JERNINDUSTRIENS FORLAG

Microcomputer-teknik

Svagstrømsteknisk værkstedskursus/skole

1985

Instruktioner

Jern- og Metalindustrien

Best. nr. 4017

Forord

I forbindelse med kursusplaner for svagstrømsteknisk værkstedskursus/skole udgives følgende lærebøger:

Lodde- og montageteknik - Instruktioner
Analogteknik - Instruktioner
Impulsteknik - Instruktioner
Microcomputerteknik - Instruktioner
Analog- og impulsteknik - Øvelser og opgaver

Lærebøgerne er udarbejdet på foranledning af faglærerne ved Håndværkerskolen i Sønderborg og Frederiksberg tekniske Skole i samarbejde med Jernindustriens Forlag.

Lærebøgerne vil være anvendelige til undervisning i elektronik ved HTX-uddannelsen.

Teoriinstruktioner

Teoriinstruktionerne er elevernes værktøj til at arbejde selvstændigt med stoffet. Instruktionerne tilgodeser lærerens pædagogiske frihed.

Der er en så fyldestgørende dækning af stoffet, at nogle emner på grund af tidsnød ikke kan dækkes af klasseundervisning, specielt på det halvårlige værkstedskursus.

Teoriinstruktionerne kan anvendes som opslagsværk af eleven i praktikperioden.

Øvelser

Øvelserne giver eleven mulighed for at arbejde med praktisk elektronik, herunder at udvikle færdigheder i måleteknik.

Øvelserne angiver hensigt og resultater, men overlader fremgangsmåden til lærer og elever.

Da det anvendte materiel afviger fra skole til skole, er øvelser, som er knyttet til bestemte apparater, ikke medtaget.

I microcomputerundervisningen er der dog anvendt Intels 8080/85 processor for at udnytte det hardware- og softwareudstyr, som elektronikmekanikeruddannelsen allerede anvender.

Teoriopgaver/øvelser

Teoriopgaver/øvelser afvikles således, at eleven først bearbejder stoffet teoretisk, hvorefter beregningerne bekræftes gennem tilhørende praktiske måleøvelser.

Opgaver

Opgaver giver eleven og læreren mulighed for at kontrollere indlæringen.

Opgaverne kan udføres som hjemmearbejde, idet de ikke kræver måleudstyr.

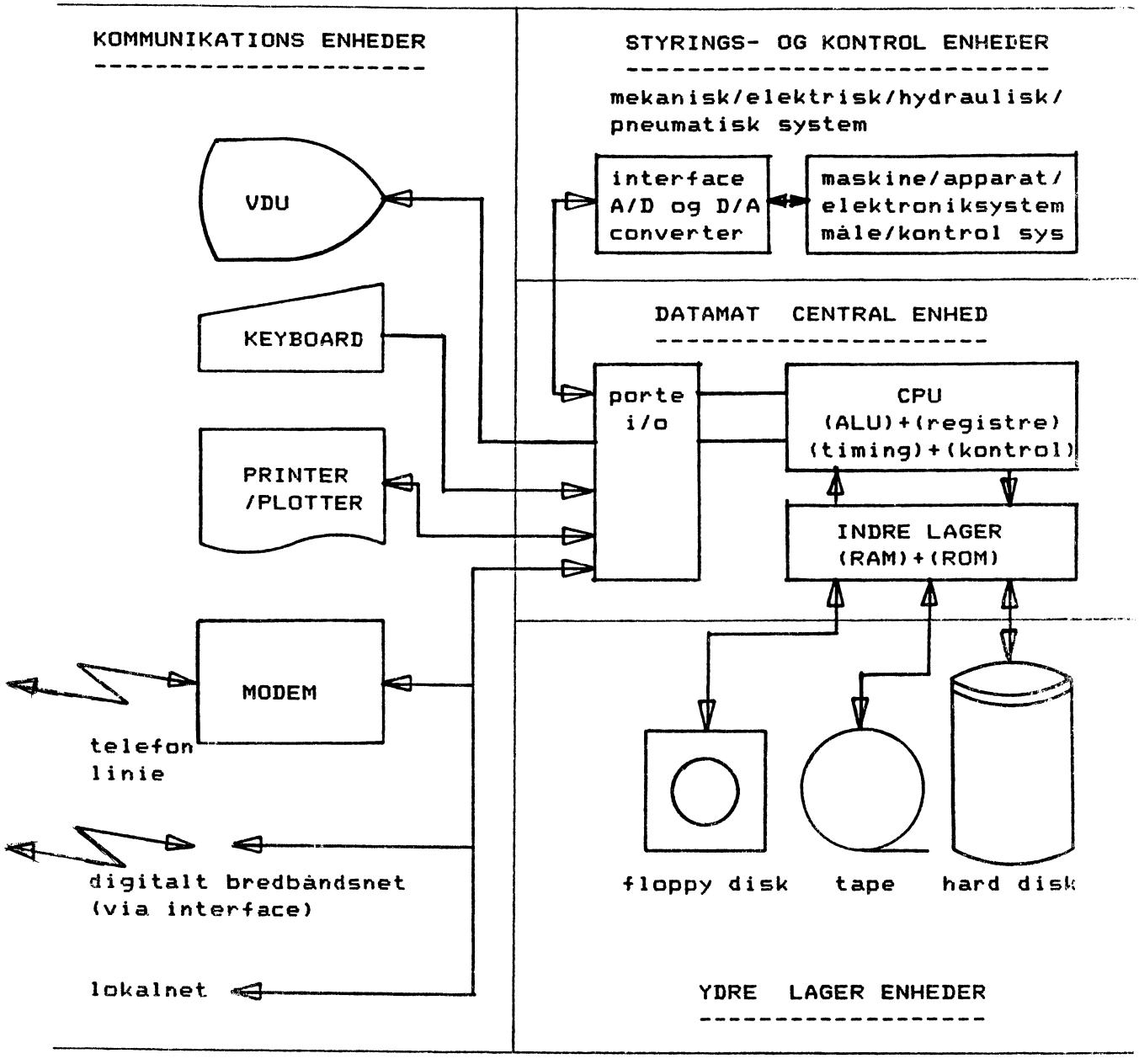
Øvelses- og opgavesamlingen må ikke anses for fyldestgørende, men kan suppleres af den enkelte skole og lærer i overensstemmelse med kursusplanen.

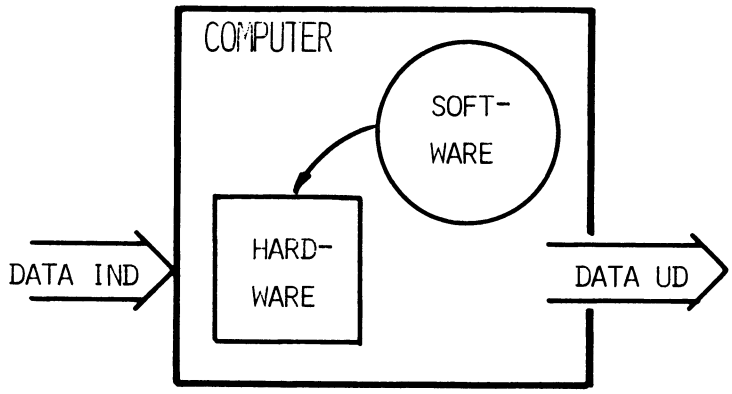
Forlaget modtager gerne forslag til ændringer og rettelser fra lærere, elever og andre interesserede.

København, maj 1985

JERNINDUSTRIENS FORLAG

Side:

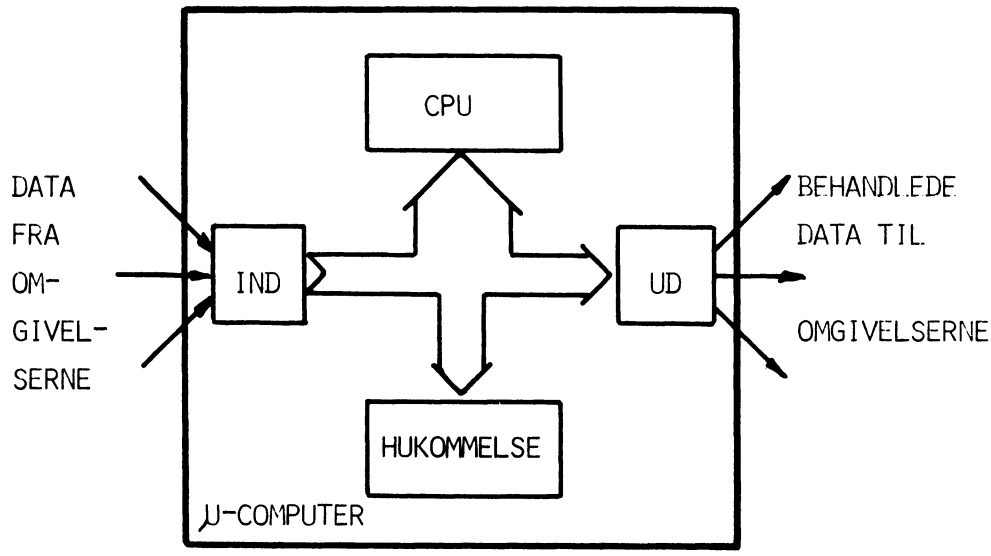




Hardware/software.

Computeren kan opfattes som bestående af to hovedbestanddele, hardware og software. Ved hardware forstås selve kredsløbet med dets IC'r, ledninger, kontakter o.s.v.. Softwaren er det program der får hardwaren til at udføre den aktuelle funktion. Medens det normalt vil være forbundet med vanskeligheder at ændre hardwaren, vil en ændring af softwaren nemt kunne udføres, og da det er softwaren der bestemmer hardwarens funktion, vil det ses, at en computer vil kunne tilpasses til at løse de forskelligste opgaver.

Af ovenstående fremgår også, at hvor arbejdet med et nyt kredsløb tidligere hovedsaglig lå i værkstedet, foregår en betydelig del af arbejdet nu ved skrivebordet med udvikling af programmer, så meget mere som man ofte køber hardwaren færdig i form af standard kredsløbskort.



CPU'en.

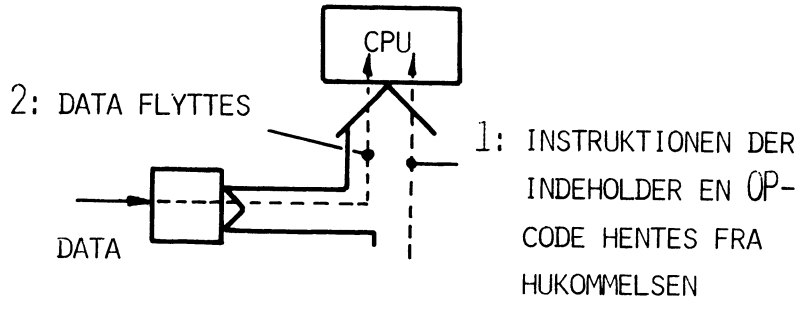
En µ-computer kan opdeles i fire hovedblokke som vist på blokdiagrammet.

Den centrale af disse blokke er microprocessoren (µP) også kaldet CPU'n, hvilket betyder Central Proces Unit.

Det er i CPU'n alle beregninger og beslutninger foregår, ligeledes kontrolleres input-og output kredsene samt hukommelsen af CPU'n.

Op-coder.

CPU'n er "født" med evnen til at udføre et begrænset antal små operationer, der set enkeltvis er uden mening, men som anvendt i rækkefølge kan bringes til at danne en hel proces.



Et eksempel på en af CPU'ens små operationer er at flytte data fra inputkredsen til CPU'n. For at få udført denne dataflytning skal CPU'n påvirkes med et tal, en såkaldt operationskode - (OP-code).

En CPU har alt efter fabrikat omkring 5o OP-coder som tilsammen danner et OP-codesæt.

Hukommelsen.

Hukommelsen har to opgaver i en µ-computer:

1. Den rummer de nødvendige OP-coder til at styre CPU'ens funktion. Denne samling af OP-coder kaldes programmet.

2. Den skal lagre mellemresultater og data.

Langtidshukommelse/programhukommelse.

Til at løse første opgave anvendes en langtidshukommelse, hvis indhold kun vanskeligt kan ændres.

Til gengæld kan apparatet slukkes uden at hukommelsens indhold går tabt. Denne hukommelse kaldes også programhukommelsen.

Korttidshukommelse/Data hukommelse.

Til opgave to anvendes en hukommelse, hvis indhold let kan ændres, men som stiller det krav, at der hele tiden skal være tilsluttet forsyningsspænding. Afbrydes denne blot et øjeblik, går indholdet tabt. Korttidshukommelsen kan findes i CPU'en, og kaldes da ofte "Scratchpad memory", der betyder "Notesblok hukommelse".

CPU'en contra hukommelsen.

Ved opstart af en µ-computer begynder CPU'en med at "bede" langtidshukommelsen om den første OP-code. Denne sendes til CPU'en, der straks eksekverer den. Derefter "beder" CPU'en om den anden OP-code, får den og udfører den, o.s.v.
Som det fremgår er CPU'en nok den der "gør arbejdet", men det er det i hukommelsen, lagrede program der fortæller CPU'en, hvad den skal foretage sig.

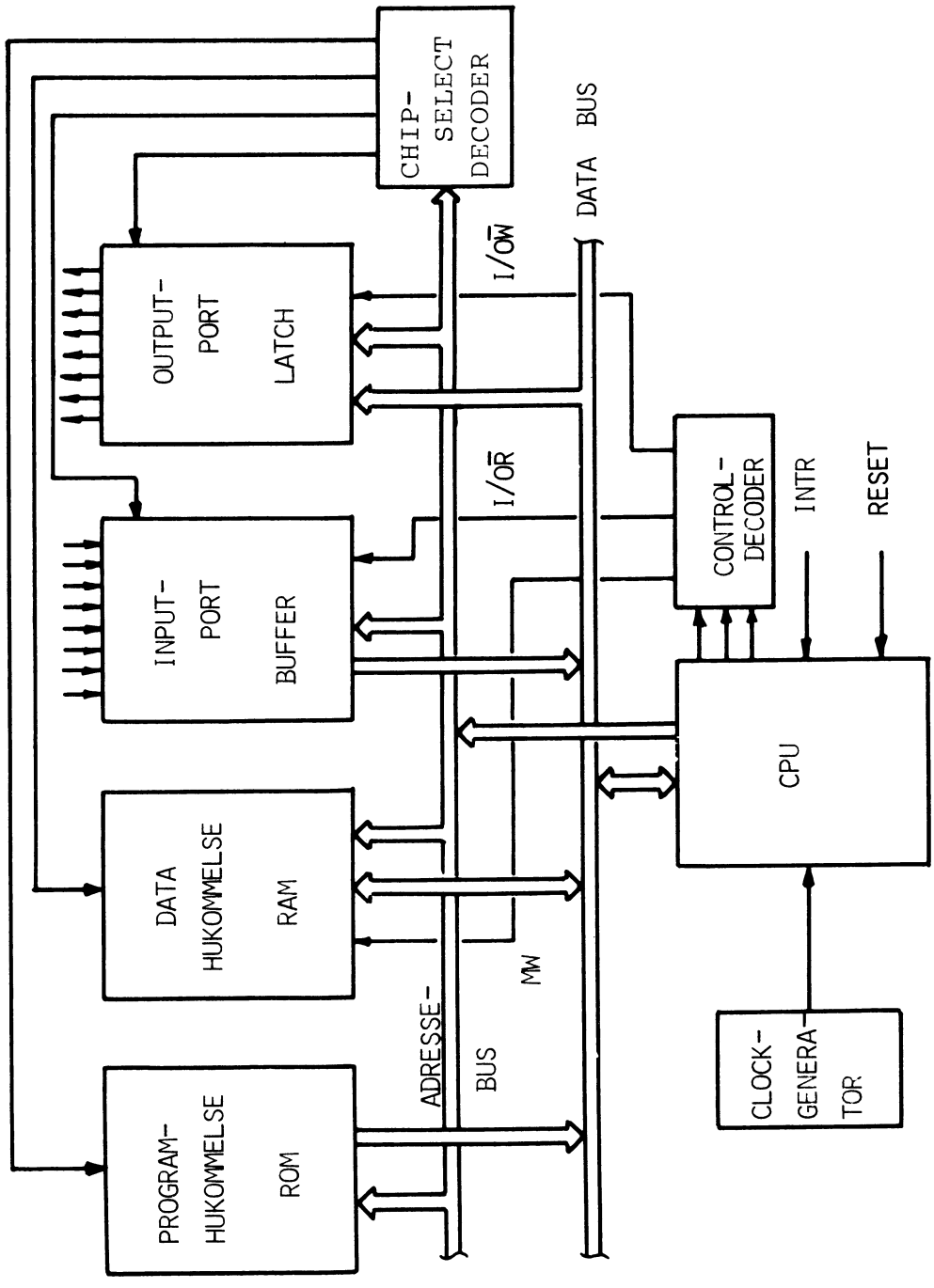
INPUT/OUTPUT.

INPUT- og OUTPUT-kredsene har til opgave at danne kontaktflade (interface) mellem µ-computeren og omgivelserne. De styres af CPU'en til at åbne og lukke når der skal overføres data fra eller til µ-computeren.

Systemdiagram.

Det herunder viste systemdiagram indeholder program-hukommelse (ROM), datahukommelse (RAM), CPU med taktgenerator, INPUT-port og OUTPUT-port.

Derudover er de nødvendige kommunikationslinier mellem ovennævnte kredsløb vist.



Hukommelsen.

Som tidligere nævnt kan CPU'en ikke udføre meget selv, medmindre den får de nødvendige ordrer.

Disse ordrer, kaldet programmet er lagret i programhukommelsen.

Krav.

Programhukommelsen skal kunne holde sit indhold i lang tid, også uden der er spænding tilsluttet apparatet, til gengæld er det ikke nødvendigt, at kunne ændre indholdet.

ROM.

Disse egenskaber findes i en ROM. ROM er en forkortelse af Read Only Memory, det vil sige en hukommelse, hvorfra der kun kan læses, d.v.s. hentes information.

ROM'ens indhold fastlægges allerede ved fabrikationen, hvorfor der skal anvendes et stort antal af samme type, hvis det skal være rentabelt.

PROM.

PROM'en er en fætter til ROM'en. Det tilføjede P står for Programable, d.v.s. programmerbar af brugeren. Denne kan ved hjælp af et programmeringsudstyr fremstille et enkelt styk til en foreliggende opgave, men som ved ROM'en kan indholdet ikke ændres når først programmeringen er sket.

EPROM.

Sidste skud på stammen er EPROM'en. E står for Erasable, altså sletbar. EPROM'en kan programmeres af brugeren på samme vis som PROM'en, men hvor det er en mekanisk ændring, der finder sted i PROM'en, er det blot en opladning af nogle kondensatorer, der sker i EPROM'en.

Finder brugeren på et senere tidspunkt ud af at programmet skal ændres, kan EPROM'ens indhold slettes ved af belyse den med kraftigt ultraviolet lys gennem vinduet.

Derefter kan der ske en ny programmering.

RAM.

Til lagring af DATA, det vil sige de signaler, som µ-computeren behandler, skal anvendes en hukommelse hvor man til enhver til kan lagre data, og hente dem igen.

Den såkaldte RAM har denne egenskab. RAM står for Random Access Memory, hvilket betyder "hukommelse med tilfældig adgang", altså en hukommelse, man kan komme med data til, og få dem fra på et vilkårligt sted og tidspunkt. RAM lageret benævnes også ofte read write memory, læse-skrive, hukommelse eller scratch-pad memory, notesblok hukommelse.

EAROM

Electrical Alterable ROM, er en "krydsning" mellem en ROM og en RAM, idet den kan slettes elektrisk og genprogrammeres medens den sidder i opstillingen.

OUTPUT-port.

OUTPUT-porten kan betragtes som en del af RAM-lageret, dog således at der kun kan gå data fra CPU'en til porten. Til gengæld vil disse data være tilgængelige for eksterne anvendelser.

INPUT-port.

INPUT-porten er i princippet kun 8 parallelle kontakter der slutter når porten adresseres. Derved bliver der forbindelse mellem data-indgangene og data-bussen og dermed CPU'en.

Takt-generator.

Taktgeneratoren (clockgeneratoren) leverer styreimpulser til CPU'en, og bestemmer dermed den tid det tager at udføre en instruktion. Typisk er frekvensen mellem 2oo KHz og 5 MHz. Den nominelle frekvens er givet af den anvendte CPU.

Busser.

Som vist på systemdiagrammet kommunikerer de enkelte enheder via nogle fælles ledningsbundter. Et ledningsbundt som fører signaler med samme mission kaldes en "bus".
Adressebussen består af typisk 16 og databussen af 8 parallelle ledninger. I steder for at tegne alle disse ledninger i diagrammet bruges ofte at tegne en enkelt streg og angive antallet af ledere, eller tegne et bredt bånd og med klar tekst angive arten og antallet af ledninger.

Med pile kan man angive retningen af signalerne.

Adressebussen.

Når CPU'en under eksekveringen af et program skal have tilført en ny opcode fra ROM'en eller data fra RAM'en eller inputporten, sender den via adressebussen et bitmønster ud, hvorved den ønskede memorylokation adresseres.

Som vist i systemdiagrammet bliver adressen tilført parallelt til både ROM, RAM, INPUT- og OUTPUTPORT, men kun en af blokkene reagerer, da ingen adresse kan (må) gå igen hos flere blokke, hvilket varetages af chip-select decoderen.

Databussen.

Via databussen overføres data! Data kan være:

Opcoder fra ROM'en til CPU'en	MR eller IOR
Faste data fra ROM'en til CPU'en	MR eller IOR
Variable data fra I-port til CPU'en	LOW
Variable data fra RAM'en til CPU'en	MR eller IOR
Data fra CPU'en til RAM'en	MW eller IOW
Data fra CPU'en til O-porten	LOW

Som det kan ses kan flere enheder sende signaler over databussen. For at undgå forstyrrelser er det nødvendigt at indrette kredsløbet således, at der kun er en af gangen, der sender, medens alle andre enten lytter eller er koblet ud. Den sidste mulighed betegnes "den tredie stilling", hvorfor man kalder denne form for logik "3-state logik", eller "TRI-STATE logik".

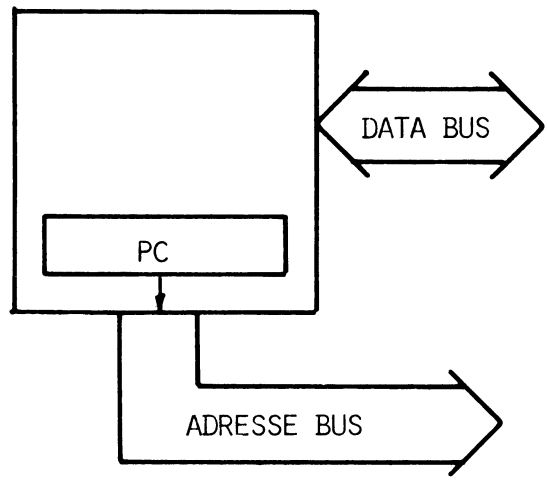
Kontrolbussen.

Kontrolbussen omfatter bl.a. clockpulser og læse/skrive kontrollene, MR. MW, IOR og IOW.

Læse/skrive signalerne kommer fra CPU'en og har til opgave at styre RAM-memoryen og I/O portene.

CPU. - Simpel blokdiagram.

CPU'en har til opgave at styre afviklingen og udføringen af et program. Til styringen har CPU'en en programtæller (PC). Denne tæller holder styr på hvor langt CPU'en er nået i et program. Hver gang programtælleren anvendes, forhøjes den automatisk med een. Ved hjælp af specielle instruktioner og/eller hardware signaler kan programtællerens værdi blive udskiftet med en anden værdi. Efter reset indholder PC værdien øøøø. PC'en er normalt den der leverer signalet til adressebussen, d.v.s. at programtællerens værdi er lig med en adresse. Adressebussen er der hvor CPU'en fortæller hvorfra den vil læse (Read) eller skrive (Write) data, der transporteres via databussen.

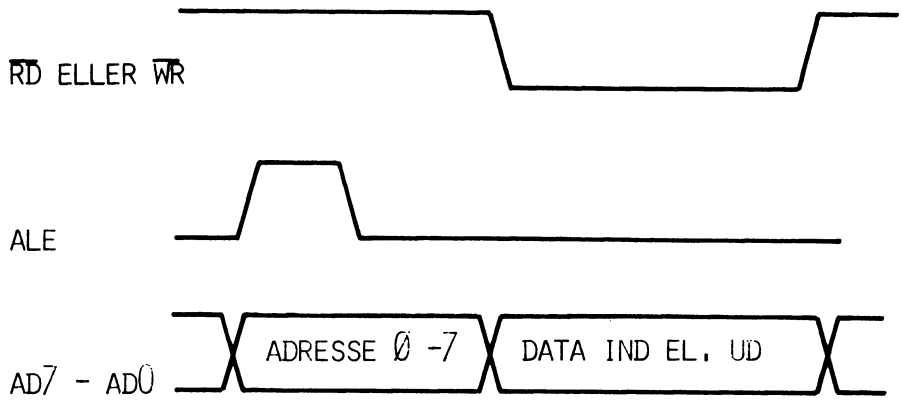


RW, WR.

For at fortælle omgivelserne om CPU'en vil læse eller skrive data, har den nogle kontrol signaler, dør ikke kun fortæller retningen af data på databussen, men angiver også hvornår data henholdsvis er eller skal være stabile. Disse styreledninger hedder RD (read) og WR (write).

Multiplexning.

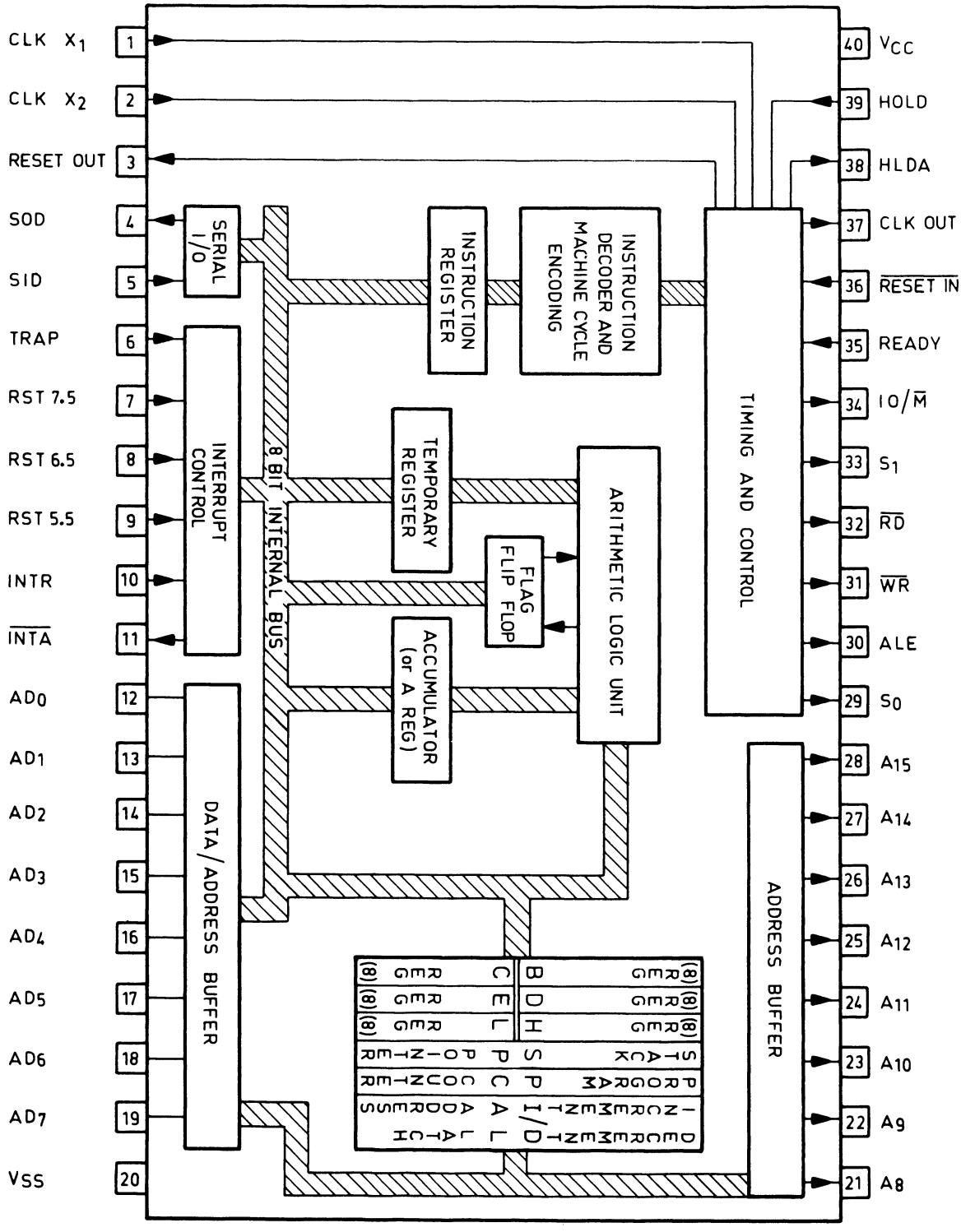
I 8085 er PC og adressebus 16 bit bred, databussen er 8 bit bred. De 8 mindst betydende adressebit og data bit'ene er ført ud på de samme 8 ben på IC'en og bliver tidsmultiplexet ud. For at fortælle hvornår disse 8 ben har adresse information, findes der et ekstra ben, der fortæller hvornår adresseinformationen er stabil, dette ben hedder ALE. Data angives stabile med $\overline{WR}$ signalet og skal være på databussen, når $\overline{RD}$ signalet indikerer det. D.v.s. ALE fortæller hvornår de 8 laveste adressebit er på adresse/databussen og $\overline{RD}/\overline{WR}$ angiver, hvornår CPU'en har eller ønsker at få data på adresse/databussen.



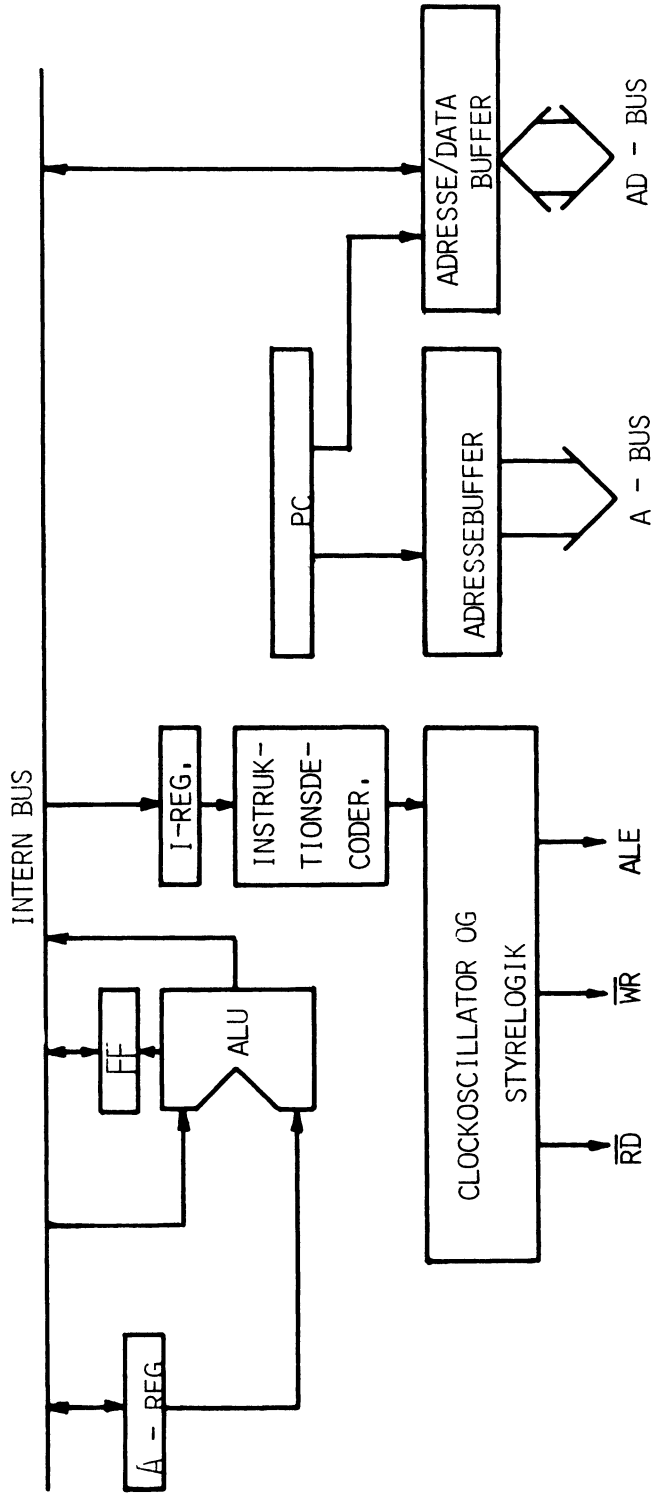
Instruktionsfetch

Første data som CPU'en henter (fetsch) efter reset anbringer CPU'en i et instruktionsregister, hvor CPU'en vil tyde bitmønsteret v.h.a. instruktionsdekoderen. Bitmønsteret fortæller CPU'en om der er tale om en 1 byte, 2 bytes eller 3 bytes instruktion. CPU'en starter nu det til bitmønsteret hørende program.

INTEL 8085 LAYOUT



SIMPELT BLOKDIAGRAM AF CPU, 8085.



Microprogram.

Microprogrammet er det program der internt i CPU'en, får den til at afvikle hver instruktion som CPU'en kan udføre.

Når CPU'en har hentet og udført en instruktion, vil den igen hente et nyt bitmønster ind i instruktions-registeret. Den første byte i en instruktion kaldes opcoden. Opcoden er den byte, der skal anbringes i instruktionsregisteret.

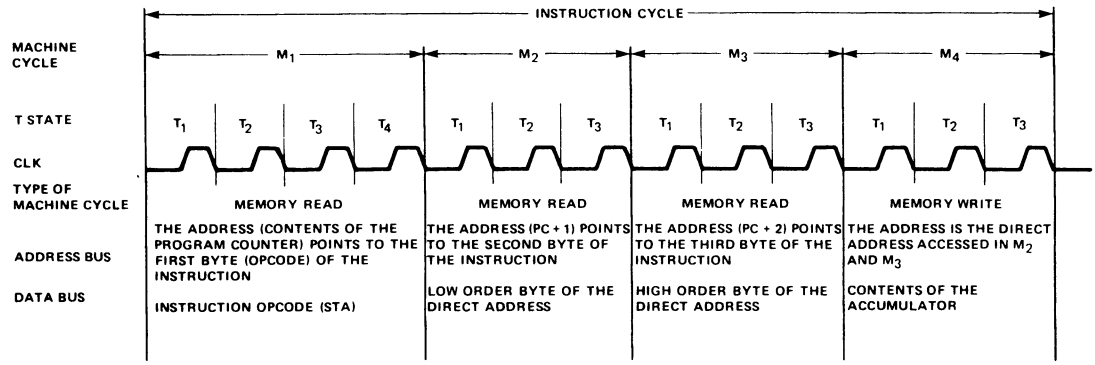
CPU'ens arbejdshastighed og timing er styret af en oscillator. For alle CPU'er findes der en maksimal arbejdsfrekvens, ligeledes findes for en hel del CPU'er også en minimum frekvens.

Interne Reg.

Når data skal andet end flyttes, anvender CPU'en en ALU (arithmetic logic unit),hertil hører også et arbejdsregister der kaldes akkumulatoren (A-reg.) samt nogle FF'ere, der fortæller noget om det resultat, der sidst er beregnet af ALU'en.

Instruktionscyclus.

En instruktioncyclus består af hentning (Fetsch) af en instruktion og udføring (execute) af instruktionen. Instruktionscyclen opdeles i nogle (1 til 5) maskincycler. Antallet af maskincycler svarer til det antal gange CPU'en har brug for at hente eller anbringe data i den eksterne memory. Hver maskincyclus kan igen deles ned i fra 3 til 6 state (trin), der hver især svarer til en tid på en intern clockperiode. For en 8085 er den interne clockfrekvens lig med halvdelen af den frekvens der tilsluttes x_1 og x_0.
Benet CLK out på 8085 er lig med CPU'ens interne clockfrekvens, blot forsinket nogle nanosekunder.



Maskincyclus.

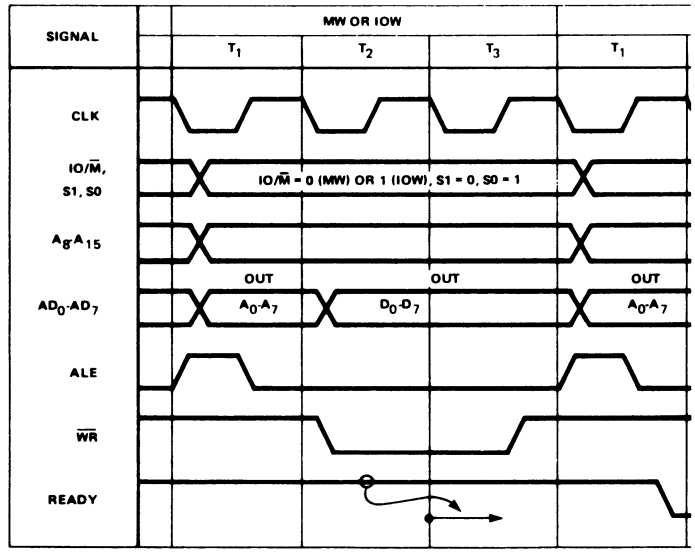
Til hver instruktion hører et bestemt antal og type maskincycler. Med benene S_1, S_o og IO/M angiver 8085 hvilken maskincyclus type der er igang med at bearbejde. Der er 7 forskellige maskincyclus typer, der angives på følgende måde:

IO/$\overline{M}$	S_1	S_o	Maskin cycle type
T.S.	Ø	Ø	Halt
Ø	Ø	1	Memory write
Ø	1	Ø	Memory read
Ø	1	1	Opcode fetch
1	Ø	1	I/O Write
1	1	Ø	I/O Read
1	1	1	Intr. Acknowledge

T.S. = Tri state

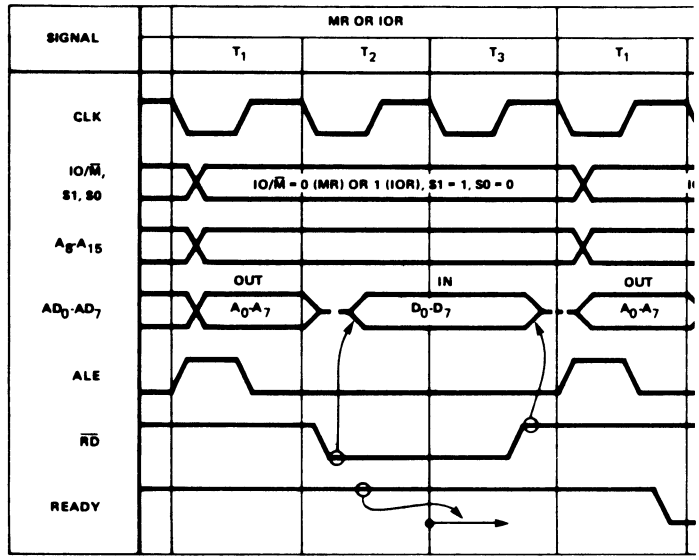
MW, IOW.

En Memory eller IO write (MW, IOW) maskin cycle består af 3 state. I første state udsender CPU'en den adresse, hvor data skal anbringes. I state 2 udsendes data og WR ledningen går low. I state 3 udsendes data og WR ledningen går high. Under alle tre state udsendes S_1= Ø, $S_Ø$=1 samt IO/$\overline{M}$ informationen der er "1" ved IO og "O" ved M.



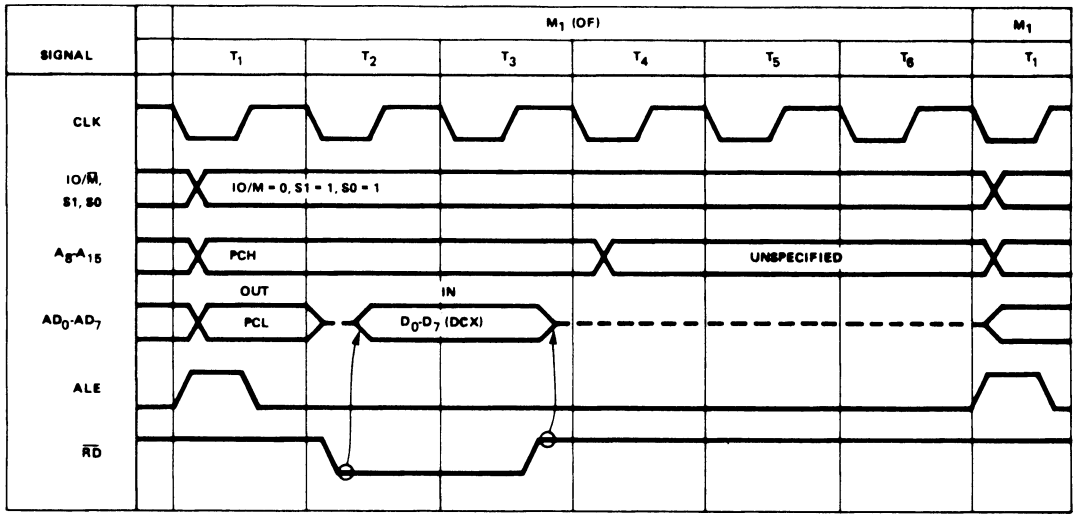
MR, IOR.

En Memory eller IO read (MR, IOR) har også 3 state men afviger fra MW ved at det er RD og ikke WR der går low under T_2 og high igen under T_3. Ligeledes er signalerne på $S_\emptyset$ og S_1 ændret til $S_\emptyset = \emptyset$ og $S_1 = 1$.

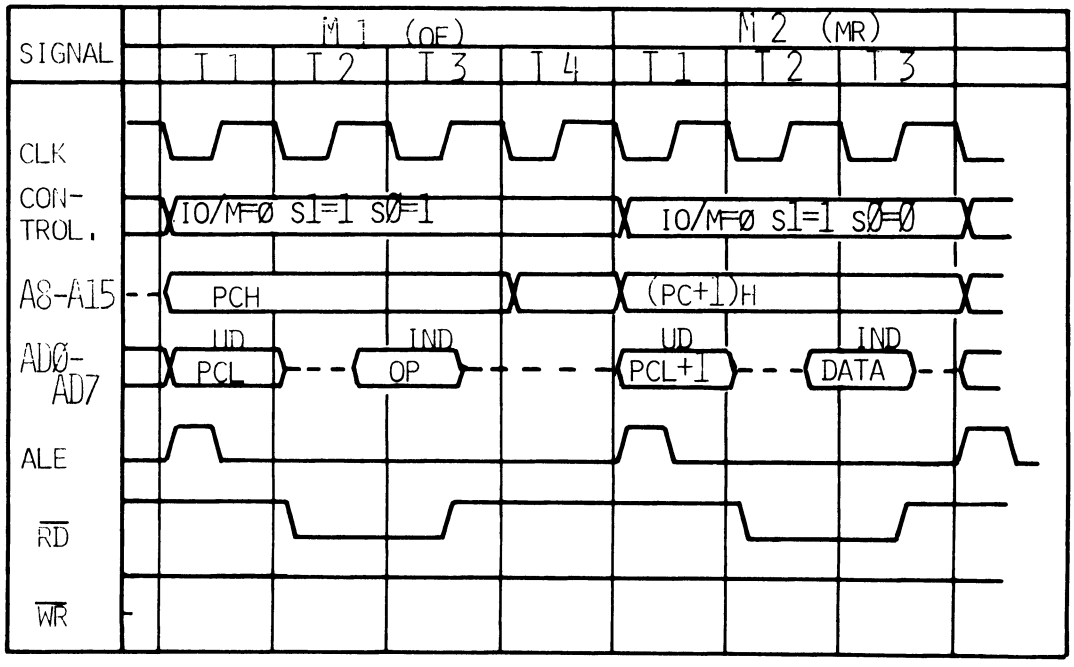


OF.

I maskincyclen opcode fetch (OF) anvendes der fra 4 til 6 state, de første tre state anvendes til at læse opcoden, det efterfølgende state anvendes til at tyde opcoden, hvorunder CPU'ens microprogram afgør om der er brug for yderligere to state for at udføre instruktionen eller en del af instruktionen efterfulgt af andre maskincycler.

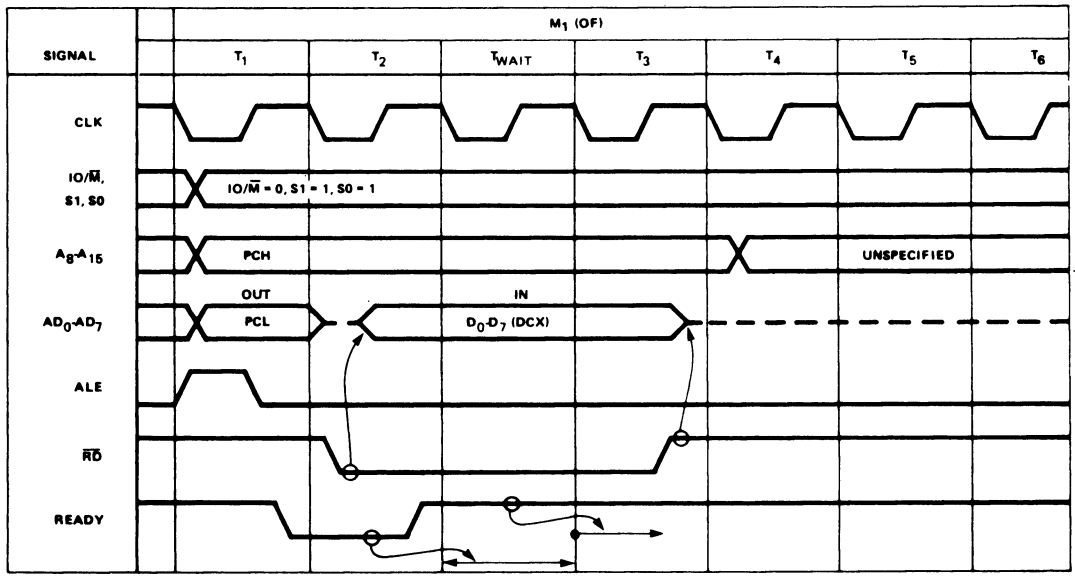


En instruktion som ANI DATA vil bestå af to maskincycler, en OF og en MR.

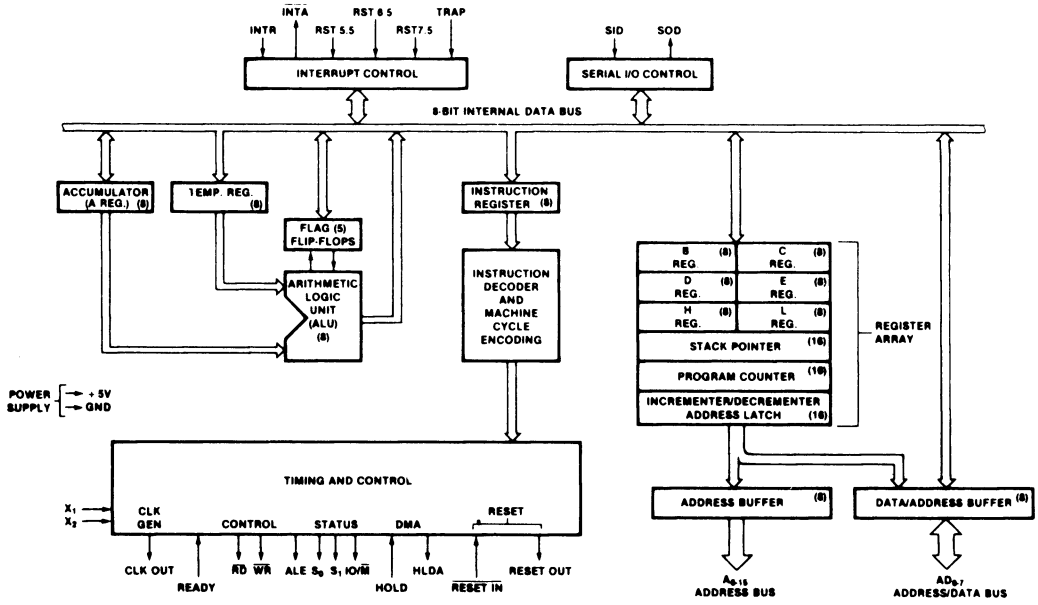


Twait.

Når CPU'en skal arbejde sammen med langsomme memory eller portkredsløb, kan der mellem T_2 og T_3 indskydes nogle ventetrin (Twait). For at T_3 indledes skal READY benet være high inden den positive clock out flanke, der går forud for indledningen af T_3.



Et blokdiagram af 8085's struktur ser således ud:



CPU'en består af tolv adresserbare 8 bit registre. Fire af disse arbejder altid som to 16 bit registre. Det er PC (Program Counter) og SP (Stack Pointer). Seks andre registre kan valgfrit for programmøren anvendes som 8 bit eller 16 bits registerpar (rp), disse registre hedder som 8 bits registre B, C, D, E, H og L og som 16 bits registre BC, DE og HL. Disse registre anvendes universelt. HL registret kan dog bruges som datapointer, d.v.s. HL kan indeholde et 16 bit mønster, der er et udtryk for den adresse, hvor data skal hentes eller placeres. Et andet 8 bit register, der anvendes som resultatregister ved aritmetiske, logiske og in/out instruktioner er akkumulatoren (A-reg.). Det sidste 8 bit register er flag registeret, i dette register anvendes kun de fem af bit'ene. Ved aritmetiske og logiske funktioner påvirkes bit'ene, idet hver bit for sig angiver noget om resultatet.

Interrupt.

I en 8085 er der to forskellige måder, hvormed CPU'en kan interruptes. Den ene er ved anvendelse af INTR og $\overline{\text{INTA}}$ benene. Når et eksternt kredsløb ønsker at påkalde sig opmærksomheden, genererer det et logisk "1" på INTR benet. Et "1" på dette ben vil få CPU'en til at stoppe PC, hvis processoren er enablet (gjort mulig) for interrupt (EI). Derefter vil der indledes en interrupt acknowledge (erkendelse) maskincyclus. Dette ligner en OF maskin cyclus blot med den forskel, at benet $\overline{\text{INTA}}$ anvendes i stedet for $\overline{\text{RD}}$ benet. D.v.s. at der nu skal genereres en opcode på databussen timet v.h.a. af $\overline{\text{INTA}}$ signalet, denne opcode kan være RST instruktion bestående af 1 byte eller en CALL bestående af 3 byte, der skal genereres i takt med $\overline{\text{INTA}}$ signalet. Med RST instruktionerne starter CPU'en på forud bestemte startadresser, der er indeholdt i opcoden.

Instruktion	Startadresse
RST Ø	ØØØØ H
RST 1	ØØØ8 H
RST 2	ØØ1Ø H
RST 3	ØØ18 H
RST 4	ØØ2Ø H
RST 5	ØØ28 H
RST 6	ØØ3Ø H
RST 7	ØØ38 H

Med CALL instruktion kan der startes på enhver adresse i systemet.

Den anden måde at generere interrupt er med TRAP, RST 5.5, RST 6.5, RST 7.5 benene. Ved signal på et af disse ben vil CPU'en interruptes og derefter selv generere en RST instruktion uden hjælp af eksterne kredsløb. Forskellen på disse ben er, hvor CPU'en vil starte restart rutinen.

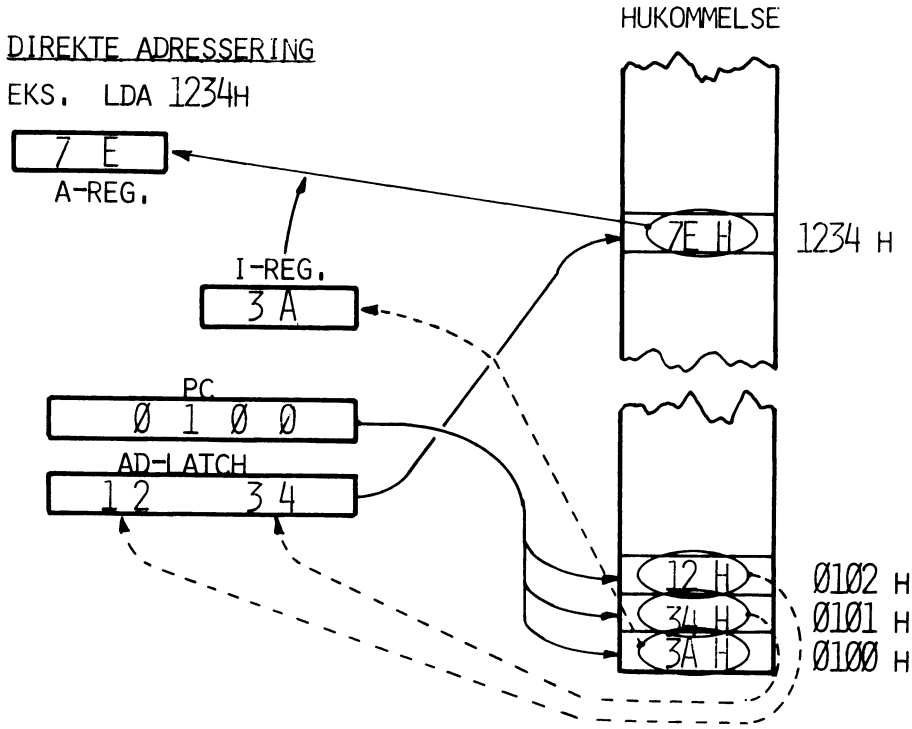
Ben	Startadresse
TRAP	ØØ24 H
RST 5.5	ØØ2C H
RST 6.5	ØØ34 H
RST 7.5	ØØ3C H

Der er ligeledes også den forskel at TRAP ikke kan blokeres med DI eller SIM instruktionerne.

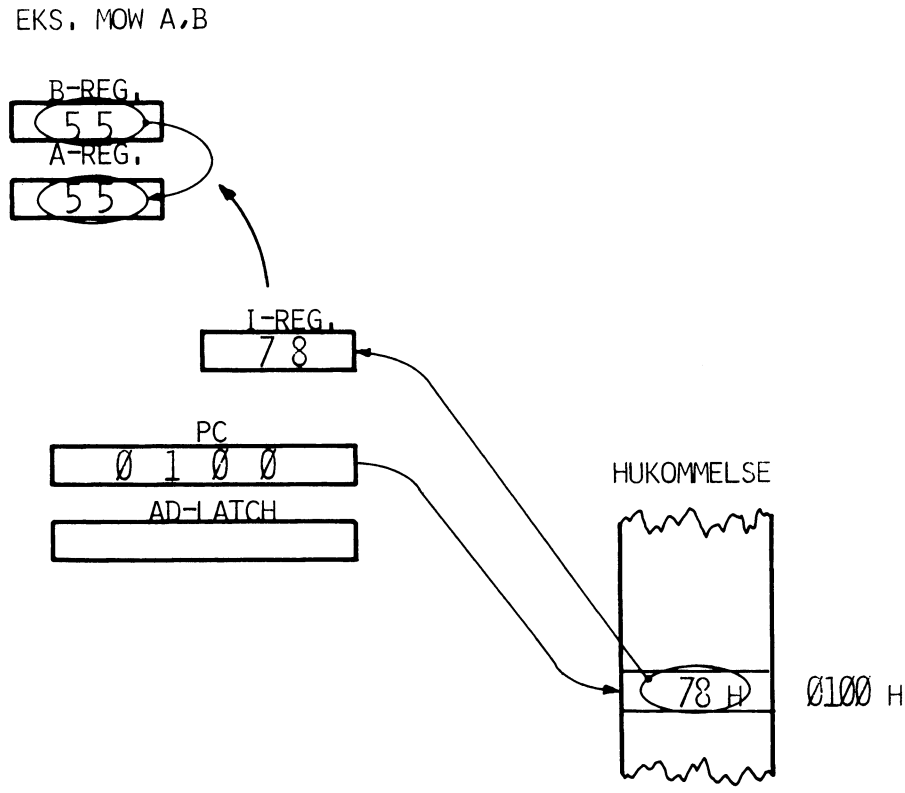
Adresseringsmåder.

8085'en har fire måder til adressering af data i lageret eller i registrene:

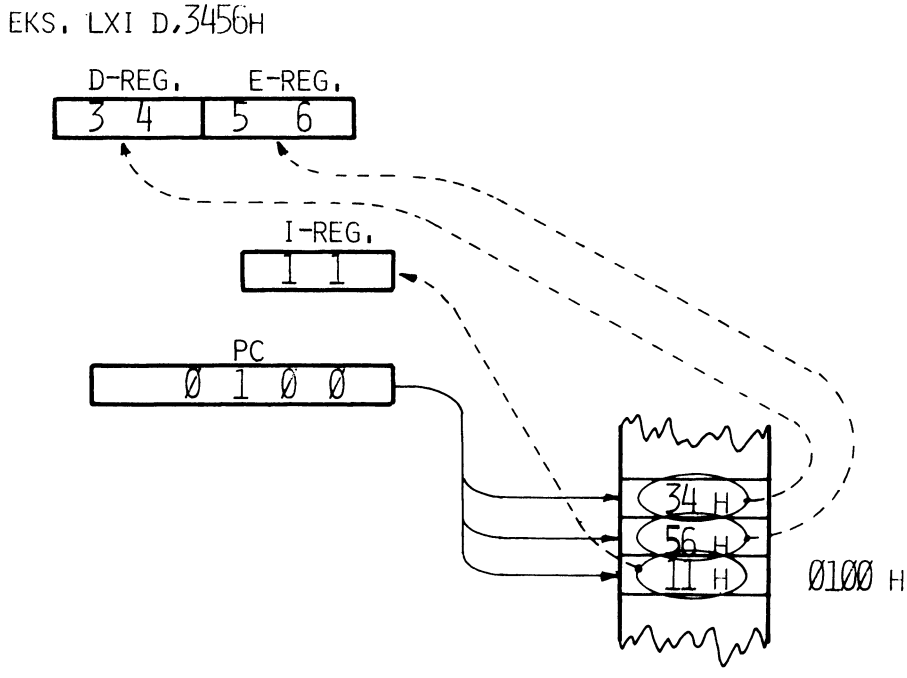
Ved DIREKTE ADRESSERING bruges 2 byte til selve adressen, således at byte 2 er de 8 mindst betydende bits af adressen og byte 3 de 8 mest betydende bits.



Ved REGISTER-ADRESSERING angiver selve op-koden det register, der skal benyttes til operationen.

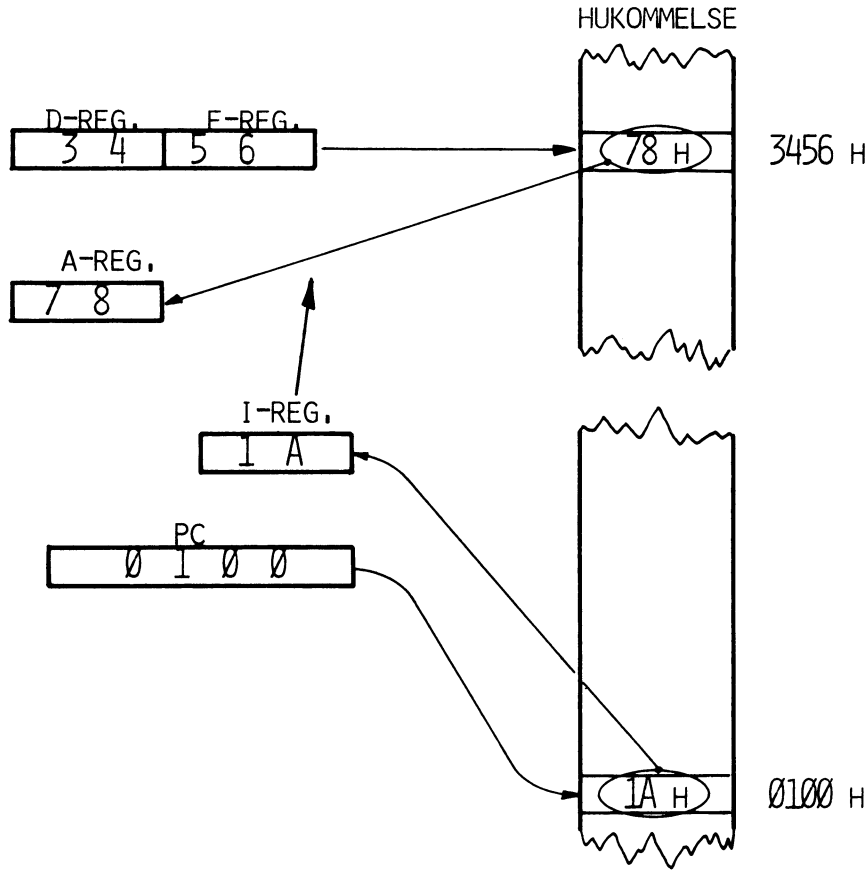


Ved "IMMEDIATE" ADRESSERING indeholder instruktionen også data. Instruktionslængden er 2 eller 3 bytes.



Ved REGISTER INDIREKTE ADRESSERING angiver selve op-koden det registerpar, f.eks. (H.L.), som indeholder adressen for data, der skal benyttes ved operationen. Register H indeholder den mest betydende byte af adressen.

EKS. LDAX D



Når et program køres, udføres instruktionerne i den rækkefølge, de står i programmet, med mindre dette ændres på grund af en hop-instruktion eller et interrupt-signal. En hop-instruktion angiver den adresse, der skal hoppes til på to måder, nemlig:

Ved DIREKTE HOP-ADRESSERING bruges 2 bytes til adresse, som ved andre direkte adresserende instruktioner.

Ved REGISTER INDIREKTE HOP-ADRESSERING angiver op-koden det registerpar, der indeholder hop-adressen.

RST-instruktionen er en speciel hop-instruktion, der anvendes ved interrupt af programmet. Instruktionen er på 8 bits, hvoraf tre bruges som adressefelt. Dette felts værdi ganges automatisk med 8, og programmet udføres fra den adresse, der svarer til dette tal.

STATUS - FLAG.

Der er fem resultatvisende flag, der virker i forbindelse med udførelsen af instruktionerne i 8085'en. De kaldes henholdsvis ZERO, SIGN, PARITY, CARRY og AUXILIARY CARRY. De kan indeholde "1" eller "0" afhængig af, om de er i stilling SET eller RESET.

Når en instruktion påvirker flags, gælder følgende regler, medmindre andet er specificeret i specielle tilfælde.

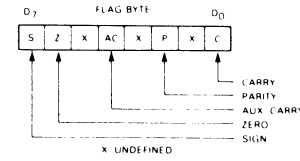
ZERO: Hvis en instruktion giver resultatet 0, sættes ZERO-FF'en, i modsat fald resættes den.

SIGN: Får samme værdi som den mest betydende bit i akkumulatoren.

PARITY: Får værdien 1 ved lige paritet for det tal, der står i akkumulatoren.

CARRY: Hvis en instruktion giver en mente (ved addition) eller en låner (ved subtraktion eller sammenligning) sættes Carry-FF'en i stilling 1, 1 modsat fald i stilling 0.

AUXILIARY CARRY: Virker som Carry for de 4 mindst betydende bit. (Anvendes oftest ved BCD-regning).



SYMBOLER OG FORKORTELSER.

I det efterfølgende forklares symboler og forkortelser, der anvendes ved beskrivelse af 8085'erers instruktionssæt.

Symbol	Betydning
Akkumulator	Register A
Adr.	16-bits adresse
Data	8-bits data størrelse
Data 16	16-bit data størrelse
Byte 2	Byte 2 i en instruktion
Byte 3	Byte 3 i en instruktion
Port	8-bit adresse for en I/O komponent
r, r1, r2	Et af registrene A,B,C,D,E,H,L

DDD, SSS Et bitmønster der henviser til et af registrene A, B. C. D. E. H, L idet

DDD står for modtagerregister (destination)

SSS står for det register, der hentes data fra (source).

DDD eller SSS	REGISTER NAVN
111	A
000	B
001	C
010	D
011	E
100	H
101	L

rp Står for et af følgende registerpar, idet det mest betydende er anført forrest. BC, DE, HL og SP, der står for stackpointer registeret (16 bits).

RP Et bit-mønster, der henviser til et af registerparene B, D, H, SP:

RP	Registerpar
00	B-C
01	D-E
10	H-L
11	SP

rh Det første register i et registerpar, f.eks. B,D,H.

rl Det andet register i et registerpar, f.eks. C,E,L.

PC 16-bits program counter register. For den mest betydende del af PC anvendes PCH (high), og for den mindst betydende del PCL (low).

SP 16-bit stack-pointer register. SPH pg SPL anvendes her ligesom ved PC.

r_m Bit m i et register f.eks. den 5'te bit noteres r_5. (mest betydende længst til venstre).

Z,S,P,
CY,AC Flagene: ZERO, ZIGN, PERITY, CARRY og AUX.CARRY.

( ) Noteres f.eks. (H), menes indholdet af register H.

(( )) Noteres f.eks. ((H)), menes indholdet i lageret på den adresse H peger på.

A Et tal i RST instruktionen fra o-7 (Dec.).

AAA Et bitmønster på 3 bit fra OOO - lll.

DATA TRANSPORT-GRUPPE.

Instruktionerne i denne gruppe transporterer data til og fra registre og lager.
FLAG-FF'erne påvirkes IKKE af instruktionerne i denne gruppe.

MOV rl, r2 (Move register)

(rl) ⟵ (r2).
Indholdet af register r2 kopieres ind i register rl.

O	1	D	D	D	S	S	S

Maskincycles: 1
Antal clockpulser: 4
Adressering: register (skal angives)
Påvirkede flag: ingen.

MOV r,M (Move from memory to register)

(r) ⟵ ((H) (L)).

Registre H og L indeholder adressen på den lagerlokation, hvis indhold kopieres ind i register.

O	1	D	D	D	1	1	O

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H.
Påvirkede flag: ingen.

MOV, M,r (Move from register to memory)

((H) (L)) ◄—— (r)

Register r's indhold kopieres over i lageret på den adresse, der står i registrene H og L.

0	1	1	1	0	S	S	S

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H.
Påvirkede flag: ingen.

MVI r,data (Move immediate)

(r) ◄—— (byte 2)

Indholdet af byte 2 i instruktionen kopieres ind i register r.

0	0	D	D	D	1	1	0
data							

Maskincycles: 2
Antal clockpulser: 7
Adressering: data hører til instruktionen.
Påvirkede flag: ingen.

MVI M, data (Move memory immedate)

((H) (L)) ◄—— (byte 2).

Indholdet af byte 2 i instruktionen kopieres over i lageret på den adresse, der står i registrene H og L.

0	0	1	1	0	1	1	0
data							

Maskincycles: 3
Antal clockpulser: lo
Adressering: data hører til instruktionen og indirekte via registerpar H
Påvirkerede flag: ingen

LXI rp, data 16 (Load register pair immediate)

(rl) ⟵ (byte 2)
(rh) ⟵ (byte 3)

Byte 2 i instruktionen kopieres ind i registerpar rp's mindst betydende del, og byte 3 i den mest betydende del.

O	O	R	P	O	O	O	1
low-order data							
high-order data							

Maskincycles: 3
Antal clockpulser: lo
Adressering: data hører til instruktionen
Påvirkede flag: ingen.

LDA adr (Load akkumulator direct).
(A) ⟵ ((byte 3) (byte 2)).

Byte 3 og byte 2 indeholder adressen på den lagerlokation, hvis indhold kopieres ind i akkumulatoren.

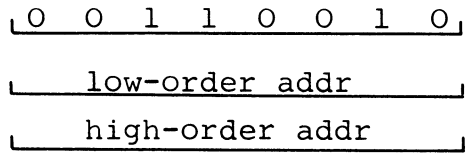
O	O	1	1	1	O	1	O
low-order adr.							
high-order adr.							

Makincycles: 4
Antal clockpulser: 13
Adressering: direkte
Påvirkede flag: ingen.

STA adr (Store akkumulator direct)

((byte3) (byte 2)) ← A

Akkumulatorens indhold kopieres over i lageret på den adresse, der angives af byte 3 og byte 2 i instruktionen.



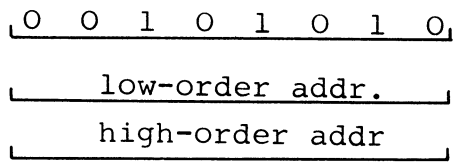
Maskincycles: 4
Antal clockpulser: 13
Adressering: direkte
Påvirkede flag: ingen.

LHLD adr (Load H and L direct)

(L) ← ((byte 3) (byte 2))

(H) ← ((byte 3) (byte 2) + 1)

Indholdet i lageret med den adresse, byte 3 og byte 2 peger på, kopieres ind i register L, og dets indhold med adressen 1 højere kopieres ind i register H.



Maskincycles: 5
Antal clockpulser: 16
Adressering: direkte
Påvirkede flag: ingen

SHLD (Store H and L direct).
((byte 3) (byte 2)) ⟵ (L)
((byte 3) (byte 2) +1)) ⟵ (H).

Indholdet i register L kopieres over i lageret på den adresse, byte 3 og byte 2 peger på.
Register H's indhold kopieres over i lageret på den efterfølgende adresse.

O	O	1	O	O	O	1	O

low-order addr.

high-order addr

Maskincycles: 5
Antal clockpulser: 16
Adressering: direkte
Påvirkede flag ingen.

LDAX rp (Load accumulator indirect).
(A) ⟵ ((rp))
Indholdet i lageret på den adressen, registerpar rp peger på, kopieres ind i akkumulatoren. Kun registerpar B (B og C) og D (D og E) må anvendes.

O	O	R	P	1	O	1	O

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar rp.
Påvirkede flag: ingen.

STAX rp (Store accumulator indirect).

((rp)) ← A

Akkumulatorens indhold kopieres over i lageret på den adresse, registerpar rp indeholder.
Kun registerpar B (B og C) og D (D og E) må anvendes.

O	O	R	P	O	O	1	O

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar rp.
Påvirkede flag: ingen.

XCHG (Exchange H and L with D and E).

(H) ⟷ (D)
(L) ⟷ (E)

Indholdet i registrene H og L bytter plads med indholdet i registrene D og E.

1	1	1	O	1	O	1	1

Maskincycles: 1
Antal clockpulser: 4
Adressering: register (skal ikke angives)
Påvirkede flag: ingen.

ARITMETISKE GRUPPE.

Instruktionerne i denne gruppe udfører aritmetiske operationer på data hentet fra registre og lageret.

FLAGENE PÅVIRKES, med mindre andet er anført. Ved udførelsen af instruktioner, der benytter subtraktion, anvendes 2-komplement regning, idet CARRY-FLAGET benyttes til låneregister (borrow).

ADD r (Add register to accumulator)

(A) ← (A) + (r).

Indholdet i register r adderes til indholdet i akkumulatoren. Resultatet placeres i akkumulatoren.

1	0	0	0	0	S	S	S

Maskincycles: 1
Antal clockpulser: 4
Adressering: register (skal angives).
Påvirkede flag: alle.

ADD M (Add memory to accumulator)

(A) ← (A) + ((H) (L))

Indholdet i lageret på den adresse registrene H og L peger på, lægges til indholdet i akkumulatoren. Resultatet placeres i akkumulatoren.

1	0	0	0	0	1	1	0

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H
Påvirkede flag: alle

ADI data (Add immediate to accumulator)

(A) ← (A) + (byte 2)

Indholdet i byte 2 af instruktionen lægges til akkumulatoren. Carry-FF'en deltager ikke i additionen, men påvirkes af resultatet.
Resultatet placeres i akkumulatoren.

1	1	0	0	0	1	1	0
data							

Maskincycles: 2
Antal clockpulser: 7
Adressering: immediate
Påvirkede flag: alle.

ADC r (Add register to accumulator with carry).

(A) ← (A) + (r) + (CY)

Indholdet i register r og carry FF'en lægges til indholdet i akkumulatoren. Resultatet placeres i akkumulatoren.

1	0	0	0	1	S	S	S

Maskincycles: 1
Antal clockpulser: 4
Adressering: register (skal angives)
Påvirkede flag: alle.

ADC M (Add memory to accumulator with carry)

(A) ← (A) +((H) (L)) + (CY)

Indholdet i lageret på den adresse, registrene H og L peger på og carry FF'en lægges til indholdet i akkumulatoren. Resultatet placeres i akkumulatoren.

1	0	0	0	1	1	1	0

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H

ACI data (Add immediate to accumulator with carry).

(A) ← (A)+(byte 2) + (CY)

Instruktionens byte 2 og carry-FF'ens indhold lægges til akkumulatoren. Resultatet placeres i akkumulatoren.

1	1	0	0	1	1	1	0
data							

Maskincycles: 2
Antal clockpulser: 7
Adressering: data hører til instruktionen.
Påvirkede flag: alle.

SUB r (Subtract register from accumulator)

(A) ← (A) - (r)

Indholdet i register r trækkes fra akkumulatorens indhold. Resultatet placeres i akkumulatoren.

1	0	0	1	0	S	S	S

Makincycles: 1
Antal clockpulser: 4
Adressering: register (skal angives).
Påvirkede flag: alle.

SUB M (Subtract memory fra accumulator)

(A) ← (A) - ((H) (L))

Indholdet i lageret på den adresse, registrene H og L peger på, trækkes fra akkumulatoren. Resultatet placeres i akkumulatoren.

1	0	0	1	0	1	1	0

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H.
Påvirkede flag: alle.

SUI data (Subtract immediate from accumulator).

(A) ← (A) - (byte 2)

Instruktionens byte 2 og indholdet af akkumulatoren subtraheres. Resultatet placeres i akkumulatoren.

1	1	0	1	0	1	1	0
data							

Maskincycles: 2
Antal clockpulser: 7
Adressering: immediate
Påvirkede flag: alle.

SBB r(Subtract register and borrow from accumulator).

(A) ← (A) - (r) - (CY)

Indholdet i register r og carry-FF'en trækkes fra indholdet i akkumulatoren. Resultatet placeres i akkumulatoren.

1	0	0	1	1	S	S	S

Maskincycles: 1
Antal clockpulser: 4
Adressering: register (skal angives).
Påvirkede flag: alle.

SBB M (Subtract memory and borrow from accumulator)

(A) ← (A) - ((H) (L)) - (CY)

Indholdet i lageret på den adresse, registrene H og L peger på, og CY-FF'en trækkes fra indholdet i akkumulatoren. Resultatet placeres i akkumulatoren.

1	0	0	1	1	1	1	0

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H
Påvirkede flag: alle.

SBI data (Subtract with borrow from accumulator immediate).

(A) ← (A) - (byte 2) - (CY)

Instruktionens byte 2 og carry-FF'ens indhold trækkes fra indholdet i akkumulatoren. Resultatet placeres i akkumulatoren.

1	1	0	1	1	1	1	0
data							

Maskincycles: 2
Antal clockpulser: 7
Adressering: immediate
Påvirkede flag: alle.

INR r (Increment register)

(r) ← (r) + 1.

Indholdet i register r øges med 1.

NB: CY-FF'en påvirkes ikke.

O	O	D	D	D	1	O	O

Maskincycles: 1
Antal clockpulser 4
Adressering: register (skal angives)
Påvirkede flag: Z, S, P, AC.

INR M (Increment memory).

((H) (L)) ← ((H) (L)) + 1.

Indholdet i lageret på den adresse, registrene H og L peger på øges med værdien 1.

NB: CY-FF'en påvirkes ikke.

O	O	1	1	O	1	O	O

Maskincycles: 3
Antal clockpulser: lo
Adressering: indirekte via registerpar H
Påvirkede flag: Z, S, P, AC.

DCR (Decrement register).

(r) ← (r) - 1.

Indholdet i register r mindskes med værdien 1.

NB: CY-FF'en påvirkes ikke.

O	O	D	D	D	1	O	1

Maskincycles: 1
Antal clockpulser: 4
Adressering: register (skal angives)
Påvirkede flag: Z, S, P, AC.

DCR M (Decrement memory)

((H) (L)) ← ((H) (L)) - 1.

Indholdet i lageret på den adresse, registrene H og L peger på, mindskes med værdien 1.

NB: CY'FF'en påvirkes ikke.

0	0	1	1	0	1	0	1

Makincycles: 3
Antal clockpulser: 10
Adressering: indirekte via registerpar H
Påvirkede flag: Z, S, P, AC.

INX rp (Increment register pair)

(rh) (rl) ← (rh) (rl) + 1.

Indholdet i registerpar rp øges med værdien 1.
NB: ingen flag påvirkes.

0	0	R	P	0	0	1	1

Maskincycles: 1
Antal clockpulser: 6
Adressering: registerpar (skal angives).
Påvirkede flag: ingen.

DCX rp (Decrement register pair)

(rh) (rl) ← (rh) (rl) - 1.
Indholdet i registerpar rp mindskes med værdien 1.
NB: Ingen flag påvirkes.

0	0	R	P	1	0	1	1

Maskincycles 1
Antal clockpulser: 6
Adressering: registerpar (skal angives)
Påvirkede flag: ingen.

DAD rp (Add register pair to H and L)

(H) (L) ← (H) (L) + (rh) (rl).

Indholdet i registerpar rp lægges til registerpar H. Resultatet placeres i registerpar H.

NB: Kun CY-FF'en påvirkes, og kun hvis en mente opstår som den 17'ende bit.

O	O	R	P	1	O	O	1

Maskincycles: 3
Antal clockpulser: lo
Adressering: registerpar (skal angives)
Påvirkede flag: CY.

DAA (Decimal adjust accumulator).

Akkumulatorens 8-bit indhold justeres til 2 4-bit binærkodede decimaltal efter følgende regler:

1. 6 lægges til akkumulatoren, hvis den binære værdi for de 4 mindst betydende bit er større end 9 eller hvis AC-flaget indeholder værdien 1.

2. 6 lægges til akkumulatorens 4 mest betydende bit, hvis deres binære værdi er større end 9 eller hvis CY-flaget indeholder værdien 1.

NB: Alle flag påvirkes.

O	O	1	O	O	1	1	1

Maskincycles: 1
Antal clockpulser: 4
Adressering: ingen
Påvirkede flag: alle.

LOGISKE GRUPPE:
Instruktionerne i denne gruppe foretager binære logiske operationer på data fra registrene, flagene og lageret. Flagene påvirkes med mindre andet er anført.

ANA r (AND register with accumulator).
(A) ← (A) . (r).

Indholdet i akkumulatoren og i register r AND'es logisk med hinanden (ensplacerede bit benyttes). Resultatet placeres i akkumulatoren.
NB: CY'FF'en 0-stilles og AC-FF'en 1-stilles.

1	0	1	0	0	S	S	S

Maskincycles: 1
Antal clockpulser: 4
Adressering: register (skal angives)
Påvirkede flag: alle.

ANA M (AND memory with accumulator)
(A) ← (A) . ((H) (L)).

Indholdet i lageret på den adresse registrene H og L peger på og akkumulatorens AND'es logisk med hinanden (ensplacerede bit benyttes).
Resultatet placeres i akkumulatoren.
NB: CY-FF'en 0-stilles, og AC-FF'en 1-stilles.

1	0	1	0	0	1	1	0

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H
Påvirkede flag: Alle.

ANI data (AND with accumulator immediate).
(A) ⟵ (A) . (byte 2).

Instruktionens byte 2 og akkumulatorens indhold AND'es logisk med hinanden (ensplacerede bit benyttes). Resultatet placeres i akkumulatoren.

NB: CY-FF'en 0-stilles og AC-FF'en 1-stilles.

1	1	1	0	0	1	1	0
data							

Maskincycles: 2
Antal clockpulser: 7
Adressering: immediate.
Påvirkede flag: alle.

XRA r (Exclusive OR register with accumulator).
(A)⟵(A) ⊕ (r).

Indholdet i register r og akkumulatoren bliver EXOR'et logisk med hinanden (ensplacerede bit benyttes) Resultatet placeres i akkumulatoren.

NB: CY- og AC-FF'erne 0-stilles.

1	0	1	0	1	S	S	S

Maskincycles: 1
Antal clockpulser 4
Adressering: register (skal angives)
Påvirkede flag: alle.

XRA M (Exclusive OR memory with accumulator).
(A) ⟵ (A) ⊕ ((H) (L)).

Indholdet i lageret på den adresse, registrene H og L peger på og akkumulatoren bliver EXOR'et logisk med hinanden (ensplacerede bit benyttes).

Resultatet placeres i akkumulatoren.
NB: CY- og AC-FF'erne 0-stilles.

1	0	1	0	1	1	1	0

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H.
Påvirkede flag: alle.

XRI data (Exclusive OR with accumulator immediate).
(A) ⟵ (A) ⊕ (byte 2).

Instruktionens byte 2 og akkumulatorens indhold EXOR'es logisk sammen (ensplacerede bit benyttes).
Resultatet placeres i akkumulatoren.

NB: CY- og AC FF'erne O-stilles.

1	1	1	0	1	1	1	0
data							

Maskincycles: 2
Antal clockpulser: 7
Adressering: immediate.
Påvirkede flag: alle.

ORA r (OR register with accumulator).
(A) ⟵ (A) + (r).

Indholdet i register r og akkumulatoren bliver OR'et logisk med hinanden (ensplacerede bit benyttes)
Resultatet placeres i akkumulatoren.

NB: CY- og AC-FF'erne O-stilles.

1	0	1	1	0	S	S	S

Maskincycles: 1
Antal clockpulser: 4
Adressering: register (skal angives)
Påvirkede flag: alle.

ORA M (OR memory with accumulator)

(A) ⟵ (A) + ((H) (L)).

Indholdet i lageret på den adresse, registrene H og L peger på, og akkumularoren bliver OR'et logisk med hinanden (ensplacerede bit benyttes).
Resultatet placeres i akkumulatoren.

NB: CY- og AC-FF'erne O-stilles.

1	0	1	1	0	1	1	0

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H.
Påvirkede flag: alle.

ORI data (OR with accumulator immediate)
(A) ← (A) + (byte 2).

Instruktionens byte 2 og akkumulatorens indhold bliver OR'et logisk med hinanden (ensplacerede bit benyttes). Resultatet placeres i akkumulatoren.

NB: CY- og AC-FF'erne 0-stilles.

1	1	1	1	0	1	1	0
data							

Maskincycles: 2
Antal clockpulser: 7
Adressering: imediate.
Påvirkede flag: alle.

CMP r (Compare register with accumulator).
CY, Z: (A) - (r).

Indholdet i register r trækkes fra akkumulatoren. Akkumulatorens indhold forbliver uændret.
Hvis (A) = (r) fås Z-FF'en = 1
Hvis (A) < (r) fås CY-FF'en = 1

1	0	1	1	1	S	S	S

Maskincycles: 1
Antal clockpulser: 4
Adressering: register (skal angives)
Påvirkede flag: alle.

CMP M (Compare memory with accumulator)

CY, Z: (A) - ((H) (L)).

Indholdet i lageret på den adresse, registrene H og L peger på, trækkes fra akkumulatorens indhold. Akkumulatorens indhold forbliver uændret.

Hvis (A) = ((H) (L)) fås Z-FF'en = 1
Hvis (A) < ((H) (L)) fås CF-FF'en = 1

1	0	1	1	1	1	1	0

Maskincycles: 2
Antal clockpulser: 7
Adressering: indirekte via registerpar H.
Påvirkede flag: alle.

CPI data (compare with accumulator immediate).
CY, Z: (A) - (byte 2).

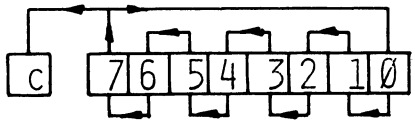
Instruktionens byte 2 trækkes fra akkumulatoren. Akkumulatorens indhold forbliver uændret.

Hvis (A) = (byte 2) fås Z-FF'en = 1
Hvis (A) < (byte 2) fås CY-FF'en = 1.

1	1	1	1	1	1	1	0
data							

Maskincycles: 2
Antal clockpulser: 7
Adressering: immediate
Påvirkede flag: alle.

RLC (Rotate accumulator left).

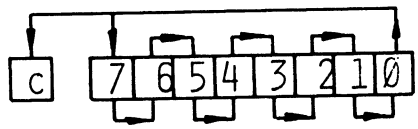


Indholdet i akkumulatoren skiftes én position til venstre, som angivet ovenfor.
NB: kun CY-FF'en påvirkes.

0	0	0	0	0	1	1	1

Maskincycles 1
Antal clockpulser: 4
Påvirkede flag: CY

RRC (Rotate accumulator right).



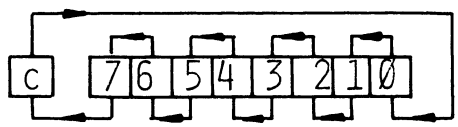
Indholdet i akkumulatoren skiftes én position til højre, som angivet ovenfor.

NB: kun CY-FF'en påvirkes.

0	0	0	0	1	1	1	1

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: CY.

RAL (Rotate accumulator left througt carry).

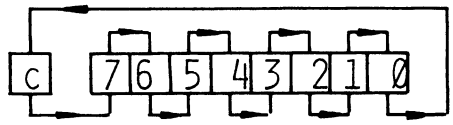


Indholdet i akkumulatoren skiftes én position til venstre igennem CY-FF'en, som angivet ovenfor.
NB: Kun CY-FF'en påvirkes.

0	0	0	1	0	1	1	1

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: CY.

RAR (Rotate accumulator right through carry).



Indholdet i akkumulatoren skiftes én position til højre igennem CY-FF'en, som angivet ovenfor.
NB: Kun CY-FF'en påvirkes.

0	0	0	1	1	1	1	1

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: CY.

CMA (Complement accumulator)
$(A) \leftarrow \overline{(A)}$.
Alle bit i akkumulatoren komplementeres.
NB: Ingen flag påvirkes.

0	0	1	0	1	1	1	1

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: ingen.

CMC (Complement carry)
$(CY) \leftarrow \overline{(CY)}$
CY-FF'ens indhold komplementeres.
NB: Kun CY-FF'en påvirkes.

0	0	1	1	1	1	1	1

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: CY.

STC (Set carry).

(CY) ← 1.

CY-FF'en får indholdet 1.

NB: kun CY-FF'en påvirkes.

0	0	1	1	0	1	1	1

Maskincycles: 1

Antal clockpulser: 4

Påvirkede flag: CY.

HOP-GRUPPE.

Instruktionerne i denne gruppe ændrer normal sekventiel programgennemløb. Flagene påvirkes ikke af instruktionerne i denne gruppe.

Der findes betingede og ubetingede hop-instruktioner.

Ubetingede instruktioner ændrer programtællerens (PC) indhold direkte.

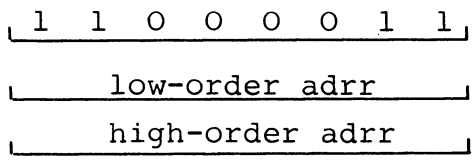
Betingede hop-instruktioner ændrer kun programtællerens indhold, hvis en bestemt tilstand er til stede i status flagene.

Tilstandene er vist herunder:

FLAG-FF	TILSTAND		CCC
NZ	Z = 0		000
Z	Z = 1		001
NC	CY = 0		010
C	CY = 1		011
PO	P = 0	(ulige par.)	100
PE	P = 1	(lige par.)	101
P	S = 0	(plus)	110
M	S = 1	(minus)	111

JMP adr (Jump unconditionally).
(PC)←(byte 3) (byte 2).

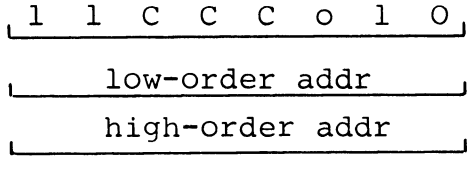
Programkontrollen overføres til den adresse instruktionens byte 2 og byte 3 angiver.



Maskincycles: 3
Antal clockpulser: lo
Adressering: immediate
Påvirkede flag: ingen.

J condition adr. (Jump conditionally).
Hvis (CCC) fås (PC)← (byte 3) (byte 2)

Hvis den angivne tilstand (CCC) er til stede, overføres programkontrollen til den adresse, der er anført i instruktionens byte 2 og byte 3, i modsat fald fortsætter programmet sekventielt.



Maskincycles: 2/3,
Antal clockpulser: 7/lo
Adressering: immediate
Påvirkede flag: ingen.

CALL adr (Call unconditionally).
((SP) - 1)← (PCH)
((SP) - 2)← (PCL)
(SP) ← (SP) - 2
(PC) ← (byte 3) (byte 2)

Den mest betydende del af næste instruktionsadresse overføres til lageret på den adresse, stackpointeren minus 1 peger på. Den mindst betydende del af næste instruktionsadresse overføres til lageret på den adresse, stackpointeren minus 2 peger på. Stackpointerens indhold mindskes med værdien 2. Programkontrollen overføres til den adresse, byte 2 og byte 3 peger på.

1	1	0	0	1	1	0	1
low-order adrr							
high-order adrr							

Maskincycles: 5
Antal clockpulser: 18
Adressering: immediate og register indirekte.
Påvirkede flag: ingen.

C condition adr. (Call conditionally.)

Hvis (CCC) fås
((SP) - 1)←(PCH)
((SP) - 2)←(PCL)
(SP) ← (SP) - 2
(PC) ← (byte 3) (byte 2).

Hvis tilstanden (CCC) er til stede, sker det samme som anført ved CALL-instruktionen, i modsat fald fortsætter programmet sekventielt.

1	1	C	C	C	1	0	0
low-order addr							
high-order addr							

Maskincycles: 2/5
Antal clockpulser: 9/18
Adressering: adresse hører til instruktionen og indirekte via register SP
Påvirkede flag: ingen.

RET (Return unconditionally)

(PLC) ← ((SP))

(PCH) ← ((SP) + 1)

(SP) ← (SP) + 2.

Indholdet i lageret på den adresse, register SP peger på, kopieres over i programtællerens mindst betydende del. Indholdet i lageret på den adresse, register SP plus 1 peger på, kopieres over i programtællerens mest betydende del. Registeret SP's indhold øges med værdien 2.

1	1	0	0	1	0	0	1

Maskincycles: 3

Antal clockpulser: 1o

Adressering: indirekte via register SP

Påvirkede flag: ingen.

R condition (Return conditionally).

Hvis (CCC)

(PCL) ← ((SP))

(PCH) ← ((SP) + 1)

(SP) ← (SP) + 2.

Hvis tilstanden (CCC) er tilstede, sker det samme som anført ved RET-instruktionen, i modsat fald fortsætter programmet sekventielt.

1	1	C	C	C	0	0	0

Maskincycles: 1/3

Antal clockpulser: 6/12

Adressering: indirekte via registeret SP.

Påvirkede flag: ingen.

RST a (Restart).

((SP) - 1) ← (PCH)

((SP) - 2) ← (PCL)

(SP) ← (SP) - 2

(PC) ← 8 x (AAA)

Denne instruktion virker meget lig CALL, som vist ovenfor. Programtælleren tilføres værdien A gange 8, og programmet fortsætter fra den adresse, der derved opstår.

1	1	A	A	A	1	1	1

Maskincycles: 3
Antal clockpulser: 12
Adressering: adressen hører til instruktionen.
Påvirkede flag: ingen.

Programtælleren indeholder følgende værdi efter en RST n instruktion.

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

PCHL (Load Program counter with H and L)

(PCH) ← (H)
(PCL) ← (L)

Indholdet i registerpar H kopieres over i programtælleren.

1	1	1	0	1	0	0	1

Maskincycles: 1
Antal clockpulser: 6
Adressering: register (skal ikke angives)
Påvirkede flag: ingen.

STACK - I/O og MASKIN-KONTROL-GRUPPE.

Instruktionerne i denne gruppe udfører input/output operationer, påvirker stack'en og kan ændre indholdet i flag-FF'erne.

FLAGENE påvirkes ikke med mindre andet er anført.

PUSH rp (Push registerpair onto stack).

((SP) - 1) ← (rh)

((SP) - 2) ← (rl)

(SP) ← (SP) - 2.

Indholdet af registerpar rp kopieres over i stacken, idet den mest betydende byte placeres på adressen, der fås ved at trække en fra SP-registeret. Den mindst betydende del af registerparret placeres i stack'en på adressen SP-2. SP'erens indhold mindskes med værdien 2.

NB: Register SP MÅ IKKE ANFØRES for rp.

1	1	R	P	0	1	0	1

Maskincycles: 3

Antal clockpulser: 12

Adressering: indirekte via register SP og registerpar (skal angives).

PUSH PSW (Push processor status word onto stack)

((SP) - 1) ← (A)

((SP) - 2) ← S Z 0 AC 0 P L CY

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

(SP) ← (SP) - 2.

Instruktionen virker meget lig den foregående instruktion PUSH, idet indholdet i akkumulatoren og status FF'erne kopieres over i lageret, hvor stacken er placeret. Dette angiver SP. SP'erens indhold mindskes med værdien 2.

1	1	1	1	0	1	0	1

Maskincycles: 3

Antal clockpulser: 12

Adressering: indirekte via register SP

Påvirkede flag: ingen.

POP rp (Pop off stack to register pair)

(rl) ← ((SP))

(rh) ← ((SP) + 1)

(SP) ← (SP) + 2.

Med denne instruktion fås den modsatte virkning af en PUSH rp. Stackens indhold kopieres tilbage til et registerpar. SP'erens indhold øges med værdien 2.

NB: Register SP MÅ IKKE ANFØRES for rp.

1	1	R	P	0	0	0	1

Maskincycles	3
Antal clockpulser	lo
Adressering	indirekte via register SP og registerpar (skal angives).
Påvirkede flag:	ingen.

POP PSW (Pop processor status word off stack to registers).

Instruktionen har modsat virkning af PUSH PSW, idet den genskaber den tilstand i akkumulatoren og status-FF'erne, der eksisterede før PUSH PSW instruktionen blev udført.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
S	Z	0	AC	0	P	1	CY	← ((SP))

(A) ← ((SP) + 1)

(SP) ← (SP) + 2.

1	1	1	1	0	0	0	1

Maskincycles:	3
Antal clockpulser:	lo
Adressering:	indirekte via register SP
Påvirkede flag:	alle.

XTHL (exchange stack top with H and L).

(L) ⟷ ((SP))

(H) ⟷ ((SP) + 1)

Indholdet i stack'en på den adresse, SP peger på, og register L ombyttes.
Indholdet i stack'en på den adresse, SP + 1 peger på, og register H ombyttes.

1	1	1	0	0	0	1	1

Maskincycles: 5
Antal clockpulser: 16
Adressering: indirekte via register SP.
Påvirkede flag: ingen.

SPHL (Move H and L to stack pointer).

(SP) ← (H) (L)

Indholdet i registrene H og L (16 bit) kopieres ind i stackpil registeret.

1	1	1	1	1	0	0	1

Maskincycles: 1
Antal clockpulser: 6
Adressering: register (skal ikke angives).
Påvirkede flag: ingen.

IN port. (Input to accumulator)

(A) ← (data)

Data placered på data-busén fra en inputport kopieres ind i akkumulatoren.

1	1	0	1	1	0	1	1
port							

Maskincycles: 3
Antal clockpulser: lo
Adressering: direkte (kun 8 bit)
Påvirkede flag: ingen.

OUT port (Output from accumulator)

(data) ← (A)

Data placeres på data-busén fra akkumulatoren, kopieres ud i en output-port.

1	1	0	1	0	0	1	1
port							

Maskincycles: 3
Antal clockpulser: lo
Adressering: direkte (kun 8 bit)
Påvirkede flag: ingen.

EI (Enable interrupt).

Interrupt-systemet gøres aktivt, så interrupt kan lade sig gøre efter næste instruktion.

1	1	1	1	1	0	1	1

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: ingen.

DI (Disable interrupt).

Interrupt-systemet gøres inaktivt umiddelbart efter instruktionen DI.

1	1	1	1	0	0	1	1

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: ingen.

HLT (Halt and enter wait state)

Processoren stoppes.
Flag-FF'erne påvirkes ikke.

0	1	1	1	0	1	1	0

Maskincycles: 1
Antal clockpulser: 5
Adressering: ingen.

NOP (No operation)

Processoren udfører intet.
Flag-FF'erne påvirkes ikke.

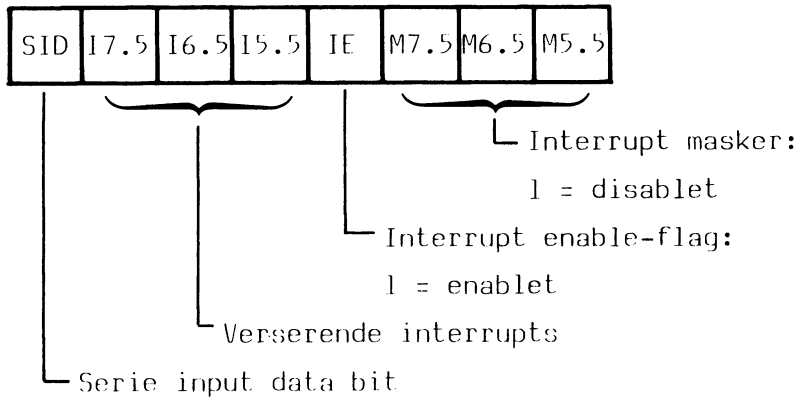
0	0	0	0	0	0	0	0

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: ingen

RIM (Read interrupt mask).

RIM instruktionen indlæser 8 databit i akkumulatoren. Det resulterende bit-mønster viser den øjeblikkelige stilling for interrupt-masken, interrupt-flagets stilling, om der er ventende interrupts og en serieinput-bit, hvis der er nogen på ledningen SID.

RIM instruktionen indlæser følgende bitmønster i akkumulatoren:



Masken og ventende (pending) flag refererer kun til RST 5.5, RST 6.5 og RST 7.5 interruptindgangene. IE-flaget refererer til hele interruptsystemet. IE-flaget virker ligesom INTE-indgangen på 8o8o'eren.
Eb 1-bit i dette flag indikerer, at hele interruptsystemet er aktivt.

O	O	1	O	O	O	O	O

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: ingen.

SIM (set interrupt mask).

SIM er en instruktion til flere funktioner. Den anvender akkumulatorens øjeblikkelige indhold til at udføre følgende funktioner:

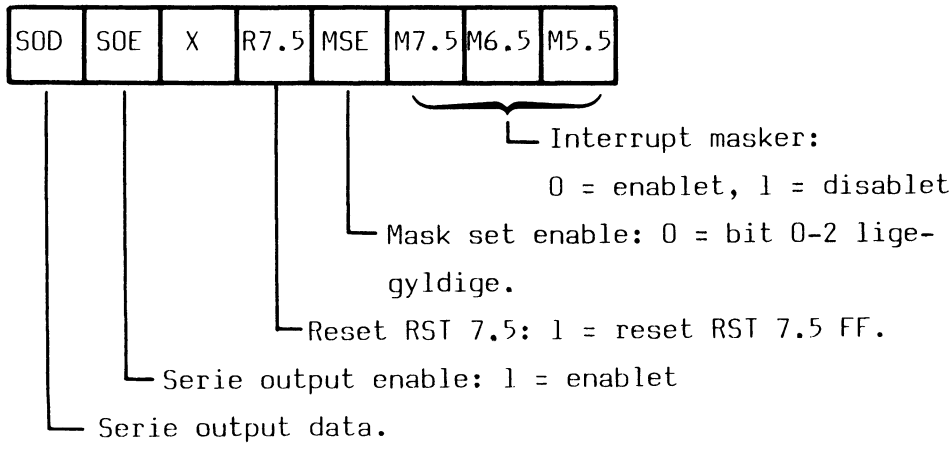
Indstiller interruptmasken for 8o85'erens RST 5.5, RST 6.5 og RST 7.5 interruptindgangen.

Resætter RST 7.5 (edge-triggered indgang).

Sender bit 7 i akkumulatoren ud til serieoutput-data-latch'en (flip-flop).

Man skal være sikker på, at det rigtige bitmønster er indlæst i akkumulatoren, før SIM-instruktionen udføres.

SIM instruktionen opfatter bittene i akkumulatoren, som vist nedenfor:



Akkumulatorens bit 3 og 6 virker som aktiveringsbit. Hvis bit 3 er 1, er indstilling af masken aktiveret. Bit 0, 1 og 2 kan nu maske (spærre for) eller tillade de respektive RST-indgange at virke. Er bit 0 f.eks. 1, spærres for RST 5.5-indgangen (og omvendt). Hvis bit 3 er 0, har bit 0, 1 og 2 ingen virkning. Dette kan anvendes, hvis man ønsker at sende seriedata ud uden at påvirke RST maskningen.

Bemærk at DI-(afbryd interrupt system) instruktionen gør SIM-instruktionen virkningsløs. Uanset maskningen virker RST 5.5 - 6.5 og 7.5 ikke når en DI-instruktion er givet. Man kan anvende RIM-(læs interrupt maske) instruktionen til at bestemme den øjeblikkelige stilling af interrupt-flagene og interrupt-masken.

Hvis bit 6 er 1, er serie-data-output-funktionen aktiv. Processoren låser bit 7 fra akkumulatoren fast i en flip-flop forbundet til SOD udgangen, som en udvendig komponent kan få adgang til. Hvis bit 6 er 0, påvirkes flip-floppen ikke.
Er 1 i akkumulatorens bit 4 sætter RST 7.5-indgangens flip-flop. I RST 7.5-indgangen sidder nemlig en flip-flop, som påvirkes af en ydre komponent med et 0 1 skift. Dette gælder ikke for RST 5.5 og 6.5-indgangene.
RST 7.5's kanttriggede indgang anvendes til at huske signaler fra komponenter, der ikke selv kan fastholde signalet indtil RST 7.5 bliver udført.

RST 7.5 flip-floppen kan resettes af:

1) system-reset,
2) interrupt-signalet konstateres - eller
3) et 1 i akkumulatorens bit 4, og SIM bliver udført.

Det faktum, at en SIM-instruktion kan resette RST 7.5 flip-floppen, tillader, at programmet kan overse en interrupt.

RST 7.5 flip-floppen påvirkes ikke af, hvordan interrupt-masken er sat eller af DI-instruktionen, og kan derfor påvirkes hele tiden. Interrupten RST 7.5 kan dog ikke serviceres, når RST 7.5 er bortmasket eller DI-instruktionen er givet.

0 0 1 1 0 0 0 0

Maskincycles: 1
Antal clockpulser: 4
Påvirkede flag: ingen.

Eksempel 1:

Antag, at akkumulatoren indeholder bitmønsteret 00011100. SIM-instruktionen resetter RST 7.5-flip-floppen og setter RST 7.5-interrupt-masken. Hvis en RST 7.5-interrupt står og venter, når denne SIM-instruktion udføres, bliver RST 7.5 ikke udført. Efterfølgende RST 7.5-interruptsignaler er også masket bort og kan ikke blive serviceret, før interruptmasken bliver resat.

Eksempel 2:

Antag, at akkumulatoren indeholder bitmøsteret 11001111. SIM-instruktionen masker RST 5.5, 6.5 og 7.5-interruptmulighederne bort og låser en 1-bit ind i SOD-indgangen.

Modsat hertil vil bitmønsteret 10000111 ikke have nogen virkning, da aktiverings-bit 3 og 6 ikke er på 1.

8085A INSTRUCTION SET SUMMARY

Mnemonic	Description	Instruction Code(1) D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock(2) Cycles
MOVE, LOAD, AND STORE										
MOV r1,r2	Move register to register	0	1	D	D	D	S	S	S	4
MOV M,r	Move register to memory	0	1	1	1	0	S	S	S	7
MOV r,M	Move memory to register	0	1	D	D	D	1	1	0	7
MVI r	Move immediate register	0	0	D	D	D	1	1	0	7
MVI M	Move immediate memory	0	0	1	1	0	1	1	0	10
LXI B	Load immediate register Pair B & C	0	0	0	0	0	0	0	1	10
LXI D	Load immediate register Pair D & E	0	0	0	1	0	0	0	1	10
LXI H	Load immediate register Pair H & L	0	0	1	0	0	0	0	1	10
LXI SP	Load immediate stack pointer	0	0	1	1	0	0	0	1	10
STAX B	Store A indirect	0	0	0	0	0	0	1	0	7
STAX D	Store A indirect	0	0	0	1	0	0	1	0	7
LDAX B	Load A indirect	0	0	0	0	1	0	1	0	7
LDAX D	Load A indirect	0	0	0	1	1	0	1	0	7
STA	Store A direct	0	0	1	1	0	0	1	0	13
LDA	Load A direct	0	0	1	1	1	0	1	0	13
SHLD	Store H & L direct	0	0	1	0	0	0	1	0	16
LHLD	Load H & L direct	0	0	1	0	1	0	1	0	16
XCHG	Exchange D & E, H & L Registers	1	1	1	0	1	0	1	1	4
STACK OPS										
PUSH B	Push register Pair B & C on stack	1	1	0	0	0	1	0	1	12
PUSH D	Push register Pair D & E on stack	1	1	0	1	0	1	0	1	12
PUSH H	Push register Pair H & L on stack	1	1	1	0	0	1	0	1	12
PUSH PSW	Push A and Flags on stack	1	1	1	1	0	1	0	1	12
POP B	Pop register Pair B & C off stack	1	1	0	0	0	0	0	1	10
POP D	Pop register Pair D & E off stack	1	1	0	1	0	0	0	1	10
POP H	Pop register Pair H & L off stack	1	1	1	0	0	0	0	1	10
POP PSW	Pop A and Flags off stack	1	1	1	1	0	0	0	1	10
XTHL	Exchange top of stack, H & L	1	1	1	0	0	0	1	1	16
SPHL	H & L to stack pointer	1	1	1	1	1	0	0	1	6
JUMP										
JMP	Jump unconditional	1	1	0	0	0	0	1	1	10
JC	Jump on carry	1	1	0	1	1	0	1	0	7/10
JNC	Jump on no carry	1	1	0	1	0	0	1	0	7/10
JZ	Jump on zero	1	1	0	0	1	0	1	0	7/10
JNZ	Jump on no zero	1	1	0	0	0	0	1	0	7/10
JP	Jump on positive	1	1	1	1	0	0	1	0	7/10
JM	Jump on minus	1	1	1	1	1	0	1	0	7/10
JPE	Jump on parity even	1	1	1	0	1	0	1	0	7/10
JPO	Jump on parity odd	1	1	1	0	0	0	1	0	7/10
PCHL	H & L to program counter	1	1	1	1	0	1	0	0	1	6
CALL										
CALL	Call unconditional	1	1	0	0	1	1	0	1	18
CC	Call on carry	1	1	0	1	1	1	0	0	9/18
CNC	Call on no carry	1	1	0	1	0	1	0	0	9/18
CZ	Call on zero	1	1	0	0	1	1	0	0	9/18
CNZ	Call on no zero	1	1	0	0	0	1	0	0	9/18
CP	Call on positive	1	1	1	1	0	1	0	0	9/18
CM	Call on minus	1	1	1	1	1	1	0	0	9/18
CPE	Call on parity even	1	1	1	0	1	1	0	0	9/18
CPO	Call on parity odd	1	1	1	0	0	1	0	0	9/18
RETURN										
RET	Return	1	1	0	0	1	0	0	1	10
RC	Return on carry	1	1	0	1	1	0	0	0	6/12
RNC	Return on no carry	1	1	0	1	0	0	0	0	6/12
RZ	Return on zero	1	1	0	0	1	0	0	0	6/12
RNZ	Return on no zero	1	1	0	0	0	0	0	0	6/12
RP	Return on positive	1	1	1	1	0	0	0	0	6/12
RM	Return on minus	1	1	1	1	1	0	0	0	6/12
RPE	Return on parity even	1	1	1	0	1	0	0	0	6/12
RPO	Return on parity odd	1	1	1	0	0	0	0	0	6/12
RESTART										
RST	Restart	1	1	A	A	A	1	1	1	12
INPUT/OUTPUT										
IN	Input	1	1	0	1	1	0	1	1	10
OUT	Output	1	1	0	1	0	0	1	1	10

Mnemonic	Description	Instruction Code(1) D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Clock(2) Cycles
INCREMENT AND DECREMENT										
INR r	Increment register	0	0	D	D	D	1	0	0	4
DCR r	Decrement register	0	0	D	D	D	1	0	1	4
INR M	Increment memory	0	0	1	1	0	1	0	0	10
DCR M	Decrement memory	0	0	1	1	0	1	0	1	10
INX B	Increment B & C registers	0	0	0	0	0	0	1	1	6
INX D	Increment D & E registers	0	0	0	1	0	0	1	1	6
INX H	Increment H & L registers	0	0	1	0	0	0	1	1	6
INX SP	Increment stack pointer	0	0	1	1	0	0	1	1	6
DCX B	Decrement B & C	0	0	0	0	1	0	1	1	6
DCX D	Decrement D & E	0	0	0	1	1	0	1	1	6
DCX H	Decrement H & L	0	0	1	0	1	0	1	1	6
DCX SP	Decrement stack pointer	0	0	1	1	1	0	1	1	6
ADD										
ADD r	Add register to A	1	0	0	0	0	S	S	S	4
ADC r	Add register to A with carry	1	0	0	0	1	S	S	S	4
ADD M	Add memory to A	1	0	0	0	0	1	1	0	7
ADC M	Add memory to A with carry	1	0	0	0	1	1	1	0	7
ADI	Add immediate to A	1	1	0	0	0	1	1	0	7
ACI	Add immediate to A with carry	1	1	0	0	1	1	1	0	7
DAD B	Add B & C to H & L	0	0	0	0	1	0	0	1	10
DAD D	Add D & E to H & L	0	0	0	1	1	0	0	1	10
DAD H	Add H & L to H & L	0	0	1	0	1	0	0	1	10
DAD SP	Add stack pointer to H & L	0	0	1	1	1	0	0	1	10
SUBTRACT										
SUB r	Subtract register from A	1	0	0	1	0	S	S	S	4
SBB r	Subtract register from A with borrow	1	0	0	1	1	S	S	S	4
SUB M	Subtract memory from A	1	0	0	1	0	1	1	0	7
SBB M	Subtract memory from A with borrow	1	0	0	1	1	1	1	0	7
SUI	Subtract immediate from A	1	1	0	1	0	1	1	0	7
SBI	Subtract immediate from A with borrow	1	1	0	1	1	1	1	0	7
LOGICAL										
ANA r	And register with A	1	0	1	0	0	S	S	S	4
XRA r	Exclusive OR register with A	1	0	1	0	1	S	S	S	4
ORA r	OR register with A	1	0	1	1	0	S	S	S	4
CMP r	Compare register with A	1	0	1	1	1	S	S	S	4
ANA M	And memory with A	1	0	1	0	0	1	1	0	7
XRA M	Exclusive OR memory with A	1	0	1	0	1	1	1	0	7
ORA M	OR memory with A	1	0	1	1	0	1	1	0	7
CMP M	Compare memory with A	1	0	1	1	1	1	1	0	7
ANI	And immediate with A	1	1	1	0	0	1	1	0	7
XRI	Exclusive OR immediate with A	1	1	1	0	1	1	1	0	7
ORI	OR immediate with A	1	1	1	1	0	1	1	0	7
CPI	Compare immediate with A	1	1	1	1	1	1	1	0	7
ROTATE										
RLC	Rotate A left	0	0	0	0	0	1	1	1	4
RRC	Rotate A right	0	0	0	0	1	1	1	1	4
RAL	Rotate A left through carry	0	0	0	1	0	1	1	1	4
RAR	Rotate A right through carry	0	0	0	1	1	1	1	1	4
SPECIALS										
CMA	Complement A	0	0	1	0	1	1	1	1	4
STC	Set carry	0	0	1	1	0	1	1	1	4
CMC	Complement carry	0	0	1	1	1	1	1	1	4
DAA	Decimal adjust A	0	0	1	0	0	1	1	1	4
CONTROL										
EI	Enable Interrupts	1	1	1	1	1	0	1	1	4
DI	Disable Interrupt	1	1	1	1	0	0	1	1	4
NOP	No operation	0	0	0	0	0	0	0	0	4
HLT	Halt	0	1	1	1	0	1	1	0	5
NEW 8085A INSTRUCTIONS										
RIM	Read Interrupt Mask	0	0	1	0	0	0	0	0	4
SIM	Set Interrupt Mask	0	0	1	1	0	0	0	0	4

NOTES: 1. DDS or SSS: B 000, C 001, D 010, E 011, H 100, L 101, Memory 110, A 111.
2. Two possible cycle times. (6/12) indicate instruction cycles dependent on condition flags.

Microcomputerens lagersektion.

Lageret (hukommelsen) i en µ-computer har følgende opgaver:

1. Det skal indeholde programmet, også efter en strømafbrydelse.
2. Det skal kunne modtage, opbevare og aflevere data fra processen.

Programhukommelsen:

Til den 1. type hukommelse anvendes en af følgende tre typer kredse:

ROM (Read only memory)
PROM (Programable read only memory)
EPROM (Erasable programable read only memory)

ROM.

ROM'en anvendes normalt kun som programhukommelse, hvor der er tale om et stort antal ensartede udstyr, idet programmeringen foregår hos IC-fabrikanten som et led i fremstillingsprocessen.

PROM.

PROM'en minder i sin opbygning om ROM'en, men har ved leveringen det samme logiske niveau på samtlige bitpositioner. Dette kan ændres af brugeren, idet han ved at programmere hukommelsen kan ændre det logiske niveau på de ønskede steder. Ved programmeringen sker der en fysisk ændring i kredsen (en forbindelse smelter), hvorfor processen er irevercibel.

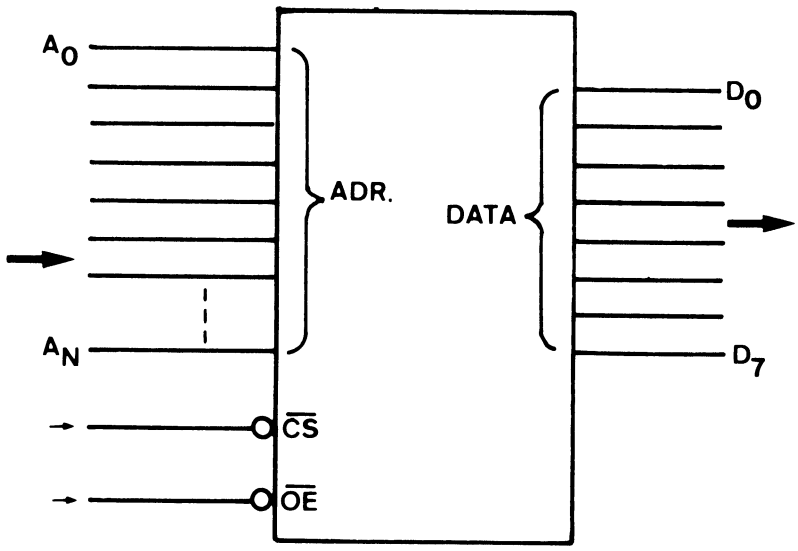
EPROM.

Til enkeltudstyr og mindre serier er EPROM'en enerådende som programhukommelse. Ved programmeringen, som foretages af brugeren på et egnet udstyr, sker der ingen fysiske ændringer, men blot en opladning af de hukommelsesceller (MOS-transistorer) hvis logiske indhold skal ændres.

Hvis det skulle vise sig nødvendigt, kan hukommelsens indhold slettes ved at belyse chippen med en passende ultraviolet lyskilde, hvorefter hukommelsen igen kan programmeres.

Programhukommelsens kontrolsignaler.

Med få undtagelser findes der de samme ben (signaler) på en ROM, PROM og EPROM.



Skitsen viser signalerne på en ROM (PROM, EPROM).

Adresse: Adresseledningernes antal afhænger af hukommelsens størrelse. Typiske størrelser er:

1 k = 1o24 adr.	1o adr. ledninger
2 k = 2o48 adr.	11 adr. ledninger
4 k = 4o96 adr.	12 adr. ledninger

Adresseledningerne forbindes til adressebussen som styres af CPU'en.

Data.

Dataoutputtet leverer det bitmønster, der er lagret på den via adresseledningerne udpegede plads i hukommelsen.

Dataoutputtet er et 3-state output. Det vil sige, at outputtet kan antage tre tilstande, nemlig low, high og off. Den sidste tilstand, hvor outputtet er højimpedant, muliggør at dataoutputtet kan kobles til databussen uden at inteferere med de øverste kredse på bussen, blot der sørges for at kun een kreds er selected af gangen.

Output enable.

Output enable ($\overline{OE}$) indgangen kontrollerer kun 3-state-bufferene i dataudgangene. Det vil sige at hukommelsen kan være chip-selected men stadig holdes 3-stated.

Chip-selected.

Chip select ($\overline{CS}$) indgangen er det overordnede indput på hukommelseskredsen, og styrer adressedecoderen i adresseindgangen samt 3-state buff erene i dataudgangene.

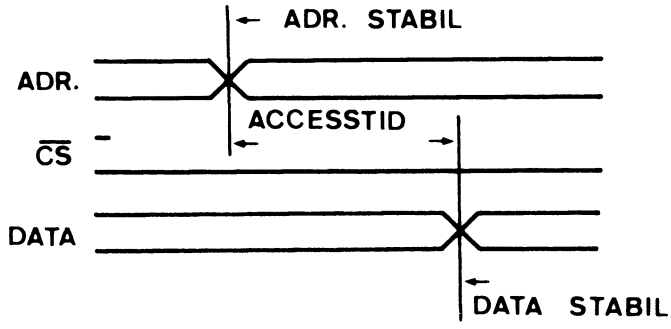
Når $\overline{CS}$ er inaktiv (high) er hukommelsens dataudgange i 3-state. Når $\overline{CS}$ er aktiv (low) er dataudgangene lavimpedante og antager de niveauer, der svarer til indholdet på den adresserede plads.

Ordlængde.

Det almindeligste antal dataledninger på ROM'en, PROM'en og EPROM'en er 8. Det vil sige, at der på hver adresse kan lagres 8 bit, svarende til en byte.

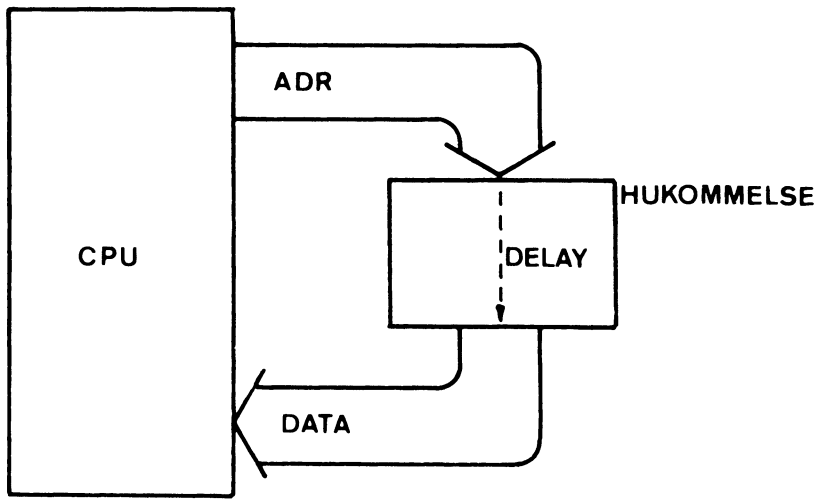
Accestid.

Når hukommelsen skal arbejde sammen med en CPU er hukommelsens accestid en egenskab af stor betydning.
Accestiden (adgangstiden) er den tid, der går fra adressen har stabiliseret sig til dataudgangene leverer stabile data.



Da CS kontrollerer adressedecoderen i kredsen vil det i praksis ofte være forsinkelsen fra CS til data, der skal tages hensyn til.

CPU'en og accestiden.



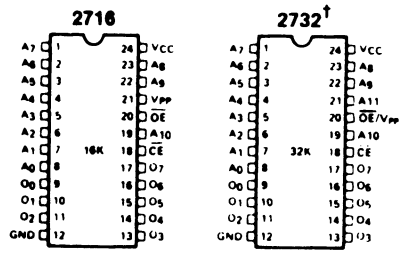
En readcycle har følgende forløb:

1. CPU'en sender adressen ud og starter "nedtællingen".
2. Hukommelsen begynder at finde de ønskede data frem, som den efter en tid (accestiden) får sendt ind til CPU'en.
3. Når CPU'en har "talt ned til nul" tager den de data ind, der findes på dens dataindgange, også selvom hukommelsen egentlig ikke var klar.

   Det betyder, at hukommelsens accestid skal modsvare CPU'ens krav.
   Hvis hukommelsen og dermed accestiden er givet, kan det blive nødvendigt at indskyde en waite-state i hver readcycle.

Data 2716.

PIN CONFIGURATION



†Refer to 2732 data sheet for specifications

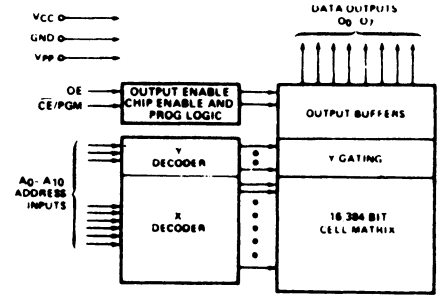
MODE SELECTION

MODE \ PINS	CE/PGM (18)	OE (20)	VPP (21)	VCC (24)	OUTPUTS (9-11, 13-17)
Read	V_{IL}	V_{IL}	+5	+5	D_{OUT}
Standby	V_{IH}	Don't Care	+5	+5	High Z
Program	Pulsed V_{IL} to V_{IH}	V_{IH}	+25	+5	D_{IN}
Program Verify	V_{IL}	V_{IL}	+25	+5	D_{OUT}
Program Inhibit	V_{IL}	V_{IH}	+25	+5	High Z

PIN NAMES

A_0–A_{10}	ADDRESSES
CE/PGM	CHIP ENABLE/PROGRAM
OE	OUTPUT ENABLE
O_0–O_7	OUTPUTS

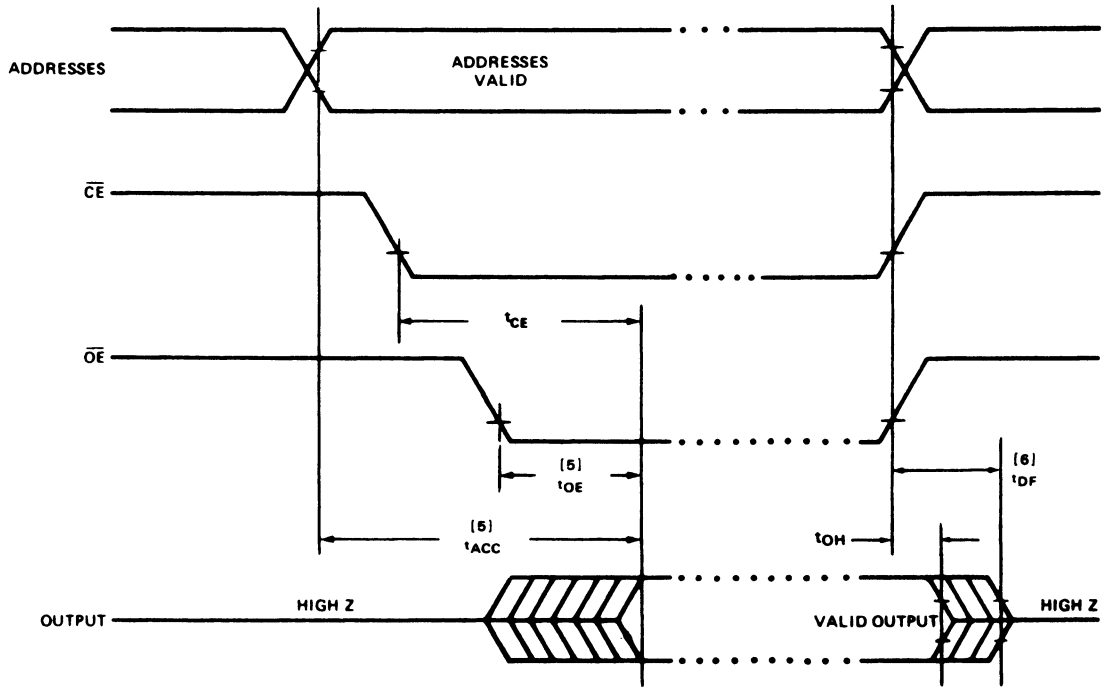
BLOCK DIAGRAM



A.C. Characteristics

Symbol	Parameter	Limits (ns)										Test Conditions
		2716		2716-1		2716-2		2716-5		2716-6		
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
t_{ACC}	Address to Output Delay		450		350		390		450		450	$\overline{CE} = \overline{OE} = V_{IL}$
t_{CE}	$\overline{CE}$ to Output Delay		450		350		390		490		650	$\overline{OE} = V_{IL}$
t_{OE}	Output Enable to Output Delay		120		120		120		160		200	$\overline{CE} = V_{IL}$
t_{DF}	Output Enable High to Output Float	0	100	0	100	0	100	0	100	0	100	$\overline{CE} = V_{IL}$
t_{OH}	Output Hold from Addresses, $\overline{CE}$ or $\overline{OE}$ Whichever Occurred First	0		0		0		0		0		$\overline{CE} = \overline{OE} = V_{IL}$

A. C. Waveforms [1]



NOTE:
1. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
2. V_{PP} may be connected directly to V_{CC} except during programming. The supply current would then be the sum of I_{CC} and I_{PP1}.
3. Typical values are for $T_A = 25°C$ and nominal supply voltages.
4. This parameter is only sampled and is not 100% tested.
5. OE may be delayed up to $t_{ACC} - t_{OE}$ after the falling edge of CE without impact on t_{ACC}.
6. t_{DF} is specified from OE or CE, whichever occurs first.

Datahukommelsen.

Til den anden type hukommelse anvendes den såkaldte

RAM, (random access memory)

Det betyder "en hukommelse med tilfældig adgang".

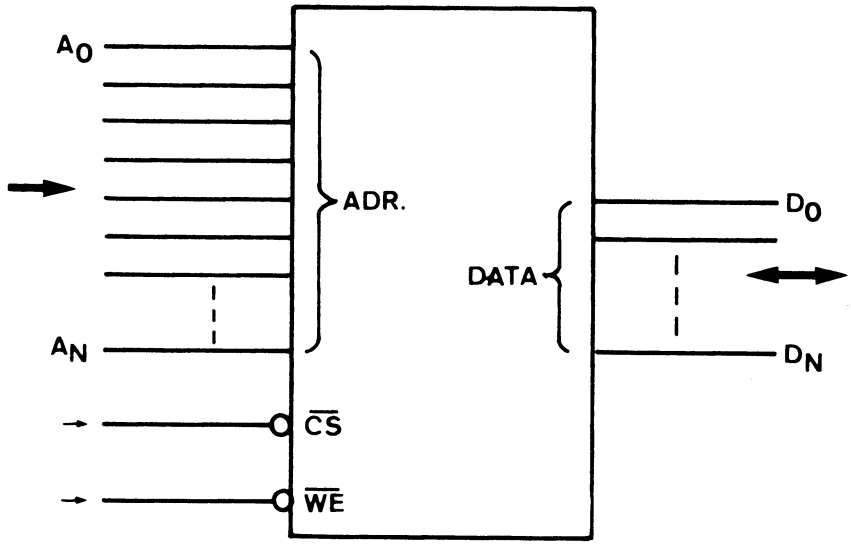
Denne type kaldes også for

Read/write memory, hvilket mere korrekt beskriver denne hukommelses egenskaber.

Teknologi.

Hukommelsescellerne i en RAM (statisk) er opbygget som bistabile flip-flops, hvorfor det er muligt at ændre indholdet i overensstemmelse med de tilførte data.
Til gengæld mistes hukommelsens indhold hvis forsyningsspændingen blot kortvarigt forsvinder.

Datahukommelsens kontrolsignaler.



Adresse. Se programhukommelse.

Chip select. Se programhukommelse.

Data.

Da RAM-hukommelsen er en læse/skrive-hukommelse er dataledningerne to-vejs, styret af write enable ($\overline{WE}$).

Når $\overline{WE}$ er high, går dataledningerne ud af kredsen og den er i read mode når $\overline{CS}$ aktiveres.

Påtrykkes $\overline{WE}$ et low niveau skiftes til write-mode, hvorfor dataledningerne nu er dataindgange, således at det tilførte bitmønster indlæses på den valgte adresse, dog under forudsætning af at $\overline{CS}$ er aktiv.

Write enable.

Write enable indgangen skifter hukommelsen mellem read mode og write mode.
Det er vigtigt, at write enable signalet er timet korrekt i forhold til adressen idet der ellers kan være risiko for utilsigtet at ændre tilfældige adressers indhold.

Ordlængde.

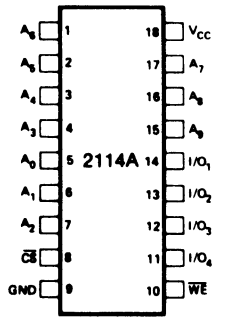
RAM-hukommelser findes med ordlængder på 1 eller 4 bit. Det vil sige at der skal anvendes henholdsvis 8 og 2 kredse i "PARALLEL" for at opnå en ordlængde på 1 byte.

Accestid.

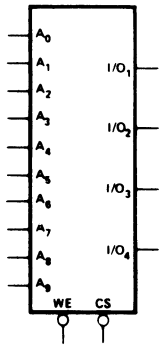
Se programhukommelse.

Data 2114.

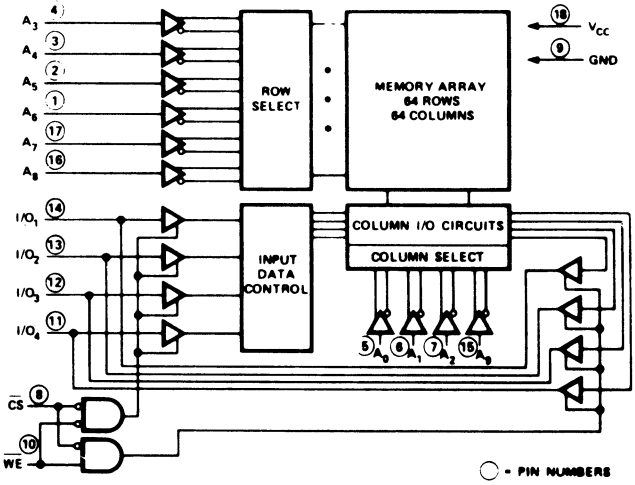
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

A_0–A_9	ADDRESS INPUTS	V_{CC}	POWER (+5V)
$\overline{WE}$	WRITE ENABLE	GND	GROUND
$\overline{CS}$	CHIP SELECT		
I/O_1–I/O_4	DATA INPUT/OUTPUT		

READ CYCLE [1]

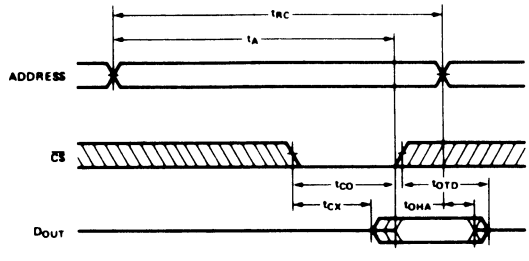
SYMBOL	PARAMETER	2114AL-1 Min.	2114AL-1 Max.	2114AL-2 Min.	2114AL-2 Max.	2114AL-3 Min.	2114AL-3 Max.	2114A-4/L-4 Min.	2114A-4/L-4 Max.	2114A-5 Min.	2114A-5 Max.	UNIT
t_{RC}	Read Cycle Time	100		120		150		200		250		ns
t_A	Access Time		100		120		150		200		250	ns
t_{CO}	Chip Selection to Output Valid		70		70		70		70		85	ns
t_{CX}	Chip Selection to Output Active	10		10		10		10		10		ns
t_{OTD}	Output 3-state from Deselection		30		35		40		50		60	ns
t_{OHA}	Output Hold from Address Change	15		15		15		15		15		ns

WRITE CYCLE [2]

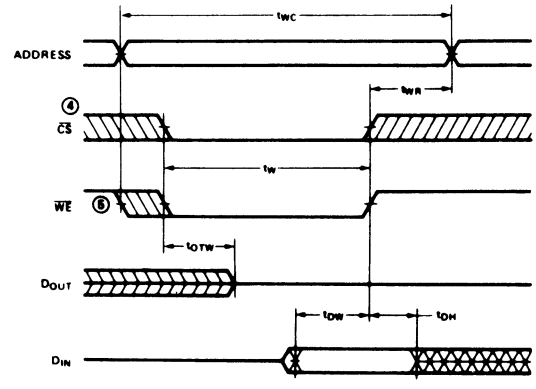
SYMBOL	PARAMETER	2114AL-1 Min.	2114AL-1 Max.	2114AL-2 Min.	2114AL-2 Max.	2114AL-3 Min.	2114AL-3 Max.	2114A-4/L-4 Min.	2114A-4/L-4 Max.	2114A-5 Min.	2114A-5 Max.	UNIT
t_{WC}	Write Cycle Time	100		120		150		200		250		ns
t_W	Write Time	75		75		90		120		135		ns
t_{WR}	Write Release Time	0		0		0		0		0		ns
t_{OTW}	Output 3-state from Write		30		35		40		50		60	ns
t_{DW}	Data to Write Time Overlap	70		70		90		120		135		ns
t_{DH}	Data Hold from Write Time	0		0		0		0		0		ns

WAVEFORMS

READ CYCLE ③



WRITE CYCLE



NOTES:

3. $\overline{WE}$ is high for a Read Cycle.
4. If the $\overline{CS}$ low transition occurs simultaneously with the $\overline{WE}$ low transition, the output buffers remain in a high impedance state.
5. $\overline{WE}$ must be high during all address transitions.

Hukommelsens opbygning.

Hukommelsen er bygget op omkring adressebussen og databussen.

Fordelen ved et bussystem er, at det er nemt at udbygge med flere kredse, idet man blot skal forbinde de tilkomne kredse til adresse- og databus.

3-state.

For at en kreds kan indpasses i et bussystem skal den have 3-state output. Det vil sige at kredsens output kan gøres højimpedanset og dermed kobles fra databussen, idet der kun må være een aktiv kreds af gangen.

Chip-select.

Til at styre de forskellige kredses til- og frakobling tjener chip-select logikken. Denne styres af en del af adressebussen og styrer de forskellige hukommelseskredse via deres chip-select input.

Hukommelsens størrelse.

Det største antal adresser (hukommelsespladser) i en u-computers lager-sektion bestemmes af det antal adresseledninger, der er ført ud af CPU'en (direkte eller multiplexed).

Det almindeligste antal adresseledninger på en 8-bit CPU er 16.

Hukommelsens (mulige) størrelse bliver da:

$$2^{16} = 65536 = 64 \cdot 1024 \rightarrow 64K$$

Opdelingen i "k" er behagelig set i relation til de almindeligste størrelser på hukommelseskredse der er 1, 2 og 4 k.

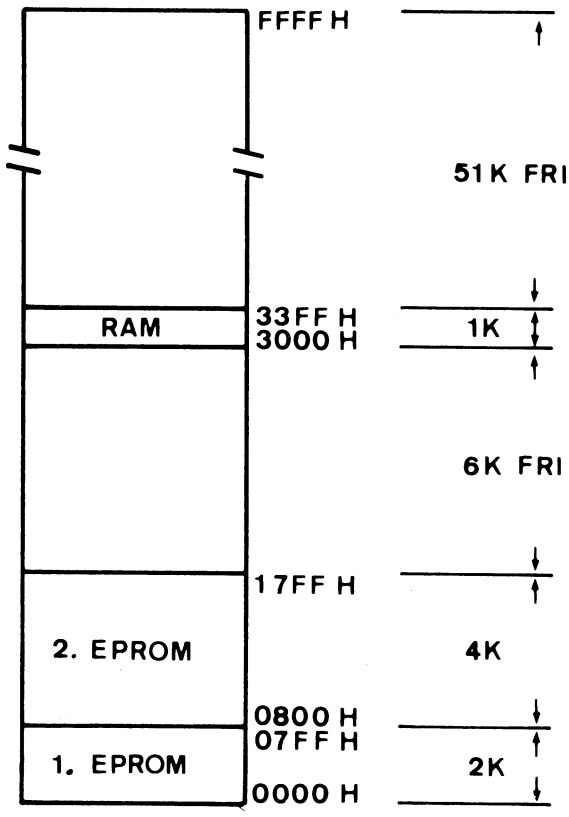
Memorymap.

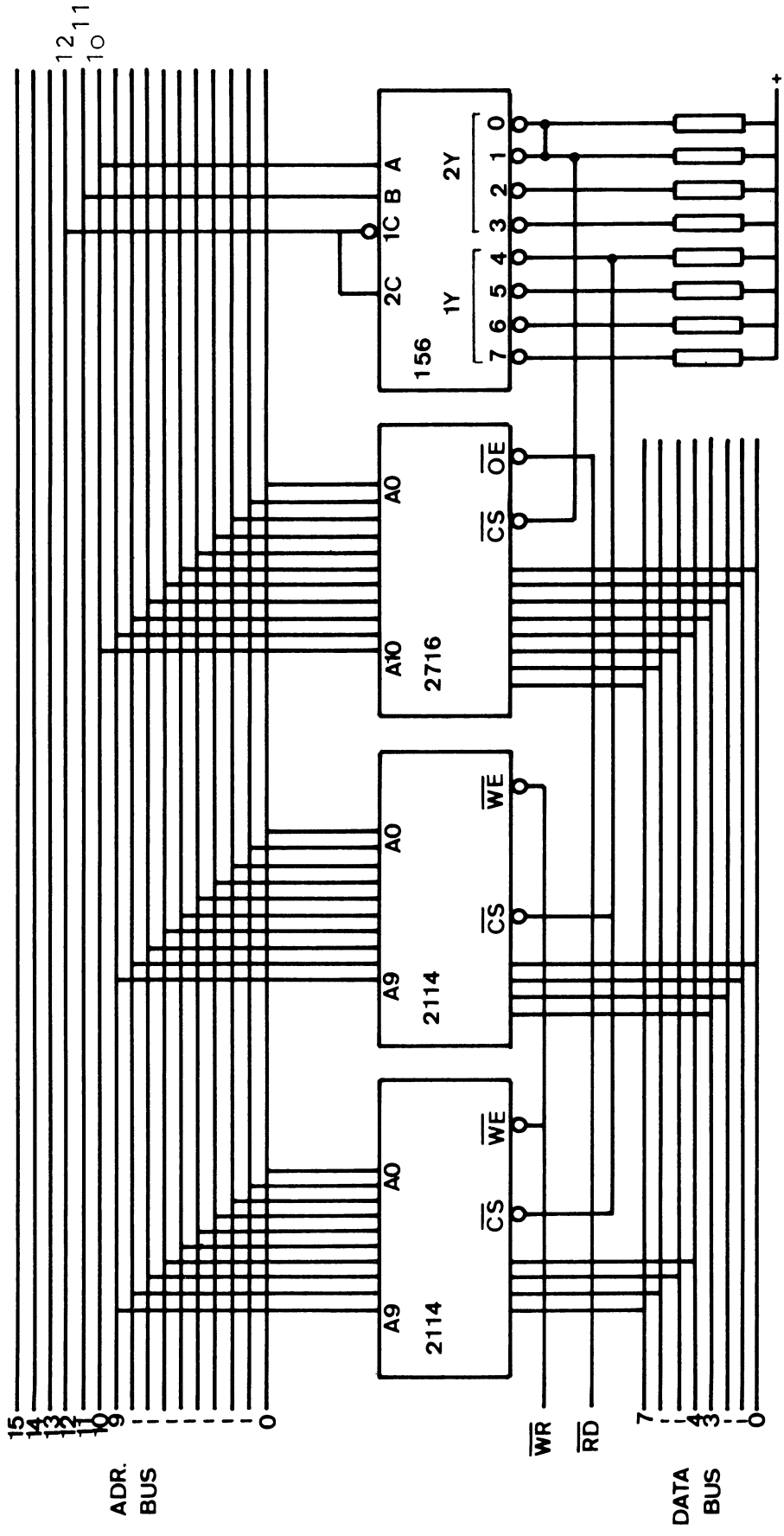
Som et middel til at beskrive hvilke typer kredse (ROM, RAM) en hukommelse indeholder, samt deres adresser anvendes ofte et memorymap.

Eksempel:
En hukommelse indeholder:

1. EPROM, 2716, 2k start-ard. ØØØØH
2. EPROM, 2732, 4k start-ard. Ø8ØØH
3. RAM, 2x2114, 1k start-ard. 3ØØØH

Dette giver følgende memorymap.





Chip-select logik med 1k opdeling.

Chip-select logikken er normalt bygget op omkring en 3 til 8 linier decoder der styres af de højere adresse-ledninger.

Se diagram

Diagrammet viser en hukommelse der består af 1k RAM og 2k EPROM. Som CS-decoder er anvendt en dual 2 til 4 linier decoder med open collektor output. Den kobles som en 3 til 8 linier decoder og styres af A_{10}-A_{12}.

Decoderens udgange dækker følgende adresseområder:

UDG.	ADR.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	fra	x	x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	ØØØØH
0	til	x	x	x	0	0	0	1	1	1	1	1	1	1	1	1	1	Ø3FFH
1	fra	x	x	x	0	0	1	0	-	-	-	-	-	-	-	-	0	Ø4ØØH
1	til	x	x	x	0	0	1	1	-	-	-	-	-	-	-	-	1	Ø7FFH
2	fra	x	x	x	0	1	0	0	-	-	-	-	-	-	-	-	0	Ø8ØØH
2	til	x	x	x	0	1	0	1	-	-	-	-	-	-	-	-	1	ØBFFH
3	fra	x	x	x	0	1	1	0	-	-	-	-	-	-	-	-	0	ØCØØH
3	til	x	x	x	0	1	1	1	-	-	-	-	-	-	-	-	1	ØFFFH
4	fra	x	x	x	1	0	0	0	-	-	-	-	-	-	-	-	0	1ØØØH
4	til	x	x	x	1	0	0	1	-	-	-	-	-	-	-	-	1	13FFH
5	fra	x	x	x	1	0	1	0	-	-	-	-	-	-	-	-	0	14ØØH
5	til	x	x	x	1	0	1	1	-	-	-	-	-	-	-	-	1	17FFH
6	fra	x	x	x	1	1	0	0	-	-	-	-	-	-	-	-	0	18ØØH
6	til	x	x	x	1	1	0	1	-	-	-	-	-	-	-	-	1	1BFFH
7	fra	x	x	x	1	1	1	0	-	-	-	-	-	-	-	-	0	1cØØH
7	til	x	x	x	1	1	1	1	-	-	-	-	-	-	-	-	1	1FFFH

x = ligegyldig.

Som det fremgår af tabellen dækker hver udgang et område på 1k.
Dette passer fint med RAMlagerets størrelse, men da EPROM'en er 2 k stor er det nødvendigt at chip-selecte den fra to udgange. Da der er anvendt en decoder med open collektor output og pull-up modstande kan man blot forbinde to naboudgange.

Sammenholdes diagrammet og tabellen findes følgende adresser på RAM og EPROM:

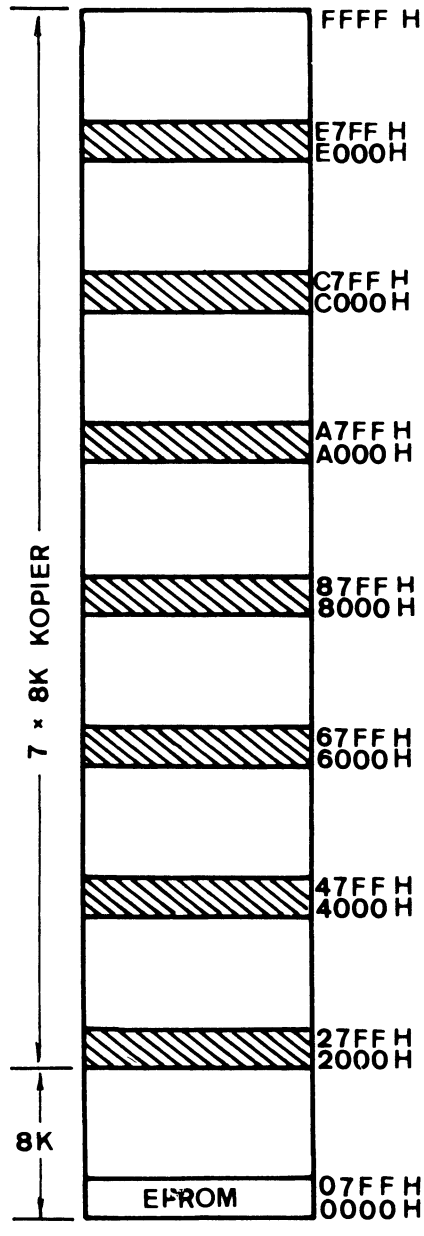
EPROM: Udg. 0 og 1 øøøøH - Ø7FFH
RAM: Udg. 4 1øøøH - 13FFH

Kopier.

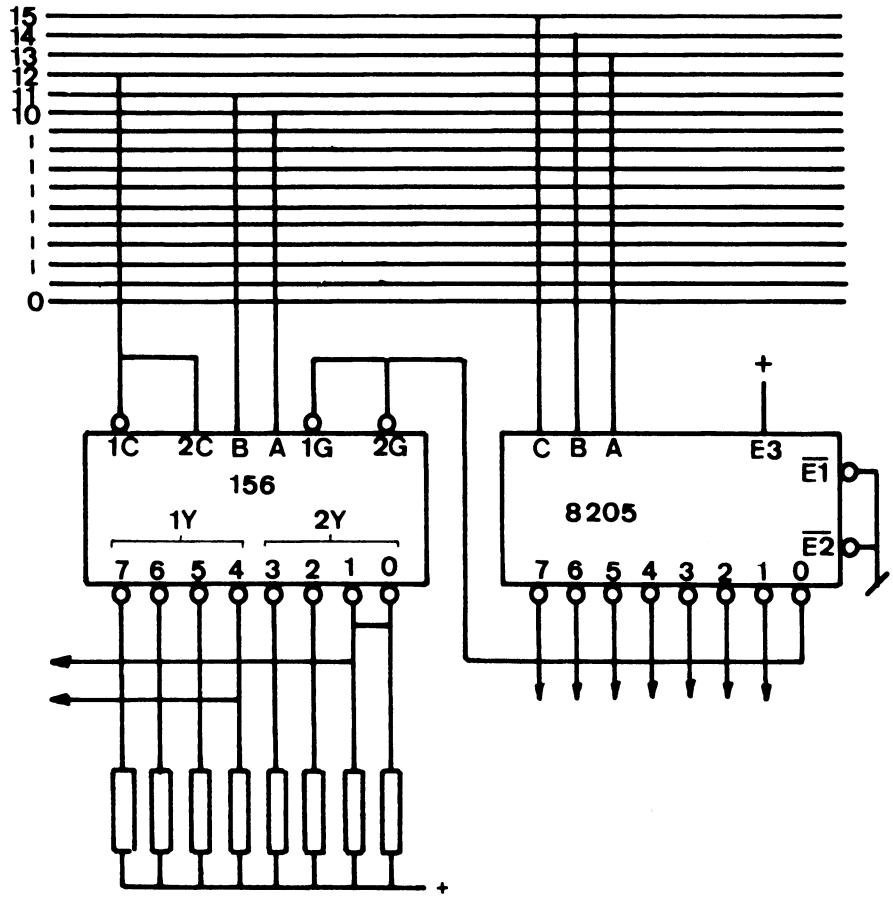
Da de tre øverste bit i adressebussen ikke er ført til decoderen kan de antage en vilkårlig værdi uden at påvirke chip-select.
Set fra CPU'en vil det virke som om den enkelte kreds kan nås på 8 forskellige startadresser.

For eksempel kan laveste adresse i EPROM'en nås på følgende 8 adresser:

ØØØØH, 2ØØØH, 4ØØØH, 6ØØØH, 8ØØØH, AØØØH, CØØØH, EØØØH.



I små systemer, hvor der kun er behov for maksimalt 8k hukommelse gøres der ikke noget for at fjerne kopieffekten. Hvis der er behov for mere end 8k hukommelse må også de tre øverste adresseledninger decodes.
Dette kan gøres ved hjælp af endnu en 3 til 8 linier decoder hvis nul-output anvendes til at enable den første decoder.



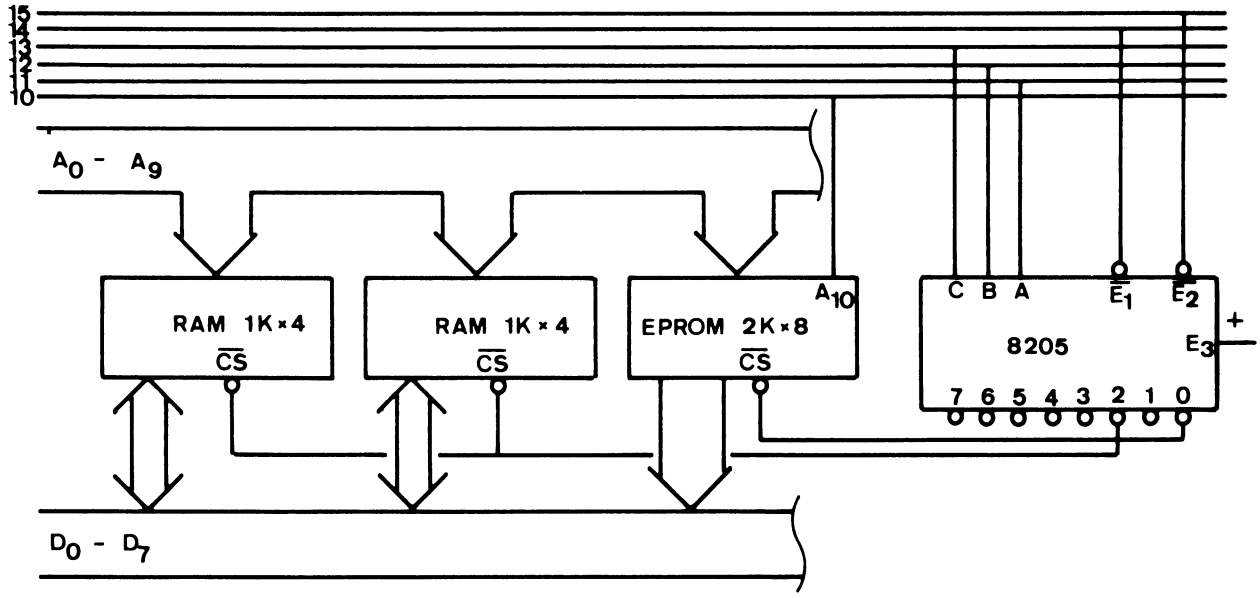
82o5's udgange dækker følgende adresseområder.

Udg. no.	Adr. fra	til
o	øøøøH	1FFFH
1	2øøøH	3FFFH
2	4øøøH	5FFFH
3	6øøøH	7FFFH
4	8øøøH	9FFFH
5	Aøøøh	BFFFH
6	Cøøøh	DFFFH
7	Eøøøh	FFFFH

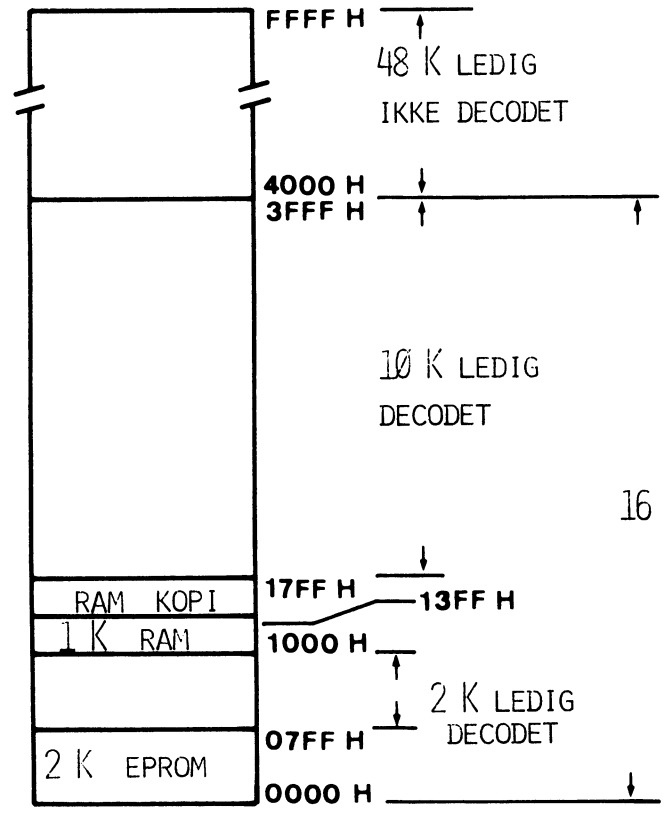
Decodning i 2k afsnit.

I stedet for at decode A_{10}, A_{11} og A_{12} hvorved der chip-selectes i 1k afsnit, kan man i stedet tage A_{11}, A_{12} og A_{13} og selecte i 2k afsnit.

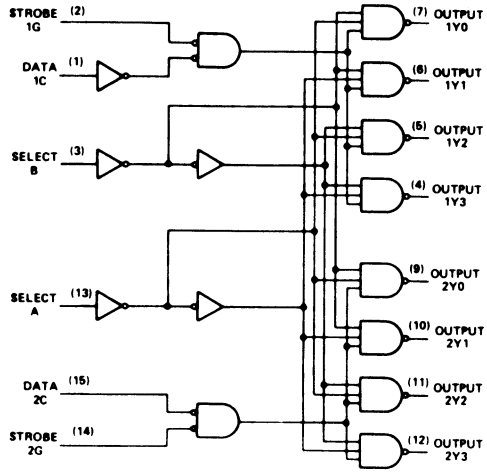
Ved at udnytte decoderens enableindgange kan kopier ud over 16k undgås.



Memorymap.
Da A_{10} hverken decodes eller tilføres RAM-lageret vil dette komme til at dække 2k



74 LS 155. Dual 2 til 4 linier decoder.



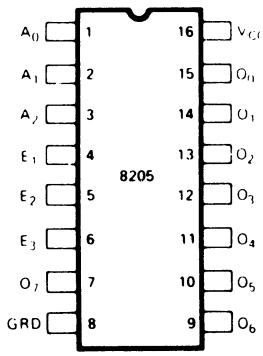
FUNCTION TABLES
2-LINE-TO-4-LINE DECODER
OR 1-LINE-TO-4-LINE DEMULTIPLEXER

INPUTS				OUTPUTS			
SELECT		STROBE	DATA				
B	A	1G	1C	1Y0	1Y1	1Y2	1Y3
X	X	H	X	H	H	H	H
L	L	L	H	L	H	H	H
L	H	L	H	H	L	H	H
H	L	L	H	H	H	L	H
H	H	L	H	H	H	H	L
X	X	X	L	H	H	H	H

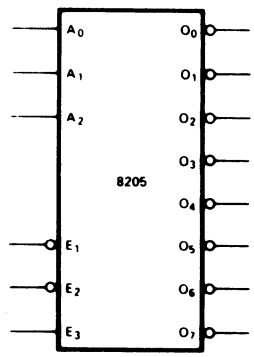
INPUTS				OUTPUTS			
SELECT		STROBE	DATA				
B	A	2G	2C	2Y0	2Y1	2Y2	2Y3
X	X	H	X	H	H	H	H
L	L	L	L	L	H	H	H
L	H	L	L	H	L	H	H
H	L	L	L	H	H	L	H
H	H	L	L	H	H	H	L
X	X	X	H	H	H	H	H

8205/74LS138. 3 til 8 linier decoder.

PIN CONFIGURATION



LOGIC SYMBOL



PIN NAMES

A_0 A_2	ADDRESS INPUTS
$\overline{E_1}$ $\overline{E_3}$	ENABLE INPUTS
$\overline{O_0}$ $\overline{O_7}$	DECODED OUTPUTS

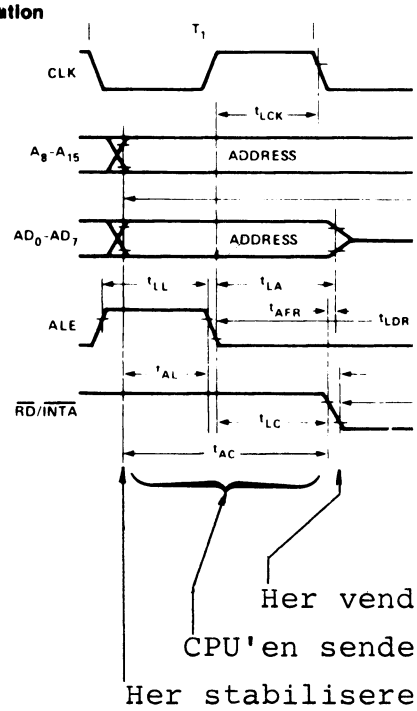
ADDRESS			ENABLE			OUTPUTS							
A_0	A_1	A_2	E_1	E_2	E_3	0	1	2	3	4	5	6	7
L	L	L	L	L	H	L	H	H	H	H	H	H	H
H	L	L	L	L	H	H	L	H	H	H	H	H	H
L	H	L	L	L	H	H	H	L	H	H	H	H	H
H	H	L	L	L	H	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
H	L	H	L	L	H	H	H	H	H	H	L	H	H
L	H	H	L	L	H	H	H	H	H	H	H	L	H
H	H	H	L	L	H	H	H	H	H	H	H	H	L
X	X	X	L	L	L	H	H	H	H	H	H	H	H
X	X	X	H	L	L	H	H	H	H	H	H	H	H
X	X	X	L	H	L	H	H	H	H	H	H	H	H
X	X	X	H	H	L	H	H	H	H	H	H	H	H
X	X	X	H	L	H	H	H	H	H	H	H	H	H
X	X	X	L	H	H	H	H	H	H	H	H	H	H
X	X	X	H	H	H	H	H	H	H	H	H	H	H

CS-TIMING.

Som tidligere nævnt, er det chip-select logikkens opgave at sikre, at der kun er en aktiv kreds på databussen af gangen, hvilket også slår til når der kun ses på kredsene i hukommelsen, men ikke nødvendigvis i relation til CPU'en.

Eksempel:

Read Operation



T_{clk} = 32o n sek
T_{AL} = 115 n sek
T_{LA} = 1oo n sek

Her vender databussen ind i CPU'en,
CPU'en sender adrl. på databussen
Her stabiliseres adressen.

Se diagram
Fra adressen er stabiliseret til der er en aktiv chip-select går der ca. 2o n sek. Hvis det er ramlageret der bliver selected (2114) går der yderligere ca. lo nsek. hvorefter dataudgangene går ud af 3-state og bliver aktive.
CPU'en sender samtidig de 8 lave adresser ud på databussen med det resultat at niveauerne bliver forkerte. I værste fald kan det føre til ødelæggelse af en kreds.
For at undgå dette skal hukommelsen forhindres i at gå på databussen inden CPU'en har "Sluppet" den.
Der er flere mulige løsninger på problemet. Har hukommelsen et "outputenable" Input (2716) kan dette styres af et egnet signal, ved 8o85 $\overline{RD}$.

Findes et sådant input ikke kan chip-select forsinkes, for eksempel ved at gate CS-decoderen med et signal der er afledt af $IO/\overline{M}$, $\overline{RD}$ og $\overline{WR}$ (8085).
Den sidste metode kan nødvendiggøre anvendelsen af wait-state.

Parrallel IN/OUT.

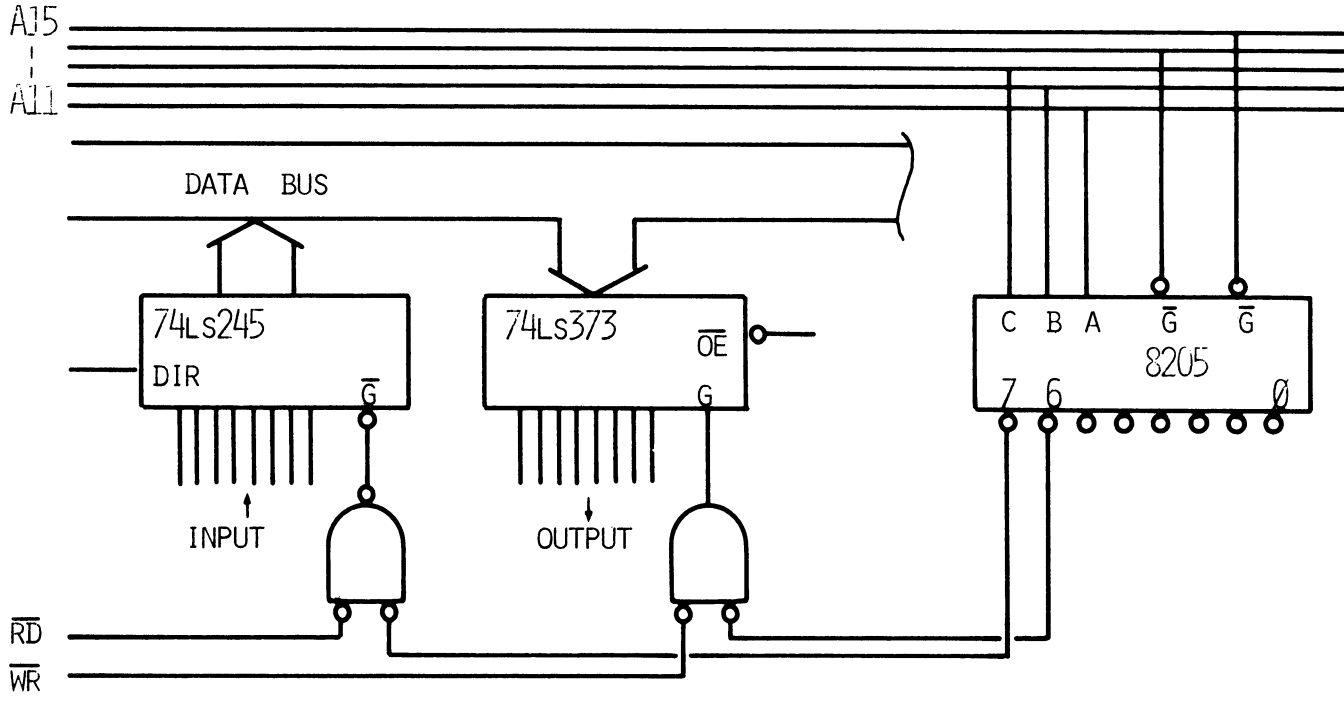
8o85A kommunikerer til både hukommelse og porte(I/O) ved hjælp af read- og write-maskincycles. Under en maskincycle sender 8o85A adresse og kontrolsignaler samtidig med at den enten sender data på databussen, eller læser data fra databussen. Er CPU'en for eksempel i en read cycle kan den læse fra ROM, RAM, I/O, - eller ingenting. Ligeledes kan en write-cycle ske til RAM, I/O, - eller ingenting.

Adressering af porte.

Der er to måder at adressere porte på i et 8o85 system. Hvis IO/$\overline{M}$ signalet fra CPU'en anvendes til at skelne mellem I/O- og hukommelsesread og write cycles kaldes det standard I/O, eller I/O-mapped, I/O.
Hvis IO/$\overline{M}$ ikke anvendes, skelner CPU'en ikke mellem I/O og hukommelse. Dette kaldes memory-mapped I/O.

Memory-mapped I/O.

Ved memory-mapped I/O indpasses portene i hukommelsen sideordnet med de øvrige kredse (RAM og ROM).



I det viste eksempel har inputporten adresse 38øøH og outputporten adresse 3øøøH.

Da de to porte chipselectes direkte af chipselectdecoderen kommer de hver især til at dække et område på 2k (A_0-A_{10} er ligegyldige).

Af hensyn til timingen skal chip-select signalerne gates med $\overline{RD}$ og $\overline{WR}$ fra CPU'en.

Ved at udbygge chip-selectlogikken til også at styres af A_0-A_{10} vil hver port kun behøve at beslaglægge en enkelt adresse.

Fordele ved memorymapped I/O.

Fordelene ved memorymapped I/O skal søges i, at de instruktioner, der arbejder på hukommelsen også kan anvendes i forbindelse med portene.

For eksempel:

MOVr,M: Input-port til ethvert register.

MOVM,r: Ethvert register til output-port.

MVI M : Data immediate til output-port.

LDA : Input-port til accumulator.

STA : Accumulator til outputport.

LHLD : Input-porte til H og L reg., 16 bit.

SHLD : H og L reg. til output-porte, 16 bit.

ADD M : Adder inputport til accumulator.
ANA M : And inputport med accumulator.

Som en fordel må også regnes det næsten ubegrænsede antal porte, der kan laves, dog på bekostning af hukommelsesareal.

Ulemper ved memorymapped I/O.

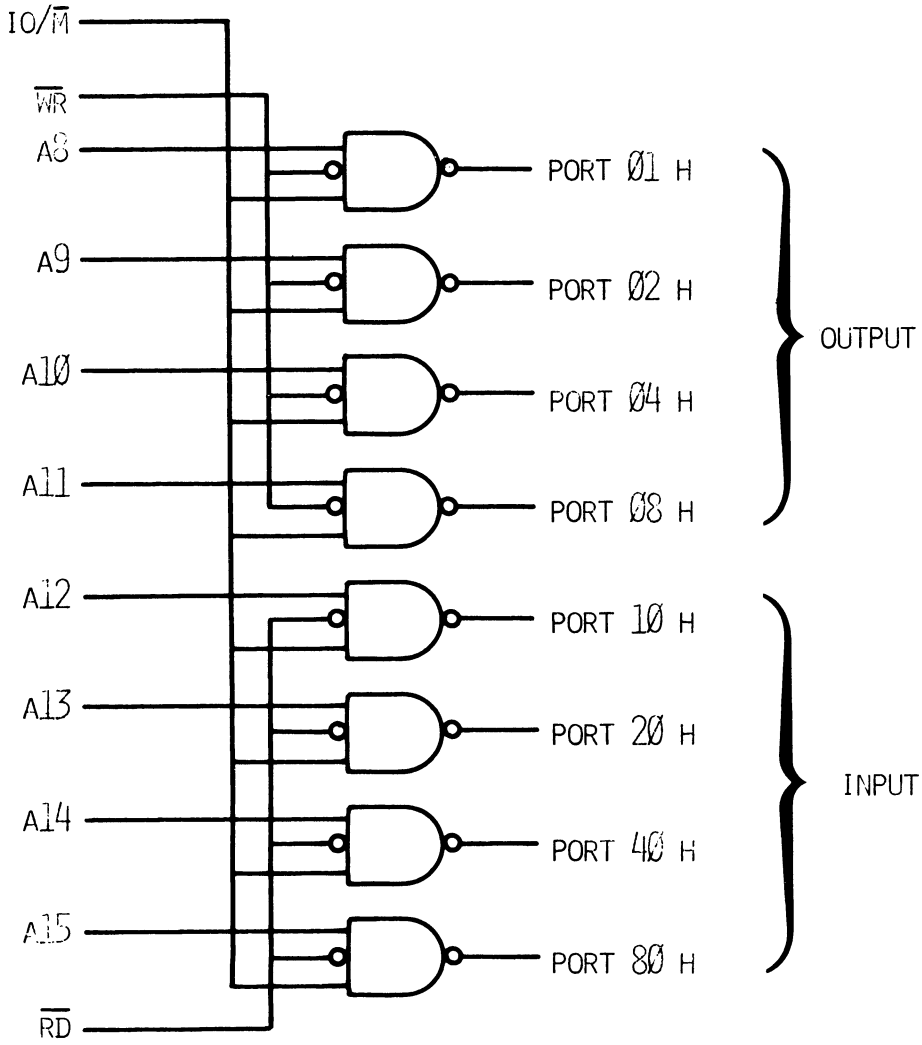
Den indlysende ulempe ved memorymapped I/O er tabet af hukommelsesareal, hvilket specielt gør sig gældende hvis portene kun chip-selectes af de højeste adressebit.

I/O-mapped I/O.

Ved I/O-mapped I/O skelner CPU'en mellem hukommelse og porte ved hjælp af IO/$\overline{M}$-outputtet.
Der er kun to instruktioner, der kan få IO/$\overline{M}$ til at blive high, nemlig IN og OUT.
IN og OUT er to-byte instruktioner, der i anden byte indeholder portens nummer. Det vil sige, at der er mulighed for 256 forskellige porte.
Når CPU'en eksekverer en in eller out instruktion trækkes IO/$\overline{M}$ high, og portnummeret der var indeholdt i instruktionens anden byte sendes ud både på A8-15 og multiplexes ud på AD_{0-7}. Iøvrigt svarer forløbet af en I/O maskincycle til en memory-read eller memory-write maskincycle.

Liniær select.

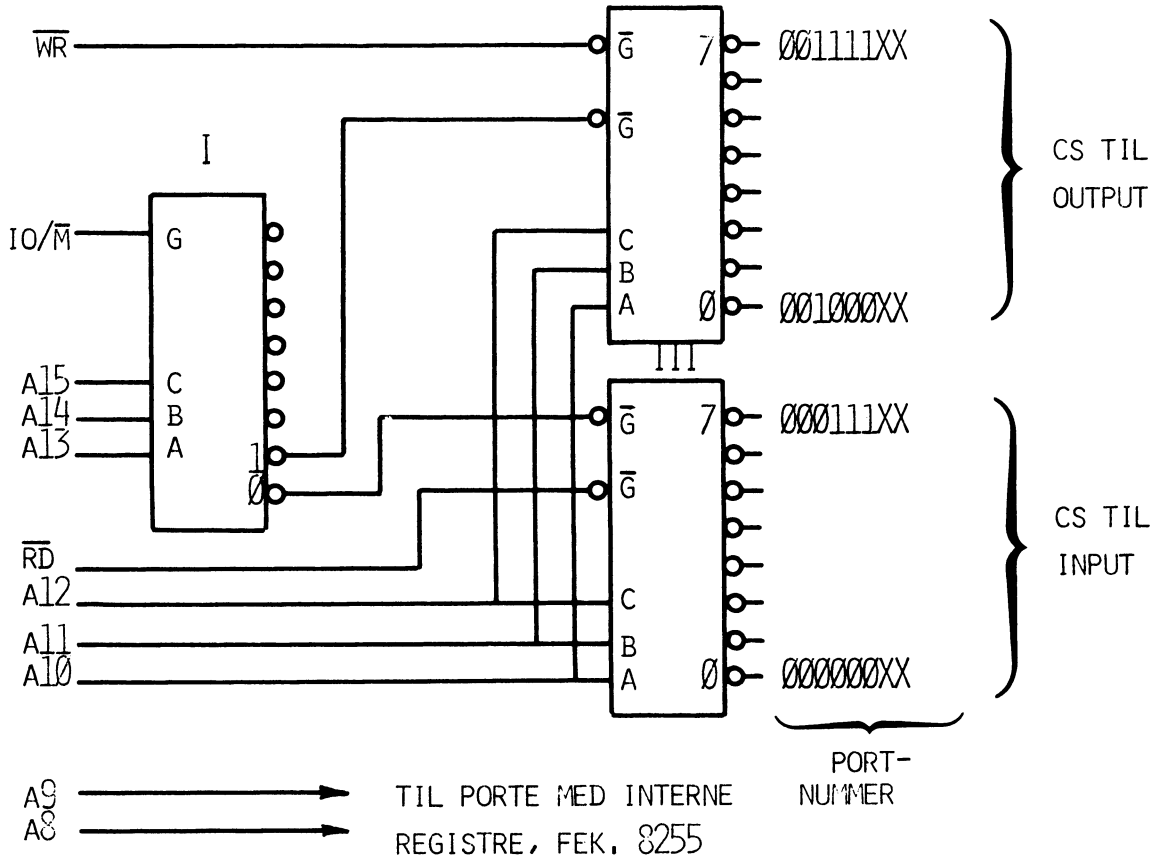
I mindre systemer med mindre end otte porte kan man klare sig med liniær select af portene. Det vil sige at hver Adresseledning vælger en bestemt port.



Portnummer decoder.

Hvis der skal være mere end otte I/O-kredse i systemet anvendes der en decoder til at vælge de enkelte porte. Computerene kommer altså til at indeholde to decoder-systemer mellem hvilken der skiftes ved hjælp af IO/$\overline{M}$.

Diagrameksempel.



Decoder I gates ved hjælp af IO/$\overline{M}$, således af decode-systemet kun er aktivt under eksekveringen af en I/O instruktion.

Timing.

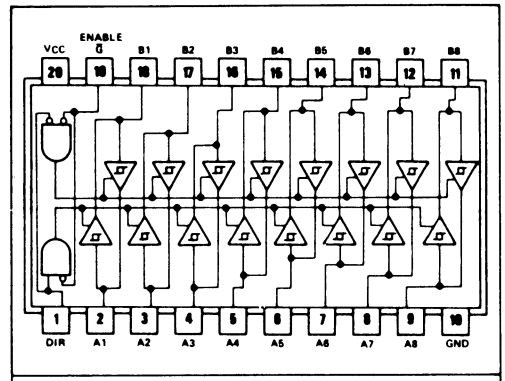
Hvis de anvendte porte er simple latch og buffere kan de sidste decodere i "træet" gates med $\overline{RD}$ og $\overline{WR}$ signalerne fra CPU'en.

Skal decodesystemet derimod anvendes til programmerbare portkredse som f.eks. INTEL 8255A, skal decoderne ikke gates med $\overline{RD}$ og $\overline{WR}$, idet disse signaler skal tilføres 8255A direkte.

Simple portkredse

74LS245 er 8 bidirectionale 3-state buffere med hysterese.Den er anvendelig som inputport.

TYPE	I_{OL} (SINK CURRENT)	I_{OH} (SOURCE CURRENT)
SN54LS245	12 mA	−12 mA
SN74LS245	24 mA	−15 mA



positive logic: see function table

description

These octal bus transceivers are designed for asynchronous two-way communication between data buses. The control function implementation minimizes external timing requirements.

The device allows data transmission from the A bus to the B bus or from the B bus to the A bus depending upon the logic level at the direction control (DIR) input. The enable input ($\overline{G}$) can be used to disable the device so that the buses are effectively isolated.

recommended operating conditions

PARAMETER	SN54LS245			SN74LS245			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			−12			−15	mA
Low-level output current, I_{OL}			12			24	mA
Operating free-air temperature, T_A	−55		125	0		70	°C

switching characteristics, V_{CC} = 5 V, T_A = 25°C

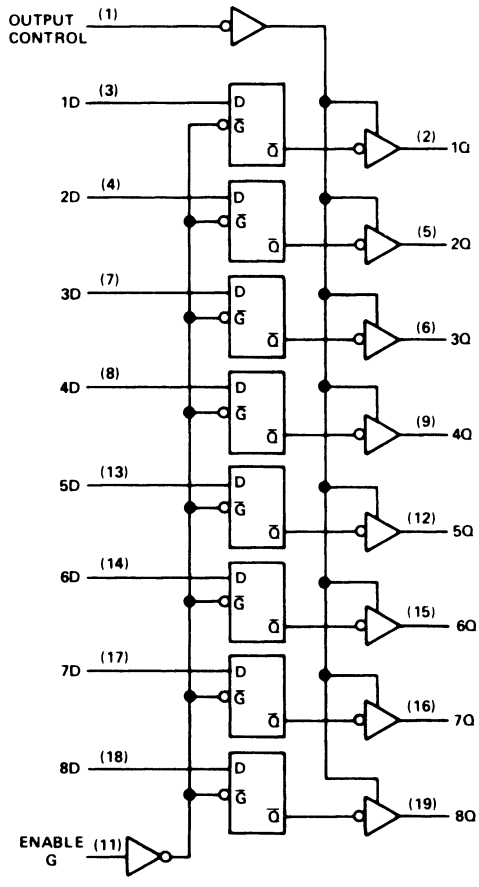
PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	Propagation delay time, low-to-high-level output	C_L = 45 pF, R_L = 667 Ω, See Note 2		8	12	ns
t_{PHL}	Propagation delay time, high-to-low-level output			8	12	ns
t_{PZL}	Output enable time to low level			27	40	ns
t_{PZH}	Output enable time to high level			25	40	ns
t_{PLZ}	Output disable time from low level	C_L = 5 pF, R_L = 667 Ω, See Note 2		15	25	ns
t_{PHZ}	Output disable time from high level			15	25	ns

NOTE 2: Load circuit and waveforms are shown on page 3-11.

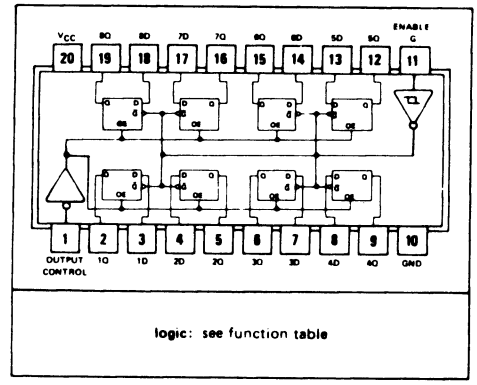
74 LS 373 er 8 latches med 3-state output.

Den kan anvendes som outputport eller inputport med eller uden latch-funktion.

'LS373, 'S373
TRANSPARENT LATCHES



SN54LS373, SN54S373 . . . J PACKAGE
SN74LS373, SN74S373 . . . J OR N PACKAGE
(TOP VIEW)



'LS373, 'S373
FUNCTION TABLE

OUTPUT CONTROL	ENABLE G	D	OUTPUT
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	Z

description

These 8-bit registers feature totem-pole three-state outputs designed specifically for driving highly-capacitive or relatively low-impedance loads. The high-impedance third state and increased high-logic-level drive provide these registers with the capability of being connected directly to and driving the bus lines in a bus-organized system without need for interface or pull-up components. They are particularly attractive for implementing buffer registers, I/O ports, bidirectional bus drivers, and working registers.

The eight latches of the 'LS373 and 'S373 are transparent D-type latches meaning that while the enable (G) is high the Q outputs will follow the data (D) inputs. When the enable is taken low the output will be latched at the level of the data that was setup.

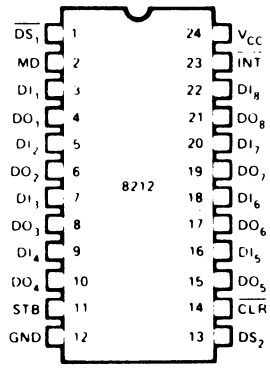
8212 er 8 latches med 3-state output samt interruptlogik.

Den kan anvendes som outputport eller inputport med latchfunktion og service request (interrupt).

The 8212 input/output port consists of an 8-bit latch with 3-state output buffers along with control and device selection logic. Also included is a service request flip-flop for the generation and control of interrupts to the microprocessor.

The device is multimode in nature. It can be used to implement latches, gated buffers or multiplexers. Thus, all of the principal peripheral and input/output functions of a microcomputer system can be implemented with this device.

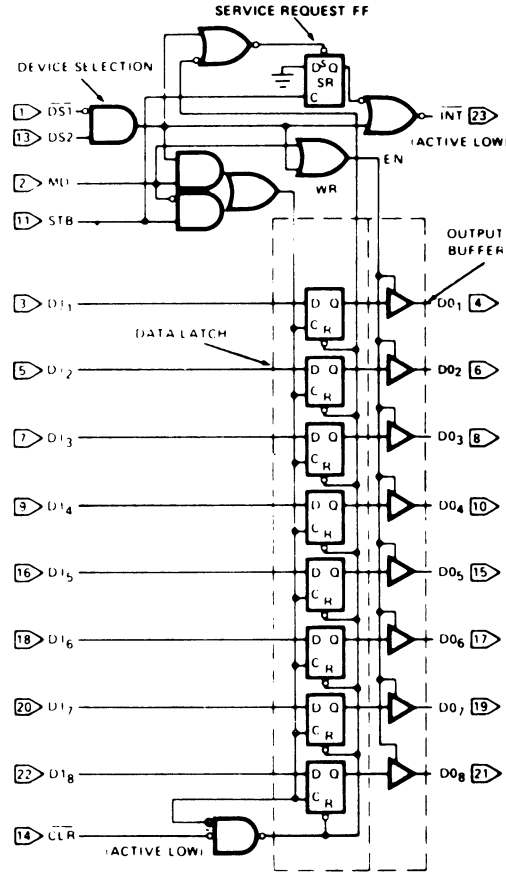
PIN CONFIGURATION



PIN NAMES

DI_1 DI_8	DATA IN
DO_1 DO_8	DATA OUT
$\overline{DS_1}$ DS_2	DEVICE SELECT
MD	MODE
STB	STROBE
$\overline{INT}$	INTERRUPT (ACTIVE LOW)
$\overline{CLR}$	CLEAR (ACTIVE LOW)

LOGIC DIAGRAM

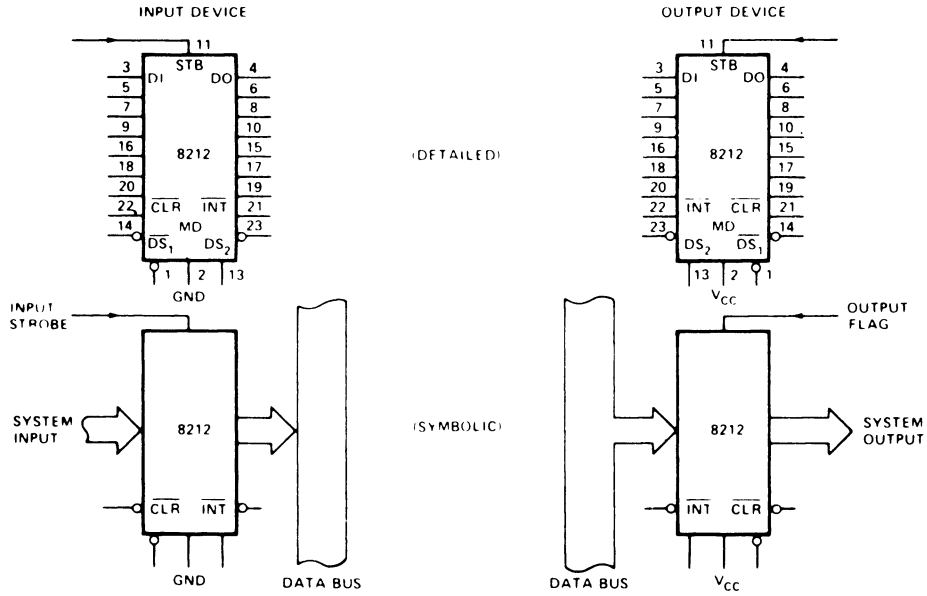


STB	MD	(DS_1 DS_2)	DATA OUT EQUALS
0	0	0	3 STATE
1	0	0	3 STATE
0	1	0	DATA LATCH
1	1	0	DATA LATCH
0	0	1	DATA LATCH
1	0	1	DATA IN
0	1	1	DATA IN
1	1	1	DATA IN

$\overline{CLR}$ RESETS DATA LATCH
SETS SR FLIP FLOP
(NO EFFECT ON OUTPUT BUFFER)

CLR	($\overline{DS_1}$ DS_2)	STB	*SR	INT
0	0	0	1	1
0	1	0	1	0
1	1	⌐	0	0
1	1	0	1	0
1	0	0	1	1
1	1	⌐	1	0

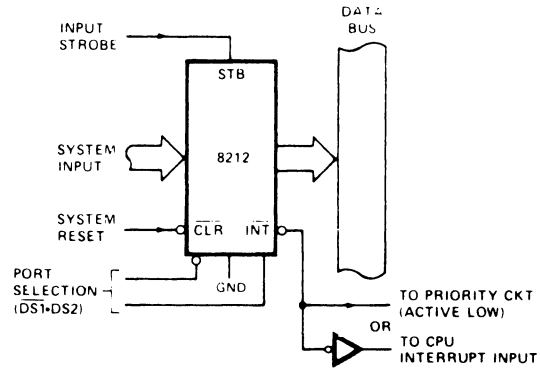
*INTERNAL SR FLIP FLOP

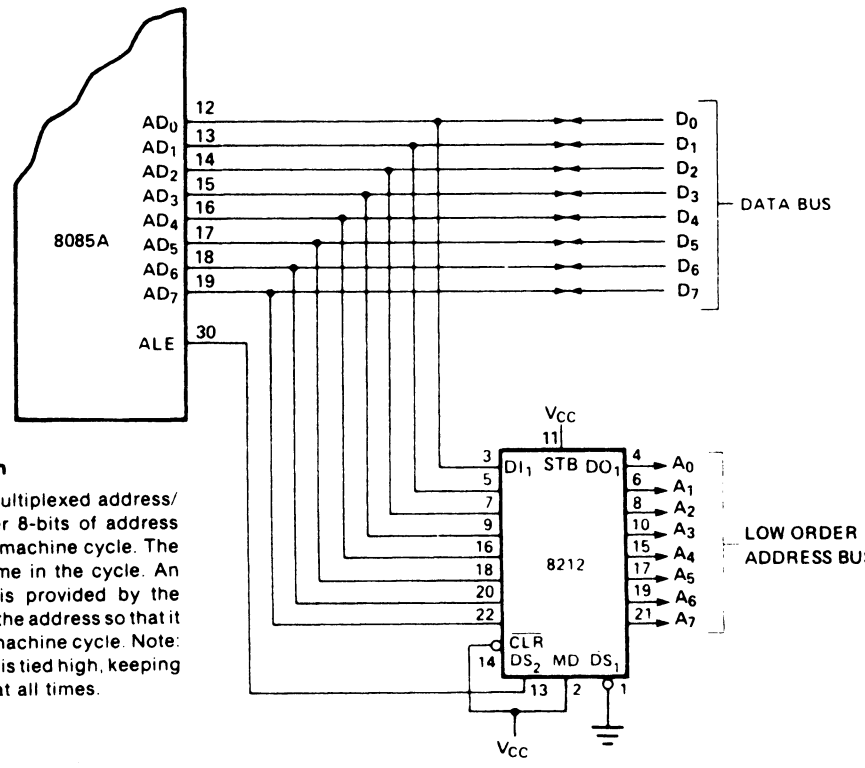


Interrupting Input Port

This use of an 8212 is that of a system input port that accepts a strobe from the system input source, which in turn clears the service request flip-flop and interrupts the processor. The processor then goes through a service routine, identifies the port, and causes the device selection logic to go true — enabling the system input data onto the data bus.

INTERRUPTING INPUT PORT





8085A Low-Order Address Latch

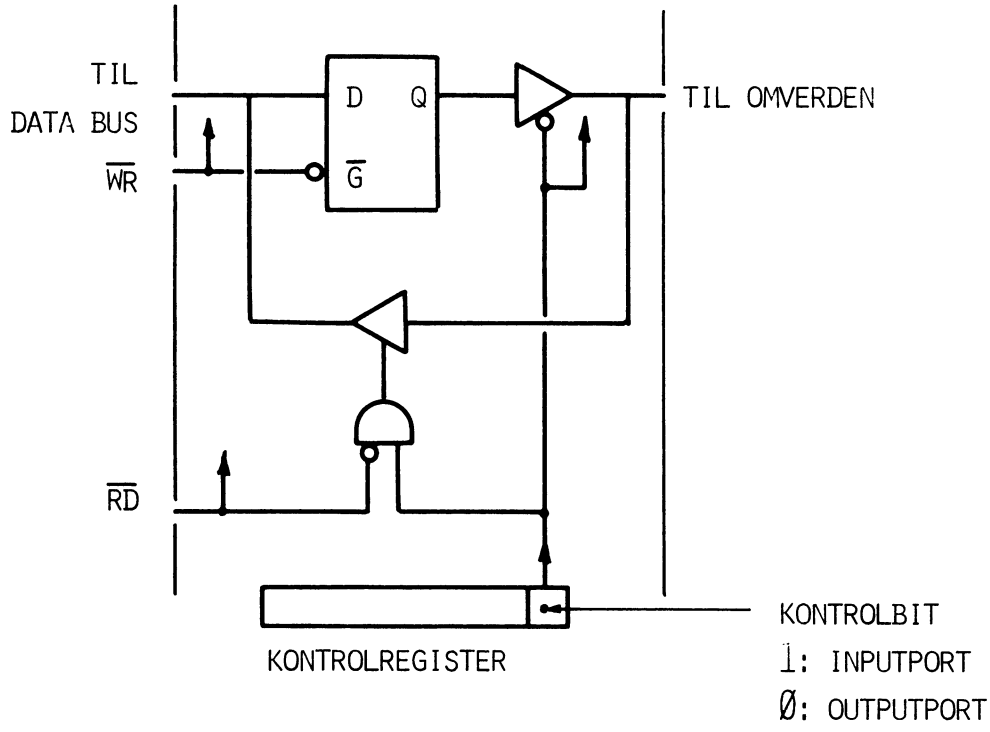
The 8085A microprocessor uses a multiplexed address/data bus that contains the low order 8-bits of address information during the first part of a machine cycle. The same bus contains data at a later time in the cycle. An address latch enable (ALE) signal is provided by the 8085A to be used by the 8212 to latch the address so that it may be available through the whole machine cycle. Note: In this configuration, the MODE input is tied high, keeping the 8212's output buffers turned on at all times.

Programmerbare I/O-kredse.

I princippet er en programmerbar I/O-kreds en kombination af en inputbuffer og en outputlatch.

Hvilken af de to, der skal indkobles bestemmes via et kontrolregister, hvis indhold kan ændres af programmet.

Dette betyder, at et givet ben på kredsen, under programkørslen, kan skiftes fra at være et output til at være et input og omvendt.



Ud over de simple portfunktioner indeholder disse avancerede portkredse ofte handshakelogik og interruptlogik. Andre omfatter timerne og skifteregistre til parallel/serie konvertering og andre igen er kombinationer af porte og hukommelse (RAM eller ROM).

Intel 8255 A.

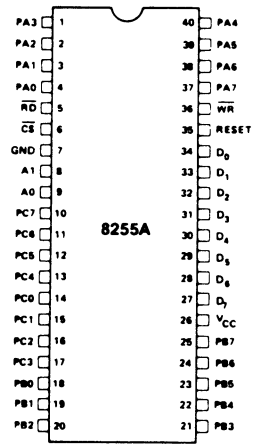
8255A er en programmable peripheral interface-kreds, PPI.

8255A kan arbejde i 3 modes, hvoraf kun mode ø vil blive beskrevet her.

The 8255A is a general purpose programmable I/O device designed for use with both the 8008 and 8080 microprocessors. It has 24 I/O pins which may be individually programmed in two groups of twelve and used in three major modes of operation. In the first mode (Mode 0), each group of twelve I/O pins may be programmed in sets of 4 to be input or output. In Mode 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining four pins three are used for handshaking and interrupt control signals. The third mode of operation (Mode 2) is a Bi-directional Bus mode which uses 8 lines for a bi-directional bus, and five lines, borrowing one from the other group, for handshaking.

Other features of the 8255A include bit set and reset capability and the ability to source 1 mA of current at 1.5 volts. This allows darlington transistors to be directly driven for applications such as printers and high voltage displays.

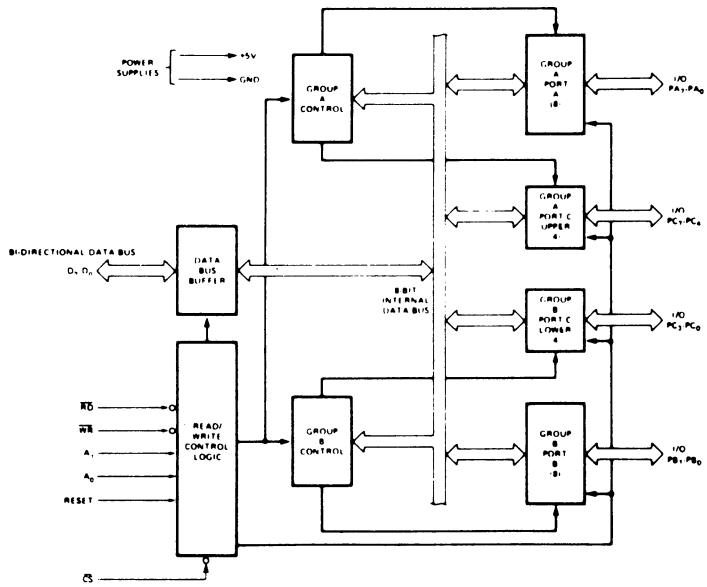
PIN CONFIGURATION



PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	ø VOLTS

8255A BLOCK DIAGRAM



Read/write controllogikken varetager al overførsel af data og kontrolsignaler, såvel internt som externt. Ved hjælp af A_O og A_1 vælges de enkelte registre og porte i 8255A.

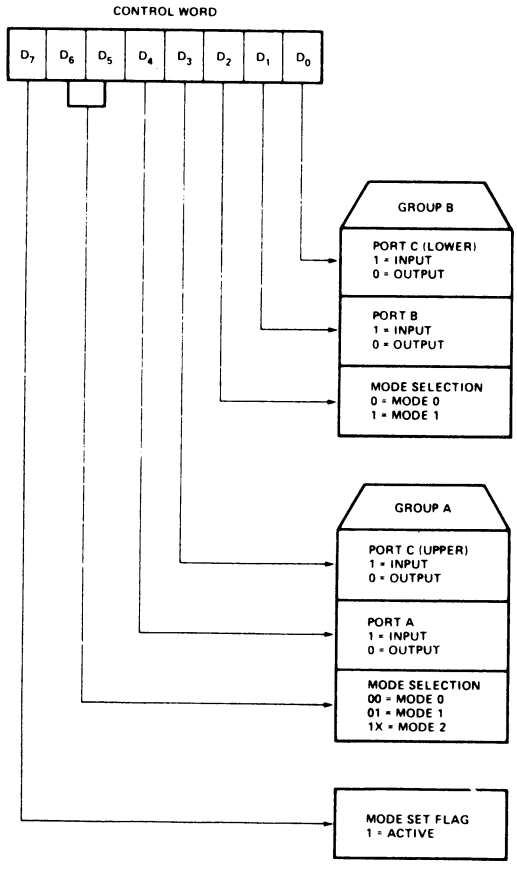
Group A-og group B - control er kredsens kontrolregister i hvilket informationen om mode og dataretning findes.

8255 BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION (READ)
0	0	0	1	0	PORT A → DATA BUS
0	1	0	1	0	PORT B → DATA BUS
1	0	0	1	0	PORT C → DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS → PORT A
0	1	1	0	0	DATA BUS → PORT B
1	0	1	0	0	DATA BUS → PORT C
1	1	1	0	0	DATA BUS → CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS → 3-STATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS → 3-STATE

Control reg.

Controlregistret nås ved at skrive til adresse xxx3H.



Eksempel.

Følgende ønskes:

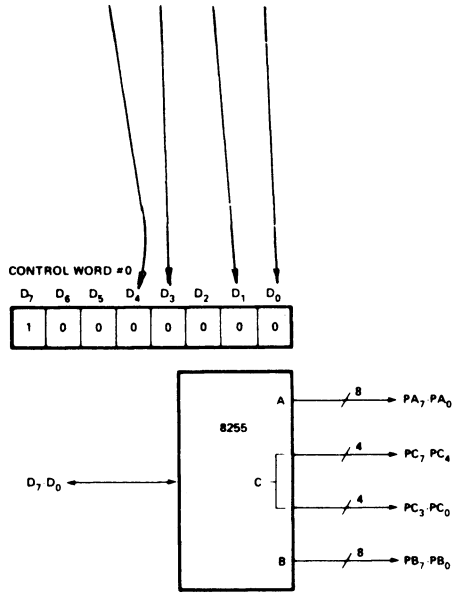
Mode ø, port A som output, port B som input og port C som output.

Controlord = løøøøølø.

Porten kan sættes op i 16 forskellige kombinationer i mode ø.

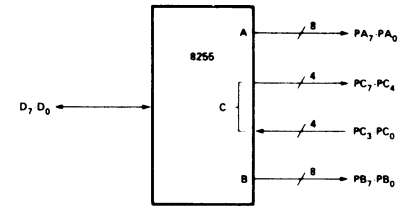
MODE 0 PORT DEFINITION CHART

A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT



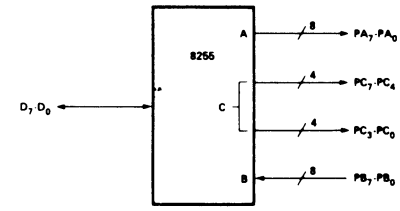
CONTROL WORD #1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	0	1



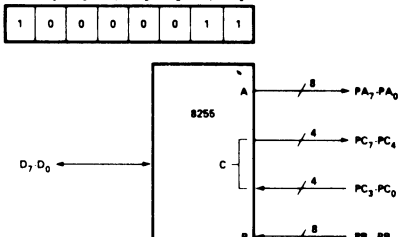
CONTROL WORD #2

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	0



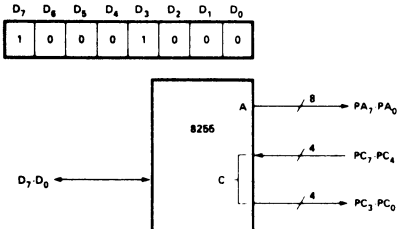
CONTROL WORD #3

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	1



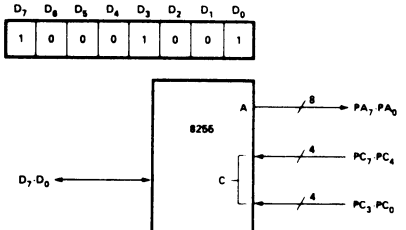
CONTROL WORD #4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



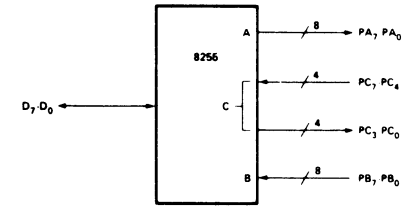
CONTROL WORD #5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



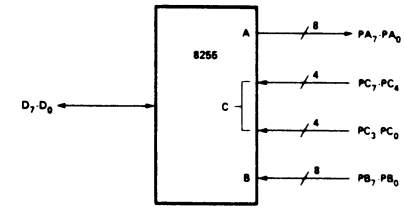
CONTROL WORD #6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



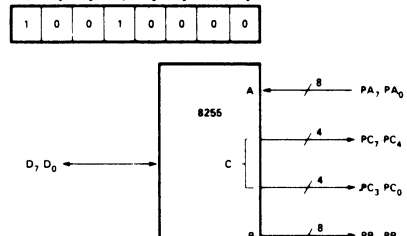
CONTROL WORD #7

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1



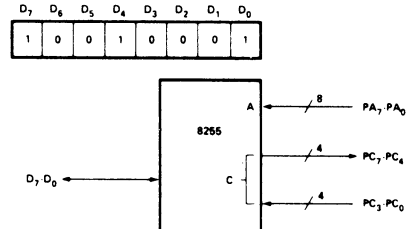
CONTROL WORD #8

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



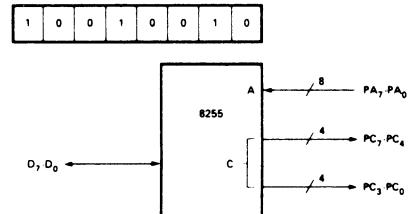
CONTROL WORD #9

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



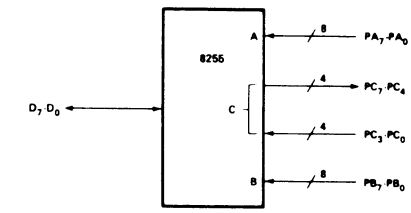
CONTROL WORD #10

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



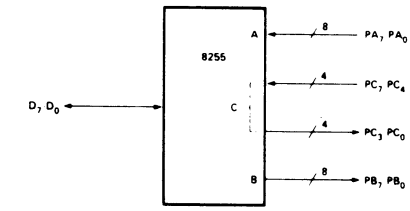
CONTROL WORD #11

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1



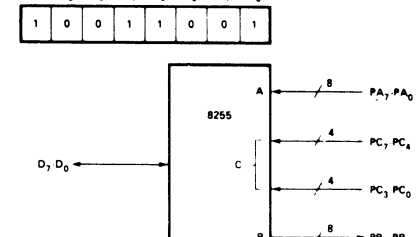
CONTROL WORD #12

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	0



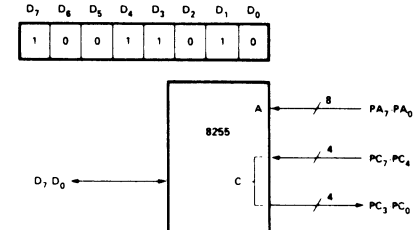
CONTROL WORD #13

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	0	1



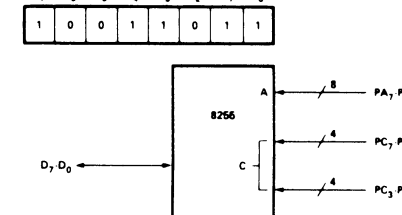
CONTROL WORD #14

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	1	0



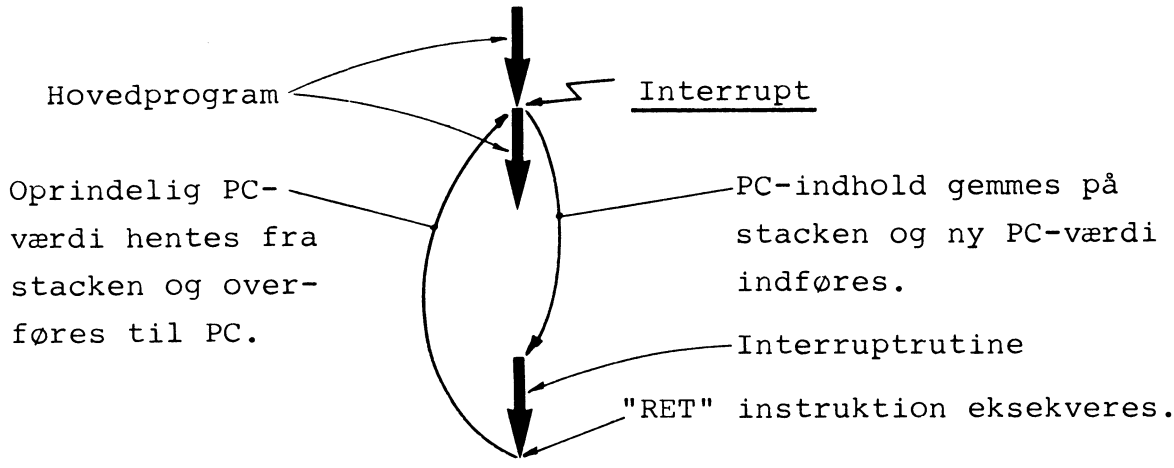
CONTROL WORD #15

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	1	0	1	1



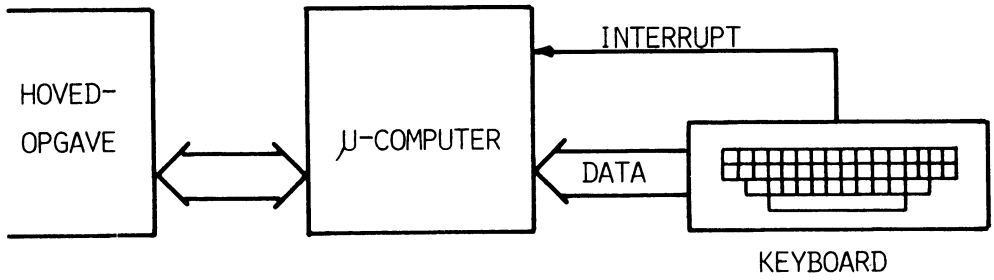
Interrupt, princip.

At interrupte en computer vil sige at afbryde afviklingen af det igangværende program. Efter afbrydelsen gemmes programcounterens indhold i en del af ramlageret, der kaldes stacken, hvorefter PC's indhold skiftes ud med en ny værdi, der peger på begyndelsen af et andet program. CPU'en vil derefter fortsætte med at eksekvere dette. Det program CPU'en hopper til ved en interrupt kaldes ofte interruptrutinen.
Interruptrutinen afsluttes med en RET (return) instruktion. Ved dennes eksekvering udskiftes PC's indhold med den oprindelige værdi, der har været gemt på stacken. Herefter fortsætter CPU'en på det sted i det oprindelige program, hvor den blev afbrudt.



Eksempel på anvendelse.

Figuren viser en µ-computer, der ud over sin hovedopgave skal modtage karakterer fra et keyboard (KB). Ved at lade KB generere en interrupt når der foretages en indtastning, behøver computeren ikke beskæftige sig med keyboardet medmindre der foretages en indtastning. Derved bliver der mere tid til computerens egentlige opgaver, idet betjeningen af KB vil beslaglægge et minimum af den samlede computertid.



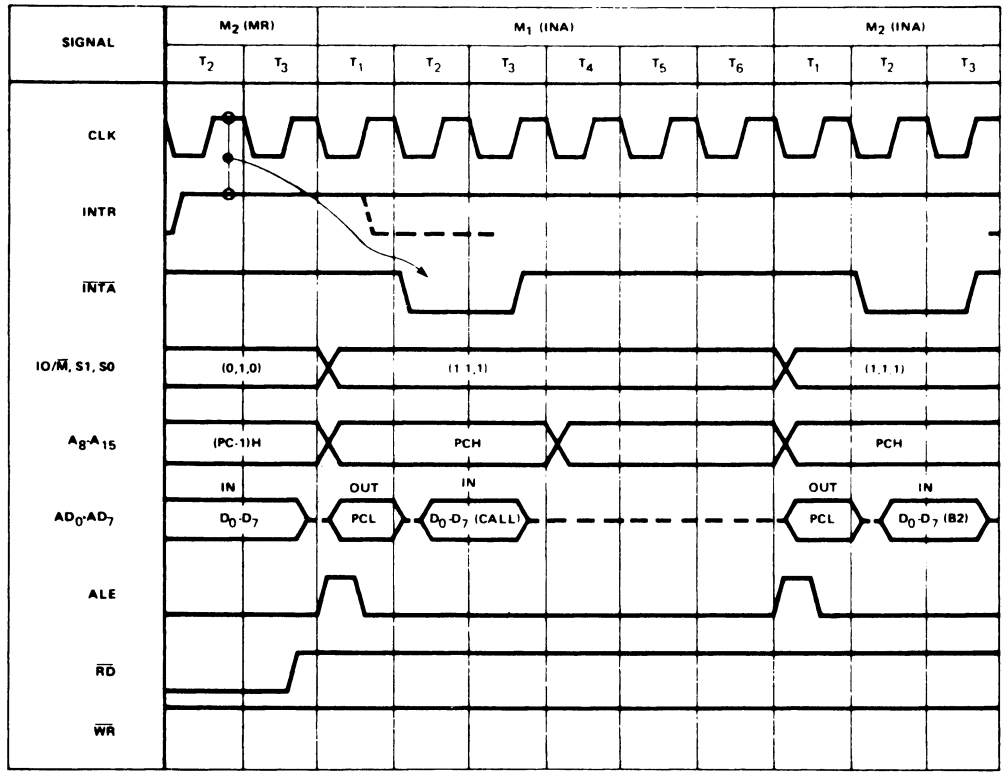
8085's interrupts.

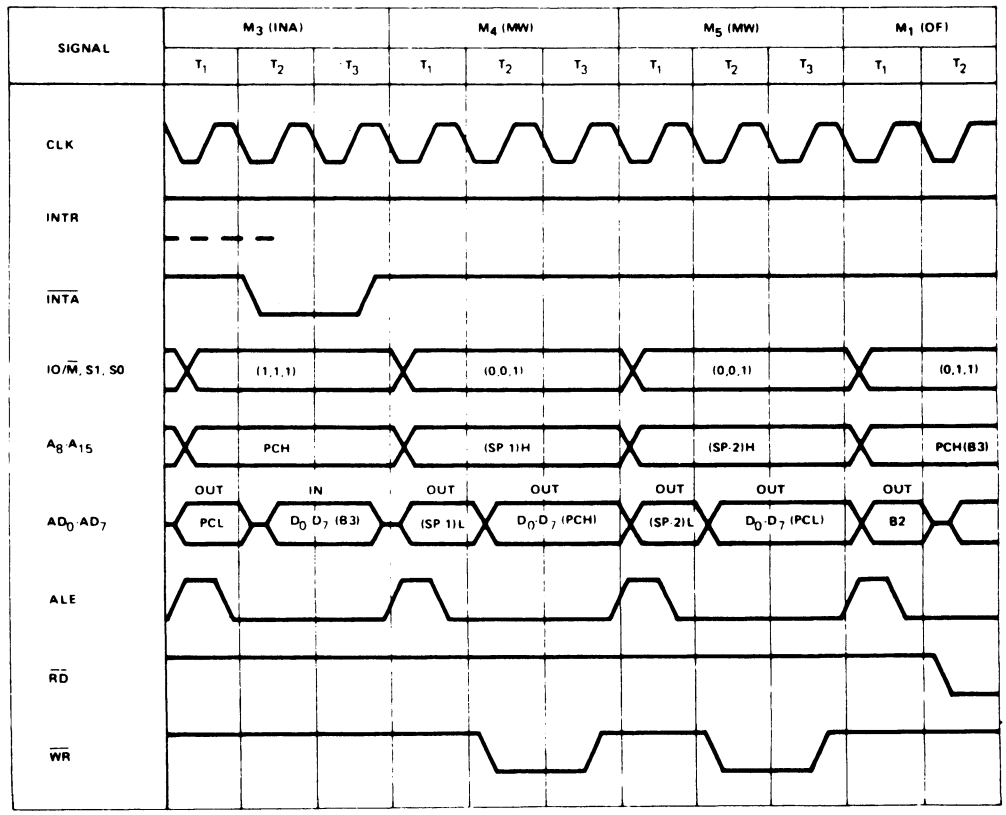
INTR:

De fem interrupts på 8085 er af tre typer. INTR svarer til 8085's INT funktion, det vil sige at det kan maskes ved hjælp af en EI-instruktion (interrupt enable), eller en DI-instruktion (disable interrupt), og CPU'en henter en RST instruktion, der samtidigt skal tilføres databussen.
Dette bevirker et hop til en ud af otte faste hukommelsespladser, hvor programeksekveringen fortsætter.

INTR kan også kontrolleres ved hjælp af 8259, der er en programmerbar interruptcontroler, som kan generere en CALL-instruktion i stedet for RST, hvorved det er muligt at hoppe til en subrutine på en vilkårlig adresse inden for hukommelsen.

INTERRUPT ACKNOWLEDGE MACHINE CYCLES (WITH CALL INSTRUCTION IN RESPONSE TO INTR)



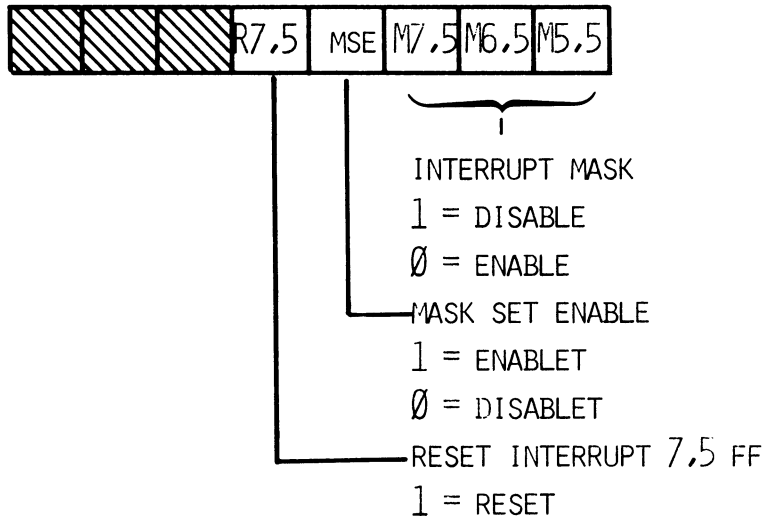


RST 5.5, RST 6.5, RST 7.5.

RST 5.5, RST 6.5 og RST 7.5 er forskellige fra INTR ved at de kan maskes ved hjælp af en SIM-instruktion som enabler eller disabler disse interrupts på basis af data i accumulatoren inden SIM-instruktionen. Den øjeblikkelige status af interruptmaskerne kan læses ved hjælp af en RIM-instruktion.

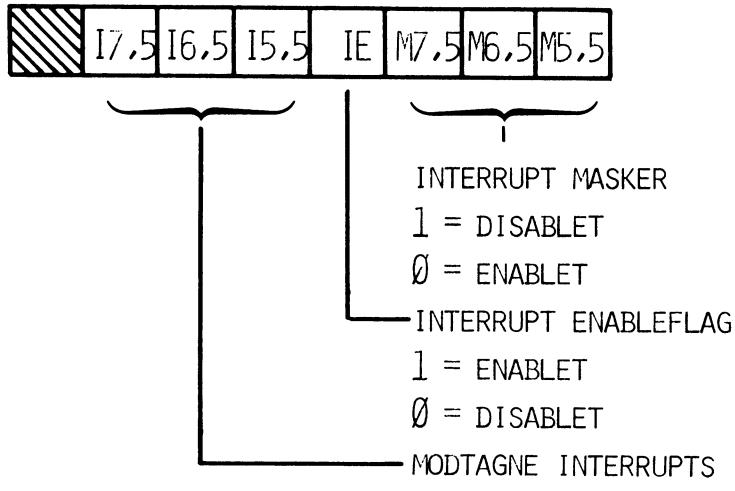
SIM - SET INTERRUPT MASK

ACCUMULATORINDHOLD INDEN SIM



RIM - READ INTERRUPT MASK

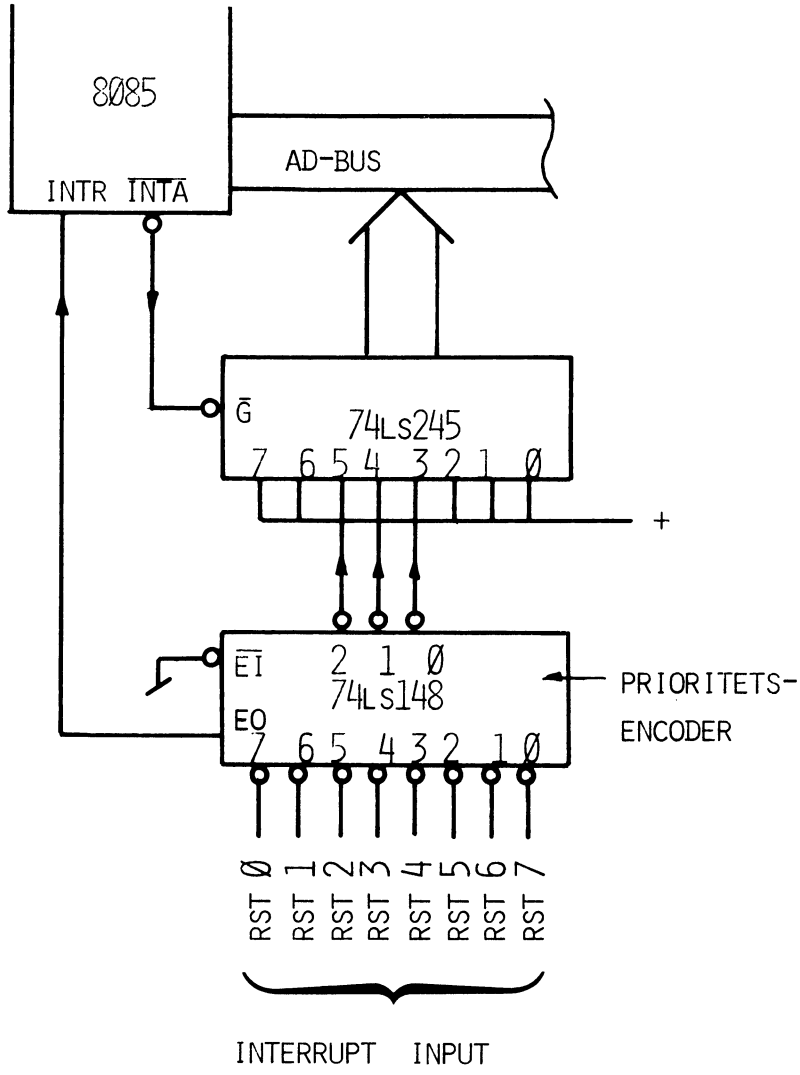
ACCUMULATORINDHOLD EFTER RIM



RST 5.5, RST 6.5 og RST 7.5 enables og disables også ved hjælp af EI- og DI-instruktionerne.

INTA:

Interrupt acknowledge er CPU'ens svarsignal på en akcepteret INTR.
INTA har samme timing som RD og genereres i stedet for denne i en interrupt acknowledge maskincycle.



Diagrammet viser et logisk kredsløb til håndtering af prioriteret RST.
74LS148 er en prioritetsencoder med højeste prioritet på input 7.
Når en af inputtene arkiveres (aktiv low) går EO outputtet high og leverer et interruptinput til CPU'en.
Encoderens output skal lægges ind som bit 3, 4 og 5 i RST-instruktionen og de resterende bit skal være logisk 1.

Sammensætningen af RST-instruktionen sker i en 74LS245 portkreds, hvis enableinput styres af $\overline{\text{INTA}}$.

INTR, RST 5.5 og RST 6.5 er niveaufølsomme input, hvilket betyder at de vil blive opfattet af CPU'en hvis de holdes på high-niveau. RST 7.5 er kantrigget ved hjælp af en intern flip-flop der settes når der kommer en stigende flanke på på RST 7.5 inputtet. Signalet behøver herefter ikke holdes højt, idet flip-floppen vil forblive sat indtil den bliver clearet af en af følgende hændelser.

1) CPU'en reagerer på interrupten og sender et internt resetsignal til RST 7.5 flip-floppen.
2) CPU'en modtager et RESET IN signal inden den når at reagere på RST 7.5 interrupten.
3) CPU'en eksekverer en SIM-instruktion med bit 4 i accumulatoren sat til 1.

TRAP-interrupt.

Den tredie type hardware interrupt er TRAP. Dette input kan ikke disables på nogen måde.
Modtagelsen af en stigende flanke på TRAP-inputtet vil trigge CPU'ens interruptsekvens, blot skal TRAP-inputtet holdes højt indtil CPU'en har akcepteret inputtet.

Interrupternes samplingtidspunkt.

Samplingen af alle interruptinput finder sted på den faldende clockflanke en clockperiode før slutningen af den instruktion i hvilken der opstod et aktivt interruptinput.
For at blive aksepteret skal der være en gyldig interrupt mindst 160 nsek inden samplingstidspunktet.
Det betyder, at INTR, TRAP, RST 5.5 og RST 6.5 skal holdes high i mindst 17 clockperioder plus 100 nsek, idet det antages at interrupten kom "et hårsbredde" for sent til at blive aksepteret ved en given instruktion, og den næste instruktion er en 18 states CALL. Dette forudsætter iøvrigt, at der ikke er nogen wait-states.

Interrupternes prioritet.

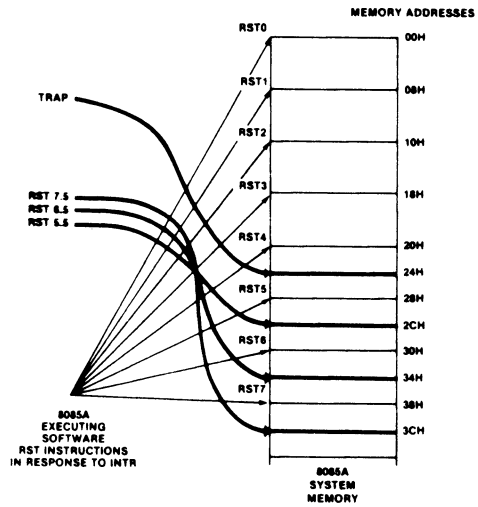
Interruptinputtenes karakteristikker og indbyrdes prioritet er vist i følgende tabel.

Navn	Prioritet	Hop adr.(1)	Trigges af
TRAP	1.	ØØ24H	
RST 7.5	2.	ØØ3cH	
RST 6.5	3.	ØØ34H	
RST 5.5	4.	ØØ2cH	
INTR	5.	(2)	

(1) Programcounterens indhold skubbes på stacken inden CPU'en barncher til den angivne adr.

(2) Afhænger af den tilførte instruktion.

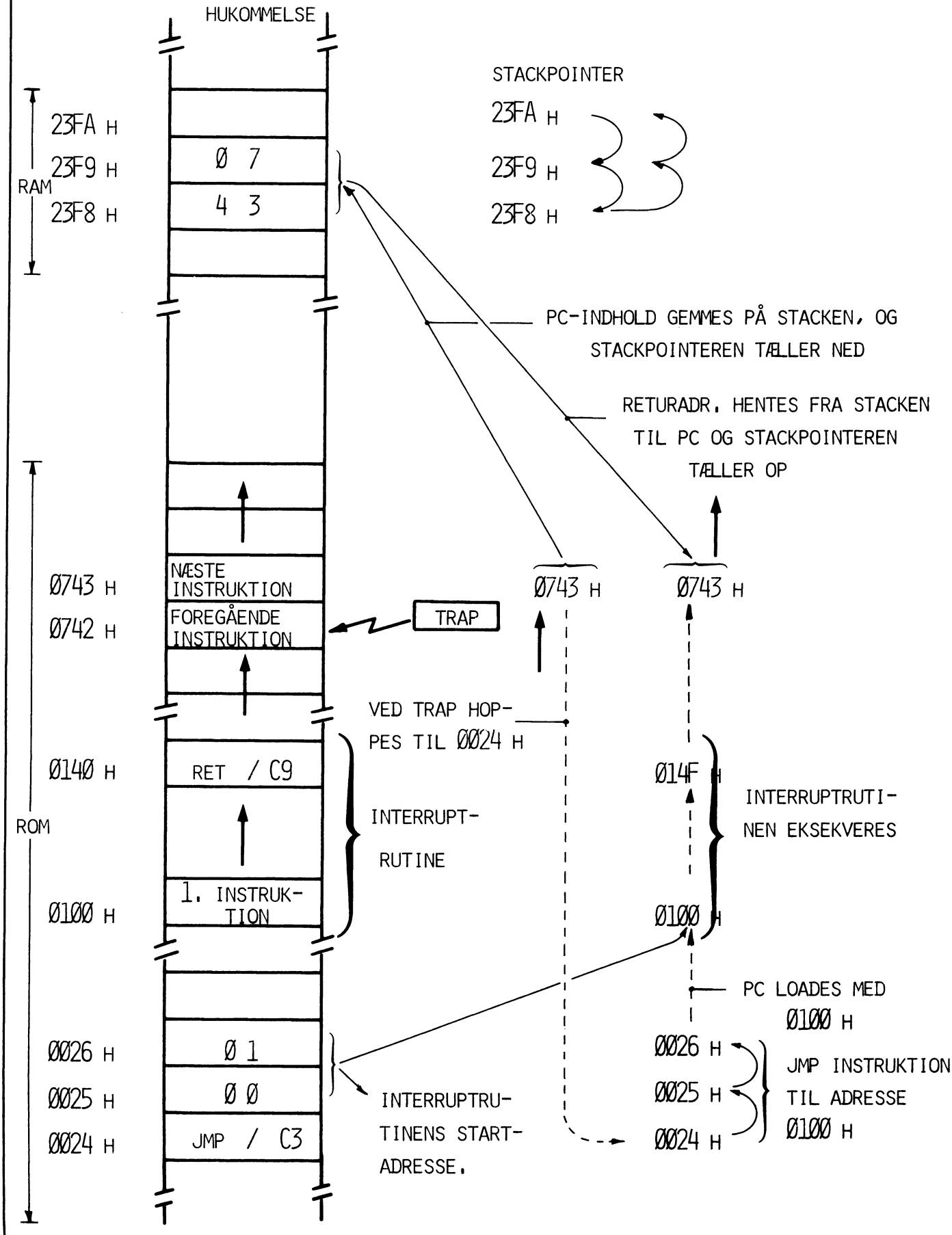
Handware og software RST HOP adresser.



Eksempel på forløb af et TRAP interrupt.

Se tegning.

Under eksekveringen af hovedprogrammet er CPU'en nået til adr. ø742H, da der modtages et trap interrupt.
Når interruptet er erkendt af CPU'en, fortsættes med en stackoperation hvorunder programcounterens indhold lagres på to på hinanden følgende pladser i stacken.
Derefter udskiftes programcounterens indhold med øø24H og CPU'en går ind i en M1 maskincycle.
På adr. øø24H, øø25H og øø26H ligger der en JMP-instruktion til adr. øløøH der er interruptrutinens startadresse.
Som den sidste instruktion i interruptrutinen findes en RET (return).
Denne bevirker at CPU'en foretager en stackoperation hvorunder indholdet af de pladser stackpointeren peger på overføres til programcounteren.
Programcounteren indeholder nu adressen på den instruktion i hovedprogrammet, der ligger efter det sted hvor interrupten opstod. CPU'en går nu ind i en M1 - maskincycle og fortsætter eksekveringen af hovedprogrammet.



MASKINKODER OG MNEONIC

DATA TRANSFER GROUP

Move

MOV	A,A	7F
	A,B	78
	A,C	79
	A,D	7A
	A,E	7B
	A,H	7C
	A,L	7D
	A,M	7E
MOV	B,A	47
	B,B	40
	B,C	41
	B,D	42
	B,E	43
	B,H	44
	B,L	45
	B,M	46
MOV	C,A	4F
	C,B	48
	C,C	49
	C,D	4A
	C,E	4B
	C,H	4C
	C,L	4D
	C,M	4E
MOV	D,A	57
	D,B	50
	D,C	51
	D,D	52
	D,E	53
	D,H	54
	D,L	55
	D,M	56

Move (cont)

MOV	E,A	5F
	E,B	58
	E,C	59
	E,D	5A
	E,E	5B
	E,H	5C
	E,L	5D
	E,M	5E
MOV	H,A	67
	H,B	60
	H,C	61
	H,D	62
	H,E	63
	H,H	64
	H,L	65
	H,M	66
MOV	L,A	6F
	L,B	68
	L,C	69
	L,D	6A
	L,E	6B
	L,H	6C
	L,L	6D
	L,M	6E
MOV	M,A	77
	M,B	70
	M,C	71
	M,D	72
	M,E	73
	M,H	74
	M,L	75

XCHG EB

Move Immediate

MVI	A, byte	3E
	B, byte	06
	C, byte	0E
	D, byte	16
	E, byte	1E
	H, byte	26
	L, byte	2E
	M, byte	36

Load Immediate

LXI	B, dble	01
	D, dble	11
	H, dble	21
	SP, dble	31

Load Store

LDAX B	0A
LDAX D	1A
LHLD adr	2A
LDA adr	3A
STAX B	02
STAX D	12
SHLD adr	22
STA adr	32

byte = constant, or logical/arithmetic expression that evaluates to an 8-bit data quantity (Second byte of 2-byte instructions)

dble = constant, or logical/arithmetic expression that evaluates to a 16-bit data quantity (Second and Third bytes of 3-byte instructions)

adr = 16-bit address (Second and Third bytes of 3-byte instructions)

* = all flags (C, Z, S, P, AC) affected

** = all flags except CARRY affected, (exception INX and DCX affect no flags)

† = only CARRY affected

00	NOP		2B	DCX	H	56	MOV	D,M
01	LXI	B,dble	2C	INR	L	57	MOV	D,A
02	STAX	B	2D	DCR	L	58	MOV	E,B
03	INX	B	2E	MVI	L,byte	59	MOV	E,C
04	INR	B	2F	CMA		5A	MOV	E,D
05	DCR	B	30	SIM*		5B	MOV	E,E
06	MVI	B,byte	31	LXI	SP,dble	5C	MOV	E,H
07	RLC		32	STA	adr	5D	MOV	E,L
08	- - -		33	INX	SP	5E	MOV	E,M
09	DAD	B	34	INR	M	5F	MOV	E,A
0A	LDAX	B	35	DCR	M	60	MOV	H B
0B	DCX	B	36	MVI	M,byte	61	MOV	H,C
0C	INR	C	37	STC		62	MOV	H,D
0D	DCR	C	38	- - -		63	MOV	H,E
0E	MVI	C,byte	39	DAD	SP	64	MOV	H,H
0F	RRC		3A	LDA	adr	65	MOV	H,L
10	- - -		3B	DCX	SP	66	MOV	H,M
11	LXI	D,dble	3C	INR	A	67	MOV	H,A
12	STAX	D	3D	DCR	A	68	MOV	L,B
13	INX	D	3E	MVI	A,byte	69	MOV	L,C
14	INR	D	3F	CMC		6A	MOV	L,D
15	DCR	D	40	MOV	B,B	6B	MOV	L,E
16	MVI	D,byte	41	MOV	B,C	6C	MOV	L,H
17	RAL		42	MOV	B,D	6D	MOV	L,L
18	- - -		43	MOV	B,E	6E	MOV	L,M
19	DAD	D	44	MOV	B,H	6F	MOV	L,A
1A	LDAX	D	45	MOV	B,L	70	MOV	M,B
1B	DCX	D	46	MOV	B,M	71	MOV	M,C
1C	INR	E	47	MOV	B,A	72	MOV	M,D
1D	DCR	E	48	MOV	C,B	73	MOV	M,E
1E	MVI	E,byte	49	MOV	C,C	74	MOV	M,H
1F	RAR		4A	MOV	C,D	75	MOV	M,L
20	RIM*		4B	MOV	C,E	76	HLT	
21	LXI	H,dble	4C	MOV	C,H	77	MOV	M,A
22	SHLD	adr	4D	MOV	C,L	78	MOV	A,B
23	INX	H	4E	MOV	C,M	79	MOV	A,C
24	INR	H	4F	MOV	C,A	7A	MOV	A,D
25	DCR	H	50	MOV	D,B	7B	MOV	A,E
26	MVI	H,byte	51	MOV	D,C	7C	MOV	A,H
27	DAA		52	MOV	D,D	7D	MOV	A,L
28	- - -		53	MOV	D,E	7E	MOV	A,M
29	DAD	H	54	MOV	D,H	7F	MOV	A,A
2A	LHLD	adr	55	MOV	D,L	80	ADD	B

*8085 Only

ARITHMETIC AND LOGICAL GROUP

Add*

ADD	A	87
	B	80
	C	81
	D	82
	E	83
	H	84
	L	85
	M	86
ADC	A	8F
	B	88
	C	89
	D	8A
	E	8B
	H	8C
	L	8D
	M	8E

Subtract*

SUB	A	97
	B	90
	C	91
	D	92
	E	93
	H	94
	L	95
	M	96
SBB	A	9F
	B	98
	C	99
	D	9A
	E	9B
	H	9C
	L	9D
	M	9E

Double Add †

DAD	B	09
	D	19
	H	29
	SP	39

Increment**

INR	A	3C
	B	04
	C	0C
	D	14
	E	1C
	H	24
	L	2C
	M	34
INX	B	03
	D	13
	H	23
	SP	33

Decrement**

DCR	A	3D
	B	05
	C	0D
	D	15
	E	1D
	H	25
	L	2D
	M	35
DCX	B	0B
	D	1B
	H	2B
	SP	3B

Specials

DAA*	27
CMA	2F
STC†	37
CMC†	3F

Rotate †

RLC	07
RRC	0F
RAL	17
RAR	1F

Logical*

ANA	A	A7
	B	A0
	C	A1
	D	A2
	E	A3
	H	A4
	L	A5
	M	A6
XRA	A	AF
	B	A8
	C	A9
	D	AA
	E	AB
	H	AC
	L	AD
	M	AE
ORA	A	B7
	B	B0
	C	B1
	D	B2
	E	B3
	H	B4
	L	B5
	M	B6
CMP	A	BF
	B	B8
	C	B9
	D	BA
	E	BB
	H	BC
	L	BD
	M	BE

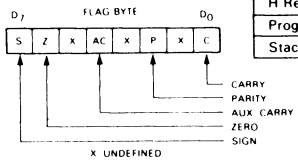
Arith & Logical Immediate

ADI byte	C6
ACI byte	CE
SUI byte	D6
SBI byte	DE
ANI byte	E6
XRI byte	EE
ORI byte	F6
CPI byte	FE

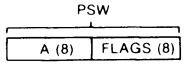
81	ADD	C	AC	XRA	H	D7	RST	2
82	ADD	D	AD	XRA	L	D8	RC	
83	ADD	E	AE	XRA	M	D9	- - -	
84	ADD	H	AF	XRA	A	DA	JC	adr
85	ADD	L	B0	ORA	B	DB	IN	byte
86	ADD	M	B1	ORA	C	DC	CC	adr
87	ADD	A	B2	ORA	D	DD	- - -	
88	ADC	B	B3	ORA	E	DE	SBI	byte
89	ADC	C	B4	ORA	H	DF	RST	3
8A	ADC	D	B5	ORA	L	E0	RPO	
8B	ADC	E	B6	ORA	M	E1	POP	H
8C	ADC	H	B7	ORA	A	E2	JPO	adr
8D	ADC	L	B8	CMP	B	E3	XTHL	
8E	ADC	M	B9	CMP	C	E4	CPO	adr
8F	ADC	A	BA	CMP	D	E5	PUSH	H
90	SUB	B	BB	CMP	E	E6	ANI	byte
91	SUB	C	BC	CMP	H	E7	RST	4
92	SUB	D	BD	CMP	L	E8	RPE	
93	SUB	E	BE	CMP	M	E9	PCHL	
94	SUB	H	BF	CMP	A	EA	JPE	adr
95	SUB	L	C0	RNZ		EB	XCHG	
96	SUB	M	C1	POP	B	EC	CPE	adr
97	SUB	A	C2	JNZ	adr	ED	- - -	
98	SBB	B	C3	JMP	adr	EE	XRI	byte
99	SBB	C	C4	CNZ	adr	EF	RST	5
9A	SBB	D	C5	PUSH	B	F0	RP	
9B	SBB	E	C6	ADI	byte	F1	POP	PSW
9C	SBB	H	C7	RST	0	F2	JP	adr
9D	SBB	L	C8	RZ		F3	DI	
9E	SBB	M	C9	RET		F4	CP	adr
9F	SBB	A	CA	JZ	adr	F5	PUSH	PSW
A0	ANA	B	CB	- - -		F6	ORI	byte
A1	ANA	C	CC	CZ	adr	F7	RST	6
A2	ANA	D	CD	CALL	adr	F8	RM	
A3	ANA	E	CE	ACI	byte	F9	SPHL	
A4	ANA	H	CF	RST	1	FA	JM	adr
A5	ANA	L	D0	RNC		FB	EI	
A6	ANA	M	D1	POP	D	FC	CM	adr
A7	ANA	A	D2	JNC	adr	FD	- - -	
A8	XRA	B	D3	OUT	byte	FE	CPI	byte
A9	XRA	C	D4	CNC	adr	FF	RST	7
AA	XRA	D	D5	PUSH	D			
AB	XRA	E	D6	SUI	byte			

INTERNAL REGISTER ORGANIZATION

A Reg (8)	
B Reg (8)	C Reg (8)
D Reg (8)	E Reg (8)
H Reg (8)	L Reg (8)
Program Counter (16)	
Stack Pointer (16)	



REGISTER-PAIR ORGANIZATION



NOTE Leftmost Byte is high-order byte for arithmetic operations and addressing Left byte is pushed on stack first Right byte is popped first

B	(B/C) (16)
D	(D/E) (16)
H	(H/L) (16)
Prog Ctr	(16)
Stack Ptr	(16)

BRANCH CONTROL INSTRUCTIONS

Flag Condition	Jump		Call		Return	
Zero True	JZ	CA	CZ	CC	RZ	C8
Zero False	JNZ	C2	CNZ	C4	RNZ	C0
Carry True	JC	DA	CC	DC	RC	D8
Carry False	JNC	D2	CNC	D4	RNC	D0
Sign Positive	JP	F2	CP	F4	RP	F0
Sign Negative	JM	FA	CM	FC	RM	F8
Parity Even	JPE	EA	CPE	EC	RPE	E8
Parity Odd	JPO	E2	CPO	E4	RPO	E0
Unconditional	JMP	C3	CALL	CD	RET	C9

ACCUMULATOR OPERATIONS

	Code	Function
XRA A	AF	Clear A and Clear Carry
ORA A	B7	Clear Carry
CMC	3F	Complement Carry
CMA	2F	Complement Accumulator
STC	37	Set Carry
RLC	07	Rotate Left
RRC	0F	Rotate Right
RAL	17	Rotate Left Thru Carry
RAR	1F	Rotate Right Thru Carry
DAA	27	Decimal Adjust Accum

REGISTER PAIR AND STACK OPERATIONS

		Register Pair					
	PSW (A/F)	B (B/C)	D (D/E)	H (H/L)	SP	PC	Function
INX		03	13	23	33		Increment Register Pair
DCX		0B	1B	2B	3B		Decrement Register Pair
LDAX		0A	1A	7E(1)			Load A Indirect (Reg Pair holds Adrs)
STAX		02	12	77(2)			Store A Indirect (Reg Pair holds Adrs)
LHLD				2A			Load H/L Direct (Bytes 2 and 3 hold Adrs)
SHLD				22			Store H/L Direct (Bytes 2 and 3 hold Adrs)
LXI		01	11	21	31	C3(3)	Load Reg Pair Immediate (Bytes 2 and 3 hold immediate data)
PCHL						E9	Load PC with H/L (Branch to Adrs in H/L)
XCHG			EB				Exchange Reg Pairs D/E and H/L
DAD		09	19	29	39		Add Reg Pair to H/L
PUSH	F5	C5	D5	E5			Push Reg Pair on Stack
POP	F1	C1	D1	E1			Pop Reg Pair off Stack
XTHL				E3			Exchange H/L with Top of Stack
SPHL					F9		Load SP with H/L

Notes 1 This is MOV A,M 2 This is MOV M,A 3 This is JMP

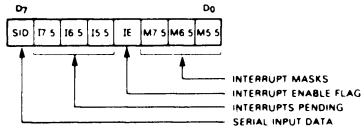
RESTART TABLE

Name	Code	Restart Address
RST 0	C7	0000_{16}
RST 1	CF	0008_{16}
RST 2	D7	0010_{16}
RST 3	DF	0018_{16}
RST 4	E7	0020_{16}
TRAP	Hardware* Function	0024_{16}
RST 5	EF	0028_{16}
RST 5 5	Hardware* Function	$002C_{16}$
RST 6	F7	0030_{16}
RST 6 5	Hardware* Function	0034_{16}
RST 7	FF	0038_{16}
RST 7 5	Hardware* Function	$003C_{16}$

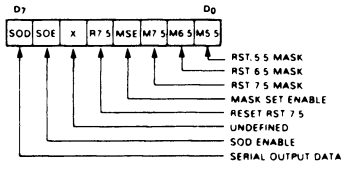
*NOTE The hardware functions refer to the on-chip Interrupt feature of the 8085 only

USE OF THE A REGISTER BY RIM AND SIM INSTRUCTIONS (8085 ONLY)

A REGISTER AFTER EXECUTING RIM



A REGISTER BEFORE EXECUTING SIM



BRANCH CONTROL GROUP

Jump

JMP adr	C3
JNZ adr	C2
JZ adr	CA
JNC adr	D2
JC adr	DA
JPO adr	E2
JPE adr	EA
JP adr	F2
JM adr	FA
PCHL	E9

Call

CALL adr	CD
CNZ adr	C4
CZ adr	CC
CNC adr	D4
CC adr	DC
CPO adr	E4
CPE adr	EC
CP adr	F4
CM adr	FC

Return

RET	C9
RNZ	C0
RZ	C8
RNC	D0
RC	D8
RPO	E0
RPE	E8
RP	F0
RM	F8

Restart

RST	0	C7
	1	CF
	2	D7
	3	DF
	4	E7
	5	EF
	6	F7
	7	FF

I/O AND MACHINE CONTROL

Stack Ops

PUSH	B	C5
	D	D5
	H	E5
	PSW	F5
POP	B	C1
	D	D1
	H	E1
	PSW*	F1
XTHL		E3
SPHL		F9

Input Output

OUT byte	D3
IN byte	DB

Control

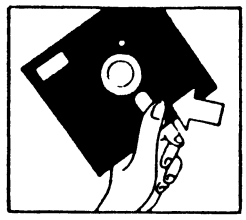
DI	F3
EI	FB
NOP	00
HLT	76

New Instructions (8085 Only)

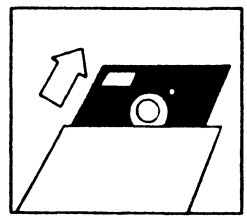
RIM	20
SIM	30

Disketter er ret følsomme og kan nemt beskadiges, hvis de ikke behandles rigtigt.

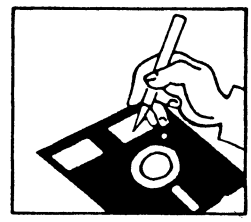
I det nedenstående er givet nogle anvisninger for, hvordan en diskette bør behandles.



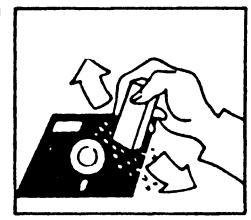
Undgå at berøre diskettens overflade.



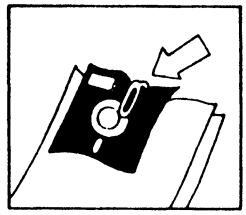
Tag ikke disketten ud af omslaget med mindre det er strengt nødvendigt.



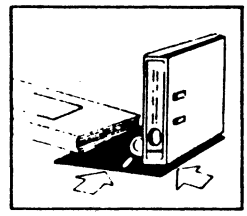
Brug ikke kuglepen eller andet hårdt skriveredskab.



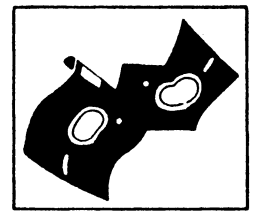
Visk ikke ud på omslaget.



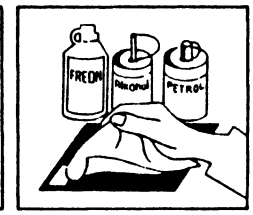
Sæt ikke clips eller gummibånd på disketten.



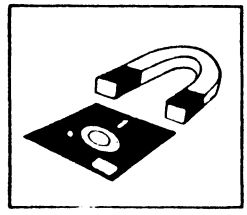
Læg ikke tunge objekter på disketten.



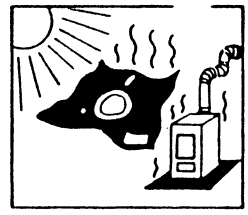
Fold eller bøj ikke disketten.



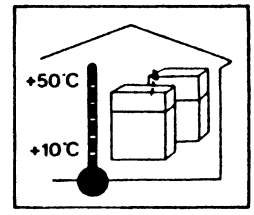
Brug ikke flydende rensevæsker, så som alkohol, freon eller lignende til rensning af disketten.



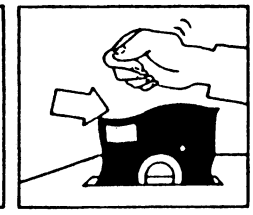
Udsæt ikke disketten for magnetfelter



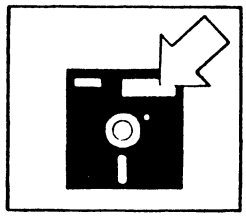
Lad ikke disketten få stærkt sollys eller varme.



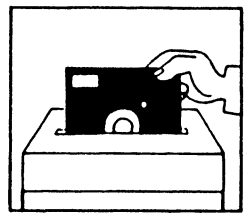
Disketten må ikke bruges eller opbevares over + 50°C og under + 10°C.



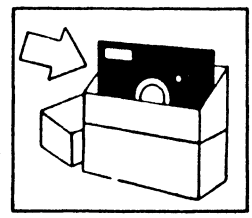
Brug ikke magt når disketten isættes.



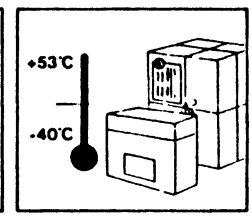
Brug altid en filtpen når De skriver på diskette omslaget.



Tag altid disketten i dens øverste hjørne – sæt den omhyggeligt i maskinen.



Disketten skal altid opbevares på højkant.



Vær altid sikker på, at disketten under forsendelse ikke udsættes for temp. over + 53°C og under – 40°C.

Maxell-importøren Christian Bruhn har lavet denne informative advarselstavle, der bestemt skal tages alvorligt!

SYSTEM SOFTWARE

For at programmere en computer, skrives programmet i hånden i et computersprog. Fra programmet er skrevet i hånden til det kører i computeren anvendes forskellige SYSTEMPROGRAMMER. Disse kan groft opdeles således:

1) LANGUAGE TRANSLATORS, - (oversættere).
Anvendes til at oversætte det af brugeren skrevne program til maskinkode.

2) EDITORS, - (tekstbehandlere).
Editoren sætter programmøren i stand til at ændre, rette og tilføje i hans program på det sprogniveau programmet er skrevet.

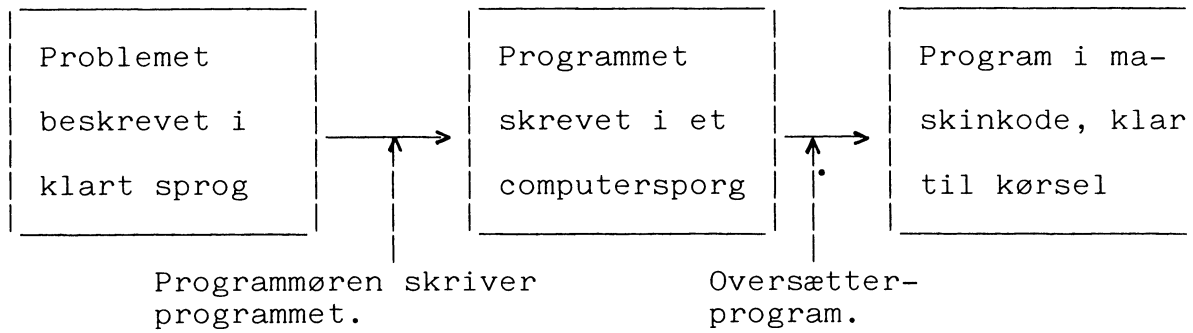
3) LINKER / LOADER.
Linkeren anvendes til at sammenkoble forskellige programmoduler. Loaderen forsyner maskinkoderne med absolutte adresser og lægger (loader) programmet ind i hukommelsen hvor det kan eksekveres (i udviklingssystemet).

4) DEBUGGERS, - (fejlfindere).
Debuggeren anvendes til at følge programeksekveringen i computeren for at kontrollere at programmet virker korrekt.

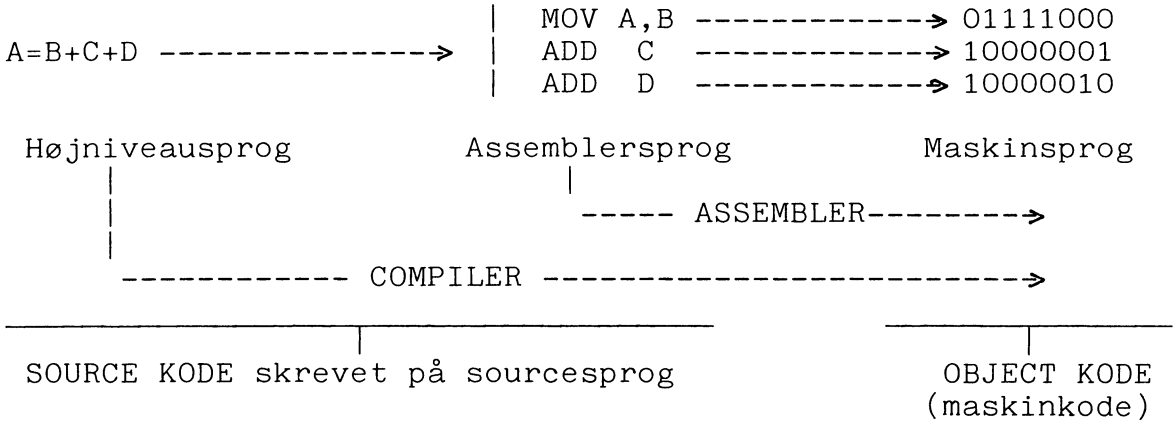
5) MONITORS.
Monitoren anvendes til at vælge mellem de forskellige systemprogrammer, samt kontrollerer computeren i forbindelse med kommunikationen til brugeren (programmøren).

LANGUAGE TRANSLATORS

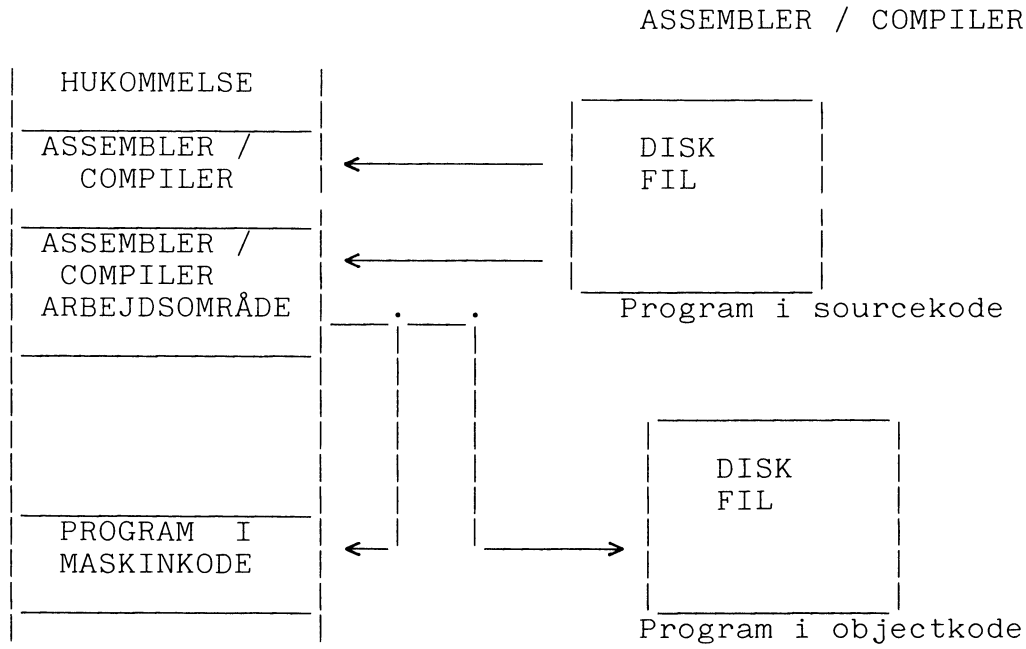
Desværre forstår computere ikke Engelsk, og programmørerne er ikke tilbøjelige til at skrive i et sprog computeren forstår, nemlig bitmønstre der kan lagres i computerens hukommelse.
Til at spænde over denne "sprogbariere" anvendes et såkaldt COMPUTERSPROG.
Et computersprog giver programmøren et begrænset sæt regler efter hvilken han kan skrive "sætninger" i computersprog.
Denne begrænsning er nødvendig, eftersom det kun er meningsfyldt at skrive i et computersprog hvis det til slut kan oversættes til maskinkode af computeren selv, d.v.s. ved hjælp af et computerprogram der allerede findes i maskinkode.



Ved oversættere skelnes normalt mellem ASSEMBLERE og COMPILERE. Assemblersprog ligger ret tæt på maskinkode. Mere avancerede computersprog er såkaldte HØJ-NIVEAU SPROG som f.eks. PLM, FOPTRAN, COBOL o.s.v.
I assemblersprog svarer hver "sætning" til een instruktion i computerens instruktionssæt.
En sætning i et højniveausprog svarer til en hel sekvens af instruktioner. Assemblersprog ligger således på et niveau mellem højniveausprog og maskinkode.



ASSEMBLER- eller COMPILERPROGRAMMET skal findes/overføres i form af maskinkode i/til computerens hukommelse. Dette program fortolker så hver sætning i SOURCEKODEN og afleverer OBJECTKODEN, enten direkte som maskinkode et andet sted i hukommelsen, eller til en extern enhed som f. eks. en disc.

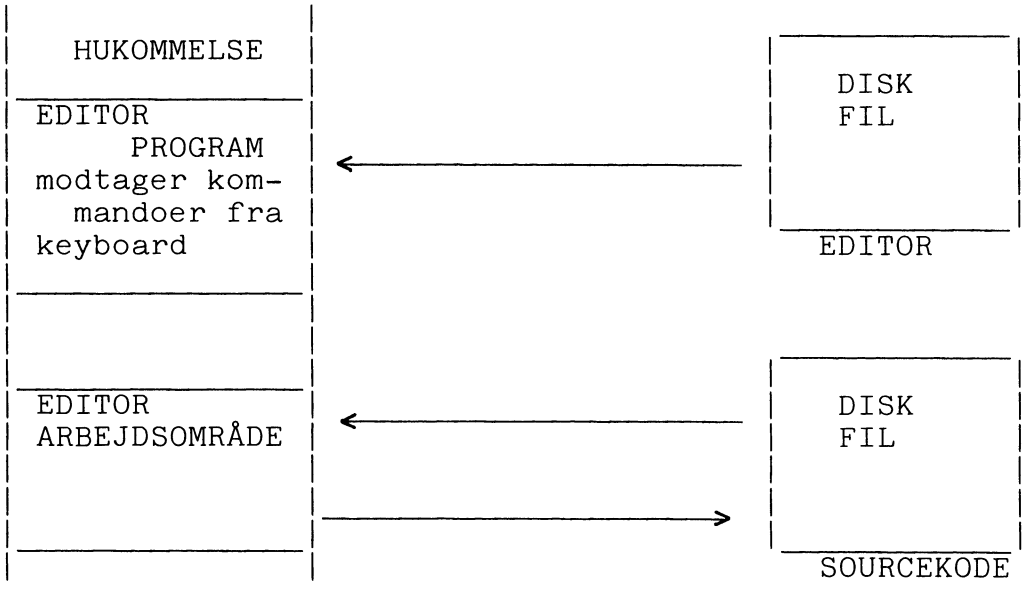


I forbindelse med microcomputere sker assembleringen normalt ikke i den maskine hvor det udviklede program skal anvendes, men derimod i en speciel computer der er udstyret med faciliteter (printer, skærmterminal og discdrives) der gør den velegnet som programudviklingssystem.
Til slut overføres objectkoden til en ROM (EPROM) der flyttes til den computer programmet er udviklet til.

EDITOR

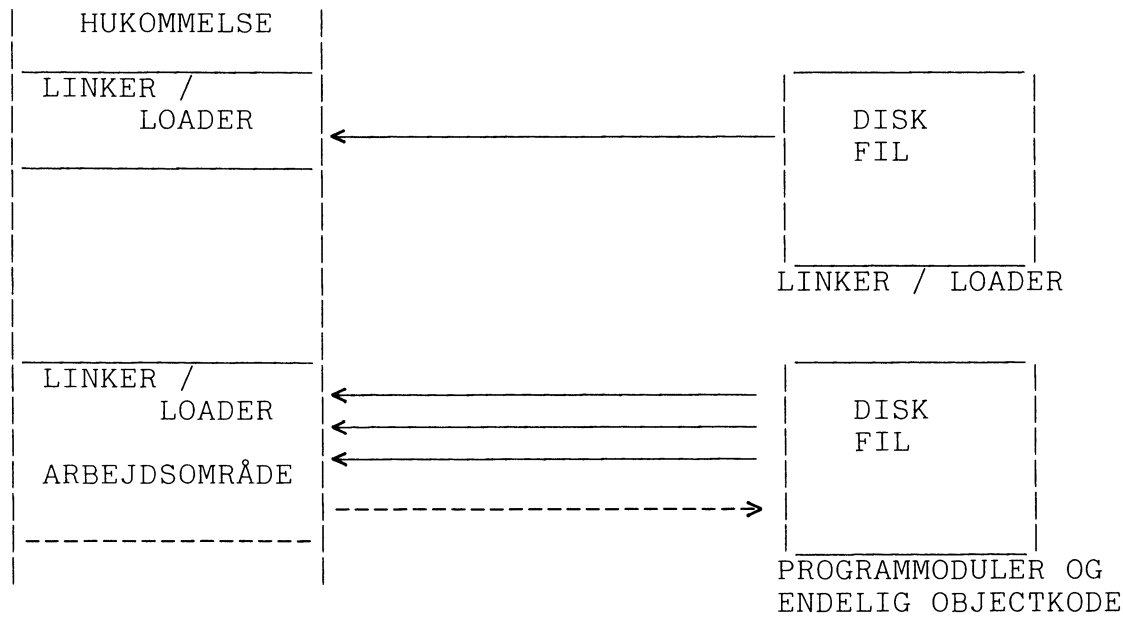
Det er ofte nødvendigt at ændre på programmer, enten på grund af fejl fundet under afprøvningen eller fordi nye krav kræver en modifikation.
Ændringer foretages normalt i SOURCEKODEN og ikke i OBJECTKODEN, fordi SOURCEKODEN er den nemmeste at læse, idet den er en detaljeret beskrivelse af objectkoden, med tilhørende kommentarer.
Hver gang SOURCEKODEN er ændret (updated) kan ASSEMBLEREN (COMPILEREN) generere en ny OBJECTKODE. EDITOREN er et program der anvendes til skrivning og modifikation af programmer skrevet i et SOURCESPROG, idet man nemt kan finde en sætning (linie) man vil ændre, eller der kan indsættes nye linier. Programmøren behøver altså ikke at skrive og indtaste et helt nyt program fra ende til anden.

Editorprogrammet tolker de kommandoer programmøren indtaster, finder den/de linie(r) der skal ændres i den enhed hvor SOURCEKODEN findes og når de ønskede ændringer er foretaget sendes den nye tekst tilbage til lagerenheden.



LINKER / LOADER

Som tidligere nævnt, kan et program kun eksekveres når det findes i form af maskinkoder i hukommelsen.
Programmet kan skrives i et eller flere moduler (SOURCE MODULER), som uafhængigt af hinanden er oversat til RELOKERBAR OBJECTKODE og lagret som objectmoduler på et externt lagermedie, f. eks. floppy disc.
LINKER / LOADER programmet kombinerer til slut de enkelte moduler til eet maskinkodeprogram som anbringes (LOADES) i computerens hukommelse hvor det kan eksekveres, eller på et externt lagermedie.



DEBUGGERS

Så snart det udviklede program findes i computerens hukommelse i form af maskinkoder kan det eksekveres, og dets funktion kontrolleres.
Til hjælp for denne afprøvning findes et specielt DEBUG-program. Ved hjælp af dette er det muligt at single-steppe igennem programmet; vise indholdet af forskellige registre og hukommelsespladser, samt ændre instruktioner og data i programmet.

MONITORS

Da der under en programudvikling er brug for at have forskellige programmer (EDITOR, ASSEMBLER, LINKER/LOADER og DEBUGGER) i hukommelsen på forskellige tidspunkter findes der normalt fast et program der styrer disse transaktioner. Ligeledes er et stort antal funktioner fælles for mange anvendelser.
Computeren er derfor normalt forsynet med et MONITOR-program. Dette kaldes også for computerens OPERATING SYSTEM eller SUPER VISOR PROGRAM.
MONITOR-programmet kan enten være permanent i computeren i form af en ROM-hukommelse, eller blive loadet ind i denne fra et externt lagermedie, ved hjælp af en lille BOOTSTRAP-MONITOR. Da monitoren konstant er i computeren kan en bruger f. eks. bede om at hente ASSEMBLER-programmet ind i hukommelsen fra en disc, for at få oversat et SOURCE-program fra den samme eller en anden lagerenhed, og til slut anbringe de resulterende maskinkoder i det samme eller et andet lagermedie.
Så snart ASSEMBLEREN er loadet ind i computerens hukommelse starter monitoren straks eksekveringen. Når oversættelsen er færdig melder monitoren klar, hvorefter det næste program kan kaldes og køres o.s.v.

Under udviklingen af et brugerprogram kommer man i kontakt med følgende systemprogrammer:

1) Der tændes for udstyret (RESET).

2) Monitoren loades ind i hukommelsen fra disc.

3) Fra monitoren kaldes editoren, og brugerprogrammet indtastes i sourcekode. Når indtastningen er færdig afsluttes editeringen hvorved sourceprogrammet lagres på disc ,og kontrollen overtages igen af monitoren.

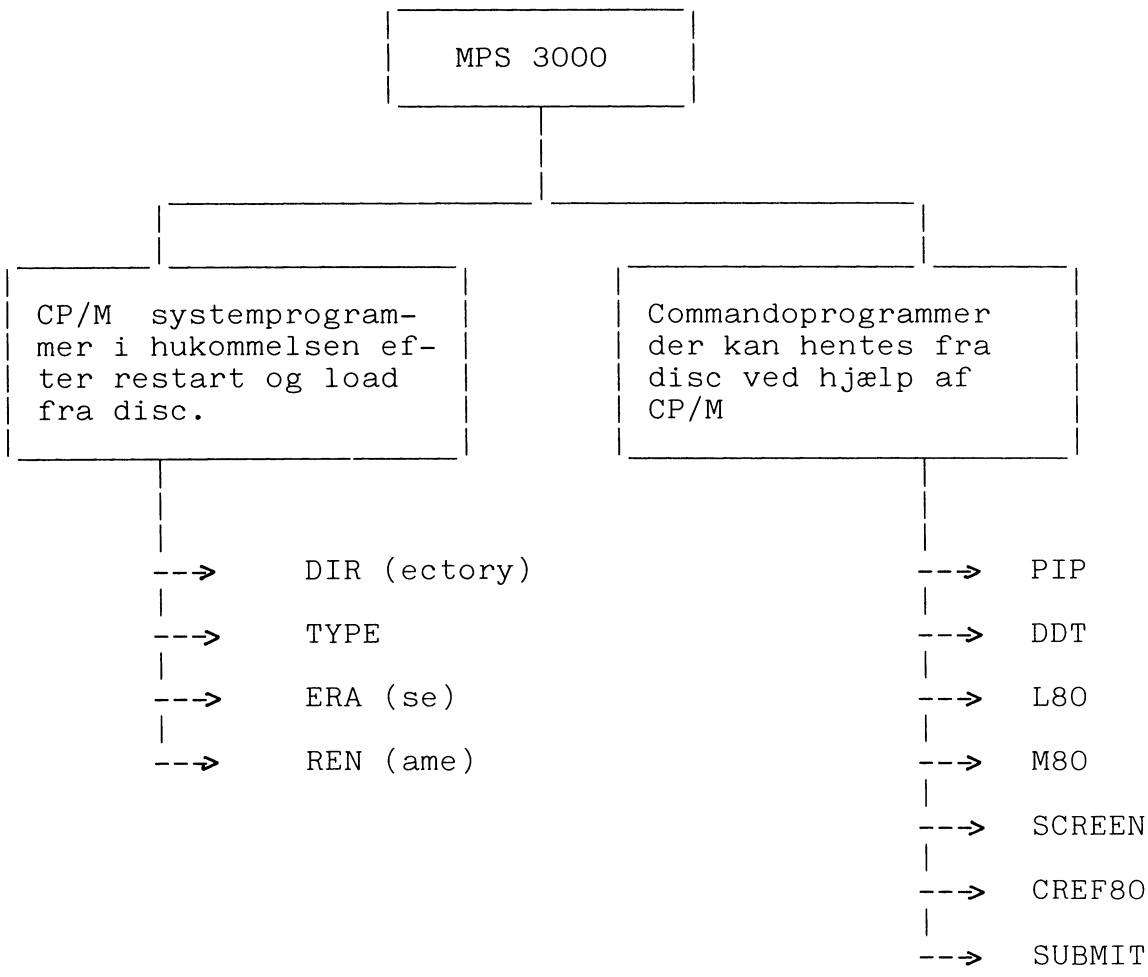
4) Fra monitoren kaldes assembleren. Den resulterende objectkode lagres på disc, og der gives forskellige fejlmeldinger, hvor efter kontrollen igen overtages af monitoren.

5) På grundlag af fejlmeldingerne kan editoren kaldes igen og fejlene rettes, hvorefter der foretages en ny assemblering. Således fortsættes til resultatet er fejlfrit.

6) Fra monitoren kaldes linker/loader-programmet, og de forskellige objectmoduler sammenkædes og plaseres i computerens hukommelse eller på disc. Herefter overtages kontrollen igen af monitoren.

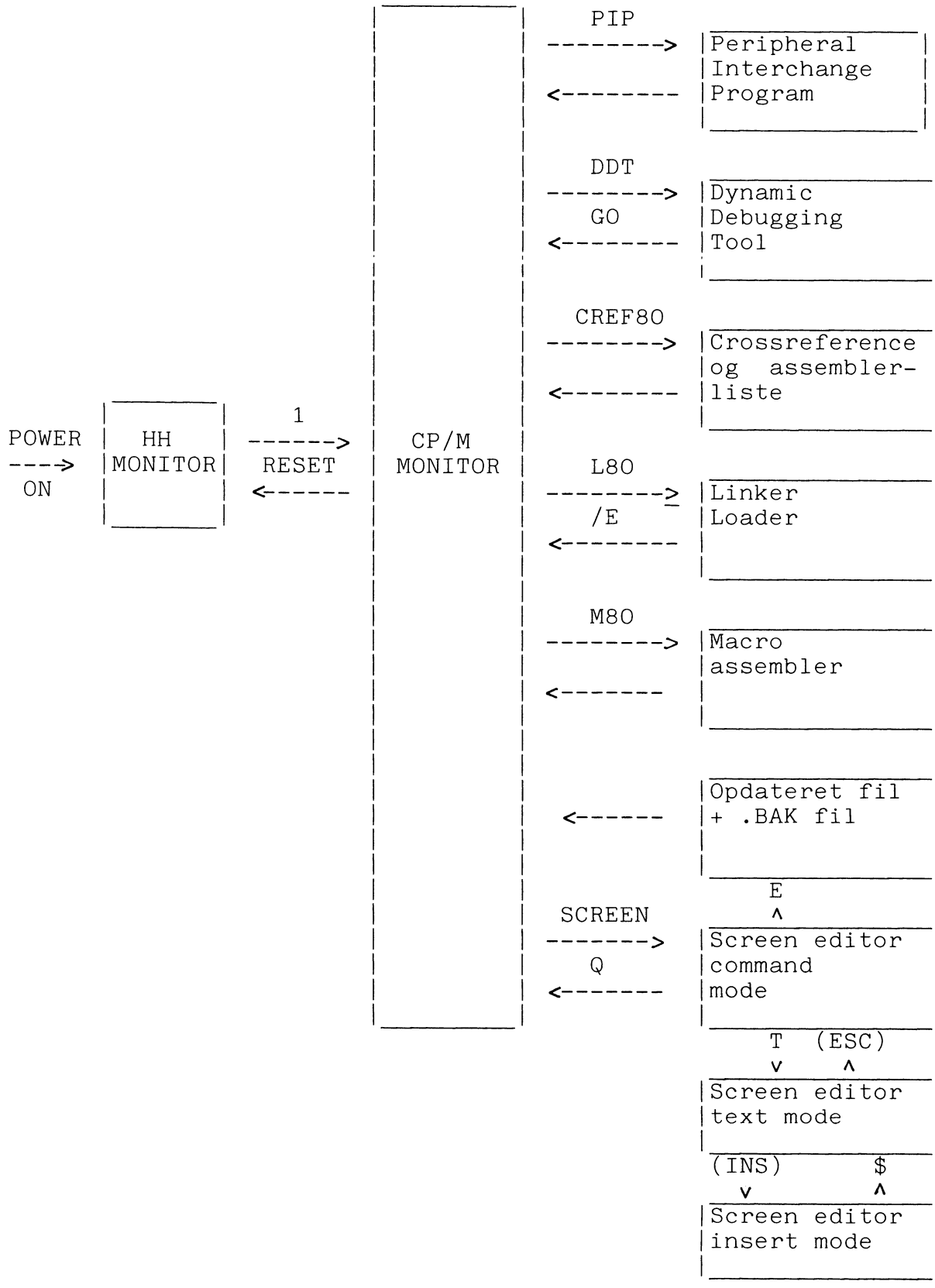
7) Til slut kaldes debugprogrammet via monitoren. Ved hjælp af dette prøvekøres det udviklede program i udviklingssystemet. Viser det sig at der er noget der skal ændres startes igen med editoren på sourcekodeniveau.

SYSTEMSOFTWARE I MPS 3000



PIP: Peripheral Interchange Program
DDT: Dynamic debuging Tool
L80: Linker 80
M80: Macroassembler 80
SCREEN: Screen editor
CREF80: Crosreference 80
SUBMIT: Styreprogram der anvendes ved samlet kørsel af f. eks. M80, CREF80 og L80.

OVERSIGT OVER KOMMANDOER I MPS 3000



$ = courser op, ned eller til højre

KORT OM CP/M

CP/M står for Control Program for Microprocessors, og er en systemsoftware der varetager kontrollen over disk-stationen samt printer og skærmterminal.
CP/M systemet findes på floppy-disk. Ved opstart loades CP/M over i computerens hukommelse således:

TÆND FOR COMPUTEREN, SKÆRMTERMINAL OG evt. PRINTER.

SÆT SYSTEMDISKETTEN I DRIVE A.

RESET.

MPS 3000 svarer så:

```
        HH-ROS V.1
1.      CP/M
2.      SERIEL PRINTER SELECT
3.      MONITOR

        SELECT: (tryk 1)
```

CP/M loades nu hvorefter MPS 3000 svarer:

```
HH-MPS-3000  60k CP/M  2.2

A>
```

Når MPS 3000 svarer med A> er CP/M systemet loadet og klar til at modtage kommandoer.
CP/M inkluderer blandt andre følgende funktioner:

(Det understregede indtastes af brugeren og afsluttes med retur)

DIR Directory — Giver oversigt over filnavne på den indsatte diskette.

```
A>DIR
  ---
```

TYPE type — Udlæser indholdet af en fil til skærmterminal og printer hvis denne er ON.

```
A>TYPE filnavn.ext
  ----------------
```

Printeren togles ON/OFF med (CTRL) P.
Udlistningen stoppes/startes med (CTRL) S

ERA Erase — Sletter filer (filnavne) på den indsatte diskette.

```
A>ERA filnavn.ext
  ---------------
```

REN rename Giver en fil et nyt navn.

```
A>REN filnavn.ext=filnavn.ext
  ------------------------
          |           |
      nyt navn    gl. navn
```

COMPUTERE MED TO DISCDRIVES

Hvis computeren har to drives skiftes mellem drive A og drive B således:

```
A>B:
  --
B>                Drive B er nu hoveddrive
```

FILNAVNE

Data overføres til og fra discketten under et filnavn.Filnavnet består af et navn på indtil otte charakterer samt en extension (udvidelse) på tre charakterer.Navnet vælges så det på bedste måde beskriver filens indhold, medens udvidelsen kan angive typen af data.

TYPISKE EXTENSIONS

.COM	command file	Objectkodefil der umiddeltbar kan eksekveres.
.MAC	Source fil	Input sourcefil til MACRO 80 assembleren.
.BAK	Bakup fil	Baggrundsfil der genereres af editoren.
.REL	Relokerbar fil	MACRO 80 relokerbar objectkodefil. Skal behandles af LINK80.
.CRF	Crossreferen-	Fil der indeholder crossreferencetabellen og listningen fra MACRO 80 assembleren.

Eksempel: SCREEN.COM

SCREEN = Tekst editeringsprogram, SCREEN EDITOR.

COM = Commandofil der udføres når den hentes ind i computeren fra disc.

OPKALD AF EN .COM-FIL

En .COM-fil kaldes blot med filnavnet:

```
A>filnavn
```

SCREEN EDITOR

Samtidig med at SCREEN editeringsprogrammet loades ind i computeren fra discen skal der gives et filnavn for den tekst der skal tastes ind når editoren er oprettet.
Ved afslutningen af editoren vil den redigerede tekst blive lagret på discen under det ved oprettelsen givne filnavn.

Eksempel på oprettelse af SCREEN-editorfil:

```
A>SCREEN TEST.MAC
         ---- ---
           |    |
           |    |_______ Extension, der fortæller at
           |            filen indeholder sourcekode
    Filens navn         til M80 macroassembleren.
```

Se videre om betjeningen af screen-editoren i afsnit 6

M80 MACRO-ASSEMBLER

Inputtet til M80 macroassembleren er en .MAC fil som indeholder sourcekoden.

Macro-assembleren kaldes således:

```
A>M80
  ---
                    M80 er nu loadet!
```

Opkaldet af M80 kan også kombineres med den efterfølgende kommandolinie.

Kommandolinier, eks 1:

```
A>M80 ,=filnavn
  -------------
```

Funktion: Fejlliste til skærmterminal og printer hvis denne er ON.

Kommandolinier, eks. 2:

```
A>M80 ,LST:=filnavn
  -----------------
```

Funktion: Fejlliste til skærmterminal. Assemblerliste med fejlkoder til printer.

Kommandolinier, eks. 3:

```
A>M80 ,=filnavn/C/R
  -----------------
```

Funktion: Relokerbar objectkodefil til disc (.REL). Crossreferencefil (.CRF) til disc.

Crossreferencefilen indeholder en tabel med de anvendte symboler (navne), samt de linienumre i assemblerlistende anvendes på. /C og /R er "switches" der tvinger assembleren til at levere de ønskede output.

Ved de første prøveassembleringer af et program kaldes M80 som i eks. 1 og 2. Når fejlene er rettet foretages den endelige assemblering som vist i eks. 3.

CREF80 CROSSREFERENCE

Hvis der ved assembleringen er genereret en crossreferencefil, .CRF, kan CREF80 ud af denne lave en crossreferenceliste. Denne kan enten lagres på discen som en .PRN fil, eller sendes direkte til printeren.
Som ved M80 kan opkaldet af CREF80 kombineres med resten af kommandolinien.

Kommandolinier eks. 1:

```
A>CREF80 LST:=filnavn
```

Funktion: Denne kommandolinie vil give en assemblerliste med linienumre, en symboltabel samt en crossreferenceliste ud til printeren.

Kommandolinier eks. 2:

```
A>CREF80 =filnavn
```

Funktion: Denne linie vil bevirke oprettelsen af en fil med navnet filnavn.PRN, med et indhold som beskrevet i eks. 1. Indholdet sendes til printer med TYPE.

L80 LINKER/LOADER

Inputtet til L80 er .REL filer fra assembleren. At en fil er relokerbar (.REL) vil sige at den er assembleret med en antaget startadresse på 0000H. Ved hjælp af L80 sammenkædes de ønskede .REL filer, samt flyttes til den adresse det endelige program skal ligge på. Outputtet fra L80 er en .COM fil der indeholder den endelige objectkode der kan køres i computeren.

L80 kommandolinien ser således ud:

```
A>L80 /P:adr,/D:adr,filnavn,filnavn/N,/E
      |     |       |       |     |
      1     2       3       4     5
```

1. Angiver startadressen på det endelige programs CSEG.
2. Angiver begyndelsesadressen på DSEG i programmet.
3. De/det filnavne der skal linkes, Flere navne adskilles med kommaer.
4. /N bevirker at de genererede objecthoder bliver sendt til discen under det givne navn som en.COM fil.
5. /E overgiver kontrollen til CP/M operativsystemet når L80 er færdig.

Hvis der ingen DSEG er i de .REL filer der skal behandles kan "2" udelades. Udelades også "1" vil programmet komme til at ligge fra adresse 0103H.

DDT, DYNAMIC DEBUGGING TOOL

Efter at sourcekoden er assembleret og linket skal programmet prøvekøres og fejlrettes (debugges). Dette gøres under kontrol af DDT !!

DDT opkaldes således:

```
A>DDT
  ---
-       (kommandoer kan nu indtastes)
```

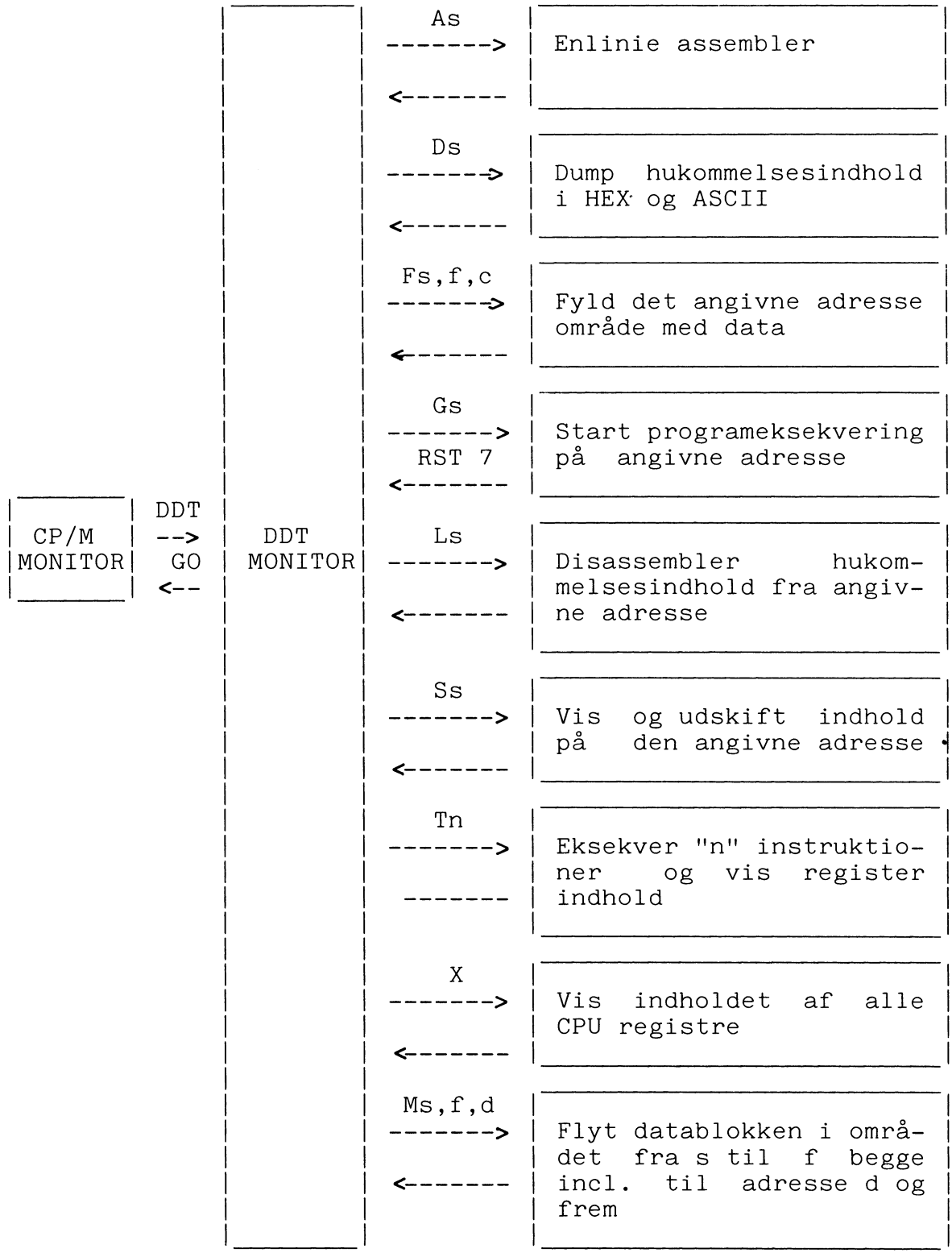
Findes det program der skal prøvekøres som en .COM fil på discen

opkaldes DDT på følgende måde:

```
A>DDT filnavn.COM
  ---------------
-       (kommandoer kan nu indtastes)
```

Ved hjælp af DDT kan programmet nu køres medens man følger og evt. ændrer registerindhold og hukommelsespladser.

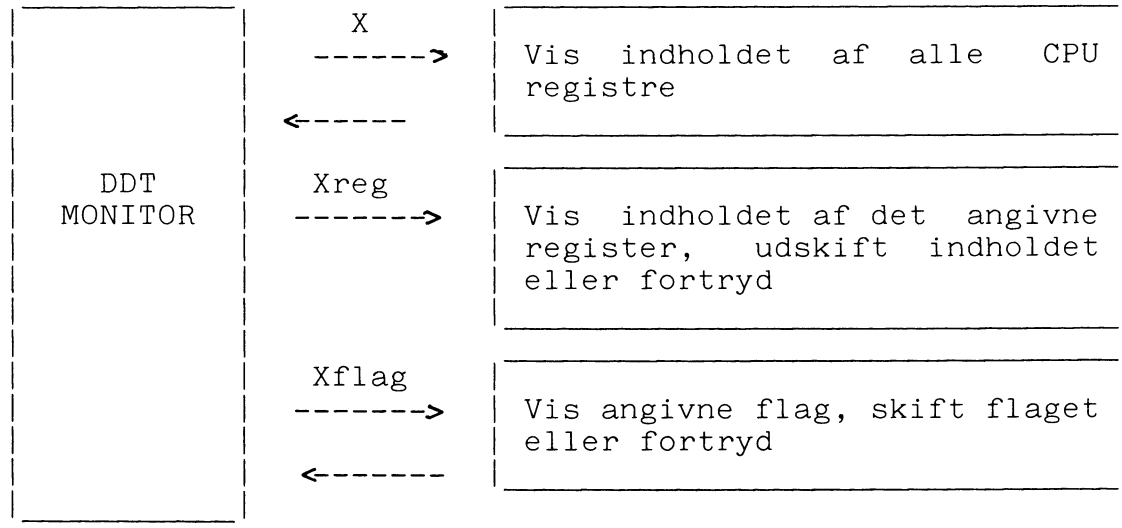
DDT KOMMANDOER



s = start adresse (start)
f = slut adresse (final)
d = mål adresse (destination)
c = data
n = antal

De indtastede kommandoer afsluttes med (return)

DDT, X, X reg og X flag kommandoer




```
reg: A    ACCUMULATOR
     B    BC REGISTERPAR
     D    DE   -  "  -
     H    HL   -  "  -
     S    STACKPOINTER
     P    PROGRAMCOUNTER


flag:C    CARRY FLAG
     Z    ZERO  FLAG
     M    MINUS FLAG
     E    EVEN PARITY FLAG
     I    INTERDIGIT CARRY FLAG
```

PIP, PERIPHERAL INTERCHANGE PROGRAM

PIP anvendes til at flytte, kopiere, kombinere og printe filer.

PIP kaldes således:

```
A>PIP
  ---
                    (kommandoer kan nu indtastes)
```

Eller PIP-opkaldet kombineres med resten af kommandolinien.

PIP kommandolinie for kopiering af fil:

```
A>PIP drive:filnavn.ext=drive:filnavn.ext
      ----- ---------- -----. ----------
        1        2        3        4
```

1: Drive hvorpå kopiet skal anbringes, A eller B
2: Filnavn og extension til kopiet
3: Drive hvorpå den fil der skal kopieres findes, A eller B
4: Filnavn og extension på den fil der skal kopieres

DIVERSE PIP KOMMANDOER

Kopiering af fil:

```
A>PIP A:NYFIL.MAC=B:GLFIL.BAK
  ---------------------------
```

Funktion: Fra drive B kopieres GLFIL.BAK over til drive A under navnet NYFIL.MAC

Sammenkædning af tekstfiler (ASCII-filer):

```
A>PIP A:STOR.MAC=A:EN.TEX,TO.TEX,TRE.TEX
```

Funktion: Filerne EN.TEX, TO.TEX og TRE.TEX kopieres og samles til en ny fil med navnet STOR .MAC, i rækkefølgen EN.TEX, TO,TEX, TRE.TEX

Kopiering af en del af en fil:

Ved at anvende S(tart) og Q(uit) parametrene og specificere to tekststrenge, er det muligt at kopiere en del af en fil. Anvendes S eller Q hver for sig, kopieres kun fra start- henholdsvis til quittekststrengen, begge incl.
Findes der små bogstaver i de specificerede tekststrenge skal PIP opkaldes separat, hvis dette ikke er tilfældet kan PIP kombineres med kommandolinien.

```
A>PIP A:DEL.TEX=A:HEL.TEXÆSBEGYND^Z QSLUT^ZÅ
                        12       3 4     56
```

1: Firkantet venstreparentes, stort Æ !!
2: Start parameter, S
3: ^Z betyder "control Z"
4: Quit parameter, Q
5: ^Z
6: Firkantet højreparentes, stort Å !!

Funktion: Fra filen HEL.TEX kopieres fra og med ordet BEGYND til og med ordet SLUT

Udskrift på linieprinter

Ved hjælp af kommandoen LST og parametrene N (numbers), T (tabulator) og P (page) er det muligt at få en formateret udskrift på printer.

```
A>PIP LST:=A:FIN.TEXÆNT8P50Å
  ------------------||||||||
                    12345678
```

1:	Firkantet venstreparentes
2:	N parameter, giver print med linienummer
3,4:	T parameter, tabulator sat til 8
5,6,7:	P parameter, Sat til 50 linier pr. side
8:	Firkantet højreparentes

Funktion: Denne kommandolinie vil bevirke at filen FIN.TEX printes med linienummer, horisontal tabulator på 8 positioner og 50 linier pr. side.

Udlistningen kan stoppes og startes ved hjælp af ^S (control S).

SUBMIT

Ved hjælp af SUBMIT er det muligt at sammenkæde CP/M kommandoer til automatisk afvikling.

Med editoren oprettes en fil med .SUB som extension. Denne .SUB fil indeholder CP/M kommandoer, men i stedet for at specificere navnene på de filer de enkelte kommandoer skal arbejde på, indsættes dollartegn med fortløbende numre ($1 $2 ..... $n).

På det tidspunkt hvor SUBMIT eksekveres oplyses om de aktuelle navne som så automatisk indsættes på dollartegnenes pladser. Det er således muligt at have et .SUB program liggende på disc for en komplet assemblering med M80, CREF80 og L80.

Eksempel:

Via editoren oprettes en .sub fil med navnet ASMBL.SUB og med følgende indhold:

```
M80 ,=$1/C/R
CREF80 LST:=$1
L80 /P:$2,/D:$3,$1,$4/N,/E
```

Ved opkald af SUBMIT skal kommandolinien indeholde de faktiske filnavne og talstørrelser som erstatning for dollartegnene i .SUB filen

Kommandolinien ser ser f. eks. således ud:

```
A SUBMIT ASMBL DEMO 100 0 MODE
  --------------|----|--|--|--
                $1   $2 $3 $4
```

Funktion: I stedet for $1 indsættes filnavnet DEMO,
i stedet for $2 indsættes CSEG start 100H,
i stedet for $3 indsættes DSEG start 0H
i stedet for $4 indsættes filnavnet MODE,
hvorefter M80, CREF80 og CREF80 automatisk
eksekveres i den angivne rækkefølge.

Eksempel på assemblering og afprøvning af et program.

```
TYPE DELAY.MAC
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
        ;               ---- DELAY ----                 ;
        ; Dette program indskyder et delay efter form-  ;
        ; len: Tdelay = ((E) x 21)+50 clock cycles      ;
        ;                                               ;
        ; Programmet kaldes som en subrutine            ;
        ;                                               ;
        ; INPUT         : (E)                           ;
        ; OUTPUT        : ingen                         ;
        ; DESTRUERER    : (E)                           ;
        ;                                               ;
        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


        CSEG

;Red A og flag på stacken!
DELAY:  PUSH PSW
        MOV     A,E
        ORA     E

;Er vi færdige?
LOOP:   JZ      FAERDIG

;Juster (E) en ned
        DCR     E
        JMP     LOOP

;Hent A og flag fra stacken!
FAERDIG:POP     PSW
        RET

        END

A>
```

Sourcekoden findes på disk under filenavnet DELAY.MAC

```
  DIR
A: PIP      COM : DDT      COM : STAT     COM : DUMP     COM
A: LIB      COM : M80      COM : L80      COM : CREF80   COM
A: ASMBL    BAK : XSUB     COM : MOVCPM   COM : FORMAT   COM
A: DEMOFEJL MAC : ASMBL    SUB : DEMO     MAC : SCREEN   BAK
A: DEMO     BAK : DEMO     CRF : SOFTKURS LST : DEMOFEJL BAK
A: DEMO     REL : TEX      $$$ : SCREEN   COM : FORLIB   REL
A: SUBTUT   COM : SCREEN   SUB : DELAY    BAK : DELAY    MAC
A: SUBMIT   COM
A>
```

Sourcekoden! (DELAY MAC)

.BAK-file, automatisk oprettet pga rettelser i .MAC-filen (DELAY BAK)

```
M80 ,=DELAY.MAC
No  Fatal error(s)
```

Prøveassemblering.
Der genereres kun
fejlmeldinger.

```
M80 ,=DELAY.MAC/C/R
No  Fatal error(s)
```

Endelig assemblering.

Der genereres crossreferencefile - /C,
og relokerbar objectkodefile - /R.

```
DIR
A: PIP      COM : DDT      COM : STAT     COM : DUMP     COM
A: LIB      COM : M80      COM : L80      COM : CREF80   COM
A: ASMBL    BAK : XSUB     COM : MOVCPM   COM : FORMAT   COM
A: DEMOFEJL MAC : ASMBL    SUB : DEMO     MAC : SCREEN   BAK
A: DEMO     BAK : DEMO     CRF : SOFTKURS LST : DEMOFEJL BAK
A: DEMO     REL : TEX      $$$ : SCREEN   COM : FORLIB   REL
A: SUBTUT   COM : SCREEN   SUB : DELAY    BAK : DELAY    MAC
A: DELAY    CRF : DELAY    REL : SUBMIT   COM
A>
```

Relokerbar objectkode-file

Crossreferencefile

```
A>CREF80 LST:=DELAY
```

Generering og printning af assemblerliste og crossreferenceliste.

Assemblerliste.

```
MACRO-80 3.36   17-Mar-80       PAGE    1

  1                                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  2                                    ;              ---- DELAY ----                 ;
  3                                    ; Dette program indskyder et delay efter form- ;
  4                                    ; len: Tdelay = ((E) x 21)+50 clock cycles     ;
  5                                    ;                                              ;
  6                                    ; Programmet kaldes som en subrutine           ;
  7                                    ;                                              ;
  8                                    ; INPUT         : (E)                          ;
  9                                    ; OUTPUT        : ingen                        ;
 10                                    ; DESTRUERER    : (E)                          ;
 11                                    ;                                              ;
 12                                    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
 13
 14
 15
 16   0000'                                    CSEG
 17
 18                                    ;Red A og flag på stacken!
 19   0000'   F5                       DELAY:  PUSH PSW
 20   0001'   7B                               MOV     A,E
 21   0002'   B3                               ORA     E
 22
 23                                    ;Er vi færdige?
 24   0003'   CA 000A'                 LOOP:   JZ      FAERDIG
 25
 26                                    ;Juster (E) en ned
 27   0006'   1D                               DCR     E
 28   0007'   C3 0003'                         JMP     LOOP
 29
 30                                    ;Hent A og flag fra stacken!
 31   000A'   F1                       FAERDIG:POP     PSW
 32   000B'   C9                               RET
 33
 34                                            END
```

Objectkoden — *Sourcekoden*

Adresse relativ til ØØØØH

Linienummer

```
MACRO-80 3.36  17-Mar-80      PAGE   S
```

Symboltabel

```
Macros:

Symbols:
DELAY   0000'   FAERDI  000A'   LOOP    0003'
No  Fatal error(s)
```

Værdi (rel. adr.)

Symbol

Crossreferenceliste

```
DELAY     19#
FAERDI    24      31#
LOOP      24#     28
```

Symbolet anvendes på linie 28 i assemblerlisten.

Symbolet defineres på linie 24 i assemblerlisten

Symbol

```
L80 /P:100,DELAY,DELAY/N,/E
Link-80  3.37  08-May-80  Copyright 1979,80 (C) Microsoft

Data    0100    010C

0000    010C    1A
```

Generering af .COM-file af .REL-filen.

Programmets start-adr.

Næste ledige adr.

Programmet fylder 1 blok (256 bytes)

```
DIR
A: PIP      COM : DDT      COM : STAT     COM : DUMP     COM
A: LIB      COM : M80      COM : L80      COM : CREF80   COM
A: ASMBL    BAK : XSUB     COM : MOVCPM   COM : FORMAT   COM
A: DEMOFEJL MAC : ASMBL    SUB : DEMO     MAC : SCREEN   BAK
A: DEMO     BAK : DEMO     CRF : SOFTKURS LST : DEMOFEJL BAK
A: DEMO     REL : TEX      $$$ : SCREEN   COM : FORLIB   REL
A: SUBTUT   COM : SCREEN   SUB : DELAY    BAK : DELAY    MAC
A: DELAY    CRF : DELAY    REL : DELAY    COM : SUBMIT   COM
A>
```

COM-filen indeholder objectkoden

Programmet afprøves nu med DDT.

```
A>DDT DELAY.COM
DDT VERS 2.2
NEXT  PC
0180 0100
-
```

DELAY.COM: Navnet på det program der skal børes.

0100: Programcounterens værdi

0180: Næste fri adr. efter det loadede program.

Disassemblering af det loadede program fra adr. 100H til 10BH.

```
-L100,10B
0100  PUSH PSW
0101  MOV  A,E
0102  ORA  E
0103  JZ   010A
0106  DCR  E
0107  JMP  0103
010A  POP  PSW
010B  RET
010C
```

Vis CPU-registre

```
-X
COZOMOEOIO A=00 B=0000 D=0000 H=0000 S=0100 P=0100 PUSH PSW
-XA
A=00 FF
-X
COZOMOEOIO A=FF B=0000 D=0000 H=0000 S=0100 P=0100 PUSH PSW
-T5
COZOMOEOIO A=FF B=0000 D=0000 H=0000 S=0100 P=0100 PUSH PSW
COZOMOEOIO A=FF B=0000 D=0000 H=0000 S=00FE P=0101 MOV  A,E
COZOMOEOIO A=00 B=0000 D=0000 H=0000 S=00FE P=0102 ORA  E
COZIMOE1IO A=00 B=0000 D=0000 H=0000 S=00FE P=0103 JZ   010A
COZ1MOE1IO A=00 B=0000 D=0000 H=0000 S=00FE P=010A POP  PSW  $010B
-
```

Programcounterværdi med tilhørende instruktion.

E-reg

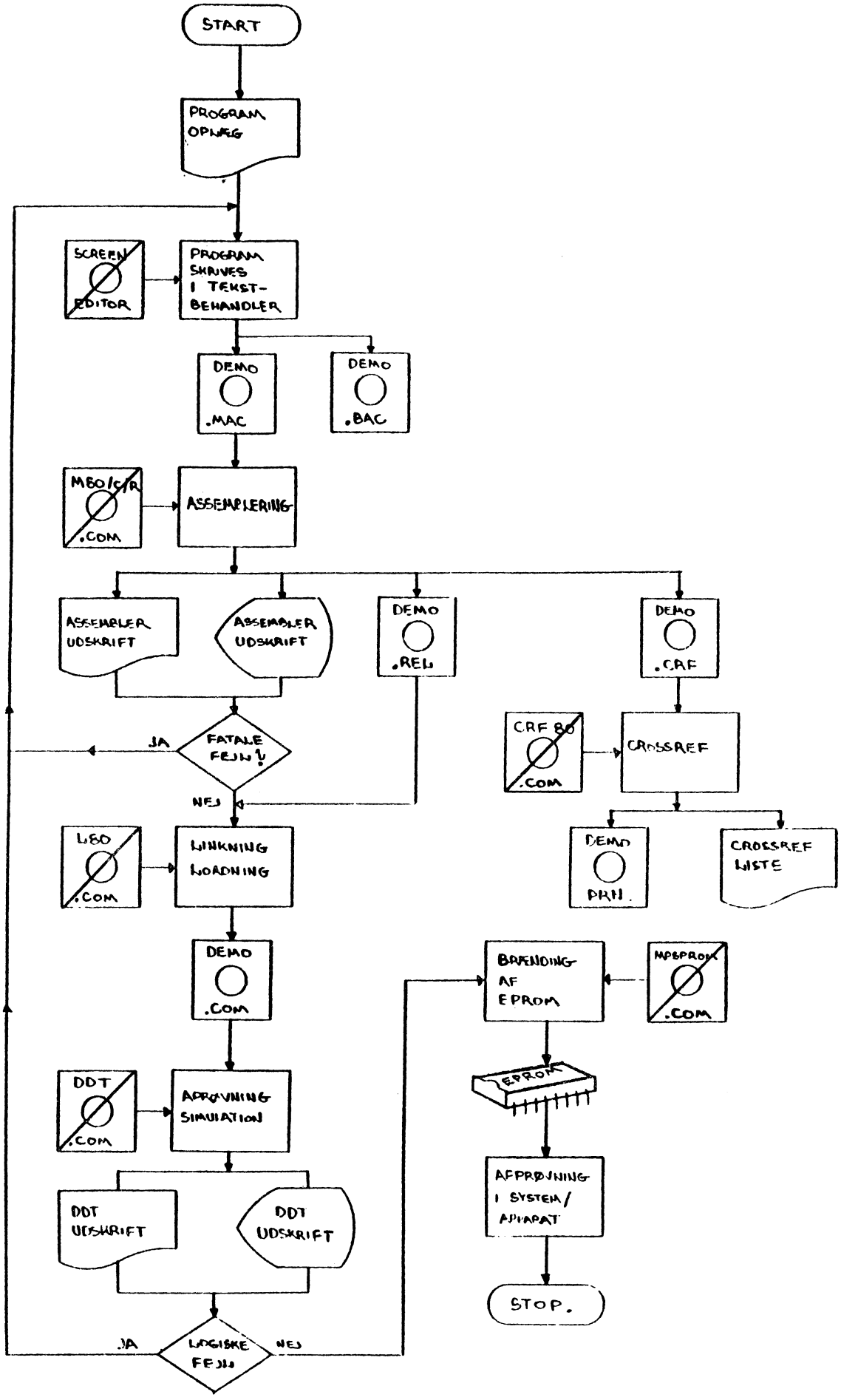
Prøvekørsel af 5 linier med ØØH i E-reg.

OK

```
-X
COZOMOEOIO A=FF B=0000 D=0000 H=0000 S=0100 P=010B RET
-XD
D=0000 0001
-XP
P=010B 100
-X
COZOMOEOIO A=FF B=0000 D=0001 H=0000 S=0100 P=0100 PUSH PSW
-T8
COZOMOEOIO A=FF B=0000 D=0001 H=0000 S=0100 P=0100 PUSH PSW
COZOMOEOIO A=FF B=0000 D=0001 H=0000 S=00FE P=0101 MOV  A,E
COZOMOEOIO A=01 B=0000 D=0001 H=0000 S=00FE P=0102 ORA  E
COZOMOEOIO A=01 B=0000 D=0001 H=0000 S=00FE P=0103 JZ   010A
COZOMOEOIO A=01 B=0000 D=0001 H=0000 S=00FE P=0106 DCR  E
COZIMOEOIO A=01 B=0000 D=0000 H=0000 S=00FE P=0107 JMP  0103
COZ1MOEOIO A=01 B=0000 D=0000 H=0000 S=00FE P=0103 JZ   010A
COZ1MOEOI0 A=01 B=0000 D=0000 H=0000 S=00FE P=010A POP  PSW  $010B
-
```

Prøvekørsel af 8 linier med Ø1H i E-reg.

OK



PSEUDO INSTRUKTIONER.

Under programskrivningen anvendes i stor udstrækning en række direktiver som har til formål at styre assembleren, samt lette arbejdet ved udviklingen. Disse direktiver kaldes under et PSEUDO-instruktioner eller PSEUDO-direktiver.

Direktiverne kan groft opdeles således:

GENERELLE direktiver:

anvendelse	form	bemærkninger
Symboldefinitioner:	EQU	Equal, anvendes som "lighedstegn" mellem et symbol og dets værdi.
	SET	Set, ligner EQU, men tillader at et symbol tildeles nye værdier gennem programmet.
Datadefinitioner:	DB	Define byte, anvendes hvor der skal indsættes en eller flere bytes i rækkefølge i programmet, f.eks. en tabel.
	DW	Define word, anvendes hvor der skal indsættes en eller flere 2-byte størrelser, f.eks adresser, i programmet idet hver 2-byte størrelse lagres i lowbyte, higt-byte orden.
Memory reservation	DS	Define space, anvendes til at reservere et antal hukommelsespladser i forhold til data segmentet (DSEG).

Betinget assemblering	IF
	ELSE
	ENDIF
		Ved at anvende IF, ELSE og ENDIF direktiverne i et program kan dele af dette udelades eller ombyttes med andre alt efter om betingelsen i operandfeltet efter IF direktivet er opfyldt eller ej.
		Ethvert IF-direktiv skal afsluttes med et tilhørende ENDIF. ELSE-direktivet kan anvendes som en option i en IF - ENDIF blok.

Programafslutning	END	Sourcekoden afsluttes med END som information til assembleren om at her slutter den kode der skal assembleres.

LOKATIONS-TÆLLER kontrol og relokering:

Lokation tæller	ASEG	Absolut segment. Ved hjælp af ASEG-direktivet samt en efter følgende ORG informeres assembleren om fra hvilke absolut adresse assembleringen skal starte. En sourcekode der omfattes af et ASEG-direktiv kan ikke senere relokeres til at ligge på en anden adresse.
	CSEG	Code segment. Et CSEG-direktiv efterfulgt af en ORG har samme effekt som ovenstående ASEG. Gives CSEG-direktivet alene vil assembleren generere den relokerbare kode relativt til ØØØØH. Ved at bruge P-switchen i L80 kan den relokerbare kode flyttes til enhver adresse.

	DSEG	Data segment. DSEG-direktivet ændrer assemblerens lokationscounter til at følge datasegmentet af hukommelsen. Datasegmentets start kan enten gives med en efterfølgende ORG, eller ved at bruge D-switch i LINK-O. DSEG-direktivet anvendes typisk i forbindelse med reservation af pladser i i RAM-hukommelsen.
	ORG	Origin-direktivet anvendes i forbindelse med ovenstående ASEG-, CSEG- og DSEG-direktiver, eller det kan anvendes selvstændigt, og knytter sig da automatisk til det segment (A,C el. D) som assembleren i øjeblikket arbejder i. I ORG direktivets operandfelt gives værdien som lokationscounteren skal antage. Enten som et symbol der allerede er defineret, eller som absolut værdi.

PROGRAM LINKING:		Ved sammenlinkningen af flere programmoduler er der tit brug for at eet modul kan anvende anvende symboler der er defineret i et andet modul, og dette andet modul skal "tillade" at dets symboler anvendes af andre moduler.
Defineret i et andet program (modul)	EXTRN	EXTRN-direktivet anvendes foran en navneliste der indeholder de symboler der er defineret i andre moduler. De vil derfor først få deres endelige værdier ved linkningen med L80.
Tilgængelige for andre moduler	PUBLIC	PUBLIC-direktivet anvendes foran en navneliste der indeholder de symboler der må anvendes af andre programmoduler. Hvis der er flere navne (symboler) i navnelisten adskilles de med et komma.

```
MACRO-80 3.36   17-Mar-80       PAGE    1


                                .COMMENT ^

                                        Det følgende program har til formål at vise brugen og virkningen
                                        af de nævnte pseudoinstruktioner, idet programmet i sig selv er
                                        uvæsentligt.


                                                PUBLIC  ADR             ;ADR erklæres public
                                                EXTRN   INPUT           ;INPUT erklæres extern

1000                            HOP     EQU     1000H           ;HOP tildeles værdien 1000H

0000'                                   ASEG                    ;Lokationstælleren sættes til det
                                        ORG     100H            ;absolutte segment, og startes på
                                                                ;værdien 100H.
0100    00                              NOP
0101                                    CSEG                    ;Lokationstælleren kobles på code-
                                                                ;segmentet.
0000'   00                              NOP
0001'   00                              NOP

0002'   F7 43 00 41            TABEL:  DB      0F7H,43H,0,'A'  ;Tre talværdier samt en ASCII-karak
                                                                ;ter indlægges i programmet.

0006'   1234 0094               ADR:   DW      1234H,94H,0F3C8H,HOP    ;Fire 2-byte størrelser ind-
000A'   F3C8 1000
                                                                ;lægges i programmet, heraf den ene
                                                                ;som et defineret symbol.

000E'                                   DSEG                    ;Lokationstælleren kobles på dataseg-
                                        ORG     2000H           ;mentet og starter på 2000H.
2000"                           SUM:    DS      4               ;4 pladser reserveres SUM
2004"                           DIF:    DS      3               ;3 pladser reserveres DIF
2007"                           DAGE:   DS      8               ;8 pladser reserveres DAGE

200F"                                   CSEG                    ;Lokationstælleren kobles tilbage til
                                                                ;codesegmentet.

0001                            OUTPUT  SET     01              ;OUTPUT får værdien 1
000E'   3A 0000#                        LDA     INPUT
0011'   32 0001                        STA     OUTPUT

0002                            OUTPUT  SET     02              ;OUTPUT får en ny værdi. 2
0014'   32 0002                         STA     OUTPUT
```

```
MACRO-80 3.36   17-Mar-80       PAGE    1-1


47     0001                             NY      SET     1               ;NY får værdien 1, sand!
48
49                                      IF      NY ;-------------er sand, det er den!
50     0017'   00                       NOP
51                                      ELSE
52                                      NOP
53                                      NOP
54                                      ENDIF
55
56     0000                             NY      SET     0               ;NY får værdien 0, falsk!
57
58                                      IF      NY ;-------------er sand, det er den ikke, så ELSE
59                                      NOP
60                                      ELSE
61     0018'   00                       NOP
62     0019'   00                       NOP
63                                      ENDIF
64
65                              .COMMENT ^
66                                      I assemblerudskriften findes forskellige tegn som sættes på adresserne
67                                      for at angive hvilken lokationstæller der har været aktiv. Tegnene er:
68
69                                              "blank" = ASEG
70                                                 '    = CSEG
71                                                 "    = DSEG
72
73                                      PUBLIC og EXTRN har ogsa deres tegn:
74
75                                                 *    = Symbolet er externt defineret
76                                                 I    = Symbolet er erklæret public (Internationalt)!
77
78     001A'                            SLUT:   END             ;Her slutter programmet.
```

```
        MACRO-80 3.36   17-Mar-80       PAGE    5


Macros:

Symbols:
ADR     0006I'  DAGE    2007"   DIF     2004"   HOP     1000
INPUT   000F$   NY      0000    OUTPUT  0002    SLUT    001A'
SUM     2000"   TABEL   0002'


No  Fatal error(s)
```

ADR	8	25#		
DAGE	34#			
DIF	33#			
HOP	11#	26		
INPUT	9#	40		
NY	47#	49	56#	58
OUTPUT	39#	41	43#	44
SLUT	78#			
SUM	32#			
TABEL	22#			

MACROER

En MACRO er i grunden en facilitet til udskiftning af et sæt parametre med et andet. Ved udvikling af et program, viser det sig ofte at mange instruktionssekvenser gentages igennem programmet, kun med andre parametre.

Tag som eksempel en rutine der flytter 5 bytes fra en memorylokation til en anden. Lidt senere skal der bruges en rutine der flytter 4 bytes blot mellem andre steder end den foregående.
Instruktionerne i de to programstykker er de samme, de afviger kun ved adresserne mellem hvilke der skal flyttes data, samt antallet af byte der skal flyttes.

Hvis det kunne lade sig gøre automatisk at få gentaget de grundlæggende instruktioner i de to omtalte dataflytning, blot med forskellige parametre ville det spare tid ved udviklingen, samt nedsætte muligheden for fejlindtastninger. MACRO-faciliteten giver netop denne og andre muligheder for gentagelse af instruktionssekvenser.

En MACRO kan beskrives som en formel sekvens af instruktioner, som når den kaldes op i et program, udskifter opkaldet med de i MACROEN definerede instruktioner, samt de i opkaldet angivne parametre.

MACROER contra subrutiner.

Forskellen mellem en MACRO og en subrutine, er at der skal hoppes til en anden del af programmet for at eksekvere subrutinen, medens macroen giver instruktioner der ligger fortløbende i programmet.
Således indeholder et program kun en version af en given subrutine, men indeholder lige så mange versioner af en given macro som den er kaldt. Bemærk ordet 'versioner', idet det er den grundlæggende forskel på en subrutine og en MACRO. En MACRO ge nererer ikke nødvendigvis den samme sourcekode hver gang den bliver kaldt.
Ved at ændre parametrene i macroopkaldet, kan man ændre den sourcekode der bliver genereret.

MACRO-direktiver.

MACRO direktivet
ENDM - " -
LOCAL - " -

Antag at der mange steder i et program er brug for at flytte data ind i A, B og C registrene. Til denne opgave kunne der skrives følgende MACRO.

```
;label  opcode  operand                    kommentater
;-----------------------------------------------------------------
        ASEG
        ORG     100H

MAC1    MACRO   X1,X2,X3    ;;Macroen opkaldes senere med nav-
                            ;;net MAC1. Den indeholder parame-
                            ;;trene X1, X2 OG X3 som skal defi-
                            ;;neres i opkaldet.
        LOCAL   FLYT        ;;Labelen FLYT er LOKAL og kun gæl-
                            ;;dende inden for macroen.
FLYT:   LHLD    X1          ;;Her følger det program der skal
        MOV     A,M         ;;indsættes hver gang der sker et
        LHLD    X2          ;;opkald af macroen.
        MOV     B,M
        LHLD    X3
        MOV     C,M

ENDM                        ;;Her slutter macroen.



        NOP             ;Dette er en del af "hovedprogrammet"
        NOP             ;hvori der findes opkald af macroen.
        NOP
        MAC1    200H,201H,202H ;Her kaldes macroen første
                               ;gang,og de værdier der skal
                               ;indsættes i stedet for X1, X2
                               ;og X3 defineres.
        NOP
        NOP
        MAC1    300H,301H,303H ;Her kaldes macroen anden gang,
                               ;og et nyt sæt værdier define-
                               ;res.
        NOP
        END                    ;Her slutter programmet.
```

```
MACRO-80 3.36   17-Mar-80       PAGE    1-1


                                ;MACRO:
                                ;
                                ;label   opcode  operand                kommentater
                                ;-------------------------------------------------------------
0000'                                   ASEG
                                        ORG     100H

                                MAC1    MACRO   X1,X2,X3       ;;Macroen opkaldes senere med nav-
                                                               ;;net MAC1. Den indeholder parame-
                                                               ;;trene X1, X2 OG X3 som skal defi-
                                                               ;;neres i opkaldet.
                                        LOCAL   FLYT           ;;Labelen FLYT er LOKAL og kun gæl-
                                                               ;;dende inden for macroen.
                                FLYT:   LHLD    X1             ;;Her følger det program der skal
                                        MOV     A,M            ;;indsættes hver gang der sker et
                                        LHLD    X2             ;;opkald af macroen.
                                        MOV     B,M
                                        LHLD    X3
                                        MOV     C,M

                                        ENDM                   ;;Her slutter macroen.


0100    00                              NOP                    ;Dette er en del af "hovedprogrammet"
0101    00                              NOP                    ;hvori der findes opkald af macroen.
0102    00                              NOP
                                        MAC1    200H,201H,202H ;Her kaldes macroen første gang, og
0103    2A 0200         +       ..0000: LHLD    200H
0106    7E              +               MOV     A,M
0107    2A 0201         +               LHLD    201H
010A    46              +               MOV     B,M
010B    2A 0202         +               LHLD    202H
010E    4E              +               MOV     C,M
                                                               ;de værdier der skal indsættes i ste-
                                                               ;det for X1, X2 og X3 defineres.
010F    00                              NOP
0110    00                              NOP
                                        MAC1    300H,301H,303H ;Her kaldes macroen anden gang, og et
0111    2A 0300         +       ..0001: LHLD    300H
0114    7E              +               MOV     A,M
0115    2A 0301         +               LHLD    301H
0118    46              +               MOV     B,M
0119    2A 0303         +               LHLD    303H
011C    4E              +               MOV     C,M
                                                               ;nyt sæt værdier defineres.
011D    00                              NOP


                                        END
```

```
        MACRO-80 3.36   17-Mar-80       PAGE    5


Macros:
MAC1

Symbols:
..0000  0103    ..0001  0111


No  Fatal error(s)









..0000      33#
..0001      44#
MAC1        11#     32      43
```

MACRO-direktiver.

REPT direktivet

Antag at der er brug for at gentage den samme instruktion et vist antal gange efter hinanden, f.eks. at rulle accumulatorens indhold fire pladser mod venstre. Følgende MACRO med REPT-direktiv vil indsætte det ønskede antal RLC instruktioner.

```
;LEFT4
;label      opcode  operand                    kommentarer
;-------------------------------------------------------------
            ASEG
            ORG     100H

            NOP
LEFT4:      REPT    4          ;;Her starter macroen.
            RLC                ;;Instruktion der skal gentages
            ENDM               ;;Her slutter macroen.
            NOP
            END
```

```
MACRO-80 3.36 17-Mar-80 PAGE 1

  1                             ;LEFT4
  2                                   ;label opcode operand        kommentarer
  3                                    ;-----------------------------------------------
  4  0000'                          ASEG
  5                                 ORG 100H
  6
  7  0100   00                     NOP
  8  0101                              LEFT4: REPT 4             ;;Her starter macroen.
  9                                 RLC         ;;Instruktion der skal gentages
 10                                 ENDM               ;;Her slutter macroen.
 11  0101   07             +       RLC         ;;Instruktion der skal gentages
 12  0102   07             +       RLC         ;;Instruktion der skal gentages
 13  0103   07             +       RLC         ;;Instruktion der skal gentages
 14  0104   07             +       RLC         ;;Instruktion der skal gentages
 15  0105   00                     NOP
 16                                 END
```

MACRO-direktiver

IRP direktivet

Hvis den samme programsekvens skal gentages et antal gange med et nyt sæt data for hvert gennemløb, kan IRP-direktivet anvendes. Antag f.eks. at tre på hinanden følgende memorypladser skal initialiseres med indholdet, 5AH, 28H OG C7H. Sourcekoden kan da have følgende udseende:

```
;INIT
;label   opcode   operand                    kommentarer
;-----------------------------------------------------------
         ASEG
         ORG      100OH

         NOP
INIT:    LXI      H,2OOOH
         IRP      X,(5AH,28H,OC7H);;Her gives IRP direktivet samt
         MVI      A,X              ;;de data der skal indsættes
         MOV      M,A              ;;ved de tre gennemløb, idet
         INX      H                ;;der repeteres lige så mange
         ENDM                      ;;gange som der er parametre i
         NOP                       ;;()-parantesen.
         END
```

```
MACRO-80 3.36   17-Mar-80      PAGE   1


 1                              ;INIT
 2                              ;label  opcode  oprand         kommentarer
 3                              ;-----------------------------------------------------
 4    0000'                             ASEG
 5                                      ORG     100H
 6    0100    00                        NOP
 7    0101    21 2000           INIT:   LXI     H,2000H
 8                                      IRP     X,(5AH,28H,0C7H);;Her gives IRP direktivet samt
 9                                      MVI     A,X             ;;de data der skal indsættes
10                                      MOV     M,A             ;;ved de tre gennemløb, idet
11                                      INX     H               ;;der repeteres lige så mange
12                                      ENDM                    ;;gange som der er parametre i
13    0104    3E 5A             +       MVI     A,5AH
14    0106    77                +       MOV     M,A
15    0107    23                +       INX     H
16    0108    3E 28             +       MVI     A,28H
17    010A    77                +       MOV     M,A
18    010B    23                +       INX     H
19    010C    3E C7             +       MVI     A,0C7H
20    010E    77                +       MOV     M,A
21    010F    23                +       INX     H
22    0110    00                        NOP                     ;;()-paratesen.
23                                      END
```

MACRO-direktiver

IRPC direktivet

IRPC direktivet ligner IRP direktivet, med den forskel at IRPC repeteres lige så mange gange som der er karakterer (tal mellem og 9) i en specificeret streng.
Skal der f.eks. sendes en serie tal til fortløbende memorylokationer kan IRPC direktivet anvendes således:

```
;UD
;label    opcode     operand                    kommentarer
;-------------------------------------------------------------------
          ASEG
          ORG        100H

          LXI        H,4000H ;;Begynd her
UD:       IRPC       X,123   ;;Her gives IRPC direktivet, samt de
          MVI        A,X     ;;karakterer der skal indsættes ved
          MOV        M,A     ;;hvert gennemløb
          INX        H       ;find næste adresse
          ENDM
          END
```

```
MACRO-80 3.36  17-Mar-80      PAGE  1

 1                                    ;UD
 2                                    ;label  opcode  operand          kommentarer
 3                                    ;-------------------------------------------------------
 4   0000'                            ASEG
 5                                    ORG     100H
 6
 7   0100   21 4000                   LXI     H,4000H
 8   0103                             UD:     IRPC    X,123        ;;Her gives IRPC direktivet, samt de
 9                                            MVI     A,X          ;;karakterer der skal indsættes ved
10                                            MOV     M,A          ;;hvert gennemløb.
11                                            INX     H            ;find næste adresse
12                                            ENDM
13   0103   3E 01          +                 MVI     A,1
14   0105   77             +                 MOV     M,A
15   0106   23             +                 INX     H            ;find næste adresse
16   0107   3E 02          +                 MVI     A,2
17   0109   77             +                 MOV     M,A
18   010A   23             +                 INX     H            ;find næste adresse
19   010B   3E 03          +                 MVI     A,3
20   010D   77             +                 MOV     M,A
21   010E   23             +                 INX     H            ;find næste adresse
22                                            END
```

PROBLEMFORMULERING

Inden et program kan skrives, skal det problem, som programmet skal løse beskrives. Dette kan gøres ud fra et brugersynspunkt, d.v.s der beskrives hvad programmet skal udføre, og ikke hvorledes det skal udføres.

Denne beskrivelse kan bestå af en verbal formulering, der angiver hvorledes det færdige produkt skal betjenes, eller hvilke uddata der skal komme af givne inddata.

Den første overordnede beskrivelse af en monitor til et øvelsespanel kan se således ud:

> Monitoren skal kunne indlæse data fra et 4 x 4 keyboard til en forudvalgt ramadresse, der er valgt med samme keyboard. Der skal kunne eksekveres programmer der er indlæst fra keyboardet.

Denne beskrivelse kan en programmør ikke anvende til at udvikle sit program efter, idet der er alt for mange løse ender, derfor må monitoren nu defineres mere detaljeret.

Ved nu at begynde at spekulere i hvorledes de forskellige knapper skal virke kommer man tættere på det en progammør kan bruge, men der er stadig lang vej inden programmøren begynder at skrive det egentlige program.

En beskrivelse af øvelsespanelets knapfunktioner kan se således ud:

Monitor til øvepanel med 8085
Følgende knapper er til input:

A	Adresse mode vælges, efterfølgende tal indlæses som adresser.
D	Data mode vælges, efterfølgende tal indlæses som data.
+	Plus indlæser data fra data displayet til adressen, der er vist i adresse displayet, forhøjer denne adresse med 1 og udlæser adressen med tilhørende data på displayet.
-	Minus, som ved plus, men trækker 1 fra adressen.
O..F	Tal, der skal angive data eller adresse afhængig af den valgte mode.
A-D	Ved tryk på A og D samtidig sættes computeren klar til at eksekvere et program. Ved indtastning af startadresse og + eksekveres programmet.

Denne beskrivelse fortæller at talkeyboardet 0..F afleverer data, der anvendes i to forskellige modes. Der findes nogle kontroltaster der styrer hvilken mode der arbejdes i (A og D), samt der er to funktionstaster der indlæser data på den viste adresse plads og forhøjer eller nedtæller adressen med 1.

PROGRAMMERINGS METODER

Når beskrivelsen er tilfredsstillende for bruger og programmør skal programmøren igang med at planlægge hvorledes programmet skal udformes. Dette arbejde er delvis afhængig af hvilket programmeringssprog programmøren har til rådighed, men uanset sproget så vil en detaljeret planlægning af programmet gøre det lettere for programmøren og hans efterfølgere at rette og vedligeholde programmet.

BUTTON-UP

Programmet kan opbygges ved først at kikke på delproblemerne og efterhånden få opbygget hele programmet, dette princip kaldes BUTTON-UP programmering. Metoden har den ulempe at det bliver svært at ændre programmet, fordi det overordnede styreprogram er blevet til på grundlag af de data et lavere liggende modul (underprogram eller del af program) afleverer samt den måde dataerne bliver afleveret på. Da de forskellige dele af et program er meget afhængig af andre dele i programmet vil det blive svært at rette eller ændre, uden at hele programmet skal ændres.

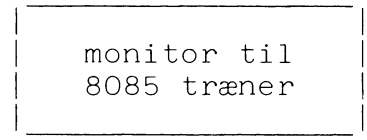
TOP-DOWN

En anden metode, til at udvikle et program med, er et TOP - DOWN princip. D.v.s. man begynder at kikke på hvad hele programmet skal udføre og forsøger derpå at dele det op i mindre delprogrammmer. Ved samtidig at dele programmodulerne op, så kun det sidste modul er hardware afhængig, bliver programmet meget lettere at vedligeholde samt at ændre til nye hardware kredsløb.
Metoden kaldes også funktionsnedbrydning eller trinvis rafinering. Den er ikke en bestemt teknik og der findes ikke kun en god løsning af problemet, men resultatet er afhængig af programmørens erfaring og evne til at opdele større problemer i delproblemer.
Efter funktionsnedbrydningen og en detaljeret beskrivelse af hvert delproblem kan disse moduler omsættes til programmoduler, der hver for sig bør testes og dokumenteres, som om det er enkeltstående programmer.
Ved opdeling på denne måde vil et problem blive beskrevet ved et hieraki af delproblemer.

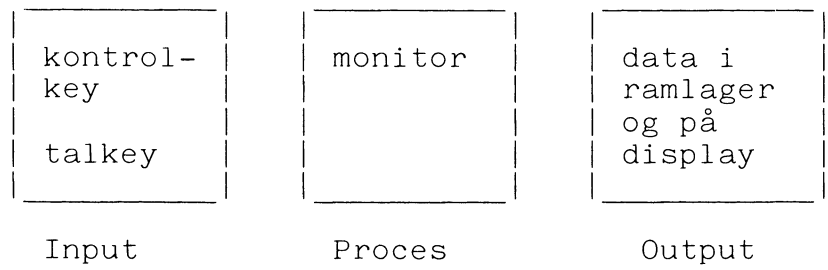
OPDELING I DELPROBLEMER

En opdeling af et problem i delproblemer kan f.eks udføres på følgende måde:

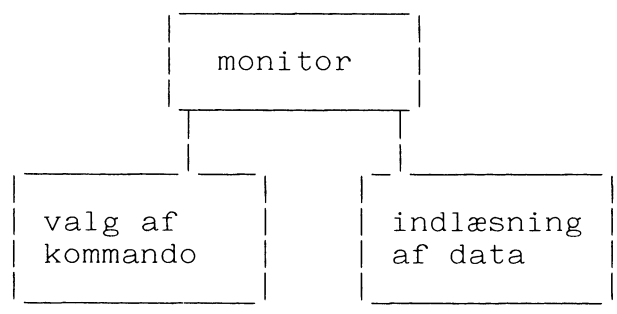
I vort eksempel er problemet en monitor til 8085 træner. Dette tegnes som en kasse med navn i.



Til dette problem må nu findes hvilke input der er til processen der skal udføre problemet, samt hvilke output der er fra processen. Dette tegnes i et I-P-O-diagram. Et I-P-O-diagram består af tre kasser ved siden af hinanden, hvor der i første kasse fra venstre er angivet hvilke input der er til processen, i midten er angivet processens navn og til højre er angivet de output processen skal give.



Efter denne beskrivelse skal processen deles op i mindre delproblemer.



Nu skal hver af disse delprocesser beskrives med et I-P-O-diagram. Derefter skal delprocesserne igen opdeles i delprocesser sa beskrivelsen bliver mere og mere detaljeret.

MODULBESKRIVELSE

Efter denne beskrivelse, der fortæller noget om hvilke ting der skal omsættes i processen, men ikke noget om hvorledes processen omsætter de indkomne data, skal der laves en beskrivelse af hver enkelt modul (delproces). Denne beskrivelse skal fortælle noget om modulets navn og egenskaber.
Ved et modul vil vi her forstå en samling programlinier der tilsammen udfører en delproces. Antallet af programlinier i et modul bør ikke være større end hvad der kan være på 1 til 2 A4 sider. Dersom programmet bliver længere bør man nok overveje at opdele programmet i mindre moduler, da overblikket ellers let tabes.
Modulet beskrives med en grænsefladebeskrivelse, funktionsbeskrivelse og en logikbeskrivelse.

GRÆNSEFLADE

De beskrivelser der har interesse for andre moduler er grænsefladen og funktionen, idet grænsefladen er den forbindelse modulet har til andre moduler. D.v.s. parametre der tilføres modulet eller parametre modulet afleverer til andre moduler.

FUNKTION

Funktionen er det der udføres når modulet aktiveres og ikke hvorledes det udføres. Der skal her bemærkes at dele af funktionen godt kan udføres af et modul der kaldes af det her beskrevne modul.

En modulbeskrivelse kan se således ud:

Name:	KEY
Input:	ingen
Process:	Modulet læser kontroltasterne, der er forbundet til PORT A på en 8255. CS for 8255 er 38XXH. Tasterne er aktiv low. Bitmønster PORT A: 7 6 5 4 3 2 1 0 x x x x - + D A Informationen inverteres og bit 4-7 nulstilles.
Output:	(A)= key værdi
Destroy's:	PSW
Call's:	ingen

LOGIK

Logikken er en beskrivelse (algoritme) der fortæller hvorledes funktionen realiseres, d.v.s de forskellige veje der er gennem et modul for at få funktionen udført.

DATATYPER

I et struktureret program er det data der danner grænsefladerne mellem modulerne, og derved skaber de bindeledet mellem modulerne.
Overføringsmetoden af data er afhængig af datastrukturen modulet skal behandle.

Der findes forskellige datatyper, nogle af disse er:

BINÆRE TAL

Binære tal, er tal der kun kan være 0 eller 1, anvendes ofte i grupper af 8 (1 byte).

0010 0110 (1 byte)

eller 16 (2 byte).

1010 1100 0010 0101 (2 byte)

HELTAL (INTEGER)

Hele tal, d.v.s. tal i et for maskinen forud fastlagt talområde f. eks. 0 - 255 eller -128 til 127.

KOMMATAL (REAL)

Kommatal, der består af et heltal og en brøkdel af en talenhed. Nøjagtigheden og talområdet er fastlagt af det antal byte der er reserveret i maskinen til opbevaring kommatal.

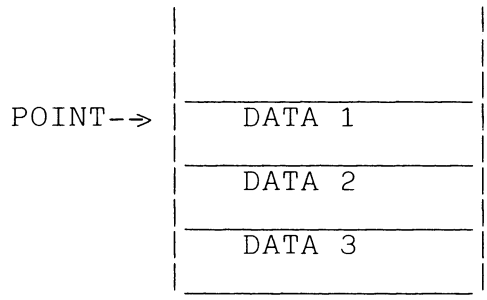
eks.: 245.37

LOGISK VÆRDI (BOOLEAN)

Logiske værdier, der har sandhedsværdierne sand og falsk. Internt representeres det ved henholdvis 1 og 0.

POINTER

Pointer, er et heltal der angiver starten af et lagerområde (buffer).



POINT angiver start af buffer hvor DATA 1, DATA 2 og DATA 3 er de data der skal behandles.

TEGN (CHAR)

Tegn består af de bogstaver, tal og symboler, som er defineret for den benyttede datamaskine.

eks. A B % $ & ? 1 9 §
samt nogle usynlige tegn såsom (nylinie, LF), (vogn retur,CR).

I datateknikken kan man, ligesom i matematikken, opdele sine data i to grupper - konstanter og variable.

KONSTANT

Data der ikke ændres under hele programafviklingen kaldes en konstant. Samtlige datatyper kan anvendes som konstant.

```
eks.:   25              (heltal)
        3.141           (kommatal)
'D''A''T''A'    (tegn)
```

VARIABLE

Data der ændres under programafviklingen kaldes variable. Variablens værdi er til ethvert tidspunkt indholdet af de pågældende lagerceller.

En variabel skal have et navn (indentifier), forskellige variable må ikke have sammme navn. Variablens type er en vilkårlig af ovenstående typer. Typen fremgår ikke dirikte af variablens navn, men skal enten erklæres direkte i programmet (højniveau sprog) eller fremgår af den måde variablens data anvendes på i programmet.

Ud over at skelne mellem datatyper, skelner vi også mellem datastrukturer, her ser vi på to typer datastrukturer.

1. Datasæt, der består af elementer, som alle har samme type (tal, boolean, pointer eller tegn).

2. Poster, der består af en række felter, der ikke behøver at have samme type.

Følgende begreber hører under datasæt.

vektor

matrix (array)

VEKTOR

Når en række data af samme type skal overføres fra et programmodul til et andet, er det uhensigtsmæssigt at angive hver enkelt af disse data eller samtlige navne for data ved dataoverførelsen.

I stedet kan man give data et fælles navn og overføre dem som et eendimensionalt datasæt under dette navn. Et sådant datasæt kaldes en vektor, en tabel eller en eendimensional matrix.

Som eksempel på en vektor kan vi betragte et talsæt bestående af de 5 heltal 10, 20, 30, 40 og 50. Vektoren tildeles navnet N, og kan skrives således:

N= (10, 20, 30, 40, 50).

Vektorens elementer nummereres startende med 0, og de enkelte elementer angives på formen:

vektor (nummer) = element.

For den betragtede vektor N får elementerne således numre fra 0 til og med 4 og kan angives således:

```
N (0) = 10
N (1) = 20
N (2) = 30
N (3) = 40
N (4) = 50
```

Sådanne størrelser kaldes indicerede variable og må anvendes overalt, hvor en variabel må benyttes.

Antallet af elementer i vektoren fastlægges altid i programteksten. Da en vektor således har en fast struktur, er det muligt at tildele et fast sammenhængende område i arbejdslageret til at rumme værdierne for vektorens elementer. Hvis man kender startadressen for dette lagerområde, kan man dermed beregne adressen for et vilkårligt element og således få direkte tilgang til dette element.

Den ovenfor betragtede vektor N tænkes at være lagret på følgende måde:

adresse	lager
100	10
	20
	30
	40
	50

Vektorens basisadresse (startadresse) er således 100, og den er blevet lagret i et sammenhængende lagerområde bestående af 5 celler.

Da vektorens basisadresse er 100, kan vi beregne den effektive adresse for et vilkårligt af vektorens elementer, det tredie element vil få den effektive adresse 102.

```
effektivadresse = basisadresse + index
                =       100    +   2
```

TEKSTRÆKKE (STRING)

En almindelig skreven tekst er en vektor, idet en tekst er en samling tegn. En tekstrække kaldes ofte en string.

```
'dette er en string'
```

MATRIX (ARRAY)

Datasæt kan foruden at være eendimensionale også være flerdimensionale. Et todimensionalt datasæt kaldes en matrix (array).

DATAOVERFØRINGSMETODER

Vi vil her behandle hvorledes almindelige former for dataoverføring mellem programmoduler realiseres.

Som nævnt afhænger dataoverføringsmetoden direkte af hvilke datastrukturer, man behandler.

Hvis man kun skal overføre nogle få data til et modul, kan man overføre disse i en eller flere af mikroprocessorens registre. Dette begrænser datamængden, der kan overføres, eftersom en mikroprocessor har et begrænset antal arbejdsregistre (f.eks. har 8085 mikroprocessoren 6 arbejdsregistre og en akkumulator, hver på 8 bit. Datamængden er således begrænset til 7 bytes). Desuden sætter anvendelsen af samtlige registre til dataoverføring visse begrænsninger på anvendelsen af registre i programmodulerne. Såfremt man skal overføre en større datamængde (mere end 4 bytes) til et modul, vil man følgelig ikke benytte registrene til overføringen, men derimod vælge at pladsere data i et sammenhængende dataområde, og derpå overføre en pointer, der angiver startadressen for dette dataområde, til modulet. Denne overføringsmetode vil typisk blive benyttet for et datasæt.

For at muliggøre en separat udvikling og afprøvning af programmodulerne i form af underprogrammer er det tvingende nødvendigt at data overføres på en standardiseret måde mellem modulerne. Hvis vi ikke indførte en sådan standard, ville det nemlig være nødvendigt at vide, hvilke registre et givet modul anvendte, hvilke lagerceller det benyttede, hvordan det ændrede indholdet af disse registre og lagerceller m.m., d.v.s. man skulle kende de enkelte programtrin i modulet, hvorved hele ideen med at opdele et program i moduler, som kan skrives af forskellige personer, ville gå tabt.

For en 8085 processor kunne en standard for parameteroverføring mellem programmodulerne se således ud:

A. Overføring af parametre til programmodul:

A.1. En parameter med længden 1 byte overføres i register C.

En parameter med længden 2 bytes (f.eks. en pointer) overføres i registerpar B.

A.2. Hvis der er to parametre, overføres den første som angivet under 1.

Den anden overføres på følgende måde:

-1 byte parameter overføres i register E.
-2 bytes parameter overføres i registerpar D.

A.3. Hvis der er mere end to parametre, overføres de to sidste som beskrevet under A.2, og de resterende overføres i stakken.

Returnering af resultater fra et programmodul:

B.1. Hvis resultatet er 1 byte returneres det i akkumulator A.

Hvis resultatet er 2 byte returneres det i registerpar H.

KONTROLSTRUKTURER

Hvorledes bør logikken i et modul udformes, for at funktionen bliver udført korrekt?

Vi vil her se på hvorledes programmets logik kan udformes så det bliver let at fejlrette og dermed også let at læse.

For at lette læseligheden af et program bør der kun anvendes bestemte kontrolstrukturer så som:

SEKVENS

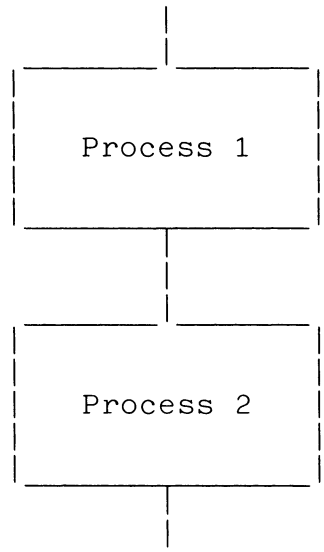
UDVÆLGELSE

GENTAGELSE

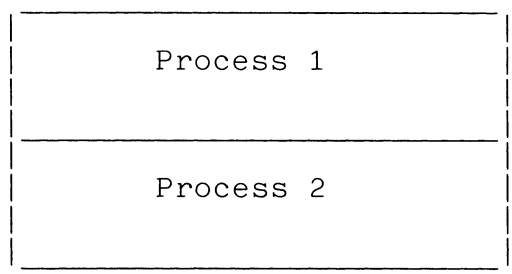
SEKVENS

Ved en sekvens forstås en række programlinier der udføres i den rækkefølge de forekommer i programmmet. En sekvens kan også bestå af nogle moduler der udføres i rækkefølge.
I en sekvens har delelementerne kun en indgang og en udgang.

Et flowdiagram af nogle sekvenser ser således ud:



Et Nassi Schneidermann diagram med nogle sekvenser ser således ud:

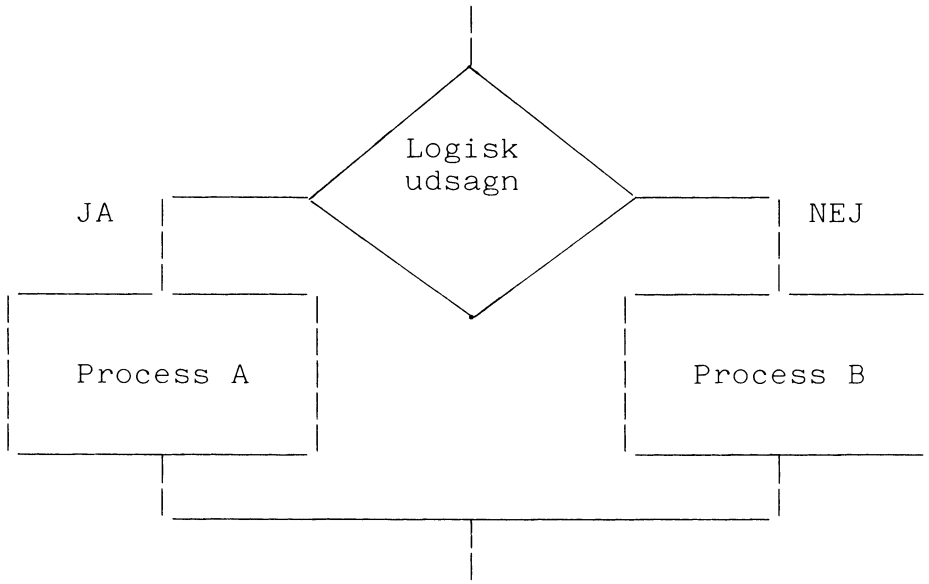


Et program vil altid set udefra bestå af en sekvens der påbegyndes når der tændes for programmet og afsluttes når der stoppes.

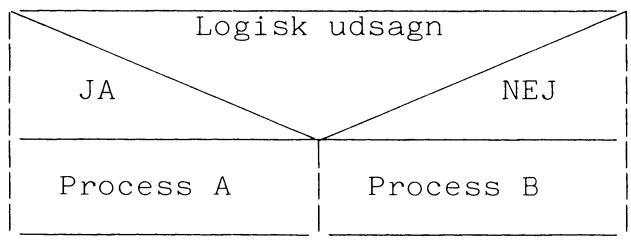
UDVÆLGELSE

En anden grundlæggende kontrolstruktur er udvælgelsen. Ved udvælgensen vil der være en indgang og to udgange.

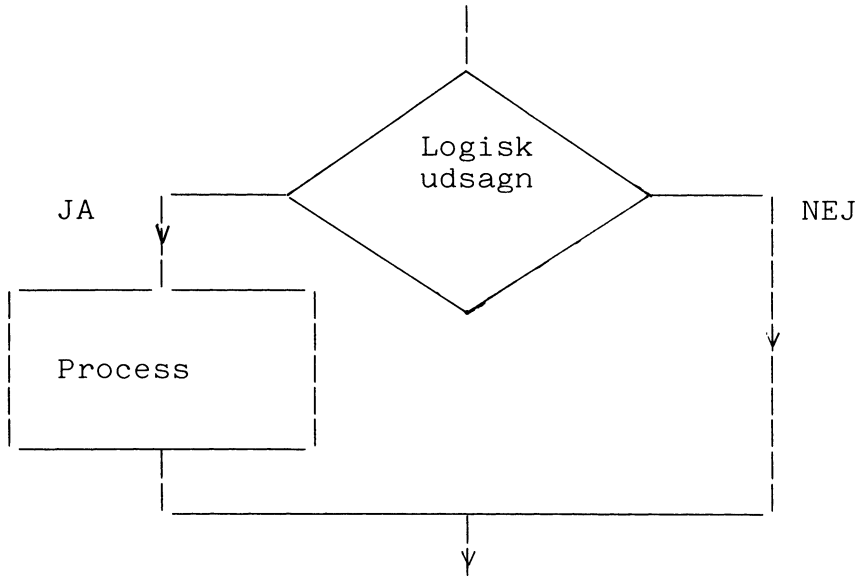
Et flowdiagram ser således ud:



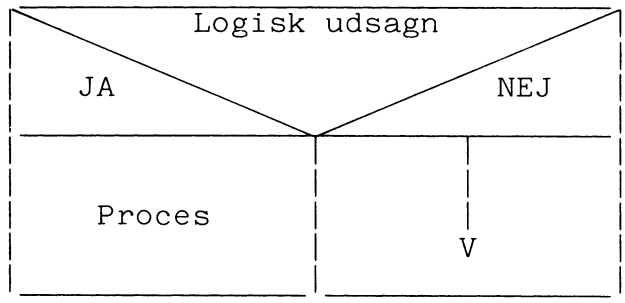
Tilsvarende Nassi Schneidermann diagram ser således ud:



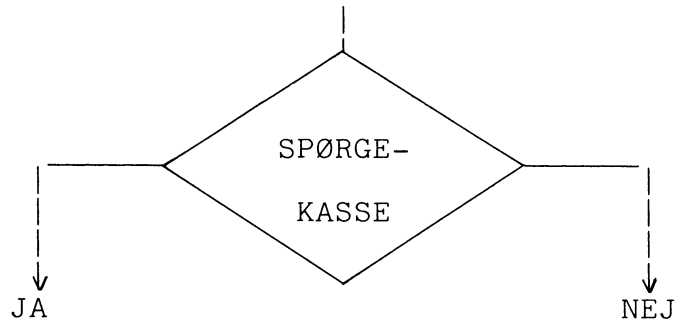
I de her viste eksempler er det et spørgsmål om at vælge mellem to forskellige processer og dersom den ene ikke udføres skal den anden udføres. I mange programmer kan der ofte være tale om at en del af et program under bestemte forudsætninger skal udelades, så er det ikke længere et valg mellem to processer, men et valg om en proces skal udføres eller springes over. Dette ser således ud i et flowdiagram:



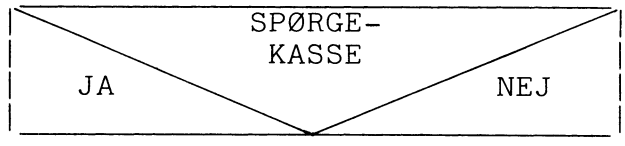
Et Nassi Schneidermann diagram ser således ud:



For at lette læseligheden vælges venstre søjle altid som ja udgang på spørgsmålet i spørgekassen.



Betingelsen i spørgekassen skal altid udformes så det kan besvares med et ja eller et nej.

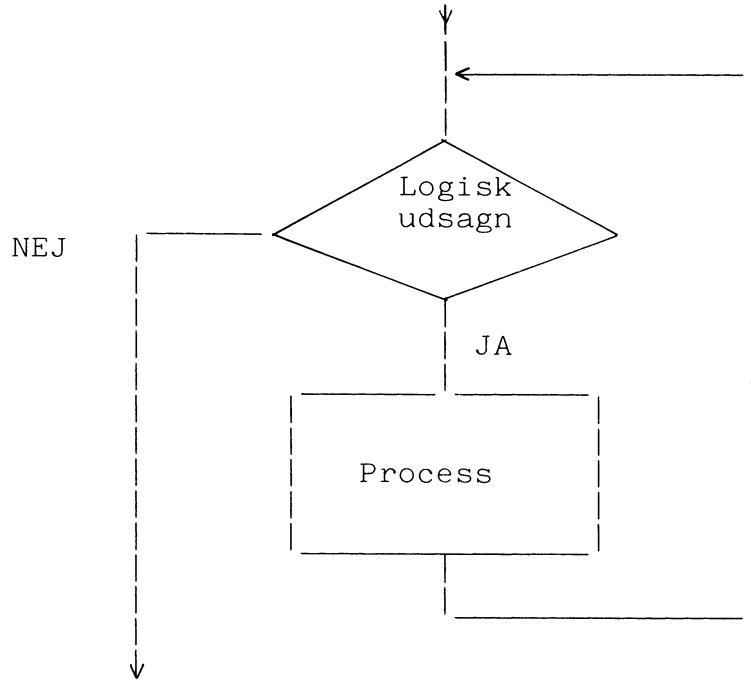


GENTAGELSE

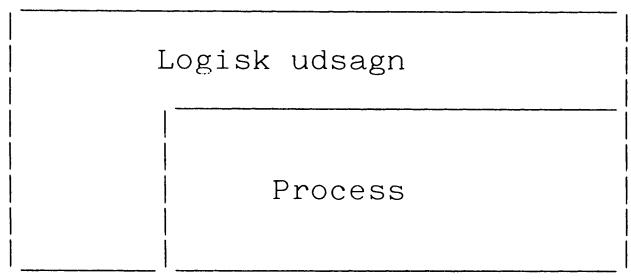
Den tredie kontrolstruktur er gentagelsen med et eller andet logisk udsagn der er betingelsen for gentagelsen. Der er to former for gentagelser, den ene skal altid behandles mindst een gang, det er "gentag indtil". Ved den anden form skal startbetingelsen være opfyldt for at der overhovedet foretages en gentagelse, dette kaldes "udfør mens".

MENS (WHILE)

Udfør mens, ser således ud i et flowdiagram:

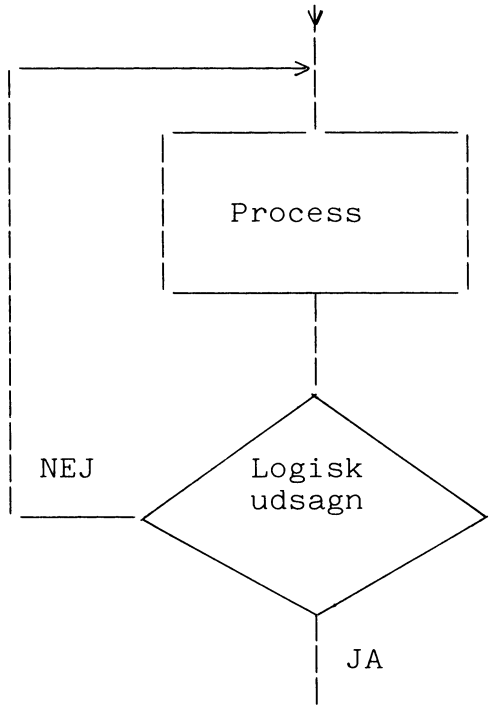


Som Nassi Schneidermann diagram ser det således ud:

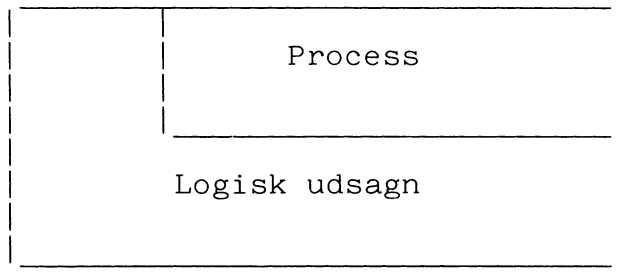


GENTAG INDTIL (REPEAT-UNTIL)

En gentag indtil vil se således ud:
Flowdiagram



Nassi Schneidermann diagram



Når logikken i modulet er opbygget ved hjælp af de her viste kontrolstrukturer bliver det lettere at læse programmet idet hver kontrolstruktur har kun een indgang og een udgang.

HØJNIVEAUSPROG

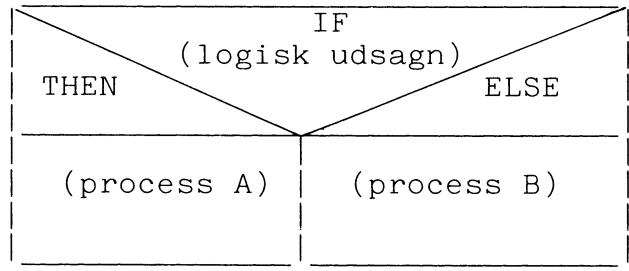
Hver af disse kontrolstrukturer kan i et struktureret højniveausprog direkte omsættes til programlinier. I assemblersprog omsættes hver kontrolstruktur til en gruppe af instruktioner.

IF-THEN-ELSE

I højniveausprog vil en udvælgelse se således ud:

```
IF (logisk udsagn)
  THEN
    (process A)
  ELSE
    (process B)
END
```

Det dertil hørende strukturgram (Nassi Schneidermann diagram) ser således ud:

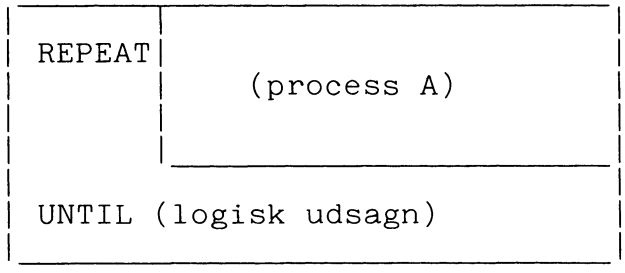


REPEAT-UNTIL

En gentagelse der altid gennemløbes een gang kan se således ud:

```
REPEAT

  (process A)

UNTIL (logisk udsagn)
```

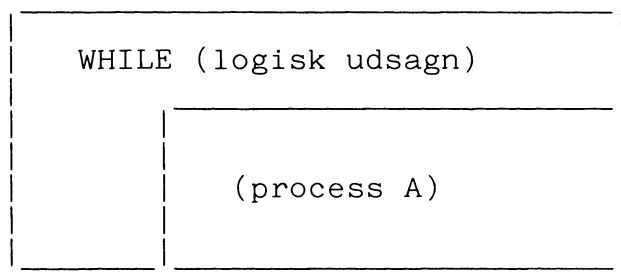
Som strukturgram ser det således ud:



WHILE

En anden gentagelse der gentages indtil det logiske udsagn ikke er opfyldt ser således ud:

```
WHILE (logisk udsagn)

  (process A)

END
```

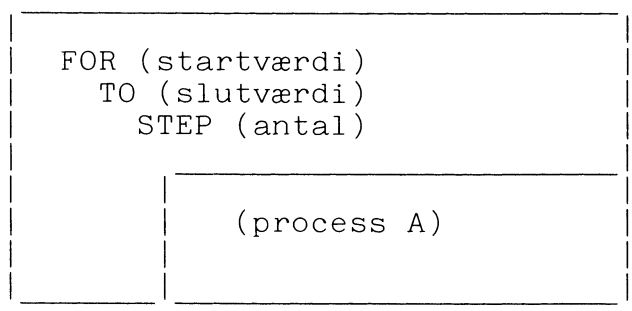


For de to viste gentagelser gælder at det logiske udsagn skal ændres v.h.a. processen da programmet ellers aldrig vil komme videre.

FOR-TO-STEP

En gentagelse der gentages et forud bestemt antal gange uanset processen ser således:

```
FOR (startværdi) TO (slutværdi) STEP (antal)

  (process A)

NEXT (værdi)
```



ASSEMBLERKODE

Når strukturgrammerne skal omsættes til assemblerkode går det ikke helt så let, idet hver kontrolstruktur altid vil bestå af flere instruktioner. Når 8085 skal udføre en sammenligning af to størrelser skal den ene af størrelserne altid være i reg A dette medfører at når et logisk udsagn skal undersøges om det er sand eller falsk skal den ene del af udsagnet (variabel data) altid flyttes til reg A. Den anden størrelse indgår i en compare (sammenligning) instruktion, vi vil her kalde denne størrelse for reference værdien (REF). Efter en compare instruktion vil CPU'ens flag være sat afhængig af forholdet mellem de to størrelser der er blevet sammenlignet.

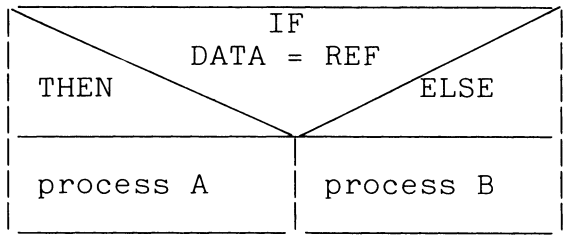
Flag vil være sat som følgende:

logisk udsagn	Z	C
(A) < REF	0	1
(A) = REF	1	0
(A) > REF	0	0

Efter compare instruktionen skal der være et eller flere betingede jump (hop). Da kontrolstrukturerne er opbygget så det logiske udsagn er sandt når processen skal bearbejdes og springes over når det logiske udsagn er falsk, skal de betingede jump springe når det logiske udsagn ikke er opfyldt.

Eks.:

IF DATA = REF THEN (process A) ELSE (process B)



Assembler programmet til ovenstående struktogram kan findes til:

```
IF:       flyt DATA til A
          compare REF
          JNZ ELSE
THEN:     (process A)
          JMP ENDIF
ELSE:     (process B)
ENDIF:    .....          ;fortsættelse af program
```

(flyt DATA til A) skal erstattes med den/de instruktioner der flytter DATA til A.

(compare REF) erstattes med den/de instruktioner der sammenligner (A) med REF.

(process A) erstattes med instruktionerne for process A.

(process B) erstattes med instruktionerne for process B.

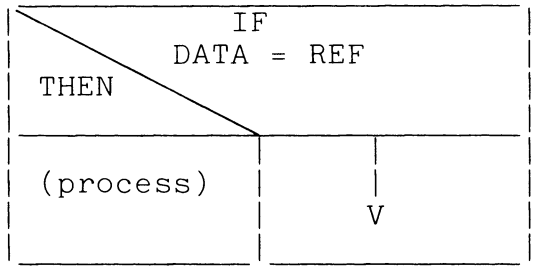
Assemblerkoderne til alle strukturerne i højniveausprog kan man naturligvis selv finde, hver gang man har brug for det, men det er hurtigere at foretage et tabelopslag til et afprøvet sæt assemblerkoder til den pågældende højniveausætning, herved opnår man en meget høj grad af sikkerhed for at man springer på de rigtige betingelser, på de rigtige flag.

På de efterfølgende sider vises assemblerkoderne til de grundlæggende højniveausætninger, i appendix kan de mere sammensatte strukturer findes.

IF - THEN

eks.:

```
                                    IF DATA = REF
                                     THEN (process)
                                    ENDIF
```

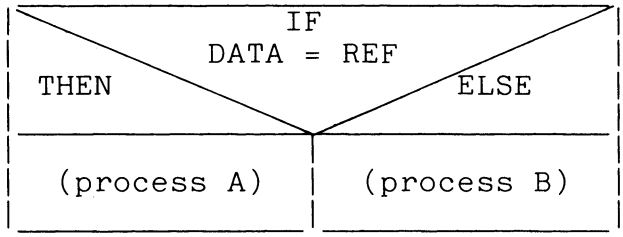



```
IF:     flyt DATA til A
        compare REF
        JNZ ENDIF
THEN:   (process)
ENDIF:  ....
```

logisk udsagn	assembler koder	
	IF:	flyt DATA til A compare REF
(A) < REF		JNC ENDIF
	THEN:	(process)
	ENDIF:	.....
(A) <= REF		JZ THEN JNC ENDIF
	THEN:	(process)
	ENDIF:	.....
(A) = REF		JNZ ENDIF
	THEN:	(process)
	ENDIF:	.....
(A) => REF		JC ENDIF
	THEN:	(process)
	ENDIF:	.....
(A) > REF		JZ ENDIF JC ENDIF
	THEN:	(process)
	ENDIF:	.....
(A) <> REF		JZ ENDIF
	THEN:	(process)
	ENDIF:	.....

IF - THEN - ELSE

eks.:

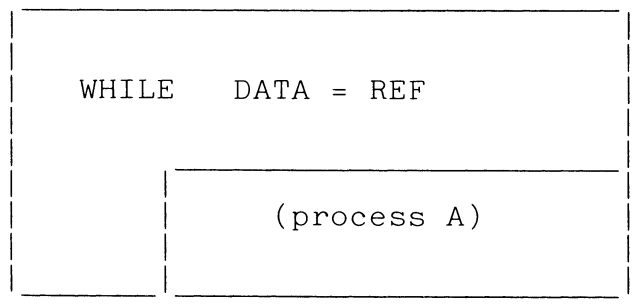



```
IF DATA = REF
 THEN (process A)
  ELSE (process B)
ENDIF

IF:      flyt DATA til A
         compare REF
         JNZ ELSE
THEN:    (process A)
         JMP ENDIF
ELSE:    (process B)
ENDIF:   ......
```

logisk udsagn	assembler kode
	IF: flyt data til A compare REF
(A) < REF	JNC ELSE THEN: (process A) JMP ENDIF ELSE: (process B) ENDIF: .....
(A)<= REF	JZ THEN JNC ELSE THEN: (process A) JMP ENDIF ELSE: (process B) ENDIF: .....
(A) = REF	JNZ ELSE THEN: (process A) JMP ENDIF ELSE: (process B) ENDIF: .....
(A) => REF	JC ELSE THEN: (process A) JMP ENDIF ELSE: (process B)
(A) > REF	JZ ELSE JC ELSE THEN: (process A) JMP ENDIF ELSE: (process B) ENDIF: .....
(A) <> REF	JZ ELSE THEN: (process A) JMP ENDIF ELSE: (process B) ENDIF: .....

WHILE - DO

eks.:

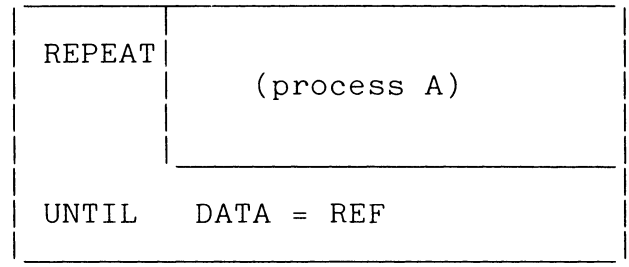



```
WHILE DATA = REF
 DO (process)
ENDWHL

WHILE:  flyt DATA til A
        compare REF
        JNZ ENDWHL
DO:     (process)
        JMP WHILE
ENDWHL: ......
```

logisk udsagn	assembler koder
	WHILE: flyt data til A compare REF
(A) < REF	JNC ENDWHL DO: (process) JMP WHILE ENDWHL: .....
(A) < = REF	JZ DO JNC ENDWHL DO: (process) JMP WHILE ENDWHL: .....
(A) = REF	JNZ ENDWHL DO: (process) JMP WHILE ENDWHL: .....
(A) = > REF	JC ENDWHL DO: (process) JMP WHILE ENDWHL: .....
(A) > REF	JZ ENDWHL JC ENDWHL DO: (process) JMP WHILE ENDWHL: .....
(A) <> REF	JZ ENDWHL DO: (process) JMP WHILE ENDWHL: .....

REPEAT - UNTIL

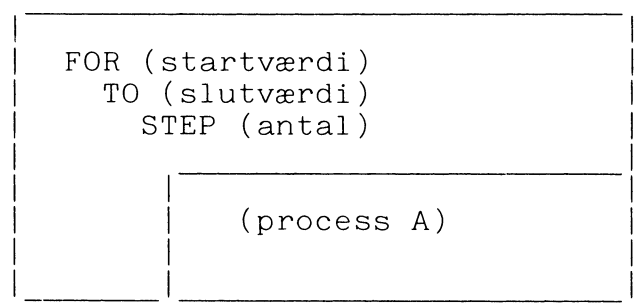
eks.:



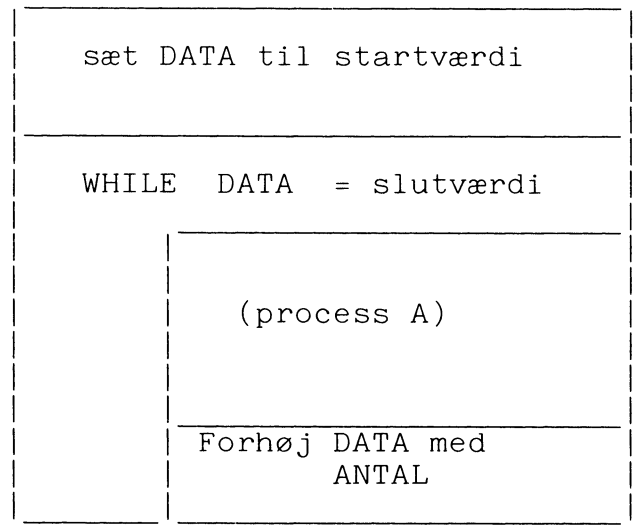

```
REPEAT (process)
 UNTIL DATA = REF
ENDREP


REPEAT: (process)
        flyt DATA til A
        compare REF
        JNZ REPEAT
ENDREP: ......
```

logisk udsagn	assembler koder
	REPEAT: (process) flyt DATA TIL A compare REF
(A) < REF	JNC REPEAT ENDREP: .....
(A) < = REF	JZ ENDREP JNC REPEAT ENDREP: .....
(A) = REF	JNZ REPEAT ENDREP: .....
(A) => REF	JC REPEAT ENDREP: .....
(A) > REF	JZ REPEAT JC REPEAT ENDREP: .....
(A) <> REF	JZ REPEAT ENDREP: .....

FOR - TO - STEP



En FOR-TO-STEP kan udføres med en WHILE struktur således:



1. I hvilke grupper kan instruktionerne opdeles ?

2. Placer følgende OP-kode i den rigtige gruppe:
   97, 27, CA, OO,21, C3, A7, 6O, 1F, 76

3. Hvor mange betydninger kan en et bytes information have i en 8o8o/85 computer ?

4. Hvad består en instruktion af ?

5. Hvor mange bytes kan en instruktion omfatte ?

6. Hvad starter et program altid med ?

7. Undersøg hvor mange bytes følgende instruktioner omfatter: 3E, 21, OO, 6C, 76

8. Nævn CPU´ens registre.

9. I hvilket register placeres OP-koden ?

1o. Hvad anvendes PC´en til ?

11. Hvad anvendes SP´en til ?

12. Hvilke 3 hovedelementer består CPU´en af ?

13. Modtager eller udsender CPU´en adresser ?

14. Modtager eller udsender CPU´en data ?

15. Nævn de 3 bussystemer og angiv hvor mange bit der er i hver bus.

16. Hvad forstås ved tri-state logik og hvorfor anvendes den i computeren ?

17. Hvor mange lagerlokationer kan en 8o8o/85 CPU adressere ?

1. Hvad forstås ved følgende udtryk ?

   a) RANDOM ACCESS.
   b) SEKVENTIEL ACCESS.
   c) VOLATILE - NON/VOLATILE.
   d) ACCESS TIME.
   e) RAM.
   f) ROM.
   g) PROM.
   h) EPROM.

2.

Sæt x ved rigtigt udsagn.	RANDOM ACCESS.	SEKVENTIAL ACCESS.	KUN READ.	BÅDE READ OG WRITE.	WRITES MED BRÆNDE UDSTYR.	VOLATILE.	NON-VOLATILE.	HURTIG ACCESS TIME.	MIDDEL ACCESS TIME.	LANGSOM ACCESS TIME.	1K - 1ooK BYTE	1ooK - 1M Byte	STØRRE END 1M BYTE
RAM													
ROM													
PROM													
EPROM													
FLOPPY-DISK													
TAPE													
HARD-DISK													

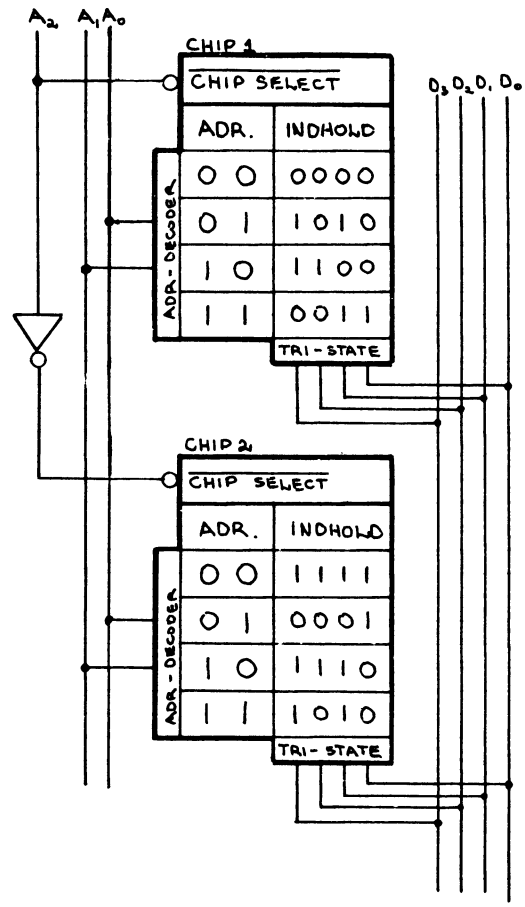
3. A) På ADR-bus tilføres 1o1.
   Hvad er output på data-bussen ?

   B) På data-bussen ønskes data 1100.
   Hvilken ADR skal tilføres ADR-bus ?

   C) Hvad forstås ved TRI-State ?

   D) Hvilken terminal på chippen styrer TRI-State udgangen ?

   E) Hvad kaldes den tid der går fra ADR er tilført A-bussen til et stabilt data opnås på udgangen.



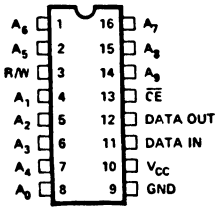
4. Hvilken kreds er en:

   2K x 8 bit EPROM.

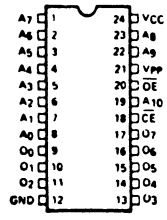
   1K x 4 bit RAM.

   1K x 1 bit RAM.

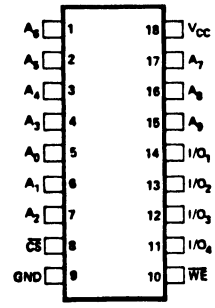
   1K x 8 bit EPROM.



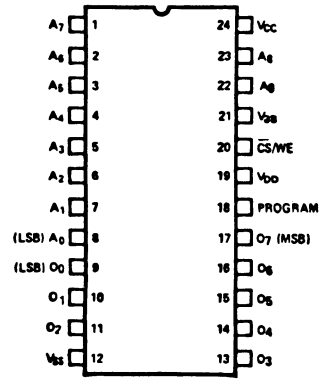
A



B



C



D

1. Sæt register D til 6. Hvorledes kan det kontrolleres.

2. Sæt register E til -5.

3. Sæt register A til AA. Sæt register B til BB.
   Sæt derefter register A til indholdet af register B.

4. Anbring indholdet af adresse 822o i register B.

5. Anbring indholdet af adresse 822o i register A og
   multiplicer med -1.

6. Sæt indholdet af adresse 823o til den negative værdi
   af adresse 824o.

7. Ombyt indholdet af ADR 824o og 8245.

8. Sæt register A til O.

9. Nulstil indholdet i ADR 823o til 8235.

1o. Anbring OO i ADR 822o
   O1 i ADR 8221, o2 i ADR 8222 og o3 i ADR 8223.

11. Fjern de 4 mest betydende BIT (nulstilles) fra ind-
   holdet i ADR 822o og flyt resultatet til ADR 8221.

12. Indlæs o3H i ADR 822o. Multiplicer o3H med 4 og placer
   resultatet i ADR 8221.

13. Se næste side.
   Udfyld de manglende felter.
   Hvad udfører programmet.

PROBLEM NO.	START ADDR	PROGRAM

UDFYLD DE MANGLENDE FELTER !

DESCRIPTION HVAD UDFØRER PROGRAMMET ?

DATE

SHEET 1 OF 1

LINE	ADDRS	DATA B0	DATA B1	DATA B2	LABEL	SYMBOLIC INSTRUCTION OPCODE	SYMBOLIC INSTRUCTION OPERANDS	COMMENT
1	2000	21	20	20		LXI H	2020 H	
2	2003					MOV A,M		
3	2004					MOV B,A		
4	2005					RRC		
5	2006	OF				RRC		
6	2007	OF				RRC		
7	2008	OF				RRC		
8		E6	OF			ANI	OF H	
9		23				INX H		
10		77				MOV M,A		
11	200D	78				MOV A,B		
12	200E	E6	OF					
13	2010	23						
14	2011	77						
15	2012	76				HLT		
16								
17								
18								
19								
20								
21								
22								
23								
24								

TEORI

Flag-byte stack format:

S	Z	O	AC	O	P	1	C

S = Sign

Z = Zero

AC = Auxilliary Carry

P = Parity

C = Carry

Udfyld de tomme felter

Computeren Adderer Følgende Data:	PSW: A-register indeholder	PSW: Flag-byte indeholder	Hvilket flag er sat	Kan der springes på dette.	Hvad hedder MNEMONICEN for springet
O2H + O3H					
O3H + O4H					
OOH + OOH					
O8H + O8H					
8OH + 8OH					
8OH + OOH					
C3H + 76H					

PRAKTIK

Ovenstående afprøves i computeren.

Stemmer PSW med hvad du har fundet ?

TEORI

1. Udfør additionen af følgende Hex-data

   a) 44H + 27H = _____

   b) 3FH + 6CH = _____

   c) 83H + 79H = _____

   d) BAH + DEH = _____

2. Hvad er de decimale værdier, når data er unsigned

   a) _____ + _____ = _____

   b) _____ + _____ = _____

   c) _____ + _____ = _____

   d) _____ + _____ = _____

3. Hvad er de decimale værdier, når data er signed notation (2-complement).

   a) _____ + _____ = _____

   b) _____ + _____ = _____

   c) _____ + _____ = _____

   d) _____ + _____ = _____

4. Skriv et program, som kan udføre ovenstående addition. Første tal anbringes i ADR 822oH. Andet tal i ADR 8221H. Resultat i ADR 8222H. Program starter i ADR 82ooH.

PRAKTIK

5. Indtast programmet og undersøg om de fundne værdier under punkt 1 stemmer med computeren.
   Husk at undersøge flag.

6. Skriv og afprøv et program, som udfører følgende:
   Et tal i intervallet OIH til 7FH placeres i ADR 8220H. Tallets 2-complement beregnes og resultatet placeres i ADR 8221H.

7. I ovenstående program indlæses et tal i intervallet 8oH til FFH på ADR 822oH. Det forudsættes at computeren arbejder med signed notation (2-complement). Hvordan skal informationen i ADR 8221 tolkes ?

TEORI

Opgave A-reg indlæses med tallet 3, som derefter decrementeres og når A = 0 standser programmet.

1. Udskriv flow-chart.
2. Udskriv programmet på vedlagte.programmeringsark. Start ADR 8200H.
3. "Run programmet i nedenstående skema trin for trin.

PC	A-REG	FLAG-BYTE	Z-FLAG

PRAKTIK

Afprøv ovenstående i computeren trin for trin.

TEORI

Opgave A-reg indlæses med tallet 6, som derefter decrementeres og når A=3 standser programmet.

1. Udskriv flow-chart.
2. Udskriv programmet på vedlagte programmeringsark. Start ADR 8200 H.

PRAKTIK

Afprøv programmet trin for trin.

PROBLEM NO.	START ADDR	PROGRAM	
DESCRIPTION			DATE / SHEET OF

LINE	ADDRS	DATA B0	DATA B1	DATA B2	LABEL	SYMBOLIC INSTRUCTION OPCODE	SYMBOLIC INSTRUCTION OPERANDS	COMMENT
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								
24								

HH SCREENEDITOR V 4.0

Editoren har tre arbejdsformer(modes):

COMMAND MODE
TEXT MODE
INSERT MODE

COMMAND MODE:

Bruges
- ved indlæsning af tekst fra og til disk.
- ved editering.

TEXT MODE:

Her kan
- cursor flyttes overalt i teksten.
- tekst overskrives med ny tekst.
- karakterer fjernes enkeltvis.

INSERT MODE:

Her kan
- tekst indskydes i bestående tekst.
- tekst tilføjes i forlængelse af bestående tekst.

Editoren viser øverst til højre på skærmen, hvor mange bytes der er tilbage i datamatens hukommelse, som kan rumme optil ca. 50.000 bytes ad gangen. Hver byte svarer til et tegn, bogstav eller tal, og normalt står der " 32767 BYTES MEMORY FREE" d.v.s. plads til mere end 32767 tegn i hukommelsen, når skrivning påbegyndes. Hvis der under skrivningen bliver mindre end 32767 bytes til overs, vil det antal, som til enhver tid er til overs, løbende blive vist. Editor giver desuden meldingen "memory overflow", når antal bytes er under 1730 (plads til en hel sides skærmtekst). D.v.s. at ved "memory overflow" kan antal ledige bytes ligge mellem 0 og 1729.
"Memory overflow" kan kun fjernes med 'W' kommandoen (se nedenfor) indtil der igen er mere end 1729 bytes ledige.

Kommandoer i COMMAND MODE:

nA	Indlæser n linier fra tekstfil på disk til memory.
OA	Indlæser fra tekstfil på disk til memory er halvt fuld.
#A	Indlæser fra tekstfil på disk til memory er næsten fuld.
nW	Udlæser n linier fra memory til tekstfil på disk.
#W	Udlæser til tekstfil på disk til memory er tømt.
T	Vælger TEXT MODE.
Q	Afslutter skrivning og annullerer rettelser i tekstfilen.
E	Afslutter skrivning og indfører rettelser i tekstfilen.
n:	Sætter tekstlinie n som øverste linie på skærmen.
+/-n	Flytter n linier frem eller tilbage i forhold til øverste linie på skærmen.
nF*	Søger efter "*" i teksten n gange begyndende fra øverste linie på skærmen. Linien med det søgte "*" bliver så øverste linie. "*" kan være et ord eller et tal, og indskydes "CONTROL L" søges efter "RETURN + linieskift". Eksempel: F LTEKST søger efter ordet TEKST, hvor det står først i en linie (nemlig efter "RETURN + linieskift).
nK	Sletter n linier begyndende med øverste linie på skærmen.
nX*	Etablerer en fil på n linier begyndende med øverste linie på skærmen. Filen får navnet "*.TXT", * kan være max. 8 bogstaver eller tal.
R*	Indskyder filen med navn "*.TXT" i teksten på skærmen foran den øverste linie.
nP	Printer n linier ud på printer, begyndende med øverste linie på skærmen.

BEMÆRK: med nX$* og R* kan tekstdele bekvemt ombyttes under editering.

I kommandoer, hvori n indgår sættes n automatisk til 1, hvis intet tal skrives. Dette gælder IKKE for +/-n.

En kommando kan bestå af flere enkeltkommandoer, i alt op til 70 tegn. Kommandoer uden * (navn, ord eller tal) skrives lige efter hinanden; men efter kommandoer med * skrives "CONTROL Z" for at skille mellem teksten og næste kommando.
Eksempel: 10XA01 Z10K etablerer en fil på 10 linier med navn A01 og sletter derefter de 10 linier.

ADVARSEL: Der er følgende begrænsninger i anvendelsen af sammensatte kommandoer:

W-kommando.	Må KUN efterfølges af A- eller :-kommando, ellers tabes data.
:-kommando.	Må KUN bruges til sidst i en sammensat kommando, ellers tabes data.
T-kommando.	Kan IKKE indgå i en sammensat kommando.

Kommandoer i TEXT MODE:

"Pil op"	Flytter cursor en linie op; hvis cursor står på øverste linie "blades" til foregående side.
"Pil ned	Flytter cursor en linie ned; hvis cursor står på nederste linie "blades" til næste side.
"Pil t.h."	Flytter cursor mod højre.
"Pil t.v."	Flytter cursor mod venstre.
"DEL"	Sletter det tegn, som cursor står på. Hvis cursor står umiddelbart efter sidste tegn på linien slettes linieskift, ("DEL" = delete = slet).
"INS"	Vælger INSERT MODE, (insert = indskyd).
"ESC"	Vælger COMMAND MODE, ("ESC" = escape = undslip, forlad).

Kommandoer i INSERT MODE:

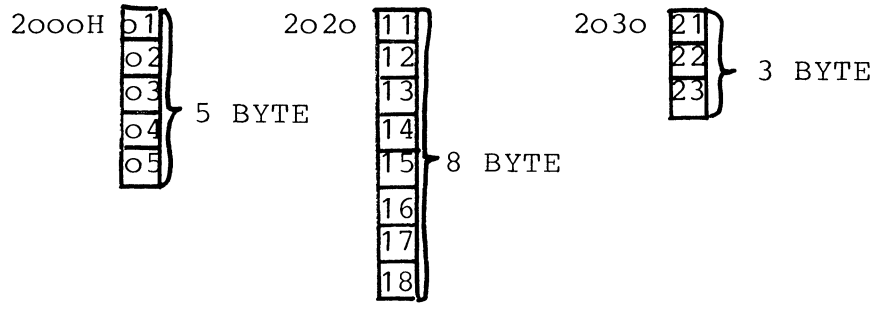
"Pil t.v."	sletter indenfor samme linie et tegn ad gangen.
"Pil t.h."	vælger TEXT MODE (uden at flytte cursor).
"Pil op"	-- "" -- -- " " --
"Pil ned"	-- "" -- -- " " --

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                     ;
;       Dette program adderer ti tal med mente        ;
;       og gemmer summen på pladsen "SUM".            ;
;       Begrænsning: max. sum = 255                   ;
;                                                     ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


SUM     EQU     020AH
START   EQU     0200H
ANTLA   EQU     10

;Startadressen er i H,L reg.
BEGYND: LXI     H,START

;Antallet er i D,E reg.
        HVI     D,00
        MVI     D,ANTAL
        DCX     D

;Clear carry og flyt det første tal ind i A.
        ORA     A
        MOV     A,M

;Juster adressen i H,L reg.
LOOP    INX     H

;Adder!
        ADD     M

Juster antal en ned
        DCX     D

;Færdig?
        JZ      LOOP

;Gem summen i "SUM"
        STA     SOM
        RST     7
        END     BEGYND
```

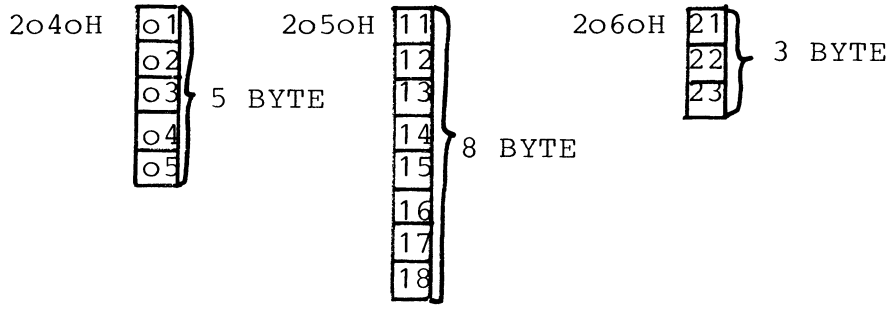
1. 16 lager pladser startende på ADR 83ooH
   ønskes aloceret til ny start ADR 832oH.

   A) Fremstil flow-chart.
   B) Udskriv program på SCREEN EDITOR.
   C) Assembler, link og load v.hj. af M8o og L8o.
   D) Run og test program i DDT.

2. 3 lagerområder ønskes aloceret på følgende måde
   FØR



   Efter program afvikling.
   NY START



   Løsningen fra opg. 1 anvendes.
   I stedet for at skrive programmet 3 gange
   anvendes MACRO FACILITETEN.

   Skriv, ASSEMBLER, LINK, LOAD, RUN og TEST PROGRAMMET.

3. Fremstil et program som kan sortere 1o,
   forskellige værdier (oo-FF) i stigende orden.
   Skriv og afprøv programmet i udviklingssystemet.