Vi vil nå se litt på assemblerspråket som brukes på ITT-1600 og telex-sentralen fra ITT.

Maskinen er på 16 bit + 1 paritetsbit.

Instuksjonslisten omfatter aritmetiske operasjoner, med undtagelse av multiplering og dividering. Logiske operasjoner. Skift-instruksjoner og jump-instuksjoner. Skip-instruksjoner (hoppe over neste statement dersom visse forutsetninger er gitt)

Operasjoner med ett eller flere bits og input-output operasjoner.

Minne-beskyttelse mot innskriving av sektorer av 512 ords størrelse.

Programbare registre er i maskinen:

A-register: 16-bit Akumulator-register

B-register: Tillegg til A-registeret. Kan brukes som forlengelse mot høyre av A-registeret. 16 bit. (Tilsvarer D-registeret i NORD-1 ?)

X-registeret: 16-bit Index register. Brukes ved adressering.

c-bit: Overflow-indikator. Brukes også ved skifting og roterings instruksjoner på registrene.

Q-registeret: STACK-peker, brukes i forbindelse med subrutine hopp til å peke på en bestemt sted i hukommelsen hvor datane og instruksjonen i forbindelse med en sub-rutinge ligger. (tilsvarer B-registeret i NORD-1). X-registeret brukes vell så til å ta hvert ord i "stack"'en.

Addressering av registre

X, A og B har adressene henholdsvis 0, 1 og 2 de tilsvarende minne-adressene brukes ikke.

C-bit adresseres implesit ved instruksjoner som bare brukes i forbindelse med den.

Registeret Q, har adressen 40 oktalt. Den tilsvarende minne-adressen brukes derfor ikke.

IKKE PROGRAMBARE-REGISTRE.

Programtelleren, PC: 16 bit teller som inneholder adressen til instruksjonen som blir utført.

Y-register: 16-bits Minne-lokaliserings-register

Tilsvarer B-registeret i NORD.

De-plice memb. reg.

En spesiell ting ved telefon prosessen er at de krever et gigantisk program å store data mengder som er semi-permanente (datamaskin synes at de er permanente (kort tid)) Alle disse ting må være direkt accets (alt må ligge i core..) Man får et behov for å adressere seg til en meget stor direkte hukommelse.

Hukommelse, reg.

M-register: 16 bit minne- buffer-register. (Danner forbindelsen mellom hukommelsen og maskinen forøvrig. (tilsvarer ~~MB~~ i Nord 1 ?) ~~xxxx eller tilsvarer h-registeret i NORD-1 ?)~~

E-register JRL 16 bit register som brukes til å holde adressen til en "execute"-instruksjon mens "execute-sequencen" blir utført. (Jump and link)

Skift-teller: 5-bits teller for intern kontroll av visse instruksjoner.

F-register: 7-bit ~~xxxxxxxxxxxxxxxxxxxxxxxxxxx~~ IR reg. register som inneholder operasjonskoden (tilsvarer IR-registeret i NORD-1)

Fase-registeret: FIAB 4 bit register som indikerer den fasen eller (som NORD-1 manualen kaller det) syklusen, som er i ferd med å utføres. Fasene er: F, "fetch" d.v.s hent instruksjonen fra ~~x~~ minnet; I, indirekte addresseringsfase, A og B er de to eventuelle fasene i hvilke instruksjonen blir utført. (F tilsvarer C0, I tilsvar C2 i Nord 1)

Cycluser

En har 5 16 bits bus-linjen nemlig:
INB, input bus
OTB, output bus
MIB, memory input bus
MOB, memory output bus
MAB, memory adress bus

dessuten
en 10 linjers adresse bus, ADB med I/O apparat adresse og funksjons kode overføring.
distribution bus Viktigste inter-regeigster veg.

Sektorer

Det 9-bit store normale adresse feltet~~x~~ (displacement er NORD-1 betegnelsen) innbyr til et minne-sektors begrep.
En har 4 sektorer i memory.

Adr. formål

Sektor ~~x~~ 0: Tabeller
Den har 512 ord med adresse fra 0 til 511 . Denne sektor brukes tabel-adresser og for arbeidslager som kan nås direkte av mange av instruksjonene.

Sektor ~~x~~ 1: Macroer
Inneholder 512 ord med adresser fra 512 til 1023. Denne sektoren brukes i forbindelse med Execute-instruksjoner og for andre formål som en finner å ville legge der.

Hopp tol sub. program.

Sektor 2. Cellenavn fra 1024 til 1535. Sektoren brukes for indirekte adresser og jump-adresser. Det første ordet i denne ~~xxx~~ sektoren inneholder adresser for programmer for intrupts og programerte operatorer (7).

Doeplacement

Own sektor. inneholder 512 ord med adresser fra L-256 til L+255, hvor L er minne-adressen til den instruksjonen som blir utført. (Dette har vi tilsvarende ~~+ 128 og + 127~~ ord – 128 og + 127 ord i NORD 1.

Hvert instuksjonsord som referer seg til minnet - har en adresse-del i de bitene som er lengst til høyre.
De fleste bruker 9 bit til adressen men noen har 5 og 7 bit av formatet tildelt adressen.

nummereringen av de 16-bitene er 0 - 15 fra venstre mot høyre. (motsatt av det som er vanlig, og som er tilfelle i UCS)
En apostophe (') før et tall betyr at tallet er i oktal form.

Adresse-delen av ordet kalles Y.
Y_i er navn på bittet i adressen. Y_{15}

(W) betyr inneholdet av det ordet x hvis adresse er W
(1234) betyr inneholdet av ordet hvis adresse er 1234.

(1) betyr inneholdet av A-registret. E_n kan også bruke
(A) i samme betydning.

Y_{15} er bittet lengst til høyere i adresseordet

$(A)_{15}$ er bittet lengst til høyre i A-registeret.

Stedet der L er den minneadressen som nettop nu utføres.
L kalles * i assemblerspråket (skal være stjerne)

Y' = Intermediate adresse . absolutte adr.

E = effektiv adresse.

SEKTORER

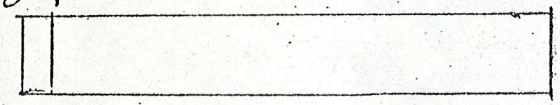
Sektor 0.
Når sektor 0 adresseres av en instruksjon er Y' = Y.

Sektor 1 Ved excute instruksjoner er Y' = Y + 1000

Sektor 2. Ved bruk av indirekte adresser er Y' = y + 2000.

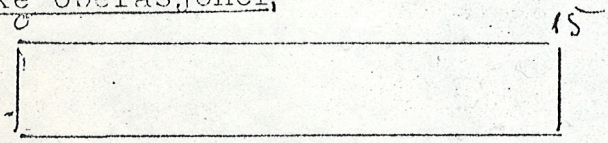
Data - formater.

Aritmetiske operasjoner



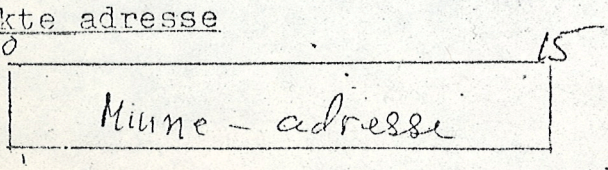
bit 0 er signbit: 0 for positive, 1 for negative.
Bit 1 til 15 gir størrelse, i 2'er komplemant for negative tall. (Samme som for NORD 1)

Logiske operasjoner



16 bit til informasjon

Indirekte adresse



Hele ordet brukes til adresse og en kunne nå 65,536 ord

kan bygges ut til

L B

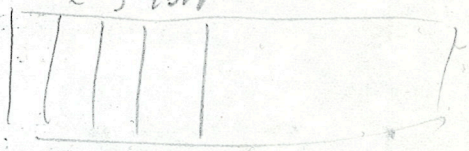
Slice eks. 3,4

3 = Lengde ordbit

4 = Bit.nr.

start

1 2 3 4 bit



C kalles operasjonskoden og er alltid bokstavene lengst til venstre i koden.

G kalles operasjonskode forlengelsen og er enten 4 eller 9 bit i lengde avhengig av typen av instruksjon.

k Er adressemoden til noen minne-referanse instruksjoner. K er alltid to bit i lengden.

B betegner bit nummer

LB er en 7-bit kode som forekommer i binær form av en instruksjon som spesifissrer 119 mulige deler av ordlengde fra 2 til 15 etterfølgende bits. betegnelsen i engelsk på en slik ord-del er slice. Programmeren skriver egentlig L,B hvor L er lengden på ordbiten og B bit-nummeret til den bitten som er lengst til venstre. I Apendix III står en tabell over 7-bit LB-kodene som brukes av ITT-1600.

M er en 2 bit kode som spesifiserer en ordbit-lengde på 1, 2 , 4 eller 8 bit i noen instruksjoner.

N representer en telling. N har en lengde på 2 bit i en Exe instruksjon , 5 bit i en skift instruksjon.

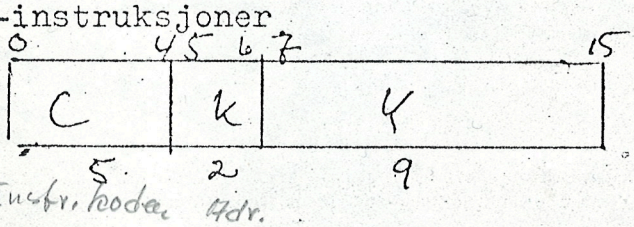
I er en 8 bit kvantitet som opptrer i "Immediate"-instruksjonene og brukes direkte som en operand (?). → argument

T er den tid en instuksjon tar i mikrosekunder.

OV betyr overlow. overflow

MINNE-referense instruksjoner.

Load og Store-instruksjoner



hvor: C er operasjonskoden, K= adresse mode.

K = 00 direkte adresse, sektor 0
k = 01 direkte sektor, own sektor (direkte relativt til P.i Nord I)
K=10 indirekte adresse, sektor 0
k = 11, indirekte indexed adresse, sektor 0
(X-registeret er med i bildet og kan ta ord for ord)

Memory og A-reg.

Her har vi da LDA,LDB LDX STA STB STX og dessuten: (forlengelse av A)
IMA Bytt inneholdet mellom hukommelse og akkumulator.
CRM Clear memory . (som tilsvarer store sero i NOED 1)
Vi har ADD og SUB men dessuten: STZ
IRS increment,replace og skip som tilsvarer MIN i NORD ^^1.
(Ta ut innholdet av cellen. Øk inneholdet med en put det deretter tilbake på plass i minnet. Skip (hopp over neste instruksjon) dersom inneholdet i cellen nu blir null)

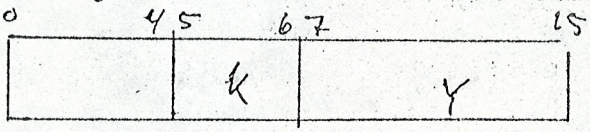
og vi har
CAS Compare & skip sammenligne innholdet i A-reg.
hvis A større enn det som er i minnet gå til neste celle.
Hvis A = det som står i minnet gå frem to plasser
Hvis inneholdet av A registeret er mindre enn den minne-cellen vi har adressert oss inn til hopp frem 3 plasser.
Sagt på en annen måte
Inneholdet av effektiv adresse og inholdet i akkumulator blir sammen liknet. Hvis inneholdet i A er større enn effektiv adresses innehold da blir neste instruksjon utført.
Hvis inneholdet i A er lik inneholdet i effektiv adresse da blir det hoppet over neste instruksjon.
Og hvis inneholde xx i A er mindre enn innholdet i minnecellen hoppes det over (skippes) to xxxx etterfølgende instruksjoner.
Xx
Vi har logisk AND, ØRx eksklusive OR
Hvis A er lik E hoppes det over neste instruksjon ellers vanlig.

Betingelsesløse hopp.

Her har k-bittet følgende mening
xx
00 = indexed adresse, egen sektor
01 direkte adresse egen sektor
k = 10 indirekte adresse sektor 2
k = 11 indirekte adresse egen sektor

Når k = 10 og Y = 0 1 eller 2 vil instruksjonen bruke inneholdet av Xregisteret, xxxxxxxxxxxxx A-registeret eller B-registeret som effektiv adresse.

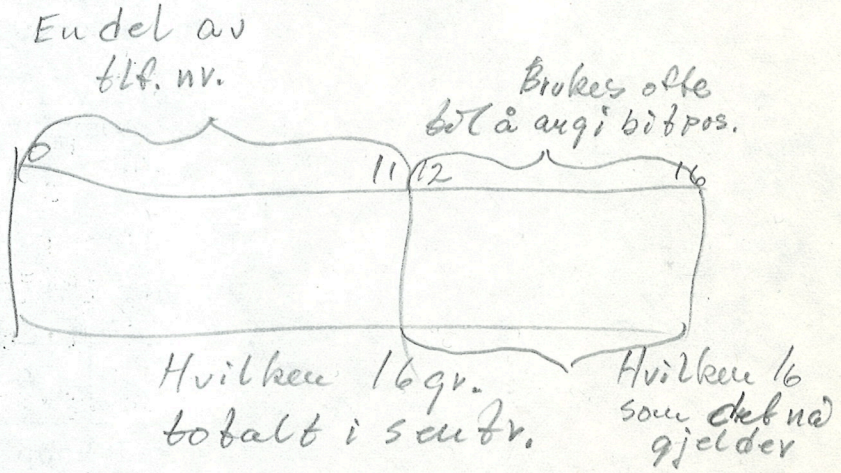
Jump JMP (tilsvarer P-registret i Nord-1)
E = PC. Den neste instuksjonen som utføres tas som effektiv adresse.



T = 1

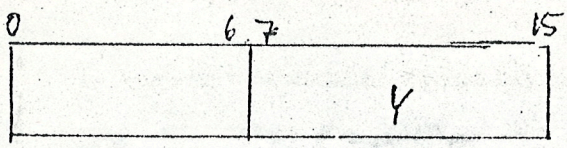
Jump & Store JST
JPL.
Denne instuksjonen brukes ved hopp til subrutiner. adresse til ordet som følger etter JST lagres i xxxxxxx minnet og neste instruksjon som skal utføres finnes i den effektive adressen. For at en ikke skal miste programmet hvis en interupt avbryter så er interupt-systemet satt ute av drift så xxxxx intil en er ferdig med JST og er kommet til neste instuksjonen . (altså etter subrutinen.
Det sted adressen tilbake til programmet lagres er i Q. Dette er i virkligheten et hardware register hvis adressex er 40. Hver gang et ord (programpeker) legres i 40 blir det først øket med 1. Etord som er plassert i 40 vil alltid når hardware pinter register PC. Hver gang det er en surutine exit (med instuksjonen DMS)blir pointer senket med 1 i verdi.

~~Formatet til disse betingelsesløse hopp er:~~



E = eventuelt innholdet i Y + innholdet i de første 12 bitene av X reg. bit posisjonen i det ordet en da er kommet frem til er angit i siste 4 bit i X.

Hopp som er betinget av at en betingelse er oppfyllt.



T = 1,25

Formatet

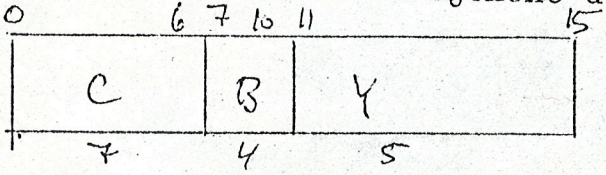
Hvis betingelsen i instruksjonen er oppfylt vil programmet hoppe til den Egne sector (own sector) adresse som beregnes fra Y. E = 1 + Y = (PC)
JZE jump if zero accumulator

JNZ jump on non-zero akkumulator
JPL jump on positive akkumulator
Jump on minus akkumulator JMI
JLZ jump on least significant bit zero og the akkumulator.
JLN jump on least significant bit nonzero on the akkumulator (HOPP DERSOM 15 bit ikke er null i A-registeret)
JIX Øk x-registeret med 1 og hopp hvis x-registerets innhold etter dette er negativt.
JDX Senk x-registerets innehold og hopp hvis x-registerets innehold er positivt.

Det er disse to instruksjonene som gjør x-registeret så velegnet til å plukke opp celle for celle i et sub-program og som har gjort x-registeret egnet til adresse-register.

BIT-INSTUKSJONER:

Vi er nå kommet frem til bit-instuksjonene de har formatet:



B er bit-nummeret og y er adressen
Når y er forskjellig fra 0-7 gir den adressen til et ord i sektor 0 mellok celle 8 og 31. Ved å addere innholdet av x-registeret til dette ordet fås den effektive adresse.
hvis y = 0 skal manupulasjonen gjøres på bitet i x-registeret
1 på B og

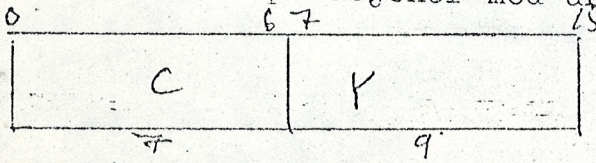
Type 1 av bit

RBI = Reset bit spesifisert i instuksjonen. (gjør lik null)

23 mulige store tabeller hvor hver enkelt bit kan nåes lettvint.

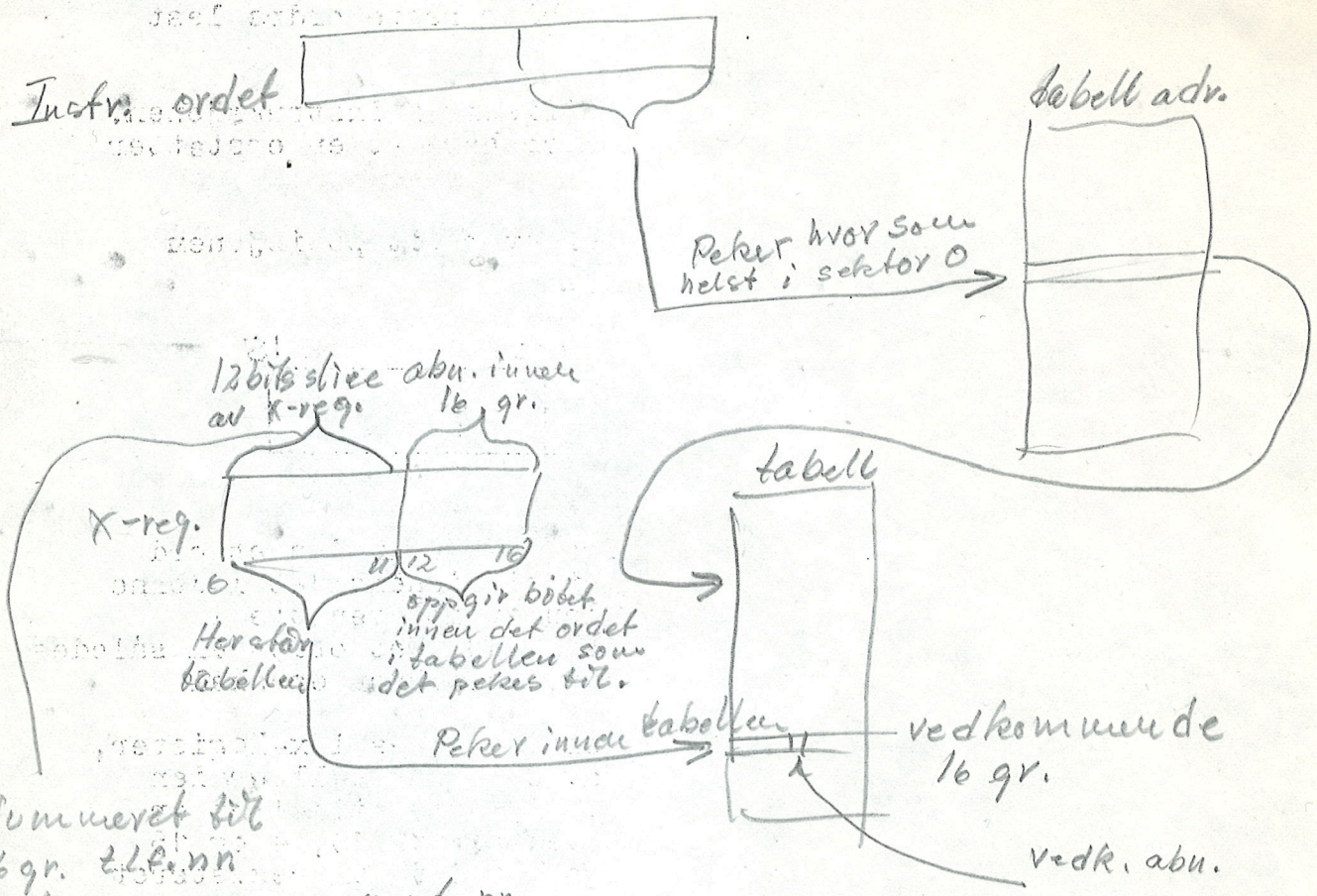
SBI = set lik 1 bittet spesifisert i instruksen

Vi får så noen bitoperasjoner med annet format:



Strykes.

Dette er operasjoner hvor bit posisisjonen er spesifisert i X-registeret.

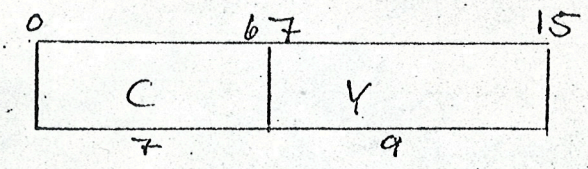

Instr. ordet
tabell adr.
Peker hvor som helst i sektor 0
12 bits slice av X-reg.
adr. innen 16 gr.
tabell
X-reg.
11
12
16
Her står tabellen
oppgir bitet innen det ordet i tabellen som det pekes til.
Peker innen tabellen
vedkommende 16 gr.
vedk. adr.
Nummerert til 16 gr. tlf.nr
Sentr. nr. = egen punkt-nr. så nær som 4 siste bit.

SRI Skip on reset bit Specified in the instruction
En hopper over en instuksjon hvis den bit som er spesifisert i instruksjonen er null i ordet som er inneholdet av den effektive adressen. HVis ikke blir neste ordre lest ogu utført.

SSI Skip on Set bit spesifisert i instruksjonen. Samme ordre som ovenfør bare at en erstatter "0" med "1".

Vi kommer så til instruksjoner hvor bit posisjonen er spesifisert i X-registeret.

Formatet er:



Når Y delen er forskjellig fra 0-7 angir den et ord i sektor 0. E_n skal så leggex til inneholdet i denne cellen 12 bitene som ligger lengst mot venstre i X registeret. (altså bit 0 til 11) Det ordet en således adresserer seg frem til er det ordet hvor en skal endre en bit i.
Hvis y = 0-7 er bit endringen å utføres i x-Register, A-register eller B-register avhengig av tallverdien av Y henholdsvis 0,1 eller 2. I dette tilfellet er er bitposisisjonen også gitt ved bit 12,13,14 og 15 i X-registeret, mens de øvrige bit i dette registeret da ikke skal tas hensyn til.

Så sandt Y ikke er null vil ingen av de fire instruksjonene av denne typen endre på inneholdet av X-registet.

RBX Reset bit specified in index-register.
Den bit hvis posisition er gitt ved de 4 siste bitene i x-registeret ~~er~~ skal gjøres lik null i det ordet som står i den effektive adressen.

SBX gjør det samme men gjør bitten lik null.

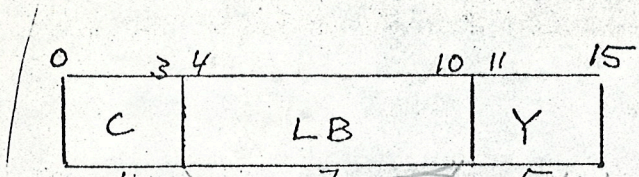
SRX Skip on reset bit specified in index-register.
E_n hopper over neste instruksjon hvis den bit i ordet som er i den effektive adresse og som har det bit nummeret de fire siste bitene i x-registeret angir, er lik null.

SSX ~~SRX~~ gjør det samme men det er "1" i bitposisisjonen som forårsaker hopp.

Vi er nå kommet til et område hvor denne maskinen er spesialisert og spesielt godt utstyrt med instruksjonsmuligheter Nemlig de såkaldte:

Slicemanupulasjoner:

Formatet er:



hvor C er operasjonskoden, LB er definisjonen av "slice" hvor mange og hvilke bits ordelen skal omfatte, og Y er adresse delen.
Som vanlig er adressedelen i Y også for 0 og 1 og 2 adresser til registre som tidligere omtalt
Det 7 bit store LB-feltet spesifiserer ord-deler på fra 2 til 15 bit. En typisk representasjon for LB er "4,6", hvor det første tallet gir lengden av ordelen og det andre gir bit-nummeret til den bit i ord-delen som ligger lengst til venstre. så 4,2 betyr en 4-bits ordlengde midt i ordet. en tabell over de 119 forskjellig 7-bits LB-koder er gitt som appendix 3.

ISI load slice defined in the instruction.
Den ord-delen som er definert i instruksjonen blir kopiert fra det effektive ordet i hukommelsen og til bakerste bitene i akkumulator-registeret. Som eksempel:
LB L,B = 4,6: Det vil si $(E)_{4,6}$ går til A_{12-15} og null

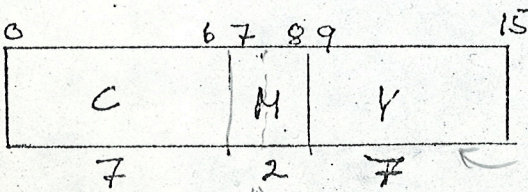
går til $(A)_{0-11}$.

JMCDEF 151
ISI Insert slice defined in the instruction
Ord-delen som står bakerst i akkumulator-registeret kopieres til spesifisert ord-del i minneordet. Akkumulator og den resterende delen av minneordet forblir uforandret.

Reset slice defined in the instruction RSI.

Ordx delen som er definert i minnet blir fyllt med nuller, resten av ordet forblir uforandret.

Vi har ordrer som spesifiserer ord-delen i X-registeret.
Formatet er:



M indikerer oxdlxxgdxxxixdxxxx ord-del-lengden i dette formatet. Formålet til denne instruksjonen er at ordlengden $L = 2^M$ slikx at
m = 00 gir 1 bit ord-del-lengde
M = 01 " 2 " "
M = 10 " 4 " "
M = 11 " 8 " "

Hver instruksjon av denne typen har altså det 2 bits M-indikasjonen på ordbit-lengden sammen med en adresse i sektor null,

hvor adressen til det første ord i en tabell vil bli funnet. Det er jo i forbindelse med store antall ord-deler med samme ord-del-lengder . De opplysninger en da ønsker er for det første ordets plassering i tabellen, dette er det x-registeret som greier opp med og så står det bare igjen å bestemme ord-delens plassering innen ordet. De første bitene i x-registeret angir ord-nummeret de siste bitene angir ord-delens plassering innen ordet. .

Dette er komplisert og sees kanskje best av et eksempel

LSX Load slice (position spesifisert i X-registet og vedkommende ord-del skal overføres til A-registeret.

$(E)_{LB}$ går inn i $(A)_{(16-L)-15}$ & 0 går inn i $(A)_{0-(15-L)}$ (Lengden)

Hvor $L = \frac{x}{M} \; 2^M$

$B = (x)2^M \bmod 16$

$E = Y + (X)2^{M-4}$ for Y mellom 8 og 127

$E = Y$ for $Y = 0-7$

Den ord-del hvis lengde gis i instruksjonen kopieres fra ordet i den effektive adresse begynnende ved den bit-posisjonen spesifisert i X-registeret og inn i bitene lengst til høyre i Akkumulator-registeret. De øvrige bit i A-registeret blir resat til % "0". Så sandt Y ikke er lik null, vil inneholdet av X-registeret ikke endre seg ved denne instruksjonen.

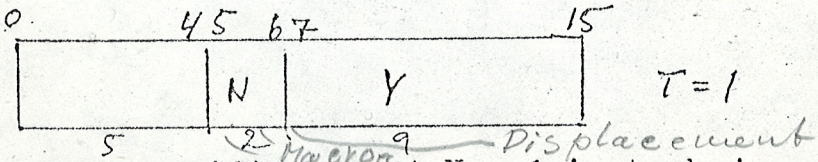
ISX Insert Slice
Samme som ovenfor men nå går ord-delen fra A-registeret til effektiv adresse.

SZX Skip if Zero slice.

Programmet hopper over én instruksjon hvis orddelen som er oppgitt i instruksjonen er lik null i ordet som er effektiv adresse. Begynnelsen til ord-delen - er gitt med den bit posisisjon som er spesifisert i x-registeret. Hivs ikke utføres neste instruksjon. Både inneholdet av ordet i effektiv-adresse og x-registerets innehold forblir uforandret.

EXE Execute

Vi er nå kommet til execute-instruksjonene.
Formatet er:



N = en 2-bit kode som spesifiserer at N + 1 instruksjoner skal utføres og en skal begynne med instruksjonen i Y'.

Hvis N = 0 , Y' = Y. I dette tilfellet vil EXE-instruksjonen føre til utførelsen av en enkelt instruksjon i lokasjonen (adressen) _ Y i sektor 0 og deretter føre kontrollen over til neste instruksjon som følger etter EXE.

Hvis N = 1,2 eller 3, y' = Y + '1000 (husk at "'" foran et tall betyr at tallet er angitt i oktal-tall) dette vil føre til at 2,3 eller 4 (N +1) instruksjoner blir utført. Den første befinner seg i adresse Y' i sektor 1, deretter retuneres kontrollen til instruksjonen som følger etter EXE i programmet.

Mens en er i ferd med å utøre instuksjoner av EXE -typen er det ikke mulig å åppnå interupt.

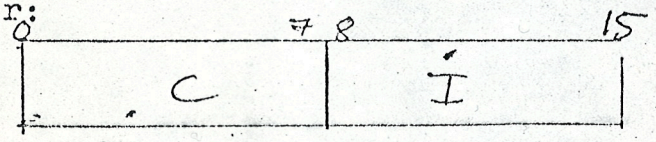
EXE med N = 0 er nyttig når instruksjonen som skal utføres er blitt beregnet.
EXE med N = 1,2 eller 3 kan spare hukommelsesplasser når samme EXE-sequence skal brukes mer enn en gang i løpet av programmet. Hvis det trengs en større sequence enn en 4 brukes heller en JST instruksjon. Den vil da være mer økonomisk med bruk av tid og minne-plasser enn flere EXE-instruksjoner.

I appendix 2 vil en gå mer detaljert inn på EXE-instruksjonen. Det finns visse restriksjoner som enn ikke kan forklare uten at bruk av instruksjonen er godt forstått.
Som hovedregel kan en dog her merke seg følgende:
Bruk ikke CAS instruksjon eller JUMP eller SKIP-instruksjon i en EXE-sequence.

Vi er nå ferdig med de instruksjonene som involverer referensen til inneholdet i en minne-celle.

Instruksjoner som ikke betjener seg av hukommelsen som referense FOR Å FÅ OPPLYSNING

Formatet er:



Hvor C er operasjonskoden og I er en 8 bit operand som skal brukes av instruksjonen.

La oss starte med:

IXP Load x-registeret = Load avg. i X
I går til $(x)_{8-15}$ % 0 går til $(X)_{0-7}$

IXM
Load 2er komplementet til operanden inn i X-register
2^{16} - I går inn i (x) (2^{16} = 65536)

IRA Load immediate i høyre halvdel av akkumulator-registeret.
I - delen av formatet (bit 8 til 15 kalles Immediate)

I går inn i $(A)_{8-15}$ 0 går inn i $(A)_{0-7}$

Tilsvarende har vi i høyre halvdel
ILA
Og til høyre halvdel av B-registeret IRB

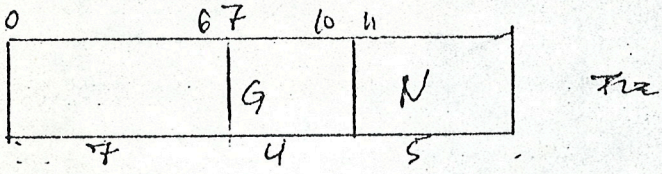
ISR subtrakt IMMEDIATE.

(A) - I går inn i (A)
og
IDR Add Immediate
(A) + I går inn i (A)

IDL Adder Immediate til venstre halvdel av A-registeret.

SKIFT-OPERASJONER.

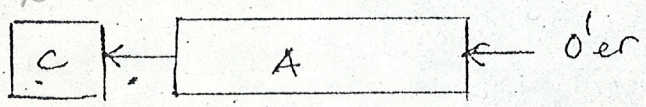
Format:



Hvor G er operasjonskodeforlengelsen og N er antall bit posisisjonen skal skiftes
Bittene i G kodes som følger:

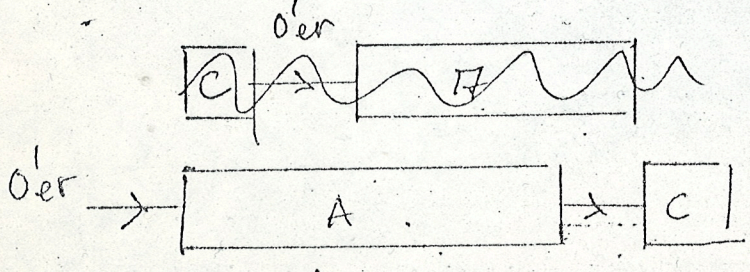
Bit Nr.	Verdi	Mening	Verdi	Mening:
7	0	Venstre	1	Høyre
8	0	kort	1	Lang
9	0	Skifte	1	Rotere
10	0	logisk	1	Arithmetisk

LGL Logisk venstre skifting av A



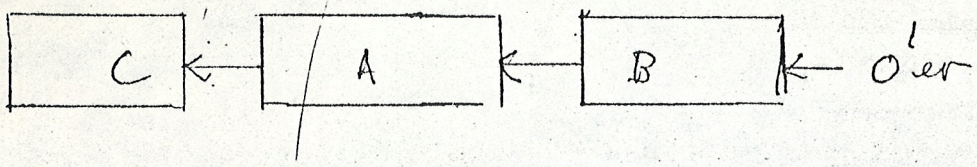
Inneholdet av A skifter N posisjoner mot venstre.
Nuller skiftes inn i de tomme posisisjoner og hvert bit bit skiftes ut av A_0 inn i C-bit.

LGR



Inneholdet av A skiftes N posisjoner mot høyre.

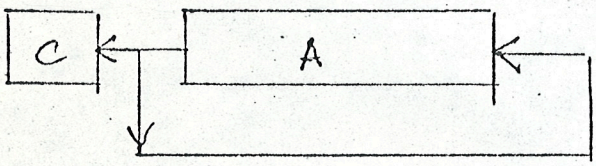
LLL lang venstre skift. T = 1,25 + 0,25 N



Inneholdet av A og B registrene former et enkelt 32 bits register, og skifter N posisjoner mot venstre.

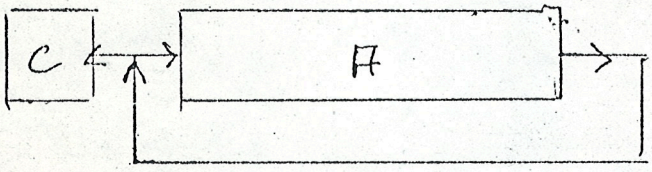
LRL Tilsvarende mot høyre

ARL logisk venstre rotering av A.

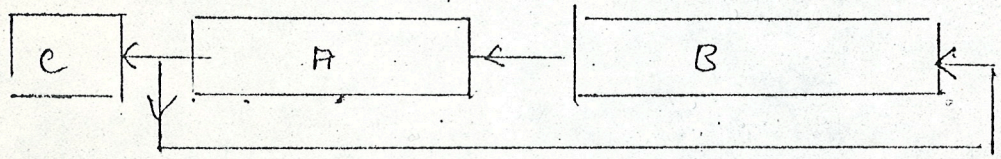


Inneholdet av A roteres N posisjoner mot venstre. Bitene skiftes ut av A_0 og entrer da i parallell både A_{15} og C-bit.

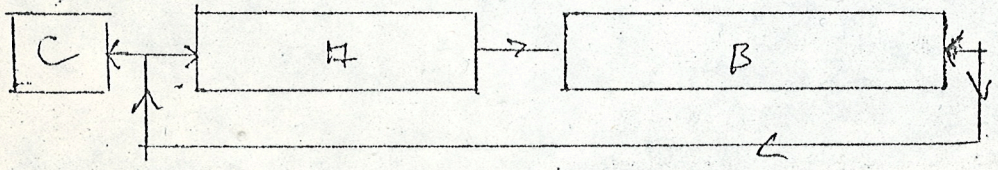
ARR logisk høyre rotering av A.



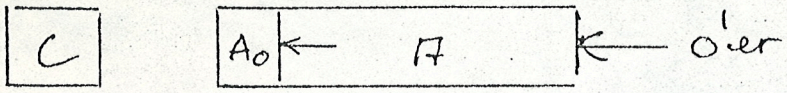
LLR lang venstre rotering



LRR Lang høyre rotering

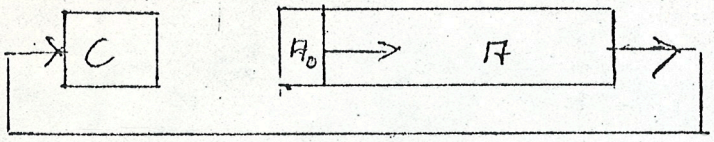


T = 1.25 + 0,25N



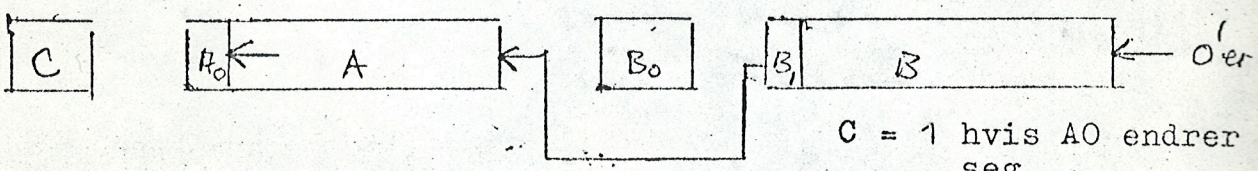
T= 1,25+0,25.

~~X~~ ALS = Aritmetisk venstre skift. Inneholdet av A skiftes N plasser mot venstre; ~~xxxx~~ C-bit settes til 1 Hvis A_0 endrer seg under skiftingen. Ellers forblir C-bit i null.



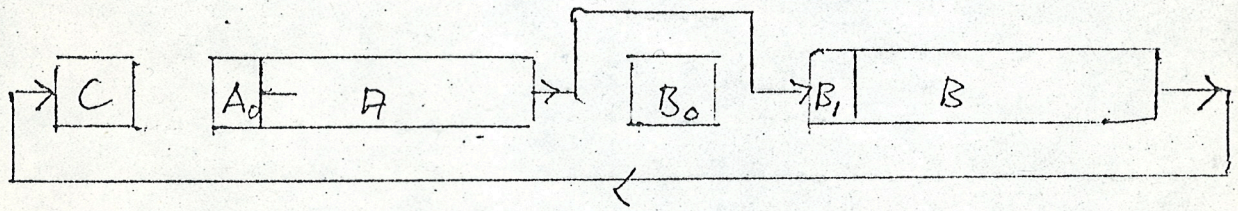
ARS Armetisk høyre skift.
Inneholdet av A skiftes N posisisjoner mot høyre, hver bit skiftes ut av A_{15} og går inn i C-bit. A_0 kopieres inn i tomme plasser.

Carry-bit



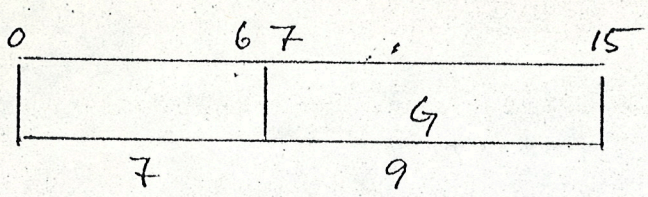
C = 1 hvis AO endrer seg
C = 0 ellers.

LLS lang aritmetisk venstre skift.
$(A)_{0-15}$ og B_{1-15} virker som et 31-bit register. B_0 endres ikke.



LRS lang artmetisk høyre skift.

SKIP- INSTRUKSJONER:



Hvis betingelsen som G angir er oppfylt skal en hoppe over neste instruksjon. Ellers skal den utføres

SS1 Skip hvis A_0 er "1". ? Det står egentlig
Skip if "Sense switch 1 " er 1.
G = 401

SS2 G=402 Skip if sense switch 2 er 1
SS3 G= 404 " 3 1
SS4 G=410 " 4 " 1
SSS 417 noen "sense switches er 1
SSR1 001 " 1 0
SSR2, SSr3, og SSR4 på samme måte.

SSR skip if all sense switches er 0
SAO skip hvis A bare har 1-ere
SNO " " ikke har bare 1'ere
SRC " " c-bit er null
SSC dito når c-bit er en
SPS Skip hvis paritetsfeil flipflop er 1
SPN " " 0

OPERATE INSTRUKSJONER MED A-REGISTERET ALENE.

Samme format som Skip men G er operasjonskodeforlengelsen.

CRA Fyll A-registeret med nuller
SOA " 1'ere
SSP Set sign-bit lik plus
SSM Set sign-bit lik minus
RLB Gjør bit 15 i A lik null
~~SK~~ SLB " " "1".
CHS Komplementer (snu ~~xxxxxxx~~ - gjør 1 til nul og 0 til 1) sign bit til A-registeret.
CLB dito med siste bittet (bit 15)
CMA Komplementer A-registeret (1'er komplement)
TCA " " (2'er komplement)
AOA Legg 1 til inneholdet i A-registeret. Hvis overflow Blir C = 1.
IAB Bytt om A og B
CRB Fyll B-registeret med nuller
RAB Fyll både A og B med nuller
RAL Fyll A, b registerene og C-bit med null.
RAC Nuller i A -reg. samt C-bit
SAC Set A-reg og C-bit til bare 1'ere
RBC. Nuller i B-registeret og C-bit
SCB "1" i C-bit
CSA Kopier Signbit til C-bit og gjør signbitet i A positivt det vil si $(A)_0 = 0$

CCS Kopier C-bit som Signbit og gjør C-bit lik null
ACA Adder C-bit til A-registeret

KONTROLL- INSTRUKSJONER

HLT Stop. Datamaskinen står inntil en trykker START-knappen på operatørpanelet. (Wait - på NORD-1. bare at den gir andre anledning til å komme inn ved interuptkjøring)

NOP Datamaskinen gjør den neste sequensiale instruksjonen uten å gjøre noe annet.

ENB Slå på interupt
INH Interuptsystemet avslått inntil instruksjonen ENB.
RMP Set minne paritetsflip-flop til "0".
DPL (Q) - 1 går til Q. Adressen til Q er '40,
Denne instruksjonen senker med én nåværende adresse til subrutine retur-adresse

DMS Dismiss
((Q)) går mot (pC) adressen til er '40
(Q) - 1 går mot (Q)
Denne instruksjonen brukes for å retunere fra en subrutine. Den får returadressen inn programpekeren å senker med 1 inneholdet av det registeret som holder telling med subprogrammet.

FFO Finn første 1.
Instruksjonen finner det bit i A-registeret som er lengst til venstre og gjør det lik null. Instruksjonen setter inn nummeret på det bittet i de 4 siste bittene i X-registeret efter å ha skiftet inneholdet av X-registeret 4 plasser mot venstre. Ingen annen endring gjøres i A-registeret enn å skifte første 1 til null.

EPM slå på minne-beskyttelsen.

DPM Slå av minnebeskyttelsen. Må eventuelt slås på ved EPM.

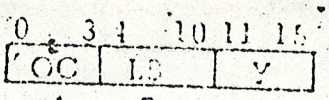
APPENDIX 3

Seven-Bit LB Code for Slice Instructions

L, Length of Slice	B, Number of Leftmost Bit														
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
2	010	021	032	043	054	065	076	107	067	056	045	034	023	012	001
3	020	031	042	053	064	075	106	117	057	046	035	024	013	002	
4	030	041	052	063	074	105	116	127	047	036	025	014	003		
5	040	051	062	073	104	115	126	137	037	026	015	004			
6	050	061	072	103	114	125	136	147	027	016	005				
7	060	071	102	113	124	135	146	157	017	006					
8	070	101	112	123	134	145	156	167	007						
9	100	111	122	133	144	155	166	177							
10	110	121	132	143	154	165	176								
11	120	131	142	153	164	175									
12	130	141	152	163	174										
13	140	151	162	173											
14	150	161	172												
15	160	171													

SLICE MANIPULATION (Defined in Instr.) (p.24)

SIMBAL-16 Format : C3A Machine Format : 0 3 4 10 11 15



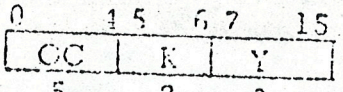
App. 2 - P.5

Notation : L : Slice length, B = leftmost bit no of slice, E = Y for Y ≤ 7
E = (Y) + (X), if 8 ≤ Y ≤ 31
A1:Y, A2 : L, A3 : B

ISI	4	05	Insert Slice	$(A)_{(16-L)-15} \longrightarrow E_{LB}$
LSI	4	06	Load Slice	$(E)_{LB} \longrightarrow (A)_{(16-L)-15}$, $0 \longrightarrow (A)_{0-(15-L)}$
RSI	4	07	Reset Slice	$0 \longrightarrow (E)_{LB}$

LOAD-STORE : ARITHMETIC-LOGICAL (p.21).

SIMBAL-16 Format : OA Machine Format : 0 4 5 6 7 15

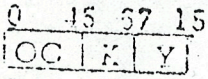


Notation : K = 00, E = Y, T = T K = 10, E = (Y), T = T + 1
K = 01, E = L ± Y, T = T K = 11, E = (Y) + (X), T = T + 1
K is not written by the programmer. If used, indirection (*) and/or indexing (,X) must by specified in the A field, along with Y.

LDX	2	100---	Load Index	(E) → (X)
STX	2	104---	Store Index	(X) → (E)
LDA	2	110---	Load Acc.	(E) → (A)
STA	2	114---	Store Acc.	(A) → (E)
LDB	2	120---	Load B-register	(E) → (B)
STB	2	124---	Store B-register	(B) → (E)
ADD	2	130---	Add	(A)+(E) → (A). If OV, 1 → (C), if no OV 0 → (C)
SUB	2	134---	Subtract	(A)-(E) → (A). If OV, 1 → (C), if no OV 0 → (C)
IRS	2	144---	Incr. Memory & Skip	(E)+1 → (E) Then if (E)=0 go to L+2, otherwise go to L+1
ANA	2	150---	And to A	(A) ∧ (E) → (A)
IMA	3	154---	Interchange Memory & A	(E) ⟷ (A)
ERA	2	160---	Exclusive OR to A	(A) v (E) → (A), then if (A)=0 go to L+2, otherwise go to L+1
CRM	2	164---	Clear Memory	0 → (E)
CAS	2	170---	Compare and Skip	If (A) > (E) go to L+1, if (A)=(E) go to L+2 If (A) < (E) go to L+3

UNCONDITIONAL JUMPS (p.21)

SIMBAL-16 Format : CA Machine Format : 0 4 5 6 7 15



Notation : K = 00, E = L ± Y + (X), T = T + 0,25 K=10, E=(Y+'2000) if Y ≥ 3
E=(Y) if Y ≤ Z, T = T + 1
K = 11, E = (L ± Y), T = T + 1
K is not written by the programmer. If used, indirection (*) or indexing (,X) must be specified in the A field, along with Y.

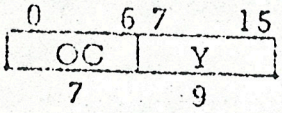
JMP	1	1[illegible]---	Jump	(E) → (PC)
JST	3	0[illegible]---	Jump and Store	(Q)+1 → (Q), (PC)+1 → ((Q)), E → (PC)

CONDITIONAL JUMPS (cont'd)

JMI	1,25	023	If Minus Acc.	If $(A)_0 = 1$, $L \pm Y \rightarrow (PC)$, otherwise go to L + 1
JLZ	1,25	024	If Last Bit Zero	If $(A)_{15} = 0$, $L \pm Y \rightarrow (PC)$, otherwise go to L + 1
JLN	1,25	025	If Last Bit Non Zero	If $(A)_{15} = 1$, $L \pm Y \rightarrow (PC)$, otherwise go to L + 1
JIX	1,25	026	If X - after incr.	$(X) + 1 \rightarrow (X)$, then if $(X)_0 = 1$, $L \pm Y \rightarrow (PC)$, otherwise go to L + 1
JDX	1,25	027	If X + after decr.	$(X) - 1 \rightarrow (X)$, then if $(X)_0 = 0$, $L \pm Y \rightarrow (PC)$, otherwise go to L + 1

BIT MANIPULATION (Bit N^0 in X) (p. 23)

SIMBAL-16 Format : OA Machine Format :

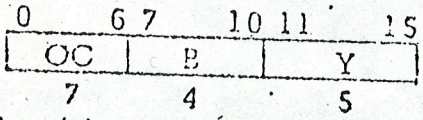


Notation : B = (X) 12 - 15, E = Y if Y ≤ 7, E = (Y) + $(X)_0$ - 11 if Y ≥ 8

RBX	4	030	Reset Bit X	$0 \rightarrow (E)_B$
SBX	4	031	Set Bit X	$1 \rightarrow (E)_B$
SRX	3	032	Skip if Bit Reset X	If $(E)_B = 0$, go to L+2, otherwise go to L+1
SSX	3	033	Skip if Bit Set X	If $(E)_B = 1$, go to L+2, otherwise go to L+1

BIT MANIPULATION (Bit N^0 in Instr.) (p.23)

SIMBAL-16 Format : O2A Machine Format :

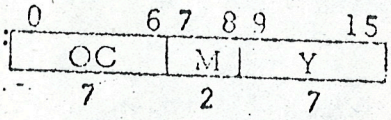


Notation : B = bit N^0, E = Y if Y ≤ 7, E = (Y) + (X) if 8 ≤ Y ≤ 31
A1 : Y, A2 : B

RBI	4	034	Reset Bit	$0 \rightarrow (E)_B$
SBI	4	035	Set Bit	$1 \rightarrow (E)_B$
SRI	4	036	Skip on Reset Bit	If $(E)_B = 0$, go to L+2, otherwise go to L+1
SSI	4	037	Skip on Set Bit	If $(E)_B = 1$, go to L+2, otherwise go to L+1

SLICE MANIPULATION (Position in X) (p.24)

SIMBAL-16 Format : O2A Machine Format :

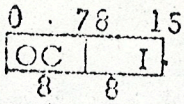


Notation : Slice length = 2^M, B = (X) 2^M mod 16, E = Y for Y ≤ 7,
E = (Y) + (X) 2^{M-4}, for 8 ≤ Y ≤ 127
A1 : Y A2 : L, Slice length.

LSX	4	044	Load Slice X	$(E)_{LB} \rightarrow (A)_{(16-L)-15}$, $0 \rightarrow (A)_{0-(15-L)}$
ISX	4	045	Insert Slice X	$(A)_{(16-L)-15} \rightarrow (E)_{LB}$
SZX	4	046	Skip if Zero Slice X	If $(E)_{LB} = 0$, go to L+2, otherwise go to L+1
RSX	4	047	Reset Slice X	$0 \rightarrow (E)_{LB}$

IMMEDIATE OPERAND (p. 23)

SIMBAL-16 Format : CA Machine Format :

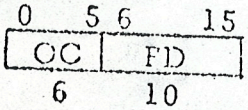


I : 8-Bit Operand

IPX	1	0040--	Imm.	Load Positive in X	$I \rightarrow (X)_{8-15}$, $0 \rightarrow (X)_{0-7}$
INX	1	0044--	"	" Negative in X	$2^{16}-I \rightarrow (X)$,
ILR	1	0050--	"	" in Right of A	$I \rightarrow (A)_{8-15}$, $0 \rightarrow (A)_{0-7}$
ILL	1	0054--	"	" " Left of A	$I \rightarrow (A)_{0-7}$, $0 \rightarrow (A)_{8-15}$
ILB	1	0060--	"	" " Right of B	$I \rightarrow (B)_{8-15}$, $0 \rightarrow (B)_{0-7}$
ISR	1	0064--	"	Subtract from A	$(A) - I \rightarrow (A)$, If CV, $1 \rightarrow (C)$, otherwise $0 \rightarrow (C)$
IAR	1	0070--	"	Add to Right of A	$(A) + 1 \rightarrow (A)$, If CV, $1 \rightarrow (C)$, otherwise $0 \rightarrow (C)$
IAL	1	0074--	"	" to Left of A	$(A)_{0-7}+1 \rightarrow (A)_{0-7}$, If CV, $1 \rightarrow (C)$, otherwise $0 \rightarrow (C)$

INPUT/OUTPUT INSTRUCTIONS (p. 27)

SIMBAL-16 Format : OA Machine Format :



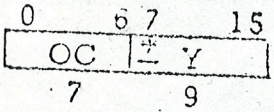
F Function, D Device

T is given for peripherals connected to the DC bus, for peripherals connected to the AC bus, add either 0,25 or 0,5.

OTA	2,25	010-	Output from A	If $(Ready)_{FD}=0$, go to L+1, no output. If $(Ready)_{FD}=1$ $(A) \rightarrow (OTB)_{FD}$ then go to L+2. OTB Output Bus
SMK	2,25	010-	Set Mask	$(FD+'54)$, $(A) \rightarrow (OTB)_{FD}$
INA	2,25	012-	Input in A	If $(Ready)_{FD}=0$, go to L+1, no input. If $(Ready)_{FD}=1$ and $(FD)_6=1$, $(INB)_{FD} \rightarrow (A)$, then go to L + 2, If $(Ready)_{FD} = 1$ and $(FD)_6 = 0$, $(INB)_{FD} \vee (A) \rightarrow (A)$, then go to L + 2.
RMK	2,25	012-	Read Mask	$(FD + '54)$, $(INB)_{FD} \rightarrow (A)$, INB Input Bus.
OCP	2,25	014-	Output Command Pulse	$(FD)_{6-15} \rightarrow (ADB)_{FD}$, ADB address Bus.
SKS	2,25	016-	Skip on Condition Set	If $(Condition)_{FD} = 0$, go to L+1, if $(Condition)_{FD}=1$ go to L + 2

CONDITIONAL JUMPS (p. 23)

SIMBAL-16 Format : OA Machine Format :



			Jump	
JZE	1,25	020	If Zero Acc.	If $(A)=0$, $L \pm Y \rightarrow (PC)$, otherwise go to L+1
JNZ	1,25	021	If Non Zero Acc.	If $(A) \neq 0$, $L \pm Y \rightarrow (PC)$, otherwise go to L+1
JPL	1,25	022	If Plus Acc.	If $(A)_0=0$, $L \pm Y \rightarrow (PC)$, otherwise go to L+1