 4. hul er der et lille hul kaldet Feed-hole (fremføringshul).

Der findes mekaniske og optiske læsere. Mekaniske læsere trækkes frem af et tandhjul, som passer til feed-hullerne, og hullerne læses ved hjælp af 8 vippearmer, som slutter en elektrisk kontakt, hvergang de møder et hul.

Mekaniske læsere er langsomme 10 - 30 karakterer pr. sek.

I optiske læsere trækkes strimmelen frem af en motor imod en modtryksrulle. Over strimmelen sidder en aflang lampe og under strimmelen sidder 9 fototransistorer. Strømpulsen, som kommer fra den fototransistor, der læser feed-hole, er kortere end de øvrige, og bruges til at angive det rigtige læsetidspunkt.

På denne måde bliver strimmelhastigheden uden betydning, og man kan klare sig med en relativ billig motor.

Hvis motoren ikke kan standse indenfor 1 karakter, er det nødvendigt, at indskyde et bufferlager mellem læseren og datamaten.

Der fås læsere, som kan læse op til 2000 tegn pr. sek. (RC 2000 fra Regnecentralen). Det normale er dog 500 tegn pr. sek.

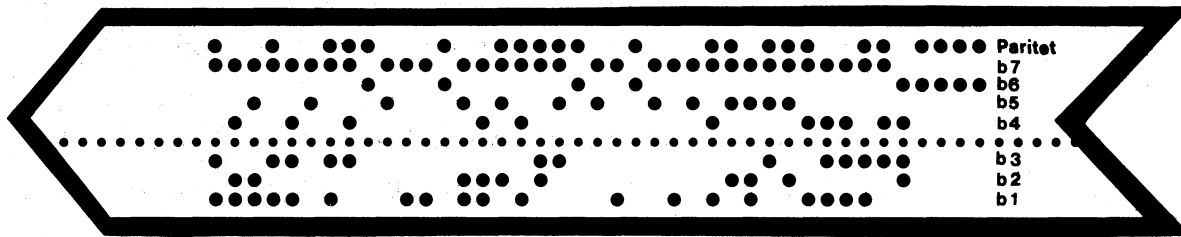


Fig. 5.11

## Strimmelhuller

En strimmelhuller eller Punch er en outputenhed, som genererer papirstrimmelen. Det er en rent elektro-mekanisk funktion, hvor der stilles store krav til nøjagtigheden, hvormed hullerne udstandses. (Se fig. 5.11)

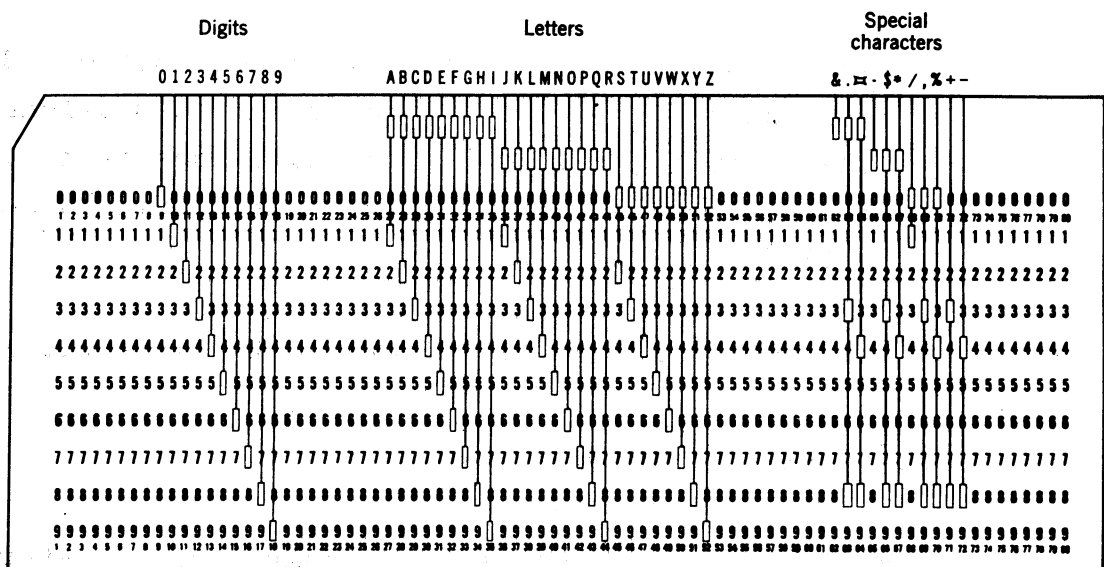
Strimmelhullere fås med hastigheder op til 75 tegn pr. sek.

## Hulkort

Et andet meget populært databærende medium er hulkort.

En af fordelene ved hulkort fremfor strimler er, at man kan rette i teksten ved at udskifte et enkelt hulkort. Skal der rettes på en strimmel, må der laves en ny strimmel.

Et hulkort indeholder 80 kolonner. I hver kolonne er der mulighed for at standse 12 huller. Hullerne benævnes 12-11-o-1-2-3-4-5-6-7-8 og 9.



Coding for punched cards.

Fig. 5.12 Hulkort.



Der er altså  $2^{12}$  kombinationsmuligheder i hver kolonne. Til det normale ASCII alfabet anvendes en speciel hulkortkode kaldet Hollerith-kode, hvor man højst har 3 huller i hver kolonne. Ulykkeligvis findes der 2 næsten identiske koder.

Hollerith 26 kode og Hollerith 29 kode, som afviger fra hinanden ved specialkaraktererne.

Hullemaskiner er normalt selvstændige maskiner, som ikke er tilkoblet en datamaskine.

Via et indlagt programkort kan de styres til kun at hulle tal i visse kolonner og kun bogstaver i andre. Der findes også et programkort-styret tabulerings-system, som gør det lettere for hulledamerne at hulle korrekt. Mange hullemaskiner skriver teksten i klar skrift i kanten af kortet.

Hulkortlæseren er en hyppigt anvendt inputenhed i større systemer. Læseren må indeholde en kodeomsætter, som omsætter fra Hollerithkode til ASCII kode, inden data sendes til datamaten.

Det er nødvendigt med en 80 karakters buffer, da læsehastigheden ikke er den samme som transmissionshastigheden. Læsningen foregår optisk, og det er nødvendigt, at kortet bevæger sig med konstant hastighed henover de 12 fototransistorer.

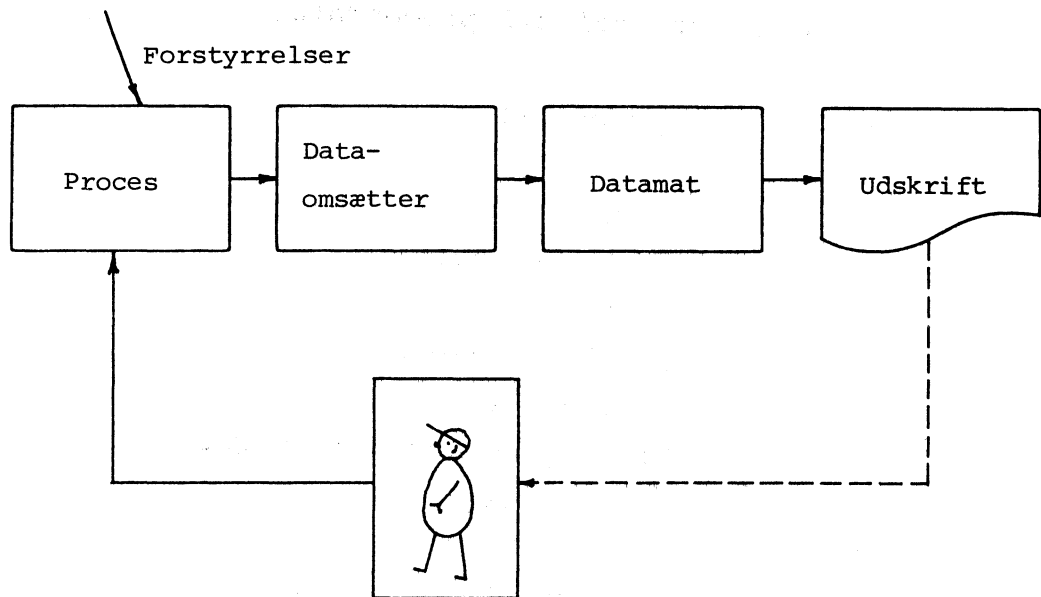
Hulkortlæsere læser fra 200 til 1200 kort pr. minut.

#### 5.4. Procesdata

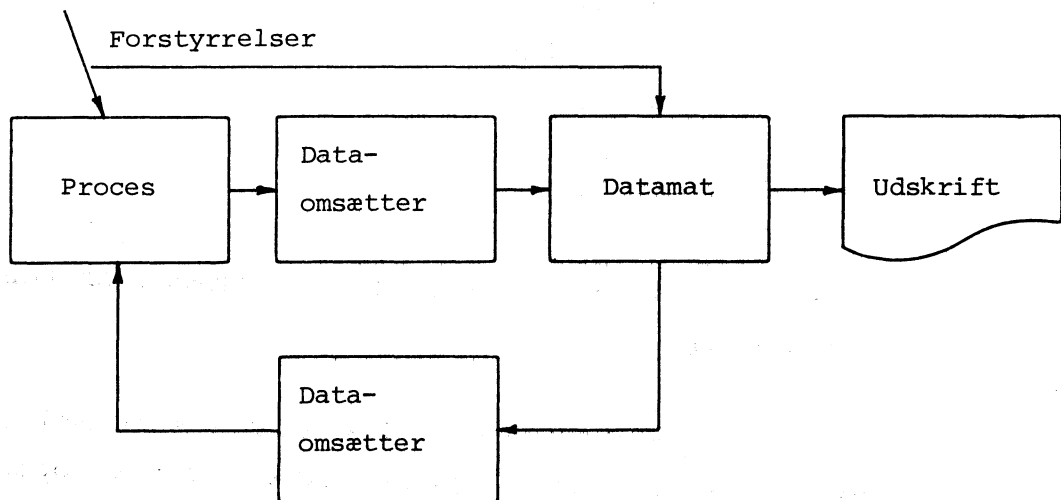
Minidatamaten anvendes i udstrakt grad til styring og overvågning af maskiner og fabriksanlæg.

Der findes forskellige automatiseringsgrader.

Det simpleste er overvågning.



Det mest komplicerede er On Line processtyring.



Vi skal her se lidt på dataomsættere fra procesdata til tal og fra tal til procesdata.

Eksempler på procesdata:

Input:

On-Off signaler

Kontakter

Spændingsfølere

Fotoceller

Ultralyd giver - modtager

Termostater

Kontinuerte signaler:

Termometre

Trykmålere

Flowmålere

Tachometre

Ampere - voltmetre

(Tællere)

Output:

Relær

Ventiler

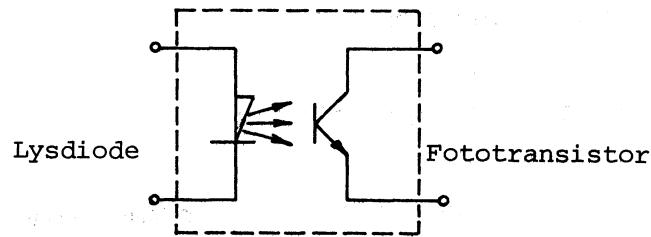
Motorer

Hydraulik

Pneumatik

Output til relær, ventiler og lignende er rent on/off signaler.

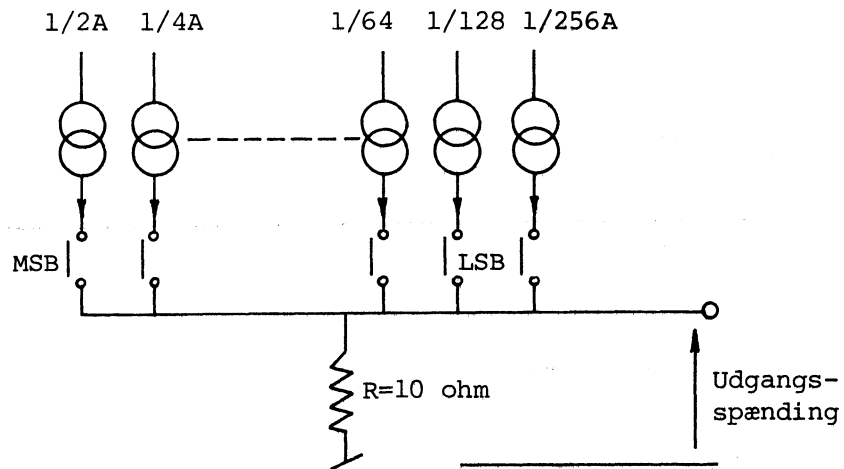
Normalt ønsker man galvanisk adskillelse mellem data-mat og proces. Dette gøres normalt v.h.a. optokoblere. Et stk. pr. bit ind eller ud.



I fototransistoren er collektorstrømmen proportional med lysstyrken.

Efter optokobleren sidder en niveauomsætter, som omsætter fra 5 volt til 24 - 48 eller 220 volt, som igen styrer relæ eller ventil on/off.

Kontinuerte (eller næsten kontinuerte) udgangssignaler genereres v.h.a. en Digital til Analog omsætter D/A.



Her er vist en 8 bit D/A konverter.

De 8 output bit styrer de 8 kontakter

1 lukket kontakt

0 åben kontakt

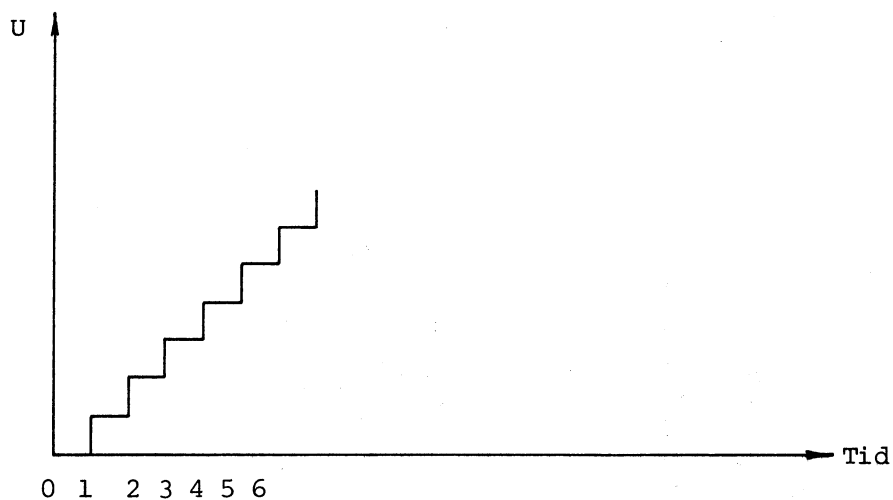
Udgangsspændingen bliver

$$U = R \times (\text{summen af de strømme som er tilkoblet})$$

Den højeste spænding fås med FF Hex. og bliver:

$$\begin{aligned} U_{\text{max.}} &= 10 \times (1/2 + 1/4 + \dots + 1/128 + 1/256) \\ &= 10 \times \frac{255}{256} = 9,96 \text{ volt} \end{aligned}$$

Overføringskarakteristikken bliver



en trappekurve.

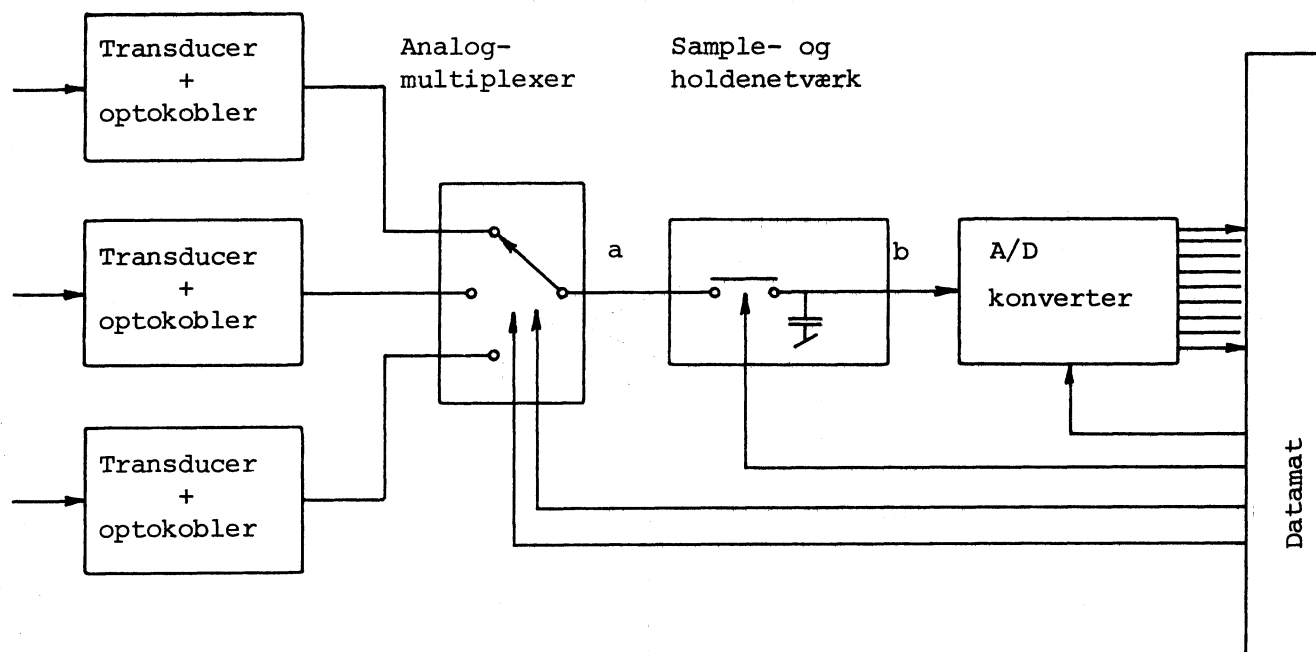
Jo flere bit der anvendes, jo finere bliver kurven.  
Prisen stiger tilsvarende; en 12 bit D/A koster ca. 4  
gange så meget som en 10 bit D/A

## Input

On-off signalerne opfattes som 1 eller 0 og kan efter transducer og optokobler indlæses direkte.

## Kontinuerte signaler

Eks. på input af kontinuerte signaler:

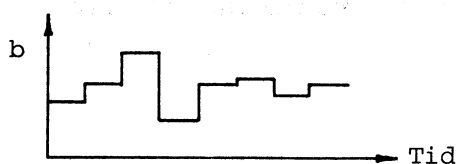
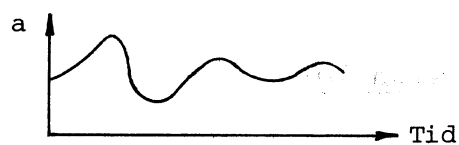


Da en A/D konverter er dyr, anvendes en multiplexer (en digitalt styret omskifter).

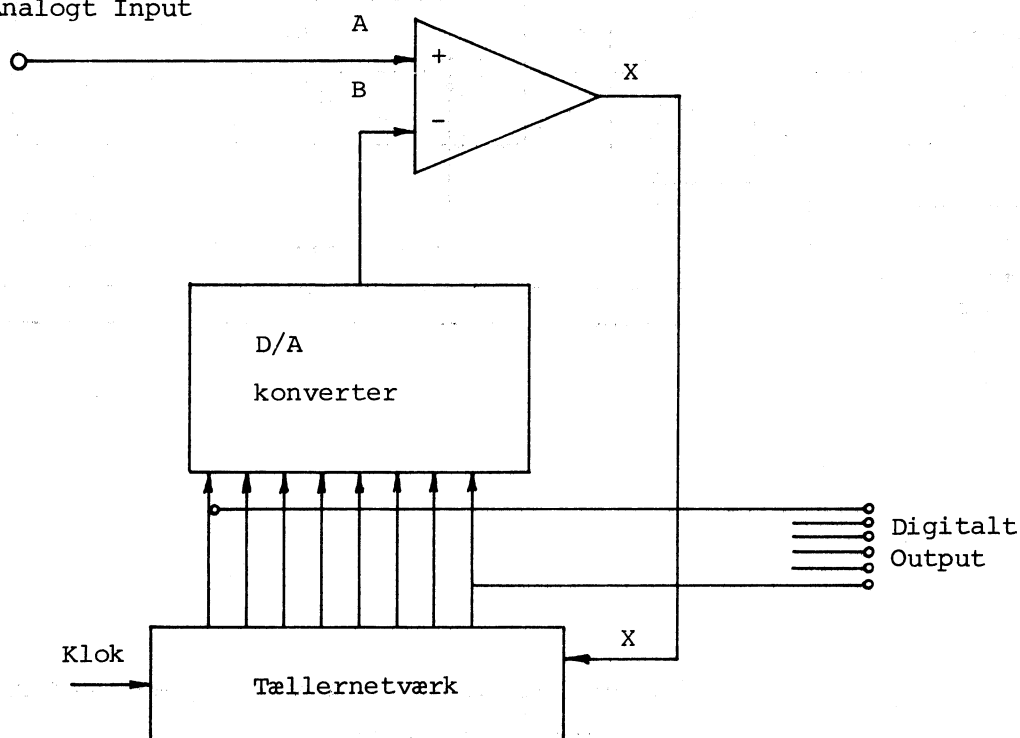
De forskellige signaler måles på forskellige tidspunkter, bestemt af programmet.

Mellem multiplexer og konverter er indskudt et Sample- og Holdenetværk. Det er nødvendigt ved hurtigt varierende signaler, da A/D konverteren kræver, at dens indgangsspænding er konstant i den tid, det tager for den at konvertere.

Eks.



Analogt Input



En A/D konverter kan bestå af en komparator, som giver  $x = 1$  for  $A > B$  og  $x = 0$  for  $A \leq B$ .

En D/A konverter og et tællernetværk, som tæller op for  $x = 1$ .

Når  $x$  bliver  $= 1$  står der et tal på udgangen som svarer til A's værdi.



## 6. DIAGRAMMERING

Diagrammering er et grafisk hjælpemiddel, som bruges ved beskrivelse af systemer. Ved et system forstås her en kombination af tilstande, aktiviteter og sammenhænge som tilsammen har en funktion. Begrebet system er helt generelt og er på ingen måde begrænset til at omfatte systemer i relation til EDB.

Normalt er et system afgrænset, eller man vil søge det afgrænset, således at det fremstår som en enhed, stor eller lille, som har en veldefineret berøringsflade med omverdenen. Hermed opnår man, også i beskrivelsen, at kunne nøjes med at betragte systemet i sig selv og behøver kun at medtage omverdenen i den udstrækning denne påvirker systemet gennem dets indgange og bliver påvirket af systemet gennem dets udgange.

Som eksempel på et system kan nævnes et kamera. Det udgør en afsluttet enhed, det har nogle indgange (indstillingskontroller, linse) hvorigennem det kan påvirkes, og det har nogle udgange (blitztilledninger, den eksponerede film) hvorigennem det "påvirker" omverdenen. Kameraet kan tænkes nedbrudt i delsystemer, som hver for sig opfylder kravene til et system. Det kan være linsesystem, lukkermekanisme, filmtransport, kamerahus.

Andre eksempler på systemer er: CPR-registeret, et firmas lønningsregnskab, telefonnettet.

Et system, der således i en eller anden hensigt er blevet afgrænset, kan beskrives på flere måder. Beskrivelsen retter sig efter formålet, og tit benytter man sig af det værktøj, som kaldes diagrammering. Her ved opnår man, som det vil fremgå af det følgende, at kunne beskrive systemet netop så detaljeret, som den påtænkte anvendelse kræver. Endvidere kan man gennem diagrammeringen tydeliggøre et systems opsplitning i delsystemer eller, modsat, vise hvordan et system

sammen med andre systemer indgår i et overordnet system. Endelig kan diagrammeringen redegøre for systemets ind- og udgange så detaljeret som påkrævet.

I de følgende tre afsnit vil diagrammeringsteknikkens anvendelse på tre forskellige planer blive gennemgået. Det er:

1. funktionsdiagrammet, som på overordnet plan beskriver et system ved dets aktiviteter og tilstande.
2. systemdiagrammet, som mere detaljeret beskriver disse aktiviteter og tilstande, samt
3. blokdiagrammet, som i detaljer og med anvendelse af logiske tilstande og betingelser beskriver alle faser i en given aktivitet.

## 6.1. Funktionsdiagram

Ved en funktionsbeskrivelse af et system, således som dette begreb tidligere er defineret, forstår man en beskrivelse af den (de) funktion(er), som sammenkæder ind- og udgange i systemet. En grafisk afbildning heraf kaldes et funktionsdiagram. Til grund for funktionsdiagrammet ligger det såkaldte black-box diagram, f.eks.:

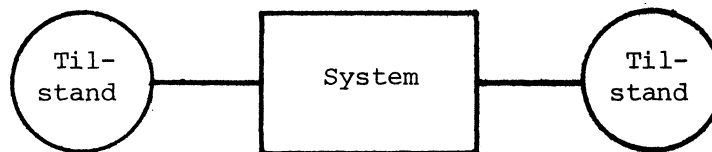


Fig. 6.1.a.

i hvilket der overhovedet ikke er angivet noget om selve systemet, men kun hvilken sammenhæng det indgår i. Rent konkret kunne black-box diagrammet se således ud:

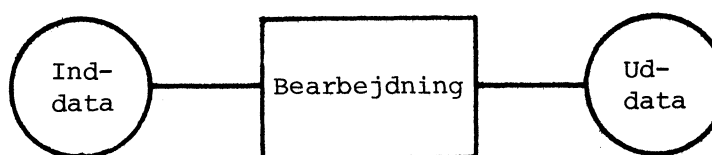


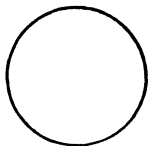
Fig. 6.1.b.

I et funktionsdiagram hørende hertil vil symbolet mærket "BEARBEJDNING" være splittet op i nogle aktiviteter og nogle tilstande, som angives med hhv. firkanter og cirkler.

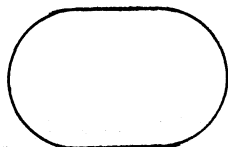
Symbolerne er knyttet sammen med forbindelseslinier, som angiver de forskellige forløbs retninger. En tekst anbragt inden i symbolet forklarer kort dets betydning.

## 6. 1. 1. Symboler for funktionsdiagrammer

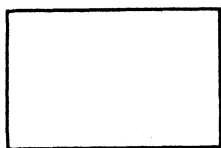
Følgende symboler bruges til funktionsdiagrammering:



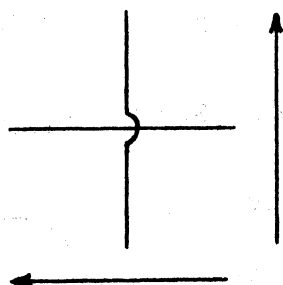
Tilstand. (state)



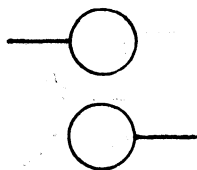
Alternativt symbol for tilstand.  
Symbolets længde kan variere efter behov.



Aktivitet. (process)

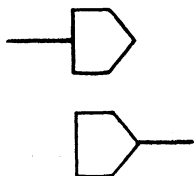


Forbindelseslinier mellem symboler. (flow line) Krydsende linier uden forbindelse angives med bro. Pile kan bruges hvis strømmen går ovenfra og ned eller fra venstre mod højre, og skal bruges hvis strømmen går modsat.

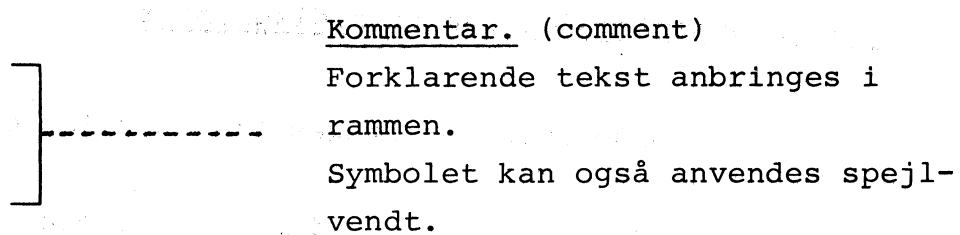


Forbindelsesled. (connector)

Forbindelsesled mellem diagramdele på samme side. Symbolerne mærkes med koder, som kæder sammenhørende symboler sammen.



Forbindelsesled mellem diagramdele på hver sin side. Symbolerne mærkes med koder, som kæder sammenhørende symboler sammen.

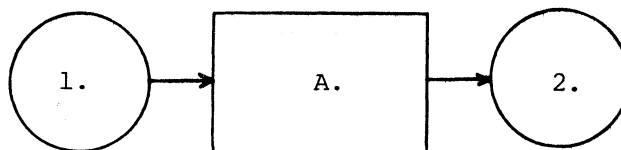


## 6.1.2. Tegneregler for funktionsdiagrammer

For tegning af funktionsdiagrammer gælder visse regler:

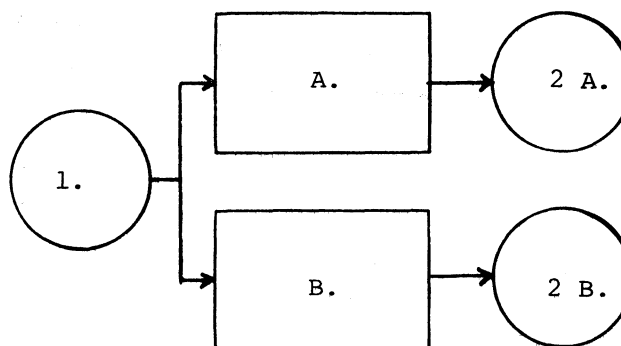
1. Tilstande og aktiviteter i et diagram følger skiftevis efter hinanden, begyndende med og afsluttende med tilstande.

Eksempel:



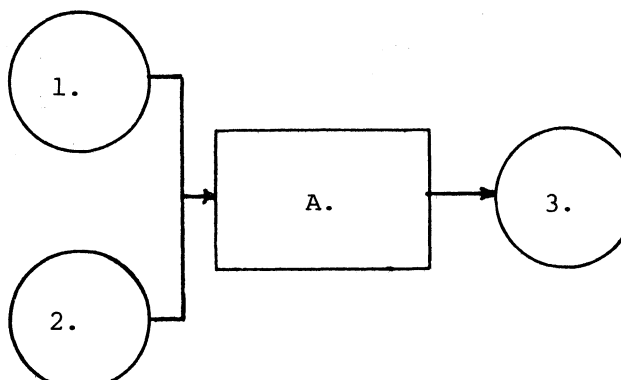
2. Een tilstand kan føre til een eller flere aktiviteter.

Eksempel:



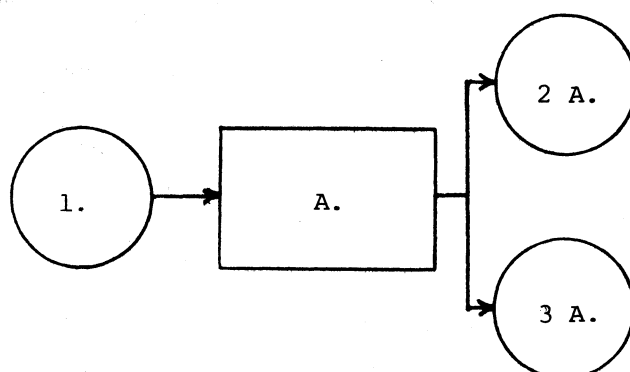
3. Flere tilstande kan føre til een og samme aktivitet.

Eksempel:



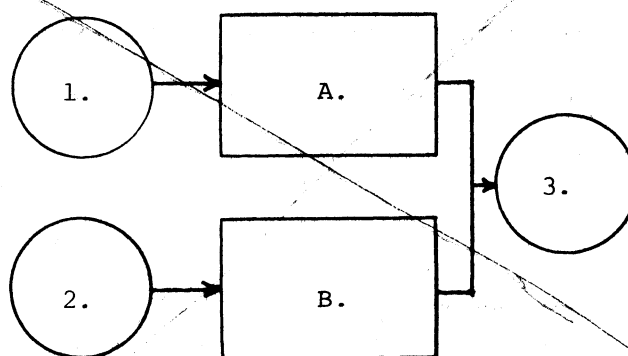
4. Een aktivitet kan føre til flere tilstande.

Eksempel:



5. Flere aktiviteter kan aldrig føre til samme tilstand.

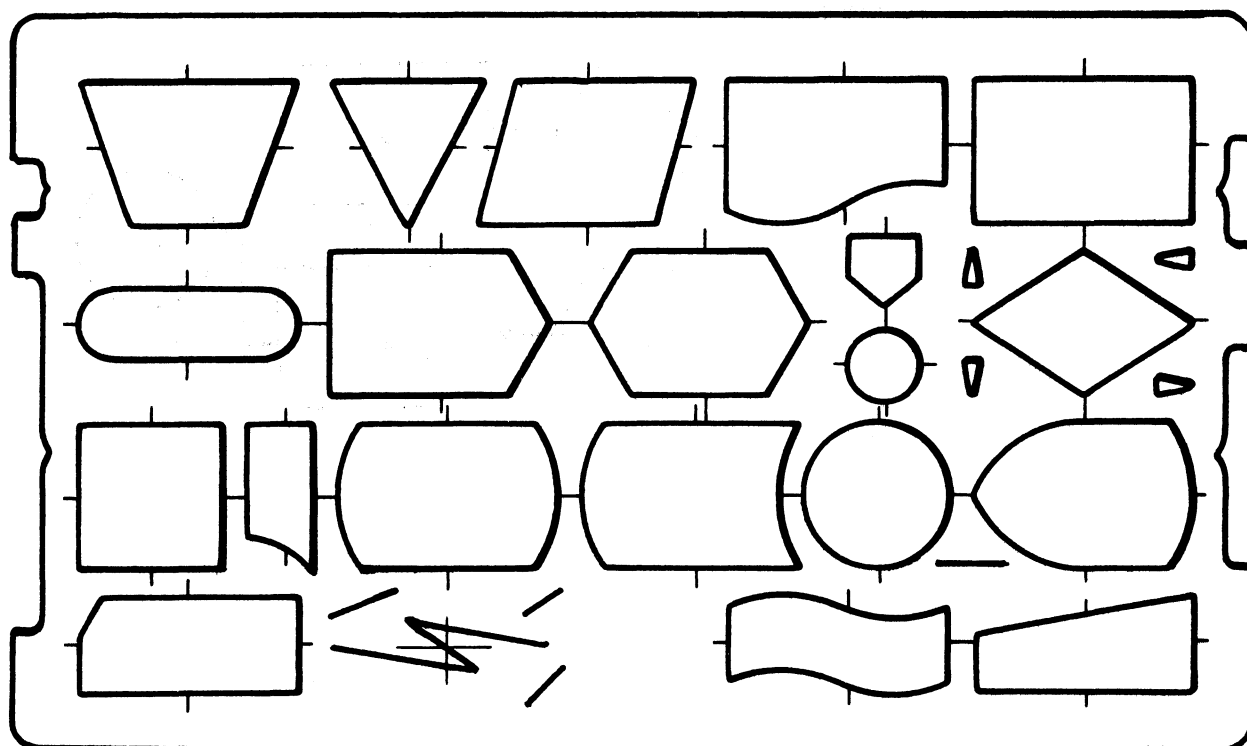
Eksempel:



### 6.1.3. Hjælpemidler ved diagrammering

Ved tegning af diagrammer kan forskellige hjælpemidler bruges til opnåelse af et standardiseret udseende. Det drejer sig om diagramskabelon og diagrammeringsark.

Diagramskabeloner fås i flere udformninger og med større eller mindre afvigelser i de enkelte symboler. I fig. 6.1.3.a. er vist en skabelon, Linex 1175, med symboler efter Dansk Standard, DS 2090: Diagramsymboler.

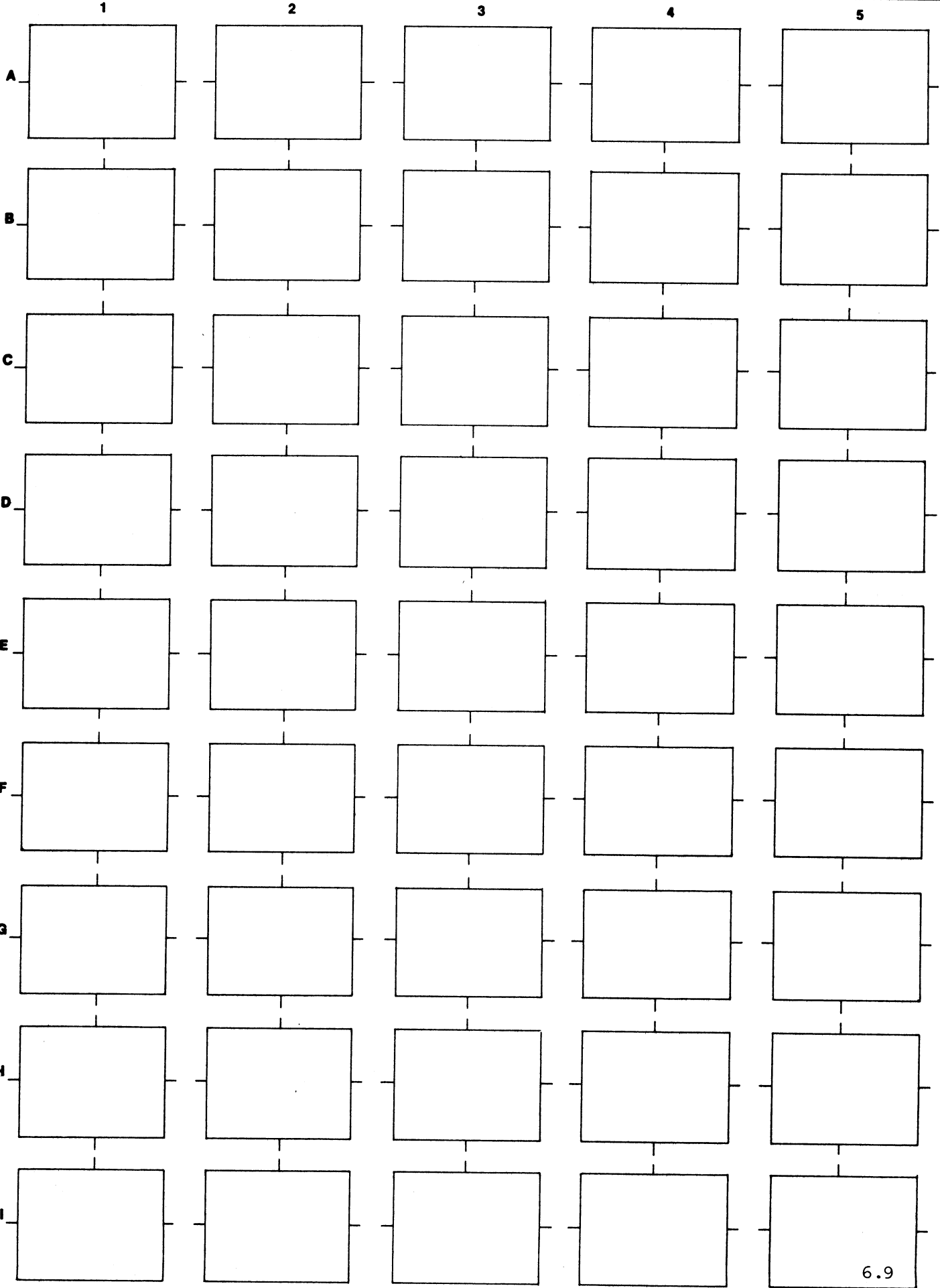


Et eksempel på et diagrammeringsark er vist i fig. 6.1.3.b. side 6.10. Felterne er trykt med svag farve og er forsynet med række- og søjlemarkering, således at deres koordinater kan angives. Dette kan f.eks. udnyttes ved mærkning af forbindelsesled.



DIAGRAMMERING

Udfyldt af	Udfyldt den	Opgavenr	Bilagnr	Sidenr
Processens navn / Nr.				



#### 6.1.4. Eksempel på funktionsdiagram.

Nedestående eksempel viser hvordan aktiviteten "bearbejdning" fra fig. 6.1.b. kan tænkes beskrevet. Eksemplet viser forløbet i en "data sortering".

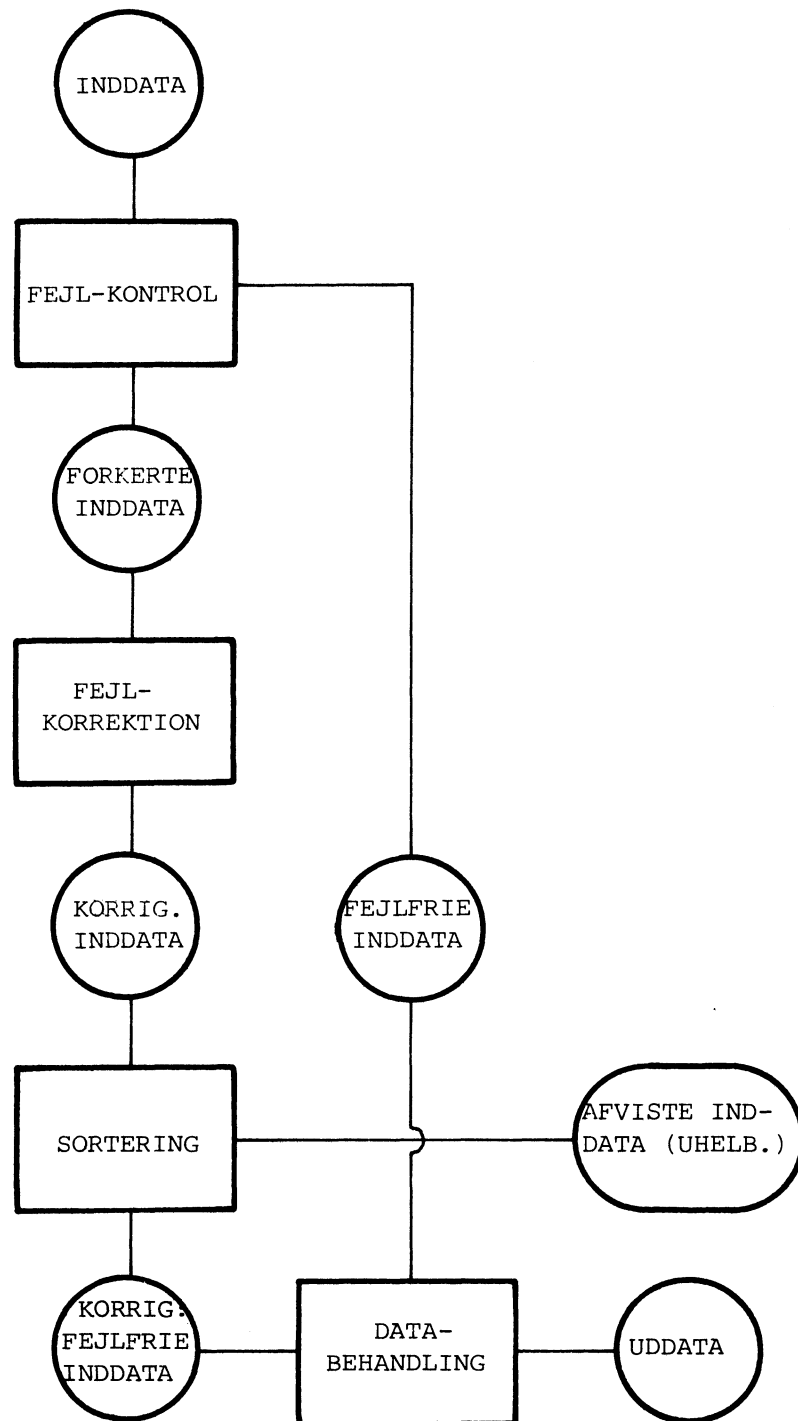
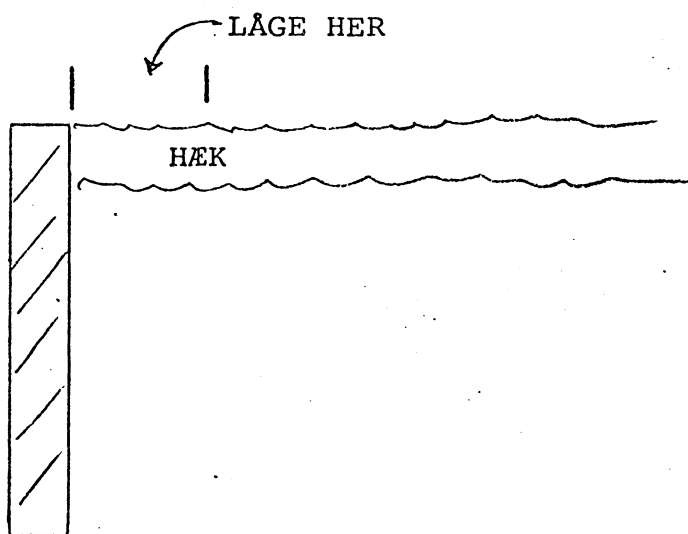


Fig. 6.1.4.a.

## O P G A V E 6.1

Lav et funktionsdiagram for forløbet opsætning af havelåge, fra idé til den færdige låge.

Lågen indsættes mellem en mur og en hæk, d.v.s. gammel hæk fjernes.



Datamaskiner-ME

[illegible]

**B**

**C**

D

E

**F**

**G**

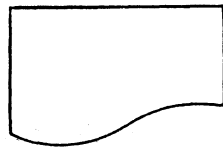
## H

1

## 6. 2. Systemdiagram

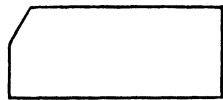
Den forholdsvis grove illustration, som et funktionsdiagram giver af et system, kan udbygges på forskellige måder, så forskellige grader af detaljer fremkommer. Det kan f.eks. ske med anvendelse af visse nye symboler til angivelse af specielle tilstande og aktiviteter i systemet, i så fald taler man om et systemdiagram. Symbolerne er følgende:

### 6. 2. 1. Tilstandssymboler (i/u betegner inddata/uddatamedier).



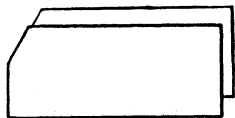
#### Udfyldt blanket. (document)

En databærende blanket, som kan bestå af papir, karton, plast el. lign. f.eks. en dataliste, et kon-tokort (evt. med magnetbåndsstri-ber) etc.



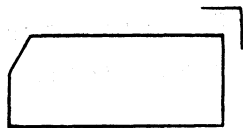
#### Hulkort. (punched card)

Et i/u medium, der f.eks. kan være et hulkort, et mærkelæsekort el. lign.



#### Kortstak. (deck of cards)

En samling hulkort



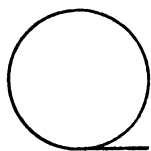
#### Kortfil. (file of cards)

En samling sammenhørende poster på hulkort.



#### Hulstrimmel. (punched tape)

Et i/u medium, som er en hulstrim-mel.



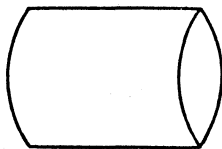
Magnetbånd. (magnetic tape)

Et i/u medium, som er et magnetbånd.



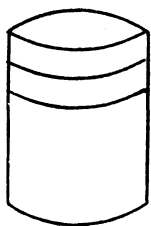
Manuel indlæsning. (manual input)

Enten en indlæsningsproces i hvilken data indføres manuelt på behandlingstidspunktet, f.eks. over tastaturer, ved skiftere eller et indlæsningsmedium.



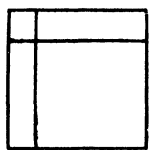
Magnettromlelager. (magnetic drum)

Et i/u medium, som er en magnetromle



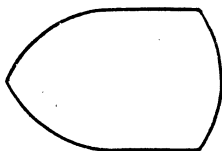
Magnetpladelager. (magnetic disk)

Et i/u medium, som kan være en kassette med magnetkort el. lign.



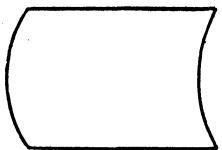
Indre lager. (core)

Et indre lager, som kan være tyndfilm el.lign.



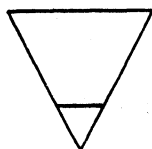
Audiovisuel dataterminal. (display)

Et i/u medium, i hvilket data fremstilles for visuel aflæsning ved hjælp af dataskærme (evt. med lyspen), konsolskrivere, plottere el. lign.



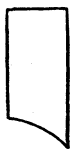
Direkte styret lager. (on-line storage)

Et direkte styret (on-line) lager, som gør brug af en hvilken som helst type af direkte (on-line) lagring, f.eks. magnetkort, tromlelager el. pladelager.



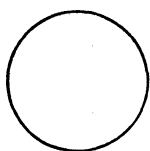
Indirekte styret lager. (off-line storage)

Et indirekte styret (off-line) lager, som kan være et hvilket som helst databærende medium.



Strimmel, optisk læselig. (transmittal tape)

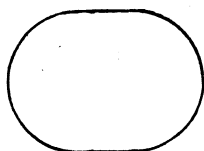
Et i/u medium, som er en optisk læselig strimmel, f.eks. fra et kasseapparat.



Uspecificeret tilstand.

Datamængde ifølge DS (data collection).

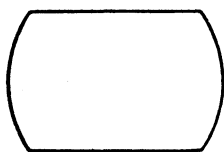
Datamængde der overføres fra én proces til en anden.



Eller

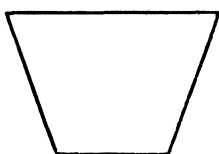
Alternativt symbol.

## 6. 2. 2. Aktivitetssymboler



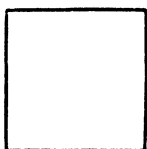
### Registrering. (keying)

En manuelt styret registrering, f. eks. hulning af hukort eller hulstrimmel.



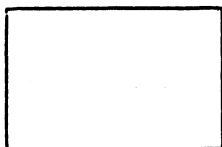
### Manual proces. (manual operation)

Enhver manuelt udført proces.



### Hjælpeproces. (auxiliary operation)

En proces udført på udstyr, som ikke er styret direkte af centralenheden.



### Proces. (process)

Enhver proces udført på EDB-anlæg.

### Hjælpesymboler:



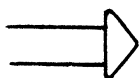
### Datatransmissionskredsløb. (communication link)

En funktion, hvor data overføres via telekommunikationsled.



### Både-og. (both-and)

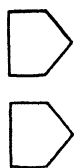
Alle forbindelseslinier skal følges samtidigt. Bøen skal altid vende mod aktiviteten.



### Enten-eller. (either-or)

Een eller flere forbindelseslinier kan følges, afhængig af informationsindholdet. Spidsen skal altid vende mod aktiviteten.



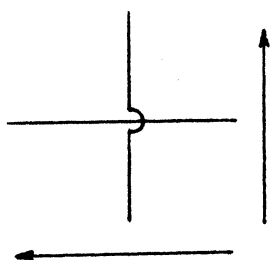


#### Forbindelsesled. (connector)

Forbindelsesled mellem diagramdele på hver sin side. Symbolerne mærkes med koder, som kæder sammenhørende symboler sammen.



Forbindelsesled mellem diagramdele på samme side. Symbolerne mærkes med koder, som kæder sammenhørende symboler sammen.



#### Forbindelseslinier mellem symboler. (flow line)

Krydsende linier uden forbindelse angives med bro. Pile kan bruges hvis strømmen går ovenfra og ned eller fra venstre mod højre, og skal bruges hvis strømmen går modsat.

#### 6.2.2. Tegneregler for systemdiagram.

Tegneregler for systemdiagrammer er de samme, som er beskrevet under funktionsdiagrammer. Side 6.7



### 6.2.3. Eksempel på systemdiagram.

Nedestående eksempel viser et systemdiagram for funktionen "bearbejdning", fig. 6.1.b.

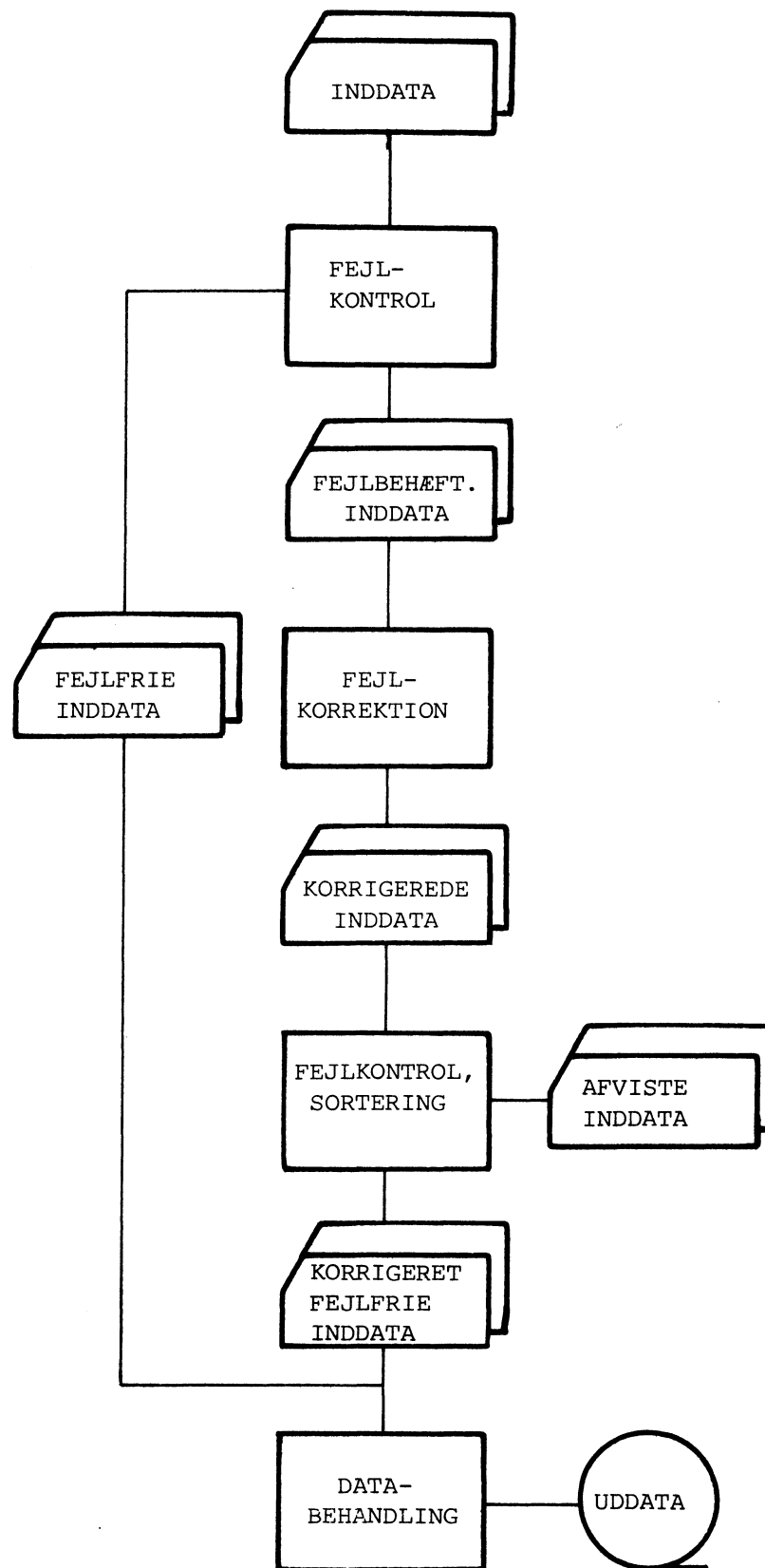


Fig. 6.2.3.a.

## Datamaskiner-ME

### Opgave 6.2

#### DIAGRAMMERING

Tegn et systemdiagram, der viser følgende forløb:

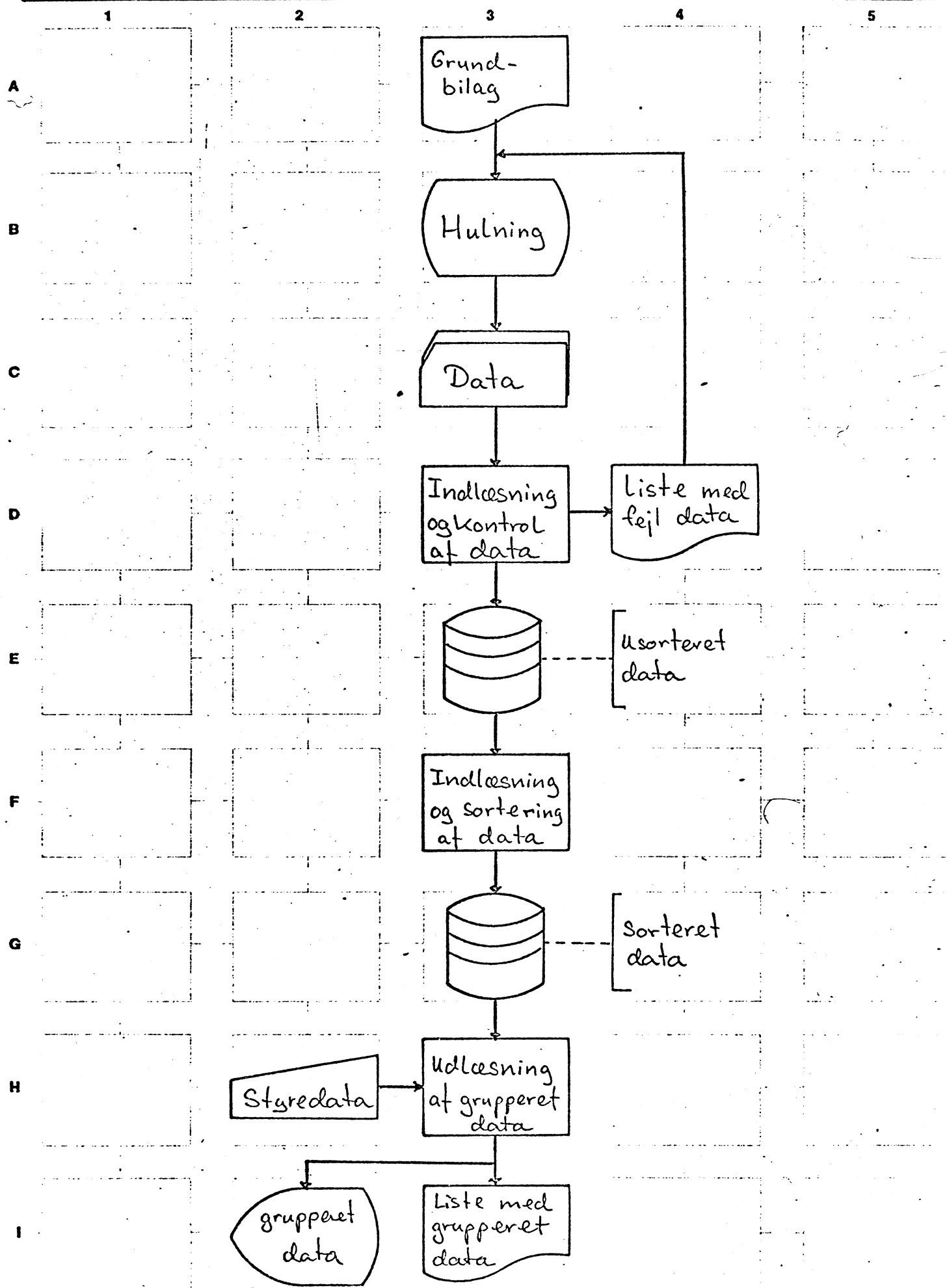
Ud fra et grundbilag, hulles manuelt hulkort som indlæses i en datamaskine. De indlæste data opbevares på pladelager.

I forbindelse med indlæsning af hulkortene listes fejlbehæftede data, hvorefter nye kort kan hulles og indlæses sammen med allerede eksisterende informationer. Det er nu muligt via en dataterminal at få listet forskellige grupper af data.



# DIAGRAMMERING

Udfyldt af <b>LB.</b>	Udfyldt den <b>77.11.02</b>	Opgavenr <b>6.2</b>	Bilag nr <b>ME.</b>	Sidenr
Processens navn / Nr.				



### 6.3. Blokdiagram

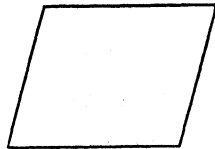
Ved hjælp af blokdiagrammet kan en aktivitet beskrives i diagramform med så mange detaljer, som er nødvendig til den givne anvendelse. Det karakteristiske ved enhver form for proces er:

1. Aktiviteterne foregår i en bestemt rækkefølge.
2. Som udgangspunkt for en handling kan der foreligge een eller flere betingelser.

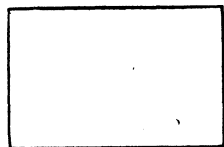
Begge punkter illustreres klart i et blokdiagram.

#### 6.3.1. Symboler for blokdiagrammer

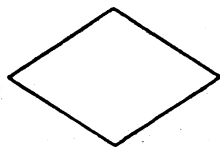
Følgende symboler anvendes i blokdiagrammer:



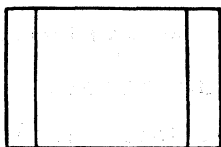
Indlæsning/udlæsning. (input/output)  
En indlæsnings- eller udlæsningsproces (i/u), f.eks. indlæsning af hukort eller udskrivning på linieskriver.



Proces. (process)  
Enhver behandling, bearbejdning, operation eller samling af sådanne, f.eks. udførelsen af en afgrænset operation eller gruppe af operationer, som resulterer i en forandring af værdiform eller lagring af data.



Forgrening. (decision)  
En beslutning eller skifteoperation, som afgør hvilken af flere mulige veje i systemet, der skal følges.



### Andet steds fastlagt proces.

(predefined process)

En navngiven proces, som består af en eller flere operationer eller programtrin, der er specificeret andet steds, f.eks. et underprogram.



### Programmodifikation: (preparation)

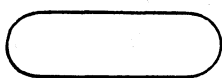
En ændring af en ordre eller gruppe af ordrer, som forandrer selve programmet.



### Gentagende udførsel: (do-continue)

Indleder en proces, der udføres mere end én gang.

### Hjælpesymboler



### Terminalpunkt. (terminal)

Et diagrampunkt, som kan være start, stop, slut eller afbrydelse.



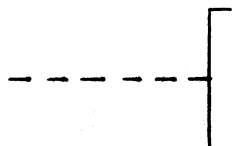
### Forbindelsesled (connector)

(til/fra samme side)



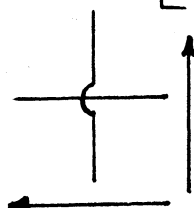
### Parallel udførsel. (parallel mode)

Påbegyndelse eller afslutning af to eller flere samtidige operationer.



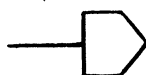
### Kommentar. (comment)

Forklarende tekst anbringes i ramme.



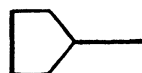
### Forbindelseslinier. (flow line)

forbindelseslinier mellem andre symboler.



### Forbindelsesled (connector)

(til/fra forskellige sider)



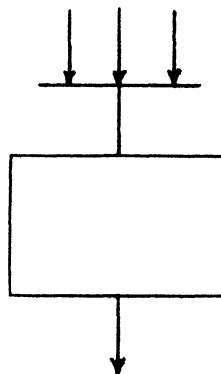


### 6.3.2. Tegneregler for blokdiagrammer

Tegnereglerne er vist som eksempler:

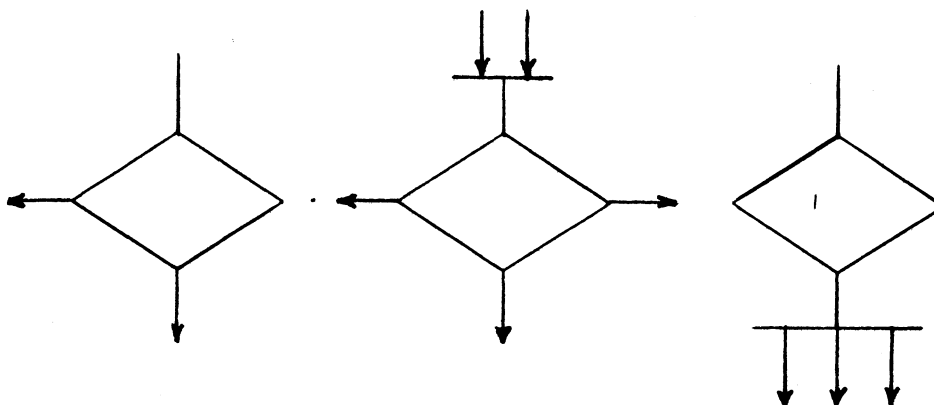
Alle symboler kan have flere indgange.

Eksempel:



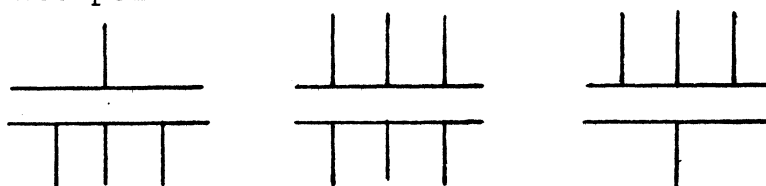
Kun forgreningssymbolet kan have flere udgange.

Eksempel:

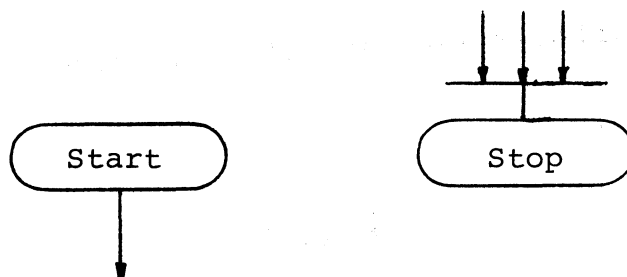


Dog kan specielt symbolet for parallel udførsel, både have flere ind- og udgange.

Eksempel:



Kun et startsymbol og et stopsymbol i et blokdiagram.



Til brug i blokdiagrammer kan endvidere følgende relationstegn bruges:

- > større end
- < mindre end
- = lig med
- > større end eller lig med
- < mindre end eller lig med
- ≠ forskellig fra
- $\left. \begin{array}{l} := \\ \leftarrow \end{array} \right\}$  sæt lig med (dynamisk lighedstegn)

Eksempel:

```
X = 5
X := X + 1
X = 6
```

### 6.3.3. Eksempel på blokdiagram.

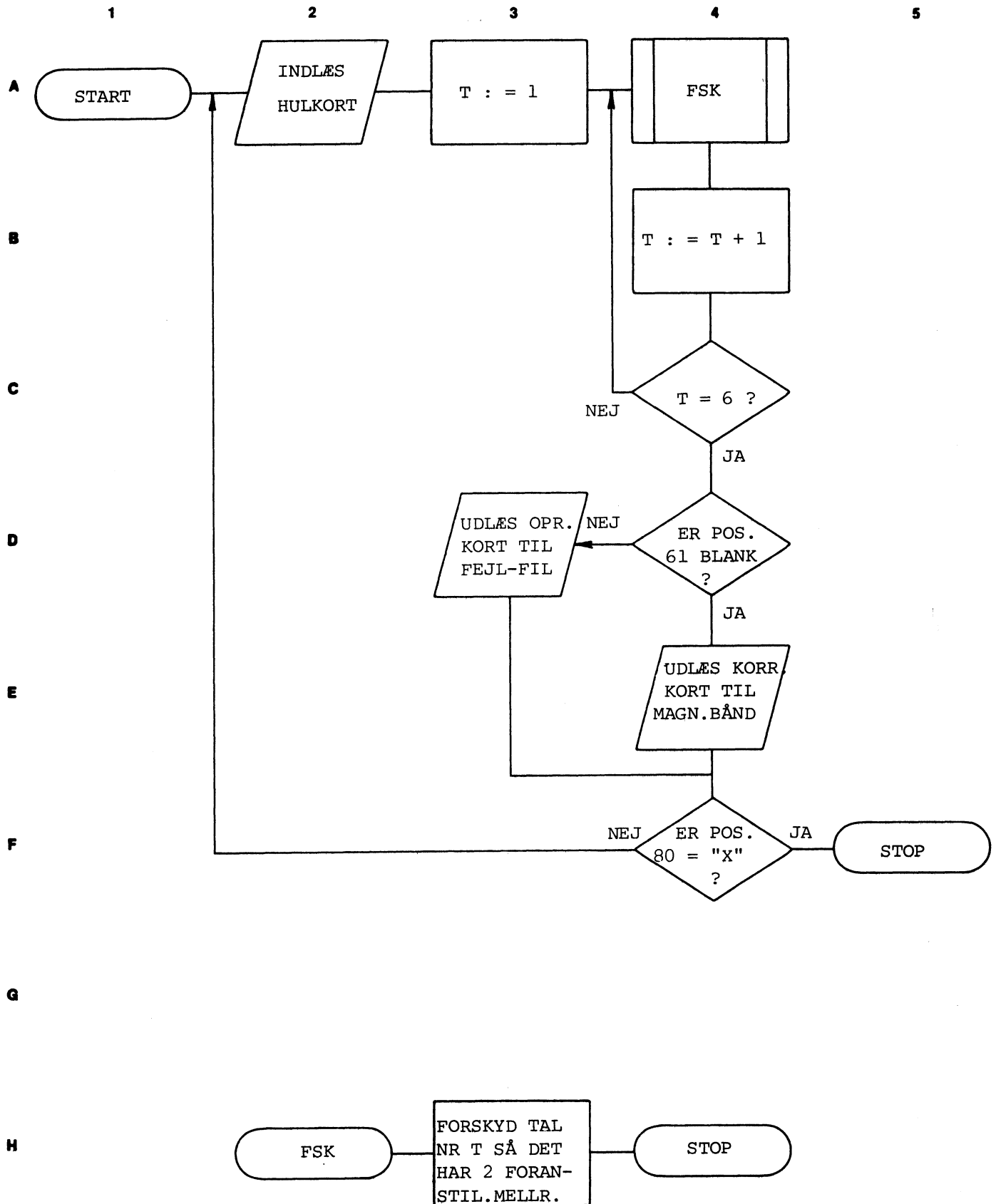
I fig. 6.3.3.a. side 6.27 er vist et eksempel på et blokdiagram. Den viste aktivitet er fra fig. 6.2.3.a. mærket "Fejlkorrektion". Det er nødvendigt for forståelsen af dette trin at kende formatet for inddata: Et hulkort med inddata rummer 5 tal, hver med 10 cifre og 2 foranstillede mellemrum, ialt 60 udnyttede positioner. Visse af kortene er imidlertid behæftet med en eller begge af følgende fejl:

1. 2 mellemrum et tilfældigt sted er ved hulning blevet til 3.
2. Et 10-cifret tal et tilfældigt sted er ved hulning blevet 11-cifret.

Processen "Fejlkontrol" har sorteret alle fejlbehæftede kort fra. Kun kort med fejltype 1 alene kan korrigeres, resten afvises. Sidste kort i stakken er markeret med "X" i position nr. 80.

# DIAGRAMMERING

Udfyldt af	Udfyldt den	Opgavenr	Bilagnr	Sidenr
Processens navn / Nr.		FEJLKORREKTION, Fig. 6.3.3.a.		



Udfyldt af <b>LB</b>	Udfyldt den <b>2/2-77</b>	Opgaven <b>6.3</b>	Bladnr <b>ME.</b>	Sider <b>5</b>
Processors navn / Nr.				

1 2 3 4 5

A

B

C

D

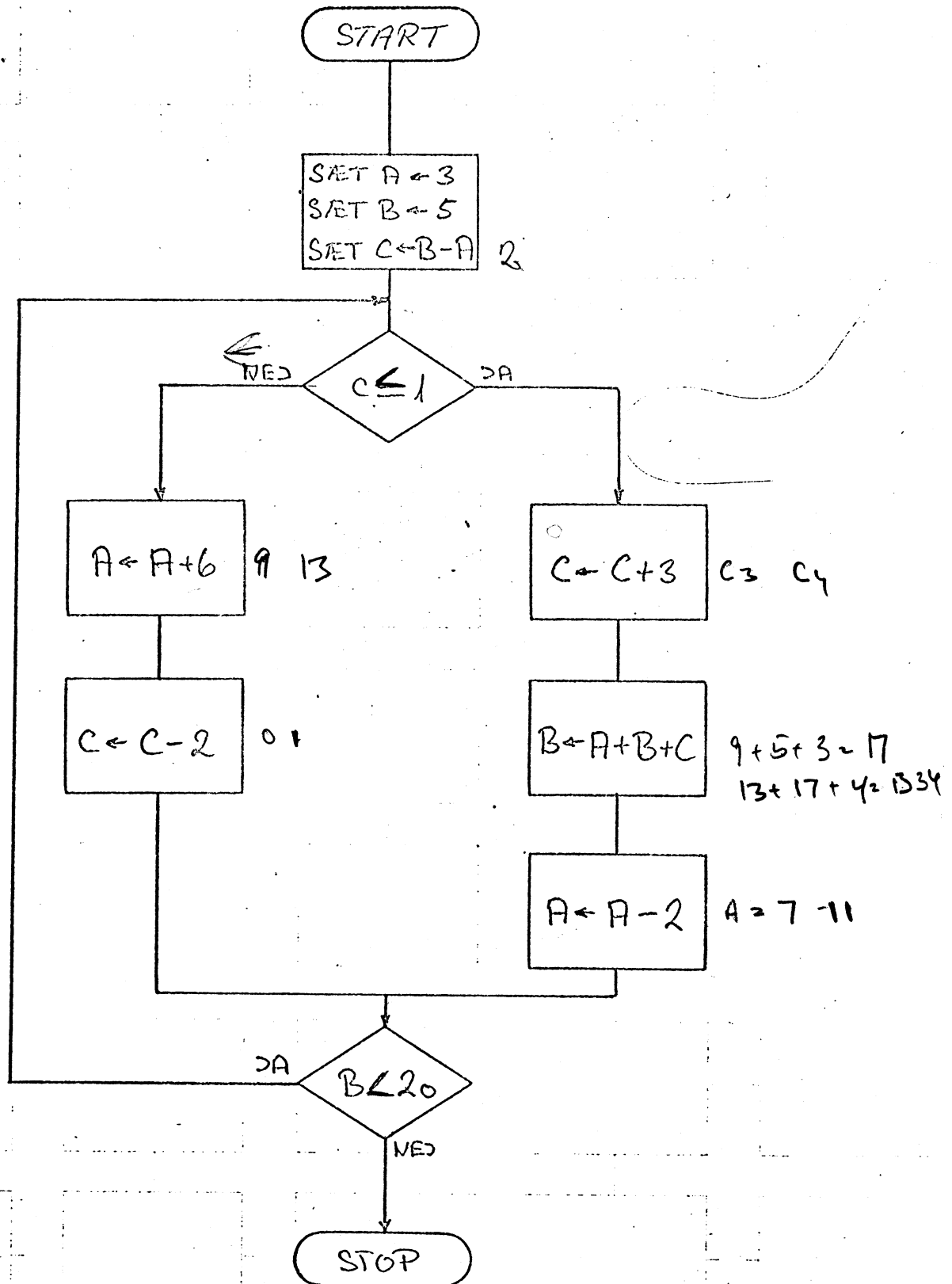
E

F

G

H

I



DIAGRAMMERING

Opgave 6.3

Analysér vedlagte blokdiagram og find værdierne for:

$$A = \underline{11}$$

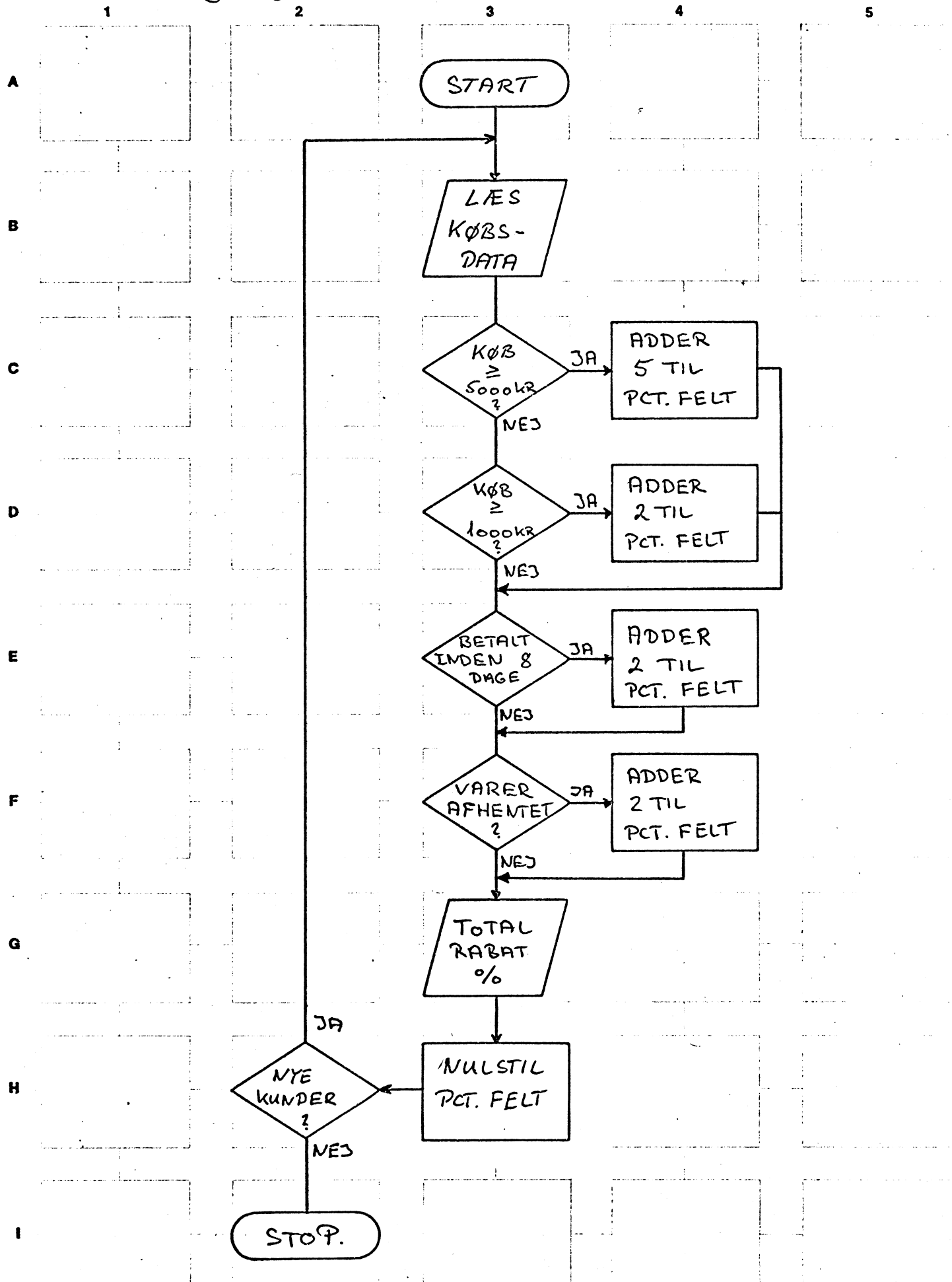
$$B = \underline{34}$$

$$C = \underline{4}$$

Bilag: 1 blokdiagram

# DIAGRAMMERING

Udfyldt af <b>L.B.</b>	Udfyldt den <b>03.08.77.</b>	Opgavenr <b>6.4</b>	Bilagnr <b>ME.</b>	Sidenr
Processens navn / Nr. <b>Rabatberegning</b>				



DIAGRAMMERING

Opgave 6.4

En virksomhed arbejder med en rabatordning efter følgende principper:

Ved et samlet køb på kr. 5.000,- - eller derover ydes 5 pct. rabat, og ved et samlet køb på kr. 1.000,- - eller derover, men under 5.000,- ydes 2 pct. rabat.

Endvidere yder der 2 pct. rabat på betaling inden 8 dage fra fakturamodtagelsen. Endelig ydes der 2 pct. rabat i de tilfælde hvor kunden selv afhenter varen.

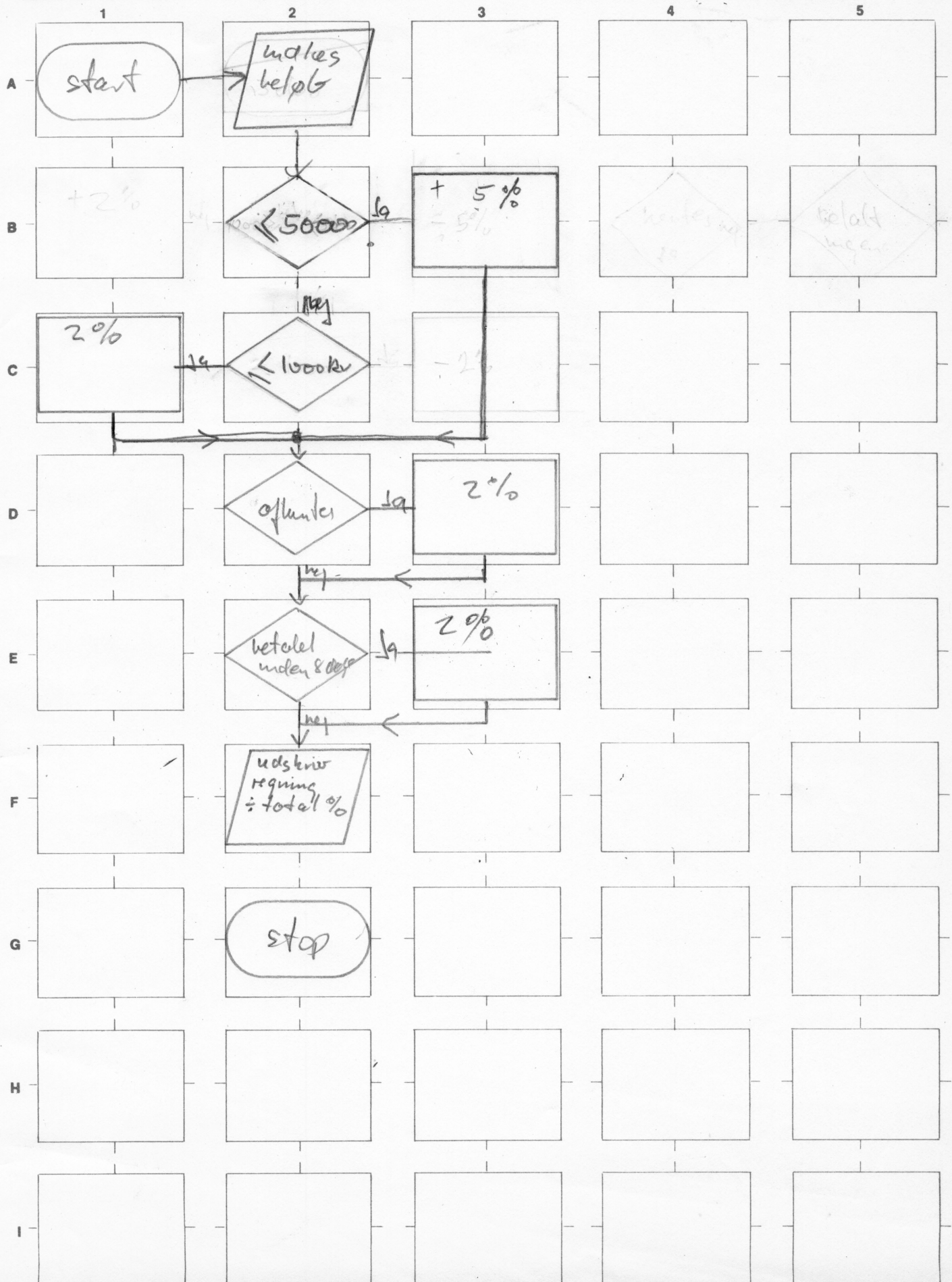
Der ønskes udarbejdet et blokdiagram, som kan benyttes til afgørelse af rabatprocenten ved et vilkårligt beløb.

Udput  
større end 5000 ja nej  
storemind 1000 ja nej  
betalt inden 8 dage ja nej  
afhentes varen ja nej



# DIAGRAMMERING

Udfyldt af	Udfyldt den	Opgavenr	Bilagnr	Sidenr
Processens navn / Nr.				



Datamaskiner-ME

DIAGRAMMERING

Opgave 6.5

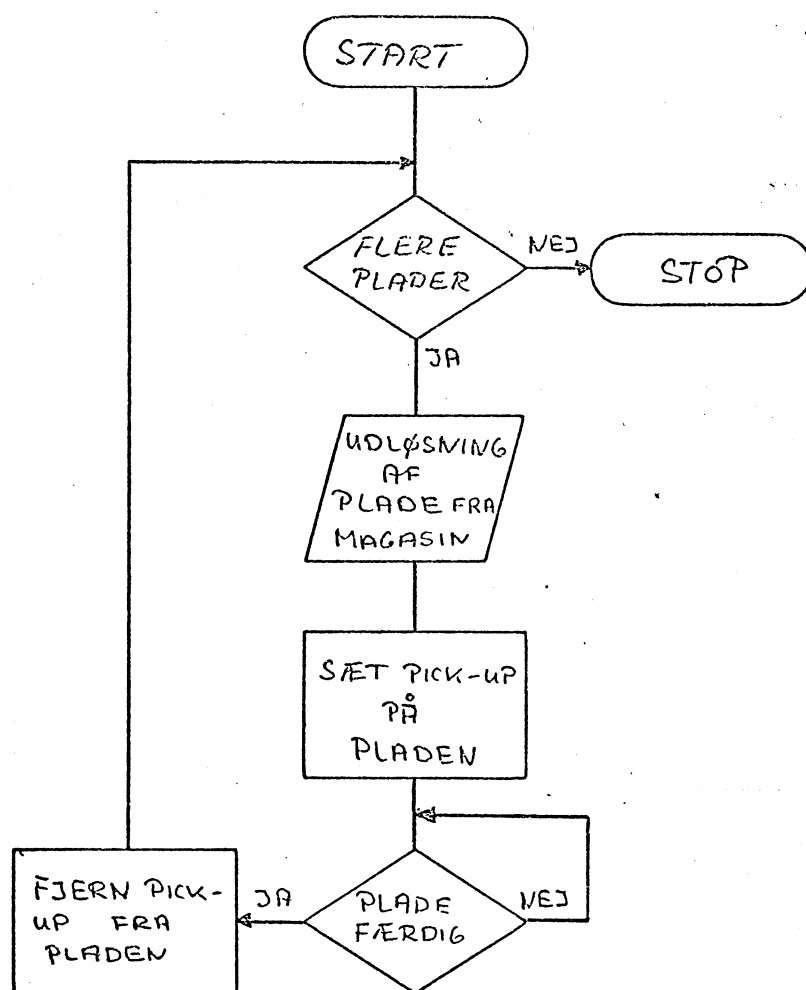
Tegn et blokdiagram for en automatisk pladespiller, med plademagasin.

Blokdiagrammet skal illustrere følgende handlinger og deres afhængighed af hinanden:

- A. Pladespilleren startes.
- B. Udløsning af plade fra magasin.
- C. Pick-up sættes på plade.
- D. Pick-up fjernes fra plade.
- E. Pladespilleren stoppes.

# DIAGRAMMERING

Udfyldt af <b>L.B.</b>	Udfyldt den <b>01.08.77</b>	Opgavenr <b>6.5</b>	Bilagnr <b>ME</b>	Sidenr
Processens navn / Nr. <b>AUTOMATISK PLADESPILLER MED PLADE MAGASIN</b>				
1	2	3	4	5



## 7.-----Programmering i maskinsprog.

En datamaskine kan kun udføre programmer i maskinsprog. Det betyder, at en datamaskines maskinsprog direkte afspejler datamaskinens konstruktion med hensyn til instruktionsrepertoire, adresseringsmåder, dataformater m.v. Programmer skrevet i maskinsprog kan derfor kun anvendes på den maskintype, hvortil de er skrevet.

Maskinsprog består af elementer af bitstreng. D.v.s. et antal 1 og 0 tilstande koblet sammen til enheder, som den pågældende datamaskine i en given situation kan fortolke som ordrer eller data.

Maskinsprog består altså af 2 typer elementer:

1. Ordre fra programmet til styreenheden.
2. Data styreenheden skal arbejde på.

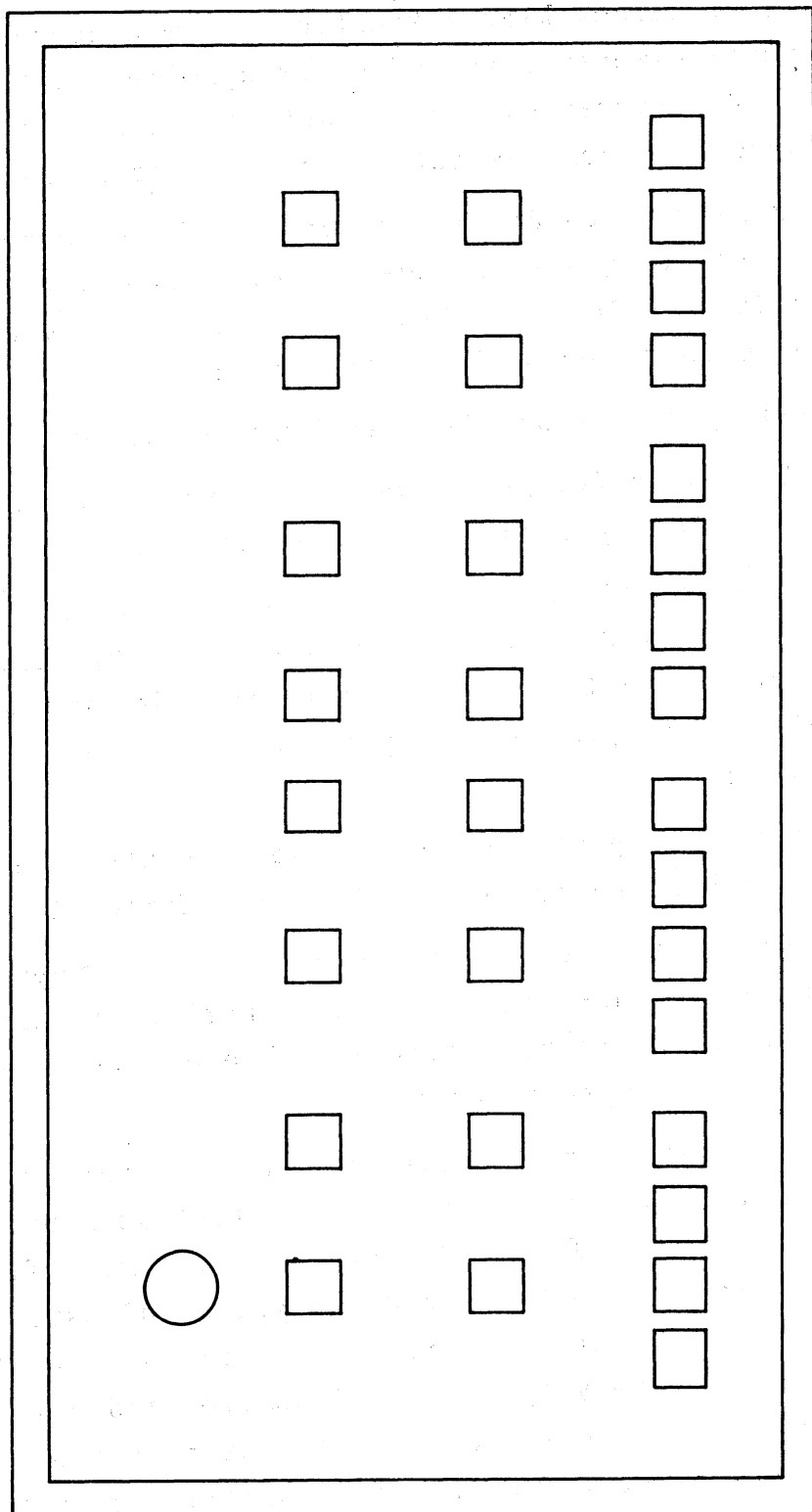
Ordrene må betragtes som de aktive elementer i et program. Det er dem, der bevirker at datamaskinen - i et på forhånd fastlagt handlingsforløb, der vil være forskelligt fra program til program - udfører en given algoritme.

Data må betragtes som de passive elementer, der ikke direkte har indflydelse på handlingsforløbet.

Programmer skrevet ved hjælp af 1 og 0 er meget uhåndterlige og vanskelige at skrive. Derfor vil man ofte se den binære repræsentation omskrevet til oktal eller hexadecimal repræsentation.

Programmering i maskinsprog anvendes i praksis kun af teknikere, når de har brug for hurtigt at kunne lokalisere og rette fejl i datamatiske systemer. Til bl. a. dette formål er de fleste datamaskiner udrustet med et styrepanel (frontpanel) med kontakter og kontrollamper, der kan benyttes til at indsætte mindre maskinprogrammer i lageret, og til at kontrollere og ændre indholdet af centralenhedens forskellige registre.

Fig. 7.1



FRONT PANEL.

## 7.1 Instruktioner.

Instruktioner eller ordrer er de elementer i maskinsprog, der tillægges størst betydning, idet de betragtes som de aktive elementer i et program.

Et sådant elements længde hænger nøje sammen med ordlængden i datamaskinen.

Har en datamaskine lange ord (16 bit og derover) vil en instruktion ofte have samme længde som et ord, og man taler om fast instruktionslængde.

Er ordlængden kortere, finder man, at det til visse instruktionstyper vil være nødvendigt at koble flere ord sammen, og dette fænomen benævnes variabel instruktionslængde.

En instruktion består af 2 dele:

1. Operationskoden.

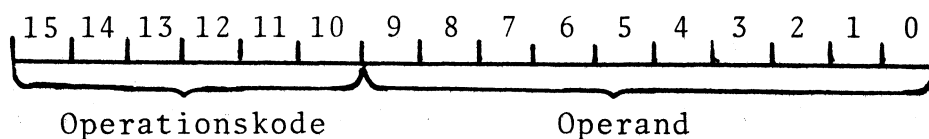
2. Operanden.

Operationskoden fortæller styreenhedens instruktionsdekoder, hvilke operationer der skal udføres.

Operanden fortæller hvor - disse operationer skal udføres.

Med en given instruktionslængde vil et antal bit blive benyttet til operationskoden og resten til operanden.

En datamaskine med 16 bit ordlængde kan f. eks. have instruktioner, der er opdelt således:



med 6 bit til operationskode og 10 bit til operand.

Ihukommende det binære talsystem, vil man straks kunne se, at denne datamaskine maksimalt kan have  $2^6 = 64$  forskellige operationskoder eller aktive instruktioner.

Med variabel instruktionslængde og med en ordlængde på 8 bit, er det almindeligt, at 1 ord benyttes til operationskode, og yderligere 1 eller 2 ord hægtes på som operand.

F.eks. således:

1. 7 6 5 4 3 2 1 0
  2. 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
  3. 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
- Operationskode                      Operand

### 7.1.1 Operationstyper.

Hvilke operationer vil det almindeligvis være ønskeligt, at en datamaskine kan udføre ?

Ser man på de eksisterende datamaskiner, kan man foretage en gruppering af forekommende operationskoder således:

1. Flytning af data.
2. Aritmetiske operationer.
3. Logiske Operationer.
4. Programstyringsoperationer.
5. Ind- / Udlæsning af data.
6. Maskinstyring.
7. Hjelpeoperationer.

#### 1. Flytning af data.

Flytning af data internt i datamaskinen har man ofte brug for. Det kan være flytning fra lageret til styreenheden og omvendt, eller det kan være flytning fra et sted i lageret til et andet sted i lageret. Jo flere forskellige slags flytninger, man ønsker at kunne udføre, jo flere af operationskodens bitkombinationer vil medgå til denne type operationer.

## 2. Aritmetiske operationer.

Skal datamaskinen kunne benyttes til regneoperationer, må der som minimum findes een aritmetisk funktion - som oftest addition. Det bliver imidlertid mere og mere almindeligt, at også mindre datamaskiner direkte kan udføre alle fire almindelige regnearter - addition, subtraktion, multiplikation og division.

Herudover kan man til den aritmetiske gruppe af operationer knytte dem, der vedrører funktioner som komplementering, inkrementering (opadtælling) og dekrementering (nedadtælling). Også her gælder det, at jo flere forskellige aritmetiske operationer, datamaskinen skal kunne udføre, jo flere af operationskodens bitkombinationer vil medgå til denne type operationer.

## 3. Logiske operationer.

Som logiske operationer vil man almindeligvis kunne finde operationer, der dækker de boolske funktioner AND (.), OR (+) og EXCLUSIVE OR (⊕) samt sammenligning af to binære værdier (COMPARE).

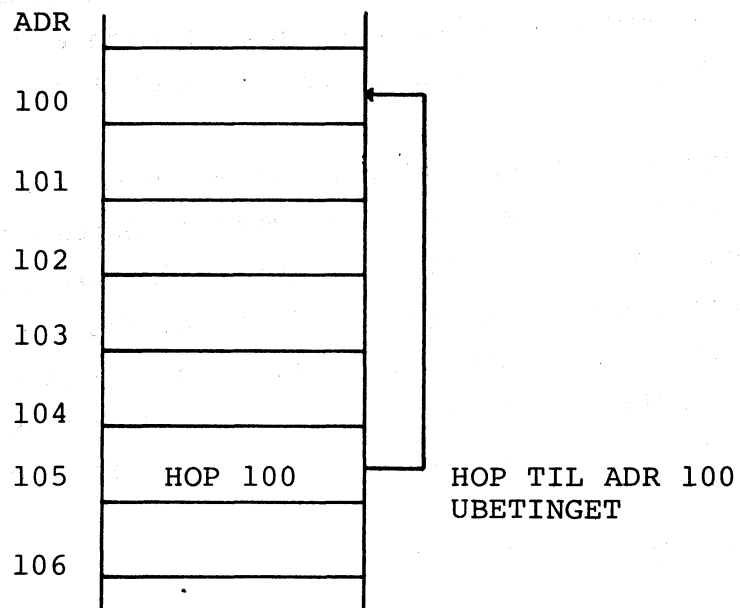
## 4. Programstyringsoperationer.

Udførelsen af et program i en datamaskine foregår i princippet sekventielt. Det er imidlertid nødvendigt at kunne bryde denne sekvens, enten betinget eller ubetinget. Man har brug for at kunne springe til et angivet punkt i programsekvensen. Sådanne spring kaldes for JUMP (Hop) eller BRANCH (Gren) eller som i visse datamaskiner SKIP (Overspring).

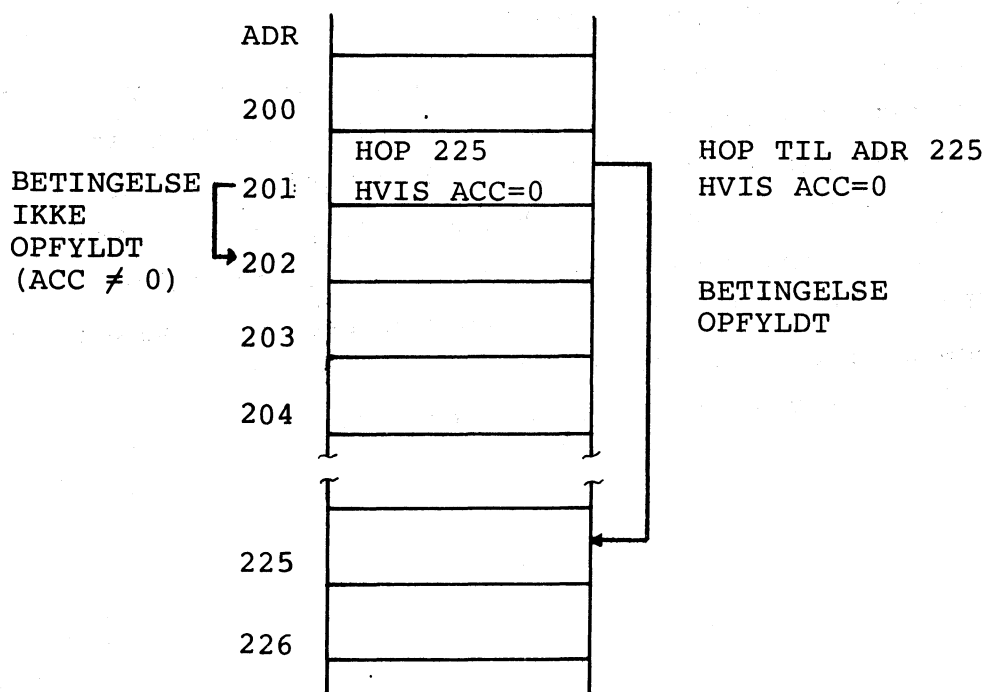
Se endvidere afsnit 7.2.3 Underprogrammer.



UBETINGET HOP:



BETINGET HOP:



## 5. Ind-/Udlæsning af data.

Udover de allerede nævnte instruktionstyper, må der eksistere instruktioner, der bevirker indlæsning af data fra ydre enheder, og instruktionstyper der bevirker udlæsning til ydre enheder. Disse instruktionstyper kan i stor udstrækning sammenlignes med de instruktioner, der bevirker flytning af data. I visse maskiner er det da også helt de samme instruktioner, der benyttes til ind- og udlæsning, som til flytning af data. (Mapped I/O).

## 6. Maskinstyring.

Til sidst skal nævnes instruktioner, som har direkte indflydelse på datamaskinens funktion, samt dem vi kalder hjælpeoperationer.

Maskinstyring kan f.eks. være en stopinstruktion, der bevirker, at programmet afbrydes, og først kan komme igang igen ved indgreb udefra. Det kan endvidere være til- og frakobling af signaler udefra. F.eks. interrupts. Se kap.4.4, Interrupt.

## 7. Hjælpeoperationer.

Som hjælpeoperationer kan betragtes instruktioner, der bevirker, at indholdet af et register parallelforskydes mod venstre eller højre (Shift), eller at indholdet roteres (Rotate), d.v.s. parallelforskydes, men hvor forreste bit er koblet sammen med bageste bit i registret. Endvidere eksisterer der ofte en instruktion, som ingenting udfører (No Operation). Denne kan være nyttig, hvis man i et program blot skal have en vis tid til at forløbe, inden der gøres videre (Dette kan ofte være tilfældet i styringsopgaver).

### 7.1.2. Operandtyper.

Ligesom der findes forskellige operationstyper, således findes der også forskellige operandtyper såsom:

1. Registeradresser.

## 2. Lageradresser.

## 3. Ind- /Uadresser.

## 4. Immediate data.

### 1. Registeradresser.

Med hensyn til registeradresser vil man kunne finde maskiner, hvor registeradressen er indbygget i operationskoden (implicit adressering). I andre maskiner er den en del af operanden (explicit adressering).

Da antallet af registre i en datamaskine sjældent overstiger et 4-cifret binært tal, vil en operand i sådanne tilfælde kunne indeholde flere adresser.

### 2. Lageradresser.

Ser vi på adressering i lageret, så kan dette foregå på flere forskellige måder, hvorfor der er afsat et helt afsnit senere til at beskrive dette fænomen. Se afsnit 7.1.3.

### 3. Ind- /Uadresser.

De ydre enheder har, ligesom registre og lagerceller, deres egne adresser. Antallet af ydre enheder, der kan kobles til en datamaskine, er derfor ofte bestemt af det antal adresser, der kan opereres på i operanddelen af en I/U-instruktion.

### 4. Immediate data.

Immediate (med det samme) data er ikke adresser, men direkte de data styreenheden skal arbejde med. Operanddelen er i disse tilfælde data og ikke adresser.

## 7.1.3 Adresseringsformer.

Som tidligere nævnt, er adressering af lagerceller, et helt kapitel for sig selv.

Adressering af lagerceller kan foretages på forskellig måde, ofte afhængigt af maskintype.

De almindeligst forekommende adresseringsformer skal nævnes her:

1. Direkte adressering.
2. Udvidet adressering.
3. Relativ adressering.
4. Indirekte adressering.
5. Indekseret adressering.
  - a. Pre-modificering.
  - b. Post-modificering.

#### 1. Direkte adressering.

Ved direkte adressering forstås, at operanden direkte peger på den lagercelle, styreenheden skal arbejde med. Hvis man har en operanddel på 10 bit, kan man direkte adressere  $2^{10} = 1024$  forskellige lagerceller. Hvis man imidlertid har en datamaskine på 64K, er dette ikke meget nytte til. Man opdeler derfor lageret i sider (Pages) á 1024 lagerceller.

Programtællerens værdi kan da benyttes til at bestemme, hvilken side der i det givne øjeblik arbejdes med. Denne side kaldes løbende side (Current Page).

Dette begrænser imidlertid også adresseringsmulighederne, så derfor kan man benytte en bit af enten operationskoden eller operanden til at fortælle, hvorvidt der er tale om adressering af side 0 eller af løbende side. (Current page).

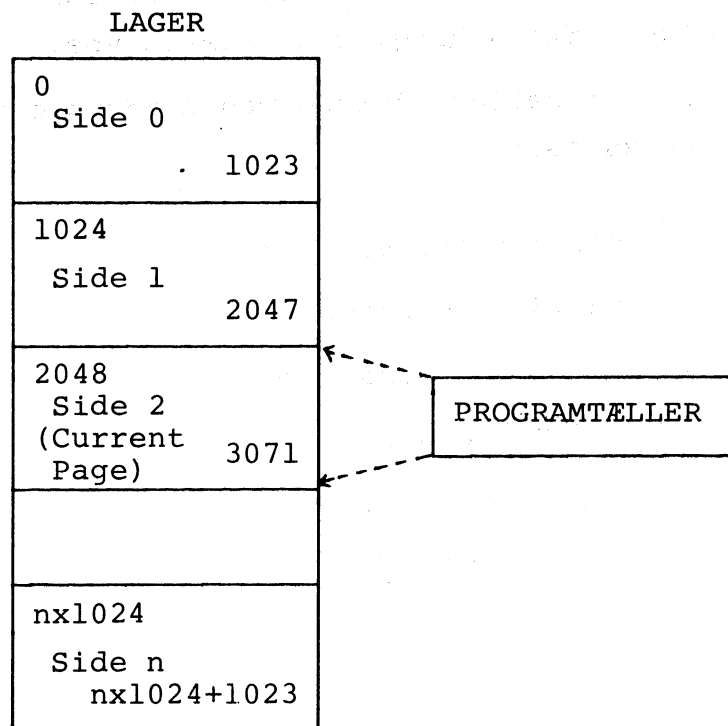


Fig. 7.2 Sideopdeling af datamaskinens lager.

Ved at benytte side 0 og indirekte adressering (omtales senere), kan man få adgang til en vilkårlig data-celle i lageret).

## 2. Udvidet adressering.

Udvidet adressering finder man ofte i maskiner med variabel instruktionslængde, hvor det er muligt at forlænge operanden til et bitantal, der kan adressere en vilkårlig celle i lageret.

## 3. Relativ adressering.

Relativ adressering er adressering i forhold til den værdi, der til enhver tid befinder sig i programtælleren. Adressen kan være positiv eller negativ i forhold til programtælleren. Princippet ligner direkte adressering af løbende side, blot med den forskel, at løbende sides under- og overgrænse ændres med værdien i programtælleren.

#### 4. Indirekte adressering.

Indirekte adressering er adressering af en celle, som peger på den reelle adresse. Anvendelsen af indirekte adressering finder meget anvendelse i de tilfælde, hvor man ikke direkte kan nå den ønskede adresse ved hjælp af direkte eller relativ adressering. Fordelen ved at benytte indirekte adressering er den, at alle bit i den indirekte adresse kan benyttes til adressering. Ved hjælp af indirekte adressering via side 0, er det muligt at adressere data eller instruktioner placeret på en vilkårlig side fra en vilkårlig anden side.

#### 5. Indekseret adressering.

Indekseret adressering findes ikke på alle datamaskiner. Indeksering kræver, at et eller flere registre i styreenheden kan benyttes som adresseringsregister. Den indekserede adresse forekommer ved, at styreenheden tager indholdet af operanden og hertil adderer indholdet af indeksregisteret. Den herved fremkomne adresse, er den adresse styreenheden vil arbejde på. Indeksadressering er velegnet i forbindelse med arbejde med tabeller, idet adressen på næste element i en tabel, kan fremkomme ved at ændre indholdet af indeksregisteret.

I visse datamaskiner kan denne ændring af indeksregisteret foregå automatisk. Denne kan f.eks. forekomme ved at en del af eller hele operanden benyttes, som den værdi, der lægges til indeksregisteret ved indekseret adressering som modifikation.

Hvis en sådan modifikation foretages før der adresseres, så kaldes det pre-indeksering eller pre-modifikation.

Forekommer modifikationen efter adresseringen, så kaldes det post-indeksering eller post-modifikation.

#### 7.1.4 Instruktionsformater.

Som nævnt tidligere, skelnes der mellem de datamaskiner, hvis instruktionslængde er fast, og de, hvis instruktionslængde er variabel. Instruktionsformatet for sådanne forskellige typer datamaskiner, må derfor være forskellige.

Vi har endvidere set, at en instruktion består af henholdsvis operationskode og operand. Det er imidlertid de færreste datamaskiner, der er indrettet således, at operationskoden er at finde i et bestemt antal bit, og at ingen andre bit i instruktionen er ubetydende for, hvorledes instruktionen fortolkes.

Almindeligvis skelner man mellem instruktioner, der adresserer forskellige elementer i datamaskinen såsom:

1. Register reference instruktioner.
2. Lager reference instruktioner.
3. Ind- / Udlæseinstruktioner.

Selvom man laver denne opdeling, vil det imidlertid ofte være således, at man ser instruktioner, som indeholder en blanding af disse referencer.

Eksempelvis er det nødvendigt, at kunne referere til såvel et register som en lagercelle i forbindelse med flytning af data fra lageret og til et register og omvendt. Det er i visse datamaskiner også nødvendigt at referere til såvel et register som en adresse på en ydre enhed ved ind- og udlæsning, ligesom visse systemer kan læse ind og ud af datalageret direkte (D.M.A. = Direct Memory Access).

Det er derfor ikke altid muligt, at klassificere en given instruktion entydigt i en af de tre nævnte grupper.

På den efterfølgende side er vist et eksempel på gruppering af forskellige instruktionstyper.

GRUNDLÆGGENDE MASKININSTRUKTIONER	LAGER REF. INSTRUKTIONER	REGISTER REF. INSTRUKTIONER	IND/UD INSTRUKTIONER
FLYTNING	LOAD STORE	LOAD STORE	READ WRITE
ARITHMETISKE	ADD	INCREMENT ROTATE ADD	
LOGISKE (BIT FOR BIT)	AND OR XOR	COMPLEMENT	
PROGRAM KONTROL	JUMP JUMP SUBROUTINE JUMP ON CONDITION INCREMENT, SKIP IF ZERO	HALT NO OPERATION JUMP ON CONDITION INCREMENT, SKIP IF ZERO CLEAR FLAG SET FLAG	SKIP IF FLAG SET CLEAR FLAG CLEAR CONTROL SET FLAG SET CONTROL
GRUPPERING AF INSTRUKTIONSTYPER			



## 7.2 Elementær programmering.

### 7.2.1 Instruktionsafviklingsrækkefølge.

Det er tidligere nævnt, at en datamaskine i princippet arbejder sekventielt. D.v.s. at den udfører instruktionerne i rækkefølge, således som de er placeret i lageret. Vi har endvidere set, at styreenheden er udrustet med et specielt register - programtælleren (Program Counter), hvis opgave er, at holde styr på, hvor næste instruktion er placeret i lageret.

Under den almindelige instruktionsafvikling vil ajourføring af programtælleren foregå automatisk, hvad enten datamaskinen arbejder med fast eller variabel instruktionslængde.

Når et program startes op, er det nødvendigt, at programtælleren initialiseres med adressen på den første instruktion i programmet.

Hvis vi forestiller os, at vi ønsker at flytte indholdet af 5 dataceller fra et sted i datalageret adr.  $100_8$ , til et andet sted i datalageret, adr.  $200_8$ , og at vi skal bruge datamaskinen fra kap. 3 så vil følgende instruktionstyper være nødvendige:

1. Hent til XR (YR), fra ADR.
2. Flyt XR (YR), til AC.
3. Gem AC i ADR.
4. Stop

Maskininstruktionerne for ovennævnte instruktionstyper er:

	Operations- kode	Operand
1.	<div>1512110 0110</div>	Hent til XR fra ADR
2.	<div>0010010000000000</div>	Flyt XR til AC.
3.	<div>0100</div>	Gem AC i ADR.
4.	<div>0000000000000000</div>	Stop

Hvis vi placerer programmet startende fra og med adresse 300<sub>8</sub>, vil programmet komme til at se således ud:

Lager adresse oktal	Indhold: maskinkode oktal	
300 <sub>8</sub>	060100	Hent til XR fra adr.100
301	022000	Flyt XR til AC
302	040200	Gem AC i adr 200
303	060101	Hent til XR fra adr.101
304	022000	Flyt XR til AC
305	040201	Gem AC i adr. 201
⋮	⋮	⋮
314	060104	Hent XR fra adr. 104
315	022000	Flyt XR til AC
316	040204	Gem AC i adr. 204
317	000000	Stop datamaskinen

Følgende flytning vil ske:

ADR:	INDHOLD		ADR:	INDHOLD
100	"A" "B"	➔	200	
101	"C" "D"		201	
102	"E" "F"		202	
103	"G" "H"		203	
104	"I" "J"		204	

Når programmet er gennemløbet, vil indholdet af datacellerne 100 til 104 være flyttet til (kopieret) datacellerne 200 til 204.

### 7.2.2 Sløjfeprogrammering.

Når man i programmeringen støder på opgaver, hvor man ønsker at gentage sammensatte operationer, som vist i foregående afsnit, er det praktisk at udføre operationerne i en såkaldt sløjfe (Loop).

Ved sløjfeprogrammering er der tre ting, man skal være opmærksom på:

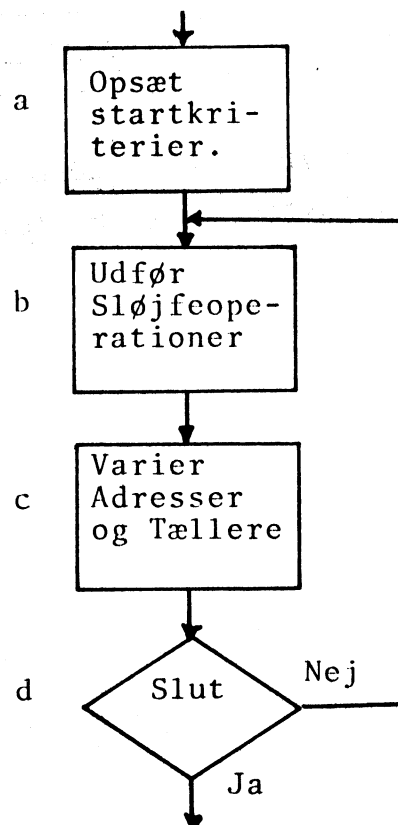
1. Startkriterier for sløjfen (a).
2. Variation af kriterier ved sløjfegennemløb (c).
3. Slutkriterier for sløjfen (d).

Startkriterier vedrører elementer, som startadresser og startværdier for tællere, som benyttes i forbindelse med bestemmelse af antal gange, en sløjfe skal gennemløbe.

Variationerne er de værdier, som adresser og tællere skal ændres med for hvert sløjfegennemløb.

Slutkriterium er den tilstand, der skal bestemme, hvornår gennemløb af sløjfen er tilendebragt.

Principblokdiagram for sløjfeprogrammering:

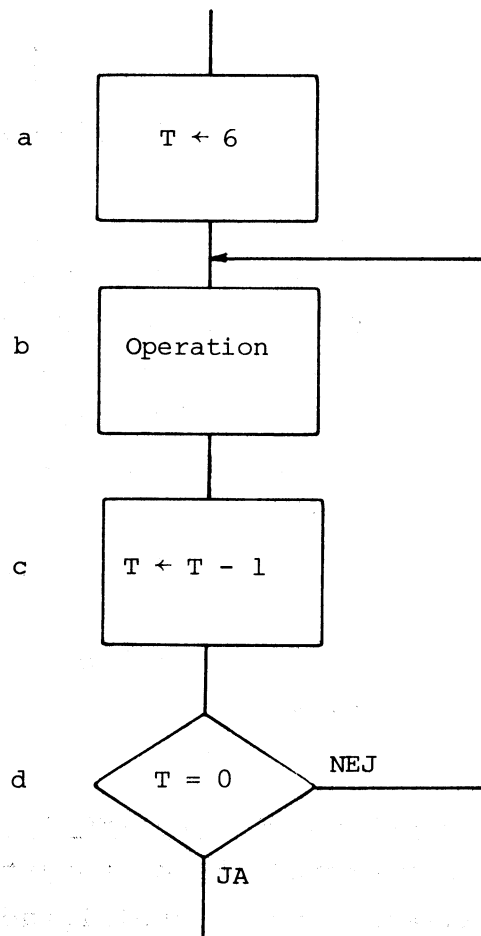


### Eksempel på simpel sløjfeprogrammering.

Som et eksempel på anvendelse af sløjfeprogrammering, er vist et program, hvor en bestemt operation ønskes udført et kendt antal gange, f.eks. 6 gange. Antallet af gange bestemmes ved hjælp af en tæller, hvis værdi kan ændres.

Til programeksemplet er anvendt instruktioner fra datamaskinen kap. 3.

Blokdiagrammet for forløbet kan se således ud:



T, er en lageradresse som bruges til tæller.

For at kunne bestemme om slutkriteriet (d.) for sløjfen er opfyldt, må anvendes en programstyringsinstruktion som f.eks. betinget hop, hvis ACC=0

PROGRAM		
Indhold:		
Lager adresser oktal	Maskinkode oktal	
150	060200	Hent til XR fra ADR 200
151	022000	Flyt XR til AC
152	040201	Gem AC i Adr 201
153	⋮	⋮
⋮	⋮	⋮
⋮	⋮	⋮
⋮	⋮	Sløjfe operation
⋮	⋮	⋮
180	070201	Dekrement adr. 201
181	060201	Hent til XR fra adr 201
182	022000	Flyt XR til AC
183	150185	Hop til adr 185 hvis AC=0 <i>relateret hop</i>
184	140153	Hop til adr 153 <i>ubetinget hop</i>
185	xxxxxx	
⋮	⋮	Program fortsat
⋮	⋮	
200	000006	Tæller konstant
201	xxxxxx	

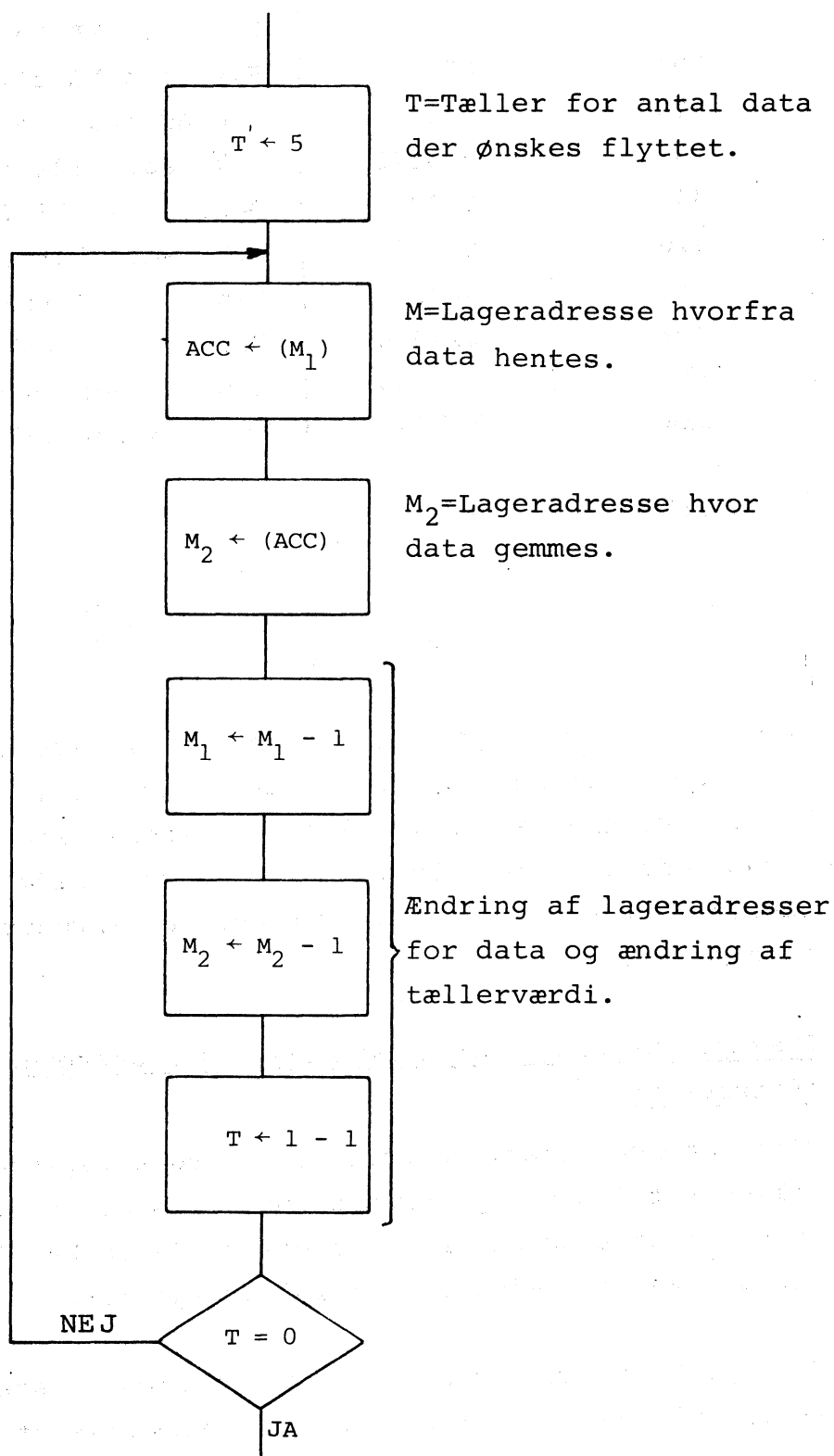
#### Eksempel på sløjfeprogrammering ved indirekte adressering.

Hvis man ved programkonstruktion udnytter en kombination af sløjfeprogrammering og forskellige adresseringsformer, er det ofte muligt at lave et program der optager mindre plads i lageret.

Som eksempel kan bruges det program som vi udarbejdede i afsnit 7.2.1 til at flytte data.

Anvender vi til dette formål en kombination af sløjfeprogrammering og indirekte adressering kræver det først og fremmest en modifikation af instruktionssættet for

datamaskinen i kap. 3, da dennes operand er en direkte adresse; det kan f.eks. gøres ved at ændre maskinen således, at bit 11 = "0" opfattes som direkte adresse. Sættes bit 11 = "1" opfattes adressen som indirekte. Programmet kan f.eks. se således ud:



# PROGRAM

## Indhold:

Lager adresse oktal	Maskinkode oktal	
300	060320	Hent til XR fra adr. 320
301	022000	Flyt fra XR til AC
302	040321	Gem AC i Adr. 321
303	064322	Hent til XR Ind. via adr. 322
304	022000	Flyt fra XR til AC
305	044323	Gem AC ind. via adr. 323
306	070322	Dek. adr. 322 (M1)
307	070323	Dek. adr. 323 (M2)
310	070321	Dek. adr. 321 (T.)
311	060321	Hent til XR fra adr. 321
312	022000	Flyt fra XR til AC
313	150350	Hop til adr. 350 hvis AC=0
314	140303	Hop til adr. 303
⋮	⋮	
320	000005	Konstant (Antal)
321	xxxxxx	tæller
322	000104	Hent DATA ADR
323	000204	Gem DATA ADR

Som det ses af programmet er det tæller-værdien, der bestemmer hvor mange data der skal flyttes i lageret, hvorfor programmets størrelse ikke ændrer sig med den datamængde, der skal flyttes, som det var tilfældet ved eksemplet i afsnit 7.2.1, hvor der var anvendt direkte adressering.

### Eksempel på sløjfeprogrammering ved indekseret adressering.

Visse datamaskiner, indeholder et indeksregister, som er velegnet til sløjfeprogrammering, hvis der i sløjfegennemløbet er tale om at ændre adresser. Dette kan i programmet ske ved at referere til indeksregisteret.

Eksemplet med at flytte data fra et sted i lageret til et andet sted i lageret, kan ved anvendelse af indekseret adressering og sløjfeprogrammering se således ud:

## PROGRAM

### Lageradresse:

### Indhold:

300	Sæt tæller lig 5
301	Sæt indeksregisteret til 100
302	Hent til AC indekseret + 0 fra lageret.
303	Gem AC indekseret + 100 i lageret
304	Forøg indeksregisteret med 1
305	Træk een fra tællerens værdi.
306	Hop hvis tæller = 0 til Adr.310
307	Hop til Adr. 302 ubetinget.

Anvendes indekseret adressering, vil dette selvfølgelig have indflydelse på datamaskinens maskinkode.

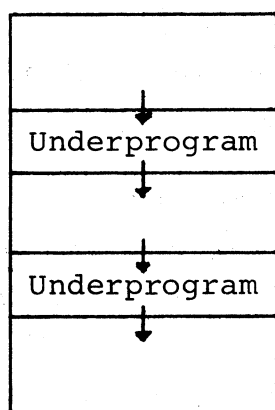
### 7.2.3 Underprogram.

Et underprogram anvendes ofte ved programkonstruktion, når en instruktionssekvens skal gentages. Dette kan gøres ved at kopiere instruktionssekvensen hver gang den skal anvendes. En sådan programdel kaldes et åbent underprogram. Se fig. 7.2 side 7.23

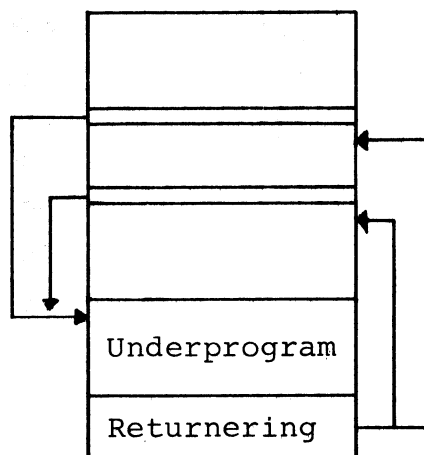
Er der imidlertid tale om lange sekvenser og/eller mange kopieringer er det lagerpladsbesparende at placere programdelen et sted i programforløbet. Dette kræver imidlertid, at man har mulighed for at fastholde til hvilket punkt i programmet der skal vendes tilbage, når programdelen er udført. En sådan programdel kaldes et lukket underprogram. Se fig- 7.2 side 7.23.



Fig. 7.2



Åbent underprogram



Lukket underprogram

Som det ses af fig. 7.2 optager underprogrammet plads hver gang det skal benyttes som åbent underprogram. Som lukket underprogram optager selve programdelen kun plads en gang. Til gengæld må der udføres ekstra instruktioner for at komme til og fra et lukket underprogram.

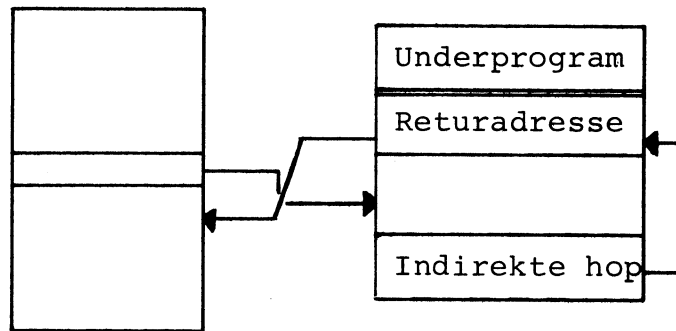
Vurderingen af, om der skal benyttes åbne eller lukkede underprogrammer, har også noget med tid at gøre. Anvendelsen af lukkede underprogrammer tager længere tid, fordi der hver gang de benyttes bruges tid til at komme til og fra underprogrammet. Dette kan være afgørende i forbindelse med tidskritiske programafviklinger (f.eks. processtyring).

Ikke alle datamaskiner benytter samme teknik i forbindelse med afvikling af lukkede underprogrammer. Her skal omtales to karakteristiske principper, som vi ofte vil finde i mindre datamaskiner.

Det ene princip bygger på det, at underprogrammet er placeret i et læse-/skrivelager (f.eks. ferritkernelager), således at returadressen kan placeres i underprogrammet. Det er her almindeligt, at returadressen placeres i den første celle i underprogrammet, og den første egentlige instruktion i celle nr. to. Returneringen vil da foregå

som et indirekte returhop via første celle i underprogrammet.

Fig. 7.3

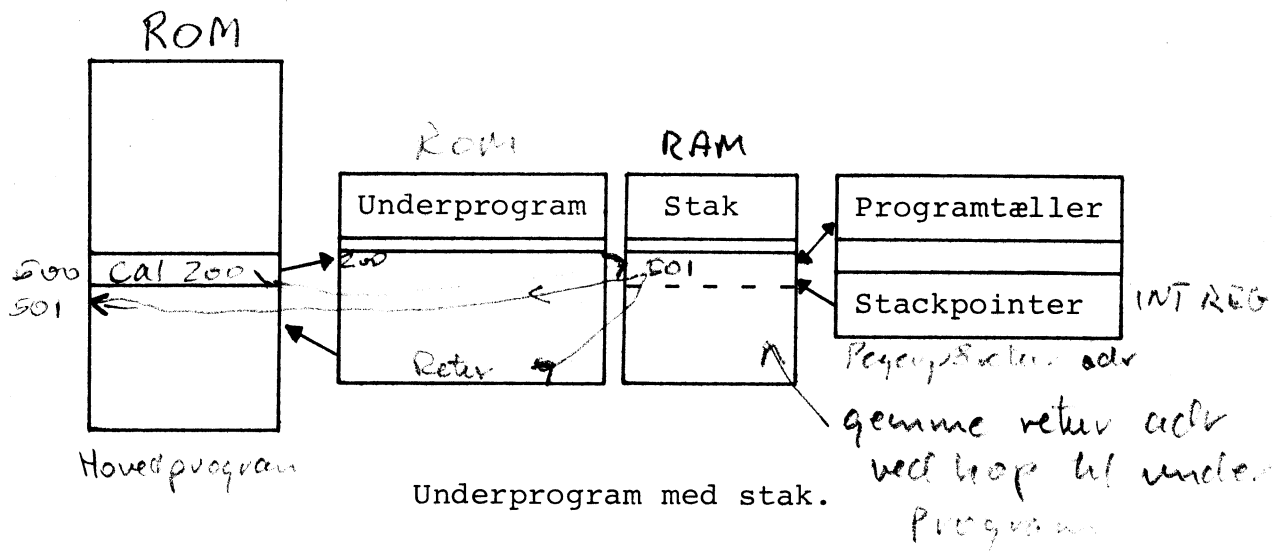


Underprogram med indirekte returhop.

Det andet princip benyttes i forbindelse med datamaskiner, hvor programmet er placeret i et læselager (ROM = Read Only Memory). Princippet er anvendeligt, dersom styreenheden er forsynet med et register, der kan pege på en ledig celle i et læse-/skrivelager (Stackpointer).

Cellen i læse-/skrivelageret benyttes til opbevaring af returhopadressen.

Fig. 7.4



Opgave 7.1

Programmering i maskinsprog

Lav i maskinsprog et program, der ombytter indholdet af to lager-  
adresser, henholdsvis adresse  $200_8$  og  $201_8$ .

Programmet placeres startende i adresse  $100_8$ .

# DIAGRAMMERING

Udfyldt af <b>L.B.</b>	Udfyldt den <b>290777</b>	Opgavenr <b>7.1</b>	Bilagnr	Sidenr <b>1/2</b>
Processens navn / Nr. <b>Ombyt to tal</b>				
1	2	3	4	5

A

B

C

D

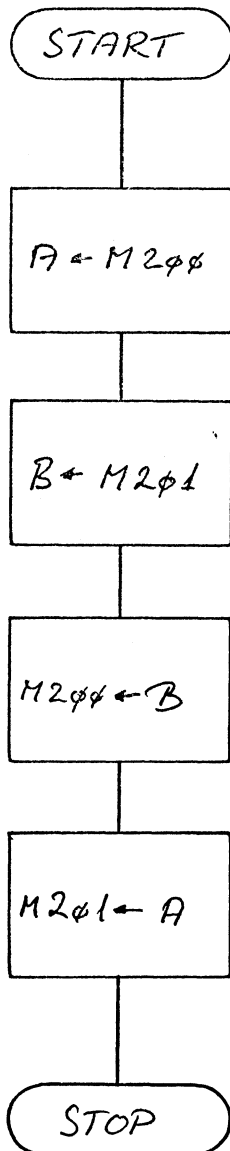
E

F

G

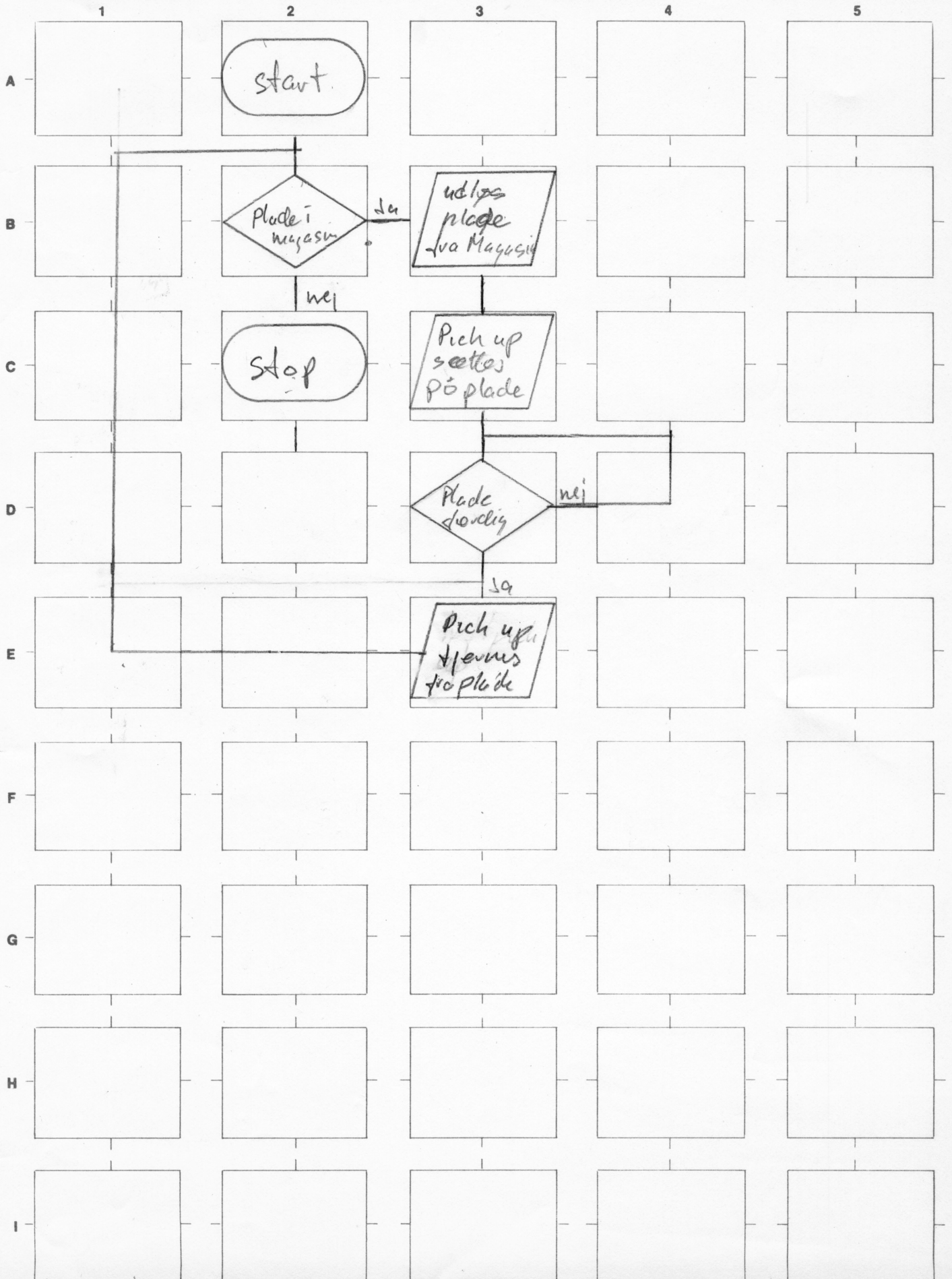
H

I



# DIAGRAMMERING

Udfyldt af	Udfyldt den	Opgavenr	Bilagnr	Sidenr
Processens navn / Nr.				



AUTOMATISERINGSTEKNIK

TEKNOLOGISK INSTITUT KØBENHAVN



Opgave 7.2

Programmering i maskinsprog

Lav i maskinsprog et program der kan udføre regneoperationen:

$$Q = X + Y - Z$$

Værdierne for X, Y og Z placeres i lageret.

Resultatet Q, udlæses på switch-registret (switch-registret adresseres som et I/o register)

Programmet placeres startende i adresse:

Hold 1    adresse    100<sub>8</sub>

Hold 2    -            200<sub>8</sub>

Hold 3    -            300<sub>8</sub>

Hold 4    -            400<sub>8</sub>      Hold 4

1    Blokdiagram

2    Kodning

3    Test

4    Retning    eventuel    ha ha

# DIAGRAMMERING

Udfyldt af <b>L.B.</b>	Udfyldt den <b>29-7-77</b>	Opgavenr <b>7.2.</b>	Bilagnr	Sidenr <b>1/2</b>
Processens navn / Nr. <b>Q = X + Y ÷ 2</b>				

1

2

3

4

5

A

START

B

$A \leftarrow X$

C

$A \leftarrow X + Y$

D

$B \leftarrow Z$

E

$B \leftarrow \overline{B}$

F

$B \leftarrow B + 1$

G

$A \leftarrow A + B$

H

$SW \leftarrow A$

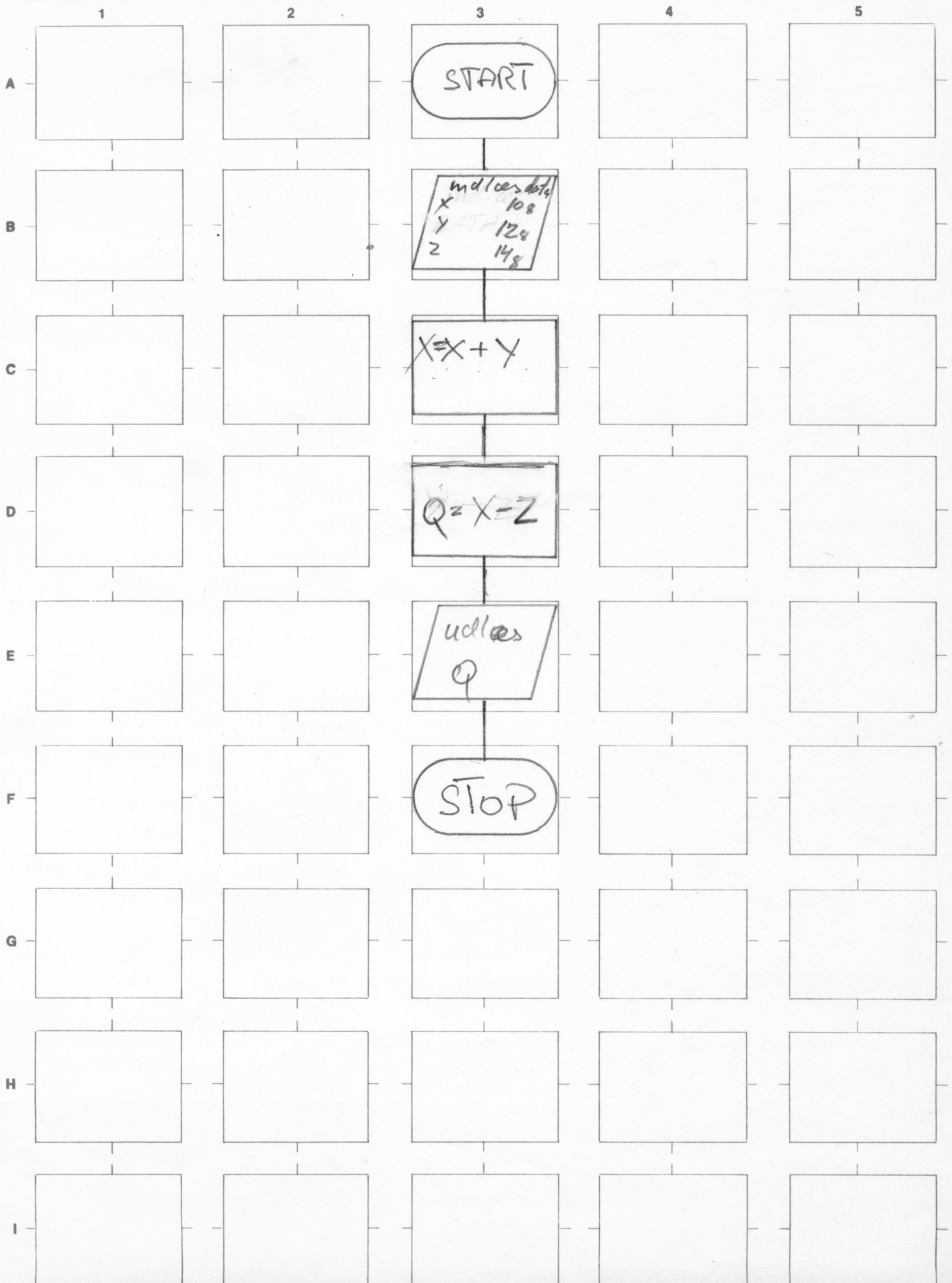
I

STOP



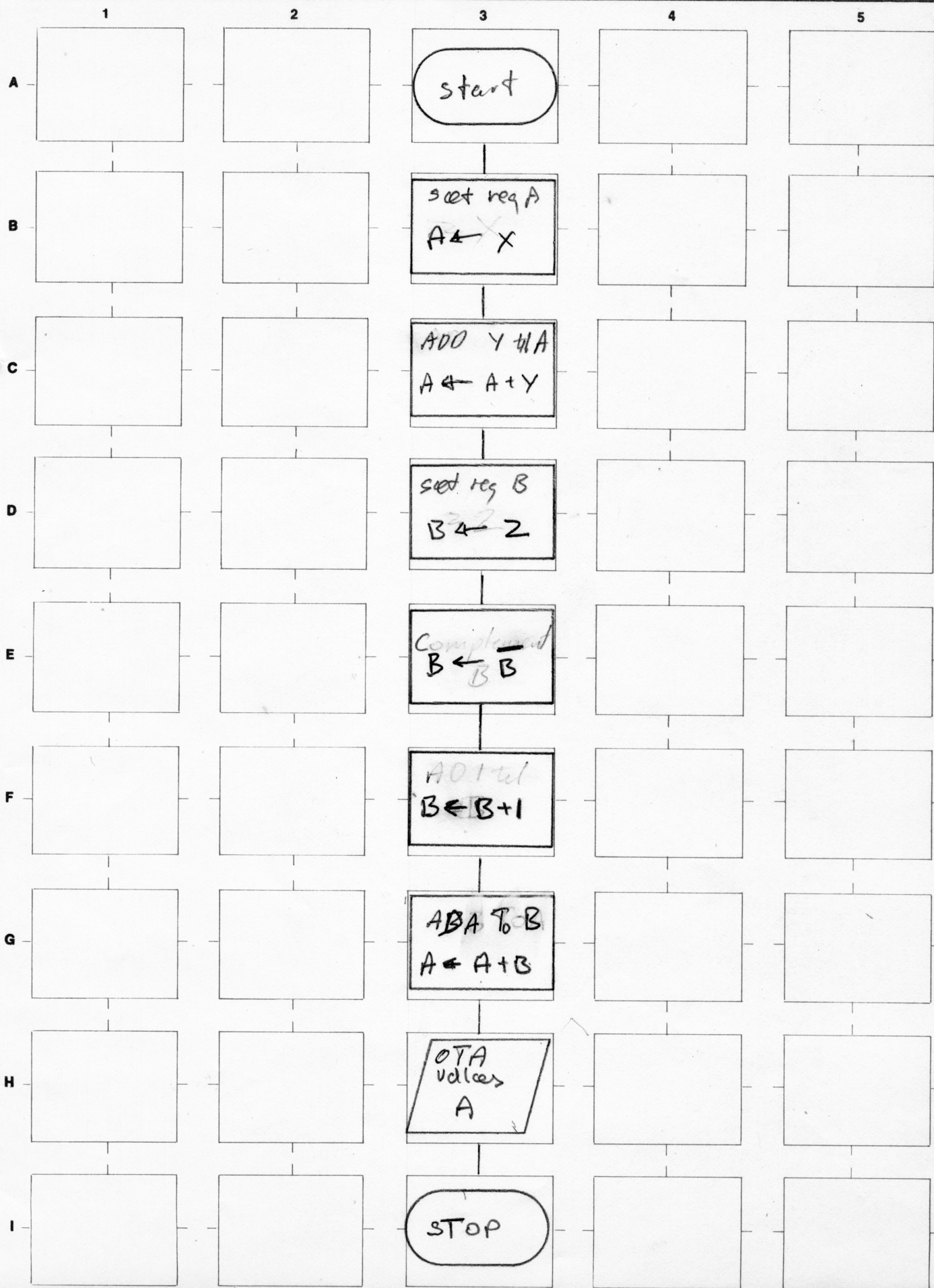
# DIAGRAMMERING

Udfyldt af <b>IHP</b>	Udfyldt den <b>19780804</b>	Opgaven <b>7.2</b>	Bilag nr <b>1 forspg</b>	Sidenr
Processens navn / Nr.				



# DIAGRAMMERING

Udfyldt af <b>WHP</b>	Udfyldt den <b>1978 08 04</b>	Opgavenr	Bilag nr <b>2 forsøg</b>	Sidenr
Processens navn / Nr.				



[illegible]



[illegible]

[illegible]





Opgave 7.3

Programmering i maskinsprog

Lav i maskinsprog et program, der optæller antal af 1-bit, som er indeholdt i et ord, indlæst ved hjælp af switch-registret.

Resultatet placeres i B-registret.

Programmet placeres startende i adresse:

Hold 1 adresse  $100_8$

Hold 2 -  $200_8$

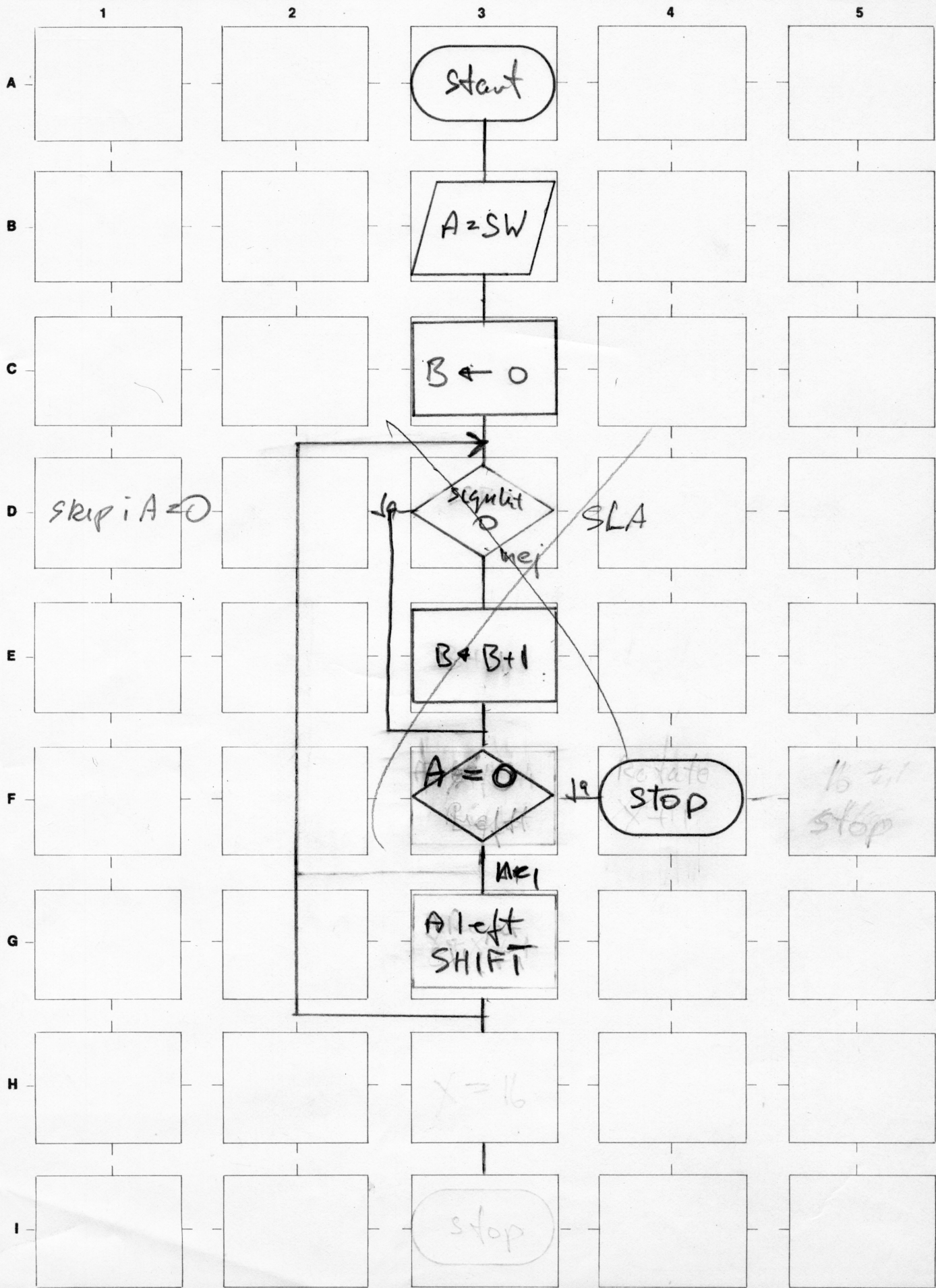
Hold 3 -  $300_8$

Hold 4 -  $400_8$

Hold

## DIAGRAMMING

Udfyldt af <b>IMP</b>	Udfyldt den <b>19780804</b>	Opgavenr <b>7.3</b>	Bilag nr <b>1 forsøg</b>	Sidenr
Processens navn / Nr.				





PAGE 0002 #01

```
0001          ASMB,A,B,L,T
0002* PROGRAM, DEP TAELLER ANTAL 1-BIT
0003 10000          ORG 10000B  STARTADRESSE
0004 10000 006400  START CLB      B=0
0005 10001 062014          LDA KONST  A=-16
0006 10002 072015          STA T      T=-16
0007 10003 102000          HLT
0008 10004 102501          LIA INPUT  INPUT BITMØNSTER
0009 10005 000010  IGEN  SLA      LSB=1
0010 10006 006004          INB      JA: B=B+1
0011 10007 001200          RAL      NEJ: ROTER TIL VENST.
0012 10010 036015          ISZ T    T=T+1; T=0
0013 10011 026005          JMP IGEN  NEJ
0014 10012 106601          OTB OUPUT JA: OUTPUT RESULTAT
0015 10013 026000          JMP START FORFRA
0016 10014 177760  KONST DEC -16  ANTAL AF ROTATIONER
0017 10015 000000  T      BSS 1    LOKATION AF TAELLER
0018 00001          INPUT EQU 1    INPUT=S-REG
0019 00001          OUPUT EQU 1    OUTPUT=S-REG
0020          END
** NO ERRORS*
```

```

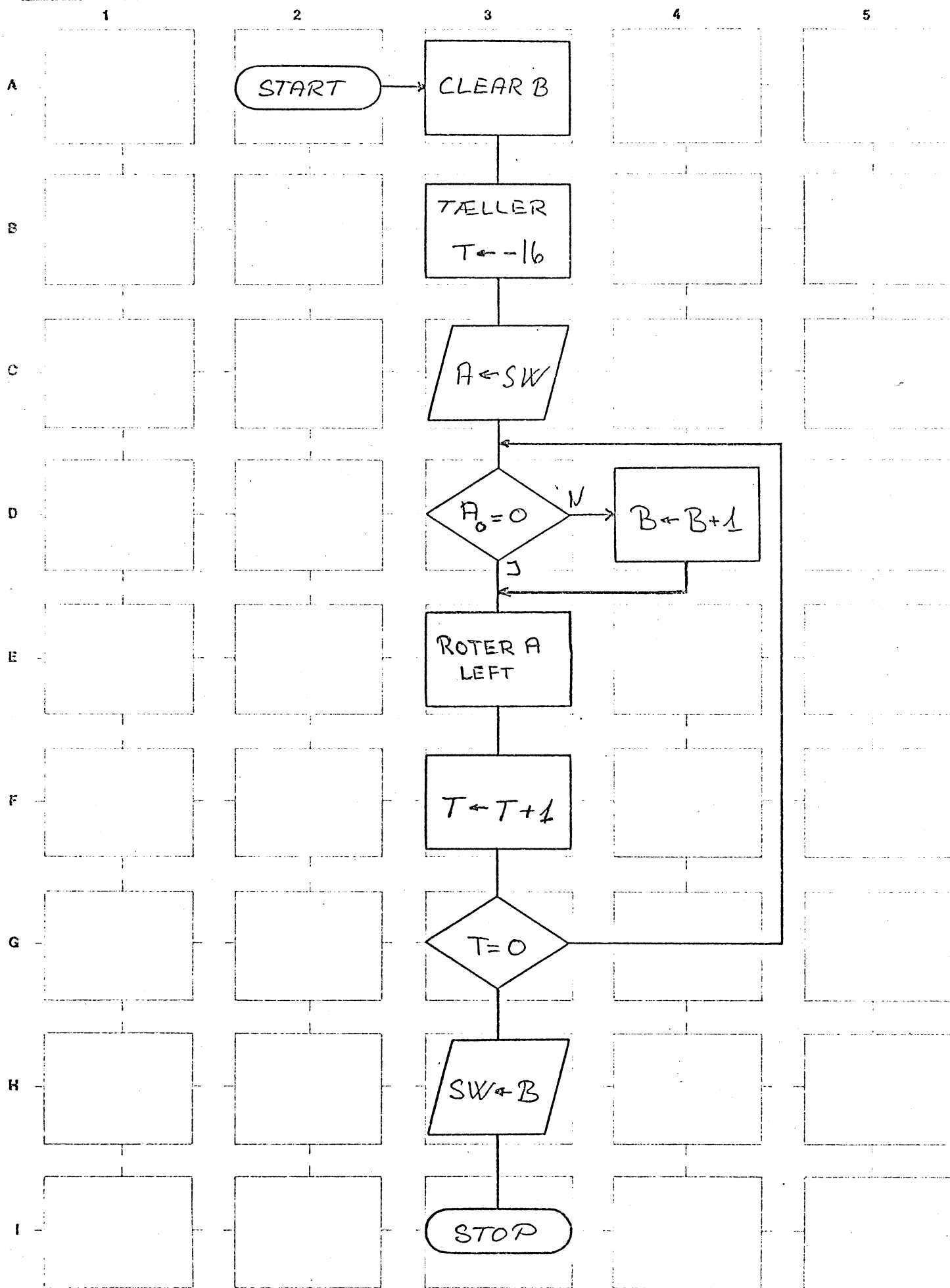
0001          ASMB,A,B,L,T
0002* PROGRAM, DER TAELLER ANTAL 1-BIT
0003 10000          ORG 10000B  STARTADRESSE
0004 10000 006400  START CLB      B=0
0005 10001 062014          LDA KONST  A=-16
0006 10002 072015          STA T      T=-16
0007 10003 102000          HLT
0008 10004 102501          LIA INPUT  INPUT BITMØNSTER
0009 10005 000010  IGEN  SLA      LSB=1
0010 10006 006004          INB      JA: B=B+1
0011 10007 001200          RAL      NEJ: RØTER TIL VENST.
0012 10010 036015          ISZ T     T=T+1; T=0
0013 10011 026005          JMP IGEN  NEJ
0014 10012 106601          OTB OUPUT JA: OUTPUT RESULTAT
0015 10013 026000          JMP START FORFRA
0016 10014 177760  KONST DEC -16  ANTAL AF ROTATIONER
0017 10015 000000  T      BSS 1   LOKATION AF TAELLER
0018 00001          INPUT EQU 1   INPUT=S-REG
0019 00001          OUPUT EQU 1   OUTPUT=S-REG
0020          END

```

\*\* NO EPRORS\*

# DIAGRAMMERING

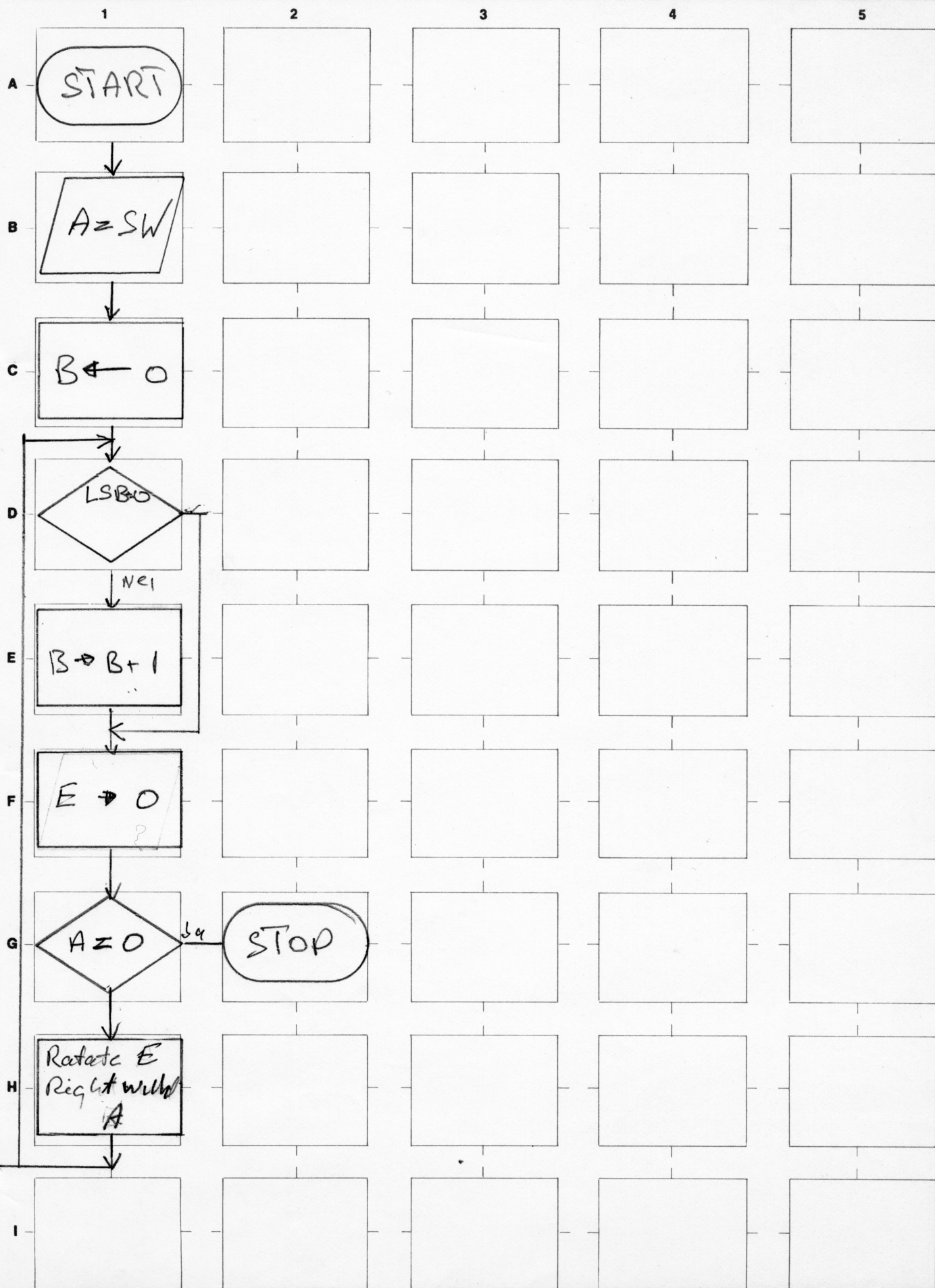
Udfyldt af <b>L.B.</b>	Udfyldt den <b>23-9-76</b>	Opgavenr <b>7.3</b>	Bilagnr	Sidenr
Processens navn / Nr. <b>TÆLLER ANTAL '1'-BIT I ET ORD.</b>				



# DIAGRAMMERING



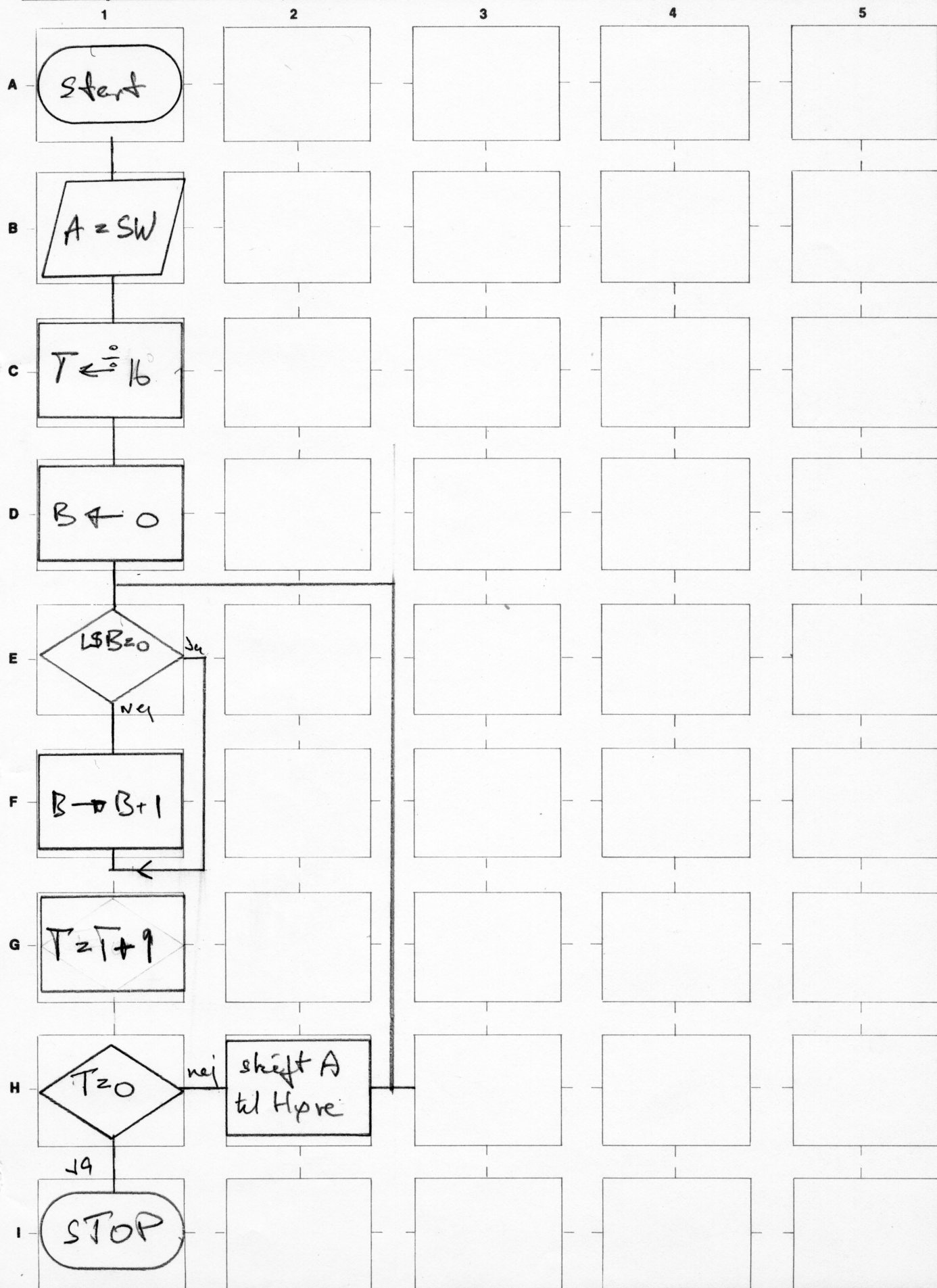
Udfyldt af <b>HLV</b>	Udfyldt den <b>19780808</b>	Opgavenr <b>7.3</b>	Bilag nr	Sidenr
Processens navn / Nr.				





# DIAGRAMMERING

Udfyldt af <b>IHP</b>	Udfyldt den <b>1978 08 08</b>	Opgavenr <b>7.3</b>	Bilagnr	Sidenr
Processens navn / Nr.				



[illegible]

Dato 5-10-76

Konstr./Tegn.af LB.

Team.nr.

Blod 2 af 2

5110g OP60VE 7.3

# Datamaskiner-ME

## Opgave 7.4

### Programmering i maskinsprog

Lav i maskinsprog, et program der addere 5 tal som placeres i lageret fra adresse  $10_{16}$ .

A-registret bruges til addition.

B-registret bruges til tæller.

Programmet placeres startende i adresse: ~~4000~~

Hold 1 adresse  $4000_{16}$

Hold 2 "  $4100_{16}$

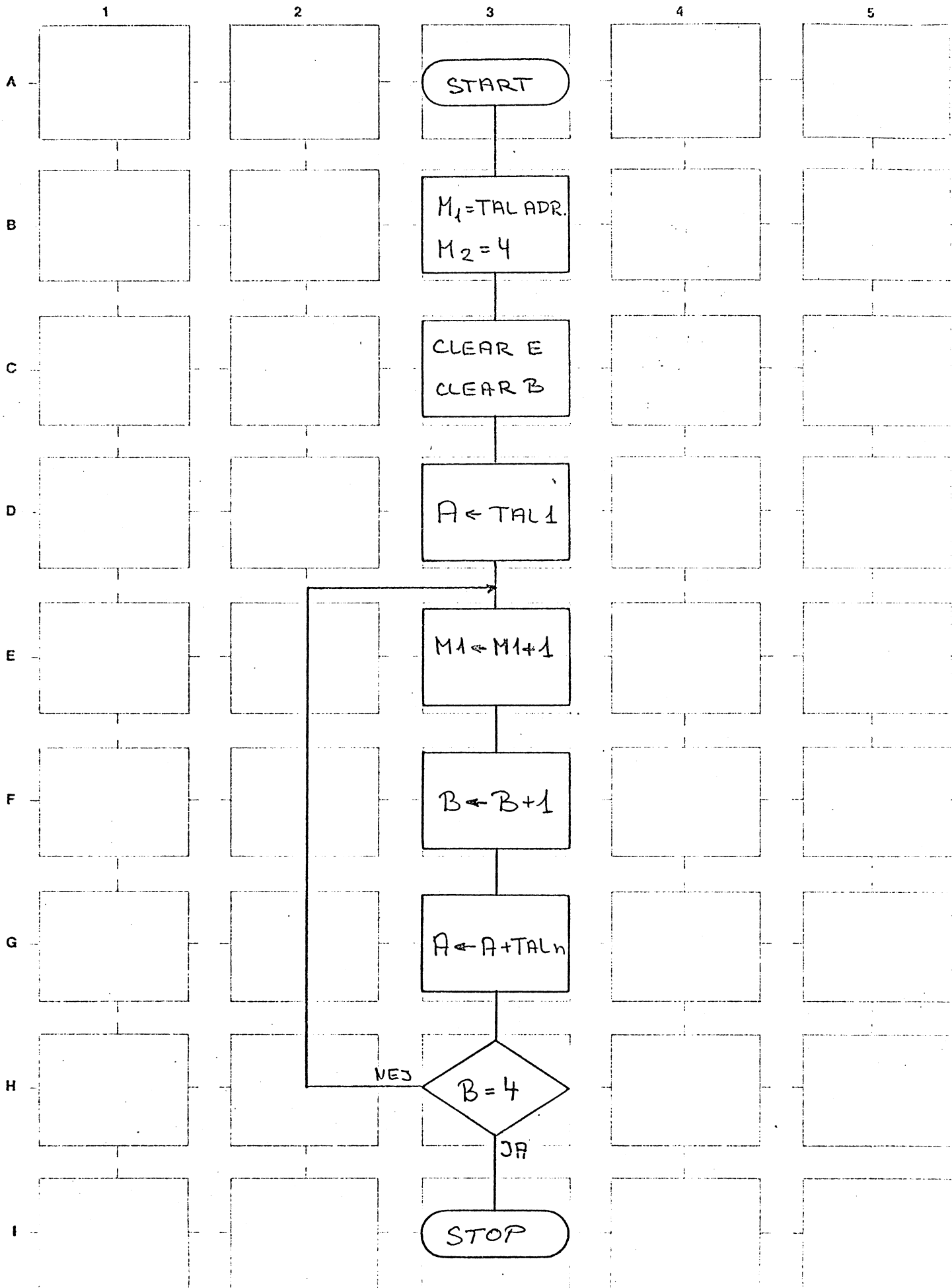
Hold 3 "  $4200_{16}$

Hold 4 "  $4300_{16}$

01000/000/000/000/000/01  
11111111111111111111

# DIAGRAMMERING

Udfyldt af <b>LB.</b>	Udfyldt den <b>1/3-77.</b>	Opgavens <b>7.4</b>	Bilag nr.	Sider nr. <b>1/2</b>
Processens navn / Nr.		<b>ADDER TAL FRA LAGERE</b>		





[illegible]

AUTOMATISERINGSTEKNIK



TEKNOLOGICKÝ INSTITUT KOLEJNIA

Date 1/3-77

Konstr./Tegn. af LB

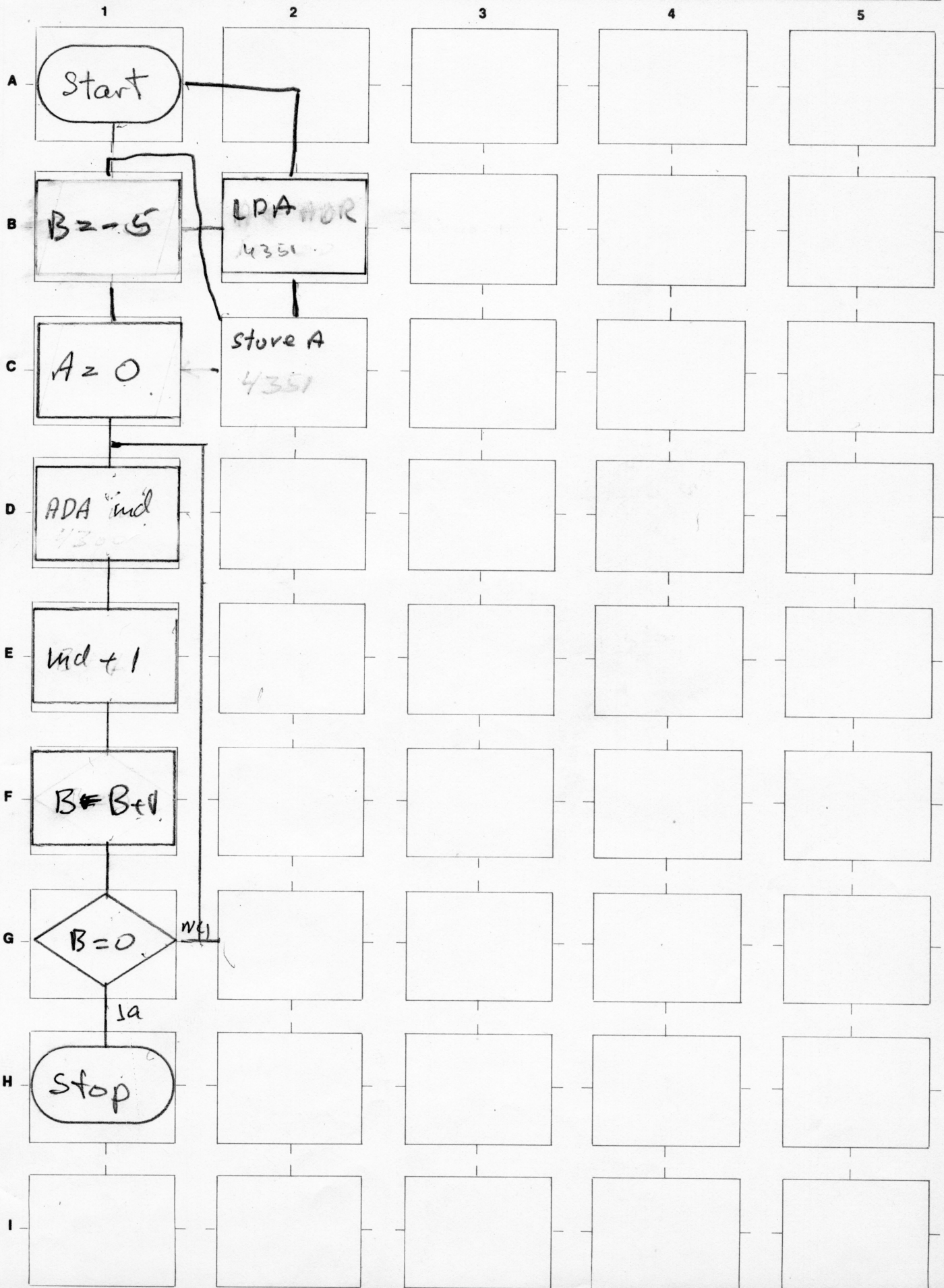
Tegn.nr.

Slad 2 af 2

# THE ZAGZAG BEIRUT

# DIAGRAMMERING

Udfyldt af <b>IMP</b>	Udfyldt den <b>19780808</b>	Opgavenr <b>7.4</b>	Bilagnr	Sidenr
Processens navn / Nr.				



[illegible]



[illegible]

DEMO-OEVELSE.

TÆLLER ANTAL AF '1'-BIT I ET ORD.

INPUT FRA SWITCH-REG.

OUTPUT TIL SWITCH-REG.

PAGE 0001

```

0001          ASMB, A, B, L, T
T            010015
IGEN        010005
INPUT       000001
KONST       010014
OUPUT       000001
START       010000
* NO ERRORS*

```

PAGE 0002 #01

```

0001          ASMB, A, B, L, T
0002* PROGRAM, DER TÆLLER ANTAL 1-BIT
0003 10000      ORG 10000B      STARTADRESSE
0004 10000 006400 START CLB      B=0
0005 10001 062014 LDA KONST      A=-16
0006 10002 072015 STA T          T=-16
0007 10003 102000 HLT
0008 10004 102501 LIA INPUT      INPUT BITMØNSTER
0009 10005 000010 IGEN SLA        LSB=1
0010 10006 006004 INB            JA: B=B+1
0011 10007 001200 RAL            NEJ: RØTER TIL VENST.
0012 10010 036015 ISZ T          T=T+1; T=0
0013 10011 026005 JMP IGEN       NEJ
0014 10012 106601 OTB OUPUT      JA: OUTPUT RESULTAT
0015 10013 026000 JMP START      FORFRA
0016 10014 177760 KONST DEC -16  ANTAL AF ROTATIONER
0017 10015 000000 T BSS 1        LOKATION AF TÆLLER
0018 00001      INPUT EQU 1      INPUT=S-REG
0019 00001      OUPUT EQU 1      OUTPUT=S-REG
0020          END

```

\*\* NO ERRORS\*

indhobler lokations tæller  
og mætter operand som  
lager adresse

reserverer plads i bogen her i plads





[illegible]

OBJEKT KODE		PROGRAM ASSEMB. OPGAVE 8.3																KOPIERING (PTR TIL PD)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
BINÆR		OCT		LABEL				MNE		OPERAND				COMMENT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
ADR	OCT.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							



Programmering i symbolsk maskinsprog.kopiering fra læser til punch.Opgave 8.3

PAGE 0001

```

0001          ASMB, A, B, L, T
PP           000014
PTR          000013
BUFF         010007
READ         010000
WRITE        010010
** NO ERRORS*

```

PAGE 0002 #01

```

0001          ASMB, A, B, L, T
0002 10000     ORG 10000B
0003 10000 103713 READ STC PTR, C
0004 10001 102313     SFS PTR
0005 10002 026001     JMP *-1
0006 10003 102513     LIA PTR
0007 10004 072007     STA BUFF
0008 10005 016010     JSB WRITE
0009 10006 026000     JMP READ
0010 10007 000000     BUFF BSS 1
0011 00013     PTR EQU 13B
0012 10010 000000     WRITE NOP
0013 10011 066007     LDB BUFF
0014 10012 106614     OTB PP
0015 10013 103714     STC PP, C
0016 10014 102314     SFS PP
0017 10015 026014     JMP *-1
0018 10016 126010     JMP WRITE, I
0019 00014     PP EQU 14B
0020          END
** NO ERRORS*

```

Opgave 8.3

Programmering i symbolsk maskinsprog.

Kopieringsprogram

Udarbejd et program, der er i stand til at kopiere fra strimmellæser til puncher.

Programmet udarbejdes efter følgende procedure:

1. Et hovedprogram læser ved "vente på flag" metoden en karakter fra strimmellæseren og lægger karakteren ind i lageret.
2. En subrutine udlæser ved "vente på flag" metoden karakteren til puncheren og derefter springes tilbage til hovedprogrammet.

Forslag til gruppeopdeling.

Gruppe 1.

Udarbejder hovedprogram

Gruppe 2.

Udarbejder subrutine

Gruppe 3.

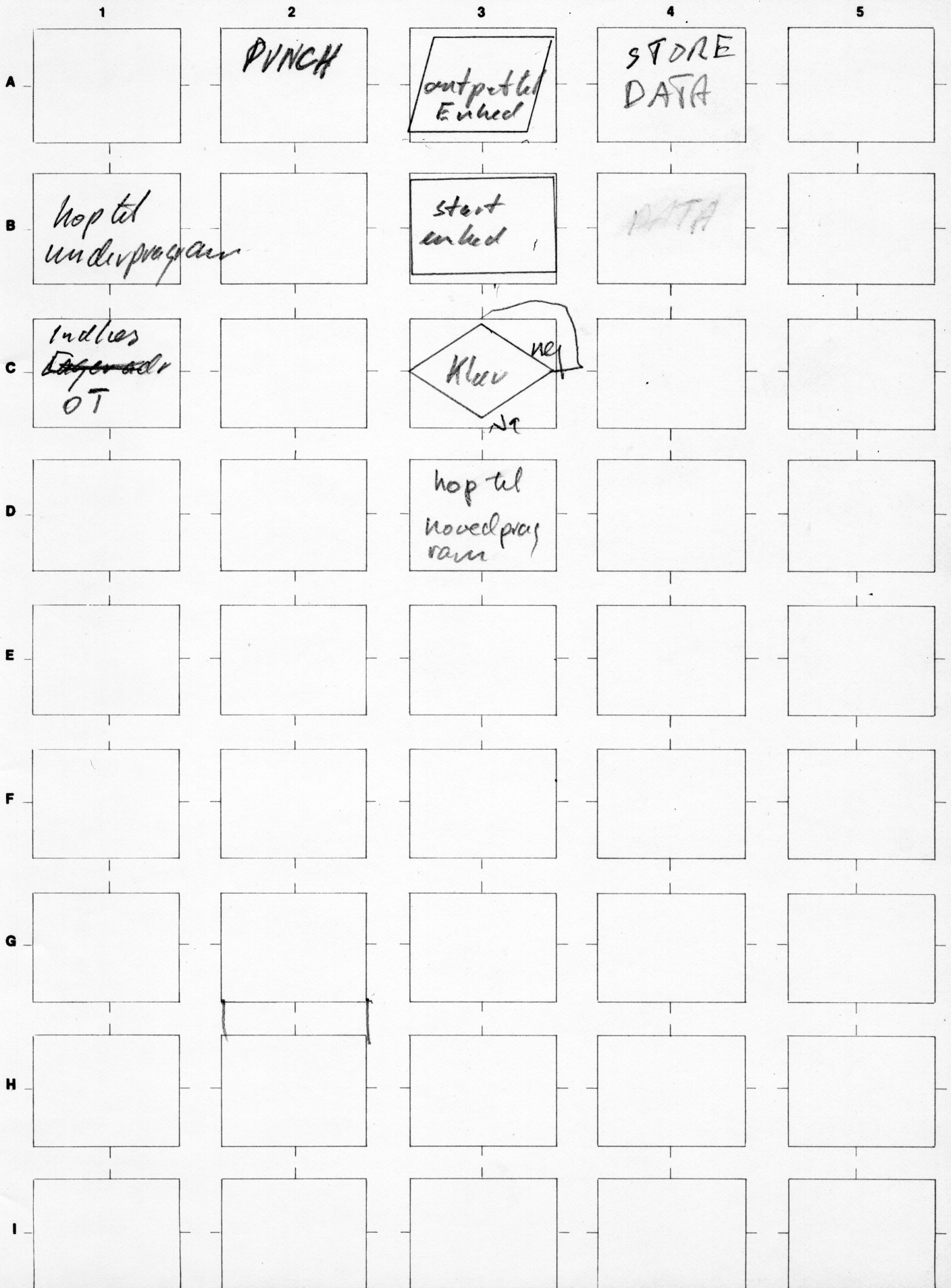
Udarbejder hovedprogram

Gruppe 4.

Udarbejder subrutine.

# DIAGRAMMERING

Udfyldt af <b>WAP</b>	Udfyldt den <b>19780811</b>	Opgavenr <b>8.3</b>	Bilagnr	Sidenr
Processens navn / Nr.				



[illegible]

## 8. Programmering i symbolsk maskinsprog

Symbolisk maskinsprog (Assembly Language - A/L) er et programmeringssprog, der ligger meget tæt op ad maskinsproget for en given datamaskine.

Når man anvender symbolsk maskinsprog i stedet for maskinsproget, så er det for at lette arbejdsgangen i forbindelse med udvikling af programmer. Lettelsen består i, at der kan benyttes såvel symbolske operationskoder som symbolske operander. Det betyder, at det ikke er nødvendigt, for den der skal skrive et program, at huske de binære værdier for operationskoderne, og beregne talværdierne for de adresser, der hægtes på instruktionerne.

Et program i symbolsk maskinsprog kaldes i visse sammenhænge for et kildeprogram (Source Program), hvor et program i maskinsprog kaldes et objektprogram (Object Program).

## 8.1 \_\_\_Oversættelse af program i symbolsk maskinsprog.

For at kunne benytte symbolsk maskinsprog til en given datamaskine, er det nødvendigt, at der til datamaskinen er udviklet et program, der kan omdanne (oversætte) de symbolske betegnelser i kildeprogrammet til binære værdier i objektprogrammet.

### 8.1.1 Hvad er en Assembler ?

Et program der kan foretage oversættelse fra symbolsk maskinsprog til maskinsprog kaldes en Assembler.

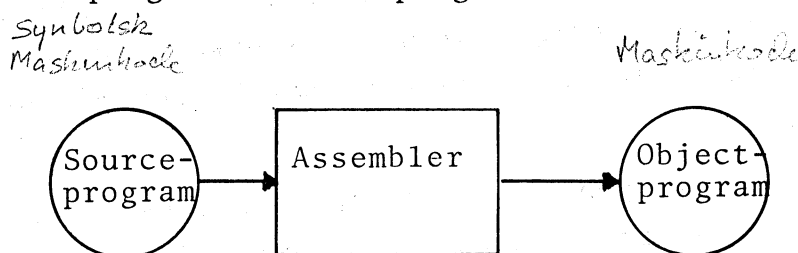


Fig. 8.1

En Assembler er altså i sig selv et program.

Et sådant hjælpeprogram leveres almindeligvis sammen med andre former for hjælpeprogrammer, når en datamaskine lejes eller købes hos en maskinleverandør.

Som oftest oversættes programmer til en datamaskine på den maskine, som programmet til sin tid skal køre på. Der kan imidlertid forekomme situationer, hvor det er hensigtsmæssigt at benytte en anden datamaskine til at udføre oversættelsen, end den maskine som programmet er skrevet til. En sådan oversættelse kaldes en krydsoversættelse (Cross Assembly). Dette benyttes ofte, når der skal udvikles programmer til meget små datamaskiner.

### 8.1.2 Hvordan skrives instruktionerne ?

Ligesom der i almindeligt skriftsprog er regler for, hvordan ord skal staves og hvordan, man kan tillade sig at sætte ord sammen for at skabe sætninger, som kan forstås af andre, således er der også regler for, hvordan instruktioner i symbolsk maskinsprog må skrives, for at en oversætter kan omdanne disse til maskinsprog.

Disse regler kaldes for sprogets syntaks.

Det er med andre ord nødvendigt, at benytte et eller andet format, når der skal skrives symbolske instruktioner.

I denne forbindelse skelnes der mellem de oversættere, der benytter fast format og de, der benytter frit format.

Ved fast format forstås instruktionslinier, hvor de enkelte elementer af en instruktion, altid er at finde på det samme sted på linien.

Ved frit format forstås instruktionslinier, hvor det ikke er nødvendigt, at et givet element altid befinder sig på samme sted i instruktionslinien. I sådanne tilfælde er det nødvendigt, at benytte skrifttegn til at adskille de enkelte elementer. Et mellemrum (Space) er i denne forbindelse at betragte som et skrifttegn.

De elementer, der kan indgå i en symbolsk instruktion, er - Instruktionsnavn (Label) - Operationskode (Mnemonic Code) - Operand og Kommentar (Comment).

LABEL	MNE	OPERAND	COMMENT
START	LDX	TIME	HENT TIMETAL TIL X-REG

Eks. på fast format.

START: LDX TIME ; HENT TIMETAL TIL X-REG

Eks. på frit format.

Et symbolsk maskinsprogs syntaks er bestemt af den, der har skrevet oversætterprogrammet. De skitserede eksempler er derfor ikke standardiserede, og der forekommer i praksis store afvigelser mellem forskellige programprodukter.



På samme måde som i maskinsprog, kan man i symbolsk maskinsprog skelne mellem aktive og passive instruktioner.

De aktive instruktioner (Imperatives) vil alle have tilknyttet en entydig mnemoteknisk kode, som vil blive omsat til et bitmønster i objekt-koden.

De mnemotekniske koder er forkortelser af engelske ord, som beskriver den handling, som ønskes udført af den pågældende instruktion.

Typiske mnemotekniske koder er:

KODE	ENGELSK	DANSK
ADD	Add	Adder
SUB	Subtract	Subtraher
MPY	Multiply	Multipliker
DIV	Divide	Divider
LDX	Load	Flyt fra lager til X-reg.
STA	Store	Flyt fra accumu. til lager
STAX	Store	Flyt fra accumu. til X-reg.
JMP	Jump	Ubetinget hop i sekvens
JAZ	Jump if Zero	Hop hvis Accumu. er lig nul
NOP	No Operation	Ingen operation
INP	Input	Indlæs data fra ydre enhed
OUT	Output	Udlæs data til ydre enhed
HLT	Halt	Stop datamaskinen

o.s.v.

De passive instruktioner (Declaratives) er de, der benyttes til at beskrive variable og/eller konstanter i et program. Sådanne instruktioner beskriver dataelementer og eventuelt deres indhold. Indholdet af sådanne dataelementer kan opfattes som tal eller tekst.

Tal er binære værdier i form af heltal eller kommatal.

Tekst er bitkombinationer, der kan betragtes som tegn i det såkaldte ASCII-tegnsæt (ASCII = American Standard Code of Information Interchange) eller i EBCDIC-tegn-



sættet (EBCDIC = Extended Binary Coded Decimal Inter-change Code).

Heltal kan som oftest defineres som binære tal ved hjælp af etter og nuller - som decimaltal - hexadecimaltal eller oktaltal afhængigt af oversætterprogrammets syntaks.

Kommatal kan defineres ved en brøkdel og en eksponent. Også disse kan angives i en base, som er bestemt af oversætteren.

Typiske koder for erklæringer er:

KODE	ENGELSK	DANSK
DEF	Define	Definer datacelle
OCT	Octal	Definer datacelle med oktalt indhold.
DEC	Decimal	Definer datacelle med decimalt indhold.
FLT	Floating Point	Definer datacelle med kommatalindhold.

I nogle systemer er indholdet afhængigt af, hvorledes operanden defineres.

F. eks.:

DEF X'1AF	Definer datacelle med hexadecimalt indhold.
DEF 105	Definer datacelle med decimalt indhold.
DEF 101101B	Definer datacelle med binært indhold.

### 8.1.3 Hvad sker der, når et program oversættes?

Et oversætterprogram adskiller sig ikke væsentligt fra andre programmer. Det indlæser data (kildeprogrammet), foretager en behandling af disse (intern oversættelse), og udlæser data (objektprogrammet).

Da det almindeligvis ikke er muligt, at have hele kildeprogrammet liggende i datamaskinens interne lager samtidig med oversætterprogrammet, kan det være nødvendigt at udføre oversættelsen af 2 gange. Dette kaldes en 2-faset oversættelse.

Når det er nødvendigt at udføre denne funktion dobbelt, så skyldes det, at oversætterprogrammet ikke kan have overblik over alle anvendte navne, før alle linier i kildeprogrammet har været læst, idet der i en linies operand kan være reference til en anden linie, hvis navn først forekommer senere i programmet.

Det skal dog nævnes, at der eksisterer oversættere, der kan udføre arbejdet i et enkelt gennemløb, men disse kræver, at alle anvendte navne defineres i begyndelsen af programmet.

Det almindeligste er en 2-faset oversætter.

Første fase benyttes som oftest til at kontrollere, at sprogets syntaks er overholdt, og at opbygge en tabel over, hvilken adresse i datamaskinens lager et anvendt navn bliver tilknyttet. Denne tabel kaldes en symboltabel.

I anden fase foretages så den egentlige oversættelse, hvor de mnemotekniske kode omsættes til binære koder. Denne omsætning sker almindeligvis ved opslag i en tabel. Adresserne svarende til de anvendte operander hentes fra symboltabellen.

# DIAGRAMMERING

Udfyldt af	Udfyldt den	Opgavenr	Bilagnr	Sidenr
Processens navn / Nr.		Principskitse for 2-faset oversætter		

1

2

3

4

5

A

B

C

D

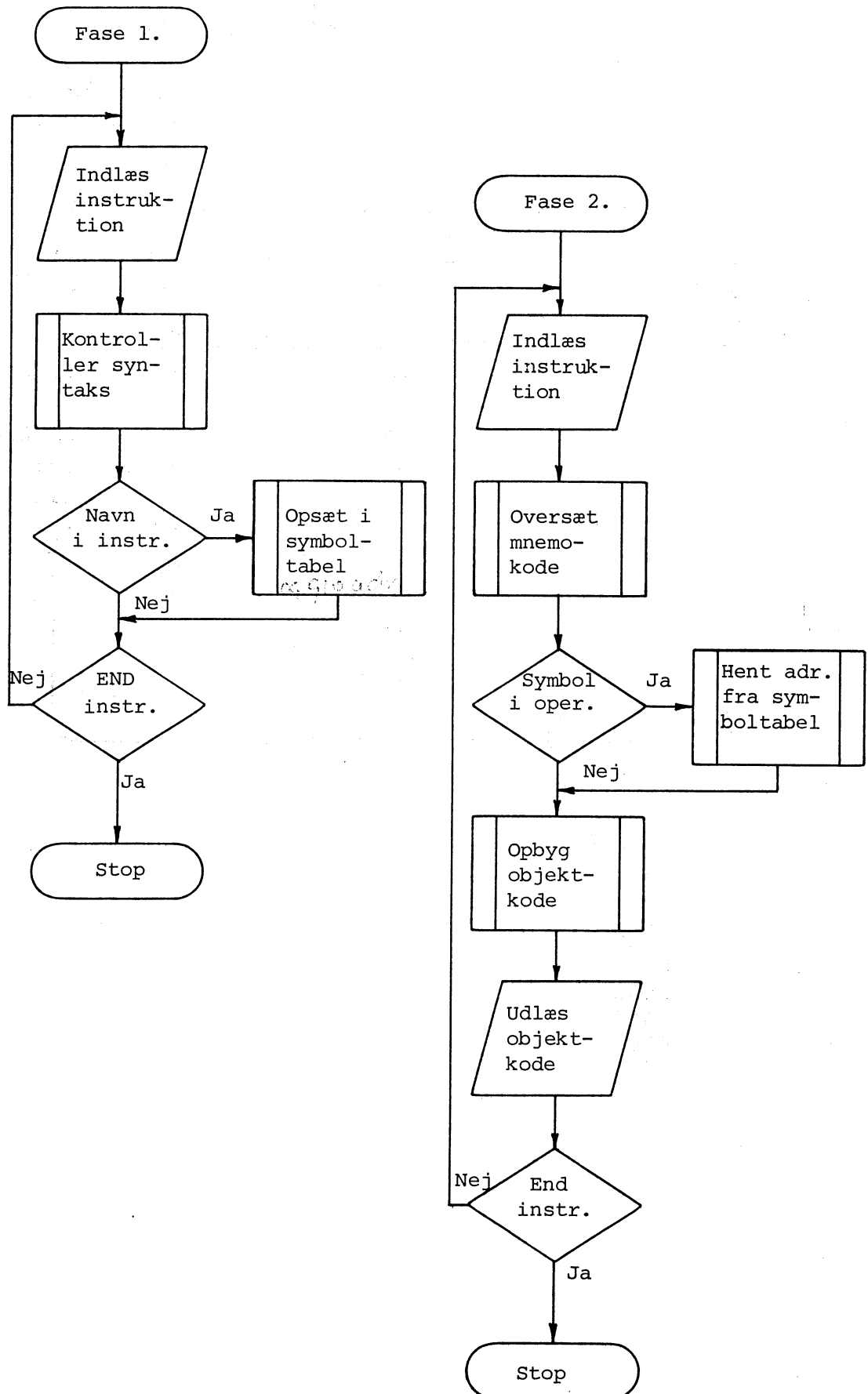
E

F

G

H

I



Eksempel på oversættelse af et program skrevet i symbolsk maskinsprog.  
 Programmet adderer 2 til Acc., indtil summen 10 nås. (Operationskoder er som for datamaskinen i afsnit 3).

Kildeprogram:

```

    ORG 100
    LDX TAL1
IGEN LDY TAL2
    ADD
    LDAX
    LDY TAL3
    SUB
    JAZ GEM
    JMP IGEN
GEM  STA SUM
    HLT
TAL1 DEC 0
TAL2 DEC 2
TAL3 DEC 10
SUM  DEC 0
    END
    
```

Oversættelse Fase 1

Symboltabel

SYMBOL ADRESSE

```

    IGEN 101
    GEM  111
    TAL1 113
    TAL2 114
    TAL3 115
    SUM  116
    
```

Oversættelse Fase 2

Operations-  
kode tabel

Symbol tabel

MNE	KODE	SYMBOL	ADRESSE
ADD	010	IGEN	101
HLT	00	GEM	111
JAZ	16	TAL1	113
JMP	14	TAL2	114
LDAX	020	TAL3	115
LDX	06	SUM	116
LDY	05		
STA	04		
SUB	011		

Objektprogram:

```

060113
050114
010000
020000
050115
011000
160111
140101
040116
000000
000000
000002
000012
000000
    
```

## 8.2. Særlige faciliteter.

### 8.2.1 Pseudoinstruktioner.

Når der skal skrives programmer i symbolsk maskinsprog, kan det være praktisk for den, der skal skrive programmet, at have indflydelse på, hvorledes oversættelsen til maskinsprog skal foregå.

I symbolsk maskinsprog vil der derfor ofte forekomme såkaldte pseudoinstruktioner. Når de kaldes sådan, hænger det sammen med, at de ligner instruktioner, men at de kun har betydning i forbindelse med oversættelsen. De har ingen funktion ved den egentlige programafvikling.

Der kan være mange muligheder for at styre en oversættelse.

F. eks. at hægte et navn på et program eller programdel eller, at bestemme hvorledes et program skal placeres i datamaskinens lager m.m.

Eksempler på pseudokoder:

ORG 100	(Origin)	Placer de efterfølgende instruktioner fra og med adresse 100 decimalt.
ADRESS EQU X'101A		Sæt navnet ADRESS til hexadecimalt 101A i Symbol tabel.
(Equate)		
HEAD "OVERSKR"		Placerer teksten OVERSKR øverst på hver side af programliste.
NAM "PRG1"		Navngiver programmet til PRG1.
END		Programafslutning.

### 8.2.2 Makroinstruktioner.

Mange assembler til datamaskinudstyr tillader brugen af makroinstruktioner.

Makro'er er et yderst vigtigt redskab, der, når det benyttes rigtigt, vil forøge effektiviteten og læsbarheden af ens programmer.

En makroinstruktion er et symbol (makro-navn), der, når det placeres i operationskodefeltet i ens kildeprogram, bevirker, at oversætteren (assembleren) genererer en hel sekvens af assemblerinstruktioner på den plads, hvor makronavnet er placeret.

Det er brugeren (programmøren), der på forhånd har defineret sekvensen af assemblerinstruktionerne.

Til denne definition benyttes makro-definitionspseudoer.

#### Makro definition.

Et eksempel på en makro-definition er vist nedenfor:

<u>Label</u>	<u>Opr-kode</u>	<u>Operand</u>	<u>Kommentar</u>
SUB	MACRO		
	INVX		Inverter X-reg
	INCX		Inkrementer X-reg
	ADD		Add X-og Y-reg.
	MACEND		

Det lille program mellem de to makro-pseudoer kan benyttes til at substrahere to tal, der står i datamaskinens X- og Y-registre. SUB er makroens symbolske navn.

Ved begrebet "macro-body" forstår man de assembleringsinstruktioner der er indeholdt i makrodefinitionen.

I dette tilfælde er det instruktionerne INVX, INCX og ADD.

Ved programmering af datamaskiner der ikke er "født"

med en subtraktions-maskininstruktion, - må man udføre subtraktioner programmæssigt, og man kan ofte få brug for at skrive den samme instruktionssekvens mange gange.

Ved programmering med makroer er det kun nødvendigt at skrive instruktionssekvensen een gang, nemlig når den defineres. Derefter skal der blot refereres til makro-navnet.

#### Makroreference eller-kald.

En makro kan, når den er defineret, kaldes som en instruktion i et kildeprogram.

<u>Label</u>	<u>Opr-kode</u>	<u>Operand</u>	<u>Kommentar.</u>
	:		
	LDX	TAL1	Hent TAL1 til X-reg
	LDY	TAL2	Hent TAL2 til Y-reg
	SUB		Subtraher (makro-navn)
	STA	DIFF	Gem ACC i DIFF
	:		

I ovenstående program er SUB en makroinstruktion, der kaldes i kildeprogrammet.

#### Makroekspansion.

Ved assemblering af et program, der indeholder makro-navne, vil det enkelte makronavn hver gang blive erstattet med de assemblerinstruktioner, der er indeholdt i makroinstruktionens definition ("Macro-body").

Ovenstående program vil ved assembleringen blive erstattet med følgende instruktionssekvens:

<u>Label</u>	<u>Opr-kode</u>	<u>Operand</u>	<u>Kommentar.</u>
	LDX	TAL1	
	LDY	TAL2	
	INX		"Macro-body"
	INCX		
	ADD		
	STA	DIFF	

Makroer bruges med fordel når samme sekvens af assemblerinstruktioner anvendes hyppigt i et program.

Programoverskueligheden forøges, ligesom antallet af småfejl reduceres.

Man har også mulighed for at definere hensigtsmæssige instruktioner, der ikke umiddelbart forefindes i datamaskinens instruktionssæt. (Den foregående makroinstruktion SUB er et eksempel herpå).

Man må være opmærksom på, at ved instruktionssekvenser, der indeholder mere end nogle få instruktioner, og som skal gentages flere gange, kan det være mere økonomisk at anvende subrutiner.



### 8.2.3 Relokerbar assemblering.

I afsnit 8.1.3 er vist et program, hvor programmøren har specificeret lageradressen for programmets første instruktion. (Ved hjælp af ORG-pseudoen).

Et sådant program kaldes et absolut program, og oversætteren (Assembleren) oversætter kildeprogrammets symbolske adresser i forhold til ORG statementets operand.

Objektprogrammets placering i lageret er derved fastlåst.

Hvis adskillige programsektioner skal være i lageret samtidigt, er det et vanskeligt og tidskrævende job at holde styr på de forskellige programmers længder og startadresser. Der er således mulighed for, at overlapning eller unødigt spild af lagerpladser finder sted.

Specielt er det ønskeligt at kunne indlægge sine programmer på vilkårlige pladser i lageret, når man i sit hovedprogram kalder allerede oversatte biblioteks-rutiner.

Ud fra den betragtning at biblioteksrutiner og andre subrutiner ofte allerede findes som oversatte programmer (objektprogrammer), er det ønskværdigt at man først samler alle sine delprogrammer til et stort program, når alle delprogrammerne er assembleret. D.v.s., at beslutningen om hvor i lageret de enkelte delprogrammer skal placeres, udskydes indtil alle programmerne forefindes som relokerbare objektprogrammer. Relokerbare programmer er flyttelige programmer, der er assembleret på en sådan måde, at de kan lagres og udføres på vilkårlige lagerpladser.

Når flere relokerbare programmer skal sættes sammen, rejser der sig en del problemer:

Hvordan sikrer vi os mod programoverlapping eller spild af lagerpladser ?

Hvorledes kan vi få to uafhængige programmer (f.eks. hoved- og biblioteksprogram) til at kommunikere med hinanden ? (Hop til underprogram, overføring af data og variable o.s.v.).

Hvis brugeren (programmøren) skal tage sig af disse ting, vil der uvilkaarligt opstå fejl.

Det er nærliggende at overlade dette job til datamaskinen.

Mange datamaskiner er da også forsynet med særskilt programmel, der er i stand til at sammensætte flere assemblerproducerede objektprogrammer.

Programmer, der kan sammensætte relokerbare objektprogrammer, kaldes ofte Relokerbare Loadere, eller Linking Programmer.

Ved relokerbar assemblering vil kildeprogrammerne blive oversat således, at alle de relokerbare objektprogrammer starter i adresse 0.

Først ved sammensætning ved hjælp af Linking Programmet sker den endelige fastlæggelse, af hvor hver enkelt programdel bliver placeret i lageret. D.v.s. at en af Linking Programmets opgaver er at modificere lageradresserne i de individuelle relokerbare objektprogrammer.

En anden af Linking Programmets opgaver er at etablere kommunikation mellem de enkelte programmer, således at spring til og fra subrutiner kan foretages. Ligeledes skal Linking Programmet sikre at evt. overføring af data kan ske fra den ene programdel til den anden.

Linking Programmer, der er i stand til at løse disse opgaver, kan være konstrueret efter forskellige

principper. Vi skal ikke komme ind på disse principper her.

Allerede under udarbejdelse af kildeprogrammerne skal programmøren benytte forskellige pseudoer, for at assembleren og senere Linking Programmet kan etablere en korrekt programkommunikation.

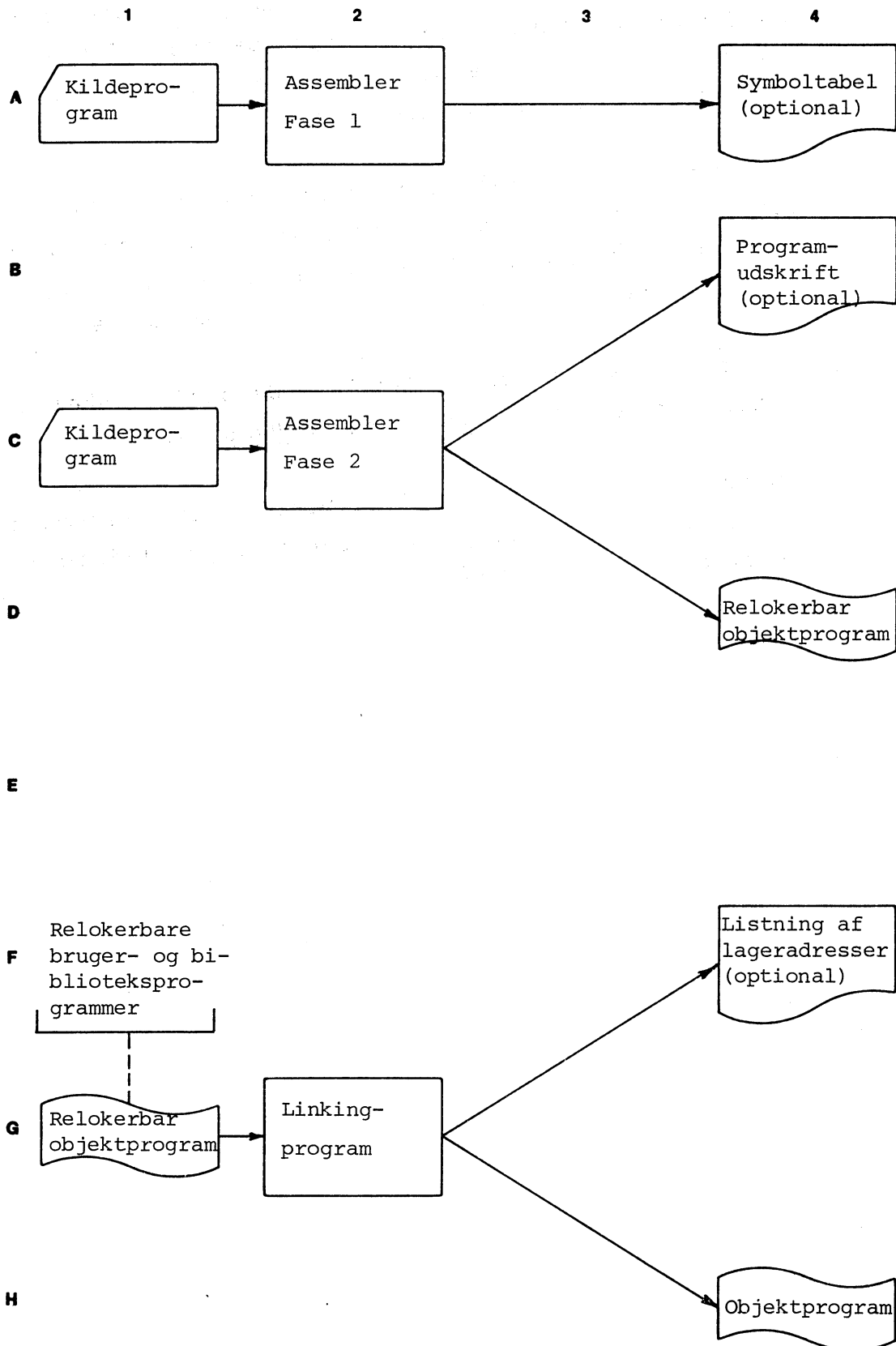
Hvis man f.eks. i sit kildeprogram refererer til symbolske adresser, der er defineret i en anden programdel, er man nødt til at fortælle assembler og Linking Program, at disse adresser er eksterne.

Relokerbare assemblere indeholder da også forskellige former for Objekt Program Linking Pseudoer.

Den praktiske arbejdsgang i forbindelse med assemblering og Linking af programmer for et papirstrimmesystem er vist på side 8.16.

# DIAGRAMMERING

Udfyldt af	Udfyldt den	Opgavenr	Bilagnr
Processens navn / Nr.		Relokerbar assemblering	



#### 8.2.4 Betinget assemblering.

En assembler med betinget assemblering, gør det muligt for en bruger at definere en sekvens af assemblerinstruktioner, der styret af en parameter værdi på assembleringstidspunktet, medtages eller udelades ved assembleringen.

En sekvens af assemblerinstruktioner gøres betingede, ved at afgrænse den med et par pseudoinstruktioner, f.eks. på følgende form:

<u>Label</u>	<u>Opr-kode</u>	<u>Operand</u>	<u>Kommentar</u>
	:		
	IF	UDTRYK	UDTRYK=parameter værdi
	:		
	Assembler statements		
	:		
	ENDIF		
	:		

I det viste eksempel assembleres instruktionerne mellem pseudoerne IF og ENDIF kun, hvis parameteren UDTRYK er tillagt en værdi forskellig fra nul.

Betinget assemblering kan f.eks. med fordel anvendes, hvor man har udviklet et generelt program med forskellige valgmuligheder (options), og hvor hver valgmulighed er betinget af hver sin parameter. For et givet antal valgmuligheder sættes de tilhørende styreparametre "til", og man vil ved assembleringen få den ønskede programversion.

Symbolisk maskinprogrammering.

Opgave 8.1

Øvelse 1: Ombytning af to tal i lageret.

Udarbejd i symbolsk maskinsprog et program, der kan ombytte to tal, der er placeret i lageret.

TAL 1: 144<sub>8</sub>

TAL 2: 12<sub>8</sub>

Programmet starter på adresse 10000<sub>8</sub>



Ombytning af to tal i lageret.

PAGE 0001

```
0001          ASMB,A,B,L,T
TAL1      010005
TAL2      010006
** NO ERRORS*
```

PAGE 0002 #01

```
0001          ASMB,A,B,L,T
0002      10000          ORG 10000B
0003      10000 062005    LDA TAL1
0004      10001 066006    LDB TAL2
0005      10002 072006    STA TAL2
0006      10003 076005    STB TAL1
0007      10004 102000    HLT
0008      10005 000144    TAL1 OCT 144
0009      10006 000012    TAL2 OCT 12
0010          END
** NO ERRORS*
```





[illegible]



PROGRAMMERING I SYMBOLSK MASKINSPROG

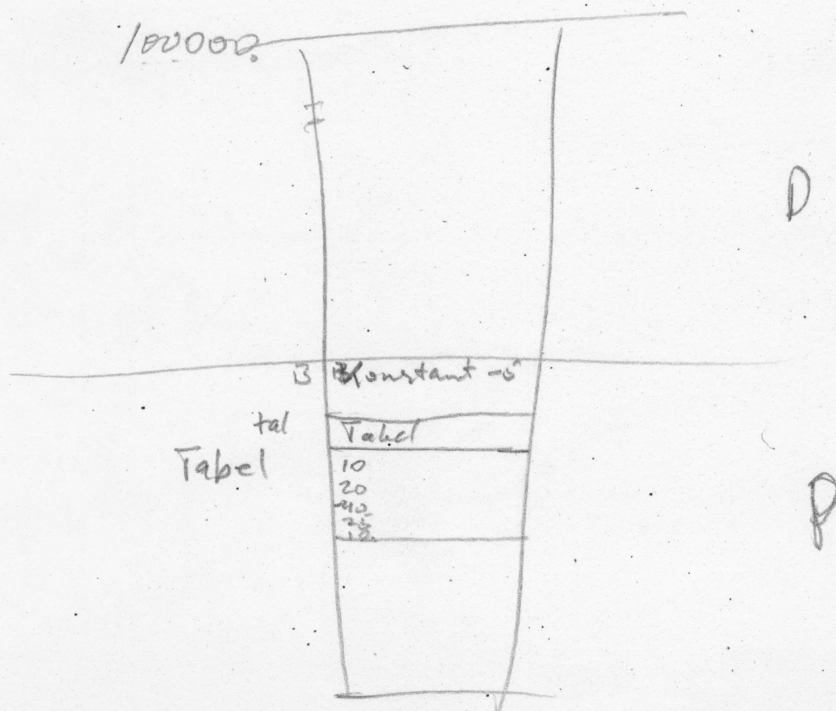
Opgave 8.2

Udarbejd et program startende i adresse 10000<sub>8</sub>,  
der adderer 5 tal som er placeret i en tabel.

A-reg bruges til additionen.

Tabellen har følgende indhold:

TABEL 10  
20  
-40  
25  
-12





ME-DATAMASKINER

PROGRAMMERING I SYMBOLSK MASKINSPROG

OPGAVE 8.2

ADDITION AF FEM TAL

PAGE 0001

```
0001          ASMB,A,B,L,T
FEM           010013
ADRPO         010016
FEMT          010014
IGEN          010005
START         010000
TABAD         010015
TABEL         010017
** NO ERRORS*
```

PAGE 0002 #01

```
0001          ASMB,A,B,L,T
0002 10000          ORG 10000B
0003 10000 066013   START LDB FEM
0004 10001 076014   STB FEMT
0005 10002 066015   LDB TABAD
0006 10003 076016   STB ADRPO
0007 10004 002400   CLA
0008 10005 142016   IGEN ADA ADRPO,I
0009 10006 036016   ISZ ADRPO
0010 10007 036014   ISZ FEMT
0011 10010 026005   JMP IGEN
0012 10011 102001   HLT 1
0013 10012 026000   JMP START
0014 10013 177773   FEM DEC -5
0015 10014 000000   FEMT BSS 1
0016 10015 010017   TABAD DEF TABEL
0017 10016 000000   ADRPO BSS 1
0018 10017 000012   TABEL DEC 10,20,-40,25,-12
      10020 000024
      10021 177730
      10022 000031
      10023 177764
0019          END
** NO ERRORS*
```





## 12. Stikordsregister

1-KOMPLEMENT	1. 21
2-FASET OVERSÆTTELSE	8. 6
2-KOMPLEMENT	1. 21
A/D KONVERTER	5. 27
AABEN KOLLEKTOR-UDGANG	9. 12
AABENT UNDERPROGRAM	7. 21
ABSOLUT PROGRAM	8. 13
ADDITION	1. 17
ADDITION OG SUBTRAKTION AF KOMPLEMENTTAL	1. 23
ADRESSE	2. 4
ADRESSE-, DATA- OG KONTROLBUS	9. 18
ADRESSEBUS	4. 7
ADRESSEBUS	9. 8
ADRESSEDEL	3. 4
ADRESSEFELT	3. 18
ADRESSELOGIK	9. 21
ADRESSELOGIK	9. 5
ADRESSEREGISTER	2. 19
ADRESSERINGSFORMER	7. 8
ADRESSETÆLLEREGISTER	2. 19
AKKUMULATOREN	2. 11
AKTIVE INSTRUKTIONER	8. 4
AKTIVITET	6. 5
ALGORITME	3. 1
ANDEN FASE	3. 10
ANDET STEDS FASTLAGT PROCES	6. 22
ARBEJDSLÅGER	2. 3
ARBEJDSREGISTRE	2. 10
ARITMETISK-LOGISK ENHED ALE	2. 10
ARITMETISKE OPERATIONER	7. 5
ARITMETISKE ORDRE	3. 6
ASCII KODE	5. 16
ASCII TEGNSÆT	8. 4
ASSEMBLER	8. 2
ASSEMBLY LANGUAGE	8. 1
ASYNKRON BUS	9. 9
ASYNKRON SERIETRANSMISSION	10. 1
ASYNKRONBUS	9. 7
ASYNKRONSYSTEM	10. 1
AUDIOVISUEL DATATERMINAL	6. 14
BAADE-OG	6. 16
BAGGRUNDSLÅGRE	5. 1
BAGGRUNDSLÅGRE	5. 5
BALANCERET TRANSMISSIONSLINIE	9. 58
BASE	1. 2
BAUD	5. 15
BAUD	10. 5
BETINGEDE HOPORDRER	3. 7
BETINGET ASSEMBLERING	8. 17
BETINGET HOP	7. 6
BIBLIOTEKSRUTINER	8. 13
BINÆR ARITMETIK	1. 17
BINÆRE TALSISTEM	1. 3
BLOK	5. 6

BLOKDIAGRAM	6.21
BLOKDIAGRAM	6.2
BLOKGAB	5.8
BRANCH	7.5
BROEKDEL	8.5
BROEKTAL	1.11
CALLING INDICATOR CI	10.28
CCITT	10.3
CCITT, S ANBEFALING V-21	10.23
CELLER	2.4
CENTRALENHEDEN	2.2
COMMENT	8.3
CONNECT DATA-SET TO LINE	10.27
CYKLUSTIDEN	2.6
CYKLUSTIDEN	2.7
DATA	2.3
DATA	3.2
DATA	7.1
DATA CHANNEL RECEIVED LINE SIGNAL DETECTOR DCD	10.10
DATA SET READY DSR	10.26
DATA TERMINAL READY DTR	10.27
DATABUFFERREGISTER	3.17
DATABUS	4.7
DATABUS	9.8
DATABUSSYSTEM	2.26
DATAMASKINENS RUSSYSTEM	9.7
DATASTROEMMEN	2.23
DATATRANSMISSION	10.1
DATATRANSMISSIONSKREDSLOEB	6.16
DATATRANSPORTVEJE	2.3
DE FIRE REGNEARTER	1.17
DEMULATOR	10.21
DIAGNOSTIKPROGRAMMER	2.1
DIAGNOSTIKPROGRAMMER	11.1
DIAGRAMMERING	6.1
DIAGRAMMERINGSARK	6.8
DIAGRAMSKABELON	6.8
DIGITAL TIL ANALOG OMSAETTER	5.25
DIRECT MEMORY ACCESS DMA	2.20
DIRECT MEMORY ACCESS	4.2
DIRECT MEMORY ACCESS DMA	4.23
DIREKTE ADRESSERING	3.5
DIREKTE ADRESSERING	7.9
DIREKTE STYRET LAGER	6.14
DIVISION	1.19
DOBBELT PRAECISION	1.29
DRIFTSYSTEM	2.1
DSR, DATA SET READY	10.25
DUPLEX	10.1
DYNAMISK HALVLEDERLAGER	2.8
E-FASE	3.10
EBCDIC-TEGN-	8.4
EKSEKVERINGSALGORITMER	3.88
EKSPONENT	1.13



EKSPONENT	1. 28
EKSPONENT	8. 5
ELEKTRONIK I IND/UDLÆSESYSTEMER	9. 1
ENTEN-ELLER	6. 16
ENVEJS- ELLER TOVEJSBUS	9. 7
ENVEJSBUS	9. 11
EPROM	2. 7
EXECUTE	3. 10
FAST FORMAT	8. 3
FELTER	5. 5
FILER	5. 5
FLOPPY DISC	5. 14
FLOW LINE	6. 17
FLOW LINE	6. 5
FLYDENDE TAL	1. 13
FLYDENDE TAL	1. 28
FLYTNING AF DATA	7. 4
FLYTTelige PROGRAMMER	8. 13
FLYTTEORDRER	3. 6
FOELESIGNAL	2. 22
FOERSTE FASE	3. 10
FORBINDELSSESLED	6. 5
FORBINDELSSESLED	6. 17
FORBINDELSSESLED	6. 22
FORBINDELSSESLINIER	6. 22
FORGRENING	6. 21
FREKVENSSKIFTMETODEN FSK	10. 24
FRIT FORMAT	8. 3
FULD DUPLEX	10. 1
FUNKTIONSDIAGRAMMET	6. 2
FUNKTIONSDIAGRAMMET	6. 3
GENPROGRAMMERBARE LÆSELAGRE	2. 9
GENTAGENDE UDFOERSEL	6. 22
GRAENSEFLADE STROEMSLØJFE	10. 15
GRUNDTAL	1. 2
HALV DUPLEX	10. 1
HALVLEDERLAGER	2. 7
HALVLEDERLAGRE	2. 8
HANDSHAKE-PROCEDURER	4. 3
HARDWARE	2. 1
HELTAL	8. 5
HENTEFASE	3. 10
HEXADECIMALE TALSYSTEM	1. 5
HJÆLPEPROCES	6. 16
HOLLERITH KODE	5. 22
HOPORDRE	3. 7
HOVED PROGRAM	11. 5
HULKORT	5. 21
HULKORT	6. 13
HULSTRIMMEL	6. 13
I-FETCH	3. 10
IFASE	3. 10
IMMEDIATE DATA	7. 8
IMPERATIVES	8. 4

IND-/UDLÆSEINSTRUKTION	7.12
IND/UD-ORDLÆNGDEN	2.19
IND/UD-ORDRER	3.5
IND/UD-ORDRER	3.8
IND/UDADRESSER	7.8
IND/UDLÆSE BUFFERE	9.29
IND/UDLÆSE BUFFERE	9.5
IND/UDLÆSEENHED	2.18
IND/UDLÆSNING AF DATA	7.7
INDEKSERET ADRESSERING	3.5
INDEKSERET ADRESSERING	7.11
INDEKSERET ADRESSERING	7.20
INDIREKTE	3.5
INDIREKTE ADRESSERING	7.11
INDIREKTE STYRET LAGER	6.15
INDLÆSEENHED	2.3
INDLÆSNING/UDLÆSNING	6.21
INDRE IMPEDANS 70	9.50
INDRE LAGER	6.14
INSTRUKTIONER	7.3
INSTRUKTIONSEFASE	3.10
INSTRUKTIONSFOMATER	7.12
INSTRUKTIONSLÆNGDE	7.3
INSTRUKTIONSNAMN	8.3
INTERNATIONALE RÅDGIVENDE TELEGRAF- OG TELEFONKOMITE	10.3
INTERRUPT	2.20
INTERRUPT	4.13
INTERRUPT I/O	4.1
INTERRUPTLOGIK	9.5
INTERRUPTLOGIK	9.39
JUMP	7.5
KARAKTERISTISKE IMPEDANS	9.52
KERNELAGRE	2.6
KILDEPROGRAM	8.1
KLARLOGIK	9.36
KOMMATAL	8.5
KOMMENTAR	6.6
KOMMENTAR	6.22
KOMMENTAR	8.3
KOMMUNIKATIONSENHEDER	5.1
KOMMUNIKATIONSMEDIER	5.3
KOMPARATOR	9.25
KOMPLEMENT FORM	1.20
KONSTANTER	8.4
KONTROLBUS	9.8
KONTROLBUS	4.7
KONTROLPALETS DATAKONTAKTER	2.23
KORTFI	6.13
KORTSTAK	6.13
KRYDSOVERSÆTTELSE	8.2
KRYDSTALE	9.50
LABEL	8.3
LADPROGRAMMER	2.1
LÆSECYKLUS	2.5

LAESELAGRE	2.8
LAGER REFERENCE INSTRUKTIONER	7.12
LAGERADRESSER	7.8
LAGERADRESSEREGISTER	2.4
LAGERADRESSEREGISTER	3.18
LAGERBUFFERREGISTER	2.4
LAGERENHEDEN	2.4
LAGERMEDIER	5.3
LAGEROVERFOERSELSORDRER	3.5
LINE DRIVERE OG RECEIVERE	9.57
LINKING PROGRAMMER	8.14
LOEBETIDSFORSINKELSEN	9.49
LOGISKE OPERATIONER	7.5
LOGISKE ORDRE	3.6
LUKKET UNDERPROGRAM	7.21
MACRO-BODY	8.10
MAGNETBAAND	6.14
MAGNETBAANDSSTATIONER	5.6
MAGNETPLADELAGER	6.14
MAGNETTROMLELAGER	6.14
MAKRO DEFINITION	8.10
MAKRO-DEFINITIONSPSEUDOER	8.10
MAKRO-NAVN	8.10
MAKRO-PSEUDOER	8.10
MAKROEKSPANSION	8.11
MAKROINSTRUKTIONER	8.10
MAKROREFERENCE ELLER -KALD	8.11
MANTISSE	1.13
MANTISSE	1.28
MANUAL PROCES	6.16
MANUEL INDLAESNING	6.14
MAR	2.4
MARK	10.4
MARK	10.36
MASKET INTERRUPT	9.46
MASKINORDRER	3.1
MASKINORDRER	3.3
MASKINPROGRAM	3.1
MASKINSPROG	7.1
MASKINSTYRING	7.7
MATERIELLET	2.1
MBR	2.4
MIKROORDRER	3.13
MIKROORDRER	3.14
MIKROPROGRAM	3.12
MIKROPROGRAM	3.14
MIKROPROGRAMLAGER	3.17
MIKROPROGRAMMERET STYREENHED	3.17
MINUENDEN	1.19
MNEMONIC CODE	8.3
MNEMOTEKNISK KODE	8.4
MODEM	10.21
MODEM GRAENSEFLADE V-28	10.35
MODULATIONSHASTIGHEDEN	10.5

MODULATOR	10. 21
MULTIPLIKATION	1. 18
NEGATIVE HELLTAL	1. 26
NEGATIVE TAL	1. 22
NORMALISERET FORM	1. 14
OBJEKT PROGRAM LINKING PSEUDOCODE	8. 15
OBJEKTPROGRAM	8. 1
OKTALE TALSYSTEM	1. 6
OKTETTER	2. 5
OPERAND	3. 4
OPERAND	3. 9
OPERAND	7. 3
OPERAND	8. 3
OPERANDTYPER	7. 7
OPERATIONSDEL	3. 4
OPERATIONSKODE	7. 3
OPERATIONSKODE	8. 3
OPERATIONSREGISTER	2. 19
OPERATIONSTYPER	7. 4
OPERATIVSYSTEM	2. 1
OPKODEN	3. 4
ORD	2. 4
ORDLAENGDE	1. 25
ORDLAENGDE	2. 4
ORDLAENGDE	2. 5
ORDRE REGISTER	2. 23
ORDRER	7. 1
ORDRETAELLER	2. 23
OVERSAETTERPROGRAMMER	2. 1
PARALLEL UDFOERSEL	6. 22
PARALLEL TRANSMISSION	10. 1
PARITETSRIT	10. 4
PASSIVE INSTRUKTIONER	8. 4
PASSIVE INSTRUKTIONER DECLARATIVES	8. 4
PLADELAGER	5. 13
POSITIONSTALSYSTEM	1. 1
POSITIVE HELLTAL	1. 26
POSITIVE TAL	1. 22
POST-INDEKSERING	7. 11
POST-MODIFIKATION	7. 11
POSTER	5. 5
PRE-INDEKSERING	7. 11
PRE-MODIFIKATION	7. 11
PRIORITERINGSLOGIK	9. 5
PRIORITERINGSLOGIK	9. 39
PRIORITERINGSSYSTEMET	9. 42
PROCES	6. 21
PROCES	6. 16
PROCESDATA	5. 22
PROGRAMLOEKKER	3. 7
PROGRAMMELI ET	2. 1
PROGRAMMERBARE LAESEL AGRE	2. 9
PROGRAMMERING I MASKINSPROG	7. 1
PROGRAMMODIFIKATION	6. 22

PROGRAMSTYRET DATAOVERFOERSEL	2. 20
PROGRAMSTYRET I/O	4. 1
PROGRAMSTYRET I/O	4. 9
PROGRAMSTYRING	3. 1
PROGRAMSTYRINGSOPERATIONER	7. 5
PROM	2. 7
PSEUDOINSTRUKTIONER	8. 9
RAM	2. 7
READY FOR SENDING RFS	10. 26
REFLEKTIONER	9. 53
REFLEKTIONSKOEFFICIENT	9. 52
REGISTER	2. 14
REGISTER REFERENCE INSTRUKTIONER	7. 12
REGISTERADRESSER	7. 7
REGISTEROVERFOERSLER	2. 16
REGISTRERING	6. 16
REGNEENHED	2. 3
REGNEENHED	2. 10
REGNEORDRER	3. 5
REGNEORDRER	3. 6
RELATIONSTEGN	6. 24
RELATIV ADRESSERING	7. 10
RELOKERBAR ASSEMBLERING	8. 13
RELOKERBARE LOADERE	8. 14
RELOKERBARE OBJEKTPROGRAMMER	8. 13
REQUEST TO SEND RTS	10. 26
RETTEPROGRAMMER	2. 1
RETURHOPADRESSEN	7. 23
ROM	2. 7
RUTINEAFPROEVNING	11. 3
SELECT TRANSMIT FREQUENCY	10. 28
SERIETRANSMISSION	10. 1
SIMPLEX	10. 1
SKJIFTEFUNKTION	2. 11
SKIP	7. 5
SKRIVE-LAESELAGER	2. 7
SKRIVECYKLUS	2. 6
SLOEJFEPROGRAMMERING	7. 16
SOFTWARE	2. 1
SOURCE PROGRAM	8. 1
SPACE	10. 4
SPACE	10. 36
SPECIELLE STYREORDRER	3. 7
STANDARD CODE FOR INFORMATION INTERCHANGE	5. 16
START/KLAR LOGIK	9. 5
STARTBIT	10. 4
STARTLOGIK	9. 23
STARTSYMBOL	6. 24
STATISKE HALVLEDERLAGER	2. 8
STOERRELSE OG FORTEGN	1. 26
STOPBIT	10. 4
STOPSYMBOL	6. 24
STRIMMEL, OPTISK LAESELIG	6. 15
STRIMMELHULLER	5. 21

STRIMMELÆSER	5.19
STYREENHED	2.22
STYREFEJL	3.18
STYREORDRER	3.5
STYREORDRER	3.7
STYRESIGNAL	2.23
SUBROUTINEBIBLIOTEK	2.1
SUBROUTINER	8.12
SUBTRAHERENDE	1.19
SUBTRAKTION	1.18
SUFFIX	1.2
SYMBOLSK MASKINSPROG	8.1
SYMBOLSKE OPERANDER	8.1
SYMBOLSKE OPERATIONSKODER	8.1
SYMBOLTABEL	8.6
SYNKRONBUS	9.7
SYNKRONSYSYSTEM	10.1
SYNTAKS	8.2
SYSTEMDIAGRAM	6.2
SYSTEMDIAGRAM	6.13
TALSYSTEMER	1.1
TEKST	8.4
TELEFAX	10.6
TERMINALER	5.14
TERMINALPUNKT	6.22
TEST-PROGRAMMER	11.1
THREE-STATE UDGANG	9.16
TIDSDIAGRAM	3.14
TIDSTRO KØRSEL	5.1
TILSTAND	6.5
TRANSMISSIONSHASTIGHED	10.5
TRANSMISSIONSHASTIGHED FOR TTY	10.15
TRANSMISSIONSLEDNING	9.49
TRAP-CELLE	4.21
TRUMLELAGER	5.11
TTY GRAENSEFLADE	10.15
UART	10.3
UART	10.7
UBETINGEDE HOPORDRER	3.7
UBETINGET HOP	7.6
UDFØRSELSFASE	3.10
UDEYLDT BLANKET	6.13
UDLÆSEENHED	2.3
UDVIDET ADRESSERING	7.10
UNDERPROGRAM	7.21
UNDERPROGRAM	11.5
UNDERSPECIFIKATION	3.6
UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER	10.7
USPECIFICERET TILSTAND	6.15
V-21 V-24 V-28	10.23
VÆGT	1.1
VÆGTEKOEFFICIENT	1.1
VENTE PÅ FLAG METODEN	9.36
WIRE-OR	9.12