3400 3600 3800

COMPUTER SYSTEMS COMPASS

TRAINING MANUAL
VOLUME II

CONTROL DATA
CORPORATION

RECORD of REVISIONS

REVISION	NOTES
A (3-1-66)	Final Edition Released.
B (4-19-67)	Publications Change Order CA16274. Manual enlarged from one to three volumes, replacing
	3600 COMPASS Programming Guide, Pub. No. 60166700, which becomes Volume 2 of the new
	3-volume set. Miscellaneous corrections to Volume 2, pages 1-1, 1-21, 1-26, 2-10, 2-14, 7-3,
	7-10, 11-5, 12-26, 12-30, 13-25, 15-13, 15-18, 15-22, 15-30, 15-42, 16-6, 17-2, 17-3, 17-6,
	18-2, 18-6, 18-10, 18-14, 18-18, and 18-22. This edition obsoletes all previous editions.

FORM CA230 REV. 1-67

Pub No. 60166700
© 1966, 1967
by Control Data Corporation
Printed in United States of America

Address comments concerning this manual to:

Control Data Corporation
Marketing Training Department
8100 - 34th Avenue South
Minneapolis, Minnesota 55440

FOREWORD

This manual is intended as a guide in learning how to program the upper 3000 computer systems. It includes a hardware concept of the systems, the use of the COMPASS programming language, and the use of the SCOPE monitor. Step-by-step example problems, with and without given solutions, are included to develop the capability of using the language.

This manual is a major revision to and a replacement for the 3600 Computer System COMPASS Programming Guide and retains the same publication number. It is now expanded to three volumes.

Volume I

This volume consists of three sections. The first section deals with the introduction to the systems. The second section deals with the central processor. The third section deals with problem-oriented exercises in which random instructions are picked to solve problems.

Volume II

This volume consists of one section. The instruction repertoire is divided into groups. Groups 1-18 are hardware instruction groups, and groups 19-25 are pseudo instruction groups. Each group is followed by explanations of new concepts and problems designed to use instructions from the group.

Volume II

This volume consists of two sections. The first section deals with the SCOPE system. It shows how to run jobs under the system and explains new concepts such as overlay processing and library preparation. The second section contains several computer output listings obtained as a result of running the example problems under SCOPE.

REFERENCES

3400 SCOPE / COMPASS Reference Manual	Pub. No. 60057800
3400/3600/3800 Instant TAPE SCOPE	Pub. No. 60059000
3600 Computer System Reference Manual	Pub. No. 60021300
3600 COMPASS Reference Manual	Pub. No. 60052500
3600 Instant COMPASS	Pub. No. 60056500
SCOPE Reference Manual	Pub. No. 60053300
3000 Series Peripheral Equipment Reference Manual	Pub. No. 60108800

CONTENTS

VOLUME II

* Denotes 48-bit instruction

CONTENTS (Cont'd)

* Denotes 48-bit instruction

CONTENTS (Cont'd)

* Denotes 48-bit instruction

CONTENTS (Cont'd)

* Denotes 48-bit instruction

CONTENTS (Cont'd)

* Denotes 48-bit instruction

CONTENTS (Cont'd)

GROUP 1

FULL WORD TRANSMISSION

GROUP 1

FULL WORD TRANSMISSION

1.	Load A	LDA
2.	Load Q	LDQ
3.	Load A Complement	LAC
4.	Load Q Complement	LQC
5.	Store A	STA
6.	Store Q	STQ
7.	Transmit	XMIT

This group of instructions transmits one or more 48-bit words from one location to another.

The first two instructions transmit a word from a storage address to a register. They are called LOAD instructions. One storage cycle is required for each.

The second two instructions do the same thing; however, the complement of the memory word is transmitted from a storage address to a register. These are called LOAD COMPLEMENT instructions. Again, one storage cycle is required for each.

The next two instructions, called the STORE instructions, transmit words from a register to a storage address. One storage cycle is required for each instruction.

The last instruction is a memory-to-memory transfer. One or more words may be transferred.

Normally, only the destination, not the source, is affected by the transmission.

The LOAD A Instruction

The LOAD A instruction is an instruction that transmits a 48-bit operand from an 18-bit storage address to the A register. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the A register are replaced by the transmitted operand.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

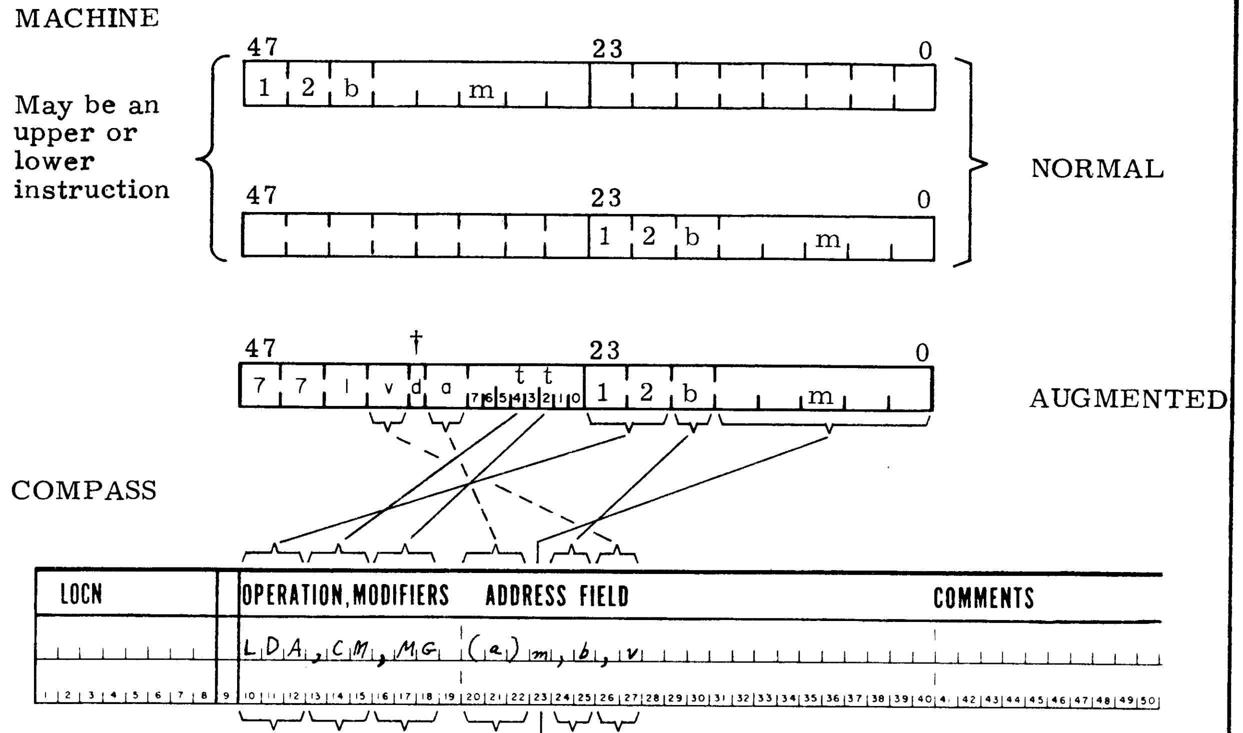
If CM is specified by the programmer, the complement of the operand is transmitted to the A register; i.e., the operand is read from memory, complemented, and then transferred to the A register.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is transmitted to the A register; i.e., the operand is read from memory, tested to obtain its absolute value, and then transferred to the A register.

FORMAT

LOAD A

LDA



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

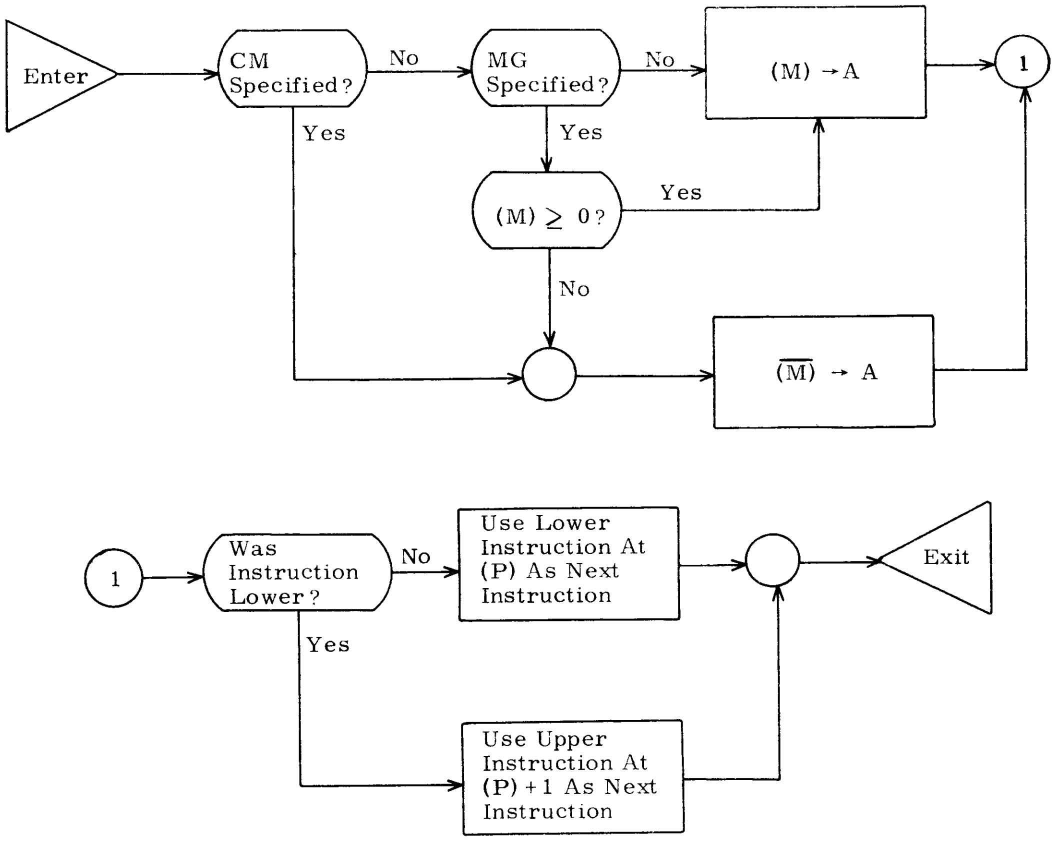
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

LDA

LOAD A FORMAT: LDA, CM, MG (a)m, b, v $M = m + (B^b) + (V^v)$

FLOWCHART



DESCRIPTION

LDA	1) (M) → A
	2) Contents of M remain unchanged
LDA, CM	1) $\overline{(M)} \rightarrow A$
	2) Contents of M remain unchanged
LDA, MG	1) $\lvert(M)\rvert \rightarrow A$
	2) Contents of M remain unchanged

PROBLEM:

Load A from address PRIME.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA	PRIME	

PROBLEM:

Load A from address PRIME modified by Index Register 3.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA	PRIME,3	

PROBLEM:

Load A with the absolute value from address NUMBER.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA,MG	NUMBER	

PROBLEM:

Load A from address NUMBER from the bank in which the LDA resides.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA	(*)NUMBER	

The LOAD Q Instruction

The LOAD Q instruction is an instruction that transmits a 48-bit operand from an 18-bit storage address to the Q register. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the Q register are replaced by the transmitted operand.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is transmitted to the Q register; i.e., the operand is read from memory, complemented, and then transferred to the Q register.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is transmitted to the Q register; i.e., the operand is read from memory, tested to obtain its absolute value, and then transferred to the Q register.

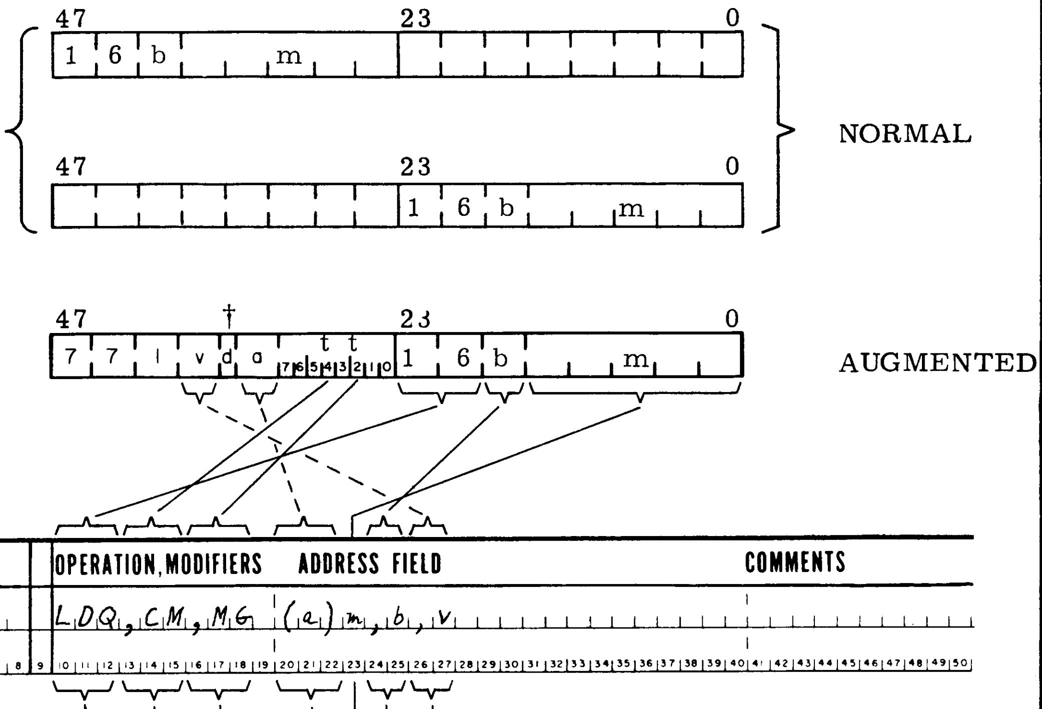
LOAD Q

LDQ

F
O
R
M
A
T

MACHINE

May be an upper or lower instruction



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

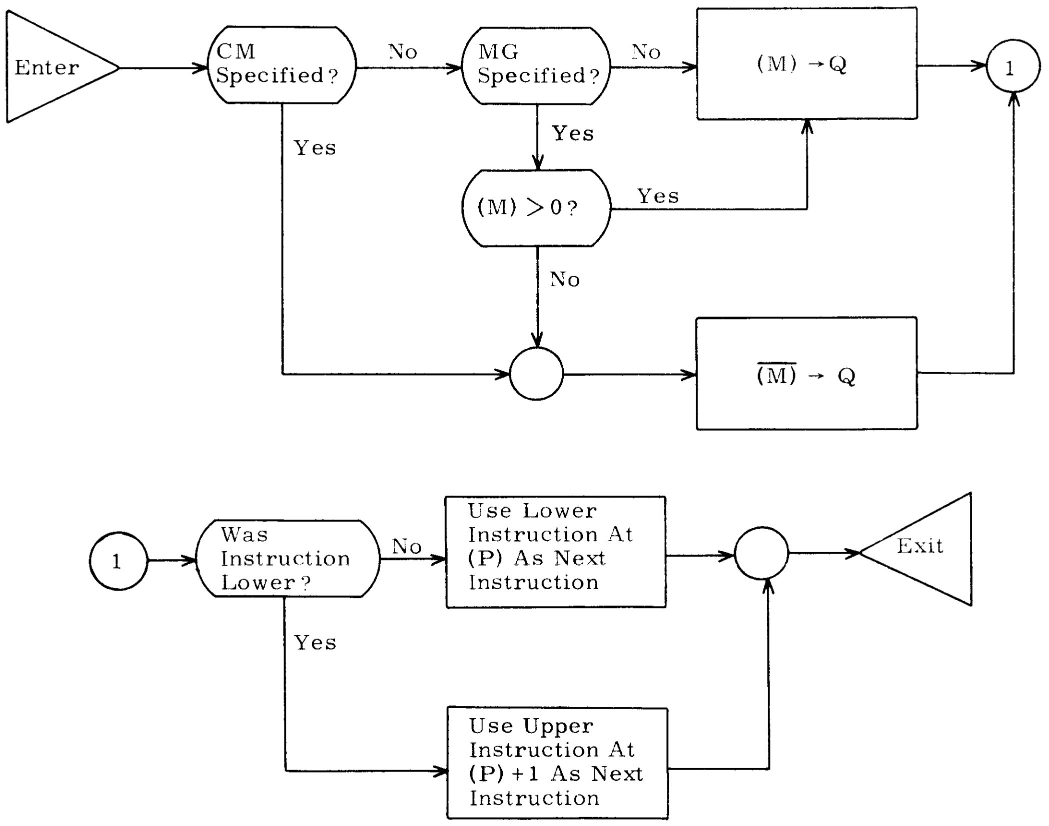
*, transmits Instruction Bank. Register to Operand Bank Register.

† d bit not programmable

LDQ

LOAD Q FORMAT LDQ, CM, MG (a)m,b,v $M = m + (B^b) + (V^v)$

FLOWCHART



DESCRIPTION

LDQ	1) $(M) \to Q$
	2) Contents of M remain unchanged
LDQ, CM	1) $\overline{(M)} \to Q$
	2) Contents of M remain unchanged
LDQ, MG	1) $\lvert(M)\rvert \to Q$
	2) Contents of M remain unchanged

PROBLEM:

Load Q from address CONST +5.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LDQ	CØNST+5	

PROBLEM:

Load Q from address CONST modified by Index Registers 2 and 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LDQ	CØNST,2,3	

PROBLEM:

Load Q with the absolute value from address ROOT.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LDQ,MG	RØØT	

PROBLEM:

Load Q from address CONSTOR from the bank in which CONSTOR resides.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LDQ	($) CØNSTØR	

The LOAD A COMPLEMENT Instruction

The LOAD A COMPLEMENT instruction is an instruction that transmits the complement of a 48-bit operand from an 18-bit storage address to the A register. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the A register are replaced by the transmitted operand.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the complement of the operand is transmitted to the A register. Effectively it would be a LOAD A instruction.

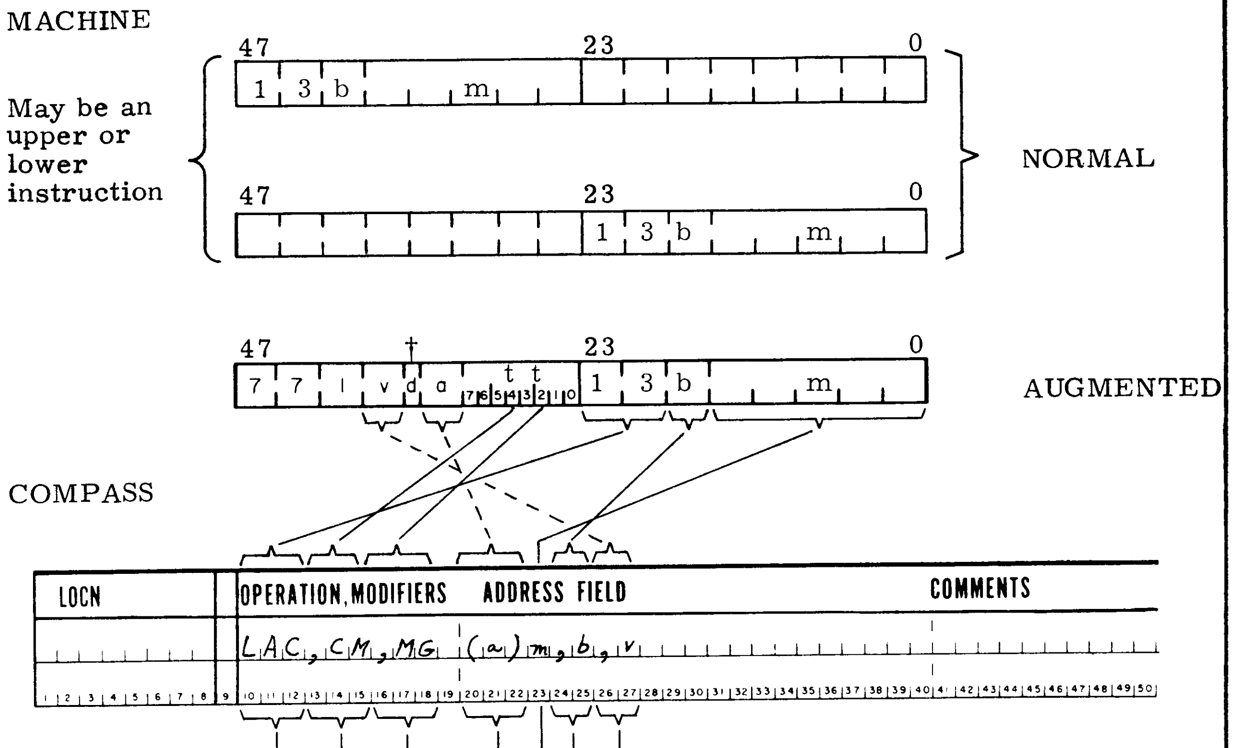
If MG is specified by the programmer, the negative value of the operand is transmitted to the A register; i.e., the operand is read from memory, tested to obtain its absolute value, complemented, and then transferred to the A register.

The CM and MG modifiers are seldom used for this instruction.

F O R M A T

LOAD A COMPLEMENT

LAC



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

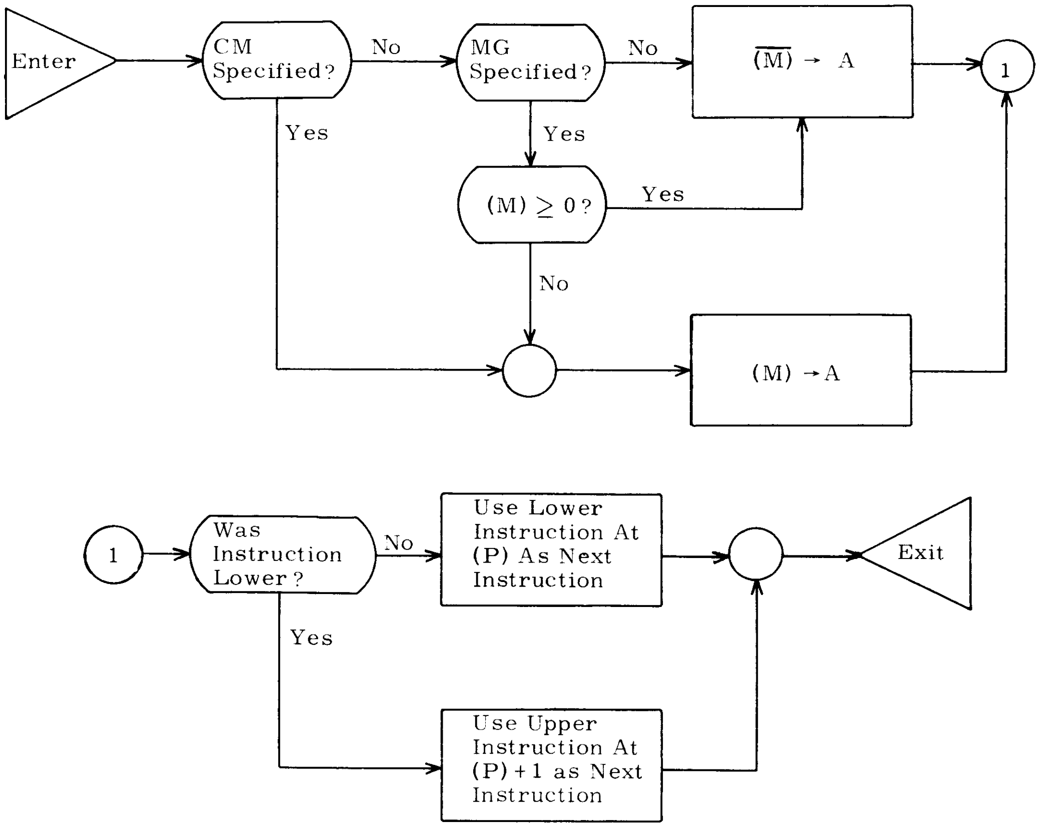
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

LAC

LOAD A COMPLEMENT FORMAT: LAC,CM,MG (a) m,b,v $M=m + (B^b) + (V^v)$

F L O W C H A R T


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
$\overline{(M)} \rightarrow A$
1
$(M) \geq 0$?
Yes
No
$(M) \rightarrow A$
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 as Next Instruction
Exit

D E S C R I P T I O N

LAC	1) $\overline{(M)} \rightarrow A$
	2) Contents of M remain unchanged
LAC, CM	1) $\overline{\overline{(M)}} \rightarrow A$
	2) Contents of M remain unchanged
LAC, MG	1) Neg. $(M) \rightarrow A$
	2) Contents of M remain unchanged

E
X
A
M
P
L
E
S

PROBLEM:

Load A with the complement of the operand at address BILKO.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LAC	BILKØ	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Load A with the complement of the operand at address BILKO modified by Index Register 4.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LAC	BILKØ,4	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Load A with the complement of an operand at an address specified in Index Register 5.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LAC	,5	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Load A with the complement of an operand at address BAKER modified by Index Registers 1 and 2.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LAC	BAKER,1,2	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The LOAD Q COMPLEMENT Instruction

The LOAD Q COMPLEMENT instruction is an instruction that transmits the complement of a 48-bit operand from an 18-bit storage address to the Q register. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the Q register are replaced by the transmitted operand.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

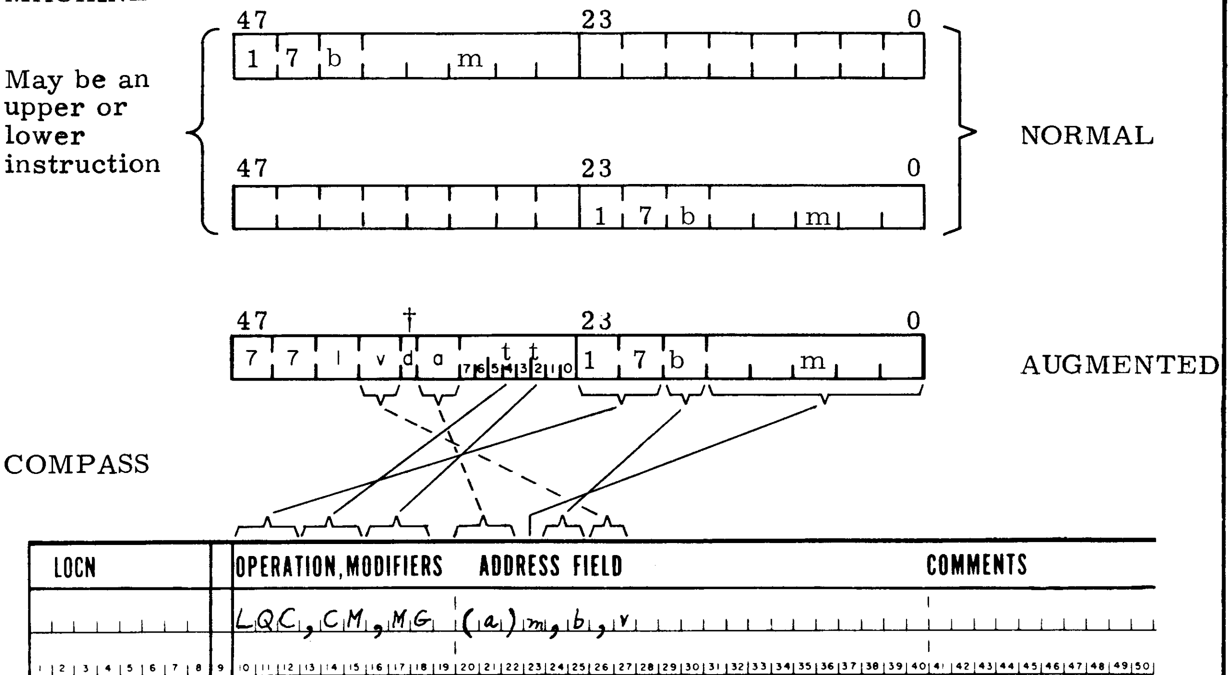
If CM is specified by the programmer, the complement of the complement of the operand is transmitted to the Q register. Effectively it would be a LOAD Q instruction.

If MG is specified by the programmer, the negative value of the operand is transmitted to the Q register; i.e., the operand is read from memory, tested to obtain its absolute value, complemented, and then transferred to the Q register.

The CM and MG modifiers are seldom used for this instruction.

FORMAT

MACHINE



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

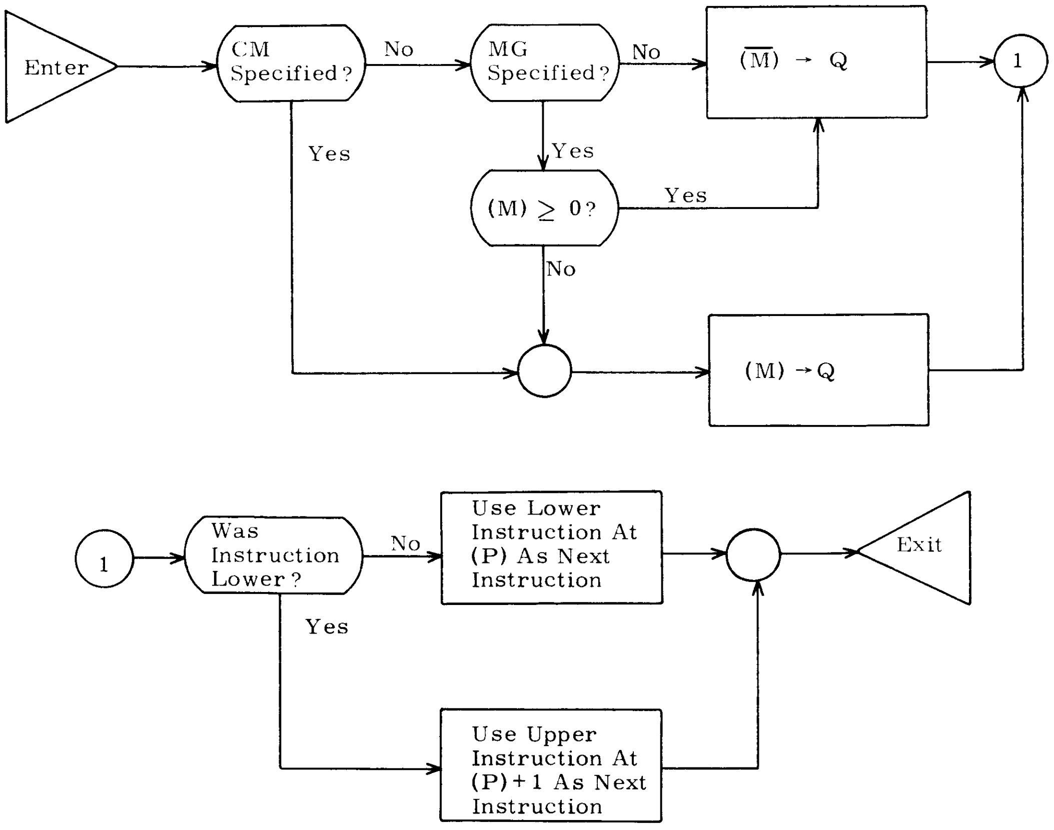
$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register

† d bit not programmable

FLOWCHART

LOAD Q COMPLEMENT FORMAT: LQC, CM, MG (a) m,b,v $M = m + (B^b) + (V^v)$



Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
$(\overline{M}) \rightarrow Q$
1
$(M) \geq 0$?
Yes
No
$(M) \rightarrow Q$
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

LQC	1) $(\overline{M}) \rightarrow Q$
	2) Contents of M remain unchanged
LQC, CM	1) $(\overline{\overline{M}}) \rightarrow Q$
	2) Contents of M remain unchanged
LQC, MG	1) Neg. $(M) \rightarrow Q$
	2) Contents of M remain unchanged

E
X
A
M
P
L
E
S

PROBLEM:

Load Q with the complement of the operand at an address specified by Index Register 4.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LQC	,4	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Load Q with the complement of an operand at address HOMER, but indirectly.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LQC	HØMER,7	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Load Q with the complement of an operand at address SNODGRASS from the bank in which the LQC resides.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LQC	(*)SNØDGRASS	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Load Q with the complement of an operand at address SNODGRASS - 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LQC	SNØDGRASS-3	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The STORE A Instruction

The STORE A instruction is an instruction that transmits a 48-bit operand from the A register to an 18-bit storage address. One memory reference is made.

The operation leaves the contents of the A register unchanged unless the modifier CL is specified (see below). The contents of A replace the contents of the storage address.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

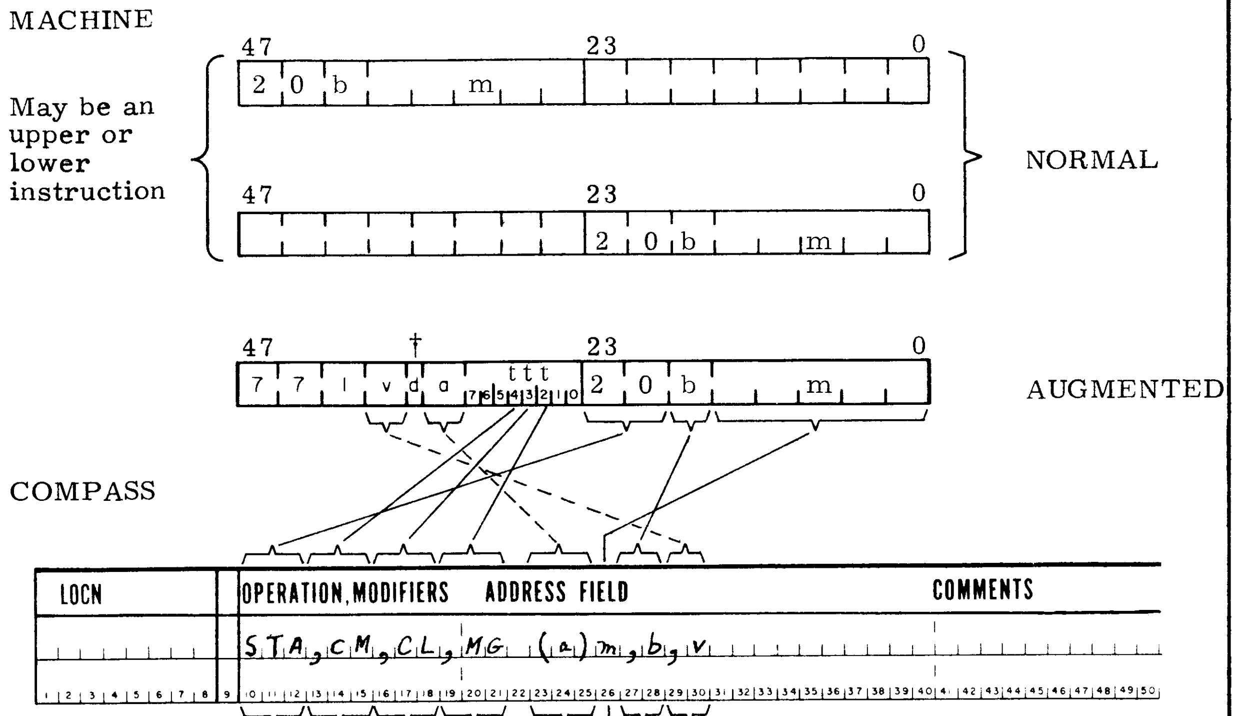
If CM is specified by the programmer, the complement of the contents of A is transmitted to the storage address.

If MG is specified by the programmer, the magnitude (absolute value) of the contents of A is transmitted to the storage address.

If CL is specified by the programmer, the contents of A are cleared <u>after</u> the operand has been transmitted to the storage address.

STORE A STA

FORMAT


MACHINE
May be an upper or lower instruction
47
23
0
2 0 b m
NORMAL
47
23
0
2 0 b m
47
†
23
0
7 7 l v d a t t t 7 6 5 4 3 2 1 0 2 0 b m
AUGMENTED
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
STA, CM, CL, MG (a) m, b, v

Function code

Optional, transmit complement of operand

Optional, clear (A) after transmission of operand

Optional, transmit magnitude of operand

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number of Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

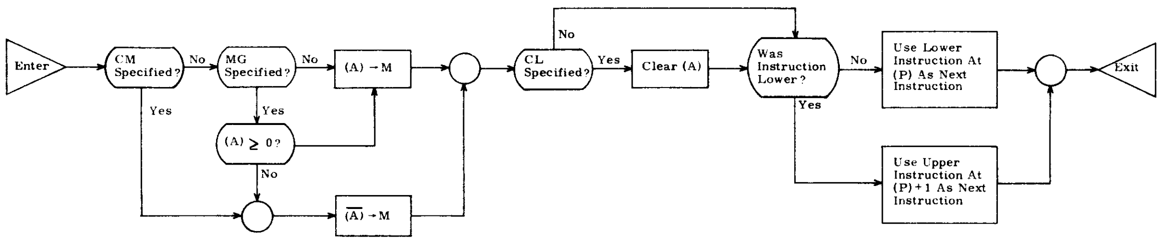
† d bit not programmable

FLOWCHART

STORE A

FORMAT: STA, CM, CL, MG (a) m, b, v *

$M = m + (B^b) + (V^v)$



DESCRIPTION

STA	1) $(A) \rightarrow M$
	2) Contents of A remain unchanged
STA, CM	1) $\overline{(A)} \rightarrow M$
	2) Contents of A remain unchanged
STA, MG	1) $\lvert (A) \rvert \rightarrow M$
	2) Contents of A remain unchanged
STA, CL	1) $(A) \rightarrow M$
	2) Contents of A are cleared after transmission

EXAMPLES

PROBLEM:

Store A at five memory locations forward.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		STA	*+5	

PROBLEM:

Store A at address COUNTER and then clear A.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		STA,CL	CØUNTER	

PROBLEM:

Store the absolute value of A at an address specified in Index Register 1.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		STA,MG	,1	

PROBLEM:

Store A indirectly at SAVE.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		STA	SAVE,7	

The STORE Q Instruction

The STORE Q instruction is an instruction that transmits a 48-bit operand from the Q register to an 18-bit storage address. One memory reference is made.

The operation leaves the contents of the Q register unchanged unless the modifier CL is specified (see below). The contents of Q replace the contents of the storage address.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the contents of Q is transmitted to the storage address.

If MG is specified by the programmer, the magnitude (absolute value) of the contents of Q is transmitted to the storage address.

If CL is specified by the programmer, the contents of Q are cleared <u>after</u> the operand has been transmitted to the storage address.

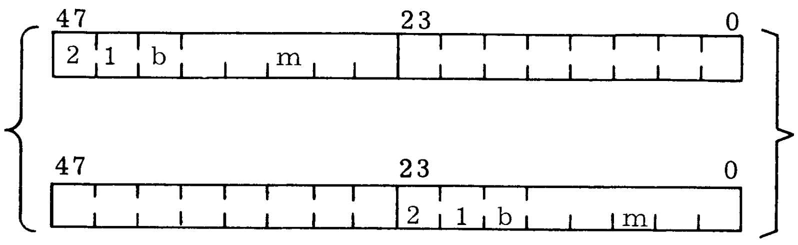
STORE Q

STQ

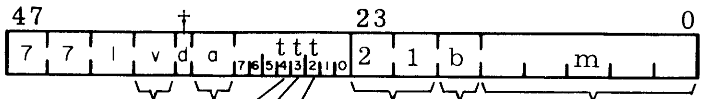
FORMAT

MACHINE

May be an upper or lower instruction

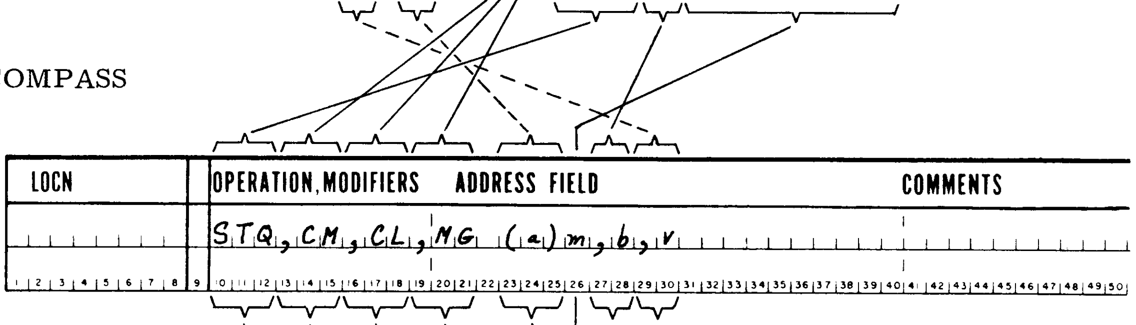


NORMAL



AUGMENTED

COMPASS



Function code

Optional, transmit complement of operand.

Optional, clear (Q) after transmission of operand.

Optional, transmit magnitude of operand.

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

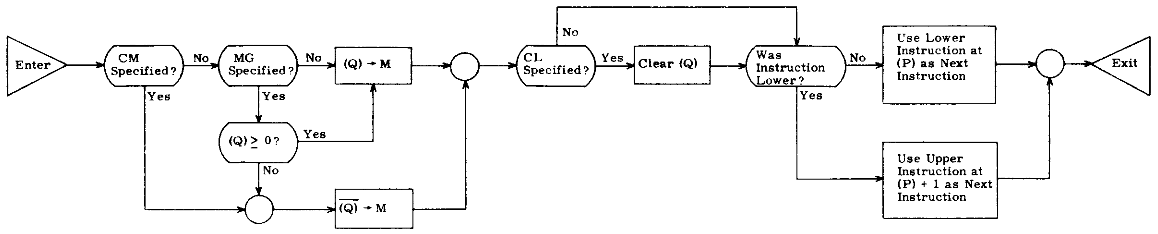
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

STQ

FLOWCHART

STORE Q FORMAT: STQ, CM, CL, MG (a) m, b, v $M = m + (B^b) + (V^v)$



DESCRIPTION

STQ	1) $(Q) \rightarrow M$
	2) Contents of Q remain unchanged
STQ, CM	1) $\overline{(Q)} \rightarrow M$
	2) Contents of Q remain unchanged
STQ, MG	1) $\lvert (Q) \rvert \rightarrow M$
	2) Contents of Q remain unchanged
STQ, CL	1) $(Q) \rightarrow M$
	2) Contents of Q are cleared after transmission

PROBLEM:

Store Q at address SMOKEY modified by Index Registers 2 and 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STQ	SMØKEY,2,3	

PROBLEM:

Store the complement of Q at address BEAR.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STQ,CM	BEAR	

PROBLEM:

Store Q at one memory location forward of the address specified in Index Register 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STQ	1,3	

PROBLEM:

Store Q at an address which is five memory locations backward from an address specified in Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STQ	-5,1	

The TRANSMIT Instruction

The TRANSMIT instruction transmits a word or a group of words from one area of memory to another area of memory. The operation leaves the contents of the source area unchanged.

If CM is specified by the programmer, the complement of the source is transmitted to the destination address.

If MK is specified by the programmer, the source is masked with the contents of Q before being transmitted to the destination address.

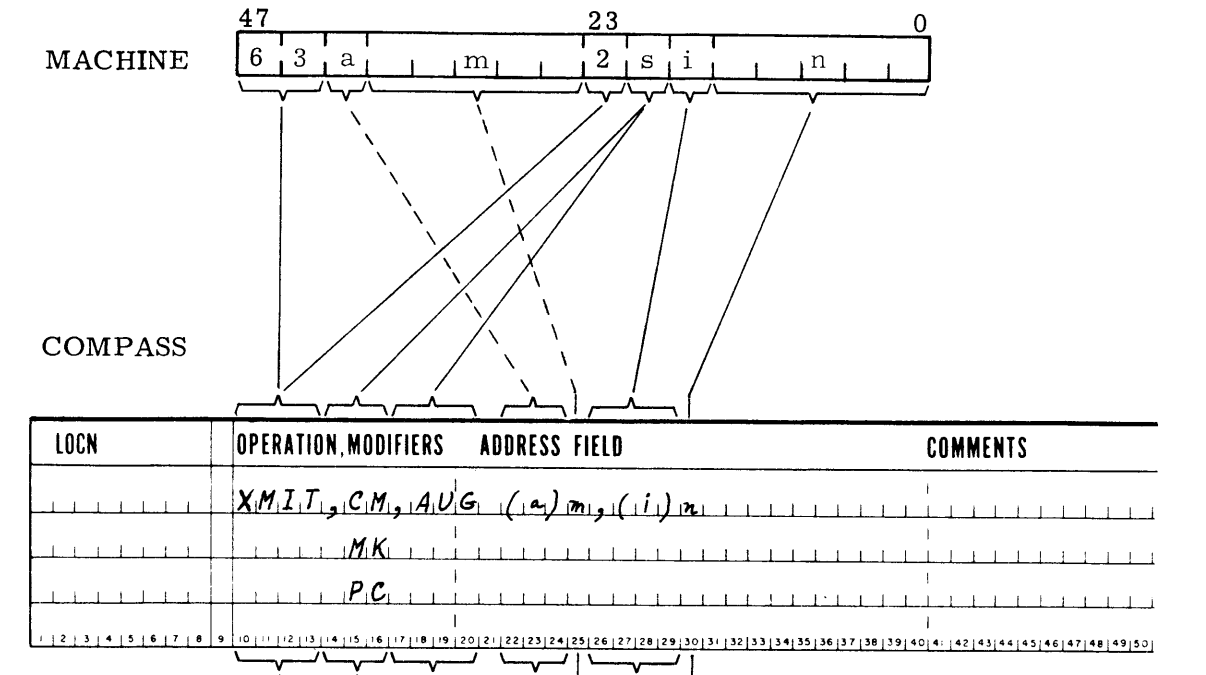
If PC is specified by the programmer, the source is incremented by the contents of A in fixed point format before being transmitted to the destination address.

If AUG is not specified by the programmer, only one word is transmitted. The source address is (a) m and the destination address is (i) n. The index registers are not significant.

If AUG is specified by the programmer, five index registers must be set up before executing this instruction.

(B^1) = number of words to transmit
(B^2) = source address modifier (normally zero to start)
(B^3) = incrementer for B^2 (normally 1 for sequential words)
(B^4) = destination address modifier (normally zero to start)
(B^5) = incrementer for B^4 (normally 1 for sequential words)

FORMAT



Function code

Optional modifier

CM, Transmit Compliment

MK, mask with Q

PC, Add to constant in A

Blank, Transmit normally

Optional modifier

Transmit according to B1-B5

Blank, transmit one word

Destination address.

Bank of destination address.

0-7, actual bank number.

$, bank in which destination address resides.

*, bank in which XMIT resides.

Source address.

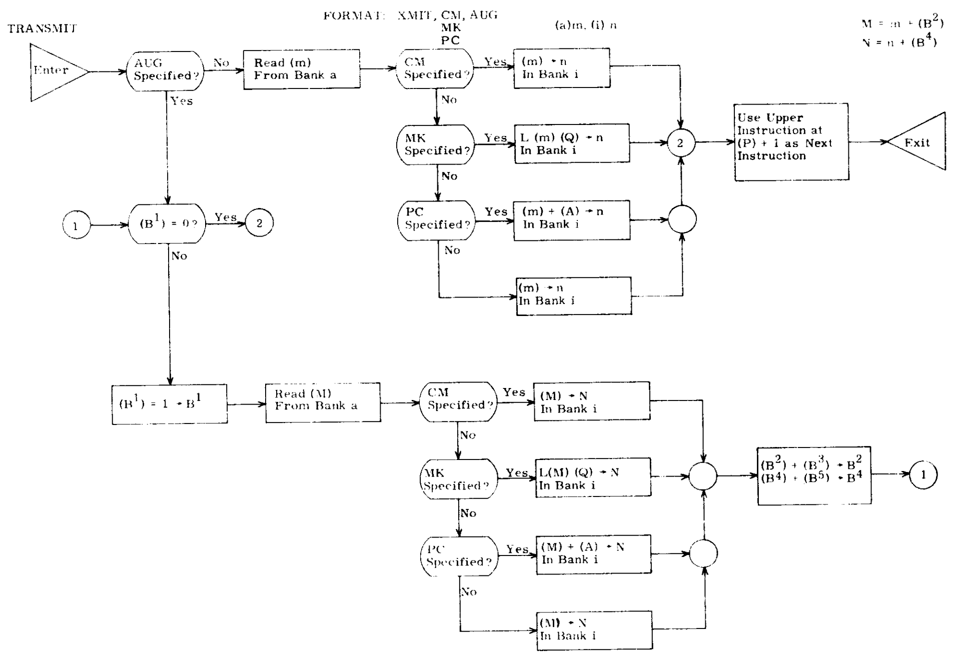
Bank of source address.

0-7, actual bank number.

$, bank in which source address resides.

*, bank in which XMIT resides.

FLOWCHART



DESCRIPTION

XMIT	Transmit one word
XMIT, CM	Transmit complement of one word
XMIT, MK	Transmit one word with contents of Q as mask
XMIT, PC	Transmit one word with value in A as constant
XMIT, AUG	Transmit block of words according to B1 - B5

PROBLEM:

Transmit an operand from Bank 2 address HOKEY to Bank 3 address POKEY.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		XMIT	(2) HØKEY, (3) PØKEY	

PROBLEM: Given: (Q) = +77B

Extract the lowest six bits from Bank 5 address MURTLE and place the result in Bank 6 address TURTLE.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		XMIT, MK	(5) MURTLE, (6) TURTLE	

PROBLEM: Given: $(B^1) = 100$, $(B^2) = 0$, $(B^3) = 1$, $(B^4) = 0$, $(B^5) = 1$

Transmit the 100 words from Bank 1 address BLOCK to Bank 1 address TABLE.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		XMIT, AUG	(1) BLØCK, (1) TABLE	

PROBLEM: Given: $(B^1) = 10$, $(B^2) =$ CARD, $(B^3) = 1$, $(B^4) =$ ZILCH, $(B^5) = 1$

Transmit the 10 words starting at CARD to the 10 locations starting with ZILCH. Both are in Bank 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		XMIT, AUG	(3), (3)	

NEW CONCEPTS OF GROUP 1

The first group contains relatively few new concepts. 48-bit words are transmitted from a source to a destination. The source always remains as it was except on the STA and STQ instructions when CL is specified as a modifier. In this case the register is cleared after the transmission takes place.

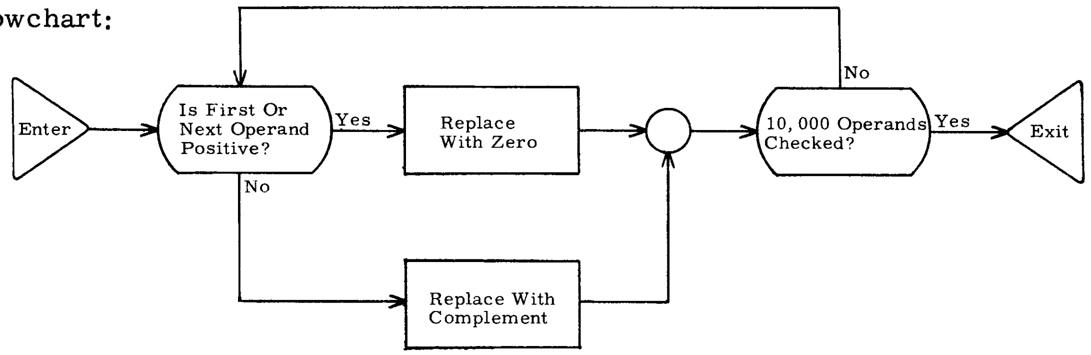
Operands can be complemented as they are transferred. The 1's complement format is used. This algebraically changes the sign of the operand.

The magnitude (absolute value) can also be transferred. In this case a check is made on the sign of the operand. If it is negative, it is complemented during transmission. If it is positive, it is a normal transmission. In either case the positive form is transferred.

Problem 1:

A 10, 000 word table in memory starting at address TAB contains operands both positive and negative. Replace every positive operand with zero and every negative operand with its absolute value.

Flowchart:



Problem 1 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	TEST	
	ENTRY	TEST	
TEST	BSS	1	
	ENQ	0	
	ENI	0, 1	
NEXT	LDA	TAB, 1	
	AJP, MI	MINUS	PØS ØR NEG
PLUS	STQ	TAB, 1	PØS
	UJP	INCR	
MINUS	STA, CM	TAB, 1	NEG
INCR	ISK	9999, 1	
	SLJ	NEXT	
	SLJ	TEST	
	END		

Somewhere within this subprogram would also be included the symbol TAB in the location field with a declaration of the prestored data or area reserved.

Student Problem 1A:

Using the XMIT instruction, transmit 10 words starting at address CARD to an area starting at address CARD1. As each word is transferred, extract the lower 9 bits (zero out the upper 39 bits) of each word.

Flowchart:

Problem 1A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 2

ADDRESS TRANSMISSION

GROUP 2

ADDRESS TRANSMISSION

1.	Load Index Upper	LIU
2.	Load Index Lower	LIL
3.	Store Index Upper	SIU
4.	Store Index Lower	SIL
5.	Substitute Address Upper	SAU
6.	Substitute Address Lower	SAL
7.	Enter Index	ENI
8.	Enter A	ENA
9.	Enter Q	ENQ

This group of instructions transmits 15-bit quantities that may be either addresses or operands.

The first two instructions transmit a portion of a memory word to an index register, requiring one storage cycle.

The second two instructions transmit the contents of an index register to a portion of a memory word at a storage address, leaving the rest of the memory word untouched. One storage cycle is required.

The next two instructions transmit the lower address portion of A to a portion of a memory word, leaving the rest of the memory word untouched. Again, one storage cycle is required.

The ENTER instructions transmit a 15-bit quantity to a register. No storage cycle is required for these instructions.

The LOAD INDEX UPPER Instruction

The LOAD INDEX UPPER instruction is an instruction that transmits a 15-bit quantity from the upper address portion of a memory word to an index register specified by index designator b. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The 15-bit upper address portion of the memory word replaces the contents of the index register.

The memory word is found at an 18-bit storage address composed of a bank term a (within parentheses) and a modified base address M where $M = m + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The v index designator allows for relative addressing. If it is not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the upper address portion of the memory word is transmitted to the index register.

If MG is specified by the programmer, the magnitude (absolute value) of the upper address portion of the memory word is transmitted to the index register.

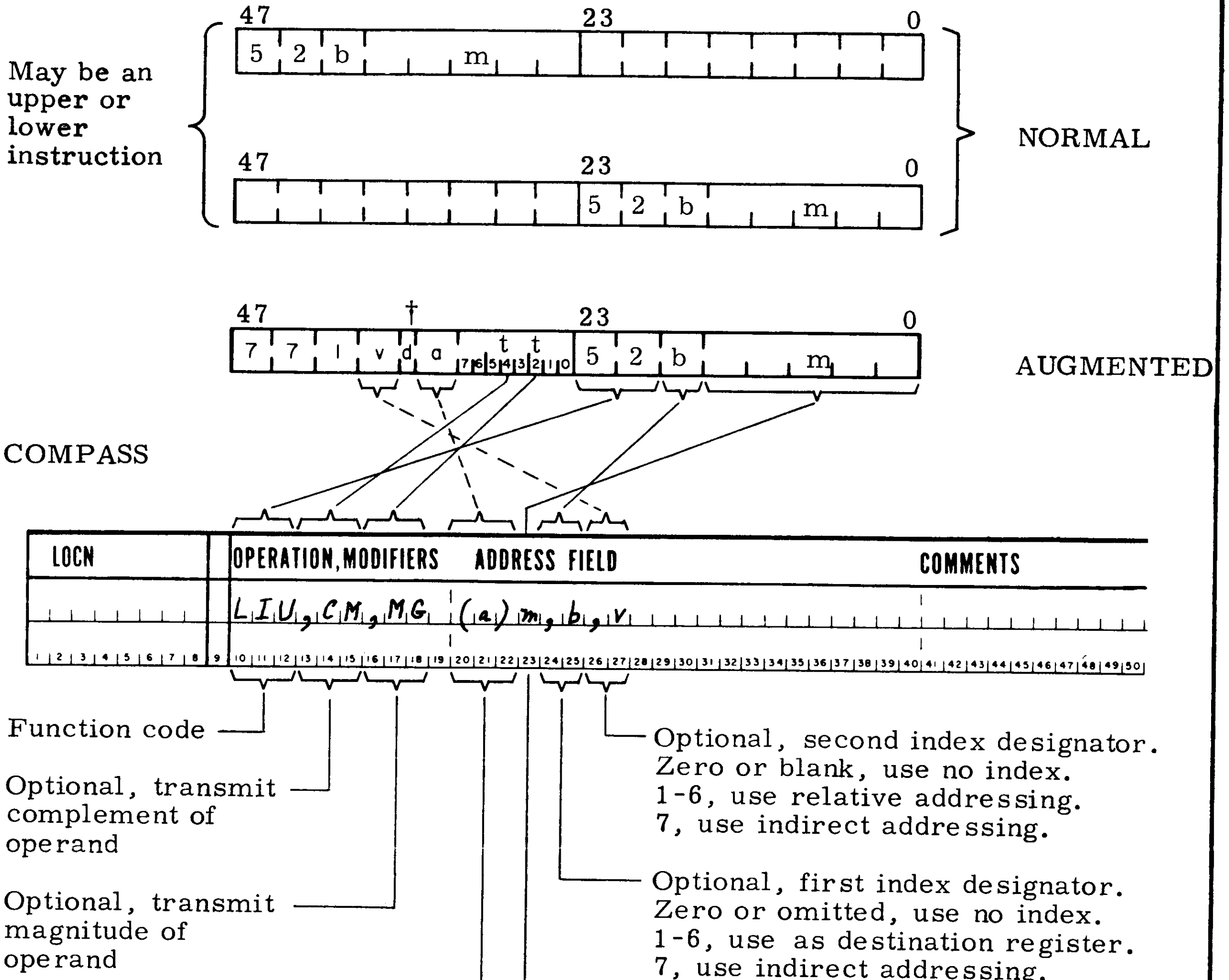
The CM and MG modifiers are seldom used for this instruction.

LOAD INDEX UPPER

LIU

F O R M A T

MACHINE



Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use as destination register.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

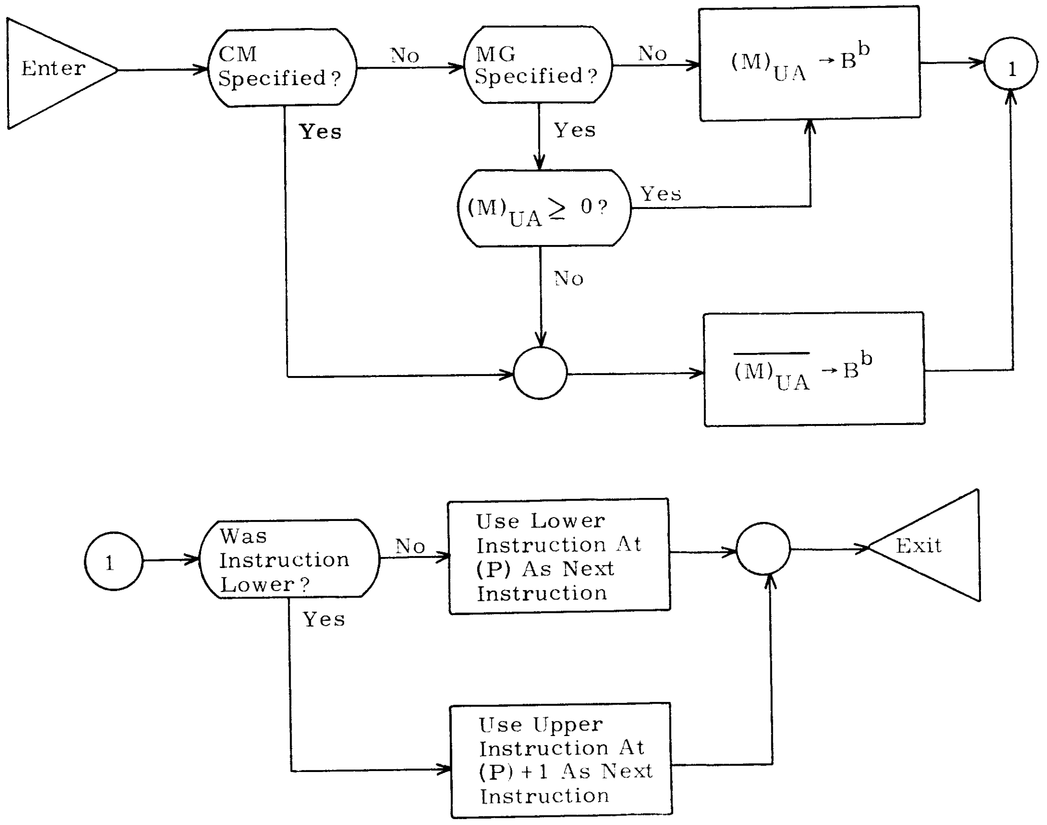
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

LIU

FLOWCHART

LOAD INDEX UPPER FORMAT: LIU, CM, MG (a) m, b, v $M = m + (V^v)$



Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
$(M)_{UA} \rightarrow B^b$
1
$(M)_{UA} \geq 0$?
Yes
No
$\overline{(M)_{UA}} \rightarrow B^b$
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

DESCRIPTION

LIU	1) $(M)_{UA} \rightarrow B^b$
	2) Contents of M_{UA} remain unchanged
LIU, CM	1) $\overline{(M)_{UA}} \rightarrow B^b$
	2) Contents of M_{UA} remain unchanged
LIU, MG	1) $\lvert (M)_{UA} \rvert \rightarrow B^b$
	2) Contents of M_{UA} remain unchanged

PROBLEM:

Load Index Register 1 with the upper address portion of the word at address MORTIMER.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LIU	MØRTIMER,1	

PROBLEM:

Load Index Register 6 with the upper address portion of the word at address SNERD modified by Index Register 5.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LIU	SNERD,6,5	

PROBLEM:

Load Index Register 2 with the upper address portion of the word at the address specified by Index Register 4.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LIU	,2,4	

PROBLEM:

Load Index Register 5 with the upper address portion of the word at an address four memory locations backward from this instruction.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LIU	*-4,5	

The LOAD INDEX LOWER Instruction

The LOAD INDEX LOWER instruction is an instruction that transmits a 15-bit quantity from the lower address portion of a memory word to an index register specified by index designator $\underline{b}$. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The 15-bit lower address portion of the memory word replaces the contents of the index register.

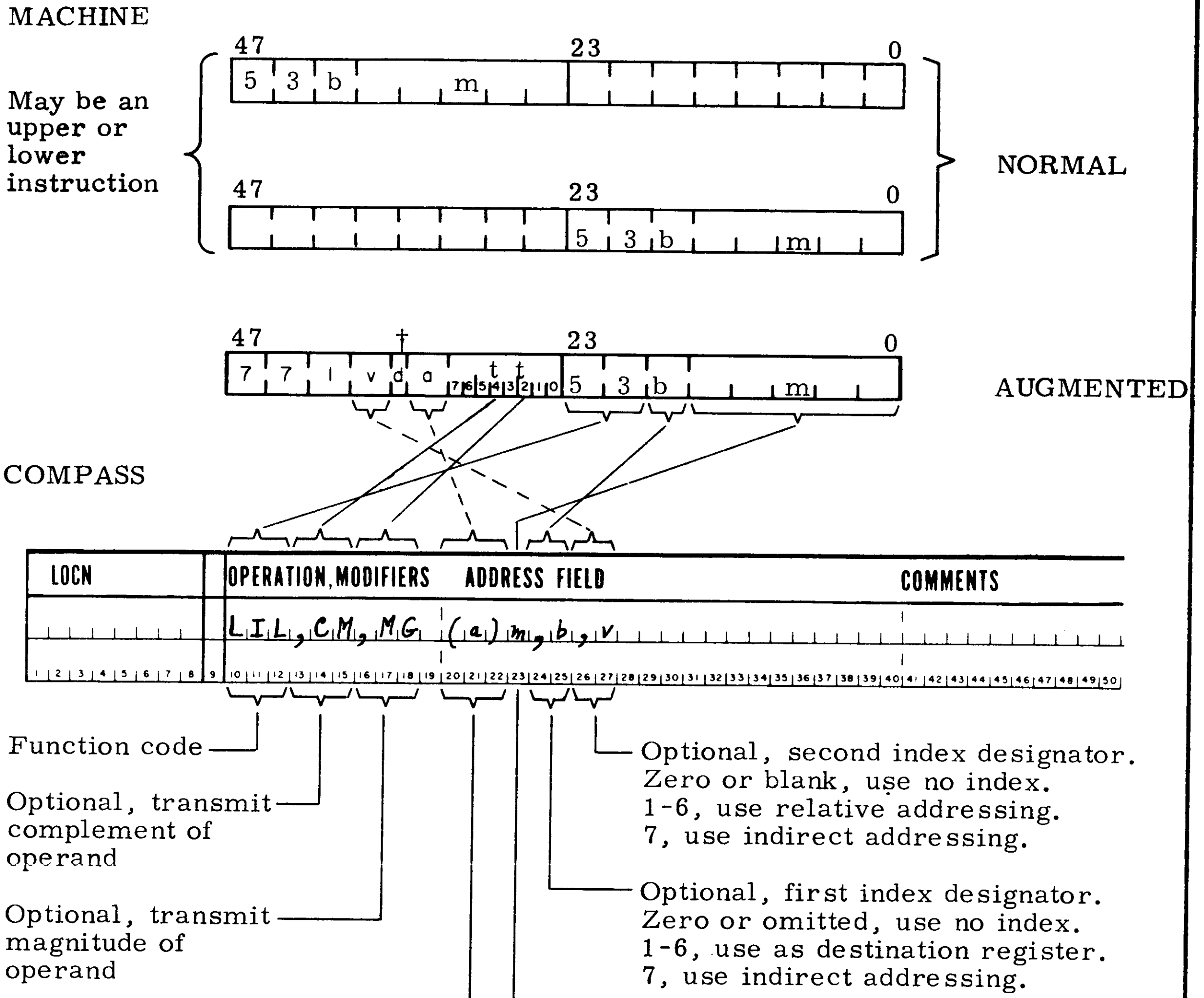
The memory word is found at an 18-bit storage address composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (V^{v})$. If the bank term is missing, the current operand bank setting is assumed. The v index designator allows for relative addressing. If it is not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the lower address portion of the memory word is transmitted to the index register.

If MG is specified by the programmer, the magnitude (absolute value) of the lower address portion of the memory word is transmitted to the index register.

The CM and MG modifiers are seldom used for this instruction.

FORMAT



Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use as destination register.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

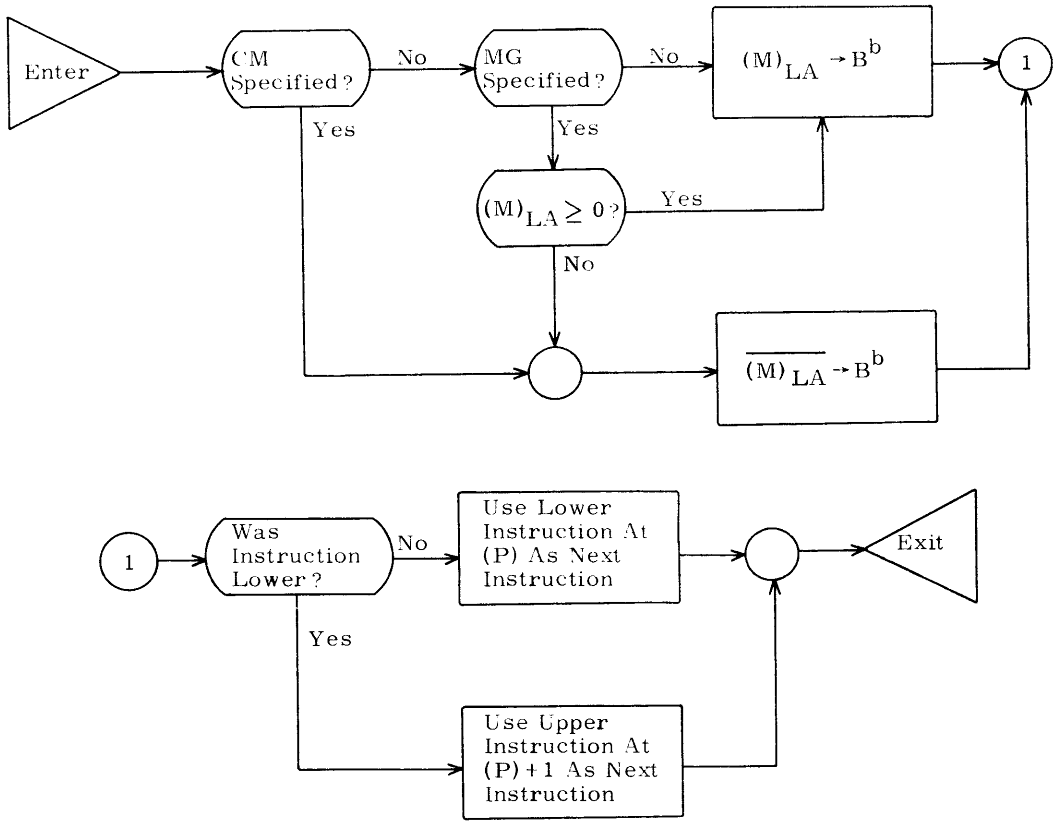
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

LIL

LOAD INDEX LOWER FORMAT: LIL, CM, MG (a) m,b,v $M = m + (V^v)$

FLOWCHART


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
$(M)_{LA} \to B^b$
1
$(M)_{LA} \geq 0$?
Yes
No
$\overline{(M)_{LA}} \to B^b$
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

LIL	1) $(M)_{LA} \to B^b$
	2) Contents of M_{LA} remain unchanged
LIL, CM	1) $\overline{(M)}_{LA} \to B^b$
	2) Contents of M_{LA} remain unchanged
LIL, MG	1) $\lvert (M)_{LA} \rvert \to B^b$
	2) Contents of M_{LA} remain unchanged

E
X
A
M
P
L
E
S

PROBLEM:

Load Index Register 1 with the lower address portion of the memory word at address CHAREAD.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LIL	CHAREAD,1	

PROBLEM:

Load Index Register 2 with the lower address portion of the memory word at address CARACDEF + 3.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LIL	CARACDEF+3,2	

PROBLEM:

Load Index Register 3 with the lower address portion of the memory word at address CALFIDNT modified by Index Register 4.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LIL	CALFIDNT,3,4	

PROBLEM:

Load Index Register 5 with the lower address portion of the memory word at an address specified in Index Register 6.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LIL	,5,6	

The STORE INDEX UPPER Instruction

The STORE INDEX UPPER instruction is an instruction that transmits a 15-bit quantity from an index register specified by index designator b to the upper address portion of a memory word. One memory reference is made.

The operation leaves the contents of the index register unchanged. The 15-bit quantity of the index register replaces the upper address portion of the memory word. The rest of the memory word is unchanged.

The memory word is found at an 18-bit storage address composed of a bank term a (within parentheses) and a modified base address M where $M = m + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The v index designator allows for relative addressing. If it is not used, direct addressing is implied.

STORE INDEX UPPER

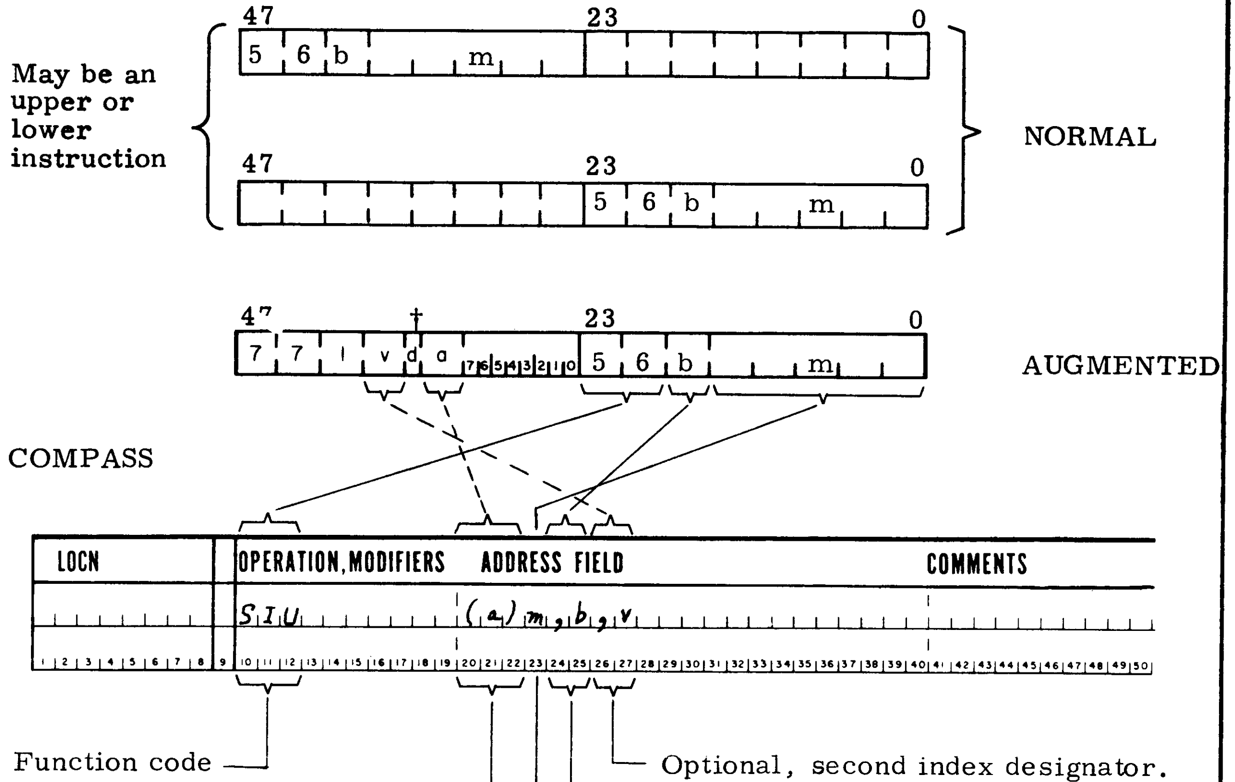
SIU

FORMAT

MACHINE

May be an upper or lower instruction

COMPASS



Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use as source register.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

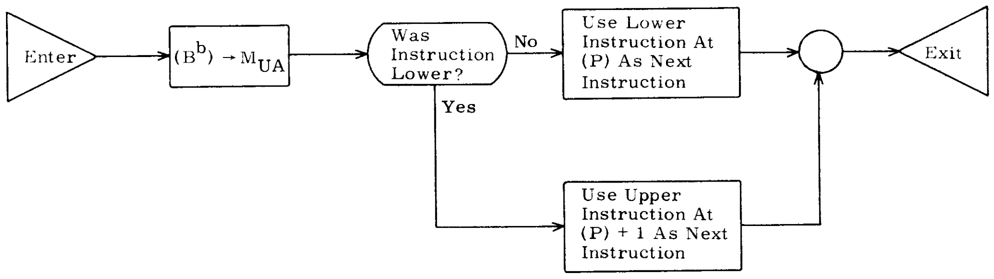
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

SIU

FLOWCHART

STORE INDEX UPPER   FORMAT: SIU   (a) m, b, v $M = m + (V^v)$


Enter
$(B^b) \rightarrow M_{UA}$
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

DESCRIPTION

SIU

1) $(B^b) \rightarrow M_{UA}$
2) Contents of B^b remain unchanged

PROBLEM:

Store Index Register 1 in the upper address portion of the memory word at address SAM.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SIU	SAM,1	

PROBLEM:

Store Index Register 2 in the upper address portion of the memory word in Bank 3 address TOM.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SIU	(3) TØM	

PROBLEM:

Store Index Register 3 in the upper address portion of the memory word at address SAVE + 4.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SIU	SAVE+4,3	

PROBLEM:

Store Index Register 4 in the upper address portion of the memory word at address JOE in the bank where JOE resides.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SIU	($)JØE,4	

The STORE INDEX LOWER Instruction

The STORE INDEX LOWER instruction is an instruction that transmits a 15-bit quantity from an index register specified by index designator $\underline{b}$ to the lower address portion of a memory word. One memory reference is made.

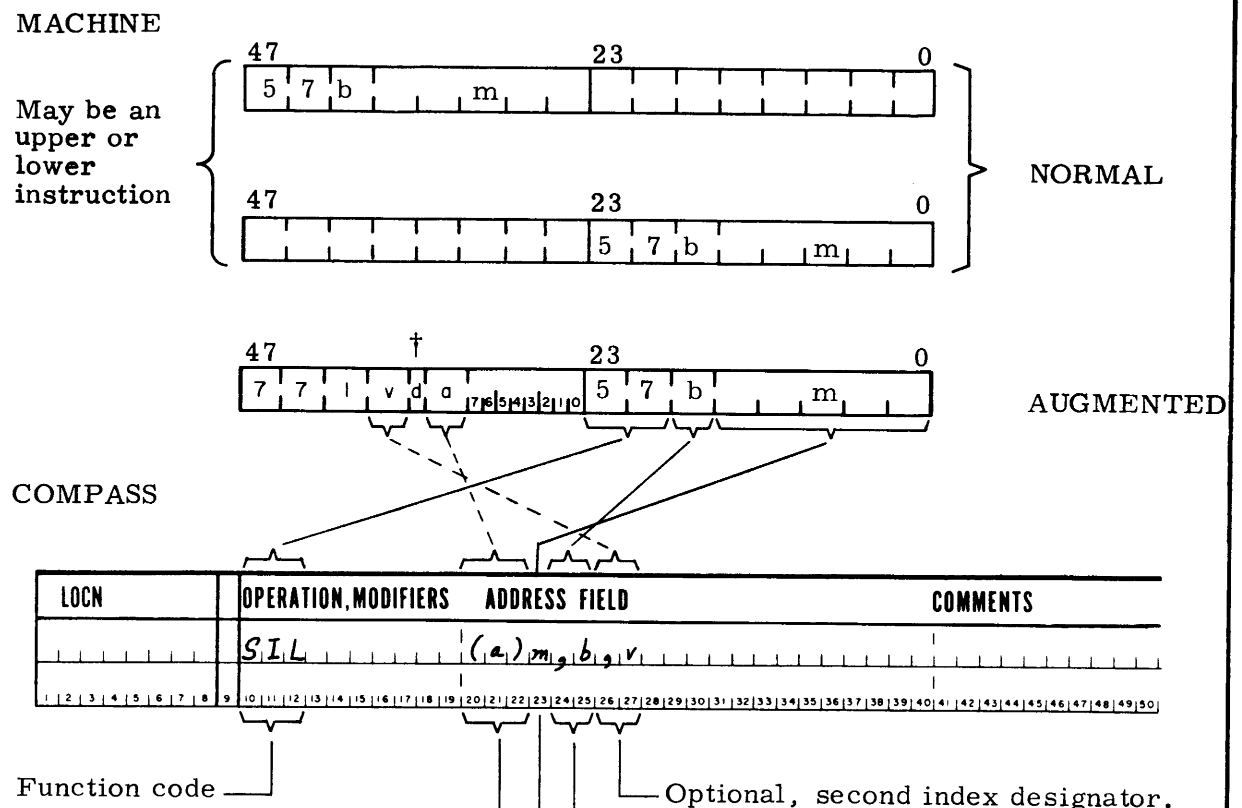
The operation leaves the contents of the index register unchanged. The 15-bit quantity of the index register replaces the lower address portion of the memory word. The rest of the memory word is unchanged.

The memory word is found at an 18-bit storage address composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The v index designator allows for relative addressing. If it is not used, direct addressing is implied.

STORE INDEX LOWER

SIL

FORMAT



Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use as source register.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

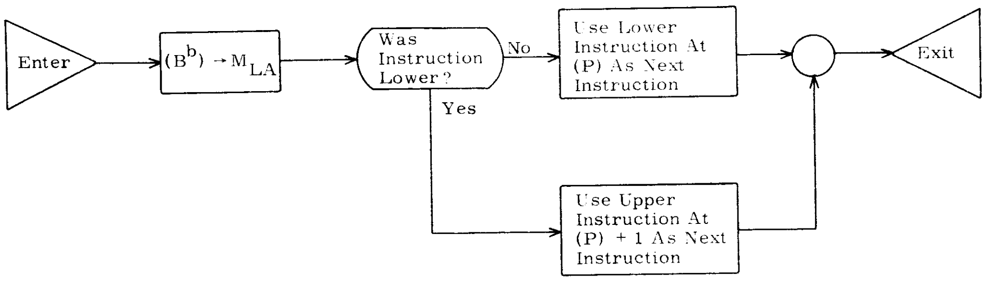
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

SIL

FLOWCHART

STORE INDEX LOWER FORMAT: SIL (a) m, b, v $M = m + (V^v)$


Enter
$(B^b) \rightarrow M_{LA}$
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

DESCRIPTION

SIL

1) $(B^b) \rightarrow M_{LA}$
2) Contents of B^b remain unchanged

PROBLEM:
Store Index Register 1 in the lower address portion of the memory word at address SAVE.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SIL	SAVE,1	

PROBLEM:
Store Index Register 2 in the lower address portion of the memory word at an address two memory locations forward of address BILL.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SIL	BILL+2,2	

PROBLEM:
Store Index Register 3 in the lower address portion of the memory word at an address specified in Index Register 5.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SIL	,3,5	

PROBLEM:
Store Index Register 4 in the lower address portion of the memory word at address JAKE in the bank in which the SIL resides.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SIL	(*)JAKE,4	

The SUBSTITUTE ADDRESS UPPER Instruction

The SUBSTITUTE ADDRESS UPPER instruction is an instruction that transmits a 15-bit quantity from the lower address portion of the A register to the upper address portion of a memory word. One memory reference is made.

The operation leaves the contents of the A register unchanged. The 15-bit lower address portion of the A register replaces the upper address portion of the memory word. The rest of the memory word is unchanged.

The memory word is found at an 18-bit storage address composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the lower address portion of the A register is transmitted to the upper address portion of the memory word.

If MG is specified by the programmer, the magnitude (absolute value) of the lower address portion of the A register is transmitted to the upper address portion of the memory word.

The CM and MG modifiers are seldom used for this instruction.

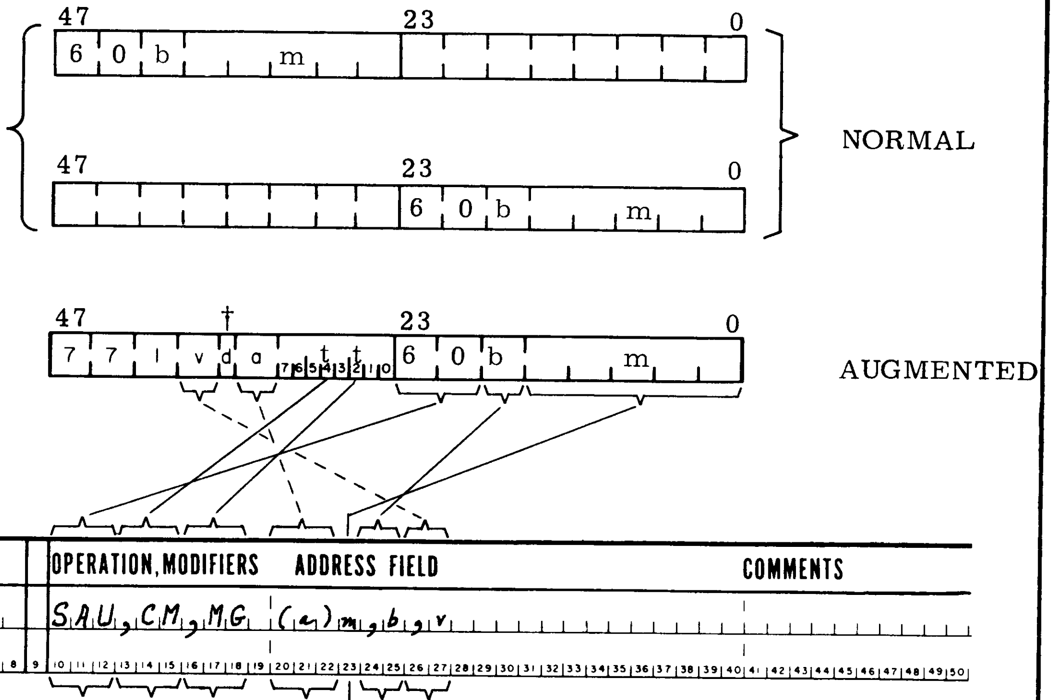
SUBSTITUTE ADDRESS UPPER

SAU

FORMAT

MACHINE

May be an upper or lower instruction



Function code

Optional, transmit complement of address

Optional, transmit magnitude of address

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

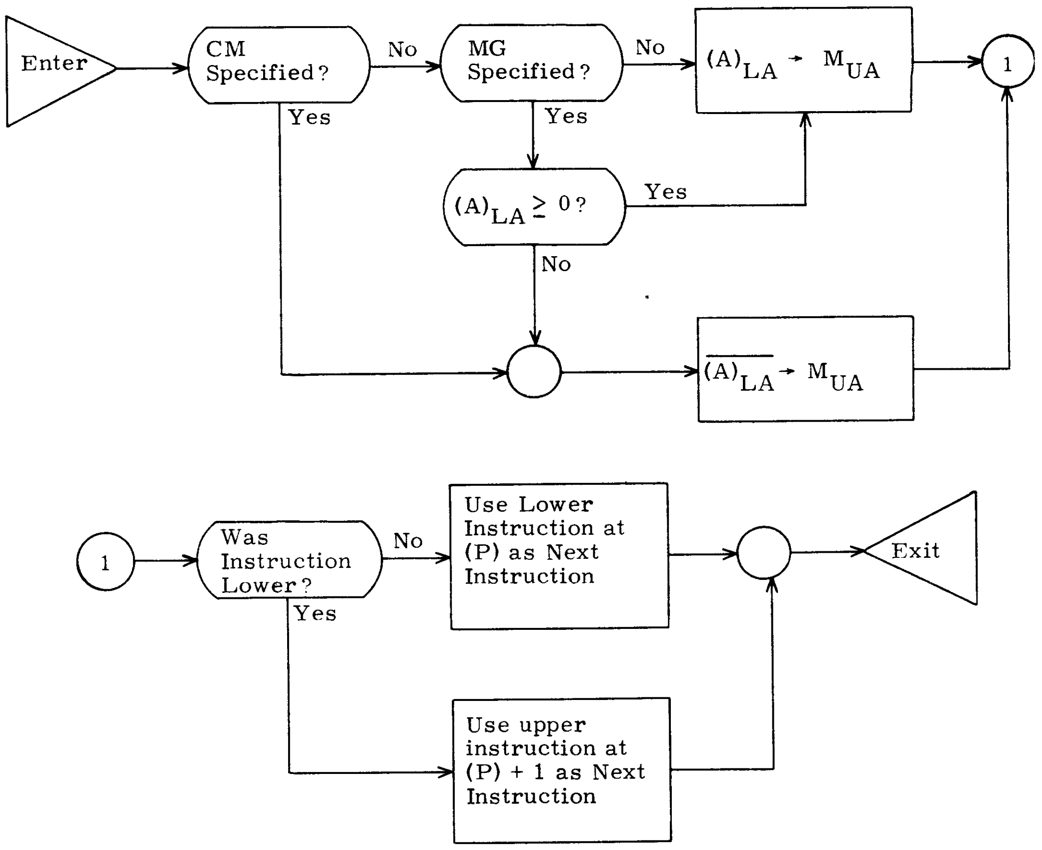
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

SAU

SUBSTITUTE ADDRESS UPPER FORMAT: SAU,CM,MG (a) m,b,v $M=m+(B^b)+(V^v)$

FLOWCHART



Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
$(A)_{LA} \rightarrow M_{UA}$
1
$(A)_{LA} \geq 0$?
Yes
No
$\overline{(A)_{LA}} \rightarrow M_{UA}$
1
Was Instruction Lower?
No
Yes
Use Lower Instruction at (P) as Next Instruction
Use upper instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

SAU	1) $(A)_{LA} \rightarrow M_{UA}$ 2) Contents of A remain unchanged
SAU, CM	1) $\overline{(A)_{LA}} \rightarrow M_{UA}$ 2) Contents of A remain unchanged
SAU, MG	1) $\lvert (A)_{LA} \rvert \rightarrow M_{UA}$ 2) Contents of A remain unchanged

EXAMPLES

PROBLEM:
Store the lower address portion of A in the upper address portion of TEMP.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		SAU	TEMP	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:
Store the lower address portion of A in the upper address portion of TEMP modified by Index Register 4.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		SAU	TEMP,4	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:
Store the lower address portion of A in the upper address portion of an address specified in Index Register 5.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		SAU	,5	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:
Store the lower address portion of A in the upper address portion of address SLOJOE in Bank 1.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		SAU	(1) SLØJØE	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The SUBSTITUTE ADDRESS LOWER Instruction

The SUBSTITUTE ADDRESS LOWER instruction is an instruction that transmits a 15-bit quantity from the lower address portion of the A register to the lower address portion of a memory word. One memory reference is made.

The operation leaves the contents of the A register unchanged. The 15-bit lower address portion of the A register replaces the lower address portion of the memory word. The rest of the memory word is unchanged.

The memory word is found at an 18-bit storage address composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the lower address portion of the A register is transmitted to the lower address portion of the memory word.

If MG is specified by the programmer, the magnitude (absolute value) of the lower address portion of the A register is transmitted to the lower address portion of the memory word.

The CM and MG modifiers are seldom used for this instruction.

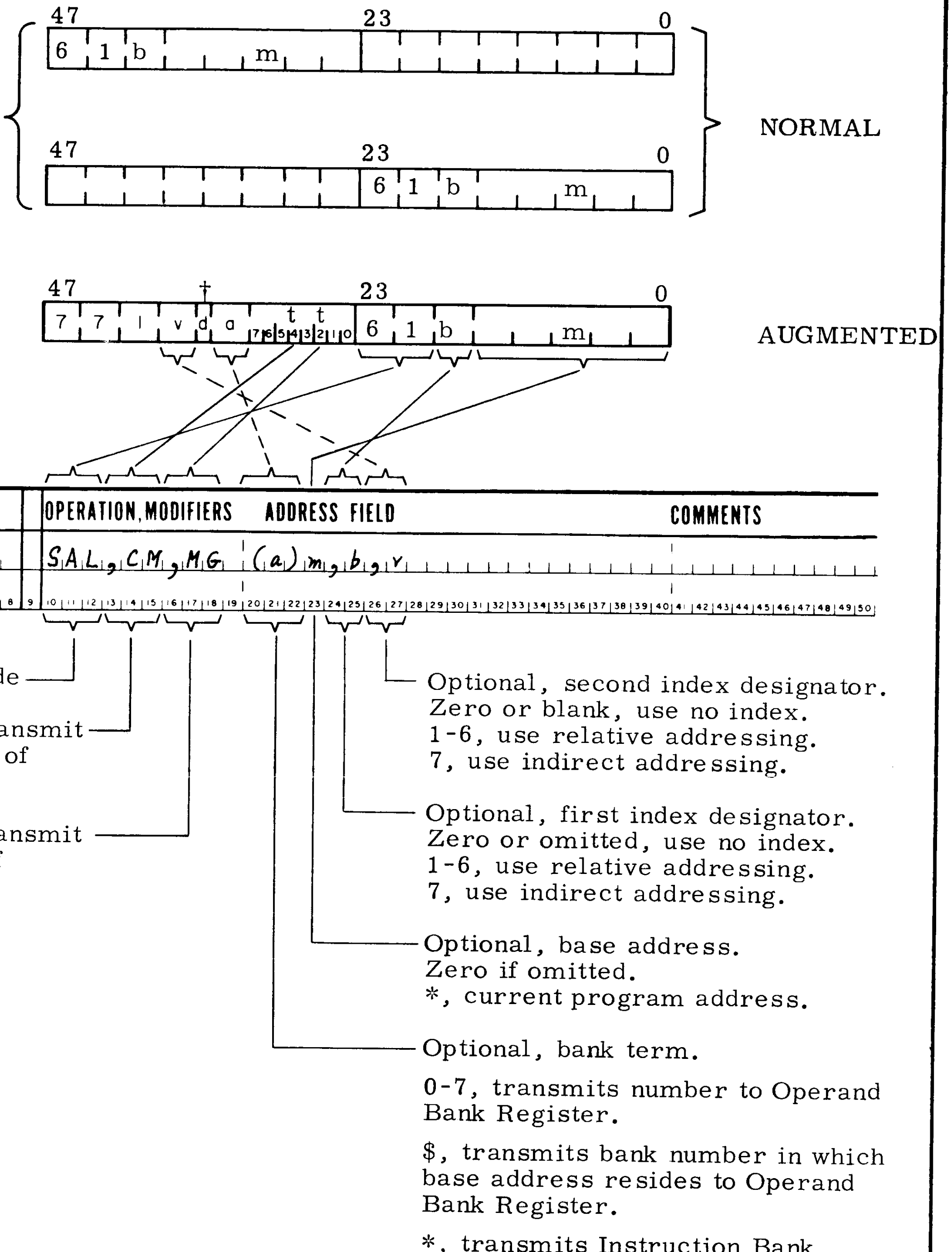
FORMAT

SUBSTITUTE ADDRESS LOWER

SAL

MACHINE

May be an upper or lower instruction



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

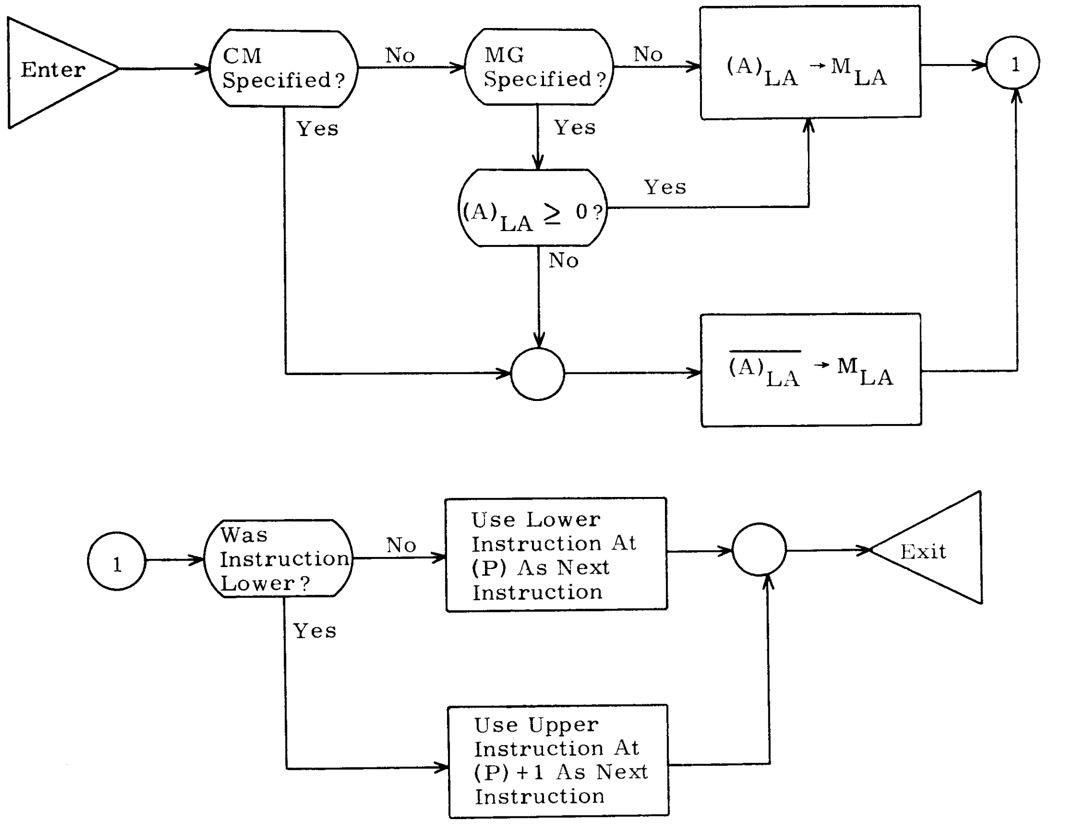
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

SAL

F L O W C H A R T

SUBSTITUTE ADDRESS LOWER FORMAT: SAL, CM, MG (a)m, b, v $M=m+(B^b) + (V^v)$


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
$(A)_{LA} \rightarrow M_{LA}$
1
$(A)_{LA} \geq 0$?
Yes
No
$\overline{(A)_{LA}} \rightarrow M_{LA}$
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

D E S C R I P T I O N

SAL	1) $(A)_{LA} \rightarrow M_{LA}$
	2) Contents of A remain unchanged
SAL, CM	1) $\overline{(A)_{LA}} \rightarrow M_{LA}$
	2) Contents of A remain unchanged
SAL, MG	1) $\lvert (A)_{LA} \rvert \rightarrow M_{LA}$
	2) Contents of A remain unchanged

EXAMPLES

PROBLEM:
Store the lower address portion of A in the lower address portion of PETE.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		SAL	PETE	

PROBLEM:
Store the lower address portion of A in the lower address portion of PETE-1 modified by Index Register 3.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		SAL	PETE-1,3	

PROBLEM:
Store the lower address portion of A in the lower address portion of TOM in the bank where TOM resides.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		SAL	($)TØM	

PROBLEM:
Store the lower address portion of A in the lower address portion of an address that is three memory locations relatively forward.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		SAL	*+3	

The ENTER INDEX Instruction

The ENTER INDEX instruction is an instruction that transmits a 15-bit quantity $\underline{Y}$, where $Y = y + (V^V)$, into an index register specified by the index designator $\underline{b}$. No memory reference is made.

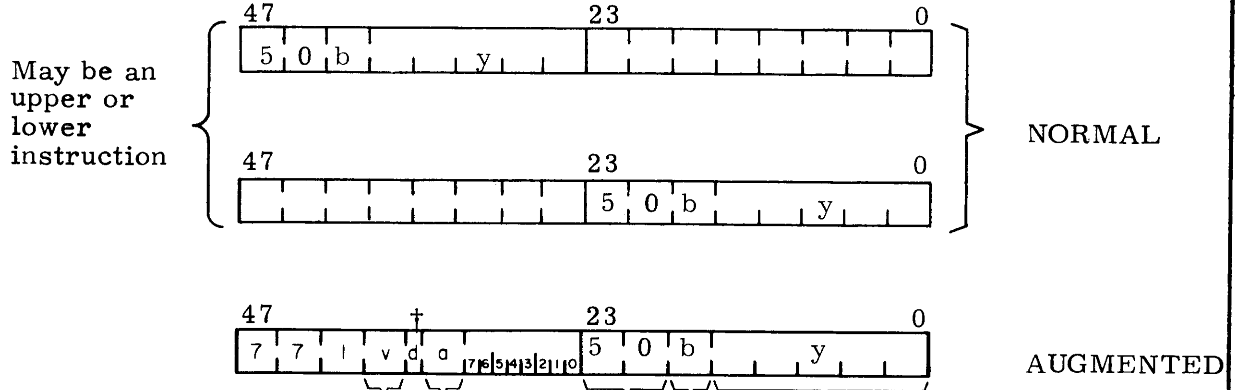
If the bank term $\underline{a}$ (within parentheses) is missing, the current operand bank setting remains. If the bank term $\underline{a}$ is used, the current operand bank setting will be replaced by the value $\underline{a}$. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

ENTER INDEX

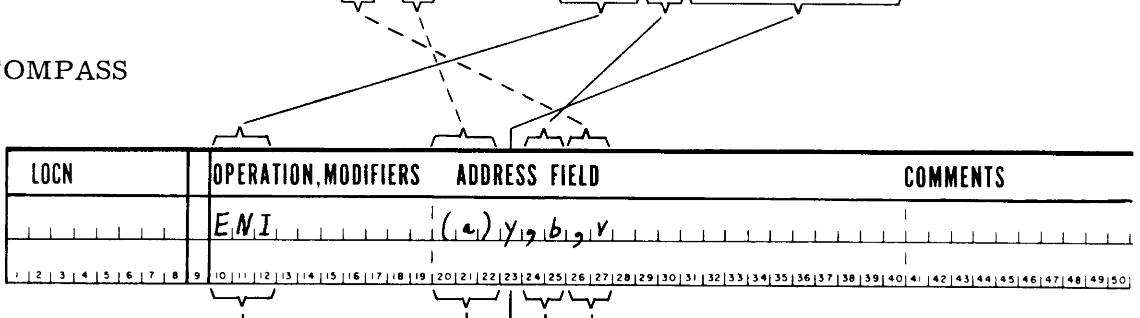
ENI

FORMAT

MACHINE



COMPASS



Function code

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base operand.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

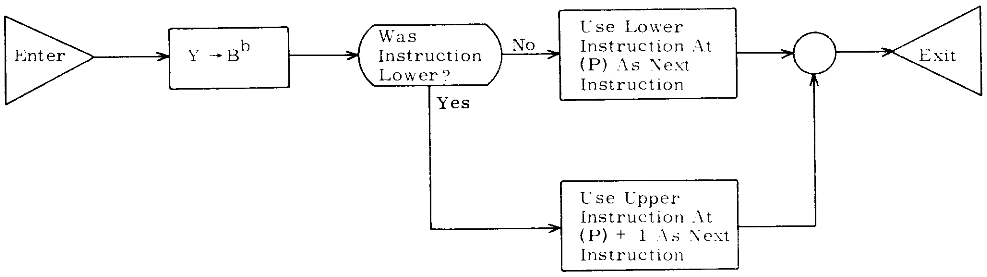
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

ENI

FLOWCHART

ENTER INDEX FORMAT: ENI (a) y, b, v $Y = y + (V^v)$


Enter
$Y \rightarrow B^b$
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

DESCRIPTION

ENI $Y \rightarrow B^b$

PROBLEM:
Enter Index Register 1 with zero.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ENI	,1	

PROBLEM:

Enter Index Register 2 with minus zero (all 7's).

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ENI	-0,2	

PROBLEM:

Enter Index Register 3 with an octal 77.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ENI	77B,3	

PROBLEM:

Enter Index Register 4 with address SNOPS.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ENI	SNØPS,4	

The ENTER A Instruction

The ENTER A instruction is an instruction that transmits a 15-bit quantity Y, where $Y = y \dotplus (B^b) + (V^v)$, into the A register. No memory reference is made.

The 15-bit quantity is transmittedright-justified into the A register with the sign bit extended automatically through the rest of A.

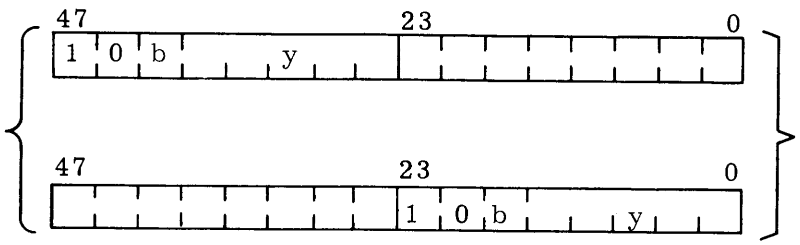
If the bank term a (within parentheses) is missing, the current operand bank setting remains . If the bank term a is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

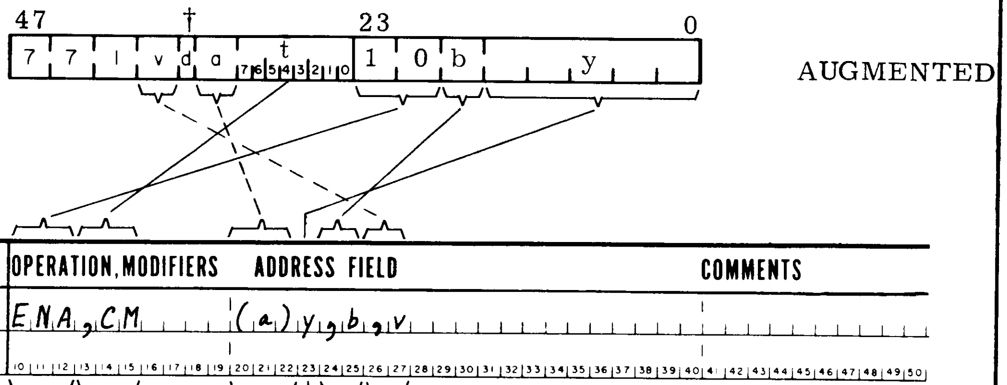
If CM is specified by the programmer, the complement of the quantity Y is transmitted to the A register.

FORMAT

MACHINE

May be an upper or lower instruction





COMPASS

Function code

Optional, transmit complement of operand

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base operand. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number of Operand Bank Register.

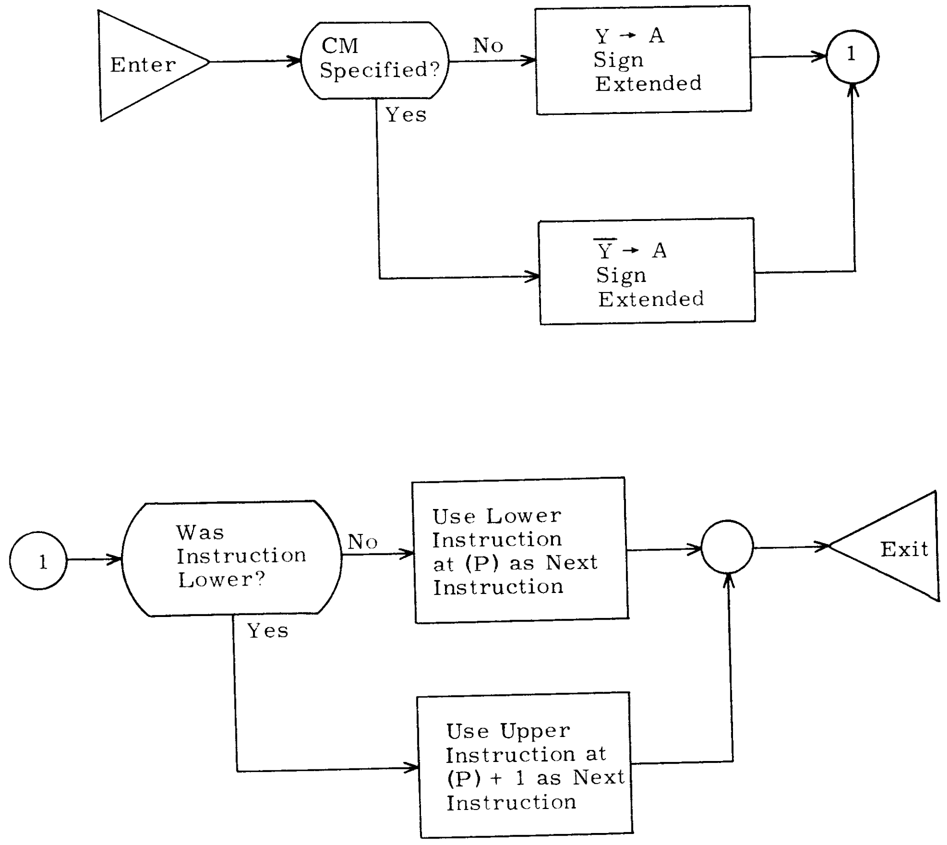
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

ENA

FLOWCHART

ENTER A FORMAT: ENA,CM (a) y,b,v $Y = y + (B^b) + (V^v)$


Enter
CM Specified?
No
Yes
Y → A Sign Extended
$\overline{Y}$ → A Sign Extended
1
1
Was Instruction Lower?
No
Yes
Use Lower Instruction at (P) as Next Instruction
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

ENA	$Y \rightarrow A$
ENA, CM	$\overline{Y} \rightarrow A$

PROBLEM:

Enter A with zero.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ENA		

PROBLEM:

Enter A with the contents of Index Register 1.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ENA	,1	

PROBLEM:

Enter A with -50.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ENA	-50	

PROBLEM:

Enter A with address POPS.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ENA	PØPS	

The ENTER Q Instruction

The ENTER Q instruction is an instruction that transmits a 15-bit quantity Y, where $Y = y + (B^b) + (V^v)$, into the Q register. No memory reference is made.

The 15-bit quantity Y is transmitted right-justified into the Q register with the sign bit extended automatically through the rest of Q.

If the bank term a (within parentheses) is missing, the current operand bank setting remains. If the bank term a is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

If CM is specified by the programmer, the complement of the quantity Y is transmitted to the Q register.

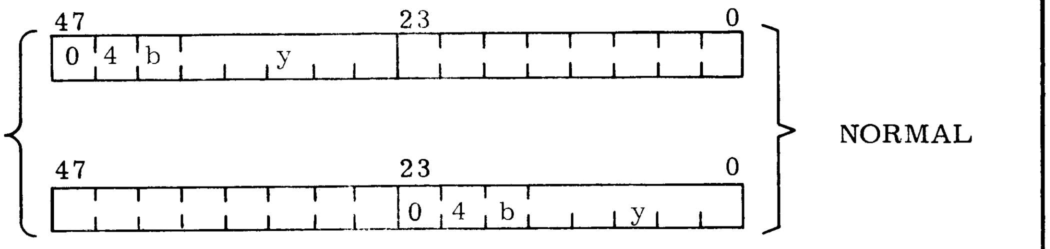
ENTER Q

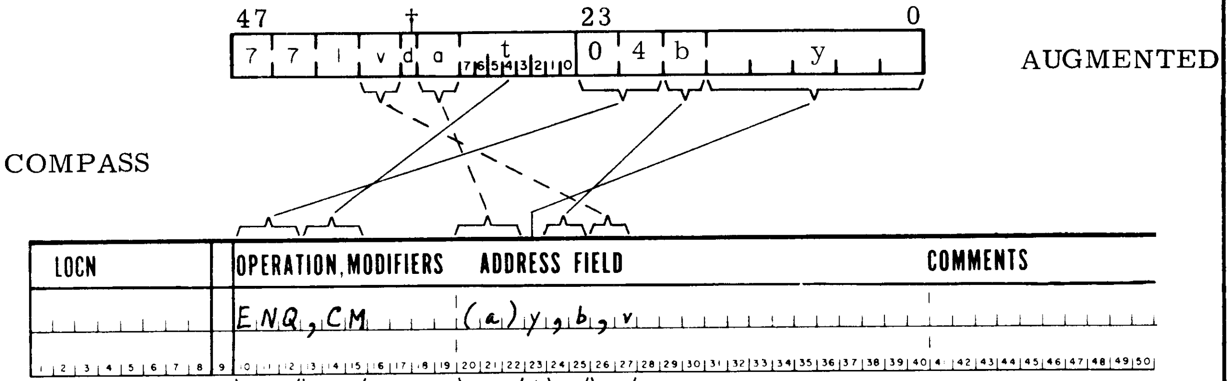
ENQ

FORMAT

MACHINE

May be an upper or lower instruction





Function code

Optional, transmit complement of operand

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base operand. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

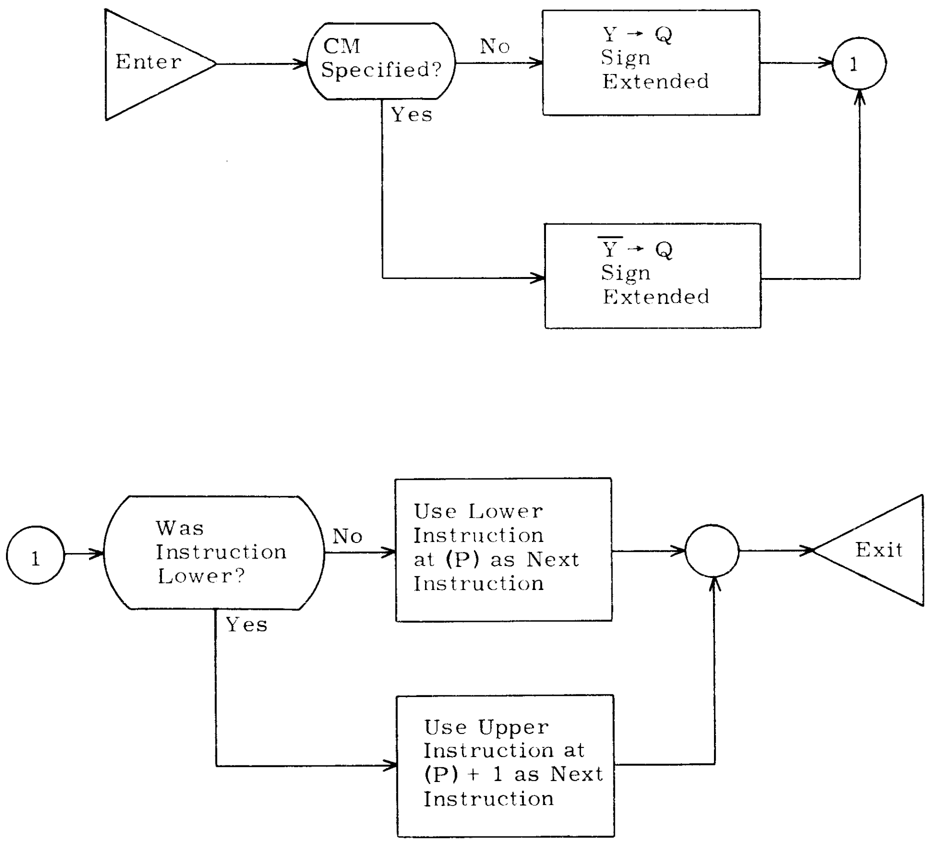
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable.

ENQ

FLOWCHART

ENTER Q FORMAT: ENQ,CM (a) y,b,v $Y = y + (B^b) + (V^v)$



Enter
CM Specified?
No
Yes
Y → Q Sign Extended
$\overline{Y}$ → Q Sign Extended
1
1
Was Instruction Lower?
No
Yes
Use Lower Instruction at (P) as Next Instruction
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

ENQ $Y \rightarrow Q$

ENQ, CM $\overline{Y} \rightarrow Q$

PROBLEM:

Enter Q with five less than the contents of Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ENQ	-5,1	

PROBLEM:

Enter Q with -35 octal.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ENQ	-35B	

PROBLEM:

Enter Q with the sum of Index Registers 1 and 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ENQ	,1,2	

PROBLEM:

Enter Q with address SAM modified by Index Register 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ENQ	SAM,3	

NEW CONCEPTS OF GROUP 2

The instructions in this group are much like those in Group 1. Here, only 15 bits are transferred, and these are usually addresses.

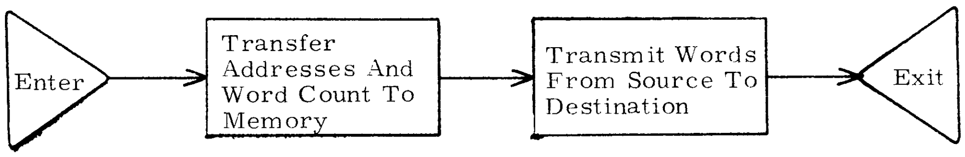
For the first seven instructions the source and destination are 15 bits each so that it is just a matter of transmission. For the last two instructions (ENA and ENQ) the source is 15 bits, but the destination is 48 bits. In this case the entire register is cleared. The 15 bits are then entered into the lowest part of the register (right justified) and the upper bits are a repetition of bit 14.

The symbol ** (double asterisk) is introduced here. This symbol prestores a 15 bit address of all 7's. It represents a "dummy" address that is to be replaced by an actual address during the execution of the program.

Problem 2:

Write a subprogram that will transfer 48-bit words from one area of memory to another. As the subprogram is entered, the initial address is in A lower, the terminal address is in A upper, and the number of words is in Q lower.

Flowchart:


Enter
Transfer Addresses And Word Count To Memory
Transmit Words From Source To Destination
Exit

Problem 2 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	TRANSFER	
	ENTRY	TRANSFER	
TEMP	BSS	1	
TRANSFER	BSS	1	
	SAU	TRANSMIT	INITIAL ADDRESS
	ARS	24	
	SAL	TRANSMIT	TERMINAL ADDRESS
	STQ	TEMP	
	LIL	TEMP, 1	WORD COUNT
CHECKCNT	ISP	TRANSMIT, 1	
	SLJ	TRANSFER	
TRANSMIT	LDA	**, 1	
	STA	**, 1	
	SLJ	CHECKCNT	
	END		

Student Problem 2A:

Write a subprogram that will solve problem 2, but use the XMIT instruction.

Flowchart:

Problem 2A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 3

ADDRESS ARITHMETIC

GROUP 3

ADDRESS ARITHMETIC

1.	Increase A	INA
2.	Increase Index	INI

This group of instructions performs an addition on the contents of A or the contents of an index register. No storage cycle is required.

The INCREASE A Instruction

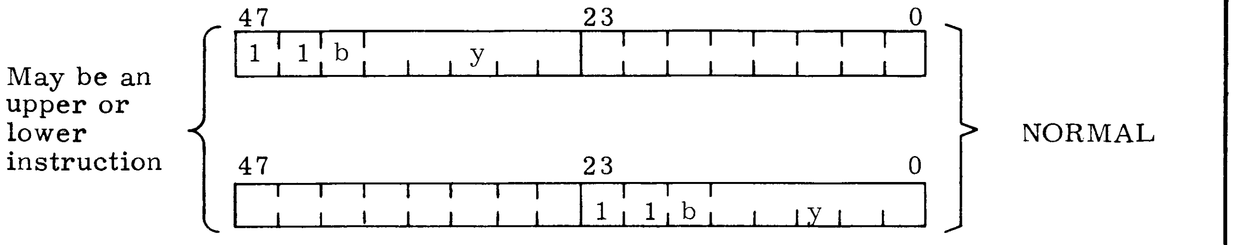
The INCREASE A instruction is an instruction that adds a 15-bit quantity $\underline{Y}$, where $Y = y + (B^b) + (V^v)$, to the contents of A. The sign of Y is extended to 48 bits before the addition takes place. No memory reference is made.

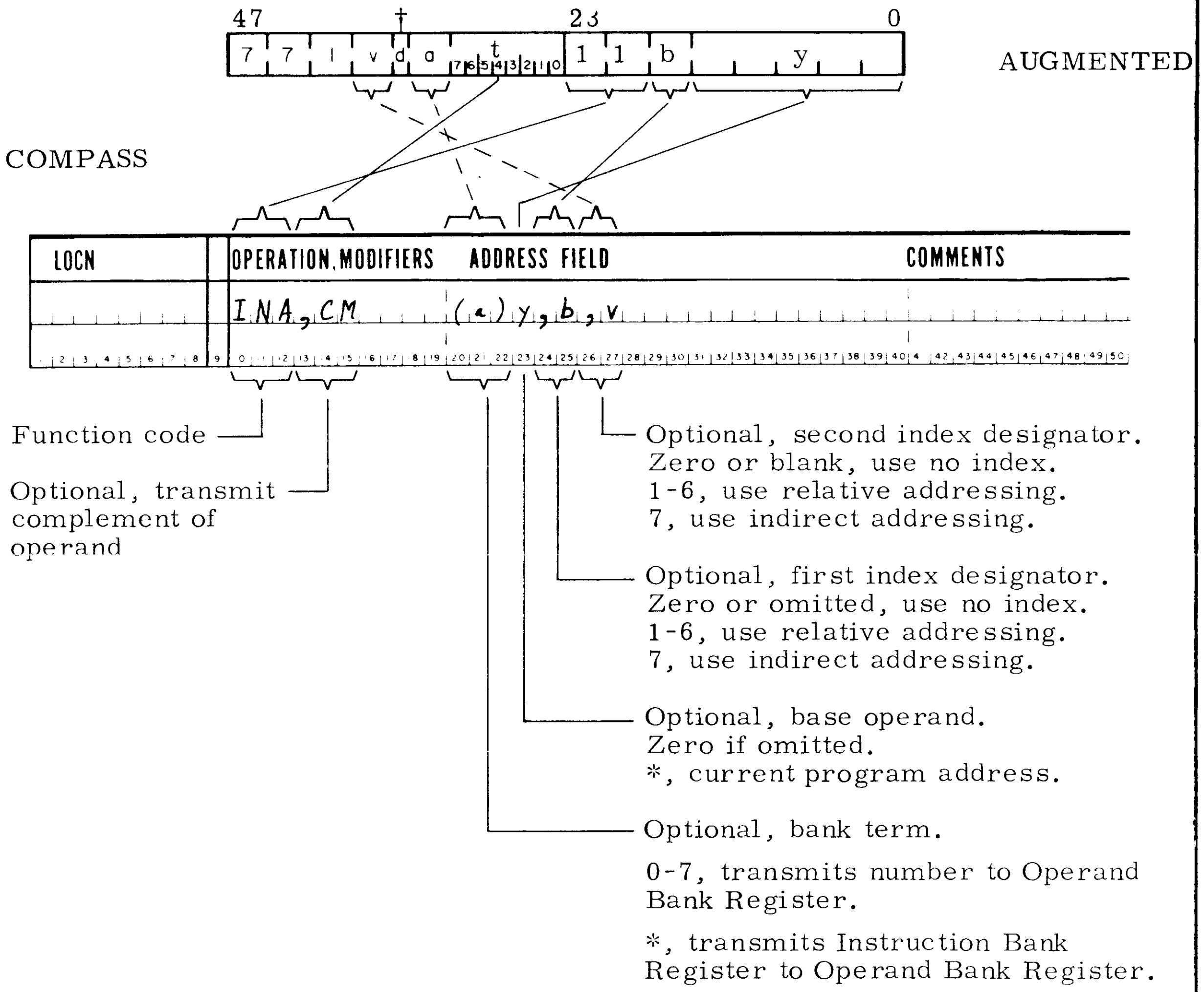
If the bank term $\underline{a}$ (within parentheses) is missing, the current operand bank setting remains. If the bank term $\underline{a}$ is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

If CM is specified by the programmer, the complement of the 15-bit quantity is added to the contents of A.

FORMAT

MACHINE



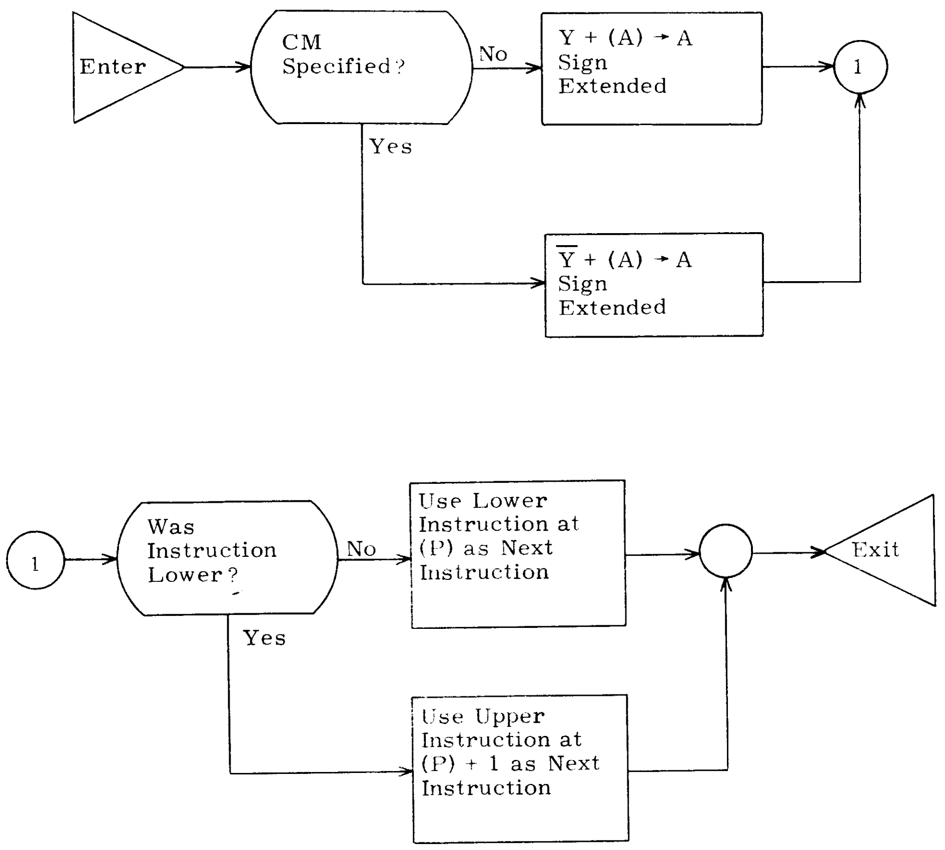

47
†
23
0
7
7
1
v
d
a
t
7 6 5 4 3 2 1 0
1
1
b
y
AUGMENTED
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
INA, CM (*) y, b, v
Function code
Optional, transmit complement of operand
Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.
Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.
Optional, base operand.
Zero if omitted.
*, current program address.
Optional, bank term.
0-7, transmits number to Operand Bank Register.
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

INA

FLOWCHART

INCREASE A FORMAT: INA, CM (a) y,b,v $Y = y + (B^{b}) + (v^{*})$


Enter
CM Specified?
No
Yes
Y + (A) → A Sign Extended
$\overline{Y}$ + (A) → A Sign Extended
1
1
Was Instruction Lower?
No
Yes
Use Lower Instruction at (P) as Next Instruction
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

INA	(A) + Y → A
INA, CM	(A) + $\overline{Y}$ → A

PROBLEM:

Increase the contents of A by two.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		INA	2	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Increase the contents of A by the contents of Index Register 1.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		INA	,1	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Decrement the contents of A by one.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		INA	-1	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Increase the contents of A by 77 octal.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		INA	77B	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The INCREASE INDEX Instruction

The INCREASE INDEX instruction is an instruction that adds a 15-bit quantity $\underline{Y}$, where $Y = y + (V^V)$, to the contents of an index register specified by index designator $\underline{b}$. No memory reference is made.

If the bank term $\underline{a}$ (within parentheses) is missing, the current operand bank setting remains. If the bank term $\underline{a}$ is used, the current operand bank setting will be replaced by the value $\underline{a}$. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

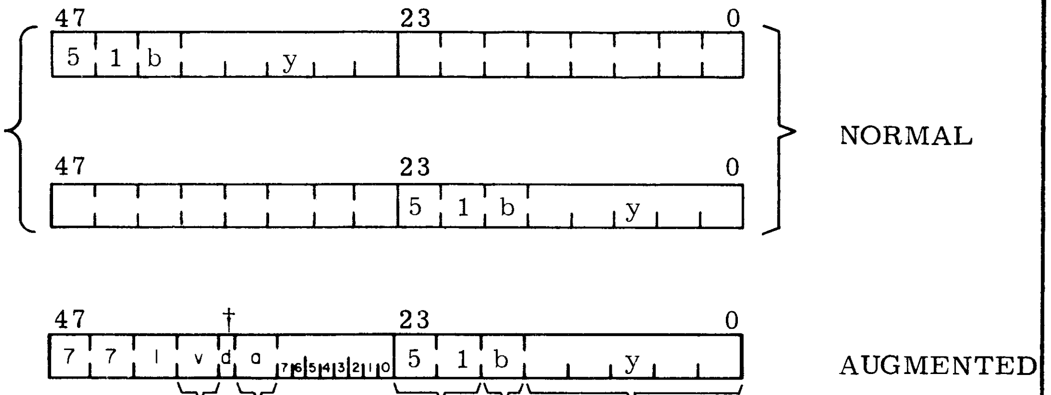
INCREASE INDEX

INI

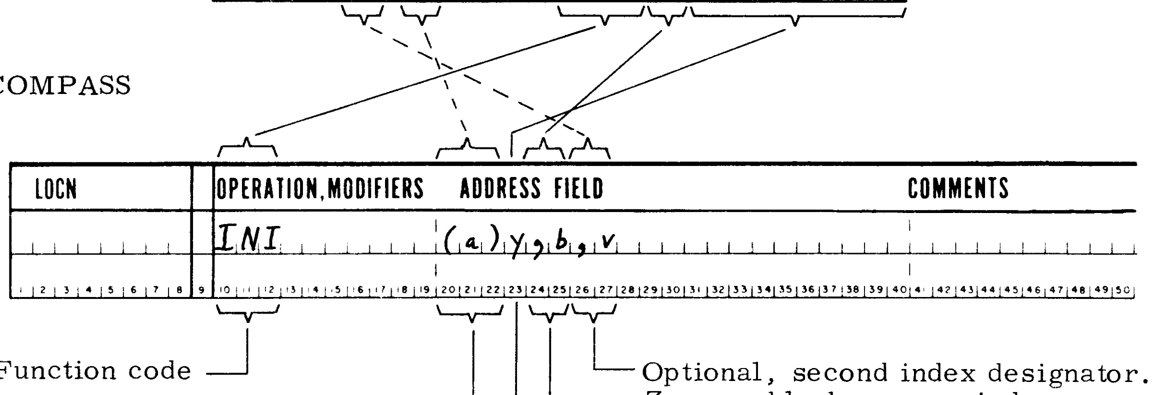
FORMAT

MACHINE

May be an upper or lower instruction



COMPASS



Function code

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base operand. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

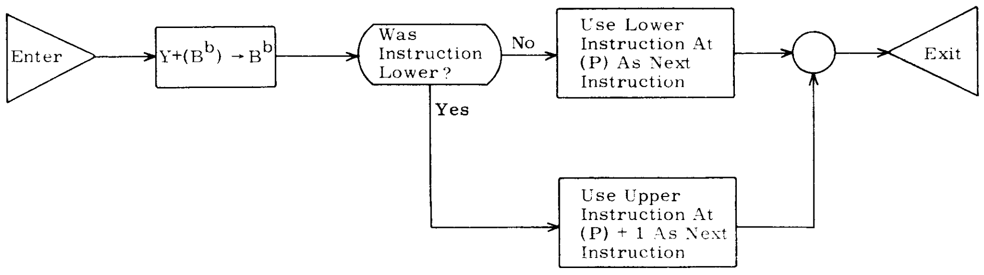
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

INI

F L O W C H A R T

INCREASE INDEX FORMAT: INI (a)y, b, v $Y = y + (V^v)$


Enter
$Y+(B^b) \rightarrow B^b$
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

D E S C R I P T I O N

INI $(B^b) + Y \rightarrow B^b$

PROBLEM:

Increase the contents of Index Register 1 by one.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		INI	1,1	

PROBLEM:

Decrease the contents of Index Register 2 by one.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		INI	-1,2	

PROBLEM:

Subtract 33B from the contents of Index Register 3.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		INI	-33B,3	

PROBLEM:

Increase the contents of Index Register 4 by SAM.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		INI	SAM,4	

NEW CONCEPTS OF GROUP 3

This group consists of fast arithmetic instructions that add positive or negative numbers to the A or index registers. The arithmetic is 1's complement.

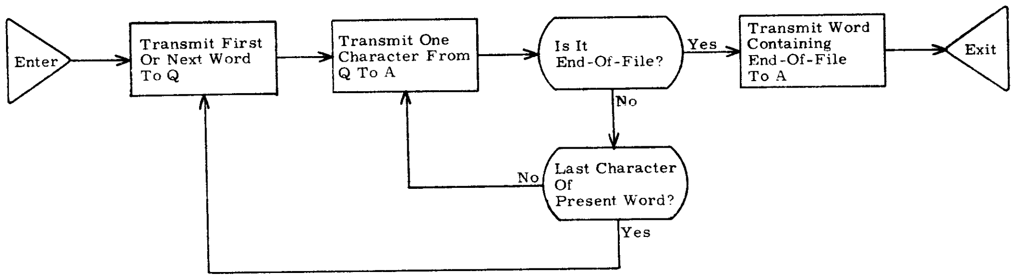
Since the number added to the register is only 15 bits, the number added is limited to ±16, 383. You can INA 5, INA -329, but not INA 543216. For this, you would have to ADD the quantity where the quantity resides in memory.

Another new concept introduced at this time is the suffixing a number with the letter B. Any number suffixed with a B is assumed octal. Examples are: 13B, -246B, and 346715B. If no B suffixes the number, the number is assumed decimal. Examples are: 59, 63, and -3456. 1348B is illegal.

Problem 3:

Numerous files containing BCD characters have been read into core starting at address BUF. An end-of-file in core is signified by 17B. Write a subprogram that will search for the first end-of-file. When it is found, load the word that contains the end-of-file into A and exit.

Flowchart:


Enter
Transmit First Or Next Word To Q
Transmit One Character From Q To A
Is It End-Of-File?
Yes
No
Transmit Word Containing End-Of-File To A
Exit
Last Character Of Present Word?
No
Yes

Problem 3 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	FILE	
	ENTRY	FILE	
FILE	BSS	1	
	ENI	0,1	ADDRESS CØUNTER
	ENI	0,2	
NEXTWØRD	LDQ	BUF,1	
NEXTCHAR	ENA	0	
	LLS	6	
	INA	-17B	
	AJP,ZR	FØUND	
	ISK	7,2	LAST CHARACTER?
	SLJ	NEXTCHAR	NØ
	INI	1,1	YES, BUMP ADDRESS CØUNTER
	SLJ	NEXTWØRD	
FØUND	LDA	BUF,1	
	SLJ	FILE	
	END		

Student Problem 3A:

Assume 1000 words of BCD characters with numerous end-of-files (17B) in memory starting at address COMPILE Count the number of end-of-files and store it at address FILE.

Flowchart:

Problem 3A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 4

INDEXING

GROUP 4

INDEXING

1. Index Skip	ISK
2. Index Jump	IJP

This group of instructions tests the contents of an index register for being a predetermined value. If it is this value, the program continues. If it is not this value, the contents of the index register are automatically incremented or decremented by one depending on the instruction.

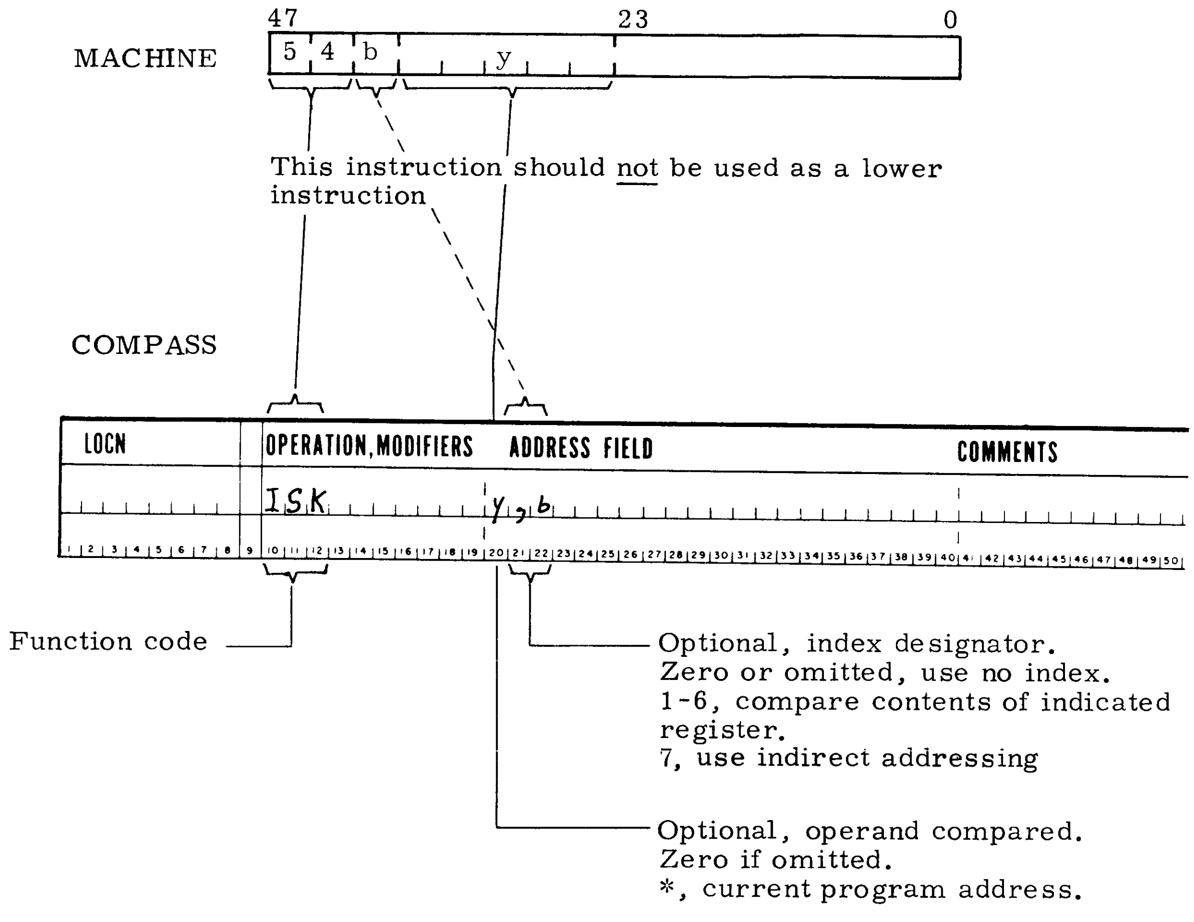
These instructions serve a two-fold purpose. First, they can be used as loop counters. Second, the same index register can be used as a modifier each time the loop is entered. In this way a programmer can process a table in the forward or backward direction.

The INDEX SKIP Instruction

The INDEX SKIP instruction is an instruction that compares the quantity y with the contents of an index register specified by the index designator b. If the two quantities are equal, a full exit is taken with the contents of the index register being cleared. If the two quantities are not equal, a half exit is taken with the contents of the index register incremented by one.

Because of the ability of this instruction to half exit or full exit, the COMPASS assembler will automatically force this instruction to the upper position of a memory word. It would have no meaning if it were assembled in the lower position of the memory word. For this reason the programmer should not use the bank term a (within parentheses) or the v index designator, because by doing so, the assembler must force the instruction lower which defeats its purpose.

FORMAT

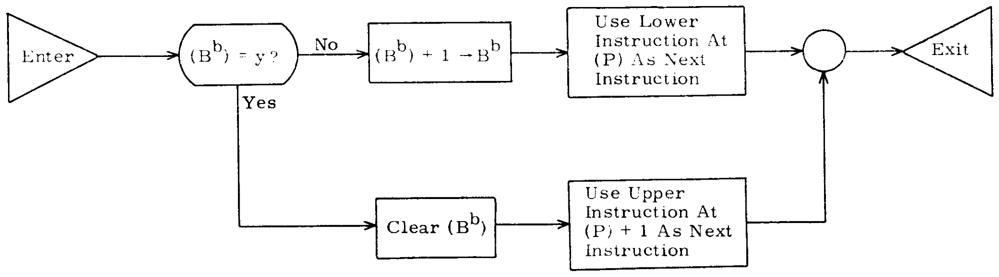

MACHINE
47
23
0
5
4
b
y
This instruction should not be used as a lower instruction
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
ISK
y, b
Function code
Optional, index designator.
Zero or omitted, use no index.
1-6, compare contents of indicated register.
7, use indirect addressing
Optional, operand compared.
Zero if omitted.
*, current program address.

ISK

F L O W C H A R T

INDEX SKIP*

FORMAT: ISK y, b


Enter
$(B^b) = y$?
No
Yes
$(B^b) + 1 \rightarrow B^b$
Use Lower Instruction At (P) As Next Instruction
Exit
Clear (B^b)
Use Upper Instruction At (P) + 1 As Next Instruction

*Forced upper by the assembler

D E S C R I P T I O N

ISK $(B^b) = y$?

Yes, clear B^b and full exit

No, $(B^b) + 1 \rightarrow B^b$ and half exit

E
X
A
M
P
L
E
S

PROBLEM:

Examine the contents of Index Register 1 for being equal to 100.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ISK	100,1	

PROBLEM:

Examine the contents of Index Register 2 for being equal to 1000 octal.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ISK	1000B,2	

PROBLEM:

Examine the contents of Index Register 3 for being equal to 16000.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ISK	16000,3	

PROBLEM:

Examine the contents of Index Register 4 for being equal to 3000 octal.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ISK	3000B,4	

The INDEX JUMP Instruction

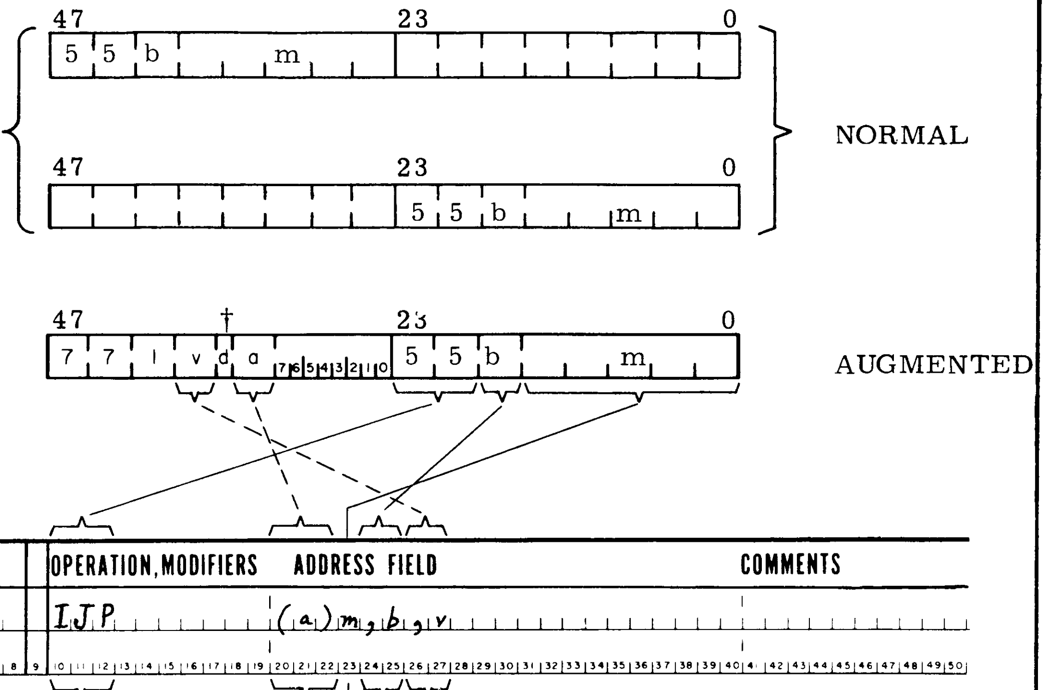
The INDEX JUMP instruction is an instruction that compares the contents of an index register, specified by the index designator b, with zero. If the content of the index register is zero, program control continues to the next instruction. If the content of the index register is not zero, program control will jump to address M where M = m + (V^V) and the index register will be decremented by one.

If the bank term a (within parentheses) is missing, the current operand bank setting remains. If the bank term a is used, the current operand bank setting will be replaced by the value a. In either case it will not affect (or, cannot jump banks with) this instruction. However, it could affect future instructions that reference memory for operands.

F O R M A T

MACHINE

May be an upper or lower instruction



COMPASS

Function code

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, index register compared.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

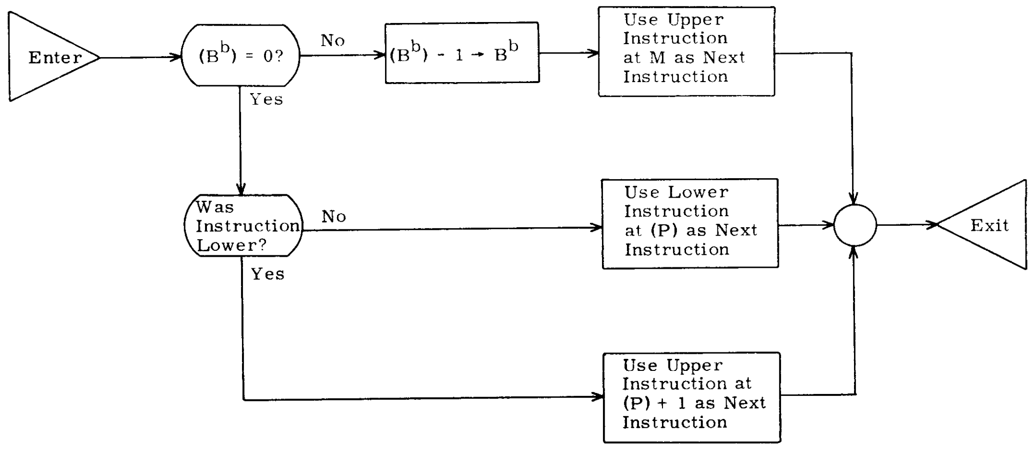
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

IJP

F L O W C H A R T

INDEX JUMP FORMAT: IJP (a) m, b, v $M = m + (V^v)$



Enter
$(B^b) = 0?$
No
$(B^b) - 1 \rightarrow B^b$
Use Upper Instruction at M as Next Instruction
Yes
Was Instruction Lower?
No
Use Lower Instruction at (P) as Next Instruction
Exit
Yes
Use Upper Instruction at (P) + 1 as Next Instruction

D E S C R I P T I O N

IJP $(B^b) = 0?$

Yes, continue next instruction

No, $(B^b) - 1 \rightarrow B^b$ and jump to M

EXAMPLES

PROBLEM:

Examine the contents of Index Register 1 for being equal to zero. If zero, continue executing. If non-zero, jump to address LOOP.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IJP	LØØP,1	

PROBLEM:

Examine the contents of Index Register 2 for being equal to zero. If zero, continue executing. If non-zero, jump to address LOOP modified by Index Register 3.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IJP	LØØP,2,3	

PROBLEM:

Examine the contents of Index Register 4 for being equal to zero. If zero, continue executing. If non-zero, jump to the address specified in Index Register 5.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IJP	,4,5	

PROBLEM:

Examine the contents of Index Register 6 for being equal to zero. If zero, continue executing. If non-zero, jump to an address five memory locations backward.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IJP	*-5,6	

NEW CONCEPTS OF GROUP 4

Introduced here is a concept of "looping". The computer has index registers that are used to form address modification for relative addressing. They can also be used to determine the number of times a routine is performed.

The ISK instruction checks a counter (contents of an index register) for being equal to a pre-set value. If the two are not equal, the counter is advanced by 1 and a half exit is taken, allowing the programmer to repeat the loop. The programmer can use the same index register as an address modifier within the loop so that he may process data sequentially. If the two are equal, a full exit is taken. This skips the lower instruction and terminates the loop.

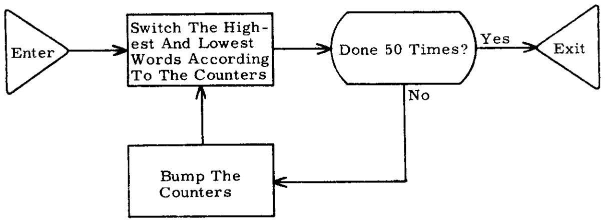
The IJP instruction checks a counter (contents of an index register) for being equal to zero. If the counter is not zero, it is decremented by 1 and a jump takes place to re-enter the loop. If the counter is zero, the next instruction is executed and the loop is terminated.

Also introduced at this time is the * (asterisk) used in the address field in place of an address. It means the present value of the location counter. Hardware-wise it would mean the contents of the P register when the instruction is executed. * + 5 means 5 memory addresses forward (not 5 instructions). * -10 means 10 (decimal) memory addresses backward.

Problem 4:

A table of 100 words exists in memory, but upside down. Write a subprogram that will re-arrange the words so that the table is right side up. The starting address of the table is TAB.

Flowchart:



Enter
Switch The Highest And Lowest Words According To The Counters
Done 50 Times?
Yes
Exit
No
Bump The Counters

Problem 4 could be solved by coding in the following manner:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	SWITCH	
	ENTRY	SWITCH	
SWITCH	BSS	1	
	ENI	0, 1	CØUNTER FØR FIRST WØRD
	ENI	49, 2	CØUNTER FØR LAST WØRD
NEXT	LDA	TAB, 1	
	LDQ	TAB+50, 2	
	LDQ	TAB+50, 2	
	STQ	TAB, 1	
	IJP	*+1, 2	
	UJP	SWITCH	(B2) = 0, FINISH
	INI	1, 1	
	UJP	NEXT	
	END		

Somewhere within this subprogram would also be included the symbol TAB in the location field with a declaration of the prestored data or the area reserved.

Student Problem 4A:

There exists in memory 30 words of data starting at address SIGMA. Write a subprogram that will move the first word to the bottom of the list and move the other words up one address.

Flowchart:

Problem 4A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 5

FIXED POINT ARITHMETIC

GROUP 5

FIXED POINT ARITHMETIC

1.	Add	ADD
2.	Subtract	SUB
3.	Multiply Integer	MUI
4.	Divide Integer	DVI
5.	Multiply Fractional	MUF
6.	Divide Fractional	DVF

This group of instructions performs the fixed point arithmetic operations on the A and Q registers.

The first four instructions use purely integer arithmetic where the point is assumed to be to the right of the register or register set. The last two use purely fractional arithmetic where the point is assumed to be just to the right of the sign bit.

The ADD Instruction

The ADD instruction is an instruction that adds a 48-bit operand (from an 18-bit storage) to the contents of A. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the A register are replaced by the sum of the two operands.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

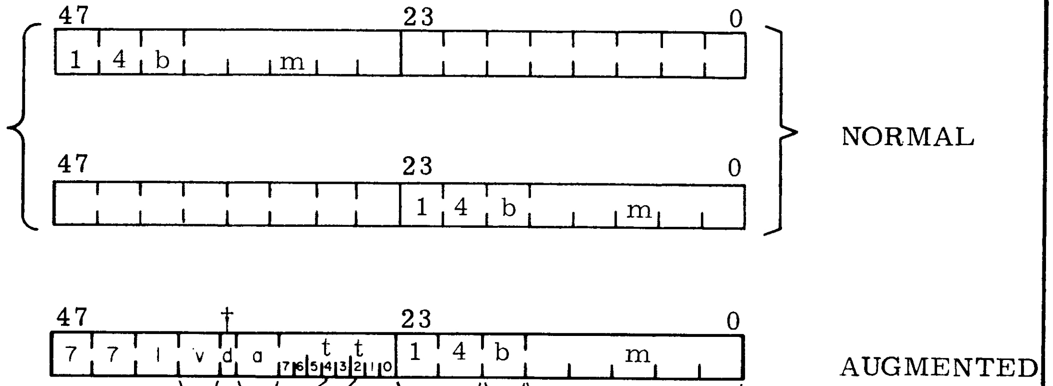
If CM is specified by the programmer, the complement of the operand is added to the contents of A.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is added to the contents of A.

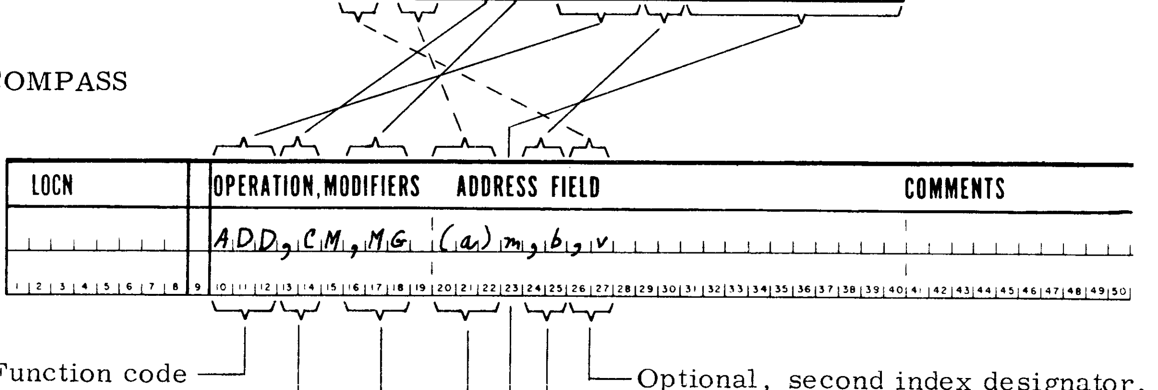
FORMAT

MACHINE

May be an upper or lower instruction



COMPASS



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

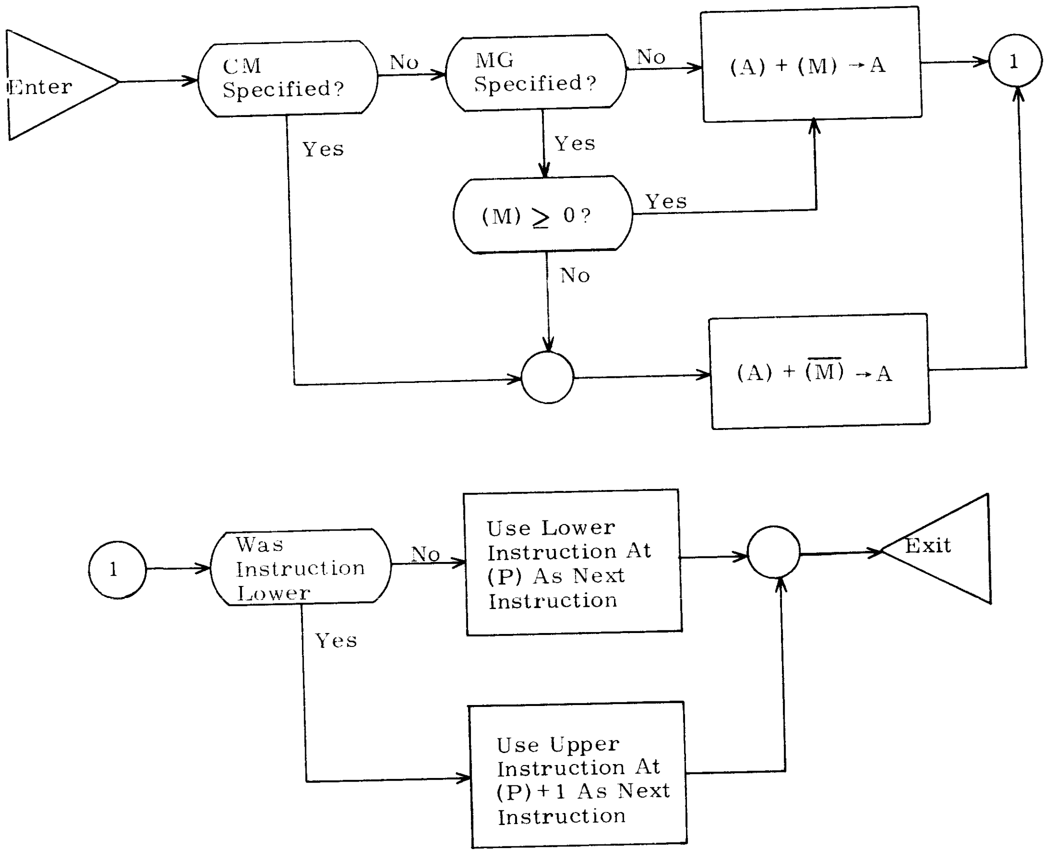
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

ADD

FLOWCHART

ADD FORMAT: ADD, CM, MG (a) m,b,v $M = m + (B^b) + (V^v)$


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(A) + (M) → A
1
(M) ≥ 0 ?
Yes
No
(A) + $\overline{(M)}$ → A
1
Was Instruction Lower
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

ADD	1) $(A) + (M) \rightarrow A$
	2) Contents of M remain unchanged
ADD, CM	1) $(A) + \overline{(M)} \rightarrow A$
	2) Contents of M remain unchanged
ADD, MG	1) $(A) + \lvert (M) \rvert \rightarrow A$
	2) Contents of M remain unchanged

PROBLEM:

Add the contents of address SMALLEST to A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ADD	SMALLEST	

PROBLEM:

Add the contents of address BIGGEST modified by Index Register 1 to A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ADD	BIGGEST,1	

PROBLEM:

Add the absolute value of the contents of address FIND to A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ADD,MG	FIND	

PROBLEM:

Add the contents of address NOFIND modified by Index Registers 3 and 4 to A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ADD	NØFIND,3,4	

The SUBTRACT Instruction

The SUBTRACT instruction is an instruction that subtracts a 48-bit operand (from an 18-bit storage address) from the contents of A. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the A register are replaced by the difference of the two operands.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is subtracted from the contents of A.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is subtracted from the contents of A.

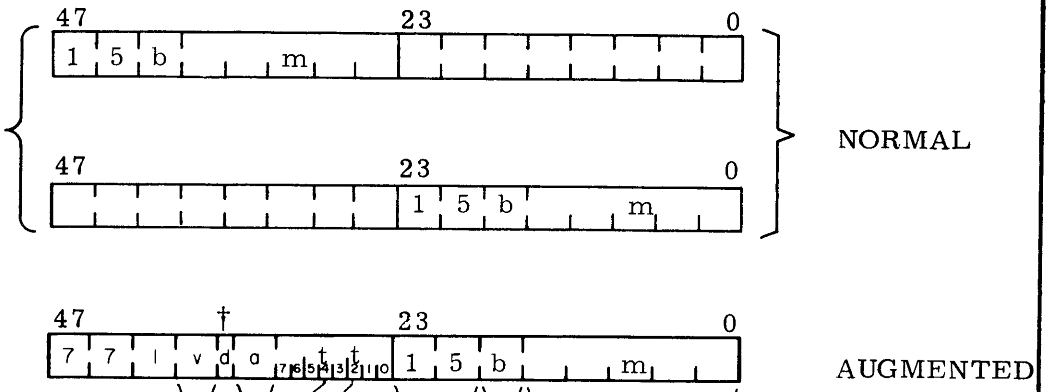
SUBTRACT

SUB

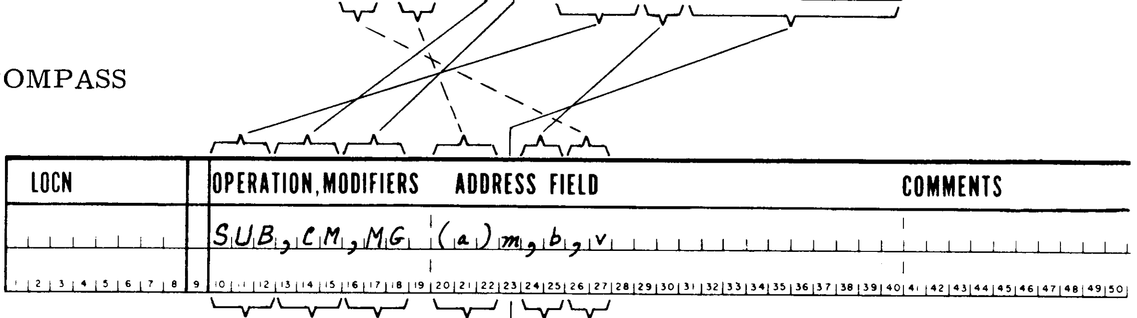
FORMAT

MACHINE

May be an upper or lower instruction



COMPASS



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

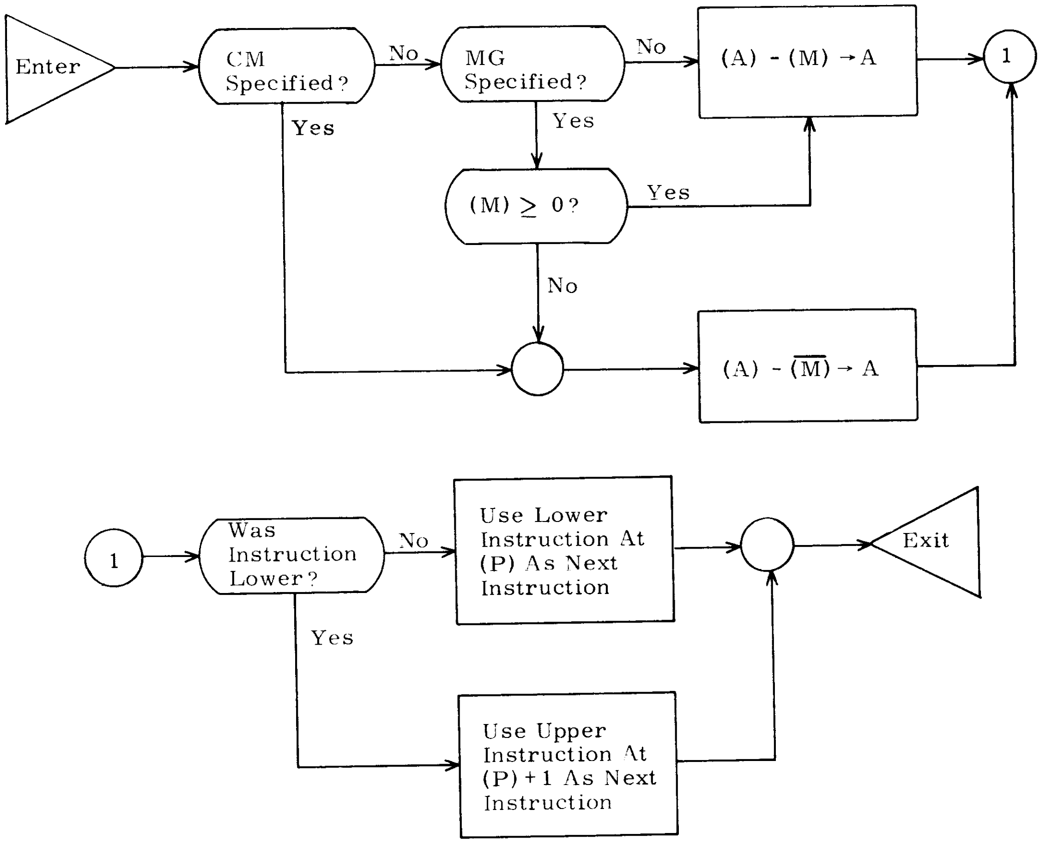
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

SUB

F L O W C H A R T

SUBTRACT FORMAT: SUB, CM, MG (a)m,b,v $M = m + (B^b) + (V^v)$


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(A) - (M) → A
1
(M) ≥ 0?
Yes
No
(A) - $\overline{(M)}$ → A
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

D E S C R I P T I O N

SUB	1) $(A) - (M) \rightarrow A$
	2) Contents of M remain unchanged
SUB, CM	1) $(A) - \overline{(M)} \rightarrow A$
	2) Contents of M remain unchanged
SUB, MG	1) $(A) - \lvert(M)\rvert \rightarrow A$
	2) Contents of M remain unchanged

EXAMPLES

PROBLEM:

Subtract the contents of address MIN from A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SUB	MIN	

PROBLEM:

Subtract the contents of address MIN modified by Index Register 3 from A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SUB	MIN,3	

PROBLEM:

Subtract the absolute value of the contents of address TOPS from A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SUB,MG	TØPS	

PROBLEM:

Subtract the contents of address SPOT modified by Index Registers 3 and 4.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SUB	SPØT,3,4	

The MULTIPLY INTEGER Instruction

The MULTIPLE INTEGER instruction is an instruction that multiplies a 48-bit operand (from an 18-bit storage address) by the contents of A in fixed-point format. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the QA registers are replaced by the 96-bit product of the two operands. The point is assumed to be to the right of A with sign extended through Q.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is multiplied by the contents of A.

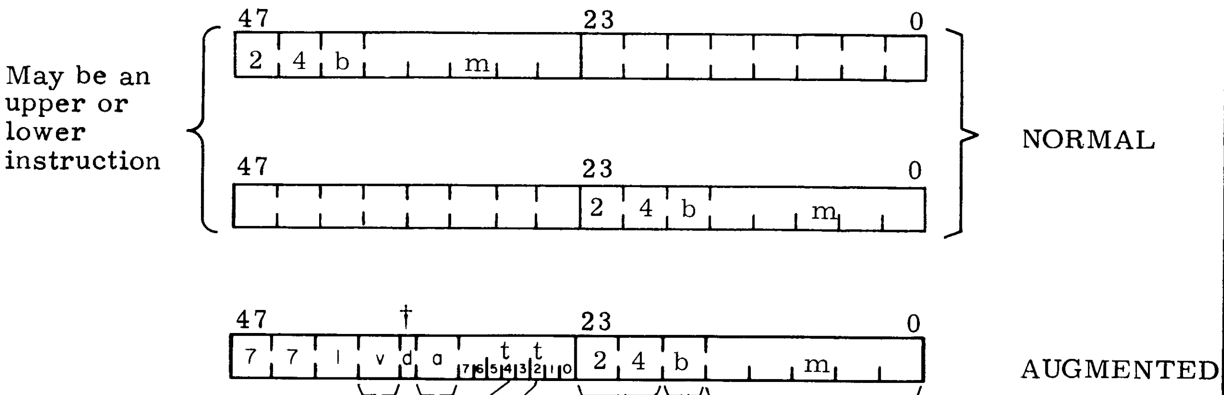
If MG is specified by the programmer, the magnitude (absolute value) of the operand is multiplied by the contents of A.

MULTIPLY INTEGER

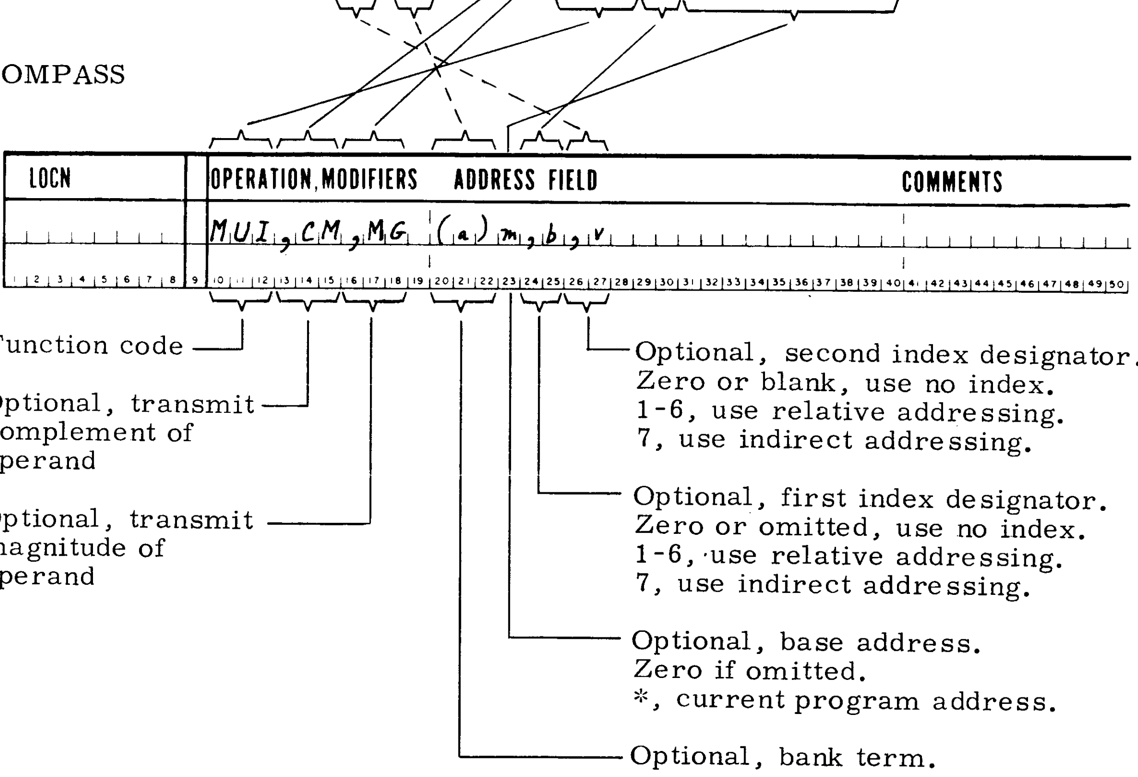
MUI

FORMAT

MACHINE



COMPASS



Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

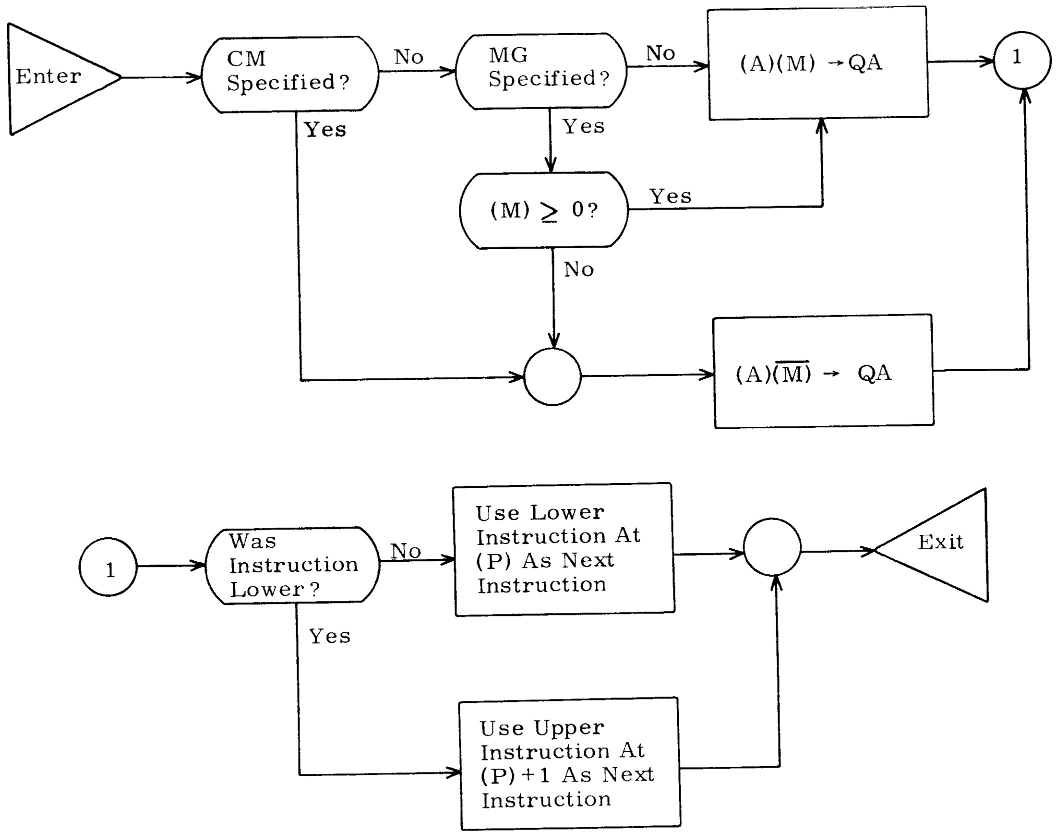
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

MUI

MULTIPLY INTEGER FORMAT: MUI, CM, MG (a)m, b, v $M = m + (B^b) + (V^v)$

FLOWCHART



DESCRIPTION

MUI	1) (A) (M) → QA
	2) Contents of M remain unchanged
MUI, CM	1) (A) $\overline{(M)}$ → QA
	2) Contents of M remain unchanged
MUI, MG	1) (A) $\lvert(M)\rvert$ → QA
	2) Contents of M remain unchanged

PROBLEM:

Multiply A by the contents of address SUR.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	MUI	SUR	

PROBLEM:

Multiply A by the contents of address PETE modified by Index Register 1.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	MUI	PETE, 1	

PROBLEM:

Multiply A by the absolute value of the contents of address JACK modified by Index Register 2.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	MUI, MG	JACK, 2	

PROBLEM:

Multiply A by the contents of HOMER from the bank where HOMER resides.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	MUI	($)HØMER	

The DIVIDE INTEGER Instruction

The DIVIDE INTEGER instruction is an instruction that divides the contents of QA by a 48-bit operand from an 18-bit storage address in fixed-point format. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the A register are replaced by the quotient of the two operands. The initial contents of the Q register are replaced by the remainder of the operation.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

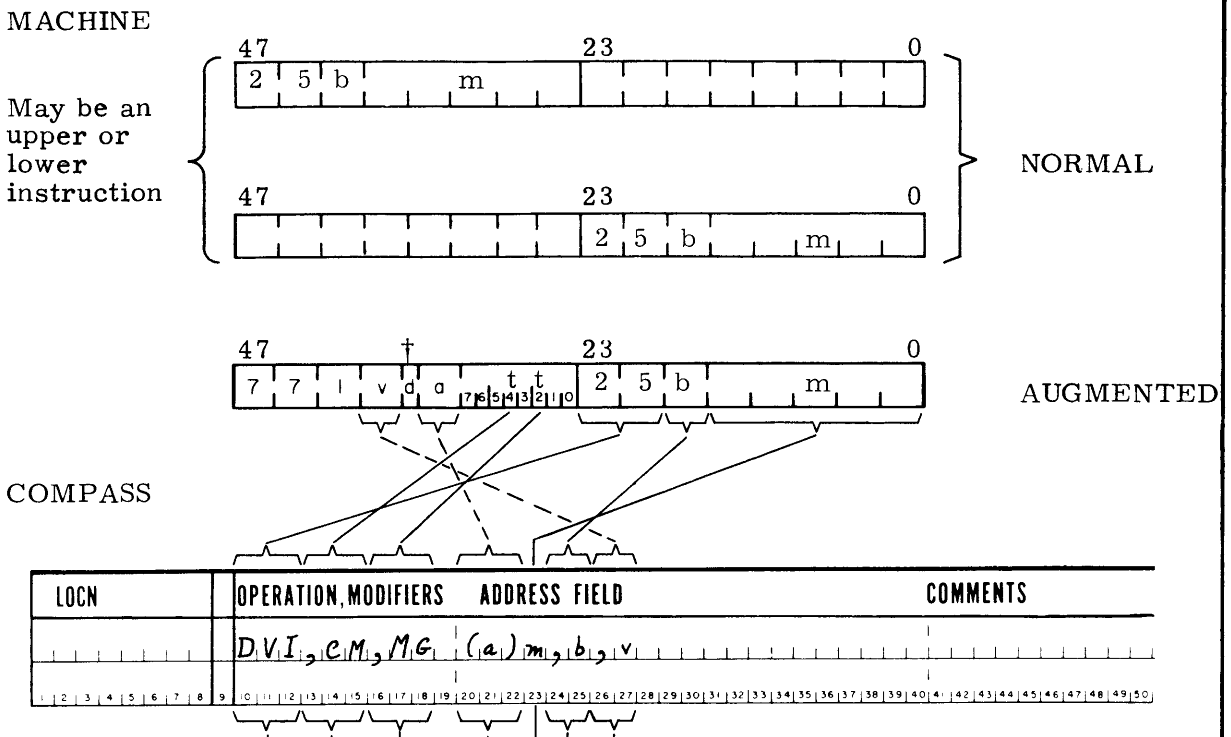
If CM is specified by the programmer, the complement of the operand from memory is used as the divisor.

If MG is specified by the programmer, the magnitude (absolute value) of the operand from memory is used as the divisor.

DIVIDE INTEGER

DVI

FORMAT


MACHINE
May be an upper or lower instruction
47
23
0
2 5 b m
NORMAL
AUGMENTED
7 7 I v d a t t 7 6 5 4 3 2 1 0
†
COMPASS
LOCN
OPERATION, MODIFIERS ADDRESS FIELD
COMMENTS
DVI, CM, MG (a) m, b, v

Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

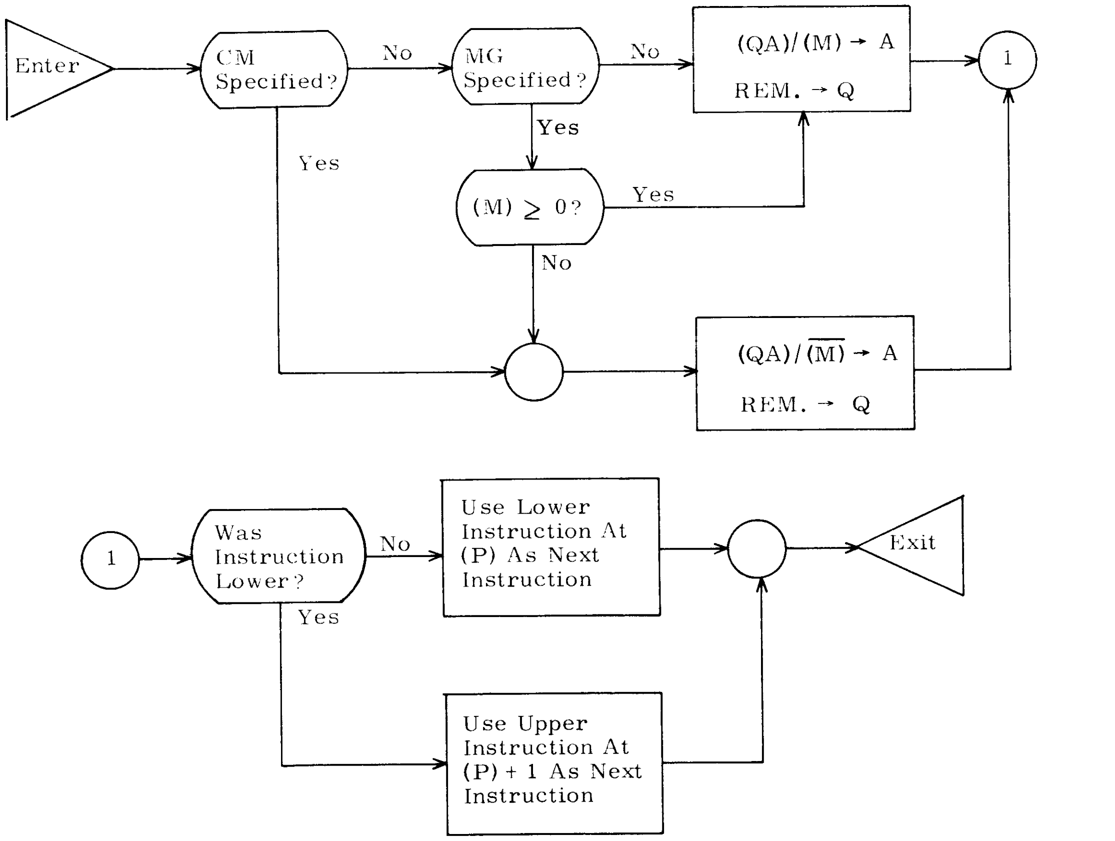
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

DVI

FLOWCHART

DIVIDE INTEGER FORMAT: DVI, CM, MG (a) m, b, v $M = m + (B^b) + (V^v)$



Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(M) ≥ 0?
Yes
No
(QA)/(M) → A
REM. → Q
(QA)/$\overline{(M)}$ → A
REM. → Q
1
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

DESCRIPTION

DVI	1) (QA) / (M) → A, Remainder → Q
	2) Contents of M remain unchanged
DVI, CM	1) (QA) / $\overline{(M)}$ → A, Remainder → Q
	2) Contents of M remain unchanged
DVI, MG	1) (QA) / $|(M)|$ → A, Remainder → Q
	2) Contents of M remain unchanged

PROBLEM:

Divide QA by the contents of address DIVIS.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	DVI	DIVIS	

PROBLEM:

Divide QA by the contents of TRAVIS modified by Index Register 3.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	DVI	TRAVIS, 3	

PROBLEM:

Divide QA by the contents of the address specified in Index Register 4.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	DVI	, 4	

PROBLEM:

Divide QA by the contents of address GUBER from the bank that DVI is in.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	DVI	(*) GUBER	

The MULTIPLY FRACTIONAL Instruction

The MULTIPLY FRACTIONAL instruction is an instruction that multiplies a 48-bit operand (from an 18-bit storage address) by the contents of A in fractional format. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the AQ registers are replaced by the 96-bit product of the two operands. The point is assumed to be to the right of the sign bit in A with the fraction extended through Q.

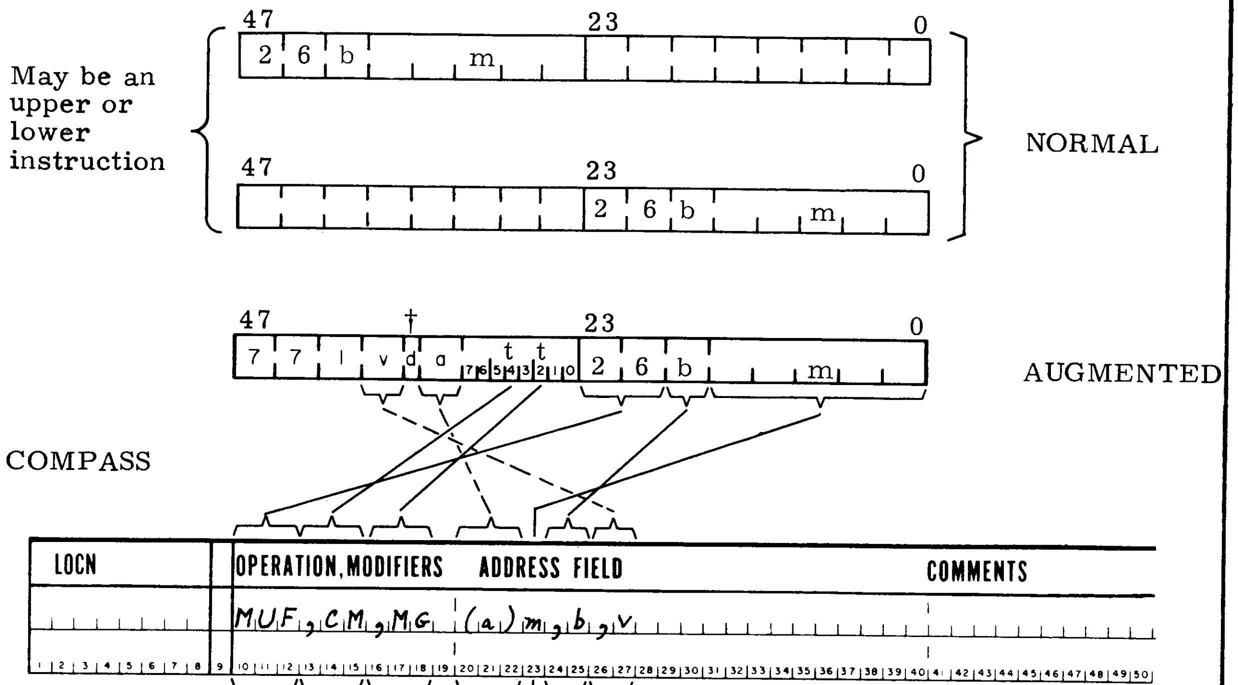
The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is multiplied by the contents of A.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is multiplied by the contents of A.

FORMAT

MACHINE



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

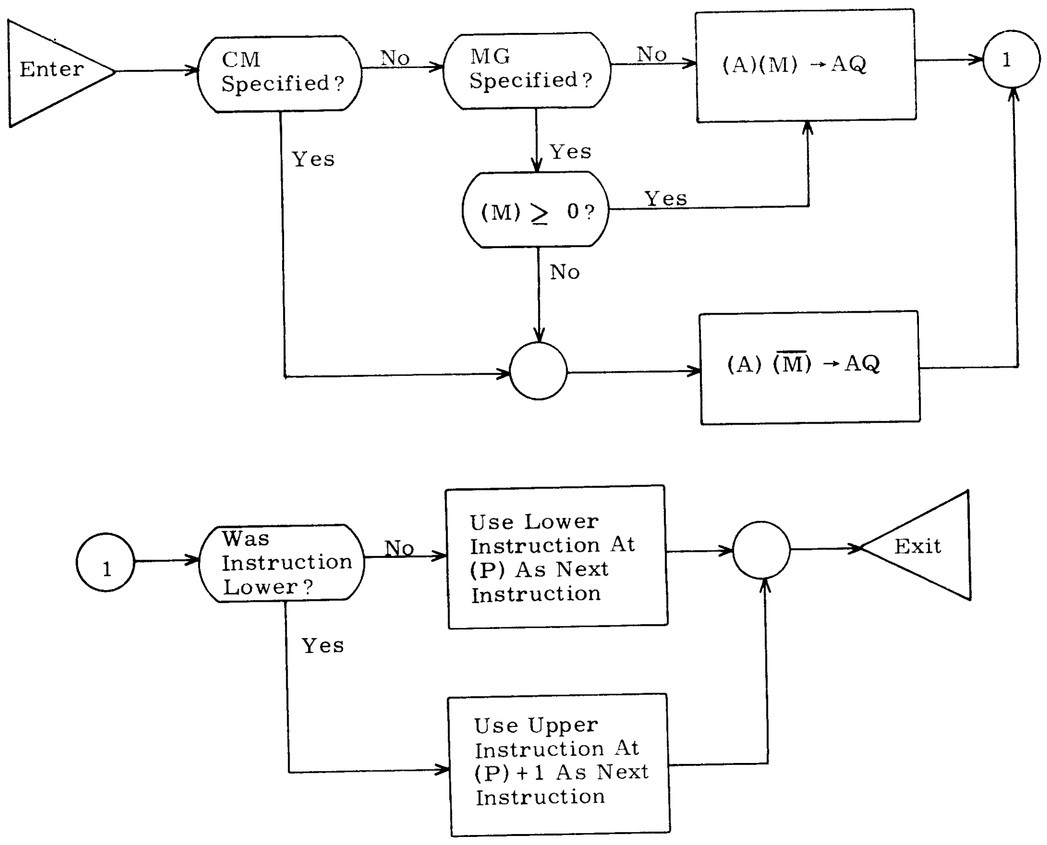
*, transmits Instruction Bank Register to Operand Bank Register.

[†] d bit not programmable

MUF

MULTIPLY FRACTIONAL FORMAT: MUF,CM,MG (a)m,b,v $M = m + (B^b) + (V^v)$

F L O W C H A R T


Enter
CM Specified?
No
MG Specified?
No
(A)(M) → AQ
1
Yes
Yes
(M) ≥ 0?
Yes
No
(A) $\overline{(M)}$ → AQ
1
Was Instruction Lower?
No
Use Lower Instruction At (P) As Next Instruction
Exit
Yes
Use Upper Instruction At (P)+1 As Next Instruction

D E S C R I P T I O N

MUF	1) (A) (M) → AQ
	2) Contents of M remain unchanged
MUF, CM	1) (A) $\overline{(M)}$ → AQ
	2) Contents of M remain unchanged
MUF, MG	1) (A) $\lvert(M)\rvert$ → AQ
	2) Contents of M remain unchanged

PROBLEM:

Multiply A by the contents of address PYLE.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	MUF	PYLE	

PROBLEM:

Multiply A by the contents of address PIKE modified by Index Register 2.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	MUF	PIKE,2	

PROBLEM:

Multiply A by the contents of the address specified in Index Register 4.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	MUF	,4	

PROBLEM:

Multiply A by the absolute value of the contents of address MOOSE.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	MUF,MG	MØØSE	

The DIVIDE FRACTIONAL Instruction

The DIVIDE FRACTIONAL instruction is an instruction that divides the contents of AQ by a 48-bit operand from an 18-bit storage address in fractional format. One memory reference is made.

The operation leaves the contents of the storage address unchanged. The initial contents of the A register are replaced by the quotient of the two operands. The initial contents of the Q register are replaced by the residue of the operation.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

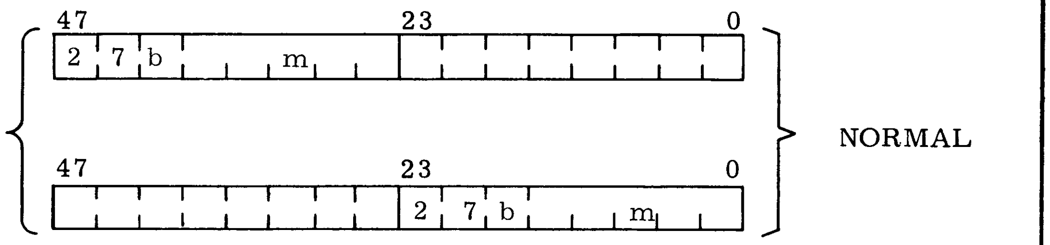
If CM is specified by the programmer, the complement of the operand from memory is used as the divisor.

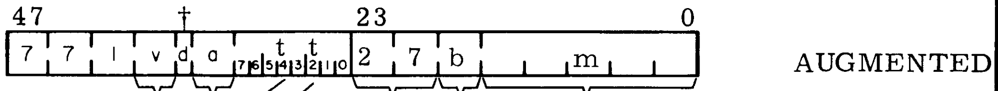
If MG is specified by the programmer, the magnitude (absolute value) of the operand from memory is used as the divisor.

FORMAT

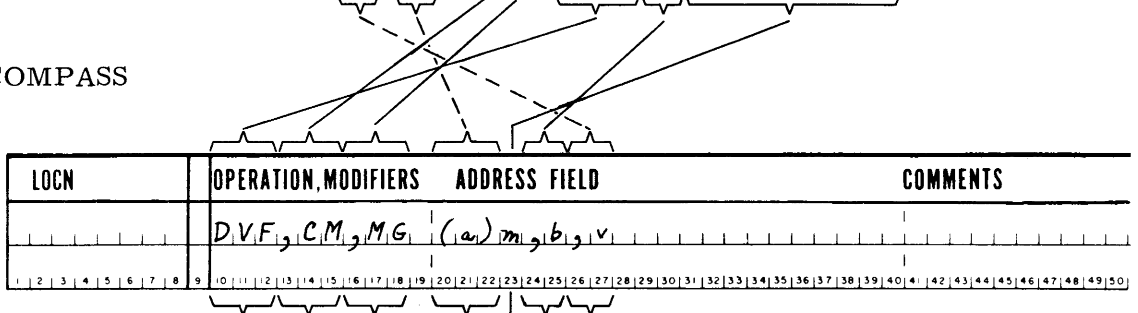
MACHINE

May be an upper or lower instruction





COMPASS



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

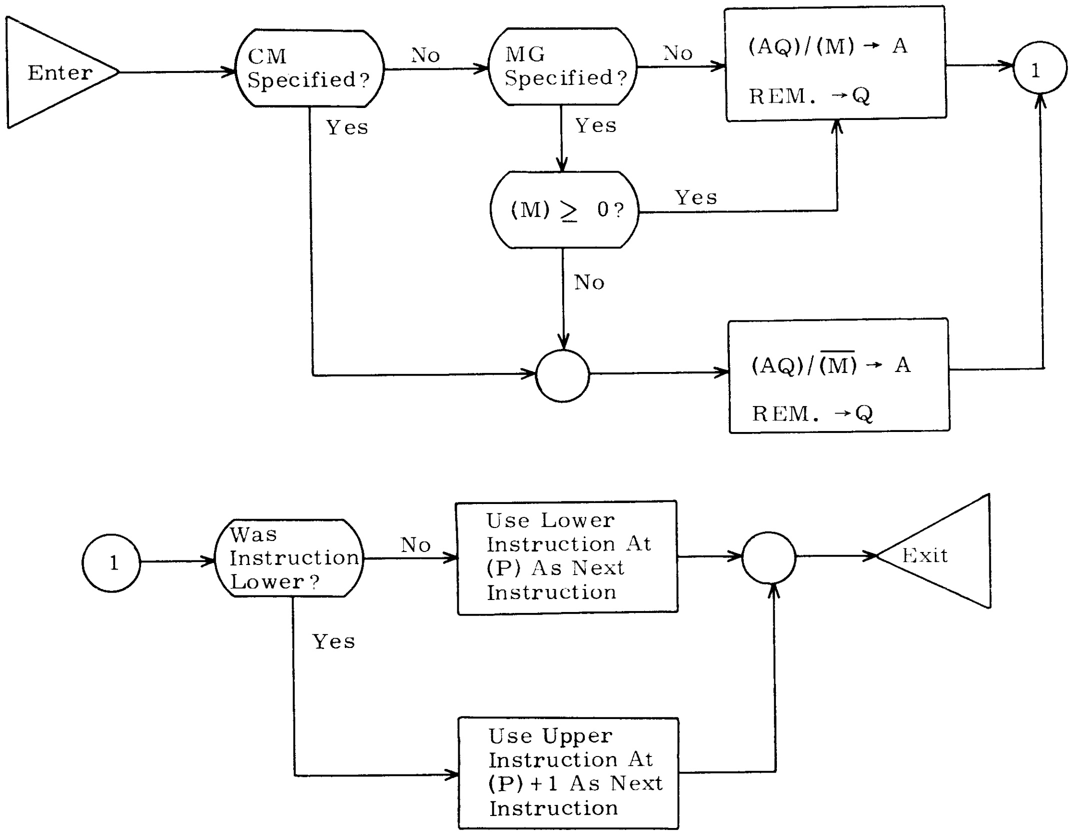
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

DVF

DIVIDE FRACTIONAL   FORMAT: DVF,CM,MG   (a)m,b,v   M = m + (B^b) + (V^v)

FLOWCHART



Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(AQ)/(M) → A
REM. → Q
1
(M) ≥ 0?
Yes
No
(AQ)/$\overline{(M)}$ → A
REM. → Q
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

DVF	1) (AQ) / (M) → A, Remainder → Q
	2) Contents of M remain unchanged
DVF, CM	1) (AQ) / $\overline{(M)}$ → A, Remainder → Q
	2) Contents of M remain unchanged
DVF, MG	1) (AQ) / $|(M)|$ → A, Remainder → Q
	2) Contents of M remain unchanged

E
X
A
M
P
L
E
S

PROBLEM:

Divide AQ by the contents of address DIV.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		DVF	DIV	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Divide AQ by the contents of JAKE modified by Index Register 5.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		DVF	JAKE,5	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Divide AQ by the contents of the address specified in Index Register 4.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		DVF	,4	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Divide AQ by the contents of address PORKY modified by Index Registers 2 and 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		DVF	PØRKY,2,3	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

NEW CONCEPTS OF GROUP 5

The instructions in this group operate on two types of operands - integer and fractional. The ADD, SUB, MUI, and DVI instructions operate on integers only. The MUF and DVF instructions operate on fractions only. Each set must be kept separate and distinct. Let's first discuss integers.

INTEGERS

Integers are the counting numbers 1, 2, 3 ... in the positive direction and -1, -2, -3... in the negative direction with 0 as the origin. In algebra we represent these numbers on a number scale as such:

```
 5
 4
 3
 2
 1
 0
-1
-2
-3
-4
-5
```

The range in both directions is unlimited. From algebra we also know that +2 is greater than -5.

On the computer these numbers are represented in binary. But binary is very difficult to read and very cumbersom to work with. Since binary expands easily to octal, we will use octal representation to illustrate the format and the range of the numbers. The format for integer numbers has the sign of the number in the most significant bit position and the rest of the positions giving the actual number. For positive numbers the correspondence between the algebraic notation and computer notation is as follows:

Algebraic Notation	Computer Notation
.	.
.	.
.	.
4	0 ⟶ 04
3	0 ⟶ 03
2	0 ⟶ 02
1	0 ⟶ 01
0	0 ⟶ 00

The range for positive algebraic numbers is to infinity. The range for the positive numbers on the computer is as much as the 48-bit word can handle. In octal the maximum number is 37 ⟶ 7. If we add 1 to this number, the number turns negative (40 ⟶ 0) which is an incorrect answer. When an operation occurs in which the result exceeds this maximum count, an "overflow" condition results which is interruptible. This is termed Arithmetic Overflow Fault.

The computer also allows negative numbers. Here is a comparison between negative algebraic numbers and the computer representation.

Algebraic Notation	Computer Notation
0	0 ⟶ 0
-1	7 ⟶ 76
-2	7 ⟶ 75
-3	7 ⟶ 74
-4	7 ⟶ 73

The smallest number represented on the computer is $40 \longrightarrow 0$. The number $7 \longrightarrow 7$ on the computer is termed "negative zero". This number will not result from an add o subtract operation, except for a special case, since 1's complement arithmetic is used. What we mean by this is that if 1 is added to $7 \longrightarrow 76$ the result will be $0 \longrightarrow 0$ just as should be in algebra.

An important concept about relating the algebraic and computer numbers system is the talk about "complementing . In algebra the complement of a number is the number wit its sign reversed. A + 5 complemented is -5. A -30 complemented is +30. In the computer it means, "for every 1 bit change to 0, for every 0 bit change to a 1", The complement of $0 \longrightarrow 04$ is $7 \longrightarrow 73$. The complement of $7 \longrightarrow 763$ is $0 \longrightarrow 014$. Try it yourself on the following table and make sure you understand the relationship between algebraic numbers and their computer representation.

Algebraic Notation	Computer Notation
	$37 \longrightarrow 77$
.	.
.	.
3	$0 \longrightarrow 03$
2	$0 \longrightarrow 02$
1	$0 \longrightarrow 01$
0	$0 \longrightarrow 00$
-1	$7 \longrightarrow 76$
-2	$7 \longrightarrow 75$
-3	$7 \longrightarrow 74$
.	.
.	.
	$40 \longrightarrow 00$

The instructions ADD, SUB, MUI, DVI operate on these integers and will yield the corresponding algebraic results.

Now let's look into fractions and their computer representation.

FRACTIONS

If you understood the integer representation of numbers, then fractional representation should not be difficult. Fractions are numbers from 0 up to but not including 1, both positive and negative directions. In algebra we can represent these numbers on a number scale as such:

```
 .3
 .2
 .1
  0
-.1
-.2
-.3
```

The machine format again has the sign of the fraction in the most significant bit position and the rest of the positions giving the actual fraction. The point is considered to be between bit positions 47 and 46. For positive fractions the correspondence between the algebraic notation and computer notation is as follows:

Algebraic Notation	Computer Notation
.	.
.	.
.3 octal	$140 \longrightarrow 0$
.2 octal	$100 \longrightarrow 0$
.1 octal	$040 \longrightarrow 0$
0	$000 \longrightarrow 0$

The computer notation may seem strange at first. But the representation is correct. Consider .1 octal:

1. .1 octal = .001 binary
2. .001 binary = 0.001 binary
3. 0.001 binary = $040 \longrightarrow 0$ octal (regroup)

The computer also allows negative fractions. Here is a table representing a comparison between the two systems.

Algebraic Notation	Computer Notation
.3 octal	140 $\longrightarrow$ 0
.2 octal	100 $\longrightarrow$ 0
.1 octal	040 $\longrightarrow$ 0
0	000 $\longrightarrow$ 0
-.1 octal	737 $\longrightarrow$ 7
-.2 octal	677 $\longrightarrow$ 7
-.3 octal	637 $\longrightarrow$ 7

The negative computer fractions may seem strange and you may wonder how they were arrived at. As a helpful aid in understanding the negative conversion, keep in mind the principle used to convert negative integers, and then note how the above negative fractions are complements of their positive counterparts.

FAULTS

What computer faults can result from the execution of any of the six instructions within this group?

	Instruction	Possible Fault
1.	ADD	Arithmetic Overflow Fault - adding two positive operands where the result exceeds the maximum positive limit, <u>or</u> adding two negative operands where the result exceeds the maximum negative limit. Adding a positive to a negative operand will never yield overflow since the answer will move closer to zero.
2.	SUB	Arithmetic Overflow Fault - subtracting a negative operand from a positive operand

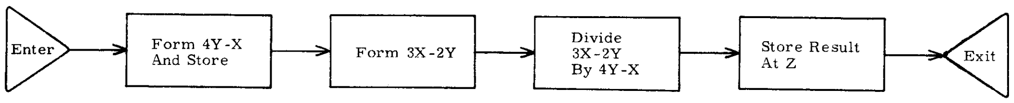
			and the result exceeds the maximum positive limit, or subtracting a positive operand from a negative operand and the result exceeds the maximum negative limit. Subtracting operands with like signs will never yield overflow.
3.	MUI	No possible fault since the product of any two operands will fit into a 96 bit register (QA).	
4.	DVI	Divide Fault -	dividing by zero or a division where the dividend so exceeds the divisor that the answer is not capable of fitting into a 48 bit register(A).
5.	MUF	No possible fault since the product of any two operands will fit into a 96 bit register (AQ).	
6.	DVF	Divide Fault -	dividing by zero or a division where the dividend is greater than or equal to the divisor. Here the result would be greater than or equal to 1 which is illegal in fractional format.

All faults mentioned are computer interruptible.

Problem 5:

Evaluate Z as close as possible if $Z = \frac{3X-2Y}{4Y-X}$ where X and Y are integers whose absolute values are less than 500.

Flowchart:



Problem 5 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	EVAL	
	ENTRY	EVAL	
DIVIS	BSS	1	
CØN1	DEC	4	
CØN2	DEC	3	
EVAL	BSS	1	
	LDA	Y	
	MUI	CØN1	
	SUB	X	4Y-X
	STA	DIVIS	
	LDA	X	
	MUI	CØN2	
	SUB	Y	
	SUB	Y	3X-2Y
	AJP,PL	EXTZER	EXTEND
	ENQ	-0	SIGN
	SLJ	DIV	THRØUGH
EXTZER	ENQ	0	Q
DIV	DVI	DIVIS	
	STA	Z	
	SLJ	EVAL	
	END		

Somewhere within this subprogram would also be included the symbols X and Y in the location field with a declaration of the prestored data or area reserved.

Student Problem 5A:

Evaluate $W = \frac{3R-S+5T}{S-T} + 1$ if R, S, and T are integers whose absolute value is less than 100.

Flowchart:

Problem 5A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 6

REPLACE OPERATIONS

GROUP 6

REPLACE OPERATIONS

1.	Replace Add	RAD
2.	Replace Subtract	RSB
3.	Replace Add One	RAO
4.	Replace Subtract One	RSO

This is a group of instructions that adds or subtracts a quantity from a storage address. The same result left at the storage address also remains in A. Two storage cycles are required for each instruction.

The first two instructions add or subtract the contents of A from the contents of the storage address.

The last two instructions increment or decrement the contents of the storage address by one. This result remains in A. The initial contents of A are not used and are destroyed by these instructions.

The REPLACE ADD Instruction

The REPLACE ADD instruction is an instruction that adds a 48-bit operand from an 18-bit storage address to the contents of A and returns the result to both the A register and the storage address in memory. Two memory references are made.

The 18-bit storage address is composed of a bank term a (within parentheses) and a modified base address M where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is added to the contents of A and the result returned to both the A register and the storage address in memory.

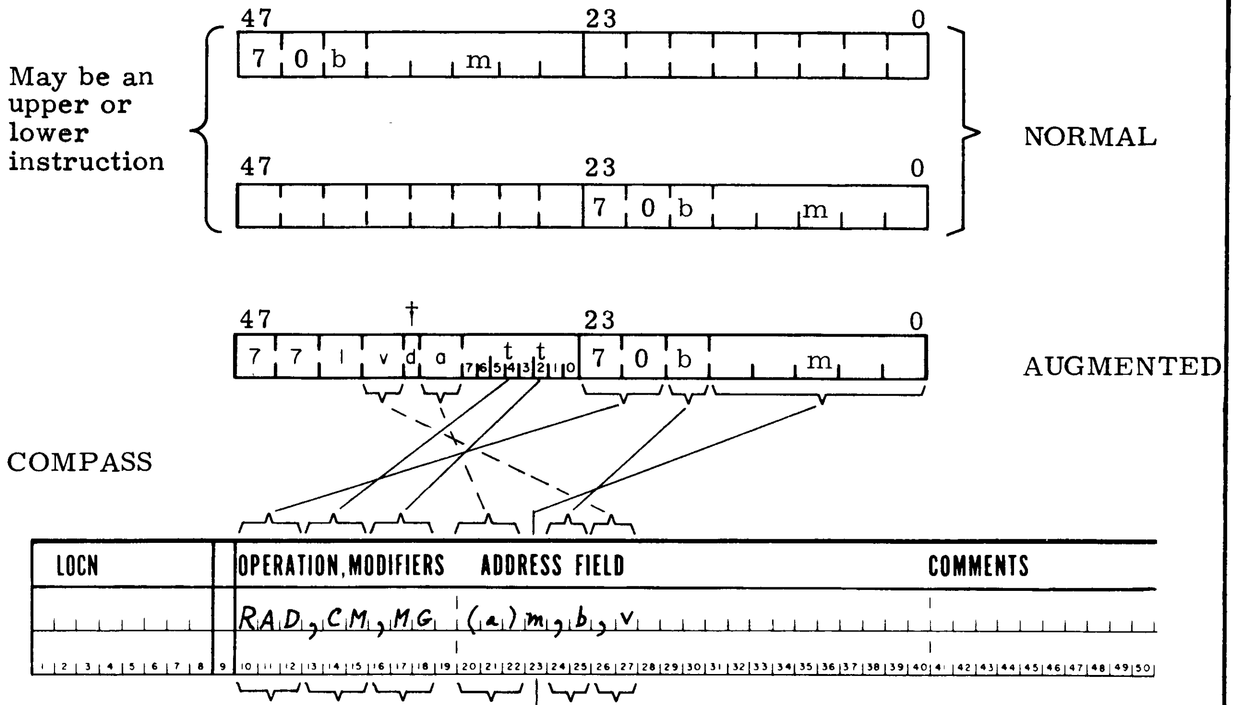
If MG is specified by the programmer, the magnitude (absolute value) of the operand is added to the contents of A and the result returned to both the A register and the storage address in memory.

REPLACE ADD

RAD

FORMAT

MACHINE



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

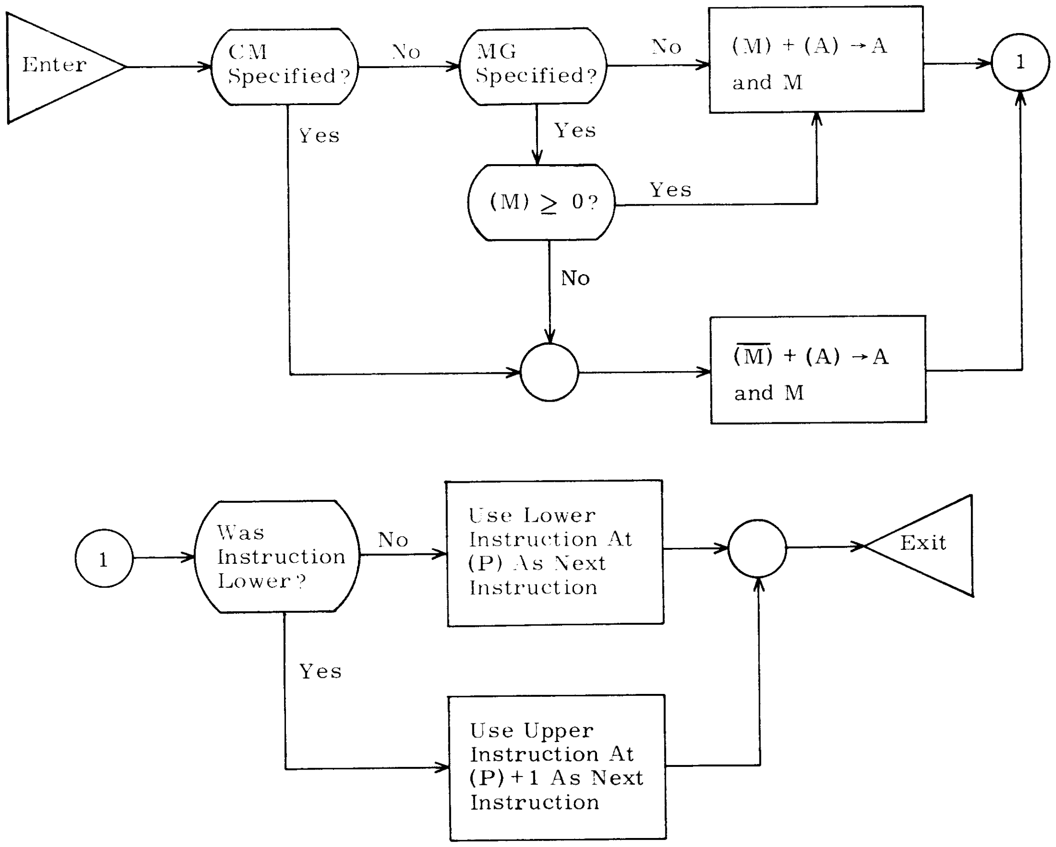
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

RAD

REPLACE ADD   FORMAT: RAD,CM,MG   (a)m,b,v $M = m + (B^b) + (V^v)$

FLOWCHART


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(M) + (A) → A and M
1
(M) ≥ 0?
Yes
No
$\overline{(M)}$ + (A) → A and M
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

RAD	(A) + (M) → A and M
RAD, CM	(A) + $\overline{(M)}$ → A and M
RAD, MG	(A) + $|(M)|$ → A and M

PROBLEM:

Add A to the contents of address POPS.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		RAD	PØPS	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Do a replace add to the contents of address BLO modified by Index Registers 1 and 2.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		RAD	BLØ,1,2	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Do a replace add to A with the absolute value of the contents of address PLUG.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		RAD,MG	PLUG	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Do a replace add to A using the contents of the address specified in Index Register 5.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		RAD	,5	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The REPLACE SUBTRACT Instruction

The REPLACE SUBTRACT instruction is an instruction that subtracts the contents of A from a 48-bit operand (from an 18-bit storage address) and returns the result to both the A register and the storage address in memory. Two memory references are made.

The 18-bit storage address is composed of a bank term a (within parentheses) and a modified base address M where M = $m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the contents of A are subtracted from the complement of the operand from memory and the result returned to both the A register and storage address in memory.

If MG is specified by the programmer, the contents of A are subtracted from the magnitude (absolute value) of the operand from memory and the result returned to the A register and storage address in memory.

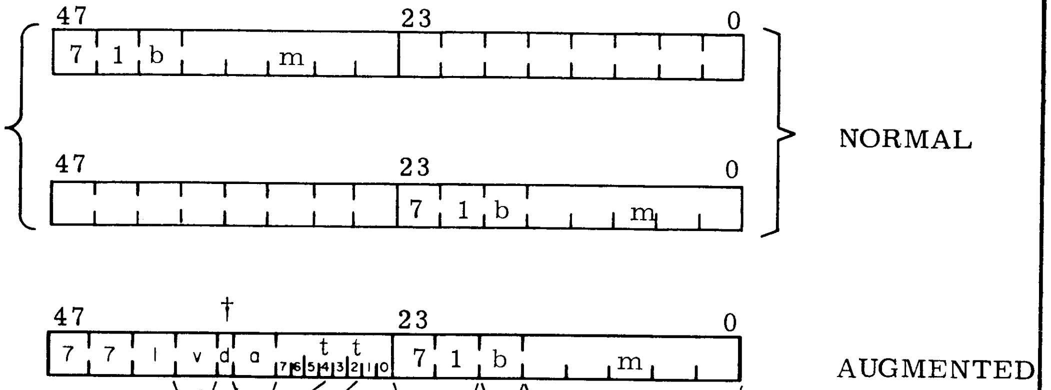
REPLACE SUBTRACT

RSB

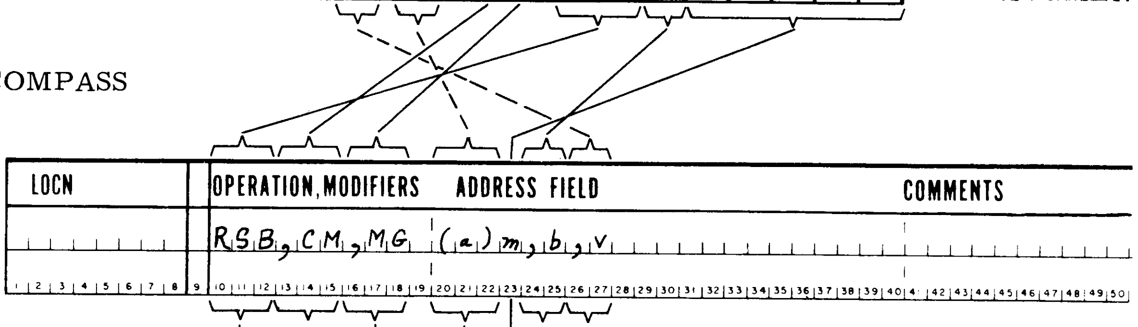
FORMAT

MACHINE

May be an upper or lower instruction



COMPASS



Function code

Optional, transmit complement of operand.

Optional, transmit magnitude of operand.

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

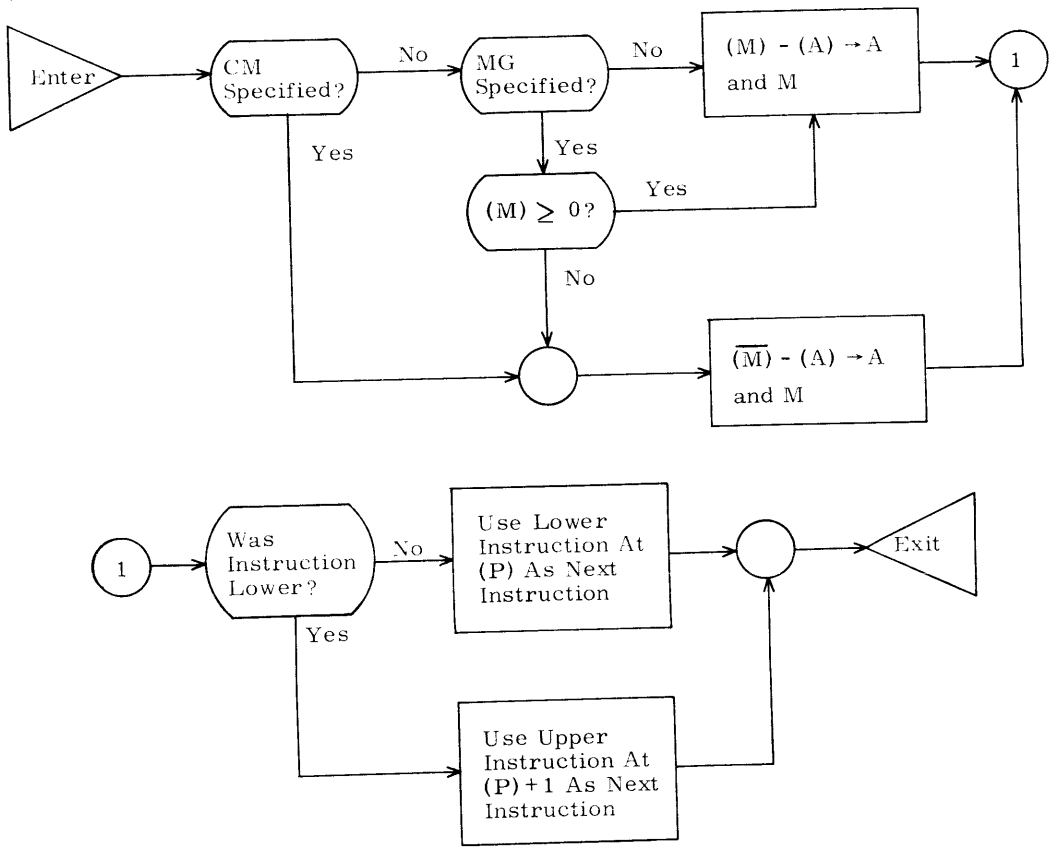
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

RSB

FLOWCHART

REPLACE SUBTRACT FORMAT: RSB,CM,MG (a)m,b,v $M = m + (B^b) + (V^v)$


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(M) - (A) → A and M
1
(M) ≥ 0?
Yes
No
$\overline{(M)}$ - (A) → A and M
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

RSB	(M) - (A) → A and M
RSB, CM	$\overline{(M)}$ - (A) → A and M
RSB, MG	$\lvert(M)\rvert$ - (A) → A and M

PROBLEM:

Subtract A from the contents of address MACK with the result transmitted to address MACK.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		RSB	MACK	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Do a replace subtract on the contents of address TABLE modified by Index Register 1.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		RSB	TABLE,1	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Do a replace subtract on the contents of address TEST modified by Index Registers 1 and 2.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		RSB	TEST,1,2	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Do a replace subtract on the contents of the address specified in Index Register 6.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		RSB	,6	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The REPLACE ADD ONE Instruction

The REPLACE ADD ONE instruction is an instruction that increments a 48-bit operand from an 18-bit storage address by 1 and returns the result to both the A register and the storage address in memory. The initial contents of A are not used and are destroyed during the operation. Two memory references are made.

The 18-bit storage address is composed of a bank term a̲ (within parentheses) and a modified base address M̲ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

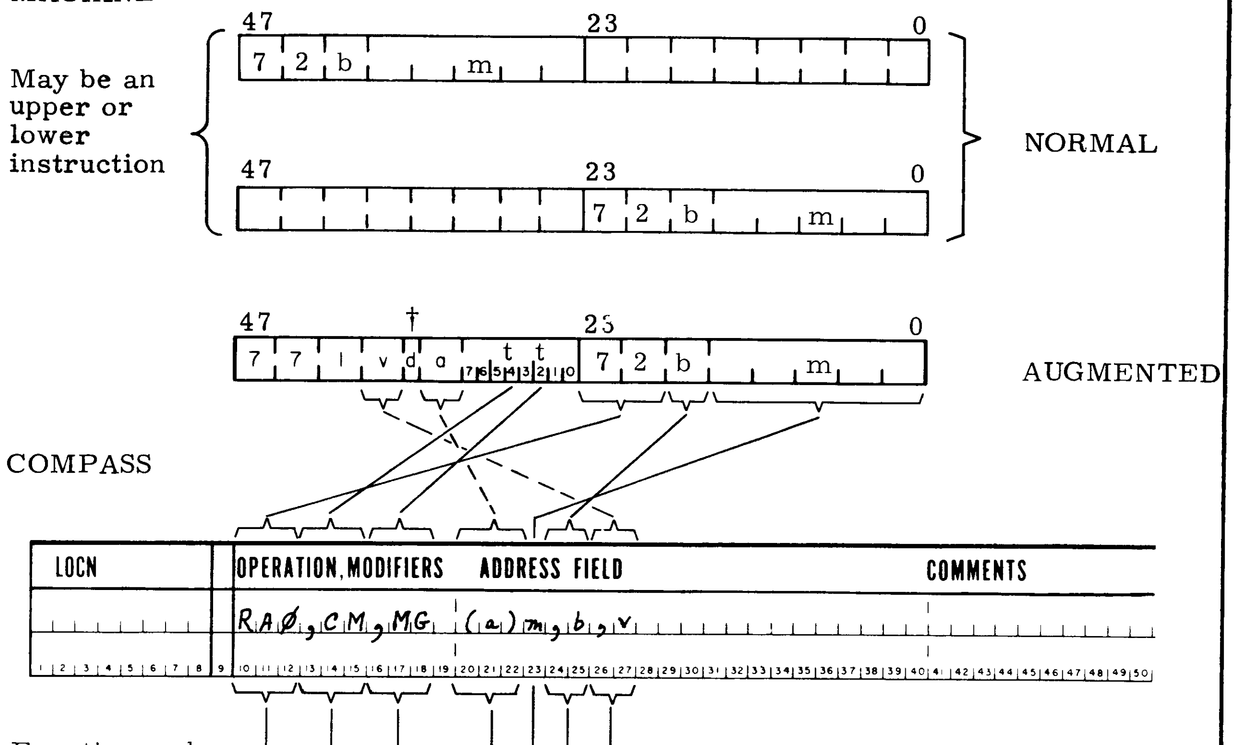
If CM is specified by the programmer, the complement of the operand from memory is incremented by 1 and the result returned to both the A register and the storage address in memory.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is incremented by 1 and the result returned to both the A register and the storage address in memory.

FORMAT

MACHINE

May be an upper or lower instruction



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

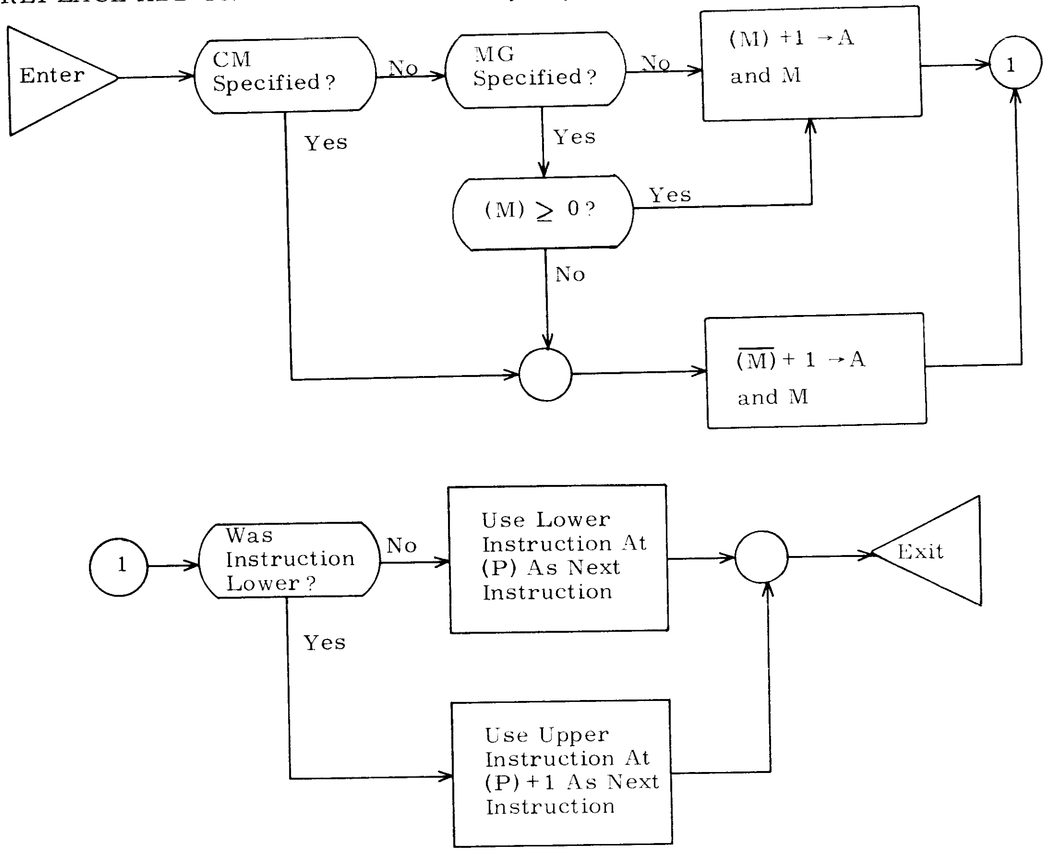
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

RAO

F L O W C H A R T

REPLACE ADD ONE   FORMAT: RAO,CM,MG   (a)m,b,v $M = m + (B^b) + (V^v)$



Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(M) +1 → A and M
1
(M) ≥ 0?
Yes
No
$\overline{(M)}$ + 1 → A and M
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

D E S C R I P T I O N

RAO	(M) + 1 → A and M
RAO, CM	$\overline{(M)}$ + 1 → A and M
RAO, MG	$\lvert(M)\rvert$ + 1 → A and M

EXAMPLES

PROBLEM:

Increase the contents of address SAM by one.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RAØ	SAM	

PROBLEM:

Add one to the contents of REPEAT indirectly.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RAØ	REPEAT,7	

PROBLEM:

Add one to the contents of the address specified in Index Register 6.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RAØ	,6	

PROBLEM:

Add one to the contents of address PUNCH in the bank that the RAO is in.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RAØ	(*)PUNCH	

The REPLACE SUBTRACT ONE Instruction

The REPLACE SUBTRACT ONE instruction is an instruction that decrements a 48-bit operand from an 18-bit storage address by 1 and returns the result to both the A register and the storage address in memory. The initial contents of A are not used and are destroyed during the operation. Two memory references are made.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

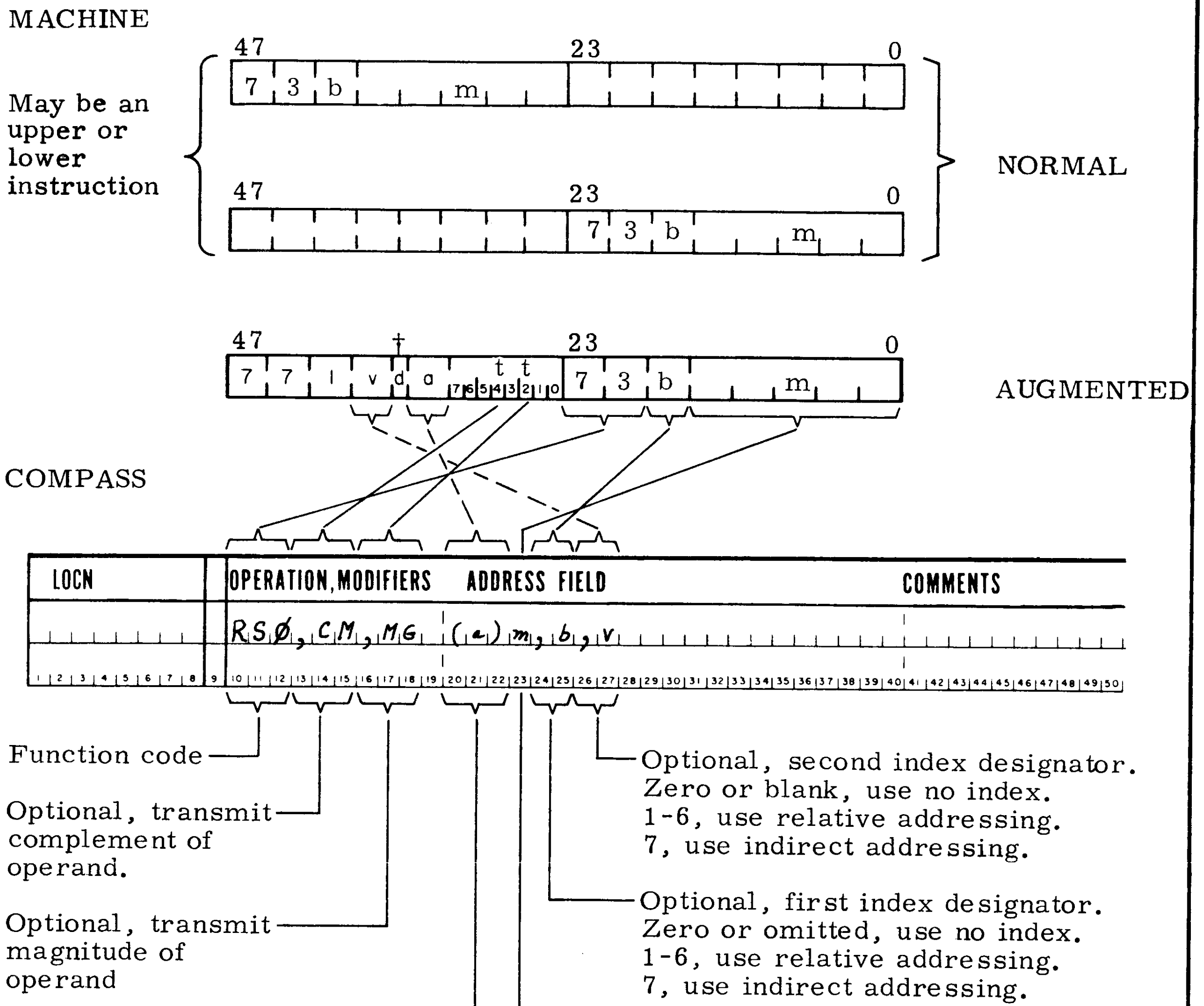
If CM is specified by the programmer, the complement of the operand from memory is decremented by 1 and the result returned to both the A register and the storage address in memory.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is decremented by 1 and the result returned to both the A register and the storage address in memory.

REPLACE SUBTRACT ONE

RSO

FORMAT



Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

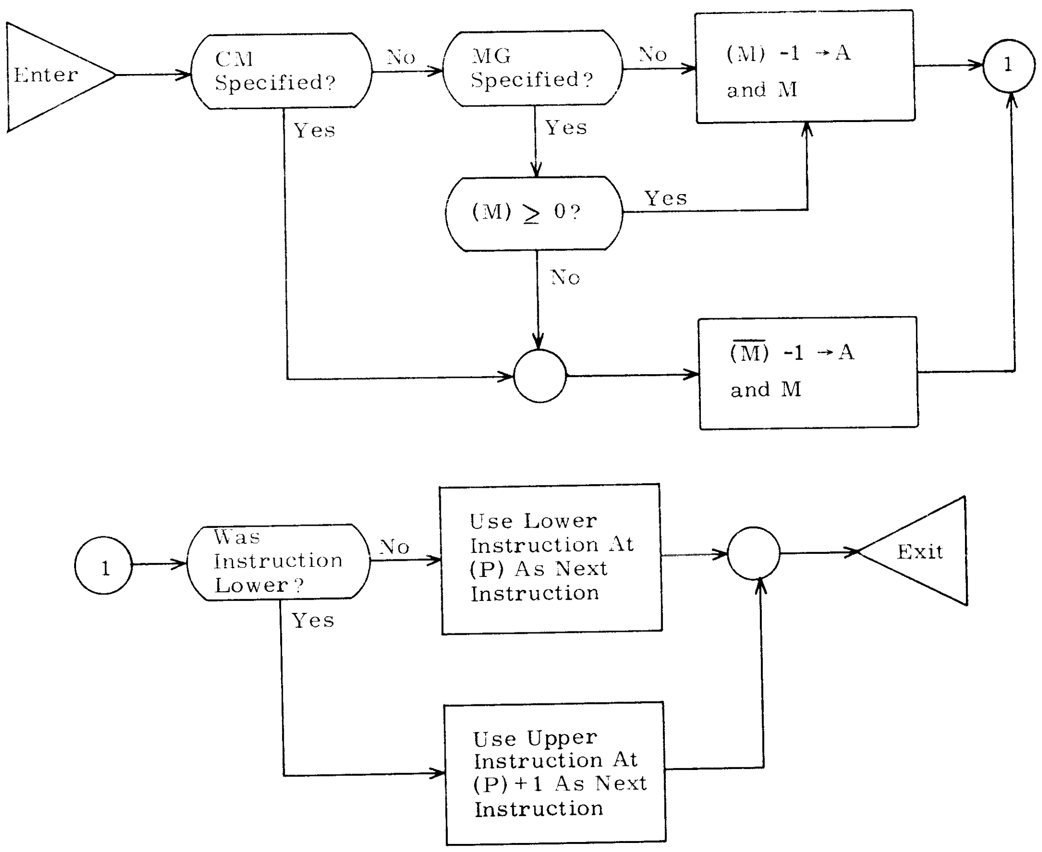
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

RSO

REPLACE SUBTRACT ONE FORMAT: RSO,CM,MG (a)m,b,v $M = m + (B^b) + (V^v)$

FLOWCHART


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(M) -1 → A and M
1
(M) ≥ 0?
Yes
No
$\overline{(M)}$ -1 → A and M
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

RSO	$(M) - 1 \rightarrow A$ and M
RSO, CM	$\overline{(M)} - 1 \rightarrow A$ and M
RSO, MG	$\lvert(M)\rvert - 1 \rightarrow A$ and M

PROBLEM:

Decrease the contents of address NUMBER by one.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RSØ	NUMBER	

PROBLEM:

Subtract one from the contents of SLIP modified by Index Register 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RSØ	SLIP,2	

PROBLEM:

Subtract one from the contents of the address specified in Index Register 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RSØ	,3	

PROBLEM:

Subtract one from the contents of the address three memory locations forward of this instruction.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RSØ	*+3	

NEW CONCEPTS OF GROUP 6

The instructions in this group provide the programmer with an easy way to add or subtract data from a memory address.

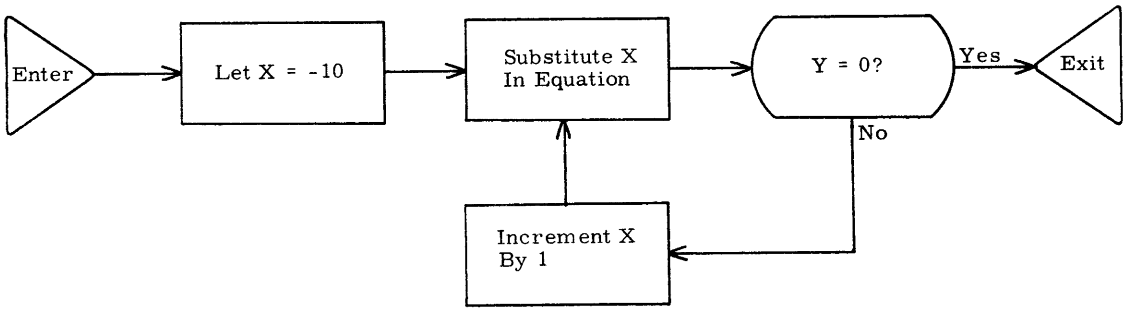
Each instruction performs the arithmetic in fixed point integer format. Arithmetic Overflow can occur if the limits $\pm(2^{47}-1)$ are exceeded.

Problem 6:

Given: $Y = X^5 - 4X^3 + 135$

There exists one integer for X between 0 and -10 such that, when it is substituted, will cause Y to be equal to zero. Find that integer and store it at address X.

Flowchart:



Problem 6 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	SØLVE	
	ENTRY	SØLVE	
CØN1	DEC	-4	
CØN2	DEC	135	
X	DEC	-10	
SAVE1	BSS	1	X CUBED
SAVE2	BSS	1	
SØLVE	BSS	1	
	LDA	X	
TRYAGN	MUI	X	X SQUARED
	MUI	X	X CUBED
	STA	SAVE1	
	MUI	X	X 4TH
	MUI	X	X 5TH
	STA	SAVE2	
	LDA	SAVE1	
	MUI	CØN1	-4X CUBED
	ADD	SAVE2	
	ADD	CØN2	X**5-4X**3+135
	AJP,ZR	SØLVE	EXIT
	RAØ	X	
	SLJ	TRYAGN	
	END		

Student Problem 6A:

Given: $Y = 3X^3 - 4X^2 - 34385$

There exists one integer for X between 0 and 30 such that, when it is substituted, will cause Y to be equal to zero. Find that integer and store it at address X.

Flowchart:

Problem 6A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 7

JUMPS AND STOPS

GROUP 7

JUMPS AND STOPS

1.	Unconditional Stop/Jumps	SLX
2.	Selective Jumps	SJX
3.	A Jump	AJP
4.	Q Jump	QJP
5.	Selective Stops (with Normal Jumps)	SSX

This group of instructions allows the programmer to stop the computer (if not illegal), and to transfer program control to another area of the program.

It is important to note that the jumps are not bank jump instructions in any way. The purpose for the bank term is to set or reset the Operand Bank setting only. The jump is still within the same bank in which current program control is operating.

Some of the jumps and stops are conditional. They are conditioned by the contents of A, the contents of Q, the Jump switches, and the Stop switches.

The UNCONDITIONAL STOP/JUMP Instruction

The SELECTIVE JUMP instruction is an instruction that transfers program control unconditionally to another point within the program.

The jump address to which control is transferred is $\underline{M}$ where $M = m + (V^V)$. The jump is made within the same bank that the program is operating, i.e., this is not a bank jump instruction.

The bank term $\underline{a}$ determines the value of the operand bank setting. If it is not used, the current operand bank setting remains. If it is used, the current operand bank setting will be replaced by the value $\underline{a}$. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

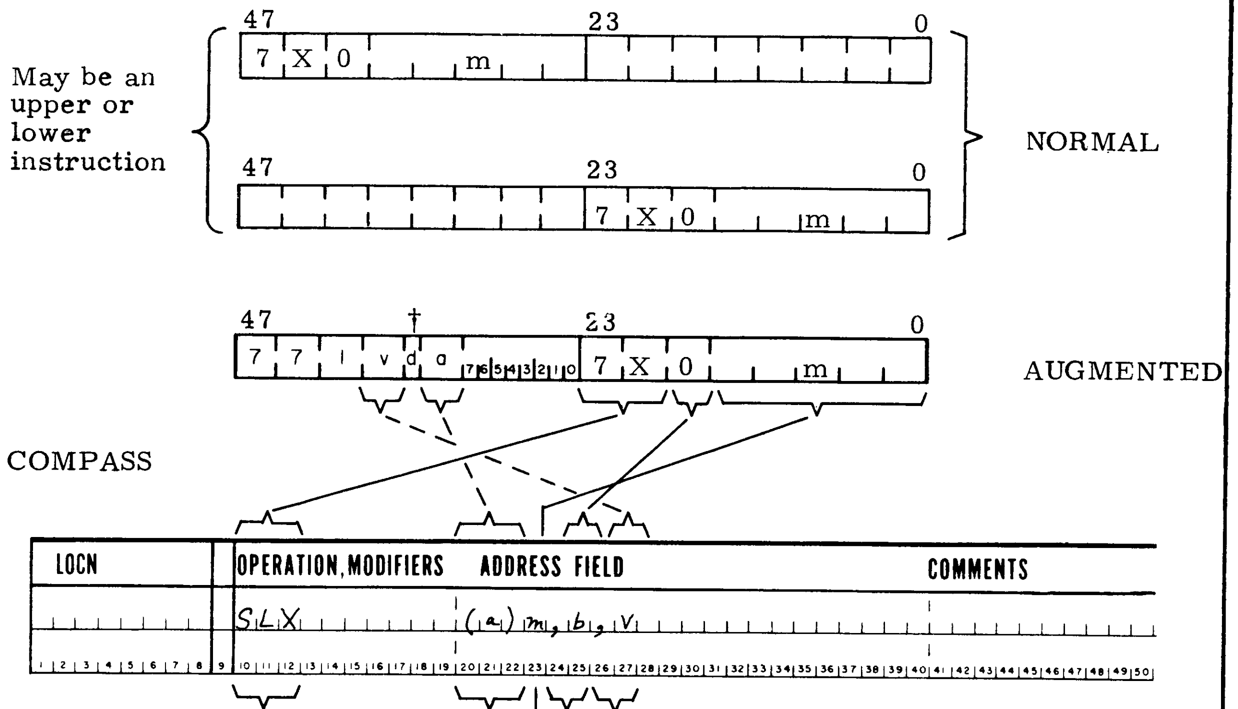
The SELECTIVE STOP instruction is an instruction that unconditionally stops the computer. When the computer is restarted, program control unconditionally transfers to another point within the program.

The SELECTIVE STOP instruction then follows the same form as the SELECTIVE JUMP (paragraphs 2 and 3 above).

FORMAT

MACHINE

May be an upper or lower instruction



Function code where:
SLJ = 750
SLS = 760

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Must be zero or omitted.

Optional, base address.
Zero if omitted.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

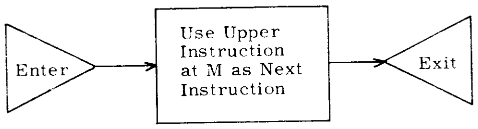
† d bit not programmable

SLX

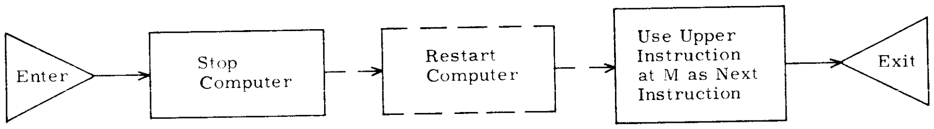
FLOWCHART

UNCONDITIONAL STOP/JUMP FORMAT: SLX (a)m, v $M = m + (V^v)$

SLJ



SLS



DESCRIPTION

SLJ	Jump to M
SLS	Stop, then jump to M upon restarting

EXAMPLES

PROBLEM:

Perform an Unconditional Jump to address LOOP.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SLJ	LØØP	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Perform an Unconditional Jump to BEGIN. The routine at BEGIN is to pick up operands from Bank 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SLJ	(3) BEGIN	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Stop the computer. When restarted, a jump to address START is to be made.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SLS	START	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Stop the computer. When restarted, a jump 3 memory locations forward is to be made.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SLS	*+3	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The SELECTIVE JUMP Instructions

The SELECTIVE JUMP instructions are conditional jump instructions that test the three jump switches on the console to see if they are ON or OFF. There is one jump instruction for each switch.

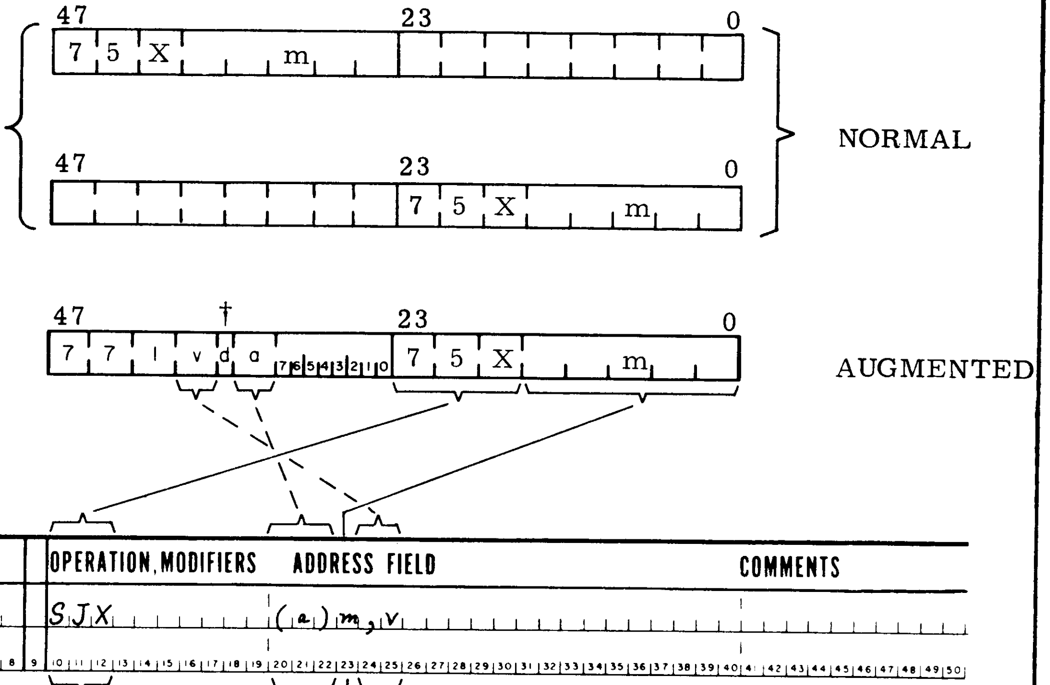
If the jump switch is ON, program control transfers to address $\underline{M}$ where $M = m + (V^V)$. This address is a 15-bit address within the same bank that the program is operating, i.e., this is not a bank jump instruction. If the jump switch is OFF, program control continues executing in its normal sequence with no jump being made.

The bank term $\underline{a}$ determines the value of the operand bank setting. If it is not used, the current operand bank setting remains. If it is used, the current operand bank setting will be replaced by the value $\underline{a}$. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

FORMAT

MACHINE

May be an upper or lower instruction



COMPASS

Function code where:
SJ1 = 751
SJ2 = 752
SJ3 = 753

Optional, index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

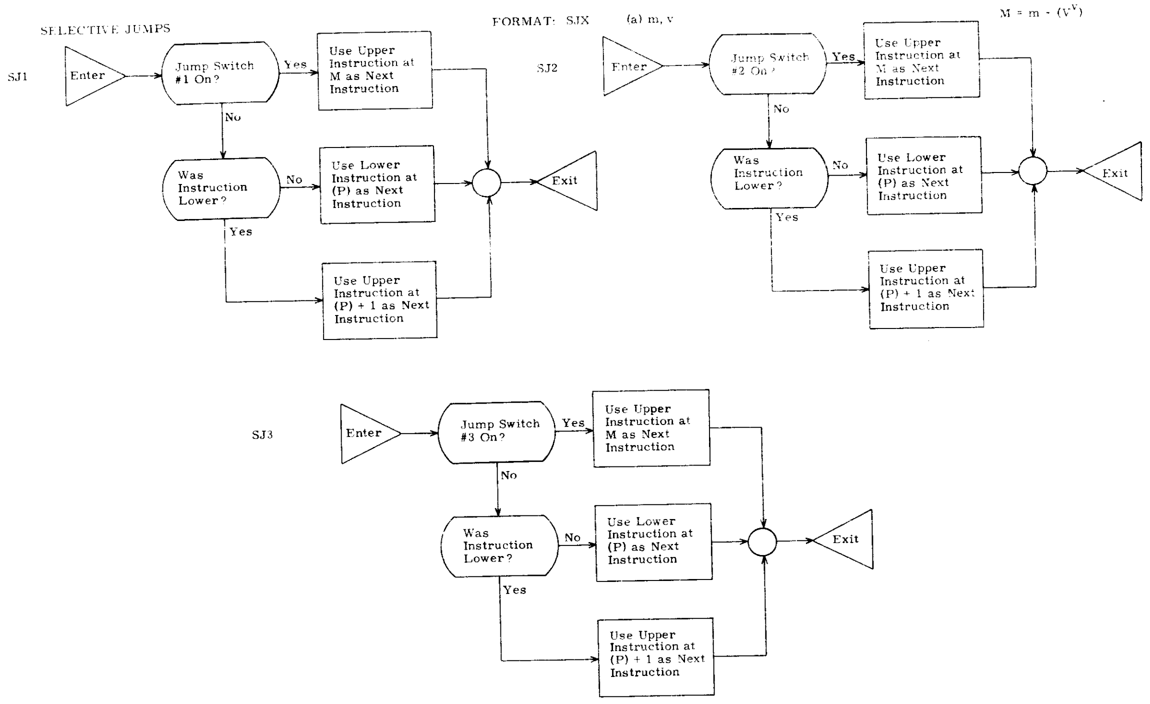
$, transmits bank number in which base address resides to Operand Bank Register.

*, Transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

SJX

FLOWCHART


SELECTIVE JUMPS
FORMAT: SJX (a) m, v
M = m + (V^v)
SJ1
Enter
Jump Switch #1 On?
Yes
Use Upper Instruction at M as Next Instruction
No
Was Instruction Lower?
No
Use Lower Instruction at (P) as Next Instruction
Exit
Yes
Use Upper Instruction at (P) + 1 as Next Instruction
SJ2
Enter
Jump Switch #2 On?
Yes
Use Upper Instruction at M as Next Instruction
No
Was Instruction Lower?
No
Use Lower Instruction at (P) as Next Instruction
Exit
Yes
Use Upper Instruction at (P) + 1 as Next Instruction
SJ3
Enter
Jump Switch #3 On?
Yes
Use Upper Instruction at M as Next Instruction
No
Was Instruction Lower?
No
Use Lower Instruction at (P) as Next Instruction
Exit
Yes
Use Upper Instruction at (P) + 1 as Next Instruction

DESCRIPTION

SJ1	Jump to M if Jump Switch 1 is on Continue program is Jump Switch 1 is not on
SJ2	Jump to M if Jump Switch 2 is on Continue program if Jump Switch 2 is not on
SJ3	Jump to M if Jump Switch 3 is on Continue program if Jump Switch 3 is not on

EXAMPLES

PROBLEM:

Check Jump Switch 1. If ON, jump to address CARDTOTA. If OFF, continue program.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SJ1	CARDTØTA	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Check Jump Switch 2. If ON, jump to address OK modified by Index Register 3. If OFF, continue program.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SJ2	ØK,3	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Check Jump Switch 3. If ON, jump to the address specified in Index Register 4. If OFF, continue program.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SJ3	,4	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Check Jump Switch 3. If ON, jump to an address two memory locations forward. If OFF, continue program.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SJ3	*+2	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The A JUMP Instruction

The A JUMP instruction is a conditional jump instruction that tests the contents of A for zero, non-zero, positive, or negative. One modifier must be specified.

If the condition tested is true, program control transfers to address M where M = m + (V^v). This address is a 15-bit address within the same bank that the program is operating, i.e., this is not a bank jump instruction. If the condition tested is not true, program control continues executing in its normal sequence with no jump being made.

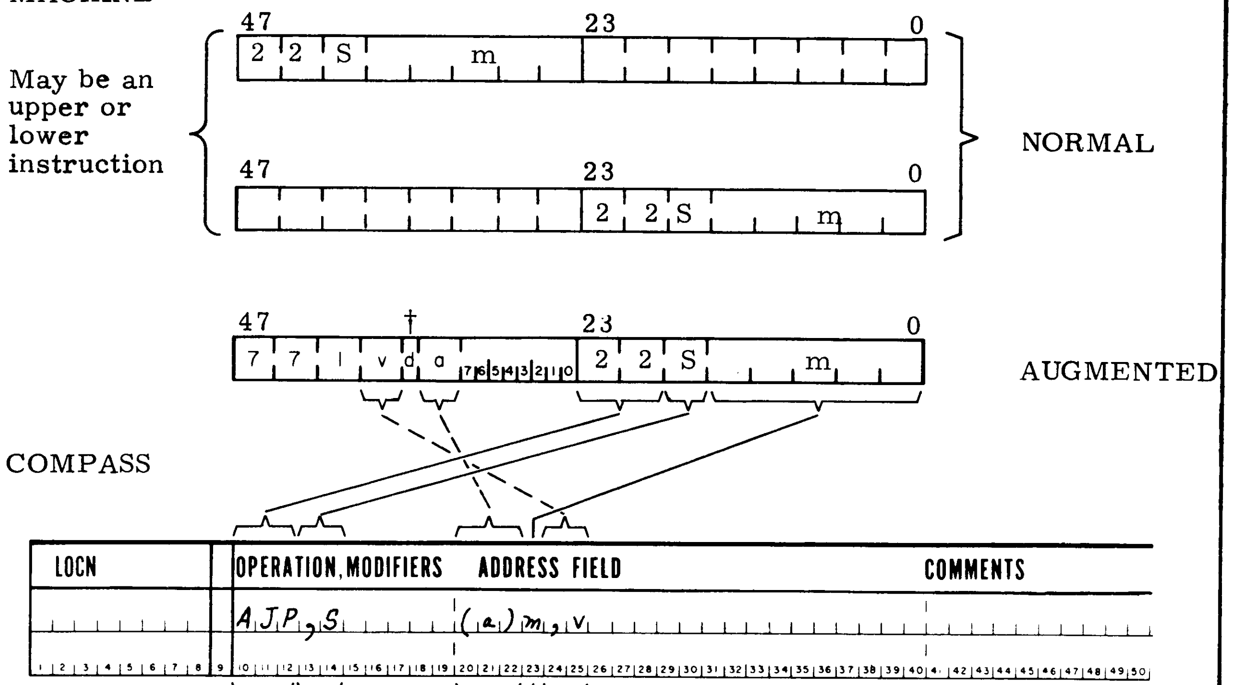
The bank term a determines the value of the operand bank setting. If it is not used, the current operand bank setting remains. If it is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

A JUMP

AJP

FORMAT

MACHINE



Function code

Modifier; One of the following must be specified:
ZR, zero (220)
NZ, non-zero (221)
PL, plus (222)
MI, minus (223)

Optional, index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

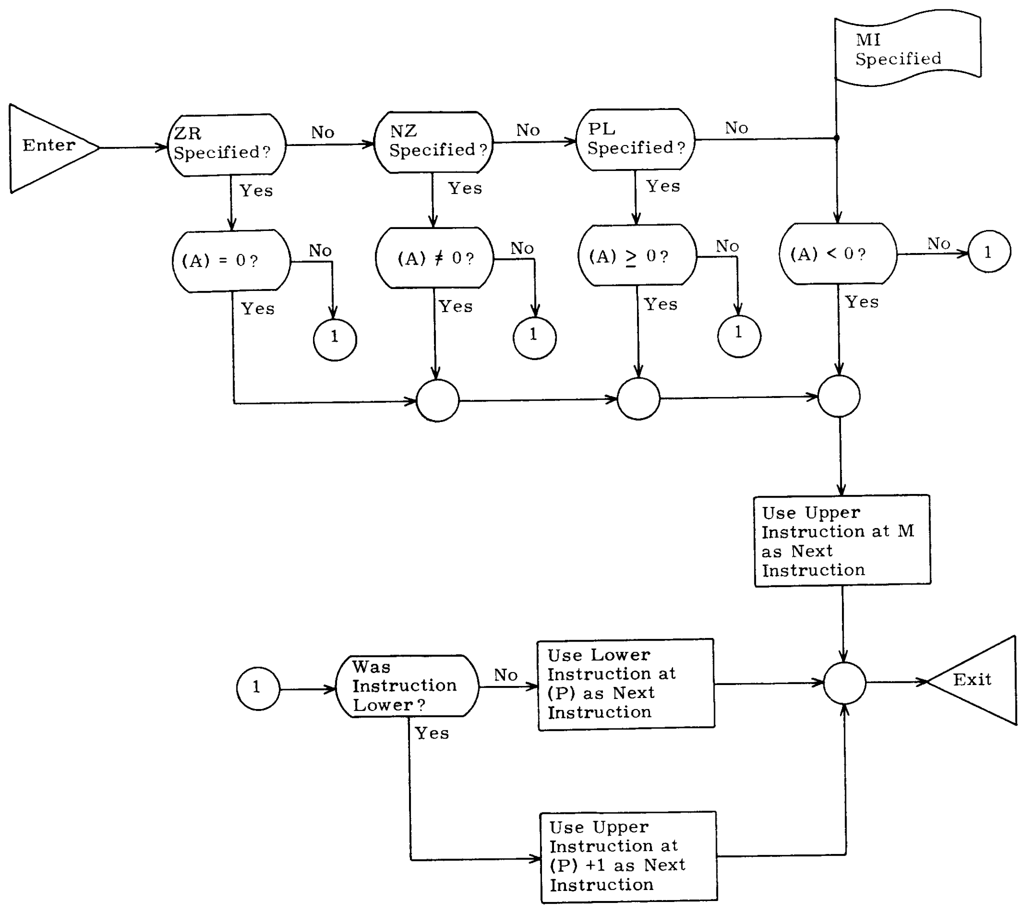
†d bit not programmable

AJP

FLOWCHART

A JUMP

FORMAT: AJP, ZR (a) m, v $M = m + (V^v)$
NZ
PL
MI



DESCRIPTION

AJP, ZR	(A) = 0 jump to M, otherwise continue
AJP, NZ	(A) ≠ 0 jump to M, otherwise continue
AJP, PL	(A) ≥ 0 jump to M, otherwise continue
AJP, MI	(A) < 0 jump to M, otherwise continue

E
X
A
M
P
L
E
S

PROBLEM:

When A is zero, a jump is to be made to address ROOT. When A is non-zero, the program will continue. What instruction will check this?

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		AJP, ZR	RØØT	

PROBLEM:

If A is positive, jump to address EPSILON modified by Index Register 3. If A is negative, continue program.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		AJP, PL	EPSILØN, 3	

PROBLEM:

If A is negative, jump to the address specified in Index Register 4. If A is positive, continue program.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		AJP, MI	, 4	

PROBLEM:

If A is non-zero, jump three locations backward. If A is zero, continue program.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		AJP, NZ	*-3	

The Q JUMP Instruction

The Q JUMP instruction is a conditional jump instruction that tests the contents of Q for zero, non-zero, positive, or negative. One modifier must be specified.

If the condition tested is true, program control transfers to address M where M = m + (V^V). This address is a 15-bit address within the same bank that the program is operating, i.e., this is not a bank jump instruction. If the condition tested is not true, program control continues executing in its normal sequence with no jump being made.

The bank term a determines the value of the operand bank setting. If it is not used, the current operand bank setting remains. If it is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

Q JUMP

QJP

FORMAT

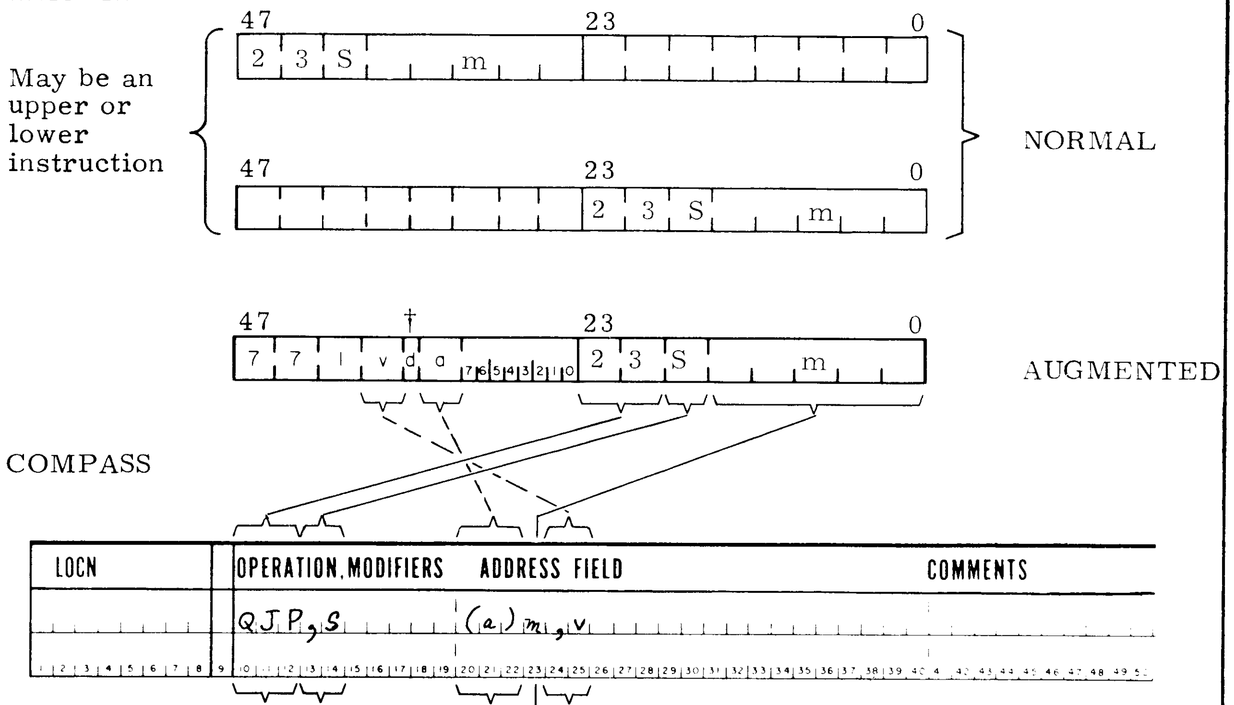
MACHINE

May be an upper or lower instruction

NORMAL

AUGMENTED

COMPASS



Function code

Modifier: One of the following must be specified:

ZR, zero	(230)
NZ, non-zero	(231)
PL, plus	(232)
MI, minus	(233)

Optional, index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

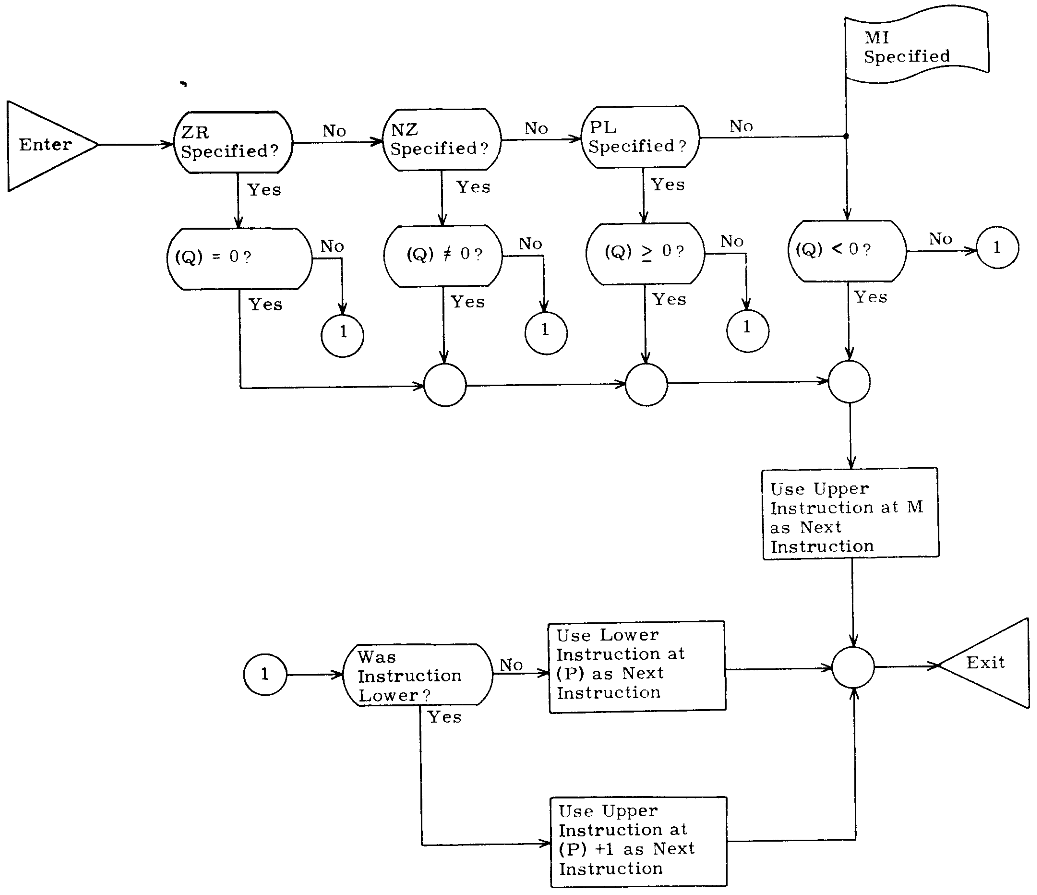
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

FLOWCHART

Q JUMP

FORMAT: QJP, ZR / NZ / PL / MI (a) m. v $M = m + (V^v)$



DESCRIPTION

QJP, ZR	(Q) = 0 jump to M, otherwise continue
QJP, NZ	(Q) ≠ 0 jump to M, otherwise continue
QJP, PL	(Q) ≥ jump to M, otherwise continue
QJP, MI	(Q) < 0 jump to m, otherwise continue

PROBLEM:

When Q is zero, a jump is to be made to address PARAMET. When Q is non-zero, the program will continue. What instruction will check this?

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		QJP,ZR	PARAMET	

PROBLEM:

If Q is positive, jump to address LAMBDA modified by Index Register 3. If Q is negative, continue program.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		QJP,PL	LAMBDA,3	

PROBLEM:

If Q is negative, jump to the address specified in Index Register 5. If Q is positive, continue program.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		QJP,MI	,5	

PROBLEM:

If Q is non-zero, jump two locations forward. If Q is zero, continue program.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		QJP,NZ	*-2	

The SELECTIVE STOP Instructions

The SELECTIVE STOP instructions are conditional stop instructions that test the three stop switches on the console to see if they are ON or OFF. There is one stop instruction for each switch.

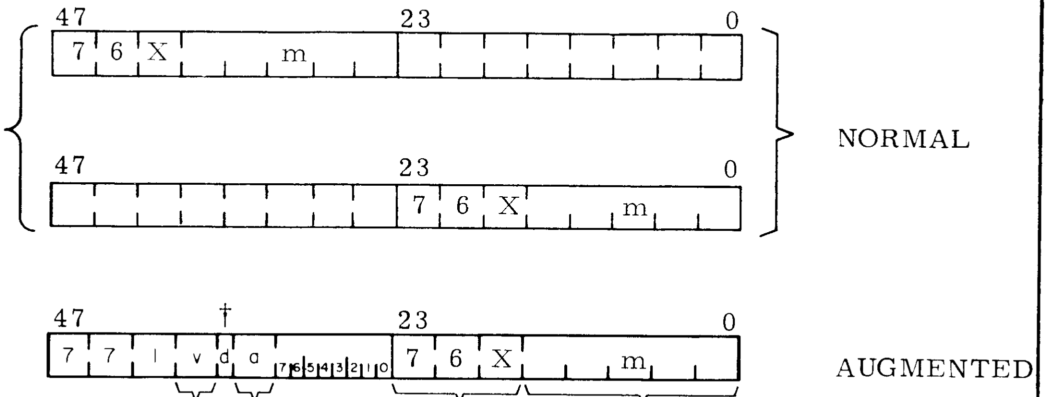
If the stop switch is ON, the computer will stop. When the computer is restarted, program control will transfer to address $\underline{M}$ where $M = m + (V^V)$. This address is a 15-bit address within the same bank that the program is operating, i.e., this is <u>not</u> a bank jump instruction. If the stop switch is OFF, the computer will not stop. However, program control still goes to address M.

The bank term $\underline{a}$ determines the value of the operand bank setting. If it is not used, the current operand bank setting remains. If it is used, the current operand bank setting will be replaced by the value $\underline{a}$. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

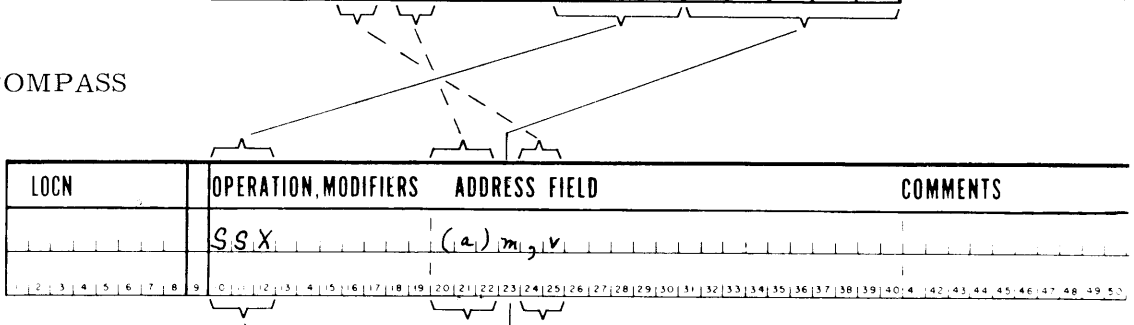
FORMAT

MACHINE

May be an upper or lower instruction



COMPASS



Function code when:
SS1 = 761
SS2 = 762
SS3 = 763

Optional, index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.

Optional, bank term.

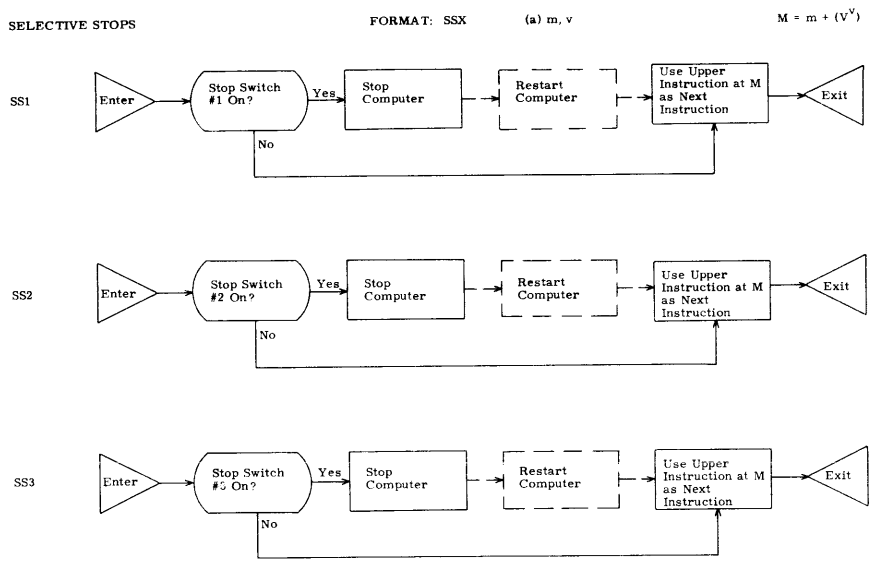
0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

FLOWCHART



DESCRIPTION

SS1	Stop if Stop Switch 1 is on, always jump to M
SS2	Stop if Stop Switch 2 is on, always jump to M
SS3	Stop if Stop Switch 3 is on, always jump to M

EXAMPLES

PROBLEM:

Stop the computer if Stop Switch 1 is ON. When restarted, go to address CONT.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SS1	CØNT	

PROBLEM:

Check Stop Switch 2. If ON, stop the computer. If OFF, do not stop. In either case, jump to address JIM modified by Index Register 3.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SS2	JIM,3	

PROBLEM:

Check Stop Switch 3. If ON, stop the computer. If OFF, do not stop. In either case jump to the address specified in Index Register 6.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SS3	,6	

PROBLEM:

Check Stop Switch 2. If ON, stop the computer. If OFF, do not stop. In either case jump to one memory location forward of the address specified in Index Register 4.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SS2	1,4	

NEW CONCEPTS OF GROUP 7

The conditional jump and stop instructions allow the program to check data and switches and perform various functions accordingly.

For the A and Q jumps the whole register is checked for zero when ZR or NZ is specified. When PL or MI is specified, only the uppermost bit is checked. This falls in line with the integer and fractional formats already discussed, and floating point which is coming soon.

For the checking of the jump switches and stop switches the number of the switch checked must be on in order for the jump or stop condition to be satisfied. For the conditional jump instructions program control will either jump or continue to the next instruction. For the conditional stop instructions program control will either stop or not stop. In either case it always jumps when it continues.

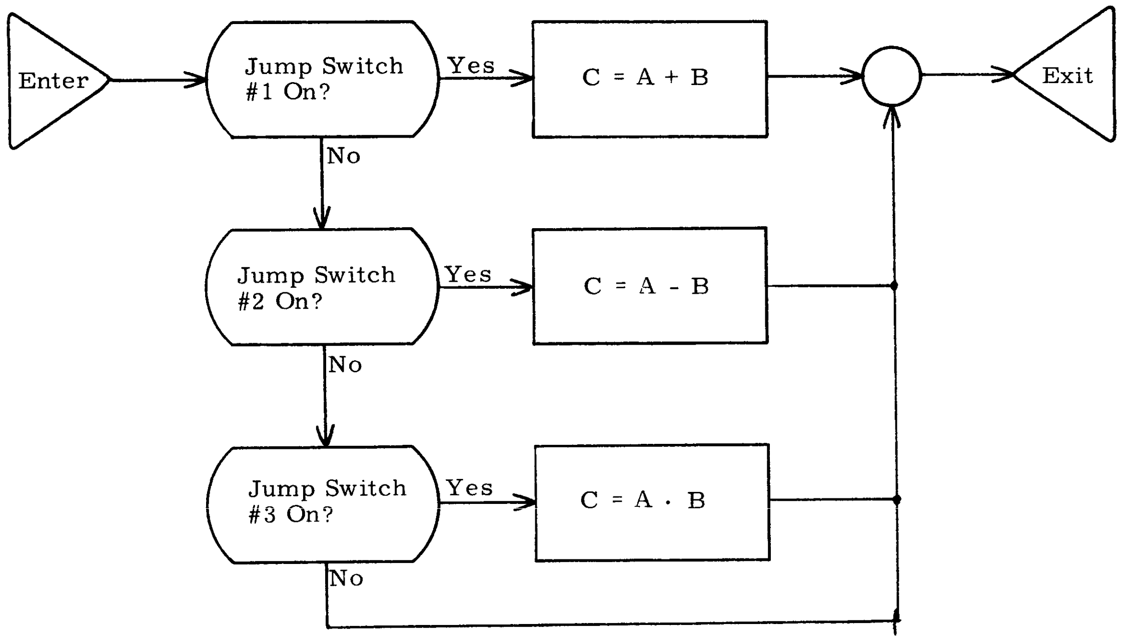
Problem 7:

Write a subprogram that will form:

1. C = A+B if jump switch #1 is on
2. C = A-B if jump switch #2 is on
3. C = A·B if jump switch #3 is on

Assume only one switch is on and that A and B are given as integers.

Flowchart:



Problem 7 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	SWITCH	
	ENTRY	SWITCH	
C	BSS	1	
SWITCH	BSS	1	
	SJ1	SUM	JUMP SW 1 ØN?
	SJ2	DIFF	JUMP SW 2 ØN?
	SJ3	PRØD	JUMP SW 3 ØN?
	SLJ	SWITCH	
SUM	LDA	A	
	ADD	B	A+B
	STA	C	
	SLJ	SWITCH	
DIFF	LDA	A	
	SUB	B	A-B
	STA	C	
	SLJ	SWITCH	
PRØD	LDA	A	
	MUL	B	AB
	STA	C	
	SLJ	SWITCH	
	END		

Somewhere within this subprogram would also be included the symbols A and B in the location field with a declaration of the prestored data or the area reserved.

Student Problem 7A:

Write a subprogram that will form:

1. Z = X+Y if jump switch 1 is on and jump switch 2 if off.
2. Z = 2X+3Y if jump switch 2 is on and jump switch 1 is off.
3. Z = 3X-2Y if both jump switches 1 and 2 are on.

Flowchart:

Problem 7A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 8

RETURN JUMPS AND STOPS

GROUP 8

RETURN JUMPS AND STOPS

1.	Unconditional Stop/Return Jumps	RTJ/SRJ
2.	Selective Return Jumps	RJX
3.	A Return Jump	ARJ
4.	Q Return Jump	QRJ
5.	Selective Stops (with Return Jumps)	SRX

This group of instructions allows the programmer to stop the computer (if not illegal), and to transfer program control to a subroutine.

It is important to note that the return jumps are not bank jump instructions in any way. The purpose of the bank term is to set or reset the Operand Bank setting only. The return jump is still within the same bank in which current program control is operating.

Some of the return jumps and stops are conditional. They are conditioned by the contents of A, the contents of Q, the Jump switches, and the Stop switches.

The UNCONDITIONAL STOP/RETURN JUMP Instructions

The RETURN JUMP instruction is an instruction that transfers program control to another point within the program. However, at that point is placed an address, such that the address forms a return path to the main program.

The point to which control is transferred is a 15-bit address M where $M = m + (V^V)$ in the same bank that the program is operating, i.e., this is not a bank jump instruction. However, execution does not start with the upper instruction at M. Instead, (P) + 1 are placed in the upper address portion of the memory word. Execution then begins with the lower instruction of the memory word.

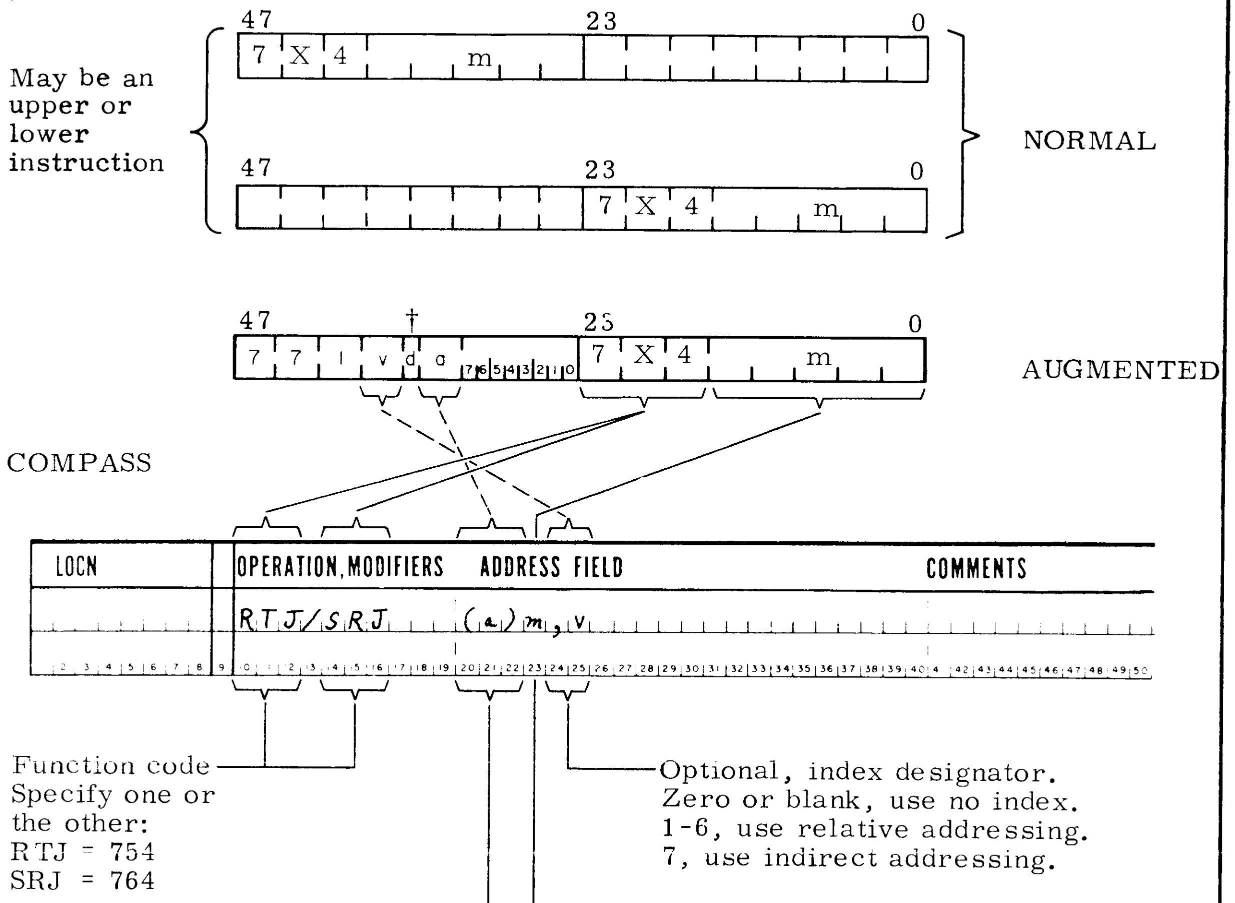
The bank term a determines the value of the operand bank setting. If it is not used, the current operand bank setting remains . If it is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

The STOP/RETURN JUMP instruction is an instruction that unconditionally stops the computer. When the computer is restarted, program control transfers to another point within the program. However, at that point is placed an address, such that the address forms a return path to the main program.

The STOP/RETURN JUMP instruction then follows the same form as the RETURN JUMP instruction (paragraphs 2 and 3 above).

FORMAT

MACHINE



Optional, index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

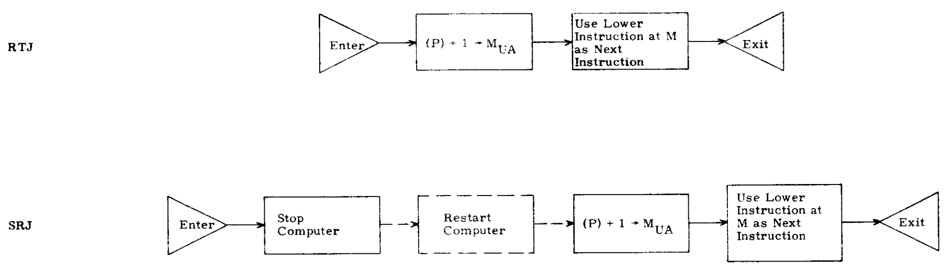
† d bit not programmable

FLOWCHART

UNCONDITIONAL STOP/RETURN JUMP

FORMAT: RTJ (a)m, v
SRJ (a)m, v

$M = m + (V^{v})$


RTJ
Enter
(P) + 1 → M_UA
Use Lower Instruction at M as Next Instruction
Exit
SRJ
Enter
Stop Computer
Restart Computer
(P) + 1 → M_UA
Use Lower Instruction at M as Next Instruction
Exit

DESCRIPTION

RTJ	Return jump to M
SRJ	Stop, return jump to M

EXAMPLES

PROBLEM:

Do an Unconditional Return Jump to address SUB.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RTJ	SUB	

PROBLEM:

Do an Unconditional Return Jump to an address specified in Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RTJ	,1	

PROBLEM:

Do an Unconditional Return Jump to address SUB modified by Index Register 2. The routine SUB is to call operands from Bank 6.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RTJ	(6) SUB, 2	

PROBLEM:

Stop the computer. When restarted, do a Return Jump to address ROUTINE.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SRJ	RØUTINE	

The SELECTIVE RETURN JUMP Instructions

The SELECTIVE RETURN JUMP instructions are conditional return jump instructions that test the three jump switches on the console to see if they are ON or OFF. There is one return jump instruction for each switch.

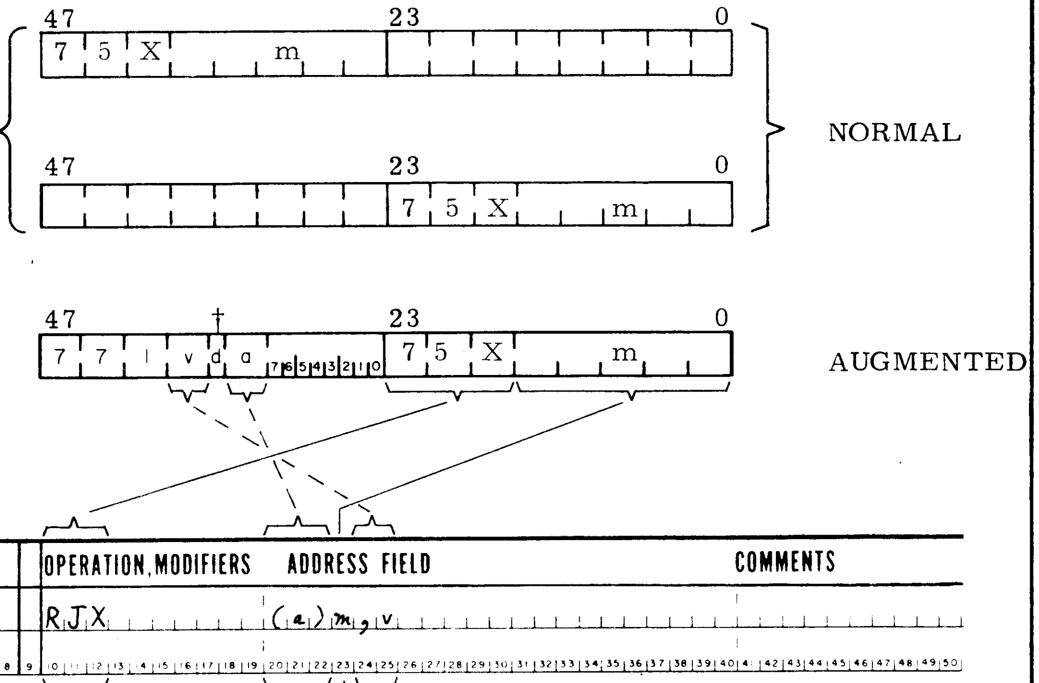
If the jump switch is ON, program control transfers to a 15-bit address M where $M = m + (V^V)$ in the same bank that the program is operating, i.e., this is not a bank jump instruction. However, execution does not start with the upper instruction at M. Instead, (P) + 1 are placed in the upper address portion of the memory word. Execution then begins with the lower instruction of the memory word. If the jump switch is OFF, program control continues executing in its normal sequence with no jump being made.

The bank term a determines the value of the operand bank setting. If it is not used, the current operand bank setting remains. If it is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

F
O
R
M
A
T

MACHINE

May be an upper or lower instruction



Function code where:
RJ1 = 755
RJ2 = 756
RJ3 = 757

Optional, index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

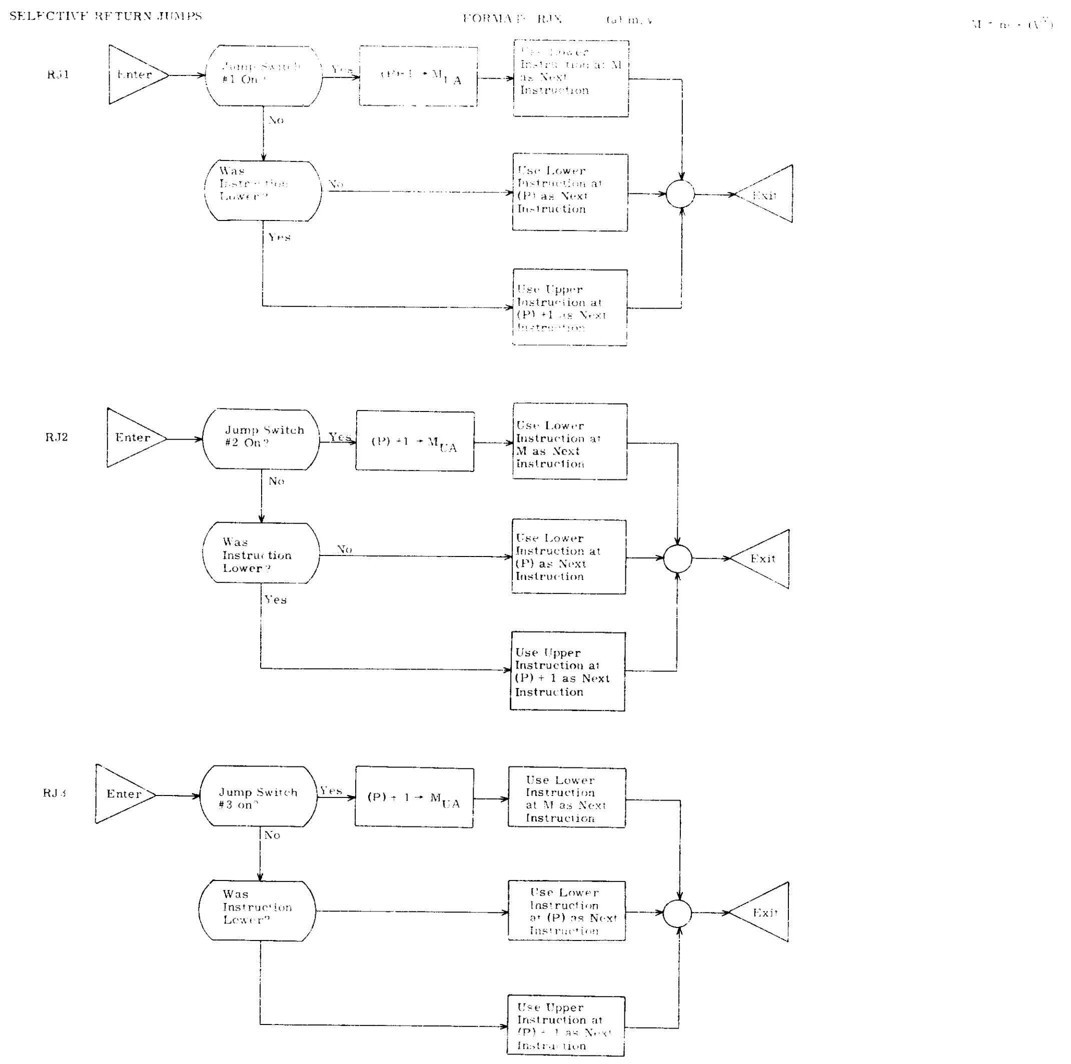
$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

RJX

FLOWCHART


SELECTIVE RETURN JUMPS
FORMAT: RJX (a) m, x
M = m + (V^x)
RJ1
Enter
Jump Switch #1 On?
Yes
(P)+1 → M_LA
Use Lower Instruction at M as Next Instruction
No
Was Instruction Lower?
No
Use Lower Instruction at (P) as Next Instruction
Yes
Use Upper Instruction at (P) +1 as Next Instruction
Exit
RJ2
Enter
Jump Switch #2 On?
Yes
(P) +1 → M_UA
Use Lower Instruction at M as Next Instruction
No
Was Instruction Lower?
No
Use Lower Instruction at (P) as Next Instruction
Yes
Use Upper Instruction at (P) + 1 as Next Instruction
Exit
RJ3
Enter
Jump Switch #3 on?
Yes
(P) + 1 → M_UA
Use Lower Instruction at M as Next Instruction
No
Was Instruction Lower?
Use Lower Instruction at (P) as Next Instruction
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

RJ1	Return jump to M if Jump Switch 1 is on, otherwise continue
RJ2	Return jump to M if Jump Switch 2 is on, otherwise continue
RJ3	Return jump to M if Jump Switch 3 is on, otherwise continue

PROBLEM:

Check Jump Switch 1. If it is ON, do a Return Jump to address PETE. If it is OFF, continue the program.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	RJ1	PETE	

PROBLEM:

Check Jump Switch 2. If it is ON, do a Return Jump to address JOE modified by Index Register 2. If it is OFF, continue the program.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	RJ2	JØE,2	

PROBLEM:

Check Jump Switch 3. If it is ON, do a Return Jump to the address specified in Index Register 4. If it is OFF, continue the program.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	RJ3	,4	

PROBLEM:

Check Jump Switch 2. If it is ON, do a Return Jump to address BILL where it is to pick up operands from Bank 3. If it is OFF, continue the program.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	RJ2	(3) BILL	

The A RETURN JUMP Instruction

The A RETURN JUMP instruction is a conditional jump instruction that tests the contents of A for zero, non-zero, positive, or negative. One modifier must be specified.

If the condition tested is true, program control transfers to a 15-bit address M where M = m + (V^v) in the same bank that the program is operating, i.e., this is not a bank jump instruction. However, execution does not start with the upper instruction at M. Instead, (P) + 1 are placed in the upper address portion of the memory word and execution then begins with the lower instruction of the memory word. If the condition tested is not true, program control continues executing in its normal sequence with no jump being made.

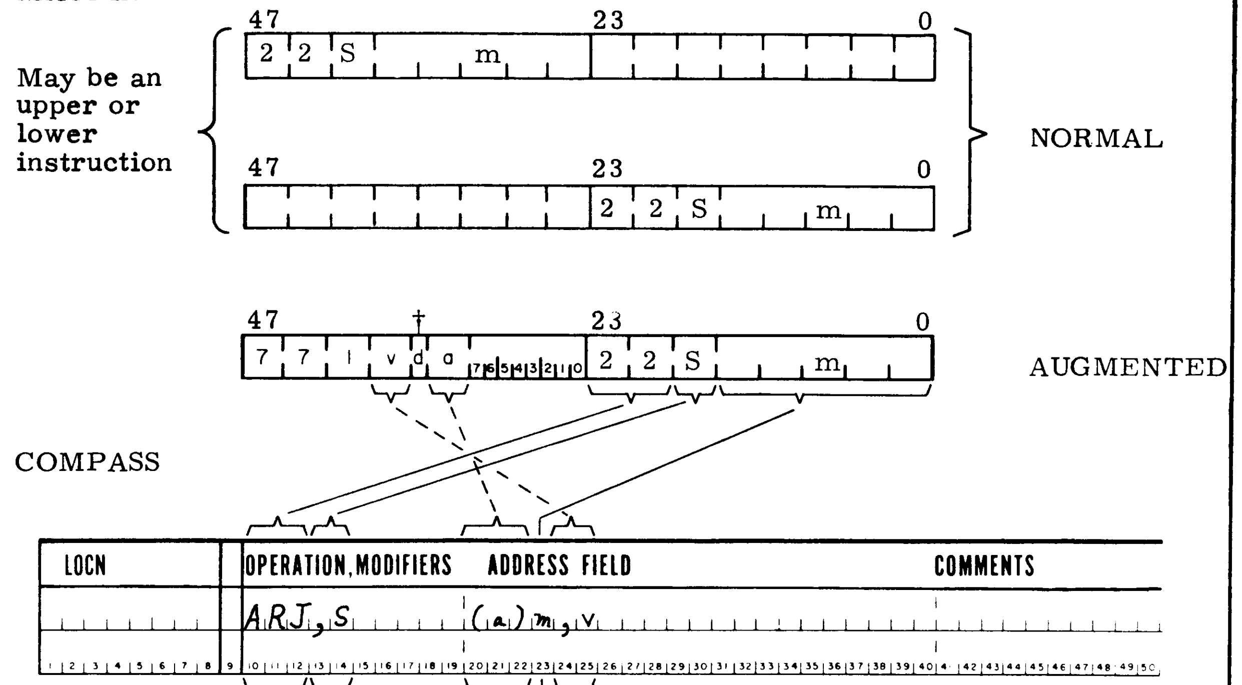
The bank term a determines the value of the operand bank setting. If it is not used, the current operand bank setting remains. If it is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

A RETURN JUMP

ARJ

FORMAT

MACHINE



Function code

Modifier; One of the following must be specified:

ZR, zero	(224)
NZ, non-zero	(225)
PL, plus	(226)
MI, minus	(227)

Optional, index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

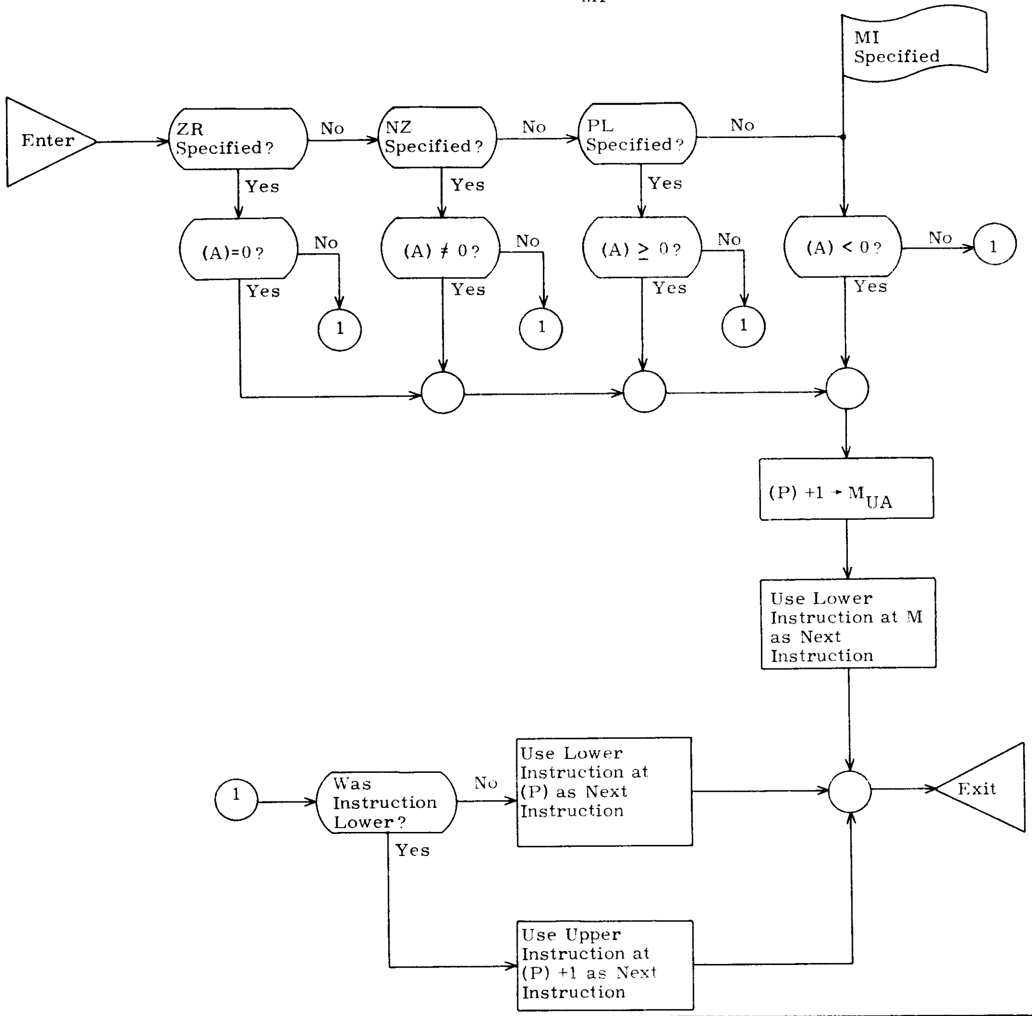
†d bit not programmable

ARJ

FLOWCHART

A RETURN JUMP

FORMAT: ARJ, ZR (a) m, v $M = m + (V^v)$
NZ
PL
MI



DESCRIPTION

ARJ, ZR	Return jump to M if (A) = 0, otherwise continue
ARJ, NZ	Return jump to M if (A) ≠ 0, otherwise continue
ARJ, PL	Return jump to M if (A) ≥ 0, otherwise continue
ARJ, MI	Return jump to M if (A) < 0, otherwise continue

PROBLEM:

A Return Jump is to be taken to address TOM if A is zero. If not, continue the program. What instruction will do this?

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ARJ,ZR	TØM	

PROBLEM:

Do a Return Jump to address ROOT if A is positive. If A is negative, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ARJ,PL	RØØT	

PROBLEM:

Do a Return Jump to address ANS modified by Index Register 3 if A is negative. If A is positive, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ARJ,MI	ANS,3	

PROBLEM:

Do a Return Jump to the address specified in Index Register 4 if A is non-zero. If A is zero, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ARJ,NZ	,4	

The Q RETURN JUMP Instruction

The Q RETURN JUMP instruction is a conditional jump instruction that tests the contents of Q for zero, non-zero, positive, or negative. One modifier must be specified.

If the condition tested is true, program control transfers to a 15-bit address M where $M = m + (V^V)$ in the same bank that the program is operating, i.e., this is not a bank jump instruction. However, execution does not start with the upper instruction at M. Instead, (P) + 1 are placed in the upper address portion of the memory word and execution then begins with the lower instruction of the memory word. If the condition tested is not true, program control continues executing in its normal sequence with no jump being made.

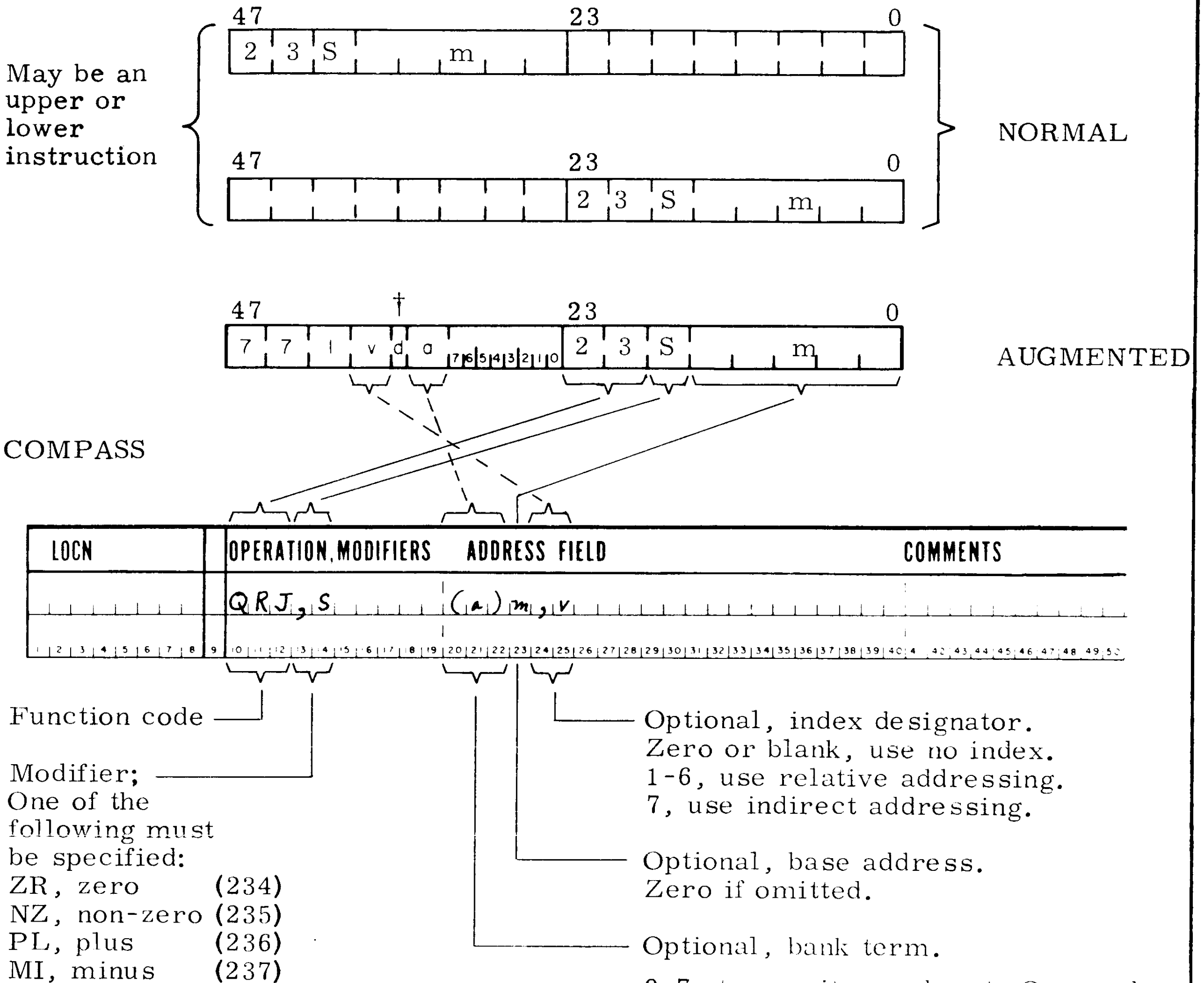
The bank term a determines the value of the operand bank setting. If it is not used, the current operand bank setting remains. If it is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

Q RETURN JUMP

QRJ

FORMAT

MACHINE



Modifier; One of the following must be specified:

ZR, zero	(234)
NZ, non-zero	(235)
PL, plus	(236)
MI, minus	(237)

Optional, index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted.

Optional, bank term.

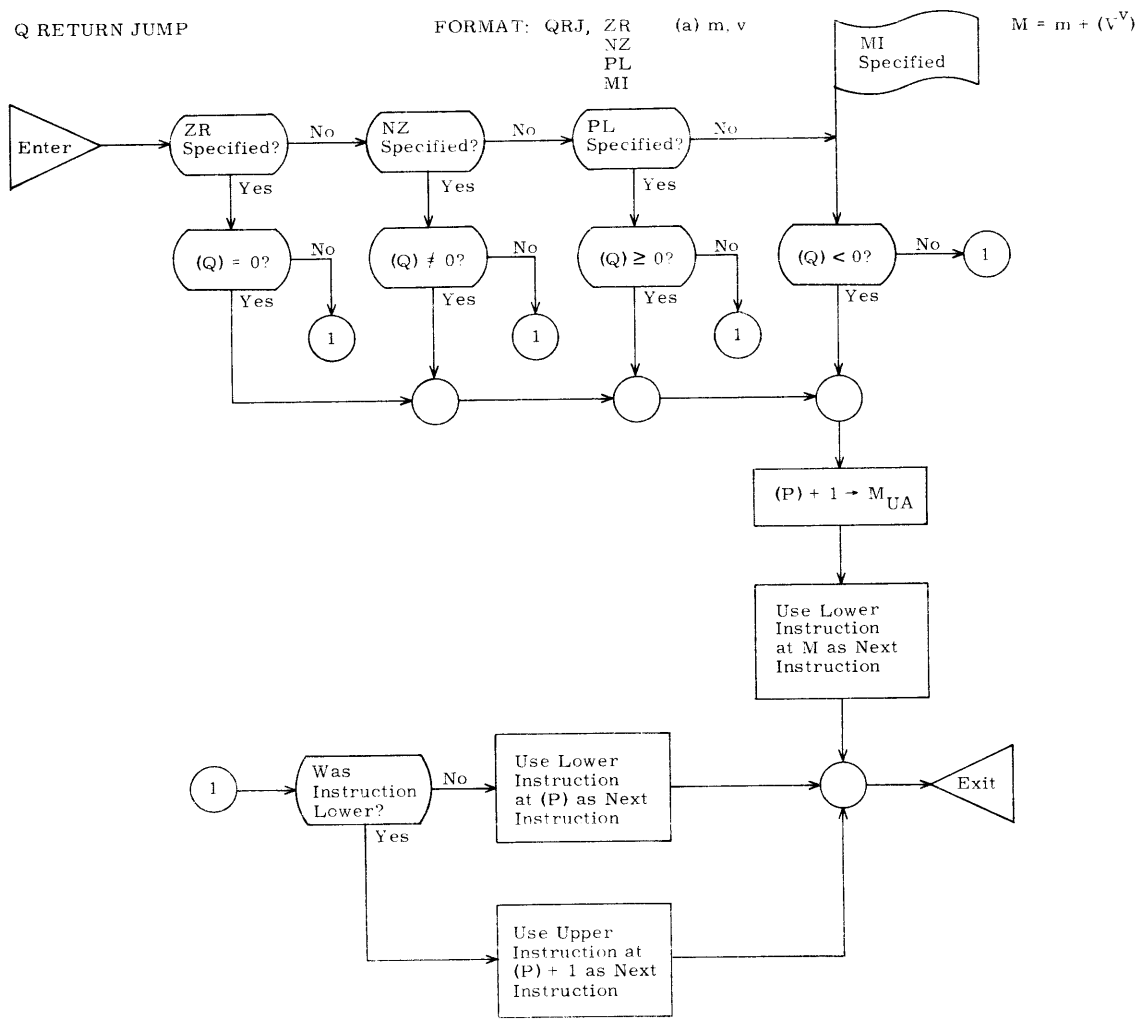
0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

F L O W C H A R T


Q RETURN JUMP
FORMAT: QRJ, ZR NZ PL MI (a) m, v
M = m + (V^v)
MI Specified
Enter
ZR Specified?
NZ Specified?
PL Specified?
No
Yes
(Q) = 0?
(Q) ≠ 0?
(Q) ≥ 0?
(Q) < 0?
1
(P) + 1 → M_UA
Use Lower Instruction at M as Next Instruction
Was Instruction Lower?
Use Lower Instruction at (P) as Next Instruction
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

D E S C R I P T I O N

QRJ, ZR	Return jump to M if (Q) = 0, otherwise continue
QRJ, NZ	Return jump to M if (Q) ≠ 0, otherwise continue
QRJ, PL	Return jump to M if (Q) ≥ 0, otherwise continue
QRJ, MI	Return jump to M if (Q) < 0, otherwise continue

PROBLEM:

A Return Jump is to be taken to address HODGE if Q is zero. If not, continue the program. What instruction will do this?

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	QRJ,ZR	HØDGE	

PROBLEM:

Do a Return Jump to address PODGE if Q is negative. If Q is positive, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	QRJ,MI	PØDGE	

PROBLEM:

Do a Return Jump to address SMODGE modified by Index Register 1 if Q is positive. If Q is negative, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	QRJ,PL	SMØDGE,1	

PROBLEM:

Do a Return Jump to the address specified in Index Register 5 if Q is non-zero. If Q is zero, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	QRJ,NZ	,5	

The SELECTIVE STOP/RETURN JUMP Instructions

The SELECTIVE STOP/RETURN JUMP instructions are conditional stop instructions that test the three stop switches to see if they are ON or OFF. There is one stop instruction for each switch.

If the stop switch is ON, the computer will stop. When the computer is restarted, program control will transfer to a 15-bit address M where $M = m + (V^V)$ in the same bank that the program is operating, i.e., this is not a bank jump instruction. However, execution does not start with the upper instruction at M. Instead (P) + 1 are placed in the upper address portion of the memory word. Execution then begins with the lower instruction of the memory word.

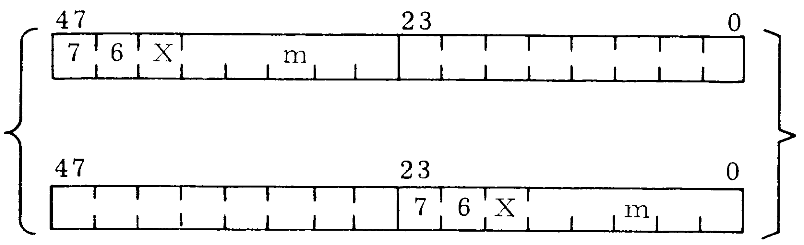
If the stop switch is OFF, the computer will not stop and the return jump will still be made.

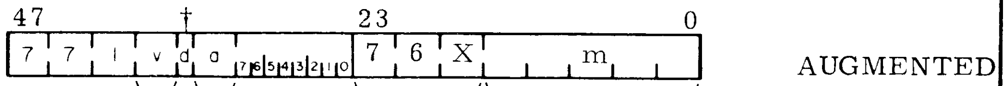
The bank term a determines the value of the operand bank setting. If it is not used, the current operand bank setting remains. If it is used, the current operand bank setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

FORMAT

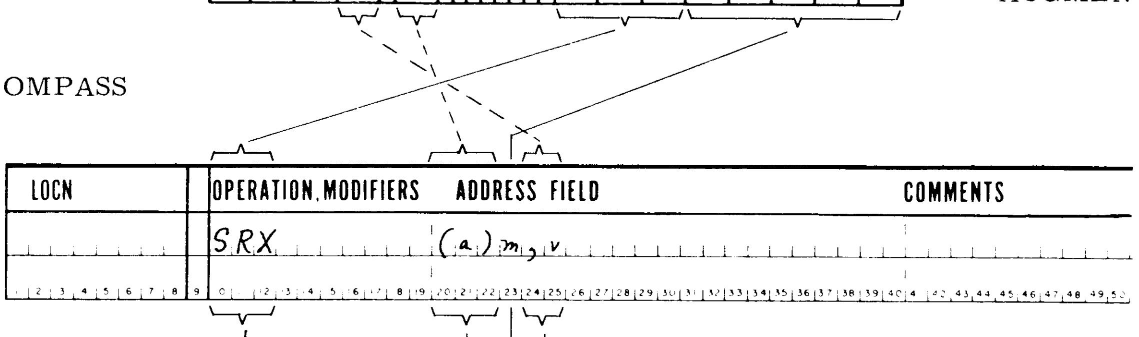
MACHINE

May be an upper or lower instruction





COMPASS



Function code where:
SR1 = 765
SR2 = 766
SR3 - 767

Optional, index designator. Zero or blank, use no index 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

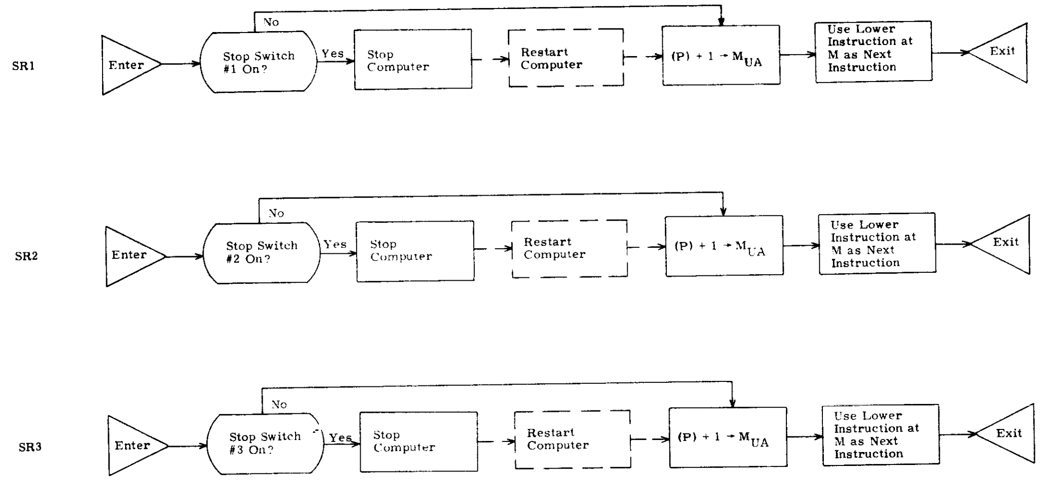
$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

FLOWCHART

SELECTIVE STOP/RETURN JUMPS FORMAT: SRX (a) m, v $M = m + (V^v)$



DESCRIPTION

SR1	Stop if Stop Switch 1 is on, always jump to M
SR2	Stop if Stop Switch 2 is on, always jump to M
SR3	Stop if Stop Switch 3 is on, always jump to M

EXAMPLES

PROBLEM:

Check Stop Switch 1. If it is ON, stop the computer. If it is OFF, do not stop the computer. In either case do a Return Jump to address SNOPS.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SR1	SNØPS	

PROBLEM:

Check Stop Switch 2. If it is ON, stop the computer. If it is OFF, do not stop the computer. In either case do a Return Jump to address POPS modified by Index Register 2.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SR2	PØPS, 2	

PROBLEM:

Check Stop Switch 3. If it is ON, stop the computer. If it is OFF, do not stop the computer. In either case do a Return Jump to the address specified in Index Register 1.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SR3	,1	

PROBLEM:

Check Stop Switch 2. If it is ON, stop the computer. If it is OFF, do not stop the computer. In either case do a Return Jump to address SPOTS modified by Index Register 2 and set the Operand Bank setting to Bank 1.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SR2	(1) SPØTS	

NEW CONCEPTS OF GROUP 8

When we introduce return jump instructions, we normally refer to them as being used in "subroutines" of the subprogram.

A subroutine is a short routine within the subprogram that is entered from the main routine, calculates answers from data set up by the main routine, and returns control back to the main routine at exactly the point from which it left.

A subroutine may be entered from various places of the main routine. Each time new data could be processed, new answers calculated, and always a return to the main routine to the point that it left off.

Let's discuss in general what happens when a "return jump" instruction takes place, how program control enters the subroutine, and how a return is made back to the main routine. Assume a return jump to address SUB as shown:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	EXAMPLE	
	•		
	•		
	•		
P	RTJ	SUB	
	•		
	•		
	•		
	•		
	•		
SUB	SLJ	**	ENTRANCE TØ SUBRØUTINE
	STA	SAVE	
	•		
	•		
	SLJ	SUB	EXIT FRØM SUBRØUTINE
	•		
	•		
	•		
	END		

When the return jump is executed at address P, the 15 bit address, P+1, is automatically stored in place of ** at address SUB by the hardware. The SLJ function code remains. If this instruction is ever executed, note how program control could return to address P+1.

The first instruction executed in the subroutine is not the SLJ but the STA. When the subroutine is finished, a jump is made to the entry address (SUB). The SLJ is then actually executed and the return is made to the main routine.

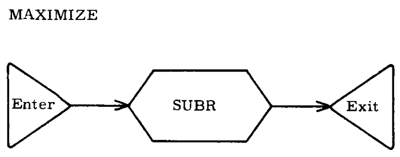
If later in the main routine, let's say at address PX, another return jump is made to SUB, address PX+1 will automatically be stored in place of **. From this you can see that a subroutine can be used at any point in the main routine and a return will be made to the correct address.

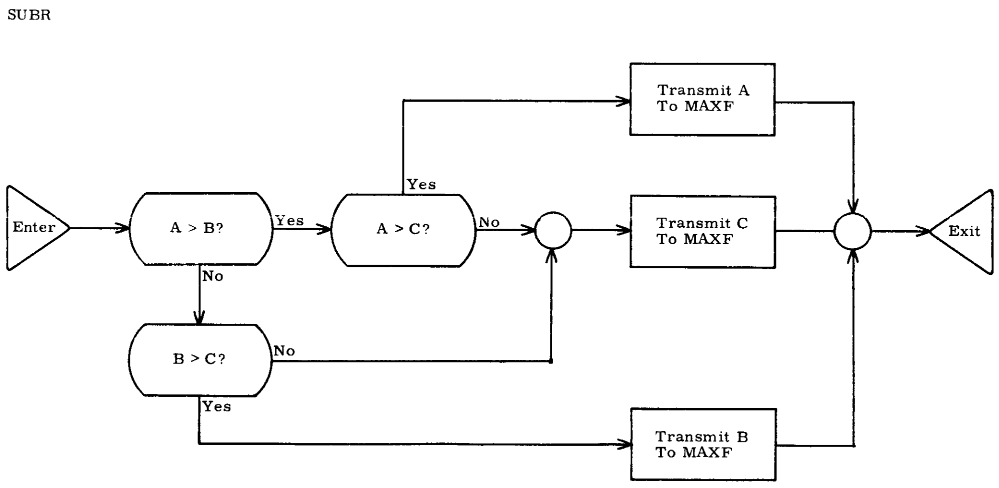
Care must be taken if the return jump instruction is an upper instruction. Since P+1 is automatically stored, a return will miss the execution of the lower instruction. For this reason the next instruction is usually forced upper (+ or symbol) or the return jump instruction is forced lower (-).

Problem 8:

A subroutine is to be written so that, when entered at any time, will determine the largest of three unequal operands. Assume the three operands to be stored at addresses A, B, and C and store the maximum operand at address MAXF.

Flowchart:





Problem 8 could be solved by coding in the following manner:

```
LOCATION  OPERATION,MODIFIERS  ADDRESS FIELD       COMMENTS
          IDENT                MAXIMIZE
          ENTRY                MAXIMIZE
MAXF      BSS                  1
MAXIMIZE  BSS                  1
          .
          .
          .
          RTJ                  SUBR
          .
          .
          .
          SLJ                  MAXIMIZE
SUBR      SLJ                  **                  ENTRANCE TØ SUBRØUTINE
          LDA                  A
          SUB                  B
          AJP,PL               AANDC               A.GT.B
          LDA                  B
          SUB                  C
          AJP,PL               TRANSB              B.GT.C
TRANSC    LDA                  C
          STA                  MAXF
          SLJ                  SUBR                EXIT SUBRØUTINE
TRANSB    LDA                  B
          STA                  MAXF
          SLJ                  SUBR                EXIT SUBRØUTINE
AANDC     LDA                  A
          SUB                  C
          AJP,PL               TRANSA              A.GT.C
          SLJ                  TRANSC
TRANSA    LDA                  A
          STA                  MAXF
          SLJ                  SUBR                EXIT SUBRØUTINE
          END
```

Somewhere within this subprogram would also be included the symbols A, B, and C in the location field with a declaration of the prestored data or the area reserved.

Student Problem 8A:

Solve problem 8 finding the smallest operand. Store at address MINF.

Flowchart:

Problem 8A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 9

REGISTER JUMPS

GROUP 9

REGISTER JUMPS

1.	Register Jump	RGJP
2.	Non-zero Bit Jump	NBJP
3.	Zero Bit Jump	ZBJP

This group of instructions consists of conditional jumps, conditioned on the contents of any operational register. If the jump condition is satisfied, a jump is taken. If the condition is not satisfied, the program continues.

The first instruction tests any operational register for some relation to a given value. This relation can be one of many used in pure and applied mathematics.

The other two instructions test any bit in any operational register for being in the set ("1") or clear ("0") state. They commonly answer the "yes or no" or "true or false" questions.

The REGISTER JUMP Instruction

The REGISTER JUMP instruction is an instruction that compares the 15-bit value y with the contents of any specified register. The designator p specifies the register compared. No memory reference is made.

There are eight possible comparisons that can be made. One of the modifiers must be specified. They are as follows:

Modifier	Comparison
1. EQ	Contents of p equal to y
2. GT	Contents of p greater than y
3. LT	Contents of p less than y
4. NE	Contents of p not equal to y
5. LE	Contents of p less than or equal to y
6. GE	Contents of p greater than or equal to y
7. LT, D	Contents of p less than y
8. GE, D	Contents of p greater than or equal to y

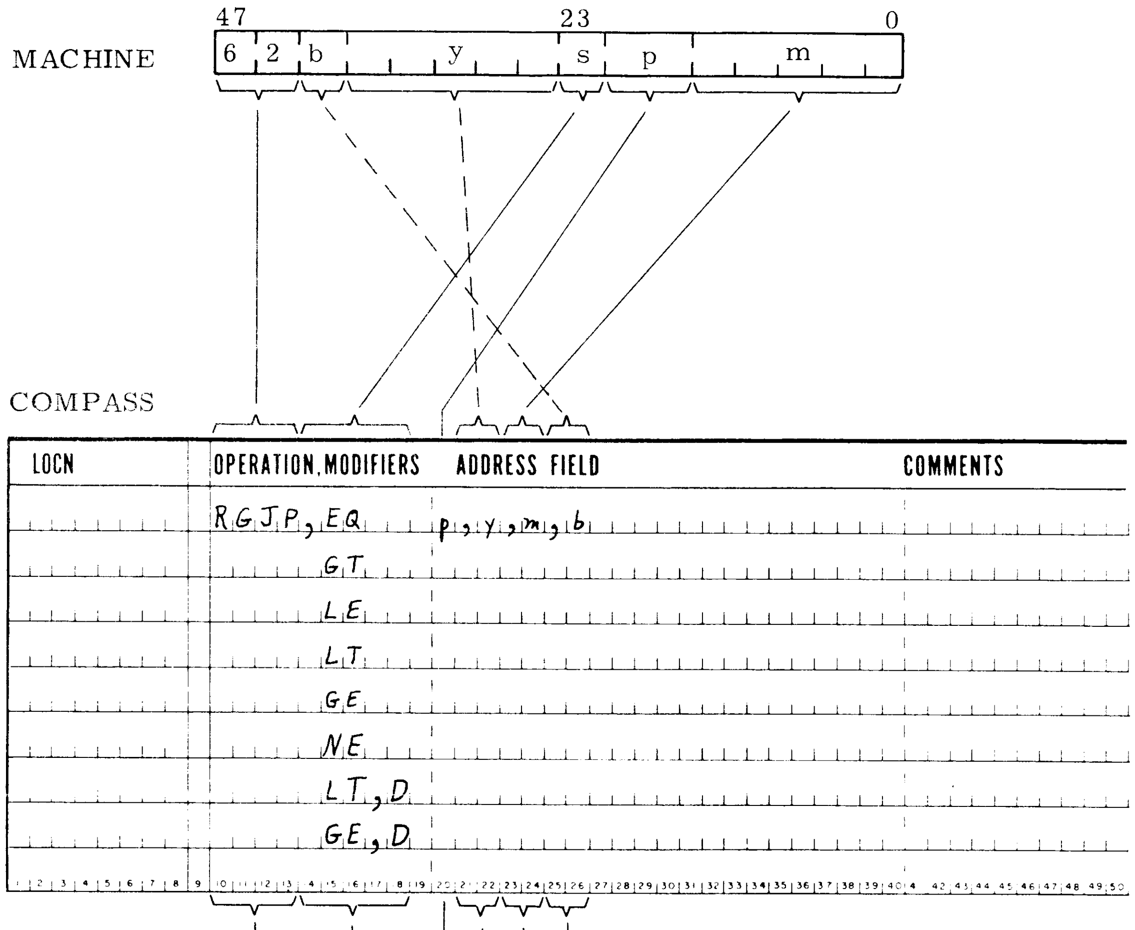
If the comparison is true, program control transfers to a 15-bit address M where M = m + (B^b). If the comparison is not true, program control continues to the next instruction. In either case all registers are left unchanged, except for the possibility of the last two modifiers being used.

With respect to the last two modifiers additional features are incorporated. When the comparison is made, and it is found that the contents of p are less than y, the contents of p are decremented by the operand y before going on to the next instruction.

REGISTER JUMP

RGJP

FORMAT



Function code

Modifier:
EQ (p) = y
GT, (p) $>$ y
LE, (p) $\leq$ y
LT, (p) $<$ y
GE, (p) $\geq$ y
NE, (p) $\neq$ y
LT,D (p) $<$ y
GE,D (p) $\geq$ y
$\left.\begin{array}{l}\text{LT,D (p)} < y\\ \text{GE,D (p)} \geq y\end{array}\right\}$ (p)-y $\rightarrow$ p for (p) $<$ y

Optional, index designator.
Blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.

Actual operand compared.

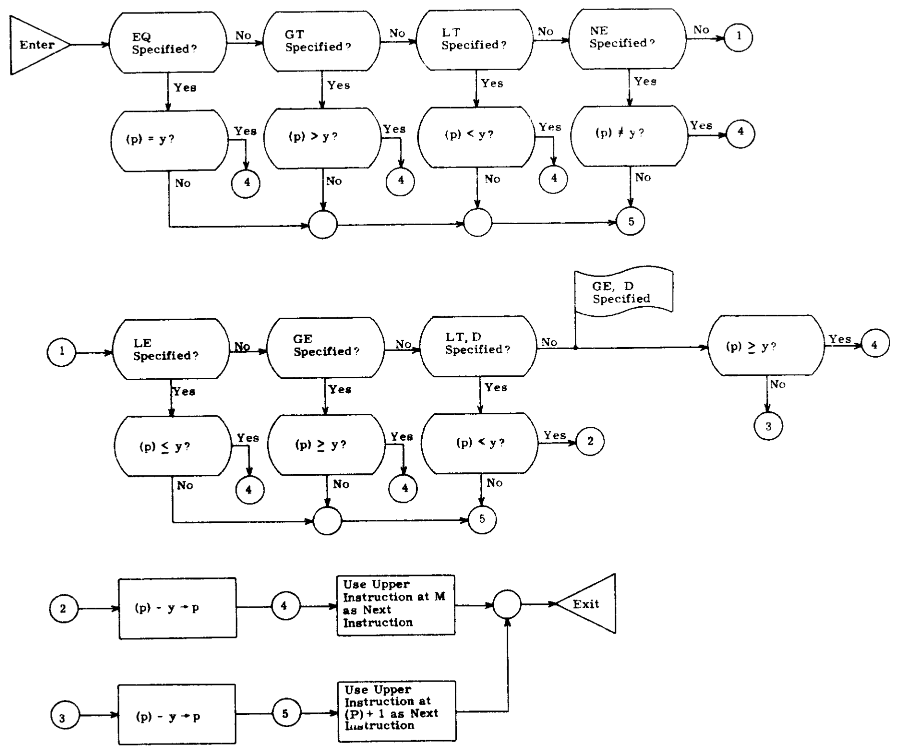
Mnemonic for specified register compared.

RGJP

FLOWCHART

REGISTER JUMP

FORMAT: RGJP, EQ;GT;LE; LT;GE; NE;LT,D;GE,D p,y,m,b $M = m + (B^b)$



DESCRIPTION

RGJP, EQ	Jump to M if (p) = y, otherwise continue.
RGJP, GT	Jump to M if (p) > y, otherwise continue.
RGJP, LT	Jump to M if (p) < y, otherwise continue.
RGJP, LE	Jump to M if (p) ≤ y, otherwise continue.
RGJP, GE	Jump to M if (p) ≥ y, otherwise continue.
RGJP, NE	Jump to M if (p) ≠ y, otherwise continue.
RGJP, LT, D	Jump to M if (p) < y, otherwise continue. Modify (p) if (p) < y.
RGJP, GE, D	Jump to M if (p) ≥ y, otherwise continue. Modify (p) if (p) < y.

EXAMPLES

PROBLEM:

Check to see if the A register contains 33 octal. If it is, jump to address ENDCARD. If not, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RGJP, EQ	A, 33B, ENDCARD	

PROBLEM:

Check to see if the D register contains 100. If yes, jump to address GETUPP modified by Index Register 4. If no, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RGJP, EQ	D, 100, GETUPP, 4	

PROBLEM:

Examine the contents of the A register. If less than or equal to nine, jump to address NUM. If greater than nine, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RGJP, LE	A, 9, NUM	

PROBLEM:

Examine the contents of the Q register. If greater than or equal to 50, jump to address OVER. If not, continue the program.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RGJP, GE	Q, 50, ØVER	

The NON-ZERO BIT JUMP Instruction

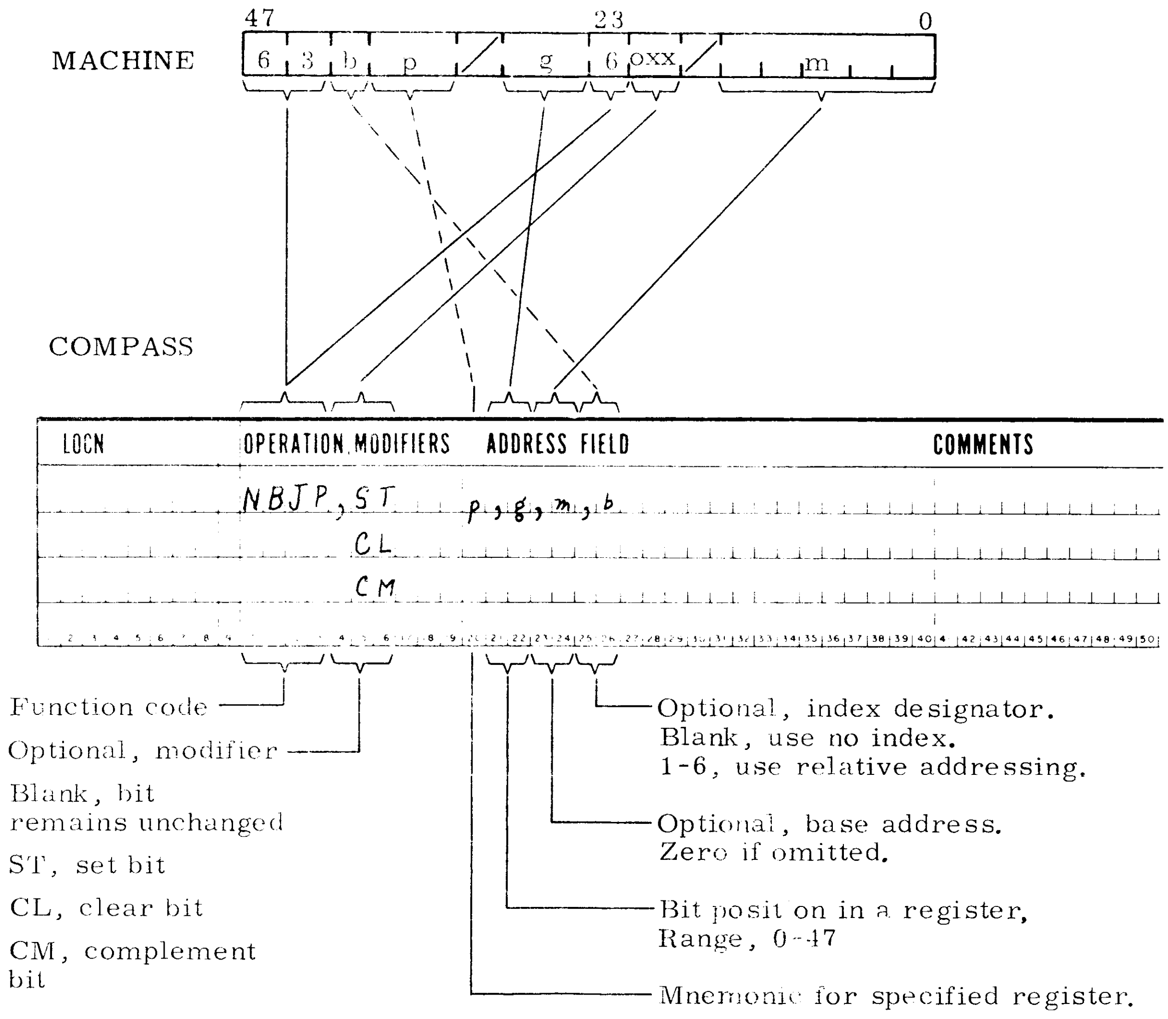
The NON-ZERO BIT JUMP instruction is an instruction that tests for a non-zero bit at any specified bit position, designated by g, in any specified register, designated by p. No memory reference is made.

If the specified bit is in the set state (binary 1), program control transfers to a 15-bit address $\underline{M}$ where $M = m + (B^h)$. If the specified bit is in the clear state (binary 0), program control continues to the next instruction. In either case, unless a modifier is used, the bit is left unchanged.

There are three modifier options available that could change the bit <u>after</u> the type of exit has been determined. They are ST, CL, and CM. ST means that the bit is to be set as the exit is taken. CL means that the bit is to be cleared as the exit is taken. CM means that the bit is to be complemented as the exit is taken.

NON-ZERO BIT JUMP | NBJP

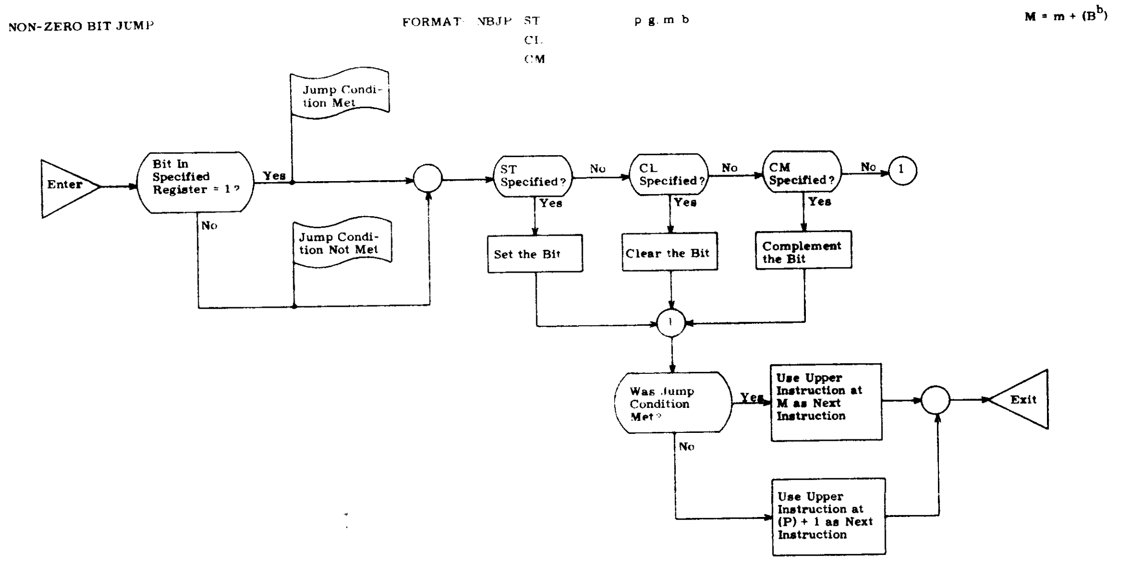
FORMAT



47
23
0
MACHINE
6
3
b
p
g
6
oxx
m
COMPASS
LOCN
OPERATION MODIFIERS
ADDRESS FIELD
COMMENTS
NBJP, ST
p, g, m, b
CL
CM
Function code
Optional, modifier
Blank, bit remains unchanged
ST, set bit
CL, clear bit
CM, complement bit
Optional, index designator. Blank, use no index. 1-6, use relative addressing.
Optional, base address. Zero if omitted.
Bit position in a register, Range, 0-47
Mnemonic for specified register.

NBJP

FLOWCHART



DESCRIPTION

NBJP	Jump to M if bit set, otherwise continue
NBJP, ST	1) Jump to M if bit set, otherwise continue 2) Always set bit after check
NBJP, CL	1) Jump to M if bit set, otherwise continue 2) Always clear bit after check
NBJP, CM	1) Jump to M if bit set, otherwise continue 2) Always complement bit after check

PROBLEM:

Examine bit 46 of the D register. If a "1", jump to address READ0. If a "0", continue the program.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		NBJP	D,46,READ0	

PROBLEM:

Examine bit 14 of Index Register 1. If a "1", jump to address WRITEON1. If a "0", continue the program. In either case clear the bit after it has been examined.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		NBJP,CL	B1,14,WRITEØN1	

PROBLEM:

Examine bit 10 of the Q register. If a "1", jump to address SWITCH modified by Index Register 3. If a "0", continue the program. In either case complement the bit after examining it

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		NBJP,CM	Q,10,SWITCH	

PROBLEM:

Set bit 5 of the Interrupt Mask register and continue the program.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		NBJP,ST	IM,5,*+1	

The ZERO BIT JUMP Instruction

The ZERO BIT JUMP instruction is an instruction that tests for a zero bit at any specified bit position, designated by g, in any specified register, designated by p. No memory reference is made.

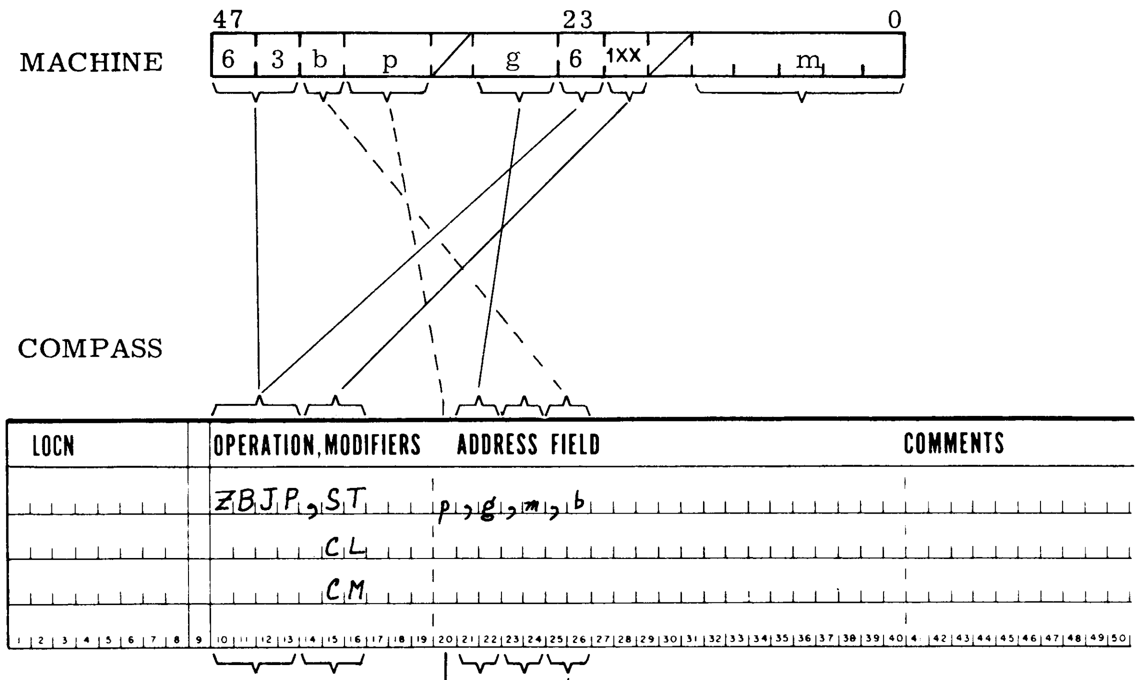
If the specified bit is in the clear state (binary 0), program control transfers to a 15-bit address $\underline{M}$ where $M=m+(B^b)$. If the specified bit is in the Set state (binary 1), program control continues to the next instruction. In either case, unless a modifier is used, the bit is left unchanged.

There are three modifier options available that could change the bit after the type of exit has been determined. They ST, CL, and CM. ST means that the bit is to be set as the exit is taken. CL means that the bit is to be cleared as the exit is taken. CM means that the bit is to be complemented as the exit is taken.

FORMAT

MACHINE

COMPASS



Function code

Optional, modifier

Blank, bit remains unchanged

ST, set bit

CL, clear bit

CM, complement bit

Optional, index designator. Blank, use no index. 1-6, use relative addressing.

Optional, base address. Zero if omitted.

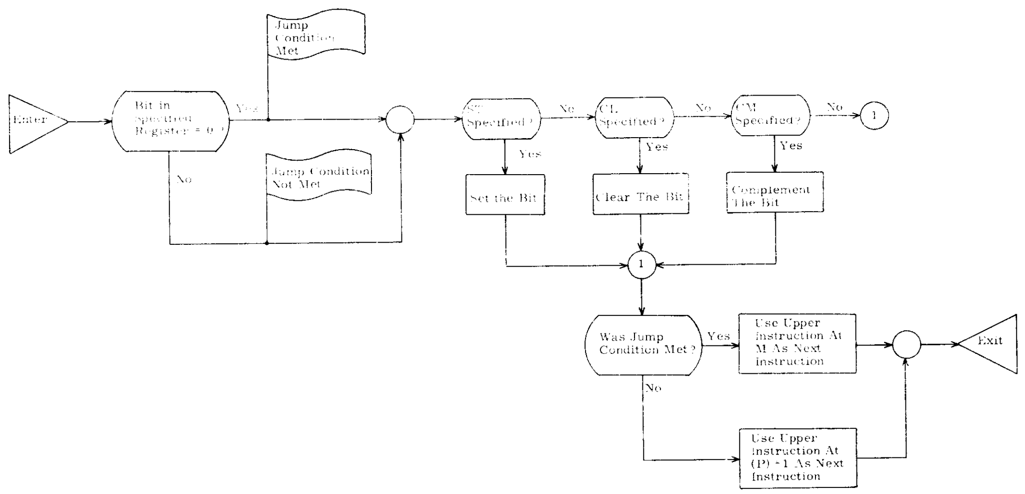
Bit position in a register, Range, 0-47

Mnemonic for specified register.

ZBJP

FLOWCHART

ZERO BIT JUMP　　　FORMAT: ZBJP, ST / CL / CM　　p, g, m, b　　$M = m + (B^b)$



DESCRIPTION

ZBJP	Jump to M if bit clear, otherwise continue
ZBJP, ST	1) Jump to M if bit clear, otherwise continue 2) Always set bit after check
ZBJP, CL	1) Jump to M if bit clear, otherwise continue 2) Always clear bit after check
ZBJP, CM	1) Jump to M if bit clear, otherwise continue 2) Always complement bit after check

PROBLEM:

Examine bit 23 of the A register. If a "0", jump to address LOWERPT. If a "1", continue the program.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		ZBJP	A,23,LØWERPT	

PROBLEM:

Examine bit 39 of the D register. If a "0", jump to address NEXTPARAM modified by Index Register 4. If a "1", continue the program.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		ZBJP	D,39,NEXTPARAM,4	

PROBLEM:

Examine bit 13 of Index Register 3. If a "0", jump to address PARERROR. If a "1", continue the program. In either case clear the bit after examining it.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		ZBJP,CL	B3,13,PARERRØR	

PROBLEM:

Set bit 1 of the A register and continue the program.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		ZBJP,ST	A,1,*+1	

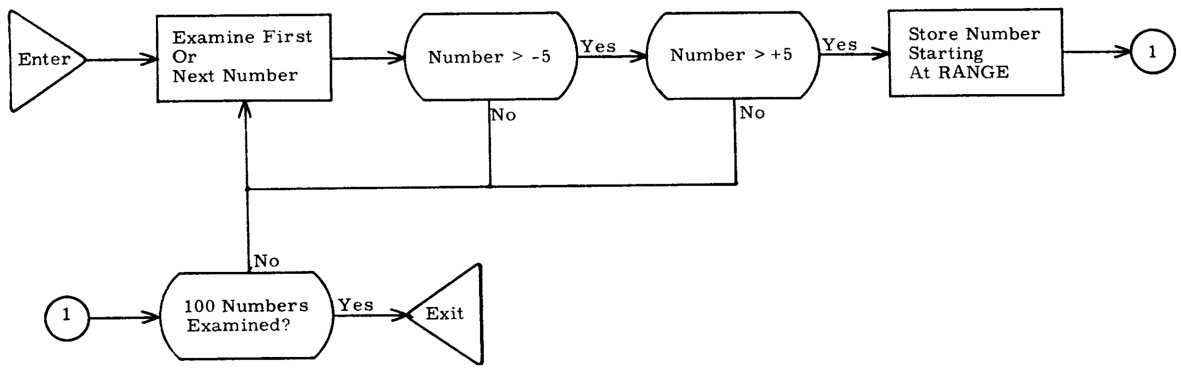
NEW CONCEPTS OF GROUP 9

The three instructions in this group check a bit or a quantity in a register. Any operational register can be checked. This includes A, Q , B1-B6, D, P, and bank registers. Since no memory reference is made, these are termed fast instructions.

Problem 9:

A table of 100 random integers ranging from -100 to +100 is in memory starting at address FLUX. Write a subprogram that will pick out all integers between but not including -5 and +5. Store them beginning at address RANGE.

Flowchart:



Problem 9 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	SEPARATE	
	ENTRY	SEPARATE	
RANGE	BSS	100	
SEPARATE	BSS	1	
	ENI	0,1	
	ENI	0,2	
NEXTWØRD	LDA	FLUX,1	
	RGJP,GT	A,-5,LT	NUM.GT.-5
CKCNT	ISK	99,1	NØ,.LE.
	SLJ	NEXTWØRD	
	SLJ	SEPARATE	
LT	RGJP,LT	A,5,ENT	NUM.LT.5
	SLJ	CKCNT	NØ,.GE.
ENT	STA	RANGE,2	
	INI	1,2	
	SLJ	CKCNT	
	END		

Somewhere within this subprogram would also be included the symbol FLUX in the location field with a declaration of the prestored data or area reserved.

Student Problem 9A:

Solve problem 9 by using the instruction LDA, CM. Also see if you can shorten the number of instructions used to solve the problem.

Flowchart:

Problem 9A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 10

BANK JUMPS

GROUP 10

BANK JUMPS

1.	Execute	EXEC
2.	Unconditional Bank Jump	UBJP
3.	Bank Jump Lower	BJPL
4.	Bank Return Jump	BRTJ
5.	Bank Jump and Set Index	BJSX

This group of instructions is the only group that has the ability to transfer program control to another bank.

The first instruction allows a maximum of two instructions to be executed from another bank. The second instruction transfers program control unconditionally to another bank.

The third instruction is the only instruction that will transfer program control to the lower instruction in another bank or within the same bank.

The last two are both bank return jumps; however, the last one allows the programmer to easily place parameters in a subroutine.

The EXECUTE Instruction

The EXECUTE instruction is an instruction that executes one 48-bit or two 24-bit instructions at an address M where $M = m + (B^b) + (V^v)$.

If the bank term a is used, the operand bank setting is replaced by the value a. If the bank term is not used, the operand bank setting remains.

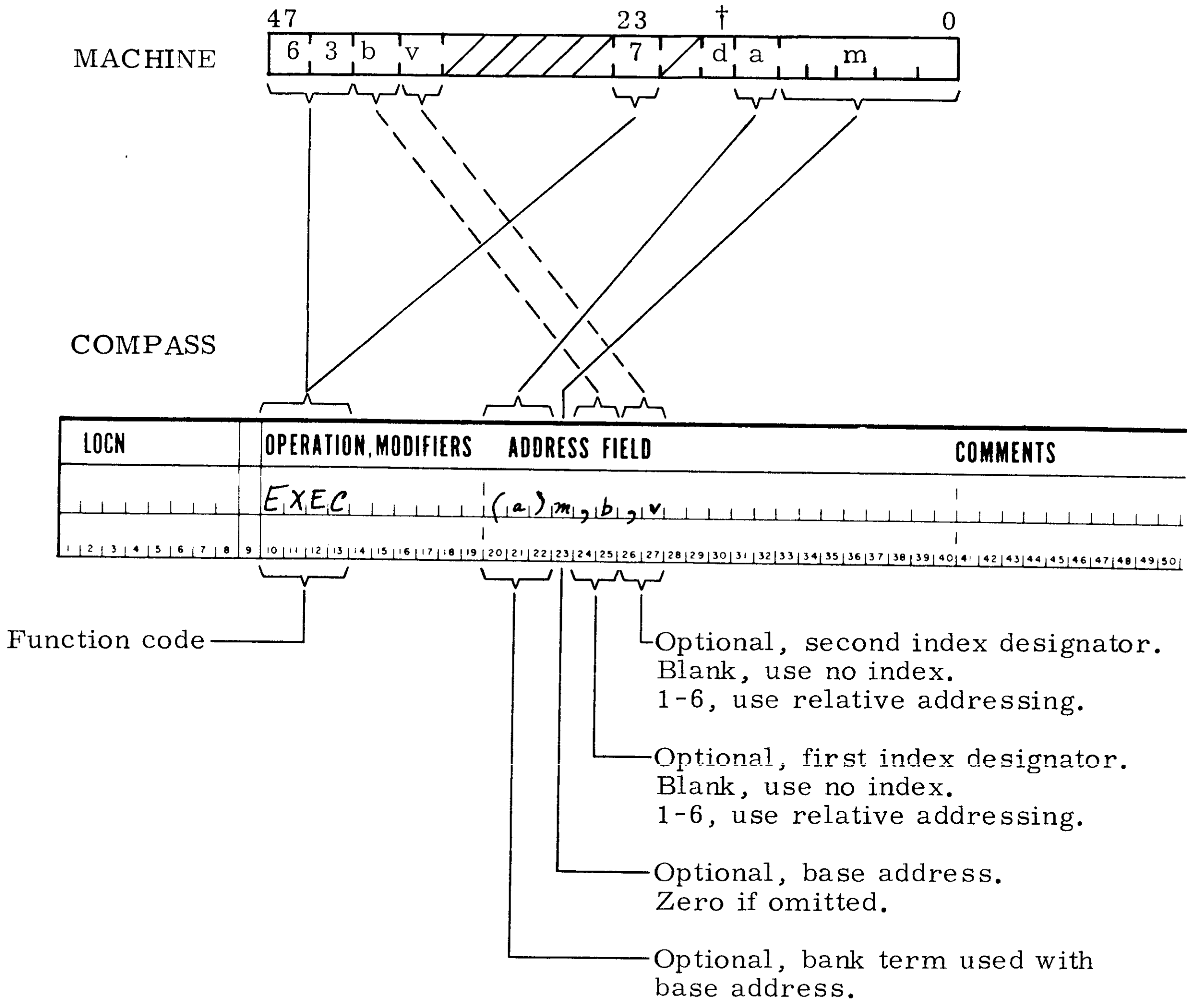
The Instruction Bank Register and the P Register do not change for this instruction even though program control "seems" to transfer to address M. The instruction(s) are at address M in the bank specified by the operand bank setting. This is the only exception to the normal rules for the instruction bank setting and the operand bank setting.

After the instruction(s) are executed, program control resumes at (P) + 1 which is the instruction immediately following the EXECUTE instruction.

EXECUTE

EXEC

FORMAT



MACHINE
47
23
†
0
6
3
b
v
7
d
a
m
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
EXEC
(a) m, b, v
Function code
Optional, second index designator. Blank, use no index. 1-6, use relative addressing.
Optional, first index designator. Blank, use no index. 1-6, use relative addressing.
Optional, base address. Zero if omitted.
Optional, bank term used with base address.

†d bit not programmable

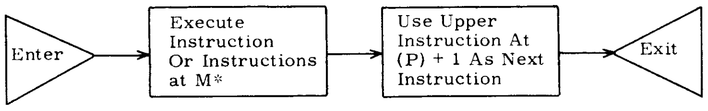
EXEC

FLOWCHART

EXECUTE

FORMAT: EXEC (a)$\dot{m}$, b, v

$M = m + (B^b) + (V^v)$


Enter
Execute Instruction Or Instructions at M*
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

* Maximum of two

DESCRIPTION

EXEC Execute maximum of two instructions at M

PROBLEM:

Execute a pair of instructions at address SOPTS.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EXEC	SØPTS	

PROBLEM:

Execute a pair of instructions at the address specified in Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EXEC	,1	

PROBLEM:

Execute a 48-bit instruction at address TAB modified by Index Register 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EXEC	TAB,2	

PROBLEM:

Execute a pair of instructions at address PUDGE in the bank that PUDGE resides.

SOLUTION:

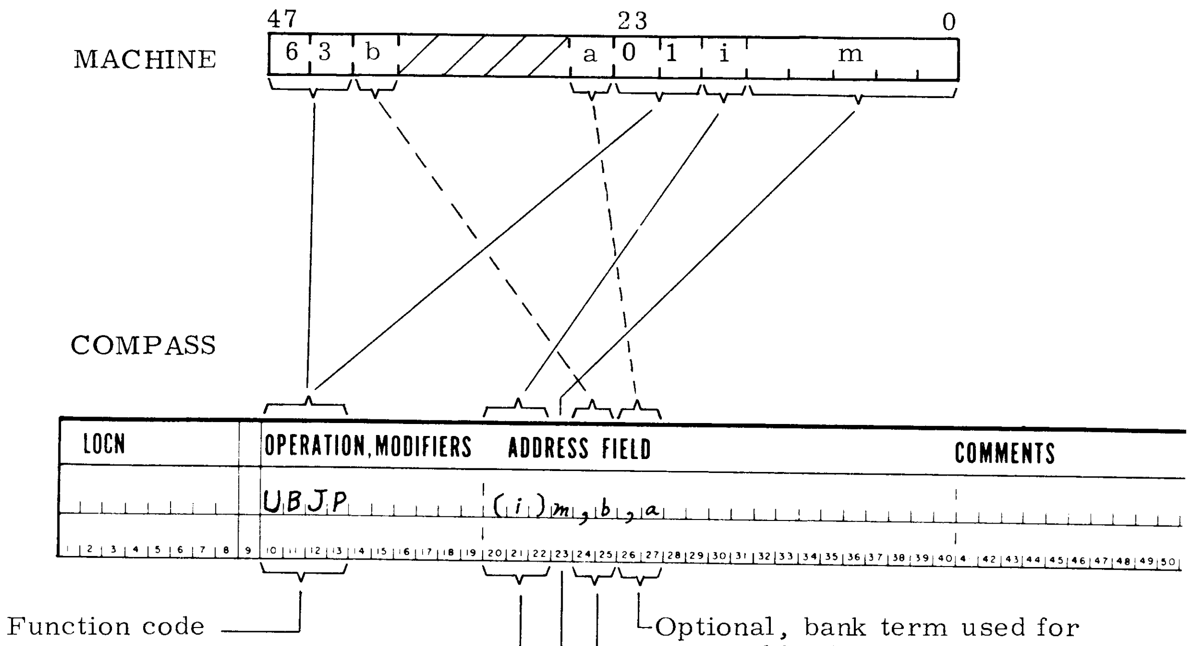
LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EXEC	($) PUDGE	

The UNCONDITIONAL BANK JUMP Instruction

The UNCONDITIONAL BANK JUMP instruction is an instruction that transfers program control to a 15-bit address in bank _i_ (within parentheses).

The jump address is _M_ where $M = m + (B^b)$. As the jump takes place, the instruction bank setting is replaced by the value _i_ and the operand bank setting is replaced by the value _a_.

FORMAT



Optional, bank term used for operand bank setting.

Current bank setting if omitted.

$, bank in which base address resides.

*, bank in which UBJP resides.

Optional, index designator.
Omitted, use no index.
1-6, relative addressing.

Optional, base address.
Zero if omitted.

Optional, bank term used with base address.

Current bank setting if omitted.

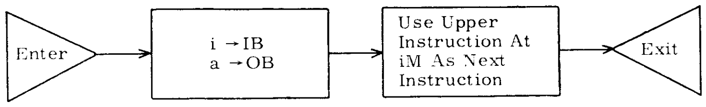
$, bank in which base address resides.

*, bank in which UBJP resides.

UBJP

FLOWCHART

UNCONDITIONAL BANK JUMP FORMAT: UBJP (i)m, b, a $M = m + (B^b)$



DESCRIPTION

UBJP Unconditional bank jump to M in bank i and set Operand Bank to a

EXAMPLES

PROBLEM:

Do a bank jump to Bank 2 address TAX and set the Operand Bank setting to Bank 1.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	UBJP	(2)TAX,,1	

PROBLEM:

Do a bank jump to Bank 2 address TAX modified by Index Register 1 and set the Operand Bank setting to Bank 1.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	UBJP	(2)TAX,1,1	

PROBLEM:

Do a bank jump to address TEST in the bank where TEST resides and switch the Operand Bank setting to the bank where the UBJP resides.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	UBJP	(#)TEST,,*	

PROBLEM:

Do a bank jump to address SMOKES in Bank 2 modified by Index Register 5 and set the Operand Bank setting to Bank 3.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	UBJP	(2)SMØKES,5,3	

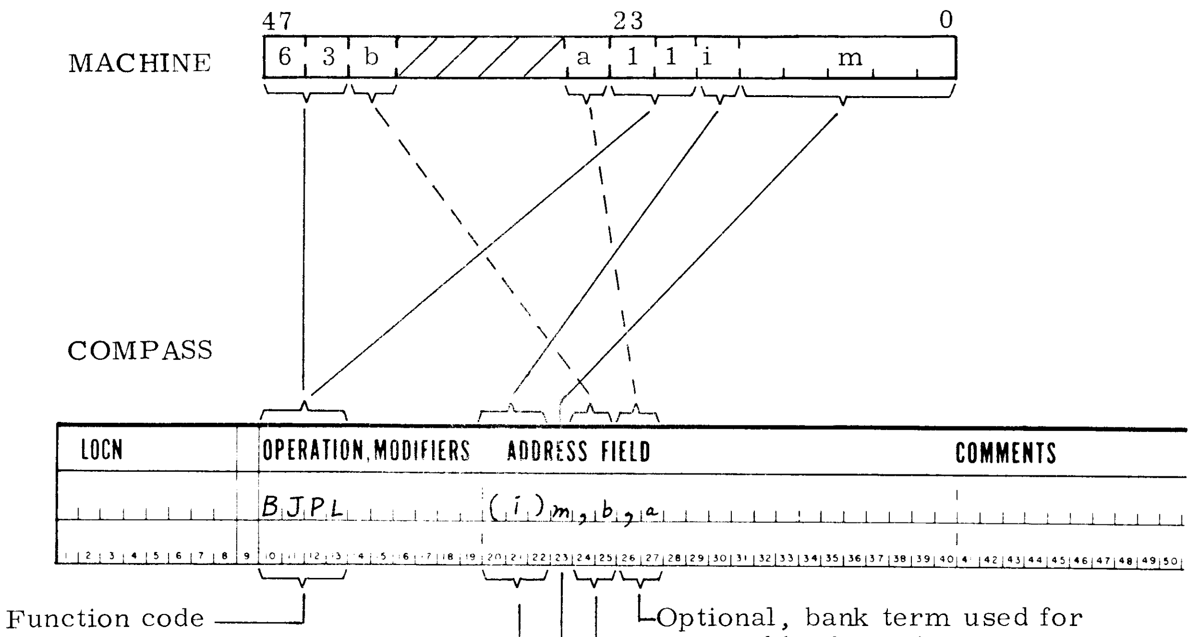
The BANK JUMP LOWER Instruction

The BANK JUMP LOWER instruction is an instruction that unconditionally transfers program control to the lower instruction of a memory word. It is the only instruction in the 3600 repertoire of instructions that will jump to the lower instruction of a memory word.

The jump is to a 15-bit address $\underline{M}$ where $M = m + (B^b)$ in bank $\underline{i}$. As the jump takes place, the instruction bank setting is replaced by the value $\underline{i}$ and the operand bank setting is replaced by the value $\underline{a}$.

If the bank terms are not specified by the programmer, the current bank settings will remain unchanged.

FORMAT



Optional, bank term used for operand bank setting.

Current operand bank setting if omitted.

$, bank in which base address resides.

*, bank in which BJPL resides.

Optional, index designator.
Omitted, use no index.
1-6, use relative addressing.

Optional. base address.
Zero if omitted.

Optional, bank term used with base address.

Current instruction bank setting if omitted.

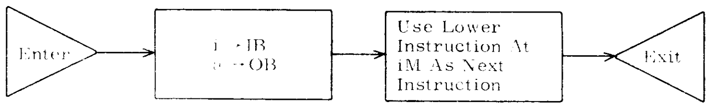
$, bank in which base address resides.

*, bank in which BJPL resides.

BJPL

FLOWCHART

BANK JUMP LOWER FORMAT: BJPL (i)m, b, a $M = m + (B^b)$



DESCRIPTION

BJPL Bank jump to lower instruction at M in bank i and set Operand Bank to a

EXAMPLES

PROBLEM:

Jump to the lower instruction at address DAVESS.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		BJPL	DAVESS	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Jump to the lower instruction at the address specified in Index Register 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		BJPL	,3	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Jump to the lower instruction at address DAMAN in the bank where DAMAN resides and set the Operand Bank to Bank 2.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		BJPL	($) DAMAN,,2	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Jump to the lower instruction at address PETE in the bank where PETE resides and set the Operand Bank to the bank where the BJPL resides.

SOLUTION:

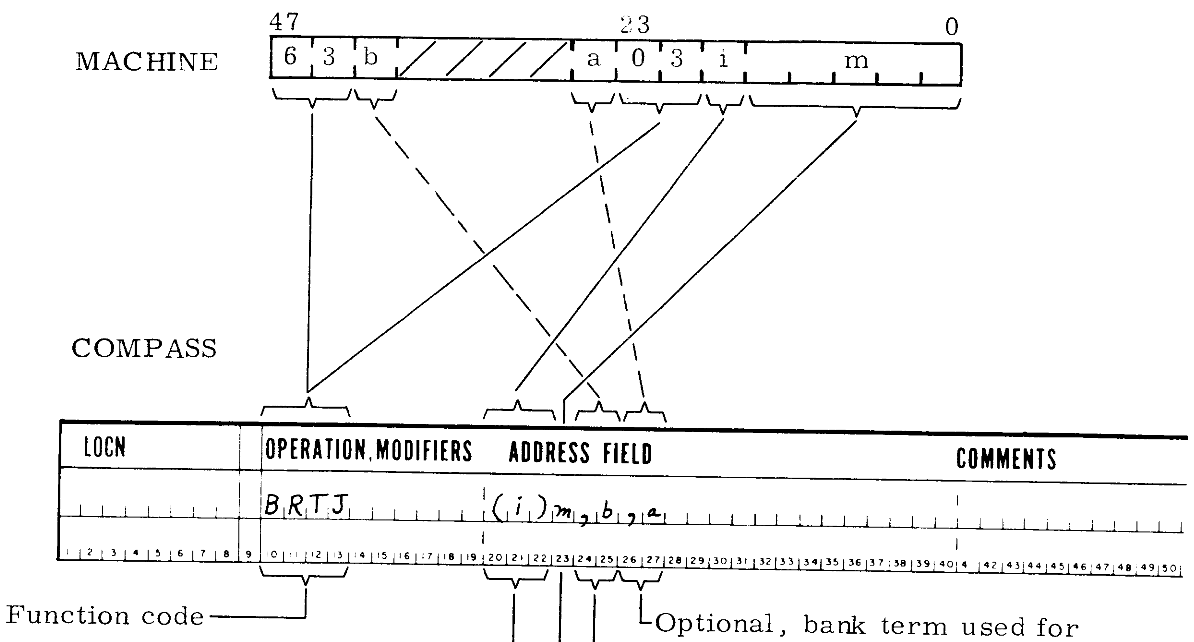
LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		BJPL	($) PETE,,*	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The BANK RETURN JUMP Instruction

The BANK RETURN JUMP instruction is an instruction that unconditionally transfers program control to the upper instruction at address M + 1 where M = m + (B^b) in bank i and sets the operand bank to the value a.

The hardware places (at address M) a "return" instruction such that, when executed, will return control to the upper instruction immediately following the BANK RETURN JUMP instruction. The instruction placed at M is an UNCONDITIONAL BANK JUMP to (P) + 1 which will also return the two bank registers to the state they were in before the BANK RETURN JUMP was executed.

FORMAT



Optional, bank term used for operand bank setting.

Current operand bank setting if omitted.
$, bank in which base address resides.
*, bank in which BRTJ resides.

Optional, index designator.
Omitted, use no index.
1-6, use relative addressing.

Optional, base address.
Zero if omitted.

Optional, bank term used with base address.

Current instruction bank setting if omitted.

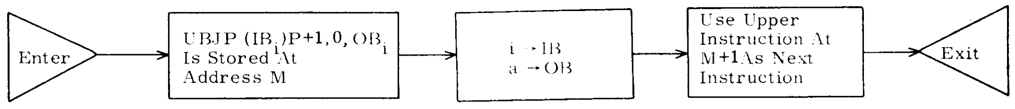
$, bank in which base address resides.

*, bank in which BRTJ resides.

BRTJ

FLOWCHART

BANK RETURN JUMP FORMAT: BRTJ (i)m, b, a $M = m + (B^b)$


Enter
UBJP $(IB_i)P+1, 0, OB_i$ Is Stored At Address M
$i \rightarrow IB$
$a \rightarrow OB$
Use Upper Instruction At M+1As Next Instruction
Exit

DESCRIPTION

BRTJ Bank return jump to M in bank <u>i</u> and set Operand Bank to <u>a</u>

PROBLEM:

Do a Bank Return Jump to address FOX in Bank 1 and set the Operand Bank to Bank 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BRTJ	(1) FØX,,2	

PROBLEM:

Do a Bank Return Jump to address FOX modified by Index Register 2 in Bank 1 and set the Operand Bank to Bank 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BRTJ	(1) FØX,2,3	

PROBLEM:

Do a Bank Return Jump to address SQRTF in the bank where SQRTF resides and set the Operand Bank to the bank in which the BRTJ resides.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BRTJ	(*) SQRTF,,*	

PROBLEM:

Do a Bank Return Jump to address TAB modified by Index Register 1 in the bank where TAB resides and set the Operand Bank to the bank in which the BRTJ resides.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BRTJ	(*) TAB,1,*	

The BANK JUMP AND SET INDEX Instruction

The BANK JUMP AND SET INDEX instruction is an instruction that unconditionally transfers program control to the upper instruction at address m + 1 in bank i and sets the operand bank to the initial value of the instruction bank.

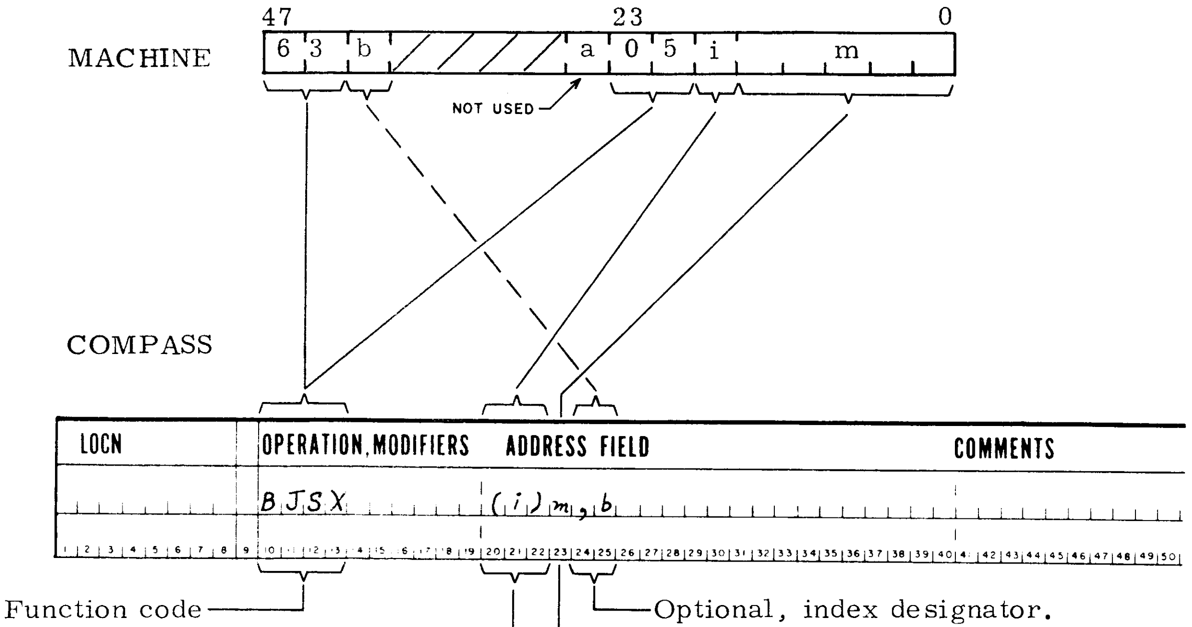
The hardware places (at address m) a "return" instruction such that, when executed, will return control to the upper instruction somewhere following the BANK JUMP AND SET INDEX instruction. To explain this, the sequence of operations by the hardware is very important.

1. UBJP $(IB_i)0$, b, OB_i is placed at address m. The index designator is the same one specified in the instruction.
2. The contents of the P register are placed in the specified index register.
3. The initial instruction bank setting is placed in the operand bank.
4. The instruction bank setting is replaced by the value i.
5. Program control goes to m + 1 in bank i.

This then is a BANK RETURN JUMP with the added features.

1. The operands used by the subroutine will be in the bank of the main routine, usually following immediately the BANK JUMP AND SET INDEX instruction.
2. Before returning to the main routine, if the programmer increments the contents of the specified index register by 1 more than the number of operand addresses used, he can jump "around" his data and his program will continue with his next executable instruction.

FORMAT



Optional, index designator.
Omitted, use no index.
1-6, (P) saved here.

Optional, base address.
Zero if omitted.

Optional, bank term used with base address.

Current instruction bank setting if omitted.

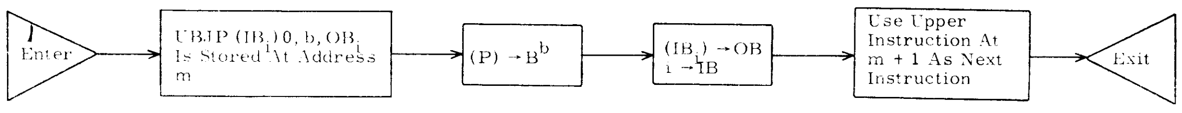
$, bank in which base address resides.

*, bank in which BJSX resides.

BJSX

FLOWCHART

BANK JUMP AND SET INDEX FORMAT: BJSX (i)m, b


Enter
UBJP (IB_i) 0, b, OB_i Is Stored At Address m
$(P) \rightarrow B^b$
$(IB_i) \rightarrow OB$
$i \rightarrow IB$
Use Upper Instruction At m + 1 As Next Instruction
Exit

DESCRIPTION

BJSX

1) Bank return jump to m in bank i and set Operand Bank to value of initial Instruction Bank

2) $P \rightarrow B^b$

PROBLEM:

Do a Bank Jump and Set Index to address PETE in Bank 2. Contain the present address of this instruction in Index Register 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BJSX	(2) PETE,3	

PROBLEM:

Do a Bank Jump and Set Index to address TOM in the bank where TOM resides and use Index Register 2 for retaining the address of the BJSX instruction.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BJSX	($) TØM,2	

PROBLEM:

Do a Bank Jump and Set Index to address JOE in Bank 4 and use Index Register 5 for retaining the address of the BJSX instruction.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BJSX	(4) JØE,5	

PROBLEM:

Do a Bank Jump and Set Index to address SUB in the bank where SUB resides. Index Register 6 is used for retaining the address of the BJSX instruction.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BJSX	($) SUB,6	

NEW CONCEPTS OF GROUP 10

The BANK JUMP instructions allow the programmer to transfer program control to any bank in a multi-bank system. If subprograms are self-contained in one bank, there is no need for the BANK JUMP instructions.

At this time we do not want to become deeply involved with the use of these instructions. The reason is that we have not yet discussed the communication between subprograms in different banks. The concept of subprograms involving a multi-bank system will be discussed later in this section.

The EXEC instruction is a quick and effective instruction to execute a short "subroutine" (two 24-bit instructions or one 48-bit instruction) without leaving main control. We can execute the instruction or instructions at the address specified by the EXEC instruction in any bank, and when they are finished, continue program control to the instruction following EXEC.

The BJPL instruction is the only instruction that can jump to the lower instruction without changing the upper instruction. The jump can be made without changing banks by simply not specifying the bank terms.

The BRTJ instruction allows subroutines, even subprograms, to be executed in another bank, and control returned to the bank of the calling subprogram automatically. When the BRTJ is executed to some address, execution starts at that address plus 1. At the address is placed a UBJP (48 bits) instruction, so that if it is ever executed a "return" will be made back to the instruction following the BRTJ. This is why each subprogram coded so far has contained a BSS 1 instruction at the entry point - so that the subprogram can be used as a subroutine, either by the monitor or by another subprogram. More will be said when we talk about subprograms.

The BJSX instruction is very much like the BRTJ. The difference is that with the BJSX, the programmer can easily pass data to the subroutine. The data is stored immediately following the BJSX instruction. A sample of coding might look like the following:

```
        BJSX            SUB, 1
        DEC             58, -63, 5
          -
          -
          -
```

Here, three parameters are to be picked up by SUB. When the BJSX instruction is executed, the address of the BJSX instruction enters the designated index register automatically (in this case B^1). With this address the subroutine SUB can reference the three operands.

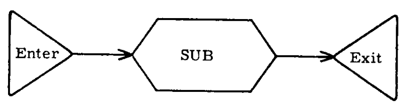
When the return is made, the subroutine must make sure that the data is bypassed when continuing the program.

Problem 10:

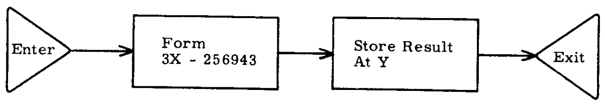
Show how the BJSX instruction can be used to form Y = 3X- 256943 in integer format.

Flowchart:

PASS



SUB



Problem 10 could be solved by coding in the following manner

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	PASS	
	ENTRY	PASS	
Y	BSS	1	
PASS	BSS	1	
	.		
	.		
	.		
READY	BJSX	SUB,1	
	DEC	3,256943	
	.		
	.		
SUB	BSS	1	
	LDA	X	
	MUI	1,1	ADDRESS READY +1
	SUB	2,1	ADDRESS READY +2
	STA	Y	
	INI	3,1	BYPASS DATA
	SLJ	SUB	
	END		

Somewhere within this subprogram would also be included the symbol X in the location field with a declaration of the prestored data or area reserved.

Student Problem 10A:

Show how subroutine SUB would evaluate Y = 405X+356432 if the constants are passed from the main routine.

Flowchart:

Problem 10A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 11

INTER-REGISTER

GROUP 11

INTER-REGISTER

1.	Register Operation	ROP
2.	Register Swap	RSW
3.	Register Transmit	RXT

This group of instructions transmits the contents of operational registers with some operation possibly taking place. No memory reference is made.

The first instruction specifies one of seven operations to take place between the contents of two registers, the result going to a third register.

The last two perform no operation, but simply position the contents of registers for maximum speed and efficiency.

The REGISTER OPERATION Instruction

The REGISTER OPERATION instruction is an instruction that performs one operation between two registers (source) and places the result in a third register (destination). No memory reference is made.

The contents of the source registers are left unchanged. The contents of the destination register are replaced by the result of the operation.

The operations allowable are:

1.	OR	Inclusive OR
2.	XOR	Exclusive OR
3.	AND	Logical Product
4.	IMP	Implication
5.	EQ	Equivalence
6.	+	Addition
7.	-	Subtraction

Some registers may not be used as a <u>destination</u> register. If they are used, a PASS instruction results. They are:

1. Interrupt Register
2. Instruction Bank Register
3. Shift Count Register
4. Miscellaneous Mode Register
5. P Register
6. Time Register (Clock)

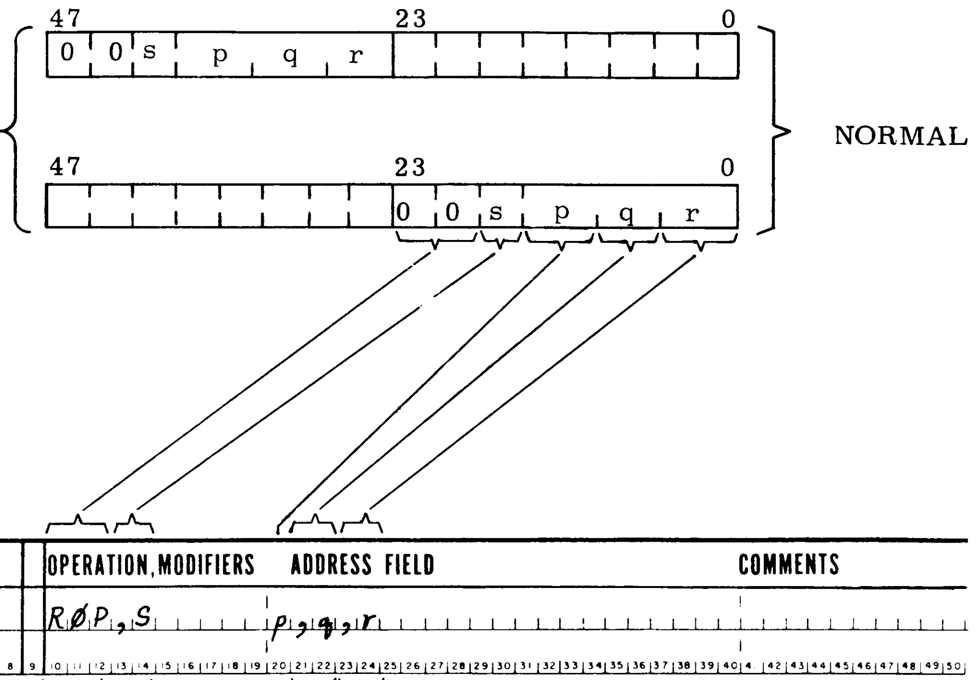
REGISTER OPERATION

ROP

FORMAT

MACHINE

May be an upper or lower instruction



Function code

Modifier, one of the following must be specified:

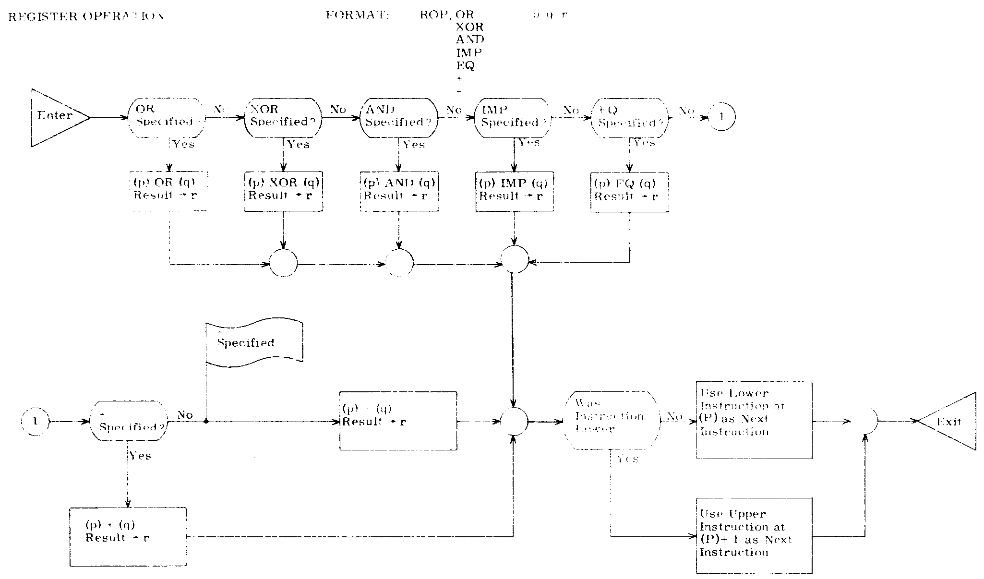
OR, inclusive OR
XOR, exclusive OR
AND, logical product
IMP, implication
EQ, equivalence
+, addition
-, subtraction

Destination register, one of 17 mnemonics.

Source register, one of 26 mnemonics.

Source register, one of 26 mnemonics.

FLOWCHART



DESCRIPTION

ROP, OR	(p) OR (q) → r
ROP, XOR	(p) XOR (q) → r
ROP, AND	(p) AND (q) → r
ROP, IMP	(p) IMP (q) → r
ROP, EQ	(p) EQ (q) → r
ROP, +	(p) + (q) → r
ROP, -	(p) - (q) → r

PROBLEM:

Add the contents of register A and Index Register 1 and transmit the result to A.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		RØP, +	A, B1, A	

PROBLEM:

Perform the "exclusive OR" between registers A and Q and transmit the result to D.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		RØP, XØR	A, Q, D	

PROBLEM:

Subtract one trom the D register and transmit the result to A.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		RØP, −	D, P1, A	

PROBLEM:

Perform the logical product between the contents of the A and Q registers and transmit the result to D.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		RØP, AND	A, Q, D	

The REGISTER SWAP Instruction

The REGISTER SWAP instruction is an instruction that performs an exchange of contents between two registers. No memory reference is made.

IF CQ is specified and the q portion of the format is either AL, AU, QL, or QU, the complete register is cleared as the contents are being exchanged.

If CR is specified and the r portion of the format is either AL, AU, QL, or QU, the complete register is cleared as the contents are being exchanged.

Some registers may not be used in the swap. If they are used, they will not be changed. They are:

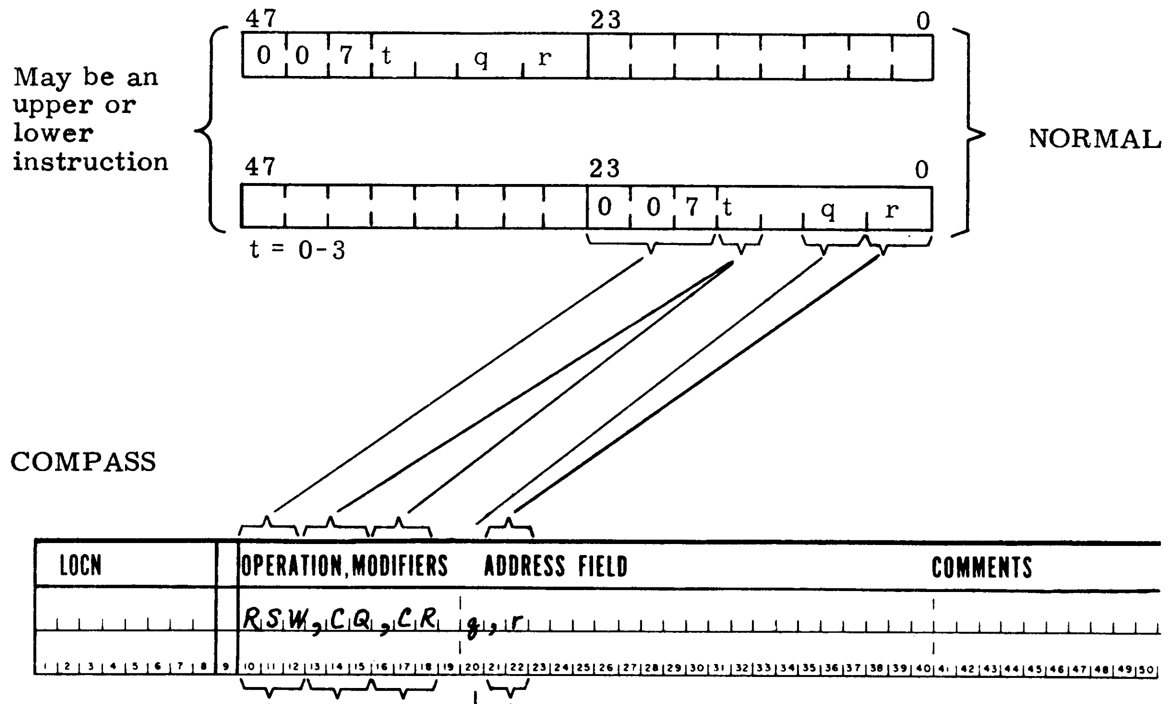
1. Interrupt Register
2. Instruction Bank Register
3. Shift Count Register
4. Miscellaneous Mode Register
5. P Register
6. Time Register (Clock)

REGISTER SWAP | RSW

FORMAT

MACHINE

May be an upper or lower instruction



Function code

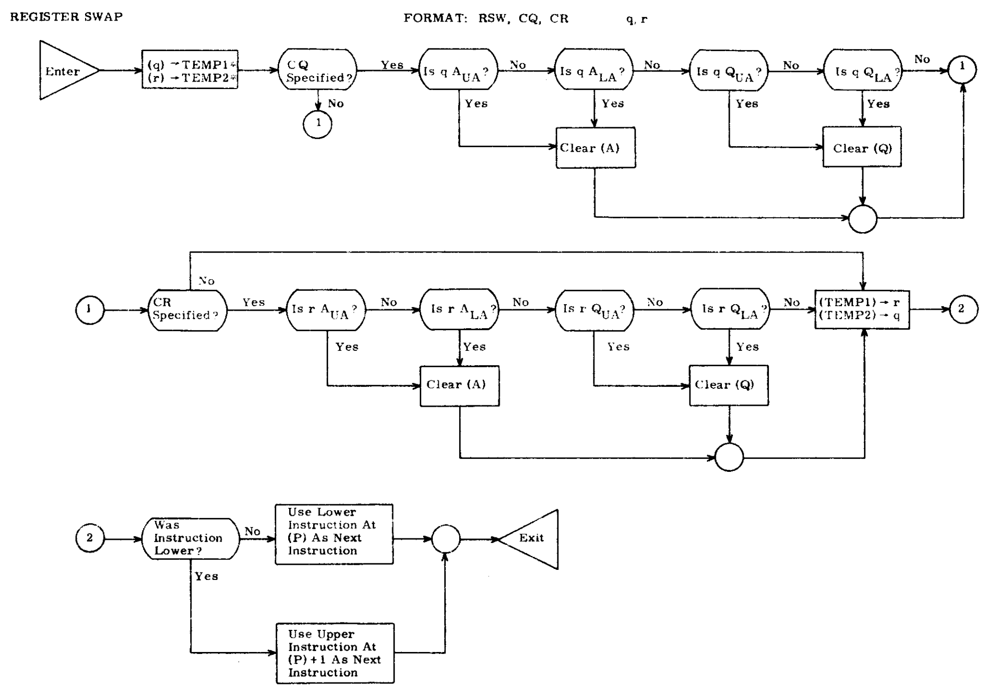
Optional modifier CQ, clear the complete register when q specifies: AU, AL, QU or QL

Optional modifier CR, clear the complete register when r specifies: AU, AL, QU or QL

Mnemonic for second register, one of 17 mnemonics.

Mnemonic for first register, one of 17 mnemonics.

FLOWCHART



*Non-program addressable

DESCRIPTION

RSW	Swap (q) and (r)
RSW, CQ	1) Swap (q) and (r) 2) Clear rest of register that q represents
RSW, CR	1) Swap (q) and (r) 2) Clear rest of register that r represents

The REGISTER TRANSMIT Instruction

The REGISTER TRANSMIT instruction is an instruction that transmits the contents of one register (source) to another (destination). No memory reference is made.

If CQ is specified, the contents of the register specified by the q portion of the format are cleared <u>after</u> the transmission takes place.

If CR is specified <u>and</u> the r portion of the format is either AL, AU, QL, or QU, the complete register is cleared <u>before</u> the transmission takes place.

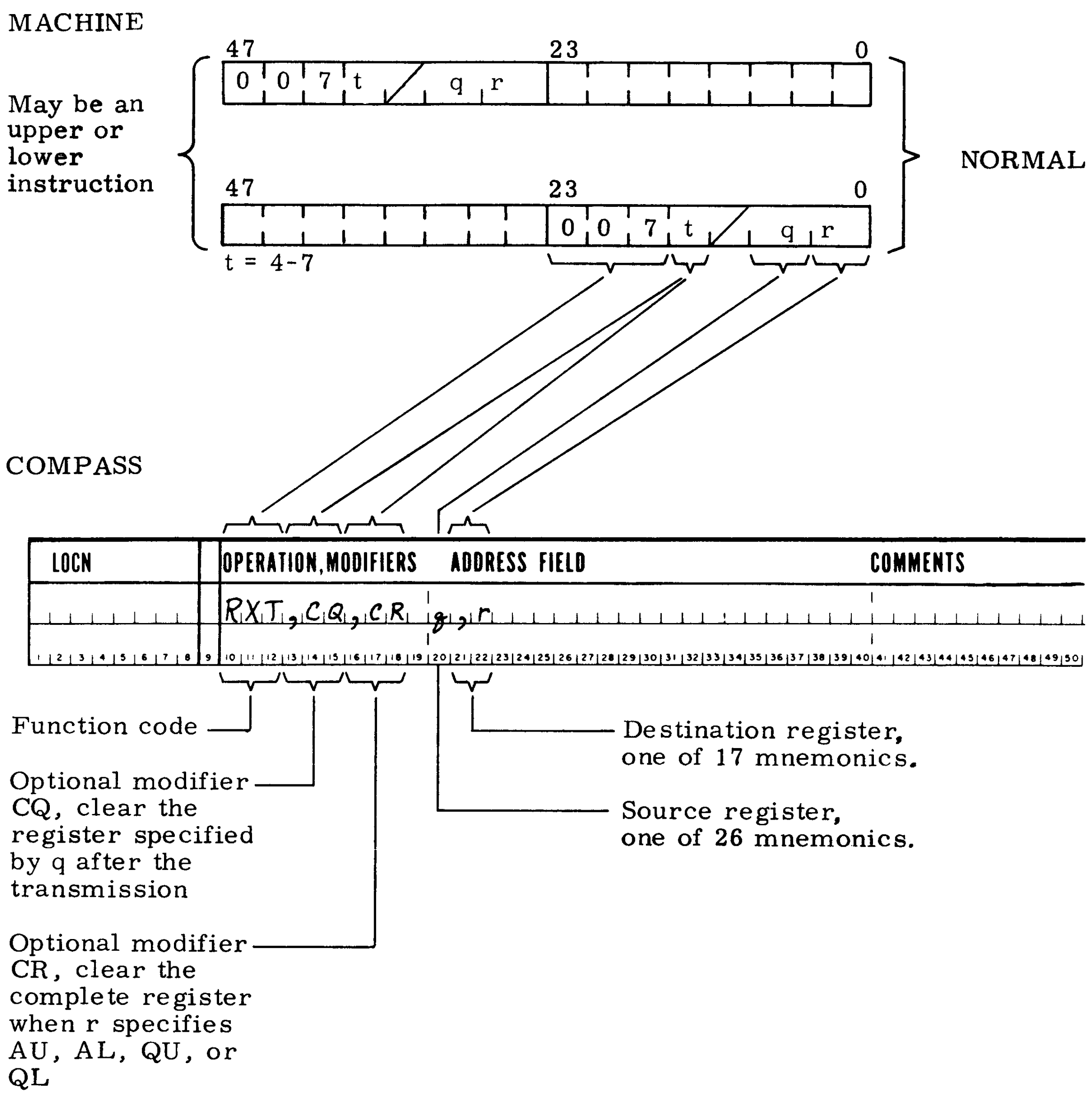
If an attempt is made to alter the contents of the following registers, no action will take place.

1. Interrupt Register
2. Instruction Bank Register
3. Shift Count Register
4. Miscellaneous Mode Register
5. P Register
6. Time Register (Clock)

REGISTER TRANSMIT

RXT

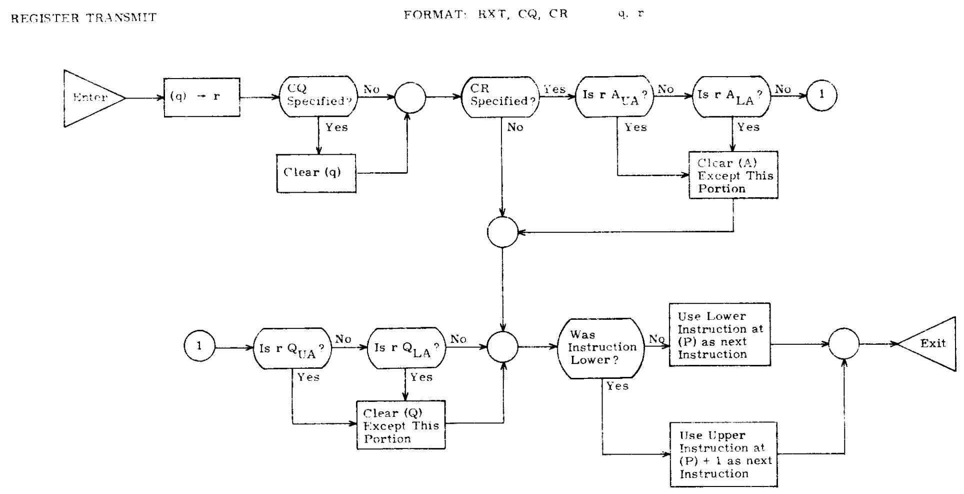
FORMAT



MACHINE
May be an upper or lower instruction
47
23
0
0 0 7 t q r
0 0 7 t q r
t = 4-7
NORMAL
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
RXT, CQ, CR q, r
Function code
Optional modifier CQ, clear the register specified by q after the transmission
Optional modifier CR, clear the complete register when r specifies AU, AL, QU, or QL
Destination register, one of 17 mnemonics.
Source register, one of 26 mnemonics.

RXT

FLOWCHART



DESCRIPTION

RXT	1) (q) → r
	2) Contents of q remain unchanged
RXT, CQ	1) (q) → r
	2) Clear contents of q after transmission
RXT, CR	1) (q) → r
	2) Clear rest of register that r represents

PROBLEM:

Transmit Index Register 1 to Q.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	RXT	B1, Q	

PROBLEM:

Transmit A to D.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	RXT	A, D	

PROBLEM:

Transmit Index Register 6 to A, then clear Index Register 6.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	RXT, CQ	B6, A	

PROBLEM:

Transmit A upper to A lower, then clear the rest of A except A lower.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	RXT, CR	AU, AL	

NEW CONCEPTS OF GROUP 11

Programmers wishing top speed out of their programs would do well to consider the instructions in this group. They are the fastest, most versatile instructions in the repertoire. None of them require a memory reference.

Any operational register can be used. This includes A, Q, D, and the index registers. There is a Compass mnemonic for each.

These instructions can be used to maneuver data through the registers once the data has been read out of memory. Operations can be performed on the data with the ROP instruction.

There are two types of operations: Arithmetic and Logical. Arithmetic operations include addition and subtraction. This is done using 1's complement mode in fixed point integer format. Logical operations are bit by bit comparisons. Here is a table showing how the bits in two given registers will yield results in the third register.

Logical Operations

First Reg. p	Second Reg. q	p and q	INC. p or q	EXC. p or q	p IMP. q	p EQUIV q
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	0
1	1	1	1	0	1	1

The last two instructions, RSW and RXT, perform no operation but can be used to interchange data or transmit data from one register to another.

Let's show a couple of examples showing the effectiveness of these instructions.

1. Question: A programmer wishes to save the contents of A temporarily. What is the fastest way to do it?

Answer: RXT A, D

This is a transmission of A to D.

2. Question: A programmer wishes to complement the contents of A. What is the fastest way to do it?

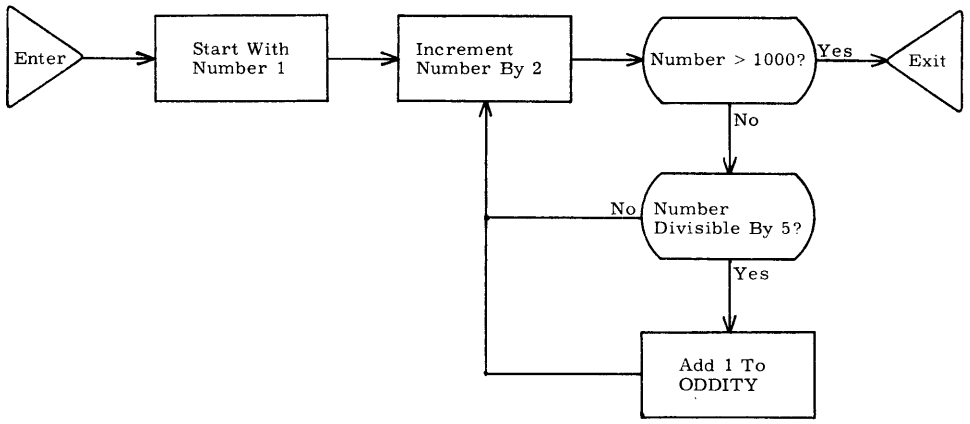
Answer: RØP, XØR A, MƵ, A

This is the "Exclusive ØR" with binary 1's.

Problem 11:

Write a program that will count the number of times that the odd numbers between 1 and 100 are divisible by 5. Store the amount that you find at address ODDITY.

Flowchart:



Problem 11 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	ØDDNUM	
	ENTRY	ØDDNUM	
ØDDITY	DEC	0	
FIVE	DEC	5	
ØDDNUM	BSS	1	
	ENA	1	
	RXT	A,D	
	ENI	2,1	
NEXT	RØP,+	D,B1,A	NEXT ØDD
	RGJP,GT	A,1000,ØDDNUM	
	ENQ	0	
	DVI	FIVE	
	QJP,NZ	NEXT	
	RAØ	ØDDITY	
	SLJ	NEXT	
	END		

Student Problem 11A:

A sequence of terms looks like the following:

S = 1, 3, 7, 13, 21, 31, 43, ...

Of the first 100 terms, collect all terms divisible by 7 and store them in a table starting at address SET7.

Flowchart:

Problem 11A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

PROBLEM:

Swap the contents of Index Registers 2 and 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RSW	B2, B3	

PROBLEM:

Swap the contents of the A and D registers.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RSW	A, D	

PROBLEM:

Swap the contents of A lower and Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RSW	AL, B1	

PROBLEM:

Swap the contents of A lower and Index Register 2, clearing the rest of A.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RSW, CQ	AL, B2	

GROUP 12

SHIFTING OPERATIONS

GROUP 12

SHIFTING OPERATIONS

1.	A Right Shift	ARS
2.	A Left Shift	ALS
3.	Q Right Shift	QRS
4.	Q Left Shift	QLS
5.	Long Right Shift	LRS
6.	Long Left Shift	LLS
7.	Scale A	SCA
8.	Scale AQ	SCQ

This group of instructions shifts the contents of A, Q, or AQ to the right or left in order to most effectively position data. No memory reference is required.

The first six instructions are normal shifts that can be augmented as "end-off", or whose direction can be reversed under certain conditions.

The last two instructions are used to scale quantities in the registers. The scaling of the quantity, along with the retaining of the scale factor, allows the programmer to solve highly complex problems of data manipulation with speed and accuracy.

The A RIGHT SHIFT Instruction

The A RIGHT SHIFT instruction is an instruction that shifts the contents of A, end-off, with sign extended (assuming there are no modifiers). The number of binary positions shifted is specified by <u>K</u> where $K = k + (B^b) + (V^v)$.

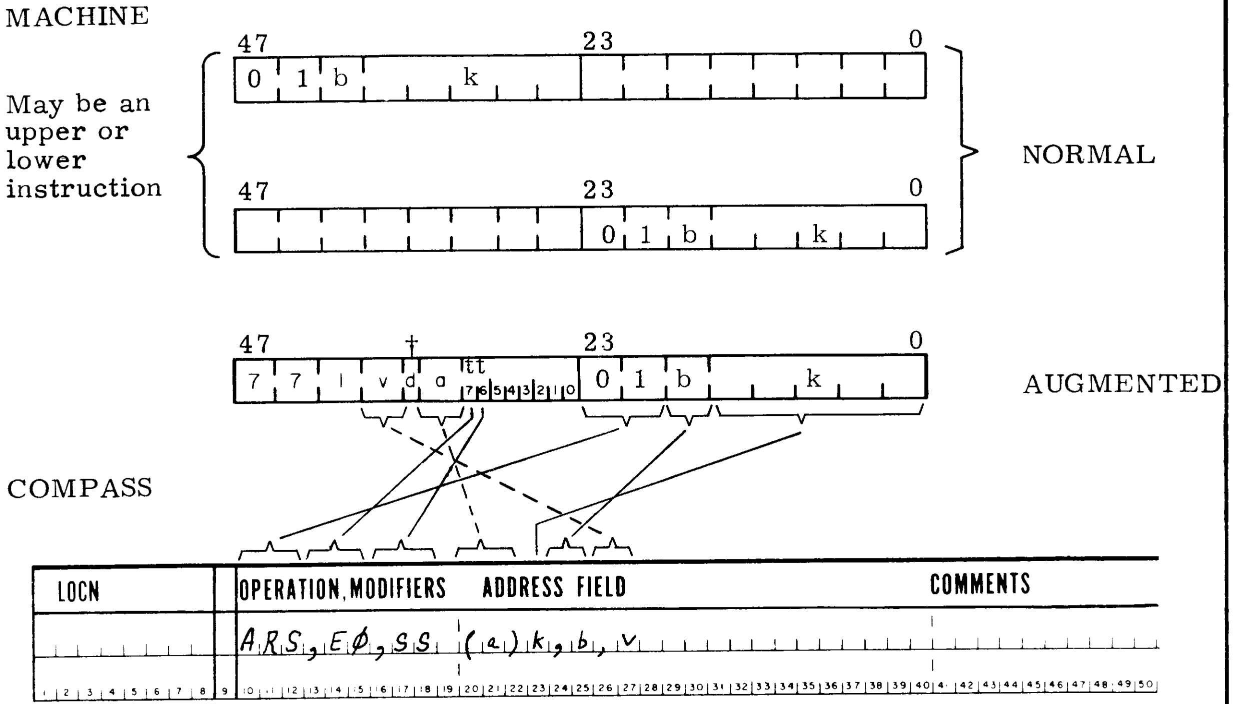
The maximum number of shifts for K is 48. If more than 48 are attempted, the shift is blocked and a Shift Fault occurs.

If EO is specified by the programmer, the shift is end-off with zeros extended.

If SS is specified by the programmer, the direction of the shift will change if $K < 0$. For example, an A RIGHT SHIFT where $K < 0$ will actually shift left, end-around. The number of positions shifted would be the absolute value of K. This is all checked and done in the hardware.

A RIGHT SHIFT | ARS

FORMAT



Function code

Optional, end-off with zeros extended

Optional, switch direction of shift if K < 0

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base shift.
Zero if omitted.
*, current program address

Optional, bank term.

0-7, transmits number to Operand Bank Register

*, transmits Instruction Bank Register to Operand Bank Register.

[†] d bit not programmable

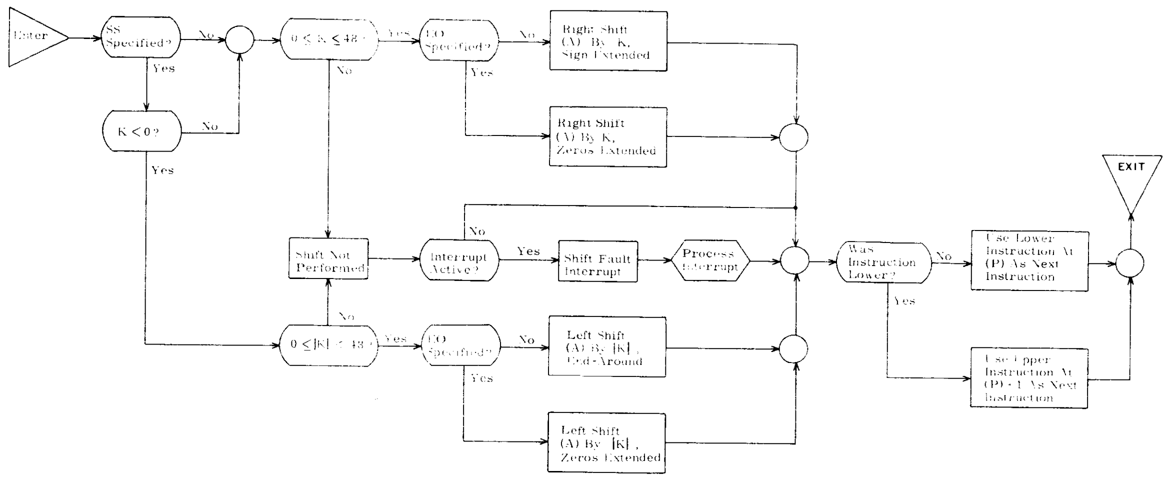
ARS

FLOWCHART

A RIGHT SHIFT

FORMAT: ARS, EO, SS (a) k,b,v

$K = k + (B^b) + (V^v)$



Enter
SS Specified?
No
Yes
K < 0?
No
Yes
0 ≤ K ≤ 48?
Yes
No
EO Specified?
No
Yes
Right Shift (A) By K, Sign Extended
Right Shift (A) By K, Zeros Extended
Shift Not Performed
Interrupt Active?
No
Yes
Shift Fault Interrupt
Process Interrupt
0 ≤ |K| ≤ 48?
No
Yes
EO Specified?
No
Yes
Left Shift (A) By |K|, End-Around
Left Shift (A) By |K|, Zeros Extended
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
EXIT

DESCRIPTION

ARS	Right shift (A) end-off with sign extended
ARS, EO	Right shift (A) end-off with zeros extended
ARS, SS	Normal right shift if $K \geq 0$ Left shift end-around if $K < 0$

PROBLEM:

Right shift the A register 15 places.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ARS	15	

PROBLEM:

Right shift the A register by the number of places specified in Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ARS	,1	

PROBLEM:

Right shift the A register six places, end-off.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ARS,EØ	6	

PROBLEM:

Perform a right shift of six places if Index Register 5 contains 6.
Perform a left shift of six places if Index Register 5 contains -6.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ARS	,5	

The A LEFT SHIFT Instruction

The A LEFT SHIFT instruction is an instruction that shifts the contents of A, end-around (assuming there are no modifiers). The number of binary positions shifted is specified by $\underline{K}$ where $K = k + (B^b) + (V^v)$.

The maximum number of shifts for K is 48. If more than 48 are attempted, the shift is blocked and a Shift Fault occurs.

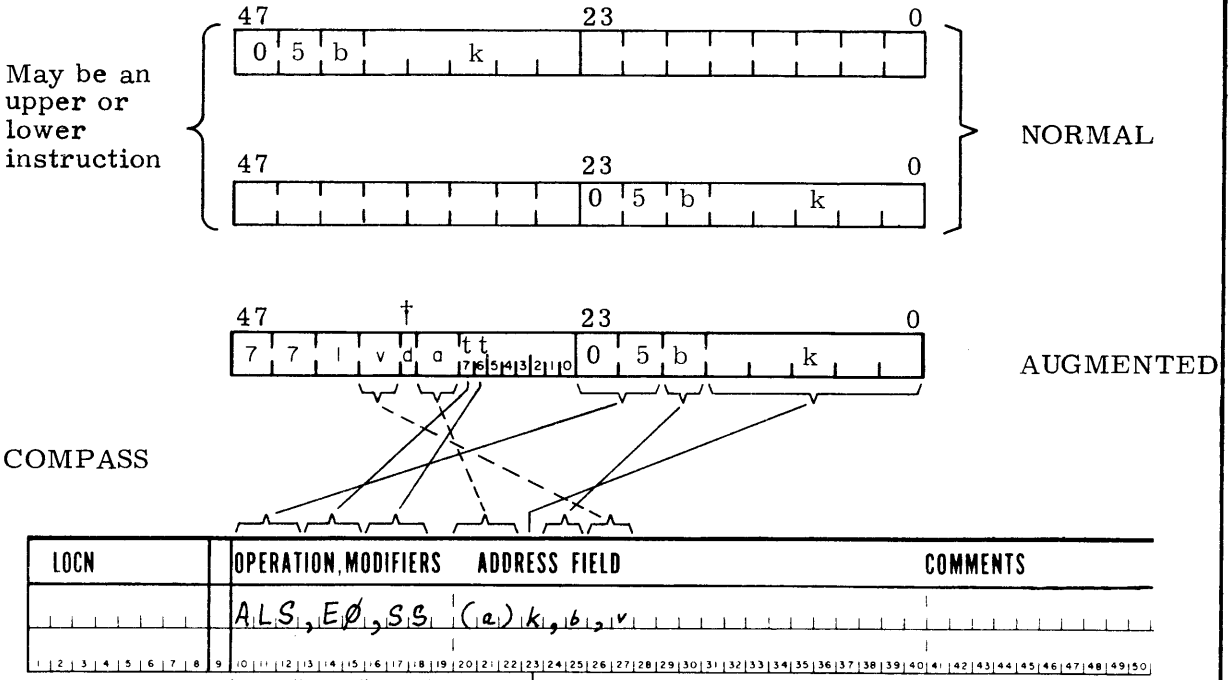
If EO is specified by the programmer, the shift is end-off with zeros extended.

If SS is specified by the programmer, the direction of the shift will change if $K < 0$. For example, an A LEFT SHIFT where $K < 0$ will actually shift right, end-off. The number of positions shifted would be the absolute value of K. This is all checked and done in the hardware.

FORMAT

MACHINE

May be an upper or lower instruction



Function code

Optional, end-off with zeros extended

Optional, switch direction of shift if K < 0

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base shift. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

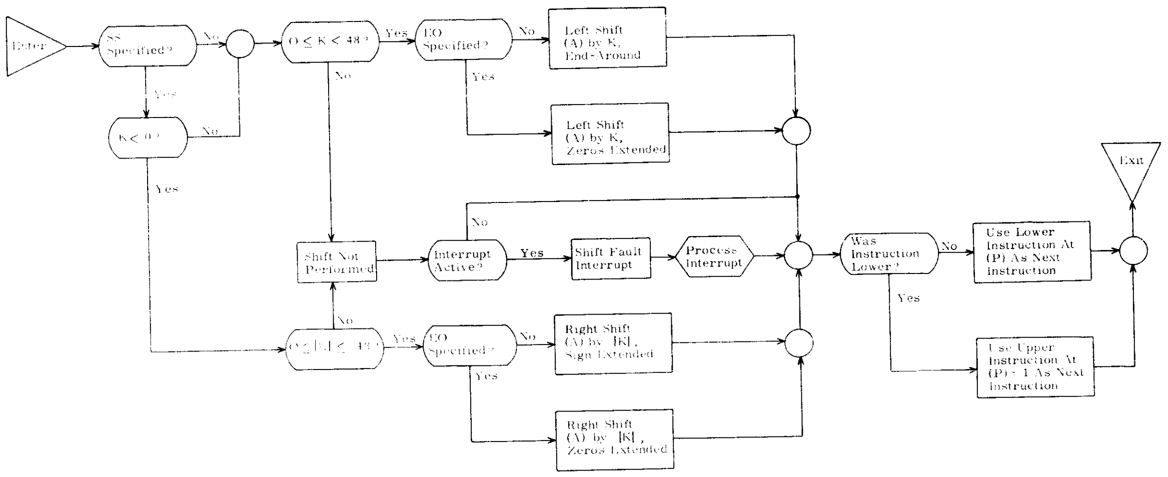
†d bit not programmable

FLOWCHART

A LEFT SHIFT

FORMAT: ALS, EO, SS (a) k b, v

$K = k + (B^b) + (V^v)$


Enter
SS Specified?
No
Yes
K < 0?
No
Yes
0 ≤ K < 48?
Yes
No
EO Specified?
No
Yes
Left Shift (A) by K, End-Around
Left Shift (A) by K, Zeros Extended
Shift Not Performed
Interrupt Active?
Yes
No
Shift Fault Interrupt
Process Interrupt
0 ≤ |K| ≤ 48?
Yes
No
EO Specified?
No
Yes
Right Shift (A) by |K|, Sign Extended
Right Shift (A) by |K|, Zeros Extended
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

DESCRIPTION

ALS	Left shift (A) end-around
ALS, EO	Left shift (A) end-off with zeros extended
ALS, SS	Normal left shift if $K \geq 0$
	Right shift end-off with sign extended if $K < 0$

PROBLEM:

Left shift the A register by 15 places.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ALS	15	

PROBLEM:

Left shift the A register by the number of places specified in Index Register 2.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ALS	,2	

PROBLEM:

Left shift the A register six places, end-off.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ALS,EØ	6	

PROBLEM: Given: (B^1) = a positive or negative number.

If $(B^1) \geq 0$, left shift A by the value in Index Register 1.
If $(B^1) < 0$, right shift A by the absolute value in Index Register 1.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ALS,SS	,1	

The Q RIGHT SHIFT Instruction

The Q RIGHT SHIFT instruction is an instruction that shifts the contents of Q, end-off, with sign extended (assuming there are no modifiers). The number of binary positions shifted is specified by $\underline{K}$ where $K = k + (B^b) + (V^v)$.

The maximum number of shifts for K is 48. If more than 48 are attempted, the shift is blocked and a Shift Fault occurs.

If EO is specified by the programmer, the shift is end-off with zeros extended.

If SS is specified by the programmer, the direction of the shift will change if $K < 0$. For example, a Q RIGHT SHIFT where $K < 0$ will actually shift left, end-around. The number of positions shifted would be the absolute value of K. This is all checked and done in the hardware.

FORMAT

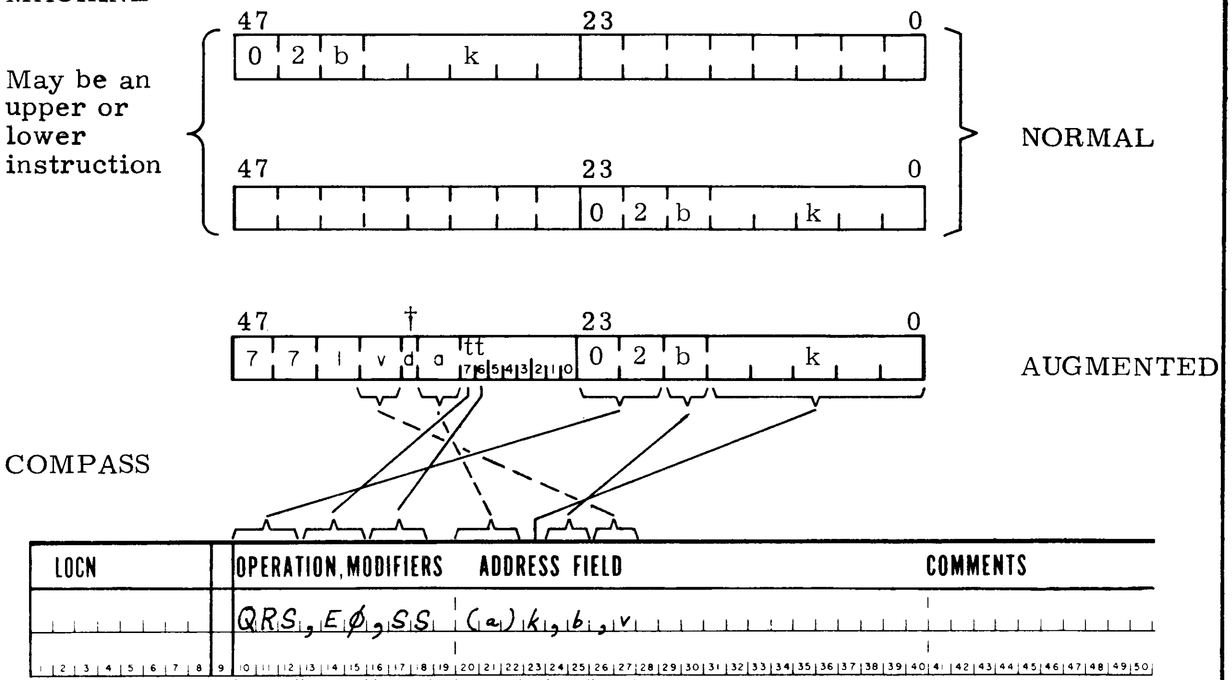
MACHINE

May be an upper or lower instruction

NORMAL

AUGMENTED

COMPASS



Function code

Optional, end-off with zeros extended

Optional, switch direction of shift if K < 0

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base shift. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

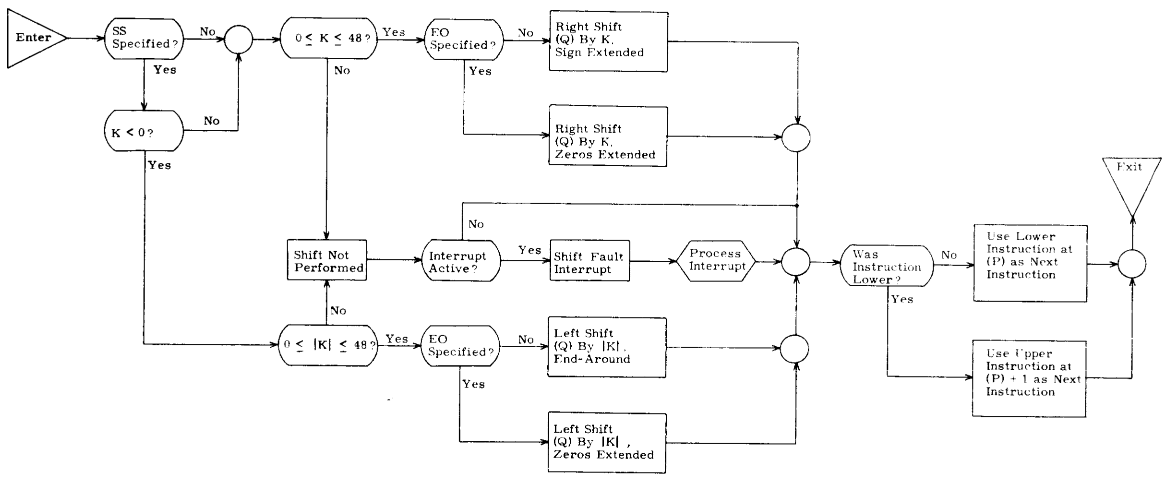
†d bit not programmable

FLOWCHART

Q RIGHT SHIFT

FORMAT: QRS, EO, SS (a) k, b, v

$K = k + (B^b) + (V^v)$


Enter
SS Specified?
No
Yes
K < 0?
No
Yes
0 ≤ K ≤ 48?
Yes
No
EO Specified?
No
Yes
Right Shift (Q) By K, Sign Extended
Right Shift (Q) By K, Zeros Extended
Shift Not Performed
Interrupt Active?
Yes
No
Shift Fault Interrupt
Process Interrupt
0 ≤ |K| ≤ 48?
Yes
No
EO Specified?
No
Yes
Left Shift (Q) By |K|, End-Around
Left Shift (Q) By |K|, Zeros Extended
Was Instruction Lower?
No
Yes
Use Lower Instruction at (P) as Next Instruction
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

QRS	Right shift (Q) end-off with sign extended
QRS, EO	Right shift (Q) end-off with zeros extended
QRS, SS	Normal right shift if $K \geq 0$ Left shift end-around if $K < 0$

EXAMPLES

PROBLEM:
Right shift the Q register by 15 places.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	QRS	15	

PROBLEM:
Right shift the Q register by the number of places specified in Index Register 6.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	QRS	,6	

PROBLEM:
Right shift the Q register by six places, end-off.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	QRS,EØ	6	

PROBLEM: Given: (B^3) = -38 during one pass of a loop.
Left shift Q by the absolute value of this quantity.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	QRS,SS	,3	

The Q LEFT SHIFT Instruction

The Q LEFT SHIFT instruction is an instruction that shifts the contents of Q, end-around (assuming there are no modifiers). The number of binary positions shifted is specified by K where $K = k + (B^b) + (V^v)$.

The maximum number of shifts for K is 48. If more than 48 are attempted, the shift is blocked and a Shift Fault occurs.

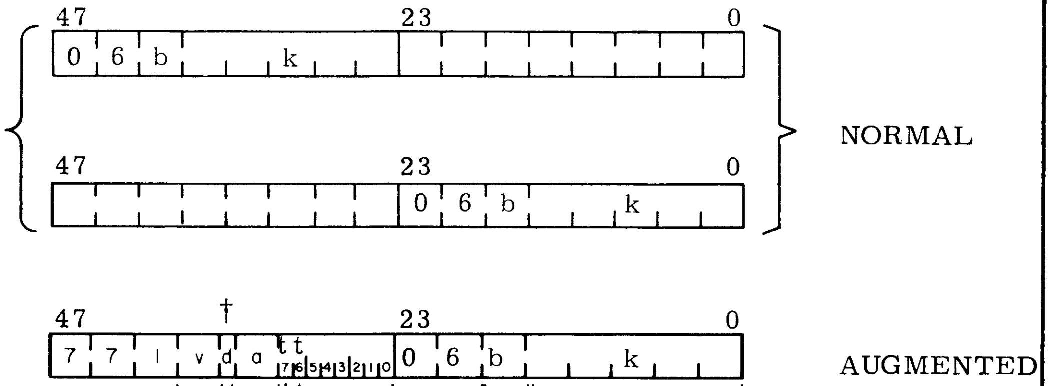
If EO is specified by the programmer, the shift is end-off with zeros extended.

If SS is specified by the programmer, the direction of the shift will change if $K < 0$. For example, a Q LEFT SHIFT where $K < 0$ will actually shift right, end-off. The number of positions shifted would be the absolute value of K. This is all checked and done in the hardware.

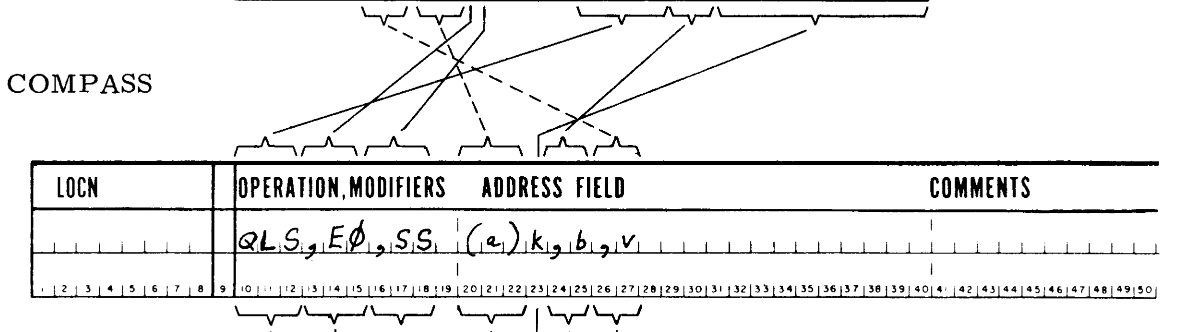
FORMAT

MACHINE

May be an upper or lower instruction



COMPASS



Function code

Optional, end-off with zeros extended

Optional, switch direction of shift if $K < 0$

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base shift. Zero if omitted. *, current program address.

Optional, bank term.

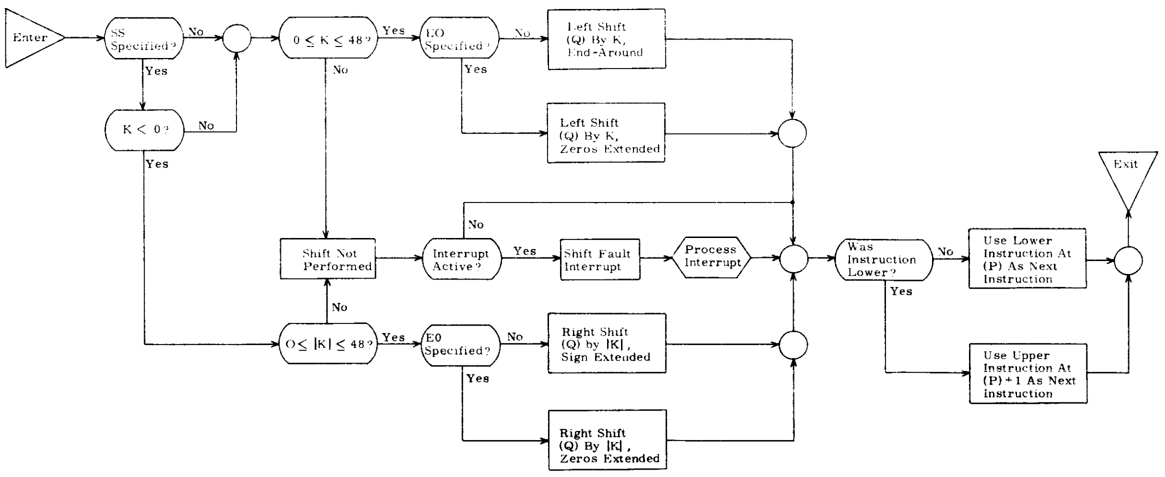
0-7, transmits number to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

FLOWCHART

Q LEFT SHIFT   FORMAT: QLS, EO, SS   ( )k, b, v $K = k + (B^b) + (V^v)$


Enter
SS Specified?
No
Yes
K < 0?
No
Yes
0 ≤ K ≤ 48?
Yes
No
EO Specified?
No
Yes
Left Shift (Q) By K, End-Around
Left Shift (Q) By K, Zeros Extended
Shift Not Performed
Interrupt Active?
No
Yes
Shift Fault Interrupt
Process Interrupt
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit
0 ≤ |K| ≤ 48?
Yes
No
EO Specified?
No
Yes
Right Shift (Q) by |K|, Sign Extended
Right Shift (Q) By |K|, Zeros Extended

DESCRIPTION

QLS	Left shift (Q) end-around
QLS, EO	Left shift (Q) end-off with zeros extended
QLS, SS	Normal left shift if $K \geq 0$
	Right shift end-off with sign extended if $K < 0$

PROBLEM:
Left shift the Q register by 15 places.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	QLS	15	

PROBLEM:
Left shift the Q register by the number of places specified in Index Registers 2 and 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	QLS	,2,3	

PROBLEM:
Left shift the Q register by six places, end-off.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	QLS,EØ	6	

PROBLEM:
Shift the Q register according to the contents of Index Register 2.
If $(B^2) \geq 0$, left shift Q by the value in Index Register 2.
If $(B^2) < 0$, right shift Q by the absolute value in Index Register 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	QLS,SS	,2	

The LONG RIGHT SHIFT Instruction

The LONG RIGHT SHIFT instruction is an instruction that shifts the contents of AQ, end-off, with sign extended (assuming there are no modifiers). The number of binary positions shifted is specified by <u>K</u> where $K = k + (B^b) + (V^v)$.

The maximum number of shifts for K is 96. If more than 96 are attempted, the shift is blocked and a Shift Fault occurs.

If EO is specified by the programmer, the shift is end-off with zeros extended.

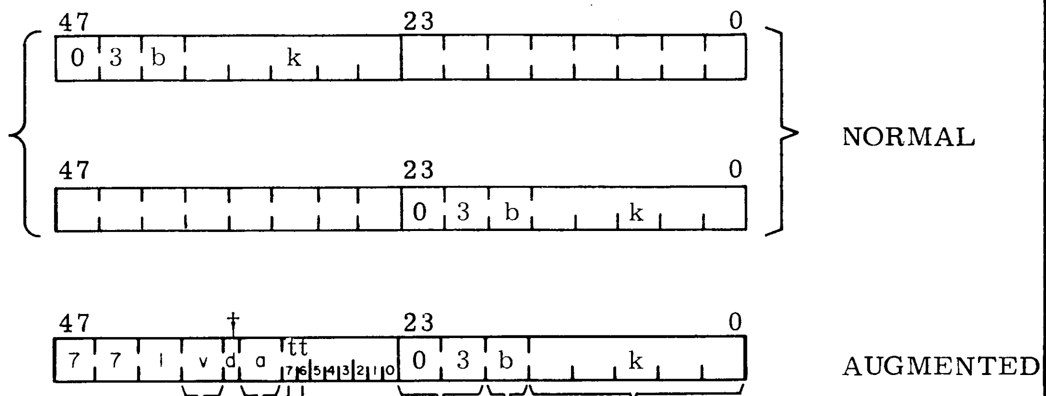
If SS is specified by the programmer, the direction of the shift will change if $K < 0$. For example, a LONG RIGHT SHIFT where $K < 0$ will actually shift left, end-around. The number of positions shifted would be the absolute value of K. This is all checked and done in the hardware.

LONG RIGHT SHIFT | LRS

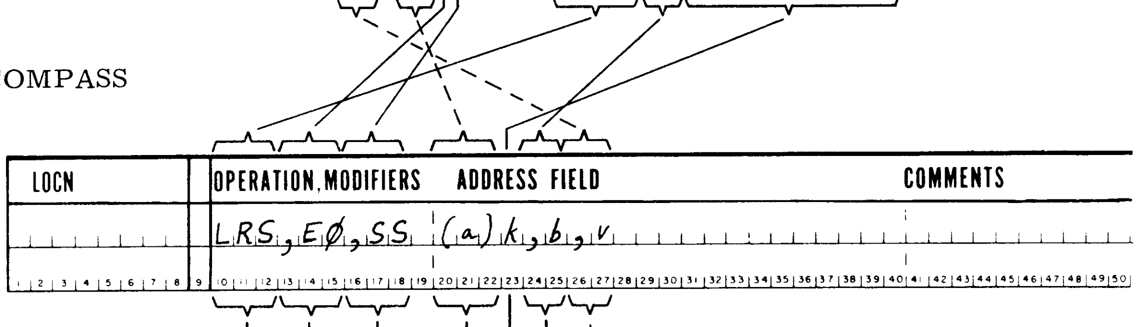
FORMAT

MACHINE

May be an upper or lower instruction



COMPASS



Function code

Optional, end-off with zeros extended

Optional, switch direction of shift if K < 0

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base shift.
Zero if omitted.
*, current program address.

Optional, bank term.

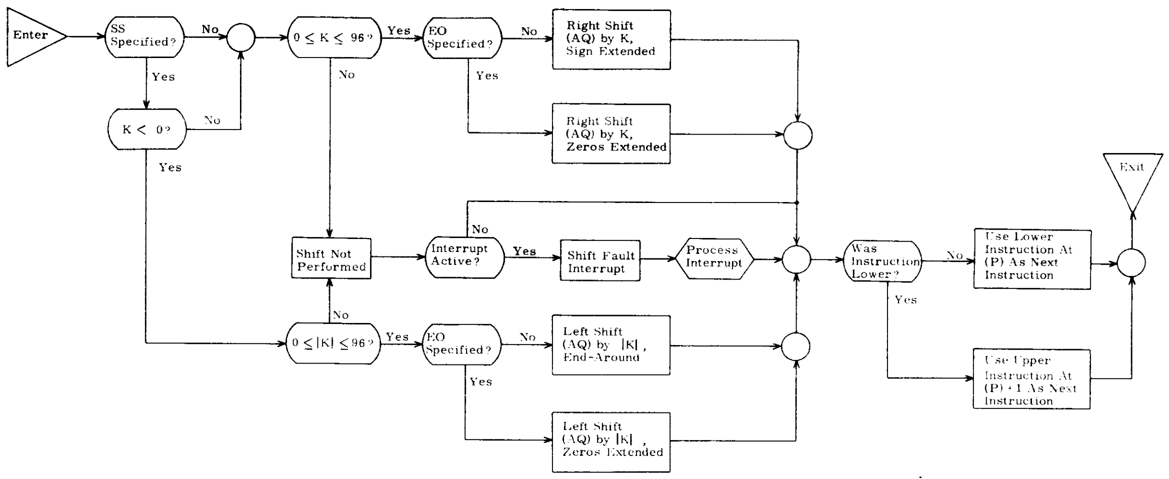
0-7, transmits number to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

FLOWCHART

LONG RIGHT SHIFT FORMAT: LRS, EO, SS (a) k b, v $K = k + (B^b) + (V^v)$


Enter
SS Specified?
No
Yes
K < 0?
No
Yes
0 ≤ K ≤ 96?
Yes
No
EO Specified?
No
Yes
Right Shift (AQ) by K, Sign Extended
Right Shift (AQ) by K, Zeros Extended
Shift Not Performed
Interrupt Active?
No
Yes
Shift Fault Interrupt
Process Interrupt
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit
0 ≤ |K| ≤ 96?
No
Yes
EO Specified?
No
Yes
Left Shift (AQ) by |K|, End-Around
Left Shift (AQ) by |K|, Zeros Extended

DESCRIPTION

LRS	Right shift (AQ) end-off with sign extended
LRS, EO	Right shift (AQ) end-off with zeros extended
LRS, SS	Normal right shift if $K \geq 0$
	Left shift end-around if $K < 0$

PROBLEM:

Right shift AQ by 38 places.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LRS	38	

PROBLEM:

Right shift AQ by the number of places specified in Index Register 4.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LRS	,4	

PROBLEM:

Right shift AQ by 24 places, end-off.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LRS,E∅	24	

PROBLEM:

Shift AQ according to the contents of Index Registers 3 and 4.
If $(B^3) + (B^4) \geq 0$, right shift AQ by the sum of their values.
If $(B^3) + (B^4) < 0$, left shift AQ by the absolute value of the sum of their values.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LRS,SS	,3,4	

The LONG LEFT SHIFT Instruction

The LONG LEFT SHIFT instruction is an instruction that shifts the contents of AQ, end-around (assuming there are no modifiers). The number of binary positions shifted is specified by $\underline{K}$ where $K = k + (B^b) + (V^v)$.

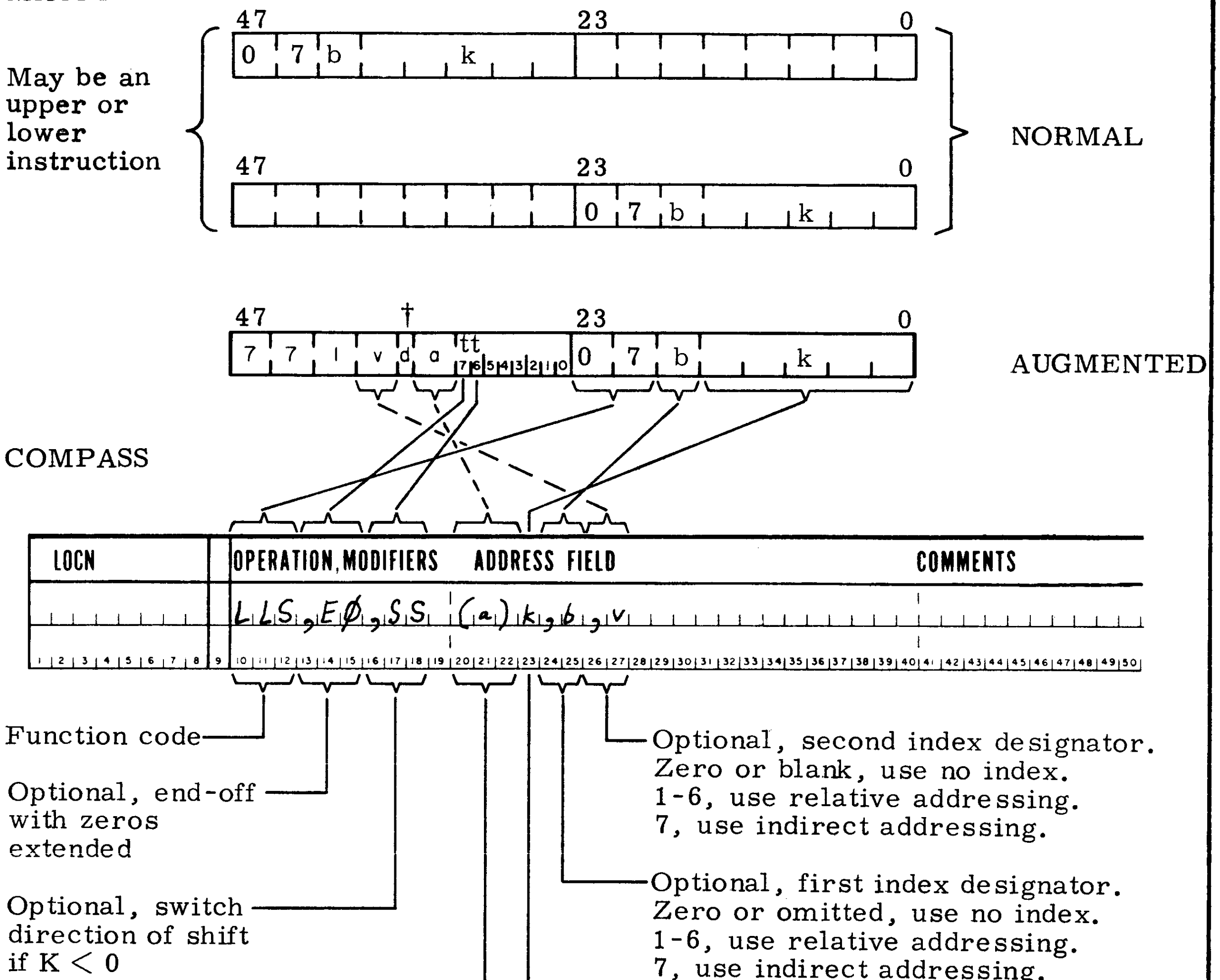
The maximum number of shifts for K is 96. If more than 96 are attempted, the shift is blocked and a Shift Fault occurs.

If EO is specified by the programmer, the shift is end-off with zeros extended.

If SS is specified by the programmer, the direction of the shift will change if $K < 0$. For example, a LONG LEFT SHIFT where $K < 0$ will actually shift right, end-off. The number of positions shifted would be the absolute value of K. This is all checked and done in the hardware.

FORMAT

MACHINE



Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base shift.
Zero if omitted.
*, current program address.

Optional, bank term.

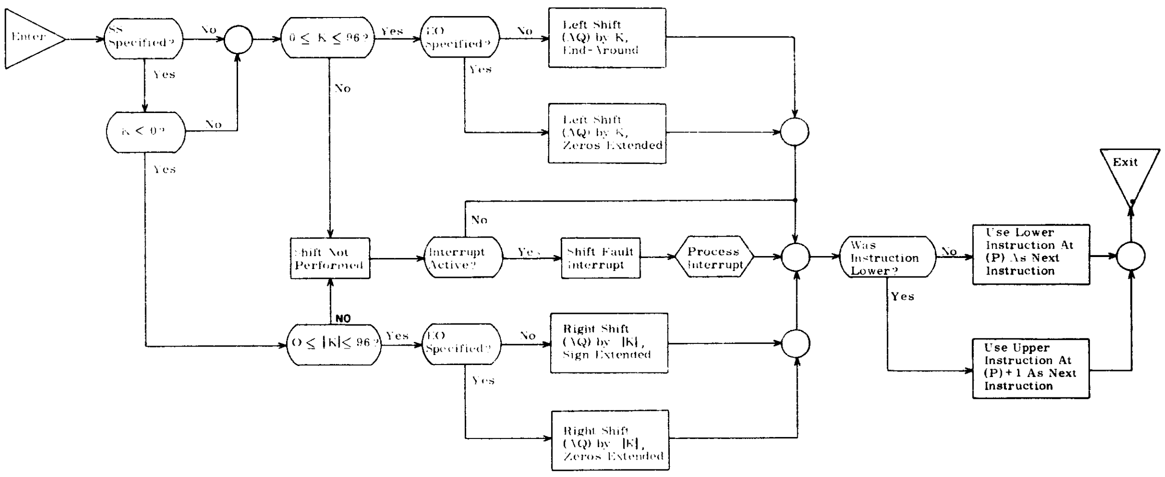
0-7, transmits number to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

F L O W C H A R T

LONG LEFT SHIFT   FORMAT: LLS, EO, SS   (a) k, b, v $K = k + (B^b) + (V^v)$



Enter
SS Specified?
No
Yes
K < 0?
No
Yes
0 ≤ K ≤ 96?
Yes
No
EO Specified?
No
Yes
Left Shift (AQ) by K, End-Around
Left Shift (AQ) by K, Zeros Extended
Shift Not Performed
Interrupt Active?
No
Yes
Shift Fault Interrupt
Process Interrupt
0 ≤ |K| ≤ 96?
NO
Yes
EO Specified?
No
Yes
Right Shift (AQ) by |K|, Sign Extended
Right Shift (AQ) by |K|, Zeros Extended
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

D E S C R I P T I O N

LLS	Left shift (AQ) end-around
LLS, EO	Left shift (AQ) end-off with zeros extended
LLS, SS	Normal left shift if $K \geq 0$
	Right shift end-off with sign extended if $K < 0$

PROBLEM:

Left shift AQ by 38 places.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LLS	38	

PROBLEM:

Left shift AQ by the value specified in Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LLS	,1	

PROBLEM:

Left shift AQ 93 places, end-off.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LLS,EØ	93	

PROBLEM:

Shift AQ according to the contents of Index Register 5.
If $(B^5) \geq 0$, left shift AQ by the value in Index Register 5.
If $(B^5) < 0$, right shift AQ by the absolute value in Index Register 5.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LLS,SS	,5	

The SCALE A Instruction

The SCALE A instruction is an instruction that scales an operand in the A register. The scale factor, K minus the number of left shifts needed to scale, replaces the contents of the designated index register specified by b.

If the contents of A are originally zero, scaling is not performed and K replaces the contents of the index register.

If the operand is already scaled, K replaces the contents of the index register.

If K is less than the number of left shifts needed to scale, only K number of left shifts will result. The operand will not be scaled and zeros will replace the contents of the index register.

The bank term a determines the value of the operand bank setting. If it is not used, the current operand setting will be replaced by the value a. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

SCALE A

SCA

F
O
R
M
A
T

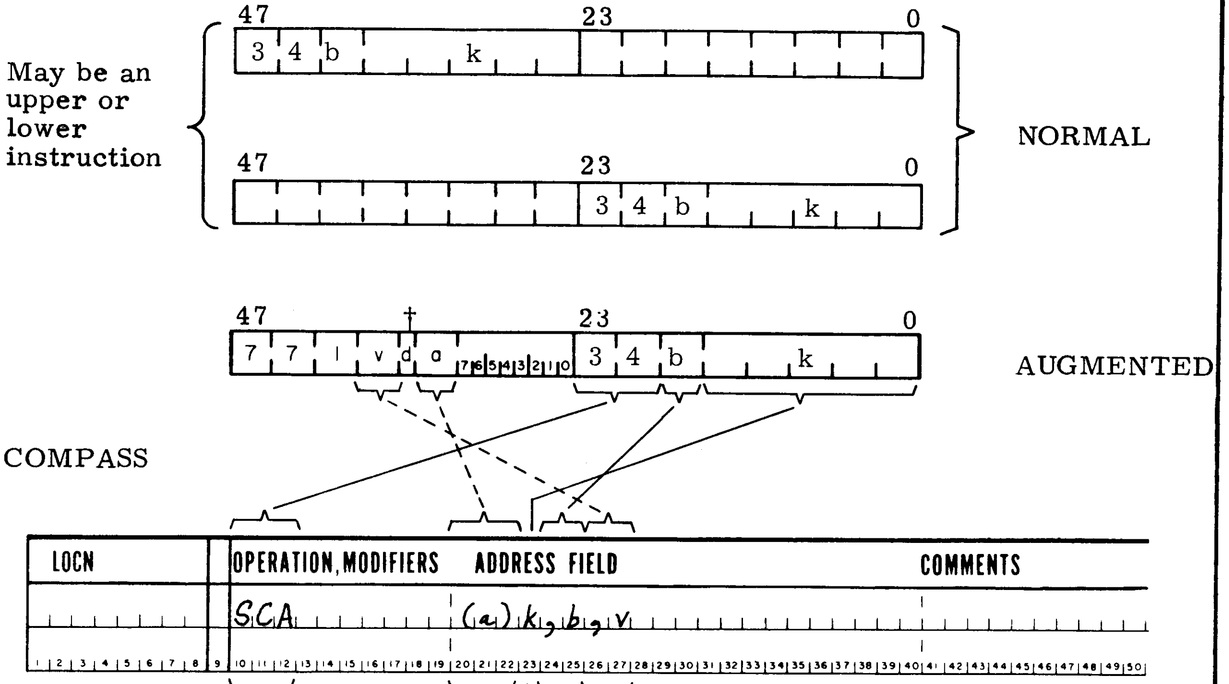
MACHINE

May be an upper or lower instruction

NORMAL

AUGMENTED

COMPASS



Function code

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.

Zero or omitted, use no index.

1-6, use register as destination for scale factor.

7, use indirect addressing.

Optional, base shift.
Zero if omitted.
*, current program address.

Optional, bank term.

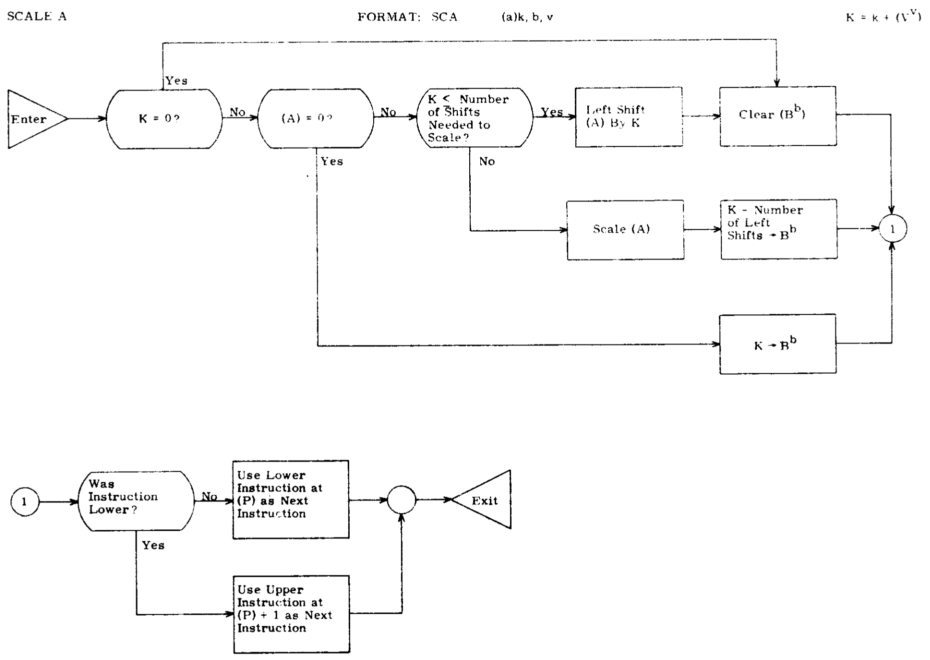
0-7, transmits number to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

SCA

F L O W C H A R T



SCALE A
FORMAT: SCA (a)k, b, v
K = k + (V^v)
Enter
K = 0?
Yes
No
(A) = 0?
Yes
No
K < Number of Shifts Needed to Scale?
Yes
No
Left Shift (A) By K
Clear (B^b)
Scale (A)
K - Number of Left Shifts → B^b
K → B^b
1
Was Instruction Lower?
No
Yes
Use Lower Instruction at (P) as Next Instruction
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

D E S C R I P T I O N

SCA

1) Scale (A) or shift until K goes to zero

2) Scale factor saved in B^b

PROBLEM:

Scale A using 2057 octal for K. Scale factor goes to Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCA	2057B,1	

PROBLEM:

Scale A using 3 for K. Scale factor goes to Index Register 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCA	3,2	

PROBLEM:

Scale A using the contents of Index Register 4 for K. Scale factor goes to Index Register 5.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCA	,5,4	

PROBLEM:

Scale A using k = 15 modified by Index Register 3 for K. Scale factor goes to Index Register 4.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCA	15,4,3	

The SCALE AQ Instruction

The SCALE AQ instruction is an instruction that scales an operand in the AQ registers. The scale factor, K minus the number of left shifts needed to scale, replaces the contents of the designated index register specified by <u>b</u>.

If the contents of AQ are originally zero, scaling is not performed and K replaces the contents of the index register.

If the operand is already scaled, K replaces the contents of the index register.

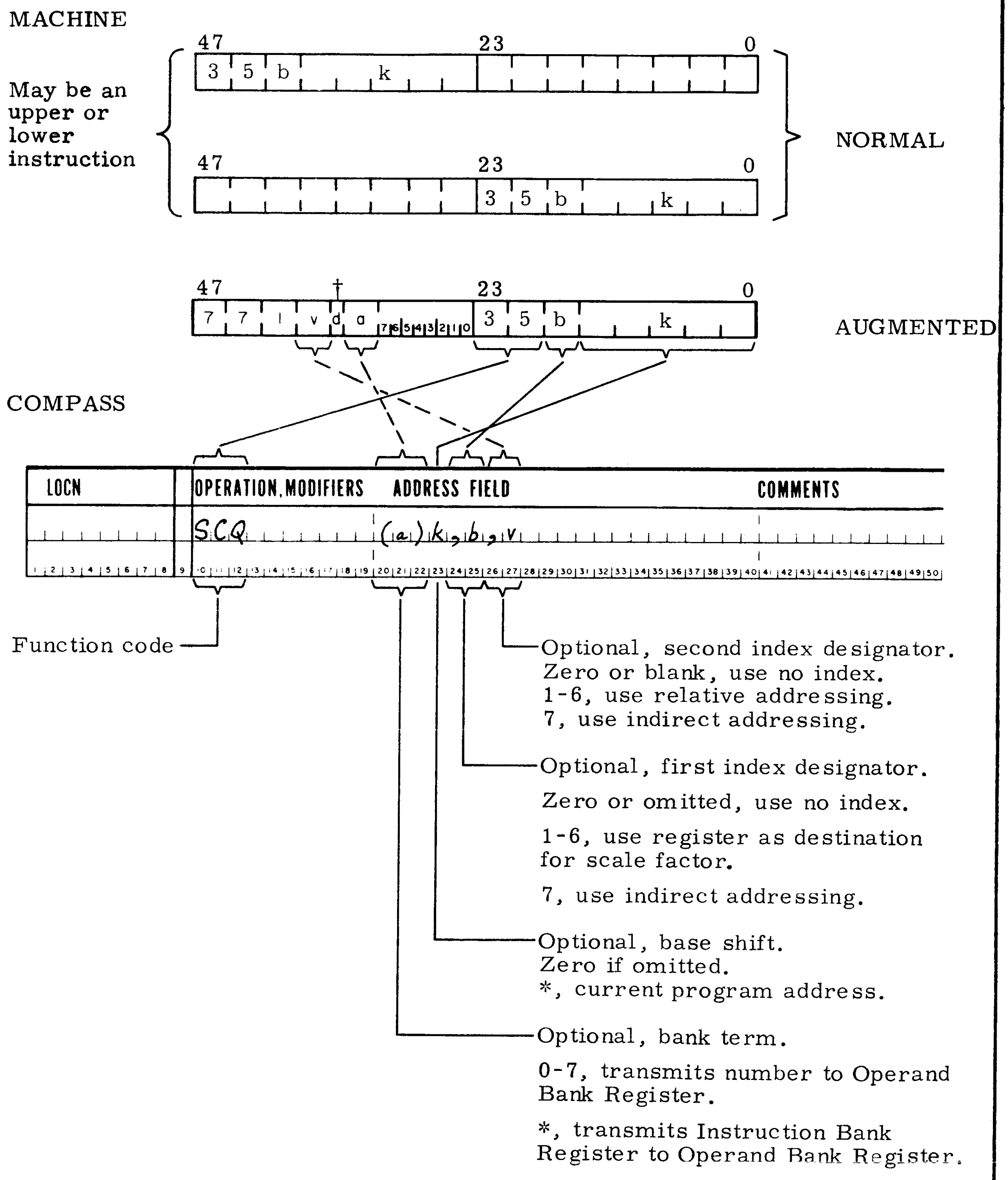
If K is less than the number of left shifts needed to scale, only K number of left shifts will result. The operand will not be scaled and zeros will replace the contents of the index register.

The bank term <u>a</u> determines the value of the operand bank setting. If it is not used, the current operand setting will be replaced by the value <u>a</u>. In either case it will not affect this instruction. However, it could affect future instructions that reference memory for operands.

SCALE AQ

SCQ

FORMAT



Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.

Zero or omitted, use no index.

1-6, use register as destination for scale factor.

7, use indirect addressing.

Optional, base shift.
Zero if omitted.
*, current program address.

Optional, bank term.

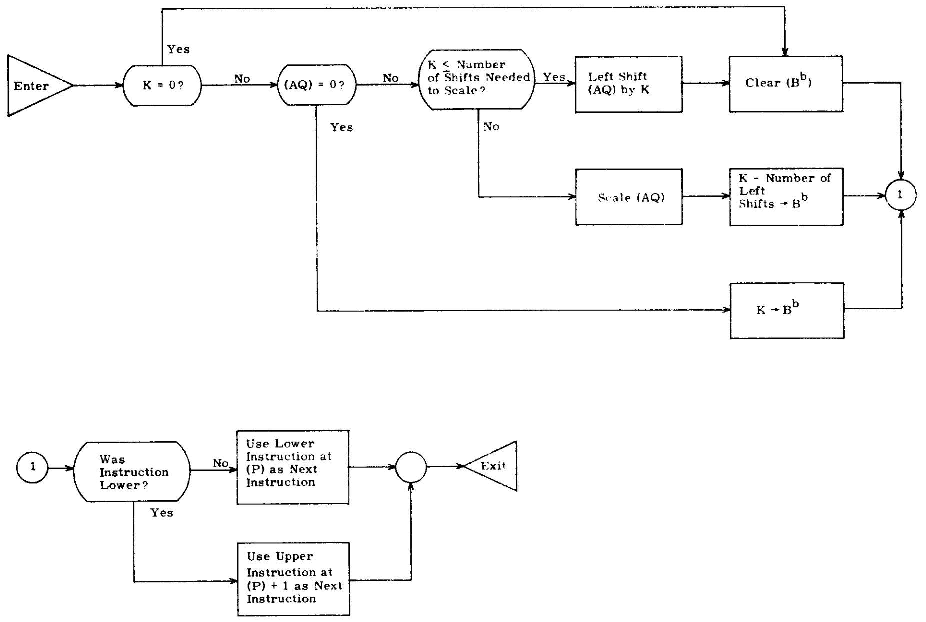
0-7, transmits number to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

FLOWCHART

SCALE AQ FORMAT: SCQ (a) k, b, v $K = k + (V^v)$


Enter
K = 0?
Yes
No
(AQ) = 0?
Yes
No
K ≤ Number of Shifts Needed to Scale?
Yes
No
Left Shift (AQ) by K
Clear (B^b)
Scale (AQ)
K - Number of Left Shifts → B^b
K → B^b
1
1
Was Instruction Lower?
No
Yes
Use Lower Instruction at (P) as Next Instruction
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

SCQ

1) Scale (AQ) or shift until K goes to zero
2) Scale factor saved in B^b

PROBLEM:

Scale AQ using 2057 octal for K. Scale factor goes to Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCQ	2057B,1	

PROBLEM:

Scale AQ using the contents of Index Register 3 for K. Scale factor goes to Index Register 4.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCQ	,4,3	

PROBLEM:

Scale AQ using 6 for K. Scale factor goes to Index Register 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCQ	6,2	

PROBLEM:

Scale AQ using 9 modified by Index Register 5 for K. Scale factor goes to Index Register 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCQ	9,1,5	

NEW CONCEPTS OF GROUP 12

For any register, whether it be A, Q, or AQ, it should be remembered that the normal shifts to the right are end-off with sign extended, and that the normal shifts to the left are end-around.

Operation field modifiers can change the pattern of shifting. The modifier EO means end-off. This means that no matter which direction the shift, it is always end-off, and zeros are always extended. It's like a table holding a line of rubber balls. As you push the balls in either direction, they fall off and none replace them. The modifier SS will change the direction of the shift if $K < 0$. Usually K varies during the program from positive to negative and vice versa allowing the programmer to position and reposition data.

The scale instructions allow the programmer to convert numbers to different computer formats. Here the left shift takes place until the most significant bit is just to the right of the sign bit. This is so that the sign of the operand does not change and so that it is in fractional format.

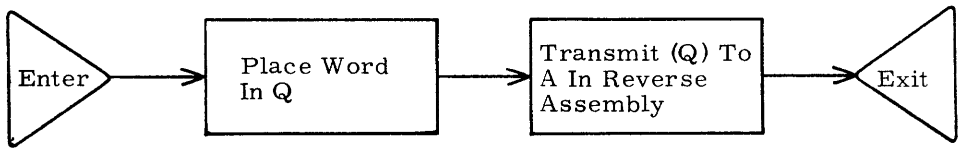
For each left shift the contents of an index register is reduced by 1. This quantity is called the scale factor. For each left shift of 1 the operand is effectively multiplied by 2. Correspondingly the exponent is reduced by 1. This means that the operand with the exponent always retains the same value. In other words an octal $5 \cdot 2^6$ is equal to an octal $50 \cdot 2^3$ (note the multiplying of the coefficient and the reducing of the exponent).

Problem 12:

Assume a set of BCD characters has been read into memory starting at address INPUTREG, but that the characters are backwards. Write a subprogram that will reverse the assembly of the first word. In other words:

(INPUTREG) = HGFEDCBA should be (INPUTREG) = ABCDEFGH

Flowchart:



Problem 12 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	REVERSE	
	ENTRY	REVERSE	
REVERSE	BSS	1	
	ENI	0,1	CHARACTER COUNTER
	LDA	INPUTREG	
	LRS,EØ	48	
	QLS	42	
	LLS	6	H
NEXTCHAR	QLS	36	
	LLS	6	
	ISK	6,1	8-2 LEFT
	SLJ	NEXTCHAR	
	STA	INPUTREG	
	SLJ	REVERSE	
	END		

Somewhere within this subprogram would also be included the symbol INPUTREG in the location field with a declaration of the prestored data or area reserved.

The forming of the reverse assembly can be seen more clearly by the following figure:

```
Instruction:  LDA    INPUTREG
              LRS, EØ    48
```

$(A) = 0 \longrightarrow 0$ (Q) = H G F E D C B A

```
    Instruction:    QLS    42
                    LLS    6
```

$(A) = 0 \longrightarrow 0A \qquad (Q) = \text{H G F E D C B 00}$

```
    Instruction:    QLS    36
                    LLS    6
```

$(A) = 0 \longrightarrow 0\text{A B} \qquad (Q) = \text{00 H G F E D C 00}$

Student Problem 12A:

A group of BCD characters at address REVASS looks like the following:

(REVASS) = GHEFCDAB

Rearrange this word so that the letters are in alphabetical order.

Flowchart:

Problem 12A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

GROUP 13

LOGICAL OPERATIONS

GROUP 13

LOGICAL OPERATIONS

1. Selective Set	SST
2. Selective Clear	SCL
3. Selective Complement	SCM
4. Selective Substitute	SSU
5. Load Logical	LDL
6. Add Logical	ADL
7. Subtract Logical	SBL
8. Store Logical	STL

This group of instructions is used to manipulate binary data, and to set, clear, extract, and insert bits of data. One memory reference is required.

The first three instructions will change bits in A depending on a mask in memory. The fourth instruction will substitute the bits of a memory word for A depending on the mask in Q.

The last four instructions perform a logical product using the contents of Q as a mask.

The SELECTIVE SET Instruction

The SELECTIVE SET instruction is an instruction that checks each bit from an 18-bit storage address. One memory reference is made.

The operation leaves the contents of the storage address unchanged. For every bit in the storage address that is in the set state (binary 1), the corresponding bit in the A register will be set, regardless of its initial state. Where there is a zero bit in the storage address, the corresponding bit in A will remain unchanged.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

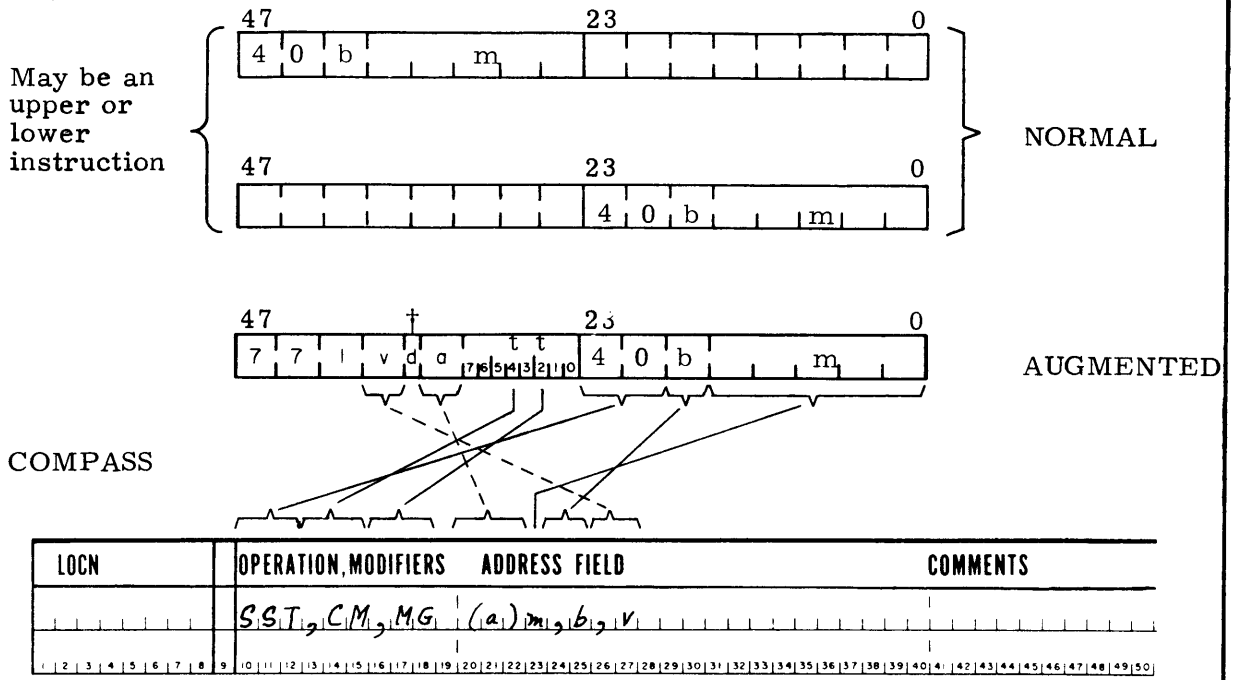
If CM is specified by the programmer, the complement of the operand is read from memory and then each bit is checked as mentioned above.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is read from memory and then each bit is checked as mentioned above.

FORMAT

MACHINE

May be an upper or lower instruction



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

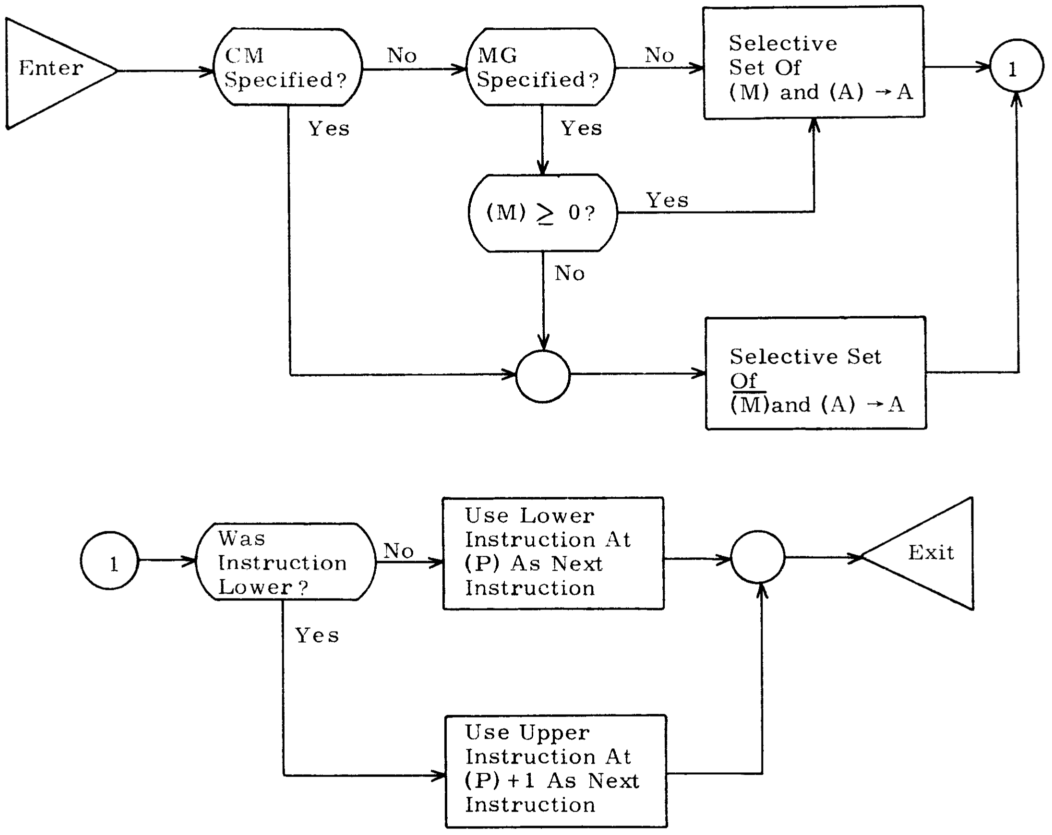
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

SST

F L O W C H A R T

SELECTIVE SET FORMAT: SST,CM,MG (a)m,b,v $M = m + (B^b) + (V^v)$


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
Selective Set Of (M) and (A) → A
1
(M) ≥ 0?
Yes
No
Selective Set Of $\overline{(M)}$ and (A) → A
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

D E S C R I P T I O N

SST	1) Selective Set of (M) and (A) → A
	2) Contents of M remain unchanged
SST, CM	1) Selective Set of $\overline{(M)}$ and (A) → A
	2) Contents of M remain unchanged
SST, MG	1) Selective Set of $\lvert(M)\rvert$ and (A) → A
	2) Contents of M remain unchanged

E
X
A
M
P
L
E
S

PROBLEM:

Perform a Selective Set with the contents of A and the storage address MSK.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SST	MSK	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Perform a Selective Set using the contents of A and the storage address TOT modified by Index Registers 2 and 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SST	TØT,2,3	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Perform a Selective Set using the contents of A and the storage address specified in Index Register 6.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SST	,6	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Perform a Selective Set using the contents of A and the storage address DICK from the bank where DICK resides.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SST	($)DICK	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The SELECTIVE CLEAR Instruction

The SELECTIVE CLEAR instruction is an instruction that checks each bit from an 18-bit storage address. One memory reference is made.

The operation leaves the contents of the storage address unchanged. For every bit in the storage address that is in the set state (binary 1), the corresponding bit in the A register will be cleared, regardless of its initial state. Where there is a zero bit in the storage address, the corresponding bit in A will remain unchanged.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m+(B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is read from memory and then each bit is checked as mentioned above.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is read from memory and then each bit is checked as mentioned above.

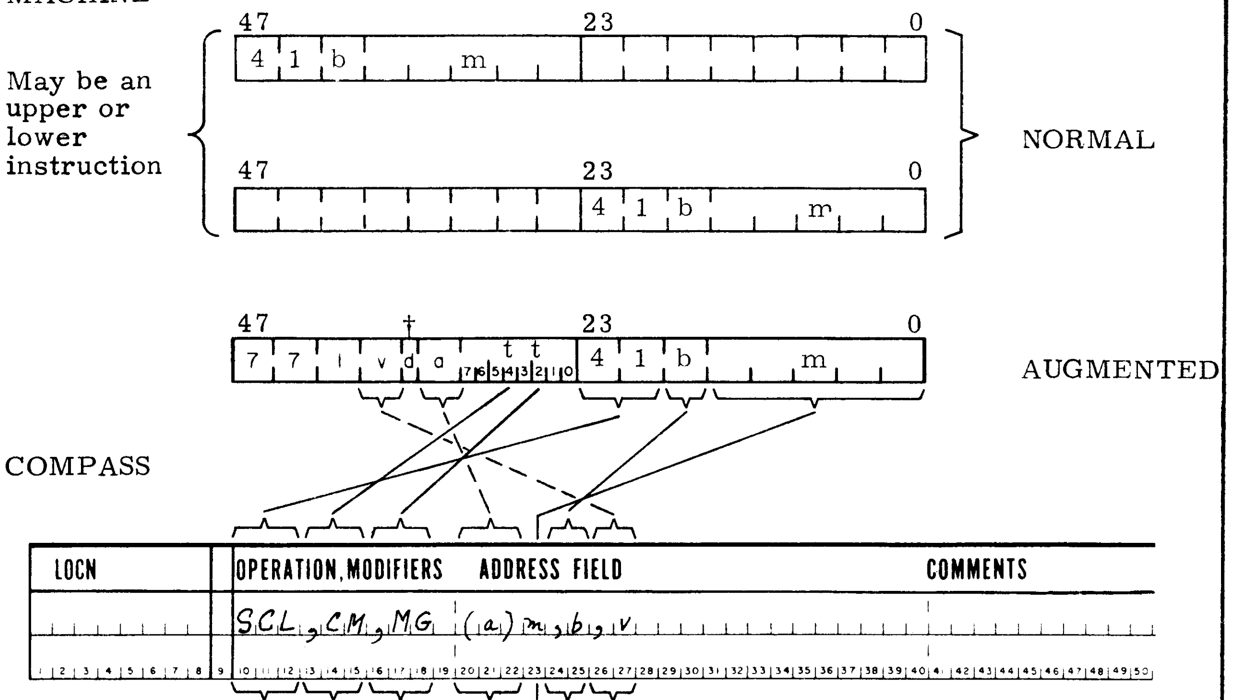
SELECTIVE CLEAR

SCL

FORMAT

MACHINE

May be an upper or lower instruction



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register

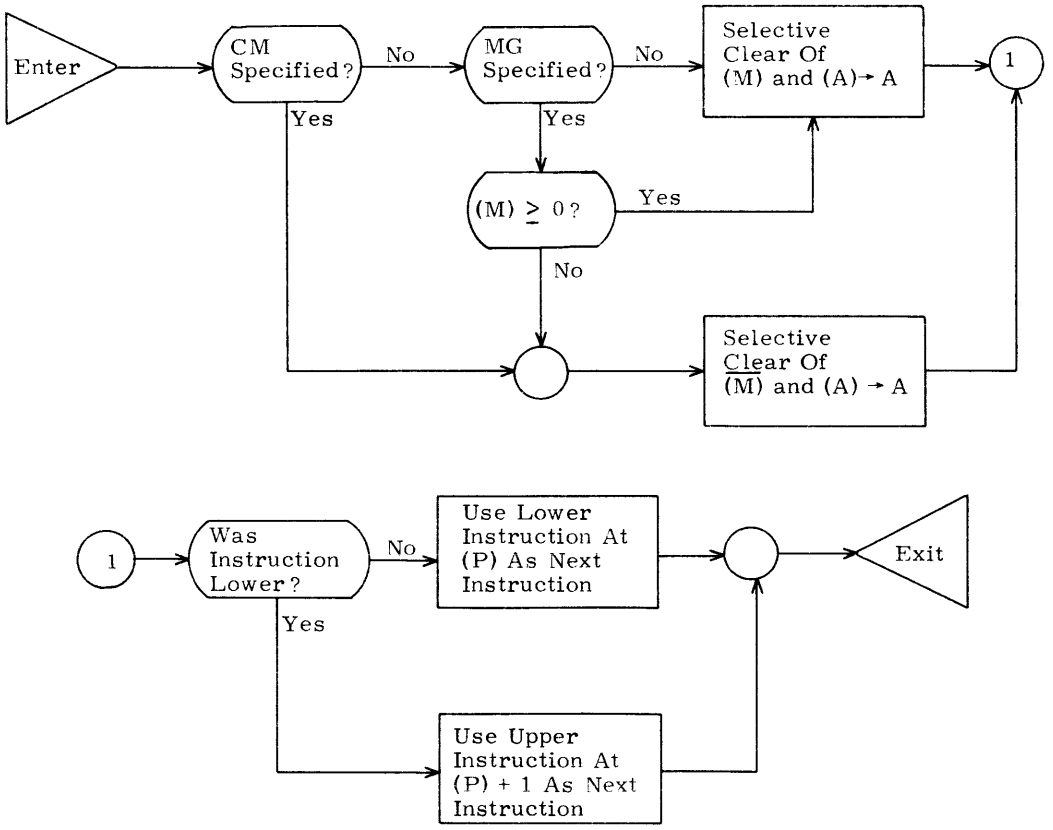
*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

SCL

F L O W C H A R T

SELECTIVE CLEAR FORMAT: SCL,CM,MG (a)m,b,v $M = m + (B^b) + (V^v)$


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
Selective Clear Of (M) and (A) → A
1
(M) ≥ 0 ?
Yes
No
Selective Clear Of $\overline{(M)}$ and (A) → A
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

D E S C R I P T I O N

SCL	1) Selective Clear of (M) and (A) → A
	2) Contents of M remain unchanged
SCL, CM	1) Selective Clear of $\overline{(M)}$ and (A) → A
	2) Contents of M remain unchanged
SCL, MG	1) Selective Clear of $\lvert(M)\rvert$ and (A) → A
	2) Contents of M remain unchanged

PROBLEM:

Perform a Selective Clear using the contents of A and the storage address MSK1.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SCL	MSK1	

PROBLEM:

Perform a Selective Clear using the contents of A and the storage address FIX modified by Index Registers 3 and 4.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SCL	FIX,3,4	

PROBLEM:

Perform a Selective Clear using the contents of A and the storage address specified in Index Register 5.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SCL	,5	

PROBLEM:

Perform a Selective Clear using the contents of A and the storage address SCRIPT in the bank where the SCL resides.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SCL	(*)SCRIPT	

The SELECTIVE COMPLEMENT Instruction

The SELECTIVE COMPLEMENT instruction is an instruction that checks each bit from an 18-bit storage address. One memory reference is made.

The operation leaves the contents of the storage address unchanged. For every bit in the storage address that is in the set state (binary 1), the corresponding bit in the A register will be complemented, regardless of its initial state. Where there is a zero bit in the storage address, the corresponding bit in A will remain unchanged.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is read from memory and then each bit is checked as mentioned above.

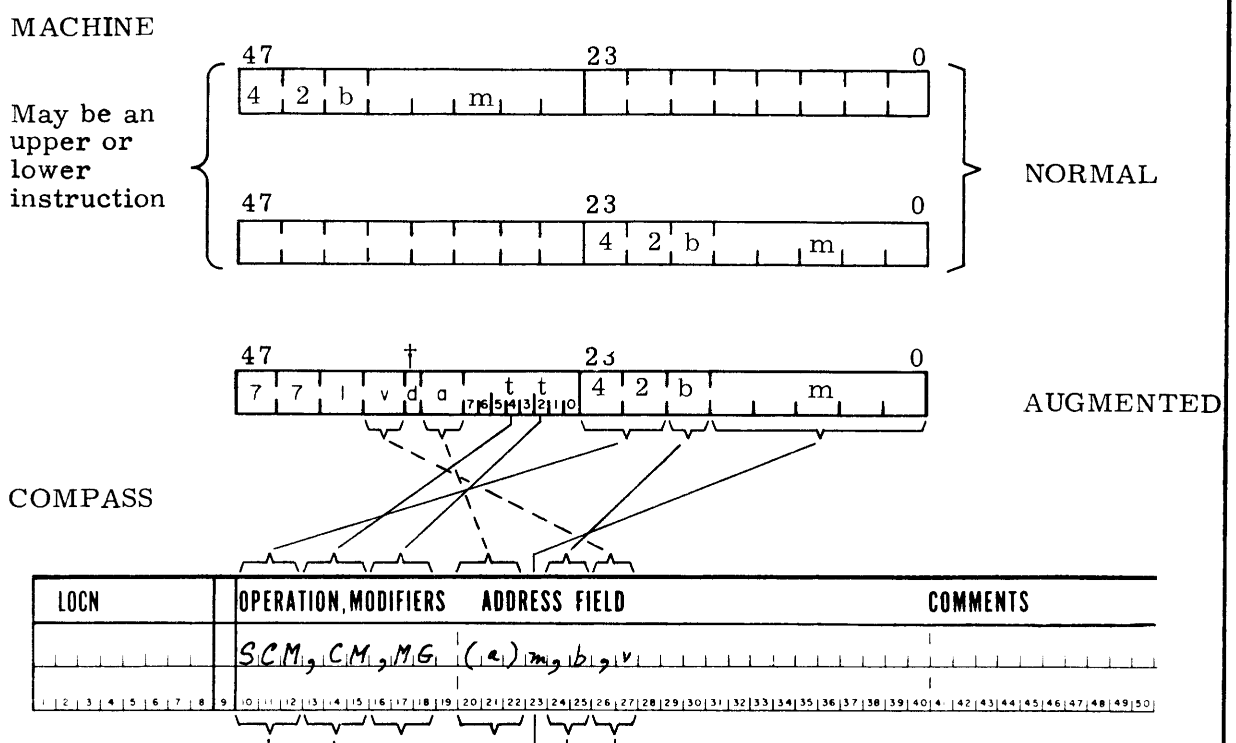
If MG is specified by the programmer, the magnitude (absolute value) of the operand is read from memory and then each bit is checked as mentioned above.

SELECTIVE COMPLEMENT

SCM

FORMAT

MACHINE



Function code

Optional, transmit complement of operand.

Optional, transmit magnitude of operand.

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

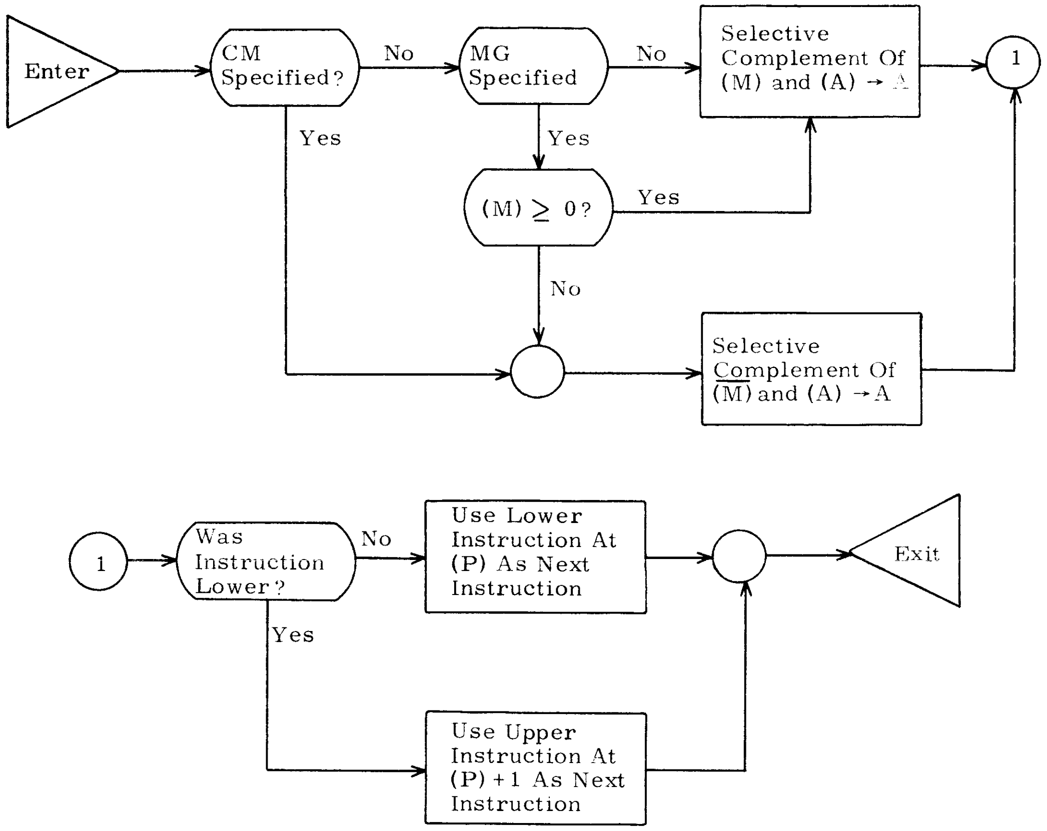
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

SCM

FLOWCHART

SELECTIVE COMPLEMENT FORMAT: SCM,CM,MG (a)m,b,v $M = m + (B^b) + (V^v)$



Enter
CM Specified?
No
MG Specified
No
Selective Complement Of (M) and (A) → A
1
Yes
Yes
(M) ≥ 0?
Yes
No
Selective Complement Of $\overline{(M)}$ and (A) → A
1
Was Instruction Lower?
No
Use Lower Instruction At (P) As Next Instruction
Exit
Yes
Use Upper Instruction At (P) + 1 As Next Instruction

DESCRIPTION

SCM	1) Selective Complement of (M) and (A) → A
	2) Contents of M remain unchanged
SCM, CM	1) Selective Complement of $\overline{(M)}$ and (A) → A
	2) Contents of M remain unchanged
SCM, MG	1) Selective Complement of $\|(M)\|$ and (A) → A
	2) Contents of M remain unchanged

PROBLEM:

Perform a Selective Complement using the contents of A and the storage address MAX.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCM	MAX	

PROBLEM:

Perform a Selective Complement using the contents of A and the storage address SUP modified by Index Registers 1 and 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCM	SUP,1,2	

PROBLEM:

Perform a Selective Complement using the contents of A and the storage address specified in Index Register 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCM	,3	

PROBLEM:

Perform a Selective Complement using the contents of A and the storage address SWITCH in the bank of SWITCH.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SCM	($)SWITCH	

The SELECTIVE SUBSTITUTE Instruction

The SELECTIVE SUBSTITUTE instruction is an instruction that checks each bit from an 18-bit storage address. One memory reference is made.

The operation leaves the contents of the storage address unchanged. For every bit in the Q register that is in the set state (binary 1), the corresponding bit from the storage address will be transferred to the A register. Where there is a zero bit in the Q register, the corresponding bit from the storage address will not be transferred to A.

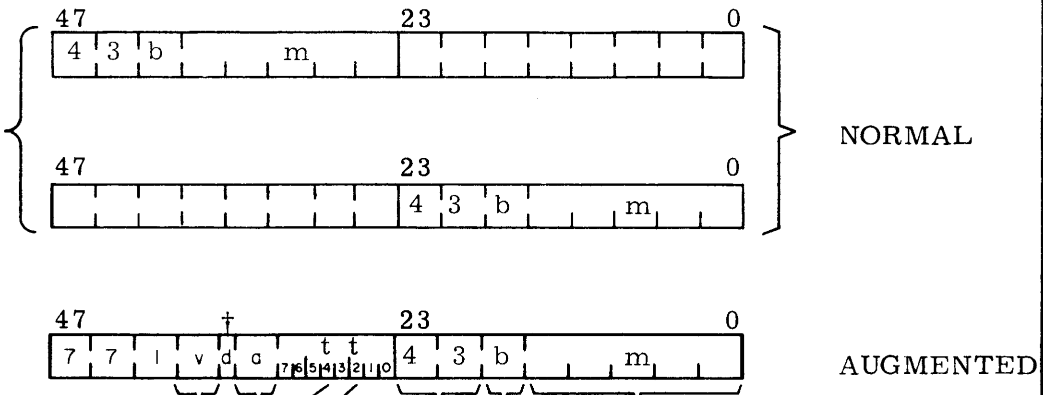
The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is read from memory and then each bit is checked as mentioned above.

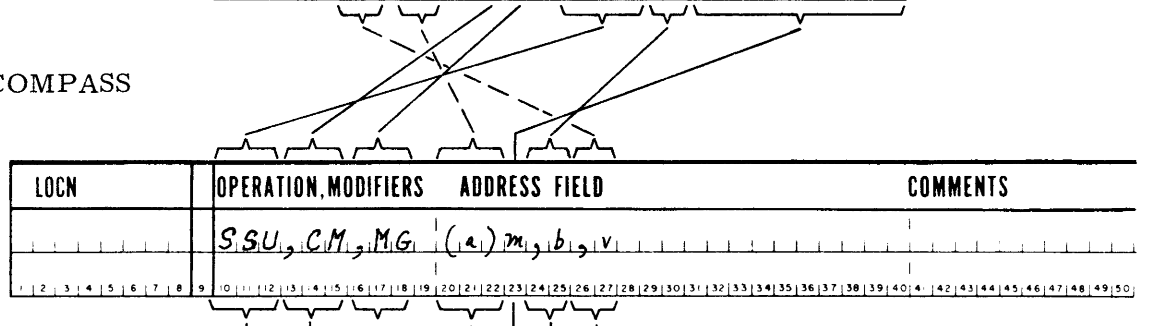
If MG is specified by the programmer, the magnitude (absolute value) of the operand is read from memory and then each bit is checked as mentioned above.

FORMAT

MACHINE



COMPASS



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator. Zero or blank, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.

Optional, base address. Zero if omitted. *, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

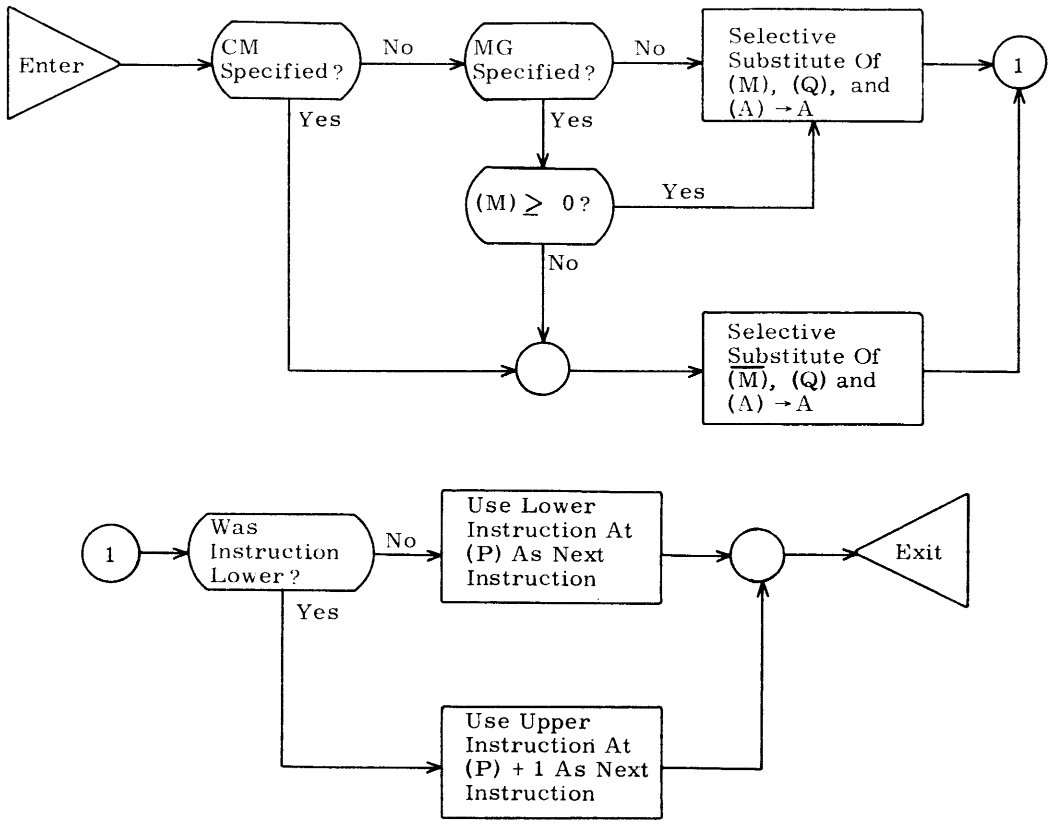
$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

[†] d bit not programmable

FLOWCHART

SELECTIVE SUBSTITUTE FORMAT: SSU,CM,MG (a)m,b,v $M = m + (B^b) + (V^v)$


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
Selective Substitute Of (M), (Q), and (A) → A
1
(M) ≥ 0?
Yes
No
Selective Substitute Of $\overline{(M)}$, (Q) and (A) → A
1
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

DESCRIPTION

SSU	1) Selective Substitute of (M), (Q), and (A) → A
	2) Contents of M remain unchanged
SSU, CM	1) Selective Substitute of $\overline{(M)}$, (Q), and (A) → A
	2) Contents of M remain unchanged
SSU, MG	1) Selective Substitute of $\lvert(M)\rvert$, (Q), and (A) → A
	2) Contents of M remain unchanged

PROBLEM: Given: (Q) = 77777 octal

Perform the Selective Substitute with the contents of A and storage address HENCH.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SSU	HENCH	

PROBLEM: Given: (Q) = 770———0 octal

Perform the Selective Substitute with the contents of A and storage address BOX modified by Index Registers 2 and 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SSU	BØX,2,3	

PROBLEM: Given: (Q) = 7 octal

Perform the Selective Substitute with the contents of A and the storage address specified in Index Register 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SSU	,3	

PROBLEM: Given: (Q) = 77 octal

Perform the Selective Substitute with the contents of A and the storage address HOMER from the bank where HOMER resides.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SSU	($)HØMER	

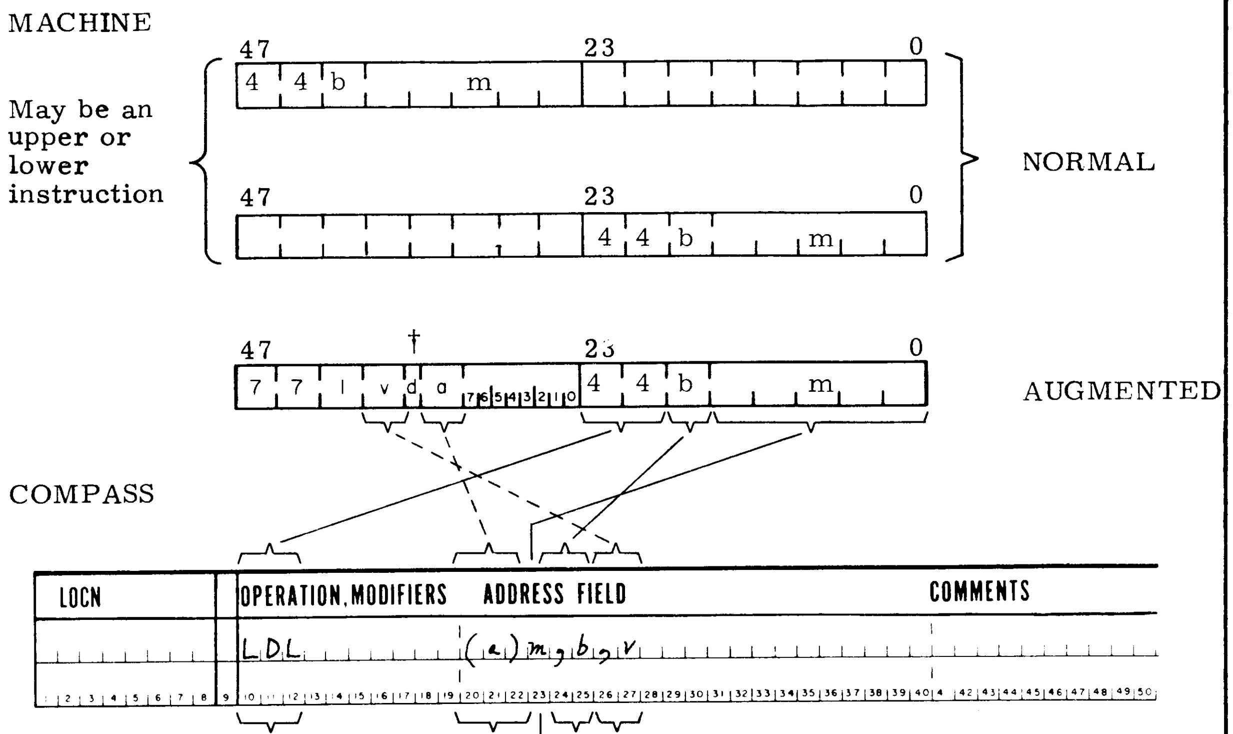
The LOAD LOGICAL Instruction

The LOAD LOGICAL instruction is an instruction that forms the logical product between the contents of an 18-bit storage address and the contents of the Q register with the result replacing the contents of A. One memory reference is made.

The operation leaves the contents of the storage address unchanged.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

FORMAT



Function code

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register

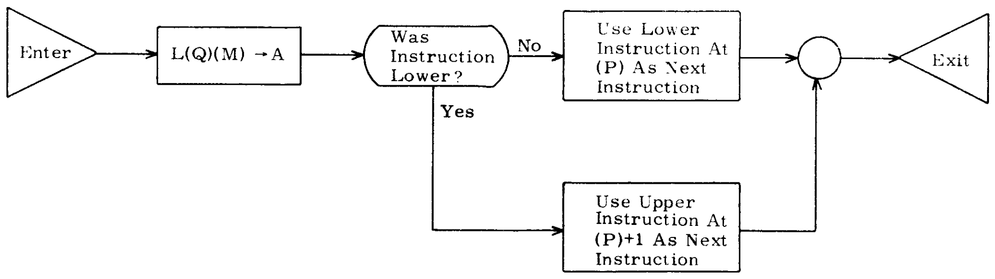
† d bit not programmable

FLOWCHART

LOAD LOGICAL

FORMAT: LDL (a)m, b, v

$M = m + (B^b) + (V^v)$



Enter
L(Q)(M) → A
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

LDL

1) L (Q) (M) → A

2) Contents of M remain unchanged

PROBLEM: Given: (Q) = 77 octal

Load the lowest six bits from address JACK into the A register.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LDL	JACK	

PROBLEM: Given: (Q) = 770——0 octal

Load the highest six bits from address CHAR modified by Index Register 3 into the A register.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LDL	CHAR,3	

PROBLEM: Given: (Q) = 77777 octal

Load the lowest 15 bits from the address specified in Index Register 6 into the A register.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LDL	,6	

PROBLEM: Given: (Q) = 777770——0 octal

Load the highest 15 bits from address KARAC from the bank where KARAC resides.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LDL	($)KARAC	

The ADD LOGICAL Instruction

The ADD LOGICAL instruction is an instruction that forms the logical product between the contents of an 18-bit storage address and the contents of the Q register and adds this result to the contents of A. One memory reference is made.

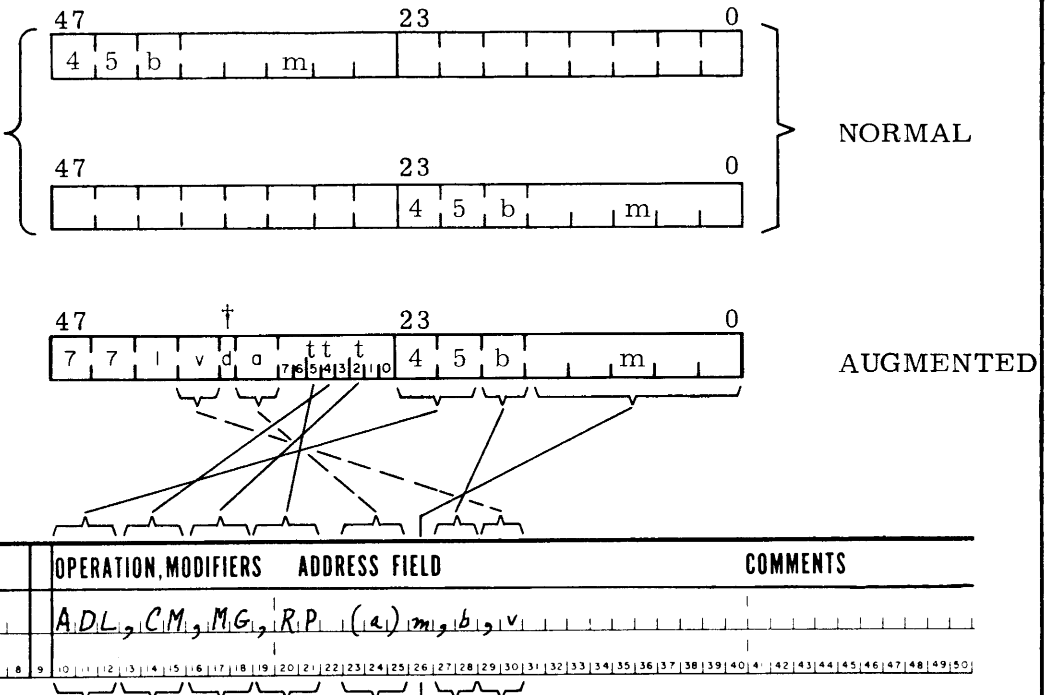
The operation leaves the contents of the storage address unchanged.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

FORMAT

MACHINE

May be an upper or lower instruction



COMPASS

Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, replace result at M

Optional, second index designator
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

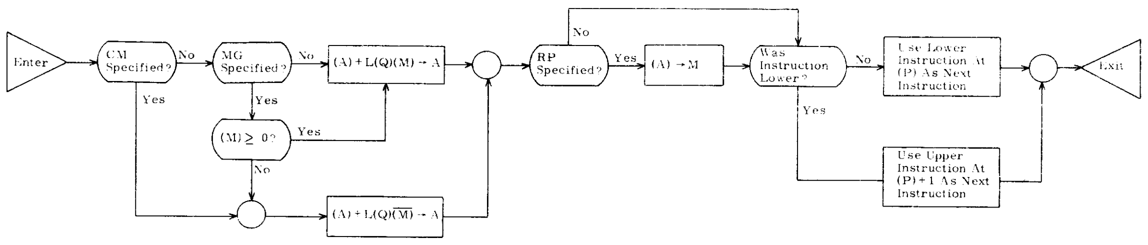
$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

FLOWCHART

ADD LOGICAL FORMAT: ADL, RP, CM, MG (a) m, b, v $M = m + (B^b) + (V^v)$


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(A) + L(Q)(M) → A
(M) ≥ 0?
Yes
No
(A) + L(Q)$\overline{(M)}$ → A
RP Specified?
Yes
No
(A) → M
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

ADL	1) (A) + L (Q) (M) → A
	2) Contents of M remain unchanged
ADL, RP	(A) + L (Q) (M) → A and M
ADL, CM	1) (A) + L (Q) $\overline{(M)}$ → A
	2) Contents of M remain unchanged
ADL, MG	1) (A) + L (Q) $\lvert (M) \rvert$ → A
	2) Contents of M remain unchanged

PROBLEM: Given: (Q) = 77 octal

Add the lowest 6-bit character from address TAB to the A register.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ADL	TAB	

PROBLEM: Given: (Q) = 770——0 octal

Add the highest 6-bit character from address SAM modified by Index Register 2 to the A register.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ADL	SAM,2	

PROBLEM: Given: (Q) = 77777 octal

Add the lowest 15 bits from the address specified in Index Register 4 to the A register.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ADL	,4	

PROBLEM: Given: (Q) = 777770——0 octal

Add the highest 15 bits from address BULDGE from the bank where the ADL resides.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	ADL	(*)BULDGE	

The SUBTRACT LOGICAL Instruction

The SUBTRACT LOGICAL instruction is an instruction that forms the logical product between the contents of an 18-bit storage address and the contents of the Q register and subtracts this result from the contents of A. One memory reference is made.

The operation leaves the contents of the storage address unchanged.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

SUBTRACT LOGICAL

SBL

FORMAT

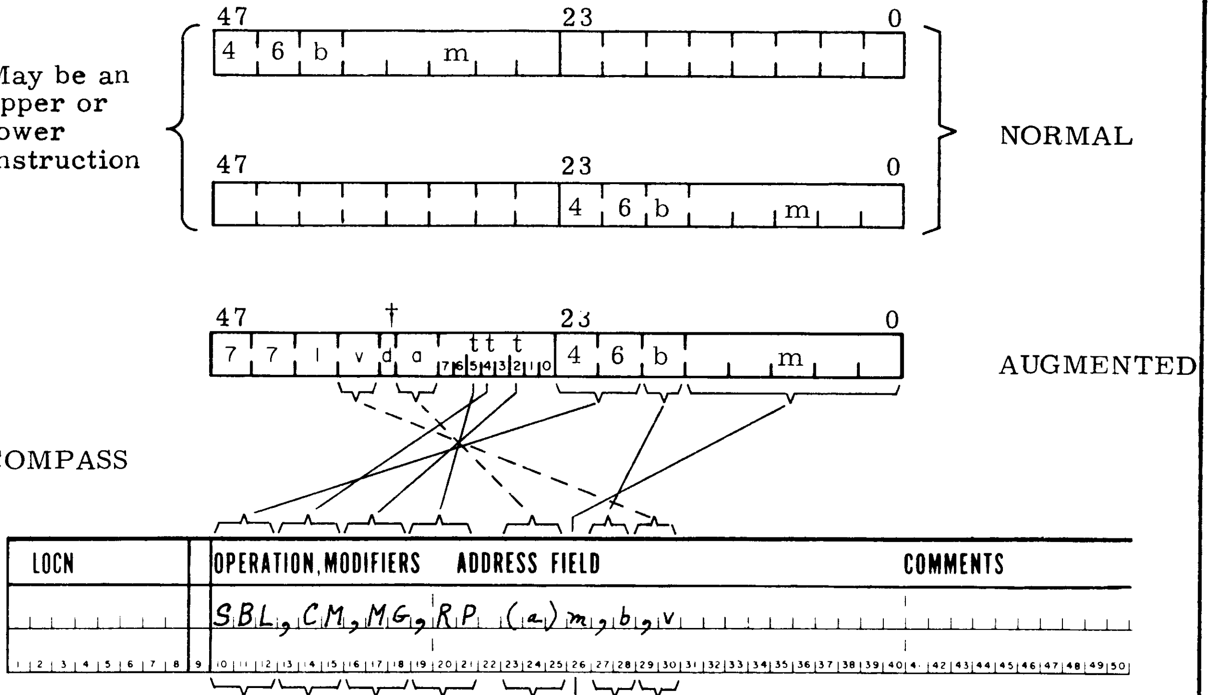
MACHINE

May be an upper or lower instruction

NORMAL

AUGMENTED

COMPASS


47
23
0
4
6
b
m
7
7
l
v
d
a
t t t
4
6
b
m
LOCN
OPERATION, MODIFIERS ADDRESS FIELD
COMMENTS
SBL, CM, MG, RP (a) m, b, v

Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, replace result at M

Optional, second index designator
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

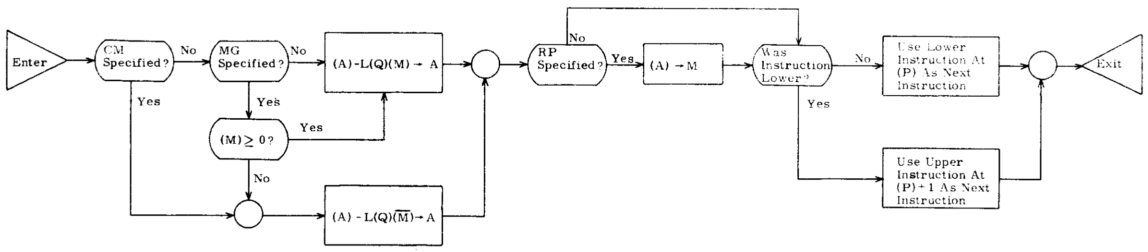
*, transmits Instruction Bank Register to Operand Bank Register.

† d bit not programmable

SBL

FLOWCHART

SUBTRACT LOGICAL FORMAT: SBL, RP, CM, MG (a) m, b, v $M = m + (B^b) + (V^v)$


Enter
CM Specified?
No
Yes
MG Specified?
No
Yes
(A)-L(Q)(M) → A
(M) ≥ 0?
Yes
No
(A) - L(Q)($\overline{M}$) → A
RP Specified?
Yes
No
(A) → M
Was Instruction Lower?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P)+1 As Next Instruction
Exit

DESCRIPTION

SBL	1) (A) - L (Q) (M) → A
	2) Contents of M remain unchanged
SBL, RP	(A) - L (Q) (M) → A and M
SBL, CM	1) (A) - L (Q) $\overline{(M)}$ → A
	2) Contents of M remain unchanged
SBL, MG	1) (A) - L (Q) $\|(M)\|$ → A
	2) Contents of M remain unchanged

EXAMPLES

PROBLEM: Given: (Q) = 77 octal

Subtract the lowest 6 bits at address BAKER from the A register.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SBL	BAKER	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM: Given: (Q) = 77777 octal

Subtract the lowest 15 bits at address ABLE modified by Index Register 5 from the A register.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SBL	ABLE,5	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM: Given: (Q) = 000777770——0 octal

Subtract the indicated 15 bits at the address specified in Index Register 1 from the A register.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SBL	,1	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM: Given: (Q) = 0———077700000 octal

Subtract the indicated nine bits at address DOG in the bank where DOG resides from the A register.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		SBL	($)DØG	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The STORE LOGICAL Instruction

The STORE LOGICAL instruction is an instruction that forms the logical product between the contents of the A register and the contents of the Q register with the result replacing the contents of an 18-bit storage address. One memory reference is made.

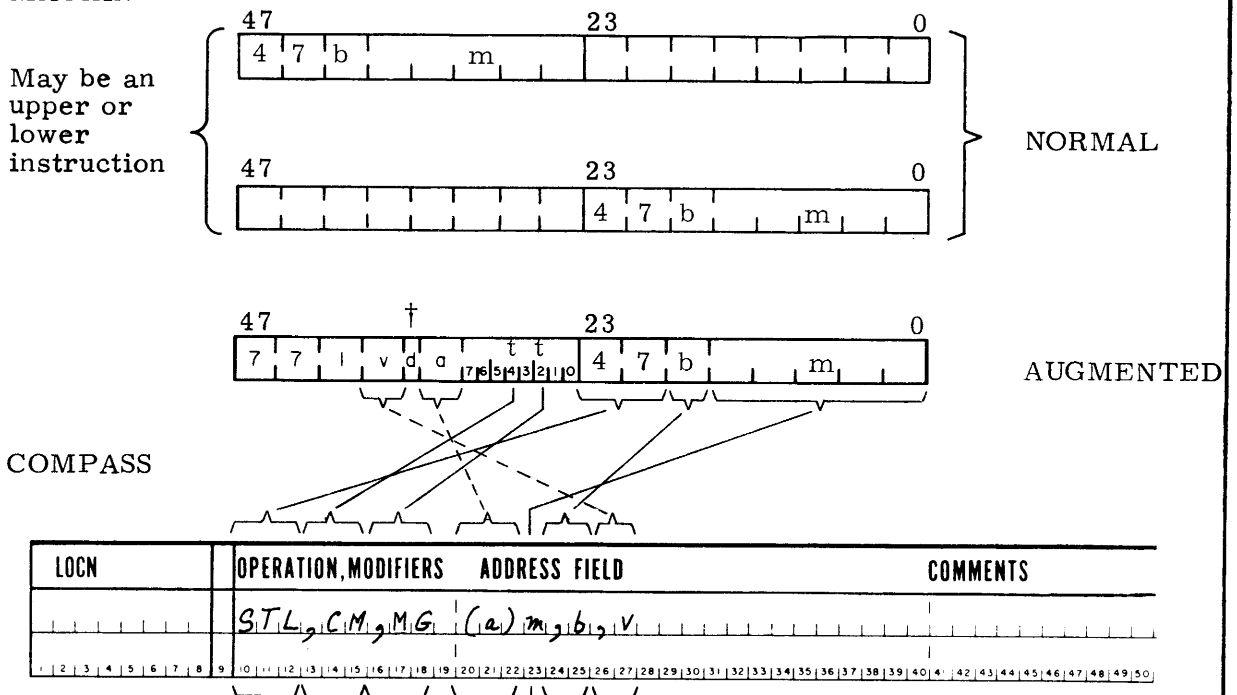
The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

STORE LOGICAL

STL

FORMAT

MACHINE



Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, second index designator.
Zero or blank, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.

Optional, base address.
Zero if omitted.
*, current program address.

Optional, bank term.

0-7, transmits number to Operand Bank Register.

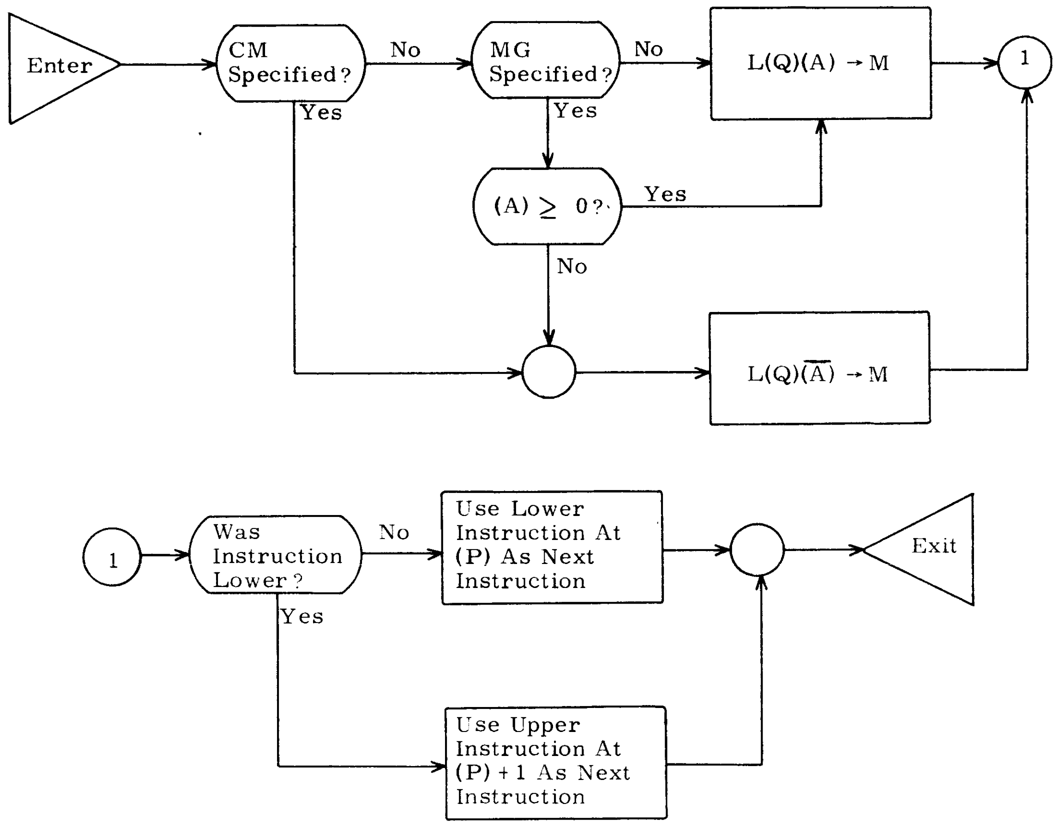
$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

†d bit not programmable

FLOWCHART

STORE LOGICAL FORMAT: STL,CM,MG (a)m,b,v $M = m + (B^b) + (V^v)$



DESCRIPTION

STL	1) L (Q) (A) → M
	2) Contents of A remain unchanged
STL, CM	1) L (Q) $\overline{(A)}$ → M
	2) Contents of A remain unchanged
STL, MG	1) L (Q) $\lvert(A)\rvert$ → M
	2) Contents of A remain unchanged

E
X
A
M
P
L
E
S

PROBLEM: Given: (Q) = 77 octal

Store the lowest 6 bits of the A register at address TOM.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STL	TØM	

PROBLEM: Given: (Q) = 770——0 octal

Store the highest 6 bits of A at address PETE modified by Index Register 2 and 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STL	PETE,2,3	

PROBLEM: Given: (Q) = 77777 octal

Store the lowest 15 bits of A at the address specified in Index Register 4.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STL	,4	

PROBLEM: Given: (Q) = 77 octal

Store the lowest 15 bits of A at address JOE in the bank where the STL resides.

SOLUTION:

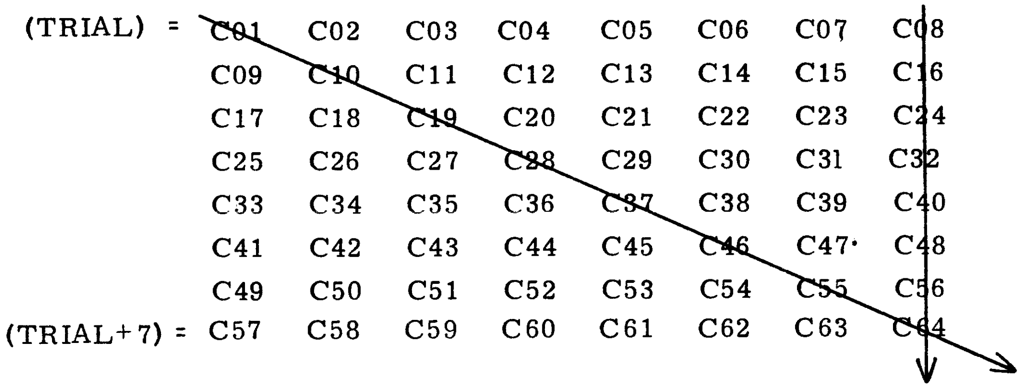
LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STL	(*)JØE	

NEW CONCEPTS OF GROUP 13

This group involves many logical operations, i. e. bit by bit comparisons. Bits are grouped in order to mask (extract) certain portions of words. Under "logical product" the multiplication of any bit by "1" returns that bit, whereas the multiplication of any bit by "0" returns zero. Using this principle the programmer can extract and analyze any part of a word.

Problem 13:

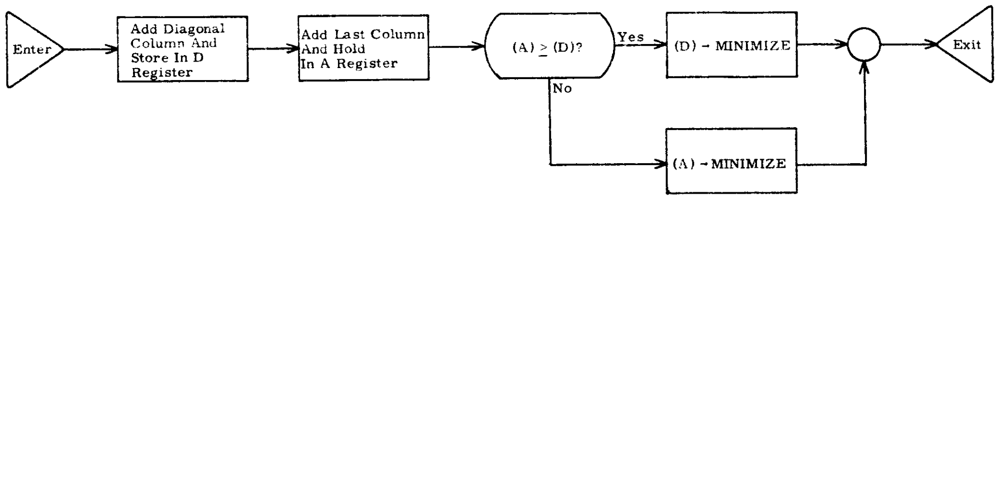
A set of 64 BCD characters representing numbers from 0 through 9 are in memory starting at address TRIAL. If each CXX represents one 6-bit character, they are set up as follows:



(TRIAL) =	C01	C02	C03	C04	C05	C06	C07	C08
	C09	C10	C11	C12	C13	C14	C15	C16
	C17	C18	C19	C20	C21	C22	C23	C24
	C25	C26	C27	C28	C29	C30	C31	C32
	C33	C34	C35	C36	C37	C38	C39	C40
	C41	C42	C43	C44	C45	C46	C47·	C48
	C49	C50	C51	C52	C53	C54	C55	C56
(TRIAL+7) =	C57	C58	C59	C60	C61	C62	C63	C64

Write a subprogram that will form the sum of the characters represented by the diagonal line and the sum of the characters represented by the vertical line. Compare the two sums and store the smaller sum at address MINIMIZE.

Flowchart:



Enter
Add Diagonal Column And Store In D Register
Add Last Column And Hold In A Register
(A) ≥ (D)?
Yes
No
(D) → MINIMIZE
(A) → MINIMIZE
Exit

Problem 13 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	MINFUN	
	ENTRY	MINFUN	
MINIMIZE	BSS	1	
MINFUN	BSS	1	
	ENI	0,1	
	ENA	0	
	RXT	A,D	
	ENQ	17B	MASK IN Q
	QLS	42	
NEXT	SSU	TRAIL,1	
	QRS	6	
	ISK	7,1	8 CHAR?
	SLJ	NEXT	
	LRS,EØ	48	
NEXT1	LLS	6	
	RØP,+	A,D,D	D+NEXT CHAR
	ENA	0	
	ISK	7,2	8 CHAR?
	SLJ	NEXT1	
	ENQ	17B	
NEXT2	ADL	TRIAL,1	
	ISK	7,1	8 CHAR?
	SLJ	NEXT2	
	RØP,-	A,D,Q	
	QJP,PL	DIAGSM	
	STA	MINIMIZE	A.LT.D
	SLJ	MINFUN	
DIAGSM	RXT	D,A	
	STA	MINIMIZE	D.LT.A
	SLJ	MINFUN	
	END		

Somewhere within this subprogram would also be included the symbol TRIAL in the location field with a declaration of the prestored data or area reserved.

It is left to the reader as a challenge to determine the contents of A and Q after each instruction, as was done in the previous problem.

Student Problem 13A:

Using problem 13, determine the minimum value between the sum of the 5th column and the opposite diagonal.

Flowchart:

Problem 13A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 14

SINGLE PRECISION FLOATING POINT ARITHMETIC

GROUP 14

SINGLE PRECISION FLOATING POINT ARITHMETIC

1.	Floating Add	FAD
2.	Floating Subtract	FSB
3.	Floating Multiply	FMU
4.	Floating Divide	FDV

This group of instructions performs floating point operations on the contents of A. One memory reference is required.

Both operands must be in floating point format. The computer will automatically normalize and round the final coefficient resulting from the operation unless UN or UR is specified. The operation always leaves the answer in A with the residue in Q.

The FLOATING ADD Instruction

The FLOATING ADD instruction is an instruction that adds a 48-bit operand from an 18-bit storage address to the contents of A in floating point format. One memory reference is made.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is added to A.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is added to A.

If RP is specified by the programmer, the result of the operation replaces the contents of the storage address besides the A register. In this case two memory references are made.

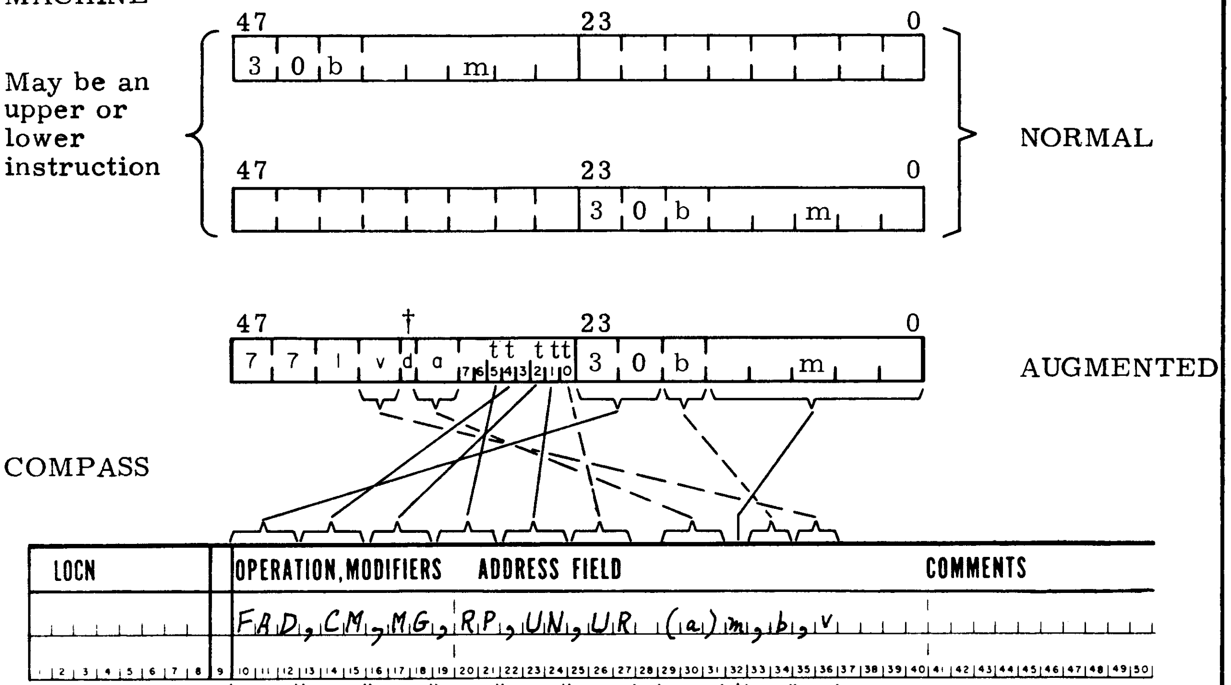
Normally the coefficient will be normalized. If UN is specified by the programmer, the coefficient will be unnormalized.

Normally the coefficient is rounded if one-half or greater. If UR is specified by the programmer, rounding will not take place.

The initial contents of the Q register are always destroyed by this instruction.

FORMAT

MACHINE


May be an upper or lower instruction
47
23
0
3 0 b m
NORMAL
AUGMENTED
7 7 1 v d a
tt ttt
†
COMPASS
LOCN
OPERATION, MODIFIERS ADDRESS FIELD
COMMENTS
FAD, CM, MG, RP, UN, UR (a) m, b, v

Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, replace result at M

Optional, leave coefficient un-normalized

Optional, leave coefficient un-rounded

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

Optional, second index designator.

Zero or blank, use no index.

1-6, use relative addressing.

7, use indirect addressing.

Optional, first index designator.

Zero or omitted, use no index.

1-6, use relative addressing.

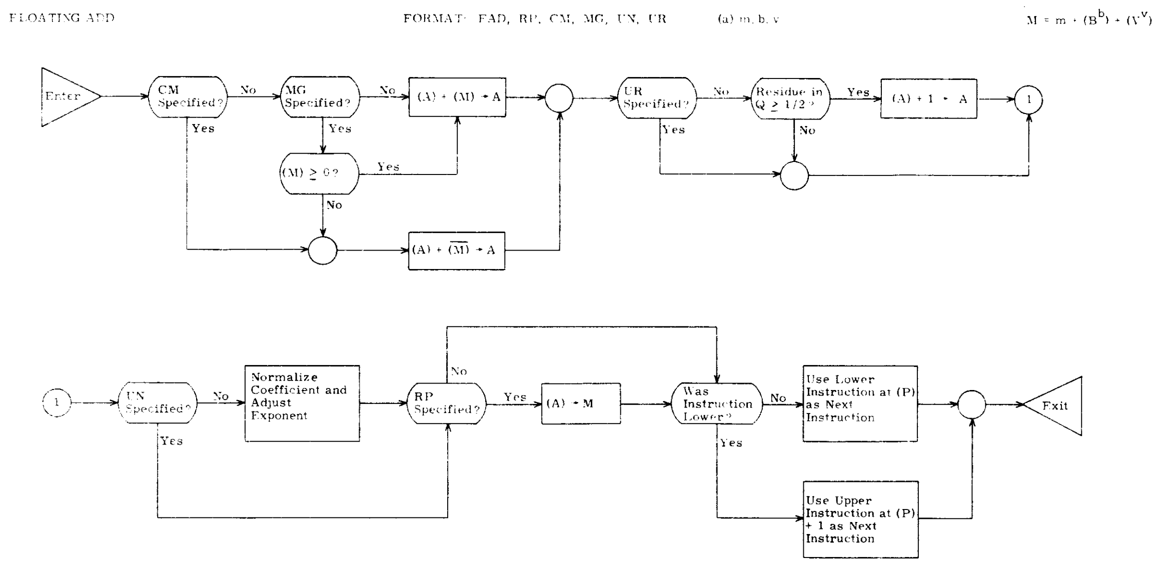
7, use indirect addressing.

Optional, base address. Zero if omitted.
*, current program address.

† d bit not programmable

FAD

FLOWCHART



DESCRIPTION

FAD	$(A) + (M) \rightarrow A$
FAD, RP	$(A) + (M) \rightarrow A$ and M
FAD, CM	$(A) + \overline{(M)} \rightarrow A$
FAD, MG	$(A) + \lvert(M)\rvert \rightarrow A$
FAD, UN	$(A) + (M) \rightarrow A$, final coefficient unnormalized
FAD, UR	$(A) + (M) \rightarrow A$, final coefficient unrounded

PROBLEM:

Add a floating point number from address NUM to the A register.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	FAD	NUM	

PROBLEM:

Add the absolute value of a floating point number from address BILL to the A register.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	FAD	BILL	

PROBLEM:

Do a replace floating add from address BOB modified by Index Register 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	FAD,RP	BØB,3	

PROBLEM:

Do a floating add from address TOJO using un-normalized arithmetic.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	FAD,UN	TØJØ	

The FLOATING SUBTRACT Instruction

The FLOATING SUBTRACT instruction is an instruction that subtracts a 48-bit operand at an 18-bit storage address from the contents of A in floating point format. One memory reference is made.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is subtracted from A.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is subtracted from A.

If RP is specified by the programmer, the result of the operation replaces the contents of the storage address besides the A register. In this case two memory references are made.

Normally the coefficient will be normalized. If UN is specified by the programmer, the coefficient will be unnormalized.

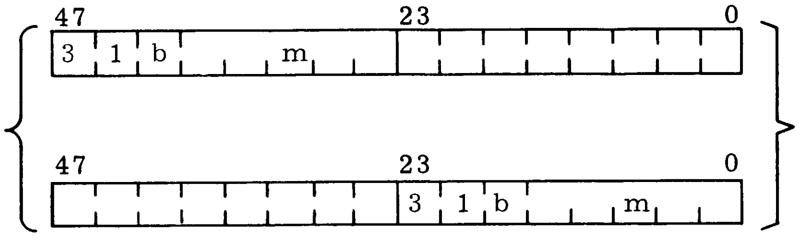
Normally the coefficient is rounded if one-half or greater. If UR is specified by the programmer, rounding will not take place.

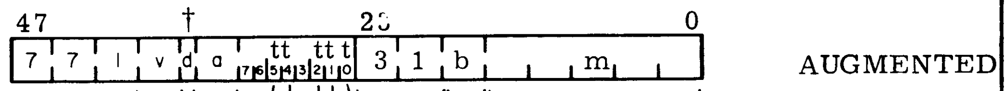
The initial contents of the Q register are always destroyed by this instruction.

FORMAT

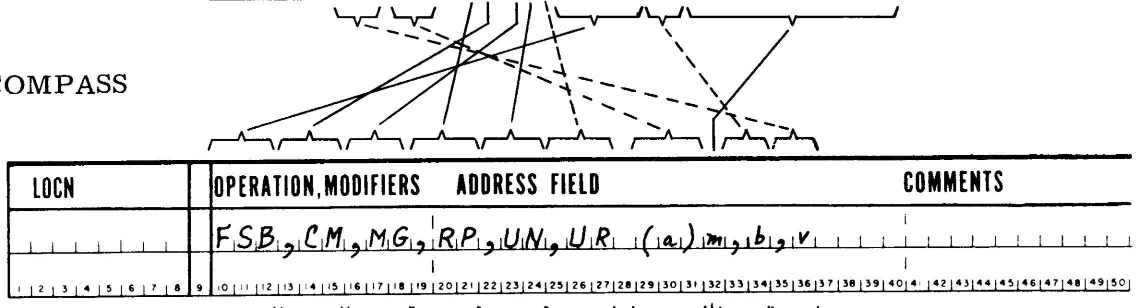
MACHINE

May be an upper or lower instruction





COMPASS


LOCN
OPERATION,MODIFIERS ADDRESS FIELD
COMMENTS
FSB,CM,MG,RP,UN,UR (a) m,b,v

Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, replace result at M

Optional, leave coefficient un-normalized

Optional, leave coefficient un-rounded

Optional, bank term.

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

Optional, second index designator.

Zero or blank, use no index.

1-6, use relative addressing.

7, use indirect addressing

Optional, first index designator.

Zero or omitted, use no index.

1-6, use relative addressing.

7, use indirect addressing.

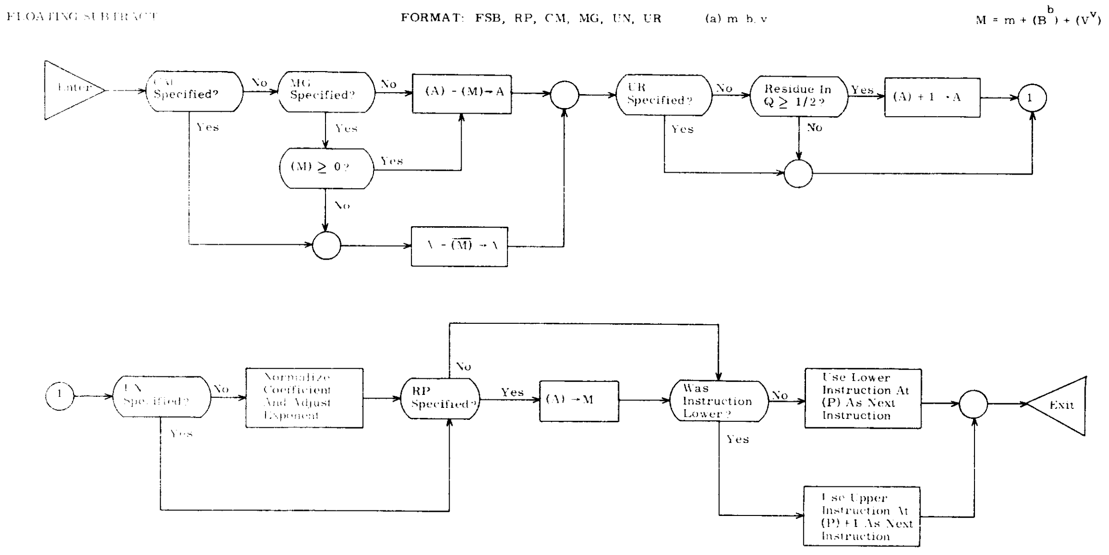
Optional, base address.

Zero if omitted.

*, current program address.

†d bit not programmable

FLOWCHART



DESCRIPTION

FSB	(A) - (M) → A
FSB, RP	(A) - (M) → A and M
FSB, CM	(A) - $\overline{\text{(M)}}$ → A
FSB, MG	(A) - |(M)| → A
FSB, UN	(A) - (M) → A, final coefficient unnormalized
FSB, UR	(A) - (M) → A, final coefficient unrounded

EXAMPLES

PROBLEM:

Subtract a floating point number at address JAKE from the A register

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	FSB	JAKE	

PROBLEM:

Do a Floating Subtract using the address specified in Index Register 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	FSB	,2	

PROBLEM:

Subtract the absolute value of the floating point number at address PILL from the A register.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	FSB,MG	PILL	

PROBLEM:

Do a Floating Subtract using address BOX with unrounded arithmetic.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	FSB,UR	BØX	

The FLOATING MULTIPLY Instruction

The FLOATING MULTIPLY instruction is an instruction that multiplies a 48-bit operand from an 18-bit storage address by the contents of A in floating point format. One memory reference is made.

The 18-bit storage address is composed of a bank term $\underline{a}$ (within parentheses) and a modified base address $\underline{M}$ where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the complement of the operand is multiplied by A.

If MG is specified by the programmer, the magnitude (absolute value) of the operand is multiplied by A.

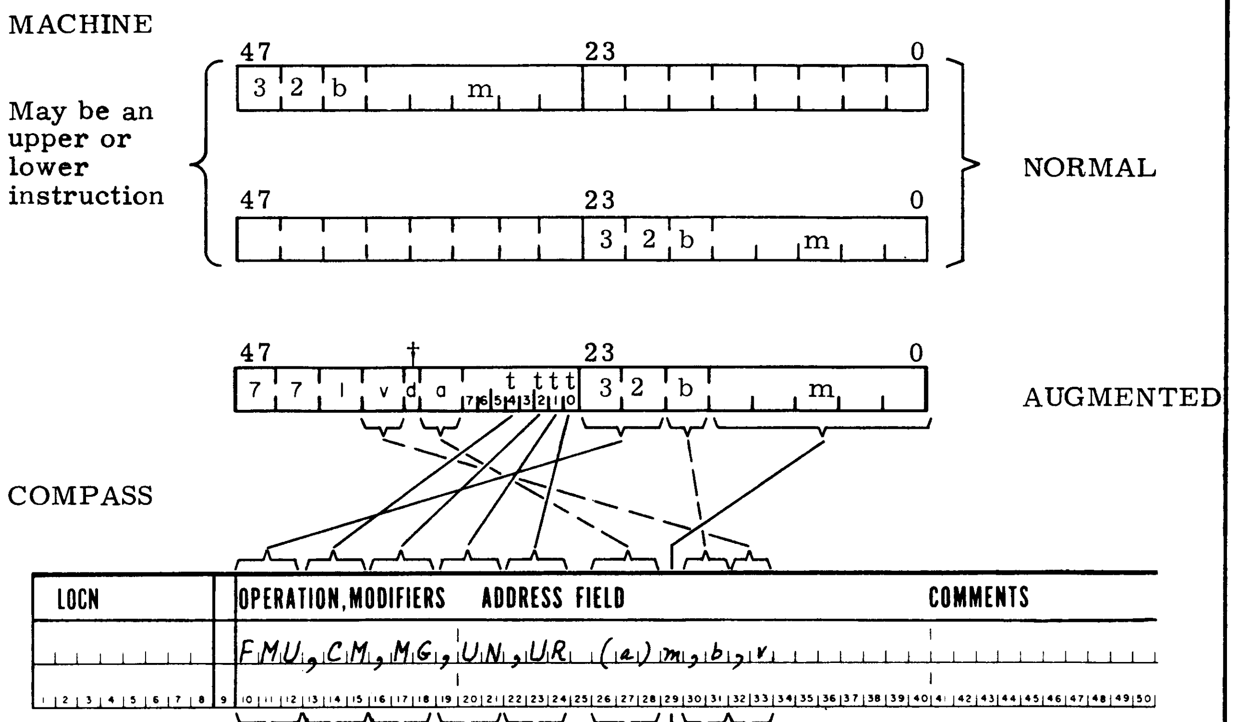
Normally the coefficient will be normalized. If UN is specified by the programmer, the coefficient will be unnormalized.

Normally the coefficient is rounded if one-half or greater. If UR is specified by the programmer, rounding will not take place.

The initial contents of the Q register are always destroyed by this instruction.

FLOATING MULTIPLY | FMU

FORMAT



MACHINE
May be an upper or lower instruction
47
23
0
3 2 b m
NORMAL
47
23
0
†
7 7 l v d a t t t t
3 2 b m
AUGMENTED
COMPASS
LOCN
OPERATION,MODIFIERS ADDRESS FIELD
COMMENTS
FMU,CM,MG,UN,UR (a) m,b,v

Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, leave coefficient un-normalized

Optional, leave coefficient un-rounded

Optional, bank term

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

Optional, second index designator.

Zero or blank, use no index.

1-6, use relative addressing.

7, use indirect addressing.

Optional, first index designator.

Zero or omitted, use no index.

1-6, use relative addressing.

7, use indirect addressing.

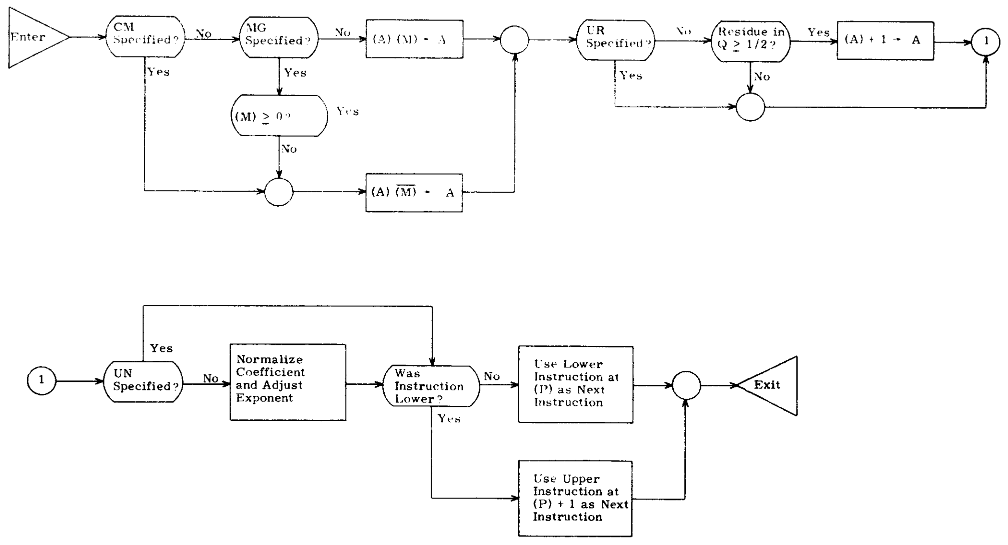
Optional, base address. Zero if omitted. *, current program address.

†d bit not programmable.

FMU

FLOWCHART

FLOATING MULTIPLY FORMAT: FMU, CM, MG, UN, UR (a) m, b, v $M = m + (B^b) + (V^v)$



DESCRIPTION

FMU	(A) (M) → A
FMU, CM	(A) $\overline{(M)}$ → A
FMU, MG	(A) $\lvert(M)\rvert$ → A
FMU, UN	(A) (M) → A, final coefficient unnormalized
FMU, UR	(A) (M) → A, final coefficient unrounded

PROBLEM:

Multiply the contents of A by the number at address SLINK.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		FMU	SLINK	

PROBLEM:

Do a Floating Multiply using the address specified in Index Register 4.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		FMU	,4	

PROBLEM:

Do a Floating Multiply using address BIGGEST with un-normalized arithmetic specified.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		FMU,UN	BIGGEST	

PROBLEM:

Multiply the contents of A by the absolute value of the number at address ONEPRCNT.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		FMU,MG	ØNEPRCNT	

The FLOATING DIVIDE Instruction

The FLOATING DIVIDE instruction is an instruction that divides the contents of A by a 48-bit operand from an 18-bit storage address in floating point format. One memory reference is made.

The 18-bit storage address is composed of a bank term a (within parentheses) and a modified base address M where $M = m + (B^b) + (V^v)$. If the bank term is missing, the current operand bank setting is assumed. The b and v index designators allow for relative addressing. If they are not used, direct addressing is implied.

If CM is specified by the programmer, the contents of A are divided by the complement of the operand at the storage address.

If MG is specified by the programmer, the contents of A are divided by the magnitude (absolute value) of the operand at the storage address.

Normally the coefficient is rounded if one-half or greater. If UR is specified by the programmer, rounding will not take place.

The initial contents are of the Q register are always destroyed by this instruction.

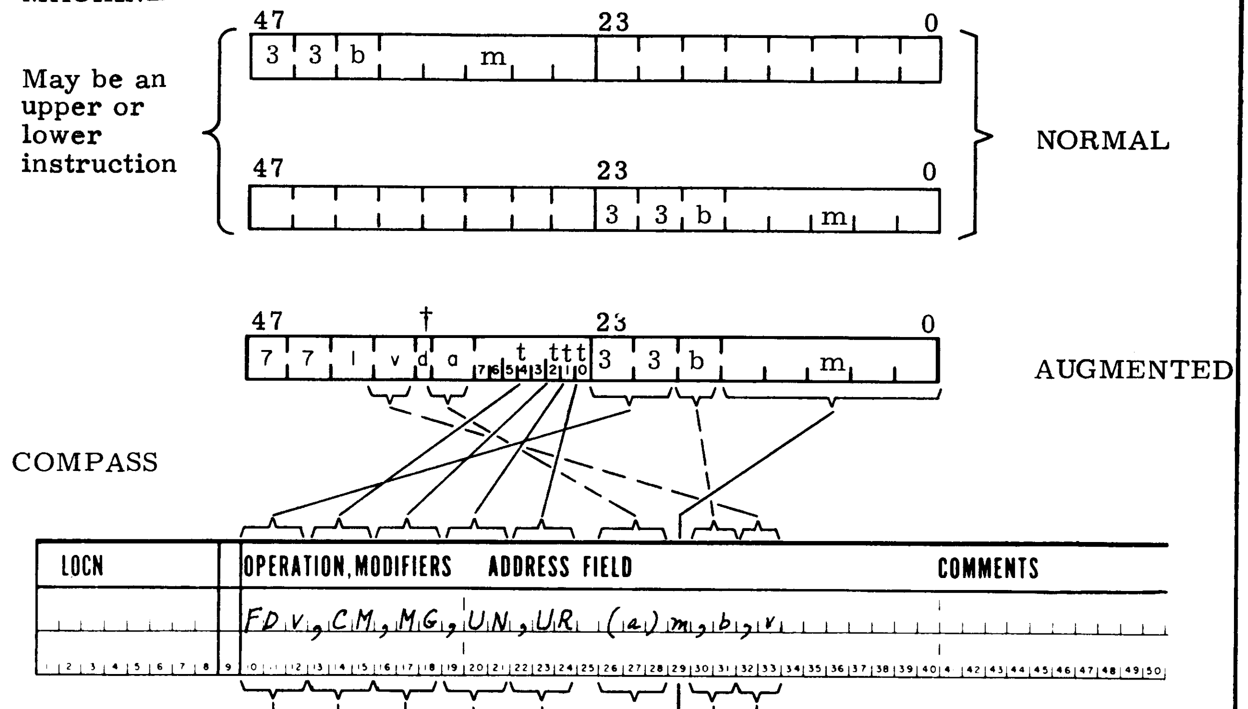
FLOATING DIVIDE

FDV

FORMAT

MACHINE

May be an upper or lower instruction



47
23
0
3 3 b m
NORMAL
3 3 b m
47
†
23
0
7 7 l v d a t 7 6 5 4 3 t t t 2 1 0 3 3 b m
AUGMENTED
COMPASS
LOCN
OPERATION, MODIFIERS ADDRESS FIELD
COMMENTS
FDV, CM, MG, UN, UR (a) m, b, v

Function code

Optional, transmit complement of operand

Optional, transmit magnitude of operand

Optional, leave coefficient un-normalized

Optional, leave coefficient un-rounded

Optional, bank term

0-7, transmits number to Operand Bank Register.

$, transmits bank number in which base address resides to Operand Bank Register.

*, transmits Instruction Bank Register to Operand Bank Register.

Optional, second index designator.

Zero or blank, use no index.

1-6, use relative addressing.

7, use indirect addressing.

Optional, first index designator.

Zero or omitted, use no index.

1-6, use relative addressing.

7, use indirect addressing.

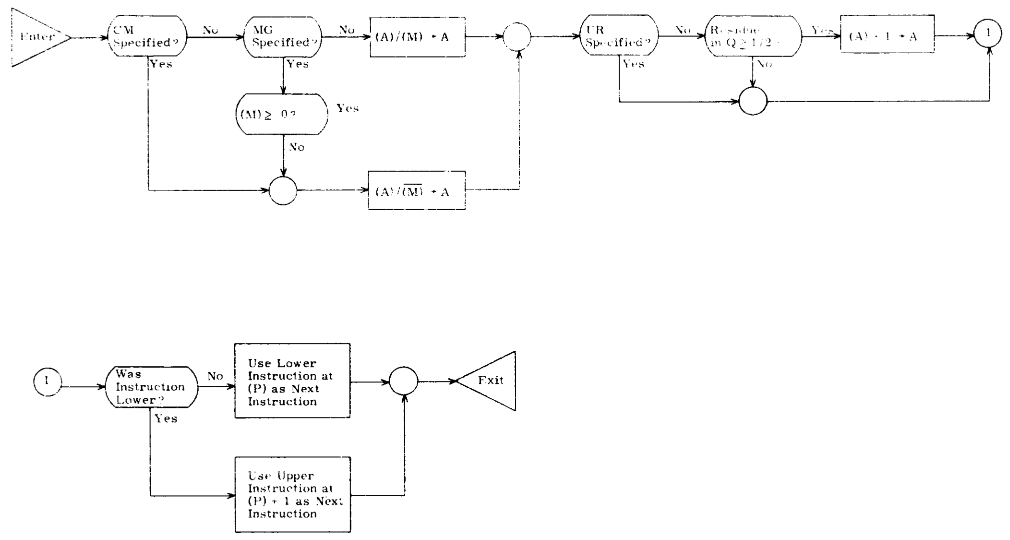
Optional, base address. Zero if omitted.
*, current program address.

†d bit not programmable.

FDV

FLOWCHART

FLOATING DIVIDE   FORMAT: FDV, CM, MG, UR   (a) m, b, v $M = m + (B^b) + (V^v)$



DESCRIPTION

FDV	(A)/(M) → A
FDV, CM	(A)/$\overline{(M)}$ → A
FDV, MG	(A)/$\lvert(M)\rvert$ → A
FDV, UR	(A)/(M), final coefficient unrounded

PROBLEM:

Divide the contents of A by a floating point number at address DENOM.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		FDV	DENØM	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Do a Floating Divide by the absolute value of the number at address PAR modified by Index Register 1.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		FDV	PAR,1	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Do a Floating Divide using address CARC with unrounded arithmetic.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		FDV,UR	CARC	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Do a Floating Divide using the address specified in Index Registers 2 and 3.

SOLUTION:

LOCN		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		FDV	,2,3	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

NEW CONCEPTS OF GROUP 14

Floating point arithmetic offers to most people the most troublesome arithmetic to work with. They can understand integer format. The octal numbers $0 \longrightarrow 013$ and $7 \longrightarrow 751$ in integer format offer no problem. They can understand fractional format The octal numbers $340 \longrightarrow 0$ and $727 \longrightarrow 7$ offer no problem But a mixture of integers and fractions using floating point format loses many people completely. Some people memorize rules. Some people carry a rulebook with them. These methods are cumbersome and often forgetful. This portion of the section deals with floating point format including the operation and use of the floating point instructions.

The method employed to show floating point format is based on the elementary rules of algebra. In order to understand floating point format, one needs to know;

1. the difference of coefficient, base number, and exponent, <u>and</u> how changing signs of the coefficient and exponent changes the value of the number.

2. the method we used in explaining integer and fractional formats.

First of all, let's consider the format. In scientific notation all numbers can be expressed like the following:

$$\pm C \cdot B^{\pm e}$$

This means that some <u>coefficient</u> is multiplied by some <u>base</u> number raised to an <u>exponent</u>. Examples are:

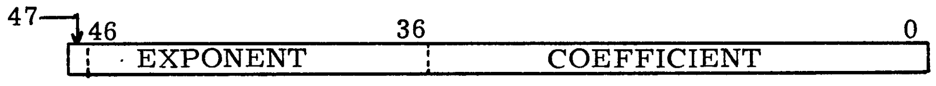
1. $1.5 \cdot 10^{3}$
2. $-3.6 \cdot 10^{5}$
3. $7.1 \cdot 10^{-4}$
4. $-6.8 \cdot 10^{-5}$

Here the base number is always 10 You may notice that the sign of the coefficient and the sign of the exponent are mixed. Let us rearrange these numbers from the largest to the smallest. They would be ordered like this:

1. $1.5 \cdot 10^{3}$ Largest
2. $7.1 \cdot 10^{-4}$
3. $-6.8 \cdot 10^{-5}$
4. $-3.6 \cdot 10^{5}$ Smallest

This we've learned from algebra. Note that the number with the positive coefficient and exponent is the largest and how the number with the negative coefficient and positive exponent is the smallest. Note also that a number above zero has a positive coefficient and a number below zero has a negative coefficient. Note also that the sign of the coefficient determines whether the number is positive or negative

When numbers are expressed on the machine, they are expressed in binary with a base 2. If the base is always 2, there are two quantities which the format must express: the coefficient and the exponent. The engineers for the 3400/3600/3800 systems have formatted a floating point operand as such:



Bits 47 and 35 $\longrightarrow$ 0 represent the coefficient. Bits 46 $\longrightarrow$ 36 represent the exponent.

The coefficient is always represented as a fraction with the point assumed to be between bits 36 and 35. The sign of the fraction is bit 47. Now you ask, "Why have the sign of the coefficient as the uppermost bit?" The answer is that it is the sign of the coefficient which determines whether the complete operand is positive or negative. If it is out in front, it can nicely be checked by the AJP, PL or AJP, MI instruction as is done with the integer and fractional formats.

Now let's return to algebra and see how binary numbers are expressed. We will use octal (easier to read) coefficients and exponents. Consider the numbers:

1. $.5 \cdot 2^{6}$
2. $.7 \cdot 2^{-4}$
3. $-.6 \cdot 2^{-7}$
4. $-.4 \cdot 2^{5}$

These numbers are already ordered from largest to smallest. Now let's say more about these numbers.

1. represents a positive number greater than +1
2. represents a positive fraction
3. represents a negative fraction
4. represents a negative number less than -1

Now how do we match these numbers with how they look in machine format? Just as in integer and fractional formats, we draw a vertical number scale and compare positive forms only.

Algebraic Notation	Machine Notation
2^{1777}(octal) -------------------------------	$3777X \longrightarrow X$
2^{1776} ----------------------------------	$3776X \longrightarrow X$
.	
.	
Positive Numbers Greater Than 1	
.	.
.	.
2^{1} ----------------------------------	$2001X \longrightarrow X$
$2^{0} = +\ 1$ ------------------------------	$2000X \longrightarrow X$
2^{-1}----------------------------------	$1776X \longrightarrow X$
.	.
.	.
Positive Fractions	
.	.
.	.
2^{-1776} -------------------------------	$0001X \longrightarrow X$
2^{-1777}________________________________	$0000X \longrightarrow X$

We found that the largest number expressible in integer format was $37 \longrightarrow 7$. We found that the largest number expressible in fractional format was $37 \longrightarrow 7$. Would you believe the largest number expressible in floating point format is also $37 \longrightarrow 7$? It is because all formats are laid out the same. The smallest number expressible in all three formats is $40 \longrightarrow 0$. Don't think though that you can mix formats and instructions at will during the program. The instruction and format of the operands must agree.

From the scale it is apparent that any positive number greater than 1 will be between 20000 $\longrightarrow$ 0 and 37777 $\longrightarrow$ 7, and any positive fraction will be between 00000 $\longrightarrow$ 0 and 20000 $\longrightarrow$ 0. Examples below point out a few of the number conversions.

Algebraic Notation		Computer Notation
1. $.5 \cdot 2^{6}$	=	200650 $\longrightarrow$ 0
2. $.5 \cdot 2^{-6}$	=	177150 $\longrightarrow$ 0

What about negative numbers? What is the computer notation for $-.5 \cdot 2^{6}$?
If you remember integer and fractional formats, all we did was complement. From number 1 above 200650 $\longrightarrow$ 0 becomes 577127 $\longrightarrow$ 7. This is the answer to the negative number, and of course, the sign shows negative. When the computer adds, subtracts, multiplies, or divides two floating point operands, it will always normalize and round the coefficient. This makes the answer as close as possible. However, the programmer may declare unnormalized or unrounded arithmetic for certain applicatives by specifying UN or UR respectively.

POSSIBLE ERRORS

The possible errors for the four floating point instructions include the following:

Instruction	Fault	
1. FAD	Exponent Overflow	- this occurs when the carry during the coefficient add causes the exponent to extend beyond 1777. 1777 octal is the maximum positive exponent allowed in the format.
	Exponent Underflow	- this occurs when the carry during the coefficient add causes the exponent to extend beyond -1777. -1777 is the maximum negative exponent allowed in the format.
2. FSB	Same faults as above.	
3. FMU	Same faults as above - this occurs when the addition of the exponents	

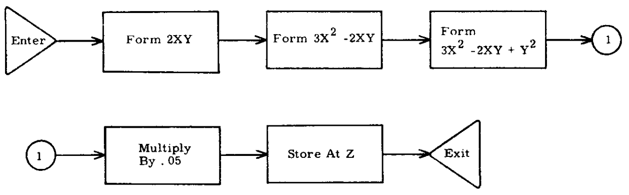
		causes the exponent to exceed 1777 for Overflow or -1777 for Underflow.
4. FDV	Same as above -	occurs when the exponents are subtracted. The result must be between -1777 and 1777.
	Divide Fault -	occurs when the divisor is zero.

One further point on floating point operands is noted. If the computer answer results in zero, the answer will be $0 \longrightarrow 0$. This is so that the AJP, ƵR or AJP, NƵ instruction can determine it as it can for integers or fractions.

Problem 14:

A formula for Z states that $Z = .05(3X^2 - 2XY + Y^2)$. Solve for Z if X and Y are given in floating point format.

Flowchart:



Problem 14 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	EVAL	
	ENTRY	EVAL	
Z	BSS	1	
SAVE	BSS	1	
CØNST	DEC	2.,3.,.05	
EVAL	BSS	1	
	LDA	X	
	FMU	Y	
	FMU	CØNST	2XY
	STA	SAVE	
	LDA	X	
	FMU	X	
	FMU	CØNST+1	3X**2
	FSB	SAVE	3X##2-2XY+Y**2
	STA	SAVE	
	LDA	Y	
	FMU	Y	Y**2
	FAD	SAVE	3X**2-2XY+Y**2
	FMU	CØNST+2	FINAL EVALUATIØN
	STA	Z	
	SLJ	EVAL	
	END		

Somewhere within this subprogram would also be included the symbols X and Y in the location field with a declaration of the prestored data or area reserved.

Student Problem 14A:

Solve for S if S = $({}^{p}/q + 1)({}^{r}/2 - 3)$ and p, q, and r are given in floating point format.

Flowchart:

Problem 14A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

GROUP 15

SEARCH

GROUP 15

SEARCH

1.	Equality Search	EQS
2.	Threshold Search	THS
3.	Masked Equality Search	MEQ
4.	Masked Threshold Search	MTH
5.	Search Equality	SEQU
6.	Search Masked Equality	SMEQ
7.	Search Within Limits	SEWL
8.	Search Magnitude Within Limits	SMWL
9.	Locate List Element Upper	LSTU
10.	Locate List Element Lower	LSTL
11.	Scan	SCAN

This group of instructions searches one or more storage words from memory until it has made a find, or until the specified list has been completely searched. One memory reference is required for each address searched.

The first four instructions use an index register for the number of addresses to be searched. If a find is made, a full exit is taken. If no find is made, a half exit is taken. Because of this principle, these four instructions are upper instructions, and the computer assembler will force these instructions to the upper position. Therefore the programmer should not modify these instructions with modifiers, bank terms, or second index designators, since this overrides the assembler and makes the use of these instructions ineffective.

The next four instructions make use of Index Registers 1, 2, and 3. By using Index Register 3 as an incrementer, some storage addresses can be skipped.

The next two instructions locate an element within a list. This list must have been previously set up by the programmer, and now an element may be specified by its order in the list.

The last instruction searches bytes of memory words rather than complete memory words.

The EQUALITY SEARCH Instruction

The EQUALITY SEARCH instruction is an instruction that searches memory for an operand equal to the contents of A. One memory reference results for each storage address searched.

The contents of the index register specified by b determine the number of storage addresses searched. For every address searched the contents of the index register are decremented by 1. If no index designator is specified, one word is searched at address m.

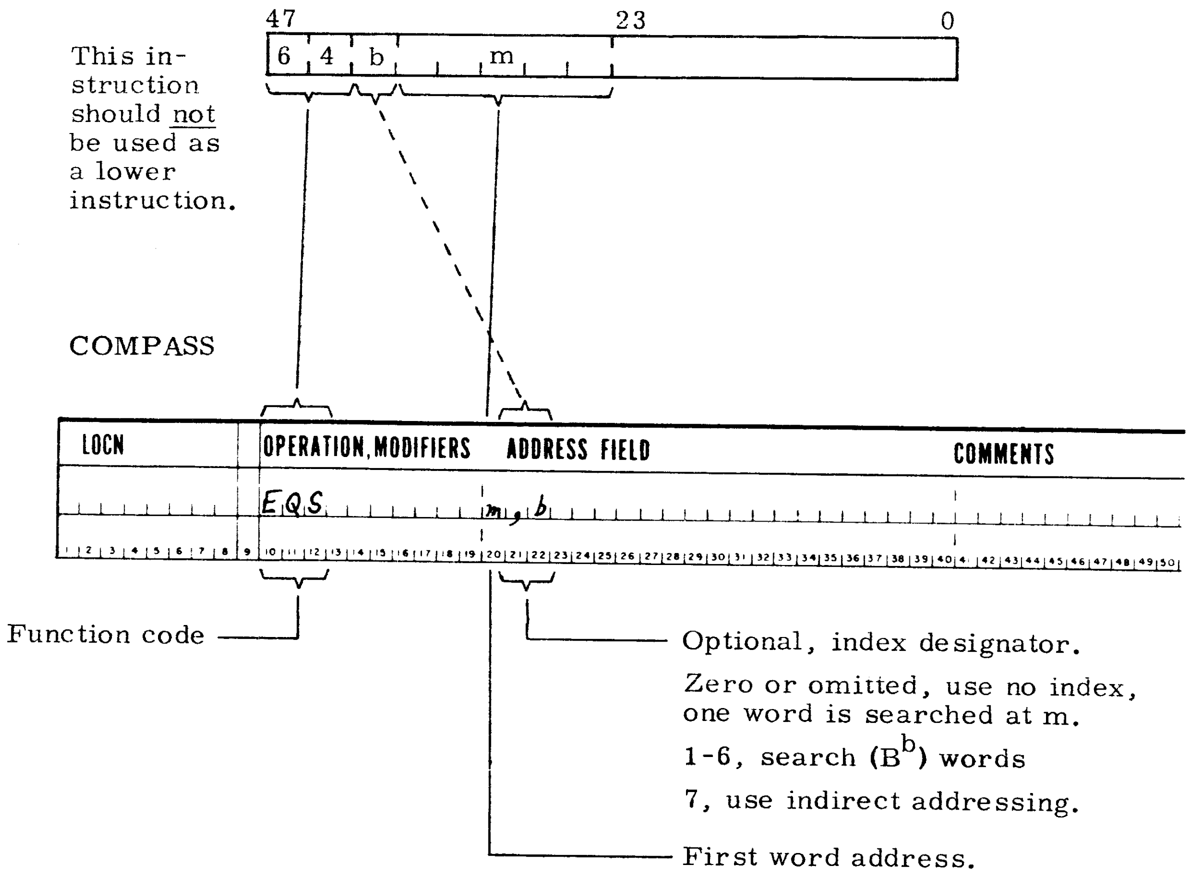
If an operand in memory satisfies the search, a full exit is taken. The address of the operand that satisfied the search will be given by m + (B^b). If no operand satisfies the search, a half exit is taken with the contents of the index register going to zero.

FORMAT

MACHINE

This instruction should <u>not</u> be used as a lower instruction.

COMPASS


47
23
0
6 4 b m
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
EQS
m, b
Function code
Optional, index designator.
Zero or omitted, use no index, one word is searched at m.
1-6, search (B^b) words
7, use indirect addressing.
First word address.

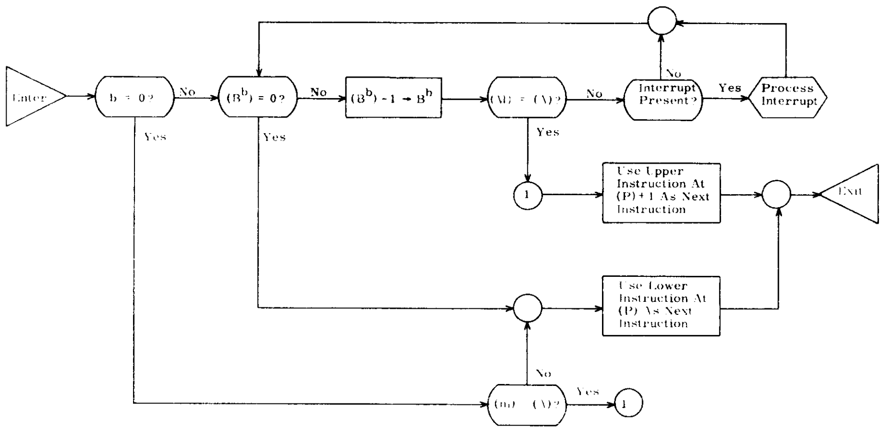
EQS

FLOWCHART

EQUALITY SEARCH*

FORMAT: EQS m, b

$M = m + (B^b)$


Enter
b = 0?
No
Yes
$(B^b) = 0?$
No
Yes
$(B^b) - 1 \rightarrow B^b$
(M) = (A)?
No
Yes
Interrupt Present?
No
Yes
Process Interrupt
1
Use Upper Instruction At (P) + 1 As Next Instruction
Exit
Use Lower Instruction At (P) As Next Instruction
No
(m) = (A)?
Yes
1

*Forced upper by the assembler

DESCRIPTION

EQS

1) (B^b) is number of addresses searched
2) m is first word address
3) Search for (M) = (A)
4) Half exit if search is not satisfied, otherwise full exit

E
X
A
M
P
L
E
S

PROBLEM:

Examine the quantity at address KLUDGE for being equal to the contents of A.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EQS	KLUDGE	

PROBLEM: Given: $(B^1) = 100$

Search a table of 100 locations starting at address TAB for some quantity equal to A.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EQS	TAB,1	

PROBLEM: Given: $(B^2) = 100$ octal

Search a table of 100 (octal) values starting at address MAT for some quantity equal to A.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EQS	MAT,2	

PROBLEM:

Examine the quantity at address SMUT for being equal to the contents of A.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EQS	SMUT	

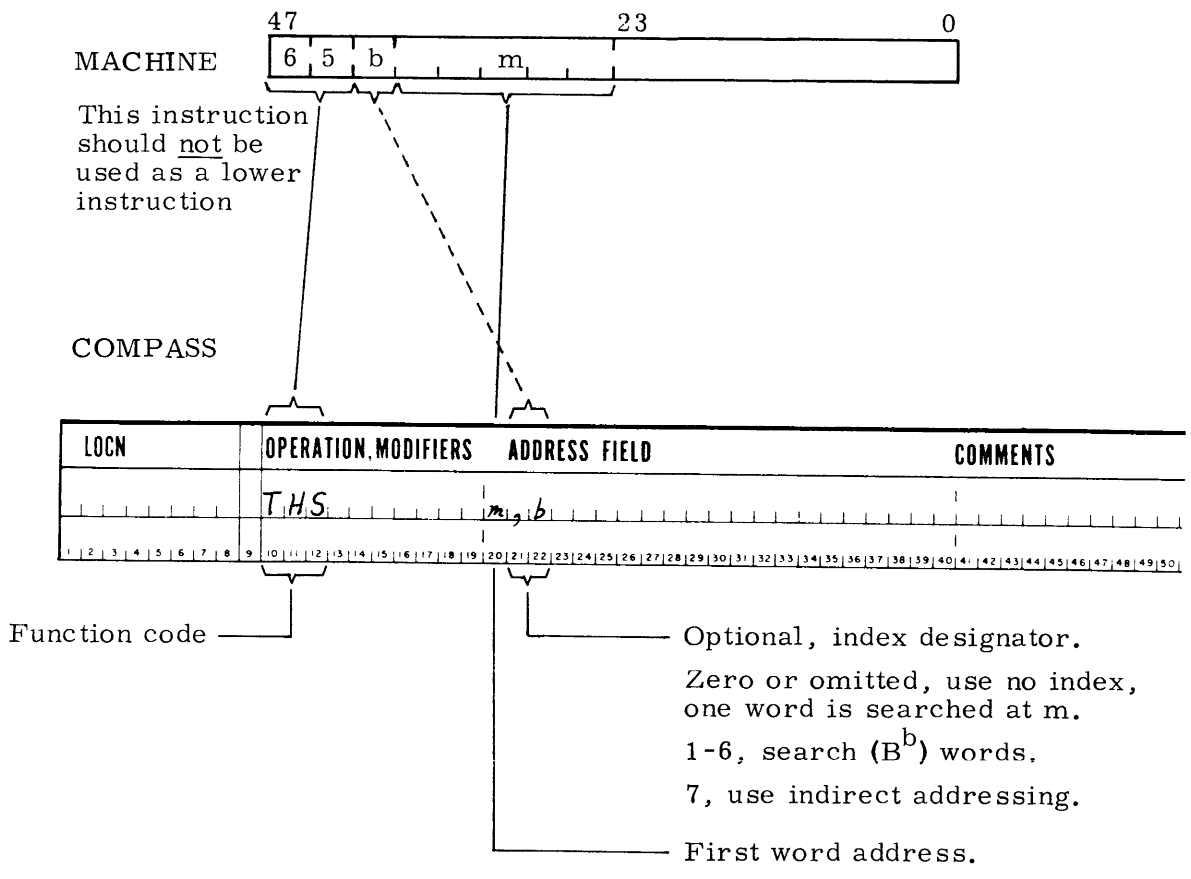
The THRESHOLD SEARCH Instruction

The THRESHOLD SEARCH instruction is an instruction that searches memory for an operand greater than the contents of A. One memory reference results for each storage address searched.

The contents of the index register specified by b determine the number of storage addresses searched. For every address searched the contents of the index register are decremented by 1. If no index designator is specified, one word is searched at address m.

If an operand in memory satisfies the search, a full exit is taken. The address of the operand that satisfied the search will be given by $m + (B^b)$. If no operand satisfies the search, a half exit is taken with the contents of the index register going to zero.

FORMAT

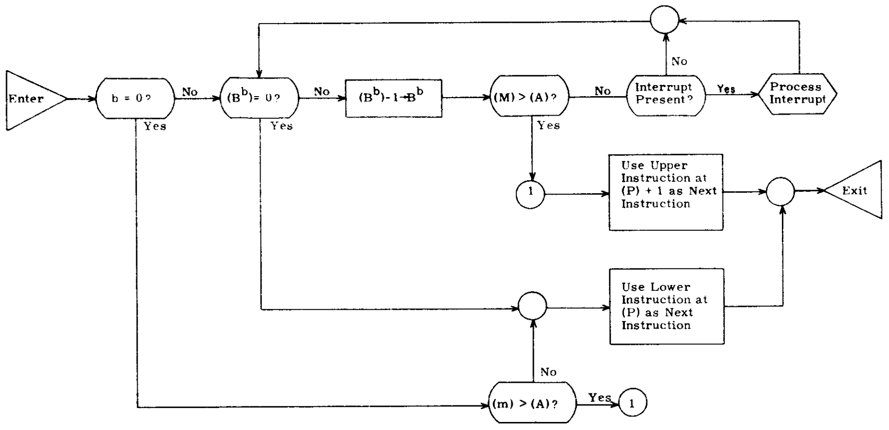


47
23
0
MACHINE
6 5 b m
This instruction should not be used as a lower instruction
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
THS m, b
Function code
Optional, index designator.
Zero or omitted, use no index, one word is searched at m.
1-6, search (B^b) words.
7, use indirect addressing.
First word address.

THS

F
L
O
W
C
H
A
R
T

THRESHOLD SEARCH* FORMAT: THS m, b $M = m + (B^b)$


Enter
b = 0?
No
Yes
(B^b) = 0?
No
Yes
$(B^b)-1 \rightarrow B^b$
(M) > (A)?
No
Yes
Interrupt Present?
No
Yes
Process Interrupt
1
Use Upper Instruction at (P) + 1 as Next Instruction
Exit
Use Lower Instruction at (P) as Next Instruction
No
(m) > (A)?
Yes
1

*Forced upper by the assembler

D
E
S
C
R
I
P
T
I
O
N

THS

1) (B^b) is number of addresses searched
2) m is first word address
3) Search for (M) > (A)
4) Half exit if search is not satisfied, otherwise full exit

PROBLEM:

Examine the quantity at address SMALLEST for being greater than the contents of A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	THS	SMALLEST	

PROBLEM: Given: $(B^3) = 5$

Search a Table of five values starting at address AREA for some quantity greater than A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	THS	AREA,3	

PROBLEM: Given: $(B^4) = 100$

Search a table of 100 values starting at address TAR for some quantity greater than A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	THS	TAR,4	

PROBLEM: Given: $(B^5) = 200$

Search a table of 200 values starting at address PATCH for some quantity greater than A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	THS	PATCH,5	

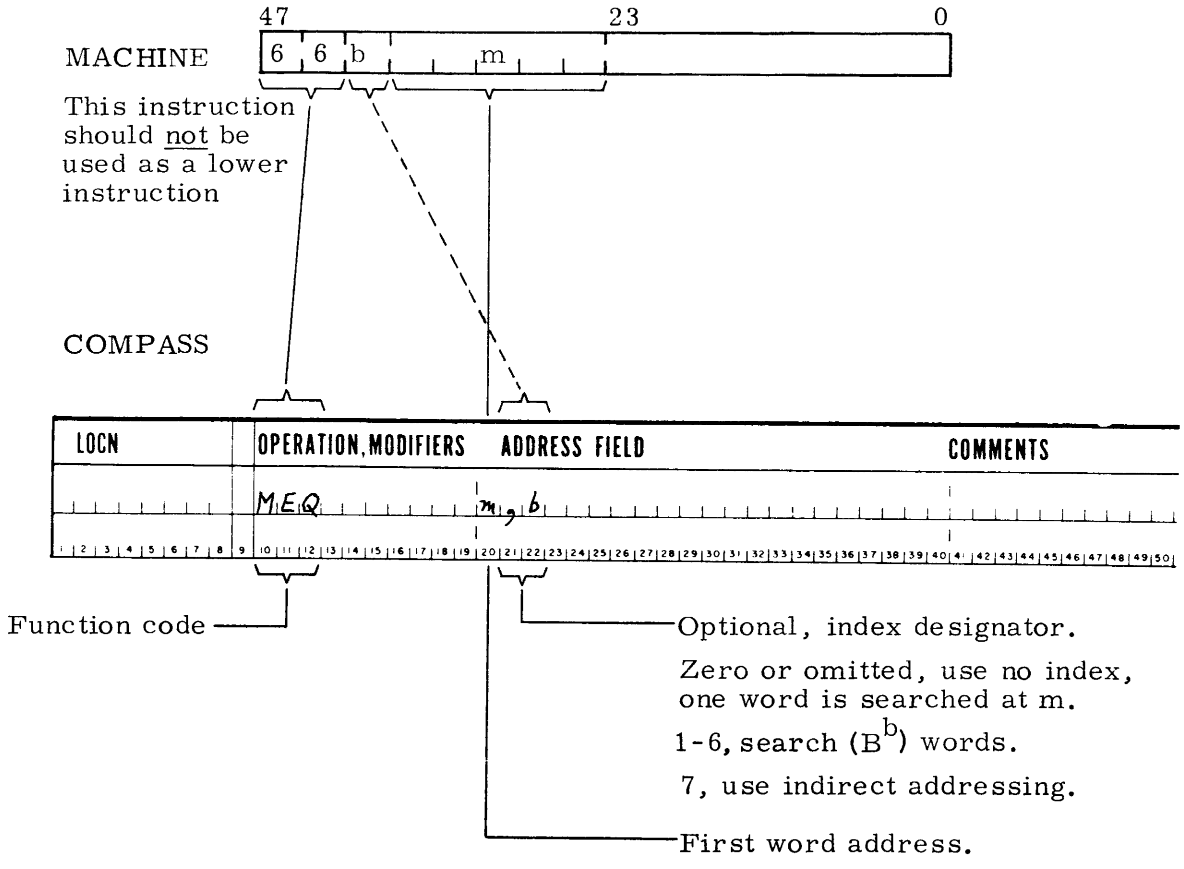
The MASKED EQUALITY SEARCH Instruction

The MASKED EQUALITY SEARCH instruction is an instruction that searches memory for an operand masked with the contents of Q that is equal to the contents of A. One memory reference results for each storage address searched.

The contents of the index register specified by b determine the number of storage addresses searched. For every address searched the contents of the index register are decremented by 1. If no index designator is specified, one word is searched at address m.

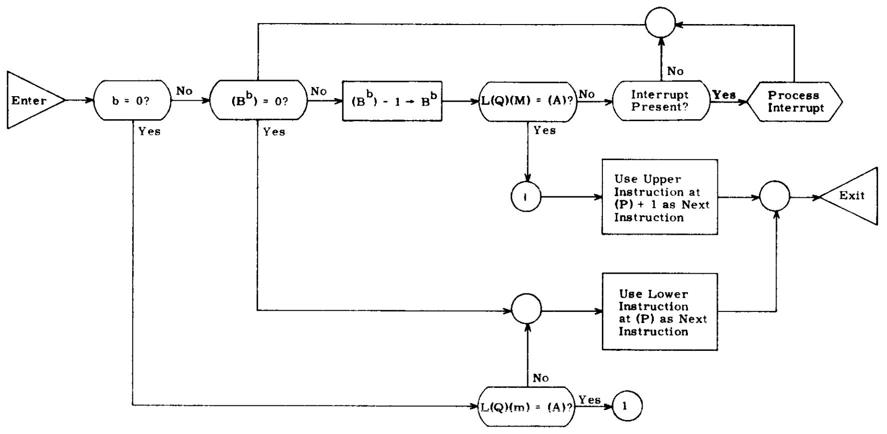
If an operand in memory satisfies the search, a full exit is taken. The address of the operand that satisfied the search will be given by m + (B^b). If no operand satisfies the search, a half exit is taken with the contents of the index register going to zero.

FORMAT


MACHINE
47
23
0
6
6
b
m
This instruction should not be used as a lower instruction
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
MEQ
m, b
Function code
Optional, index designator.
Zero or omitted, use no index, one word is searched at m.
1-6, search (B^b) words.
7, use indirect addressing.
First word address.

FLOWCHART

MASKED EQUALITY SEARCH* FORMAT: MEQ m, b $M = m + (B^b)$


Enter
b = 0?
No
Yes
$(B^b) = 0?$
No
Yes
$(B^b) - 1 \rightarrow B^b$
L(Q)(M) = (A)?
No
Yes
Interrupt Present?
No
Yes
Process Interrupt
1
Use Upper Instruction at (P) + 1 as Next Instruction
Exit
Use Lower Instruction at (P) as Next Instruction
L(Q)(m) = (A)?
No
Yes
1

*Forced upper by the assembler

DESCRIPTION

MEQ

1) (B^b) is number of addresses searched
2) m is first word address
3) Search for L (Q) (M) = (A)
4) Half exit if search is not satisfied, otherwise full exit

PROBLEM: Given: (Q) = 77 octal

Examine the lowest six bits at address TAB for being equal to the contents of A.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		MEQ	TAB	

PROBLEM: Given: (Q) = 770——0 octal
(B^1) = 100

Search a table of 100 values starting at address ENT for some upper 6 bits equal to the contents of A.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		MEQ	ENT,1	

PROBLEM: Given: (Q) = 77777777 octal
(B^2) = 10

Search a table of 10 values starting at address PETE for some lower half portion equal to A.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		MEQ	PETE,2	

PROBLEM: Given: (Q) = 777777770——0 octal

Examine the upper half of the value at address PAL for being equal to A.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		MEQ	PAL	

The MASKED THRESHOLD SEARCH Instruction

The MASKED THRESHOLD SEARCH instruction is an instruction that searches memory for an operand masked with the contents of Q that is greater than the contents of A. One memory reference results for each storage address searched.

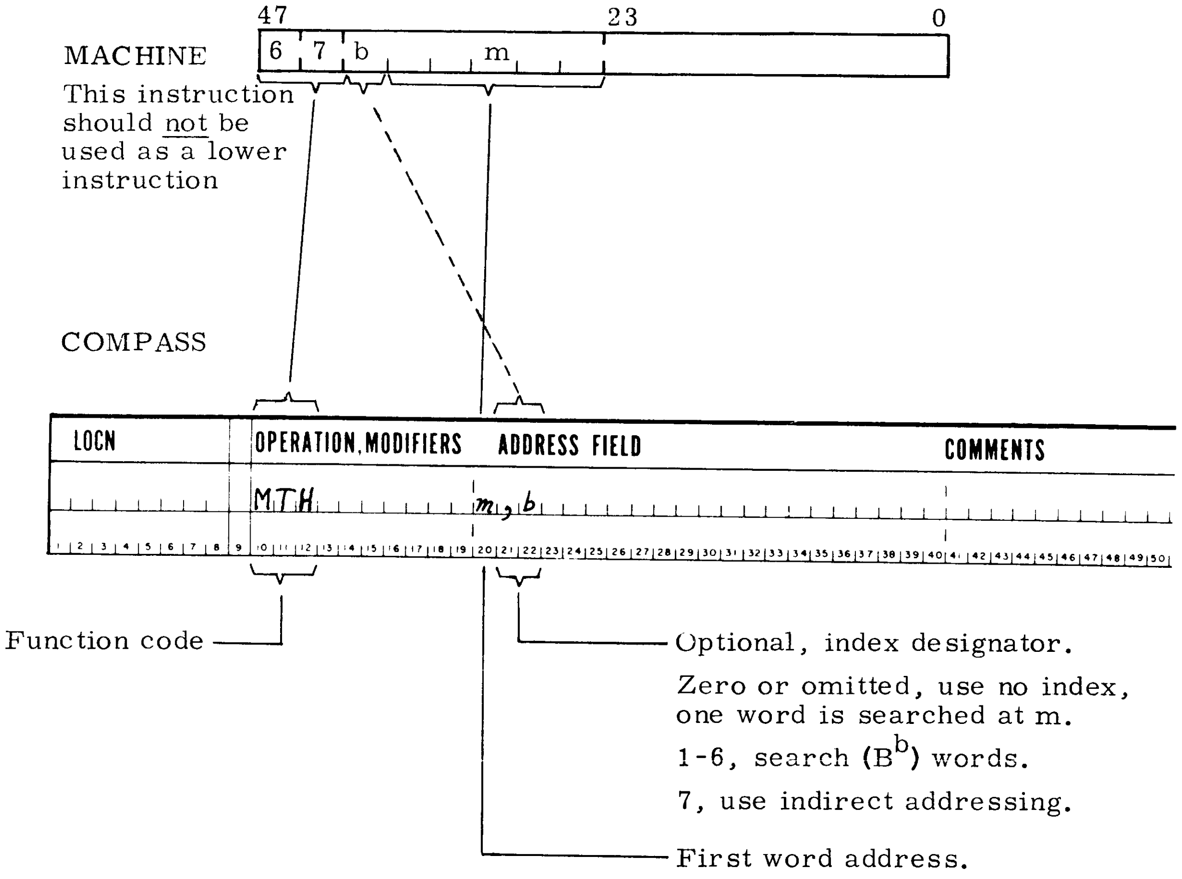
The contents of the index register specified by b determine the number of storage addresses searched. For every address searched the contents of the index register are decremented by 1. If no index designator is specified, one word is searched at address m.

If an operand in memory satisfies the search, a full exit is taken. The address of the operand that satisfied the search will be given by $m + (B^b)$. If no operand satisfies the search, a half exit is taken with the contents of the index register going to zero.

MASKED THRESHOLD SEARCH

MTH

FORMAT



MACHINE
47
23
0
6
7
b
m
This instruction should not be used as a lower instruction
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
MTH
m, b
Function code
Optional, index designator.
Zero or omitted, use no index, one word is searched at m.
1-6, search (B^b) words.
7, use indirect addressing.
First word address.

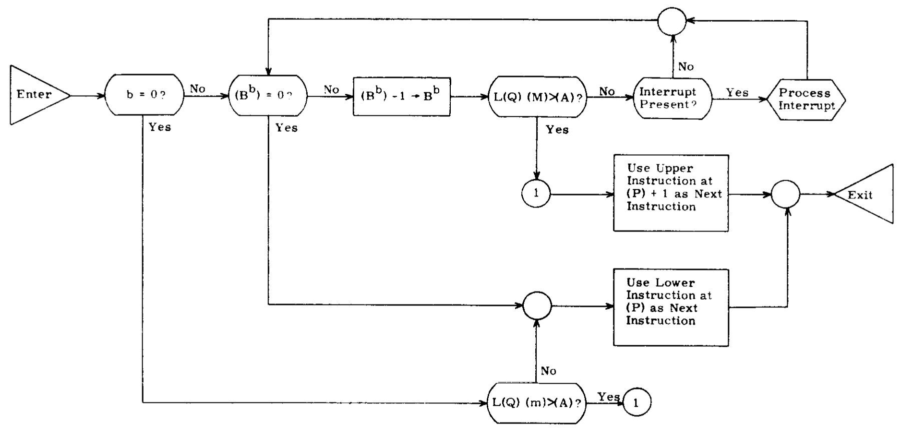
MTH

F L O W C H A R T

MASKED THRESHOLD SEARCH*

FORMAT: MTH m, b

$M = m + (B^b)$


Enter
b = 0?
No
Yes
$(B^b) = 0?$
No
Yes
$(B^b) - 1 \rightarrow B^b$
L(Q) (M)>(A)?
No
Yes
Interrupt Present?
No
Yes
Process Interrupt
1
Use Upper Instruction at (P) + 1 as Next Instruction
Exit
Use Lower Instruction at (P) as Next Instruction
No
L(Q) (m)>(A)?
Yes
1

*Forced upper by the assembler

D E S C R I P T I O N

MTH

1) (B^b) is number of addresses searched
2) m is first word address
3) Search for L (Q) (M) > (A)
4) Half exit if search not satisfied, otherwise full exit

PROBLEM: Given: (Q) = 77 octal

Examine the lowest six bits at address TUB for being greater than the contents of A.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		MTH	TUB	

PROBLEM: Given: (Q) = 770——0 octal

Examine the highest six bits at address TANK for being greater than the contents of A.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		MTH	TANK	

PROBLEM: Given: (Q) = 77 octal, (B^4) = 10

Search a table of 10 values starting at address TOP to see if the lowest 6 bits of each value is greater than A.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		MTH	TØP,4	

PROBLEM: Given: (Q) = 000777770——0 octal, (B^5) = 20

Search a table of 20 values starting at address FORM to see if the upper address portion of each value is greater than A.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		MTH	FØRM,5	

The SEARCH EQUALITY Instruction

The SEARCH EQUALITY instruction is an instruction that searches memory for an operand that is equal to the contents of A. One memory reference results for each storage address searched.

Before this instruction is executed, three index registers are defined and must be set up as follows:

1. (B^1) - number of storage words to search
2. (B^2) - modifier for address m
3. (B^3) - incrementer for (B^2)

The incrementer is set for 1 for searching sequential words, 2 for searching every other word, etc.

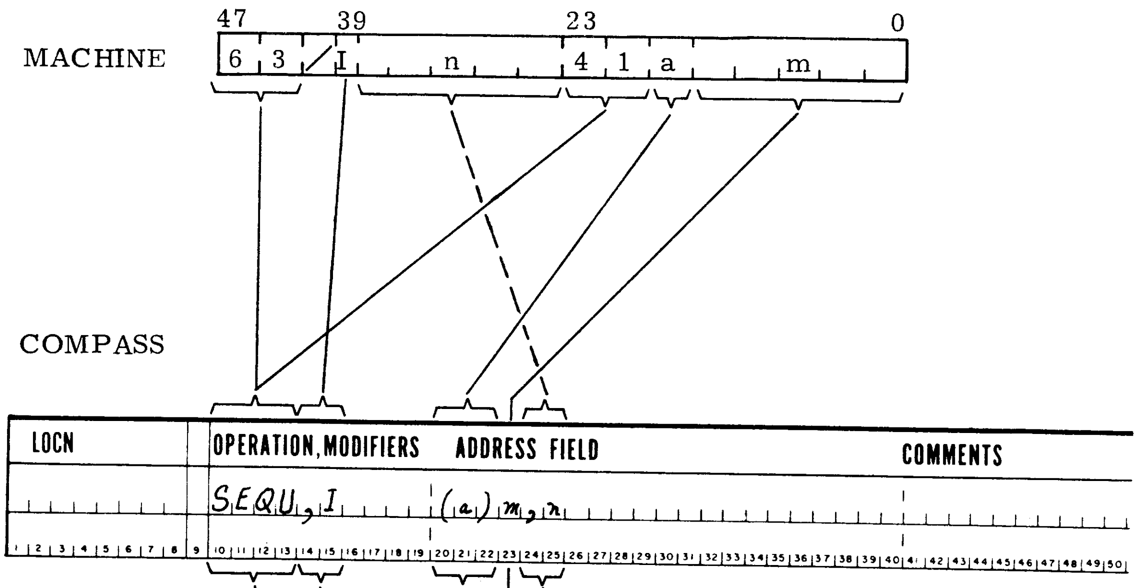
The bank term $\underline{a}$ allows the programmer to search in any bank starting with address $\underline{M}$ where $M = m + (B^2)$.

If the search is satisfied, a full exit is taken. The address of the operand that satisfied the search will be given by $m + (B^2) - (B^3)$.

If the search is not satisfied, program control will transfer to address $\underline{n}$ with the contents of Index Register 1 going to zero.

If I is specified by the programmer, the address searched is not M. Rather, the lowest 18 bits at address M become the address searched.

FORMAT



Function code

Optional modifier
Indirect addressing

Jump address if search is not satisfied.

Optional, base address. Zero if omitted.

Optional, bank term.

Current operand bank setting if omitted.

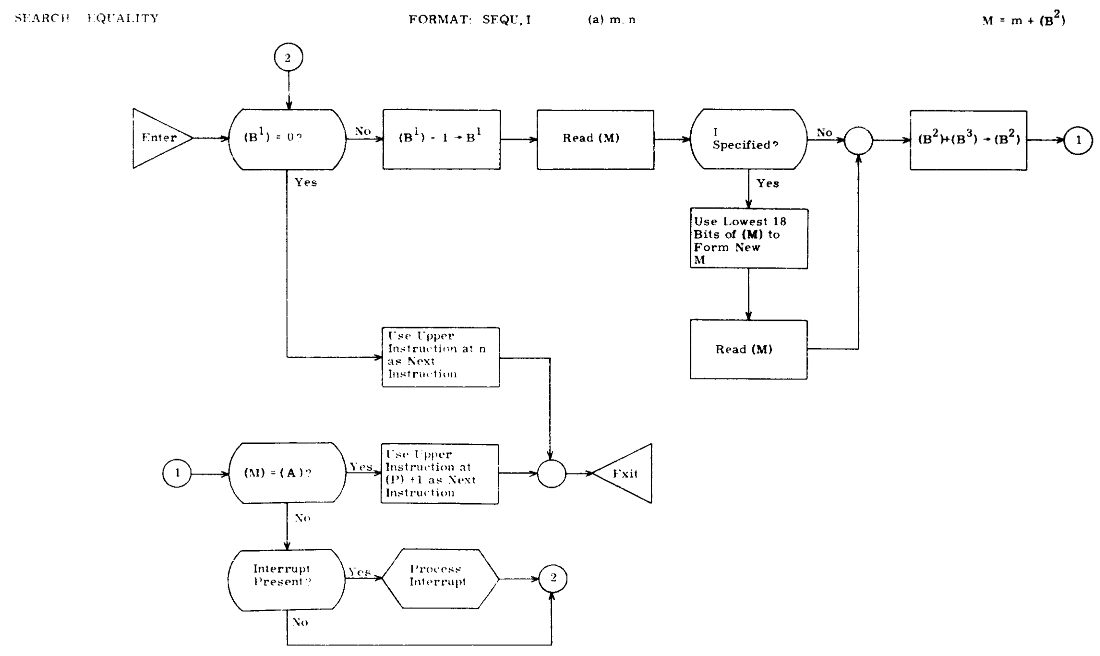
0-7, actual bank.

$, bank in which base address resides.

*, bank in which SEQU resides.

SEQU

FLOWCHART



DESCRIPTION

SEQU

1) (B^1) is number of addresses searched
2) m + (B^2) is first word address
3) Addresses searched in increments of (B^3)
4) Search for (M) = (A)
5) Jump to n if search not satisfied, otherwise full exit

SEQU, I

Search indirectly

E
X
A
M
P
L
E
S

PROBLEM: Given: $(B^1) = 100$, $(B^2) = 0$, $(B^3) = 1$

Search a table of 100 values starting at address TAB in Bank 1 for some value equal to the contents of A. If the search is satisfied, continue the program. If not, jump to address NOFIND.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SEQU	(1)TAB,NØFIND	

PROBLEM: Given: $(B^1) = 100$, $(B^2) = 0$, $(B^3) = 2$

Search a table of 200 values by searching only every other value (actually 100 values). The first address is MORT in Bank 2. If there is some value equal to A, continue the program. If not, jump to address NIL.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SEQU	(2)MØRT,NIL	

PROBLEM: Given: $(B^1) = 10$, $(B^2) = 0$, $(B^3) = 1$

Search a table of 10 values starting at address PETE in the bank where PETE resides for some value equal to A. If the search is satisfied, continue the program. If not, jump to address NIX.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SEQU	(#)PETE,NIX	

PROBLEM: Given: $(B^1) = 100$, $(B^3) = 1$

Search a table of 100 values starting at the address specified in Index Register 2 in Bank 3 for some value equal to A. If the search is satisfied, continue the program. If not, jump to address RECALL

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SEQU	(3),RECALL	

The SEARCH MASKED EQUALITY Instruction

The SEARCH MASKED EQUALITY instruction is an instruction that searches memory for an operand that, masked with the contents of Q, is equal to the contents of A. One memory reference results for each storage address searched.

Before this instruction is executed, three index registers are defined and must be set up as follows:

1. (B^1) - number of storage words to search
2. (B^2) - modifier for address m
3. (B^3) - incrementer for (B^2)

The incrementer is set for 1 for searching sequential words, 2 for searching every other word, etc.

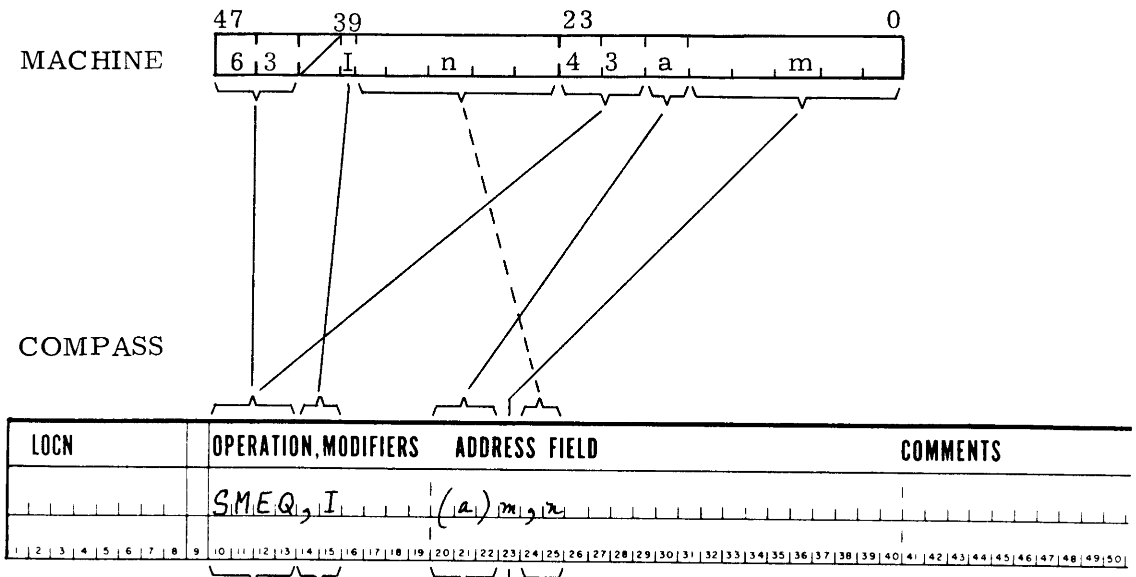
The bank term $\underline{a}$ allows the programmer to search in any bank starting with address $\underline{M}$ where $M = m + (B^2)$.

If the search is satisfied, a full exit is taken. The address of the operand that satisfied the search will be given by $m + (B^2) - (B^3)$.

If the search is not satisfied, program control will transfer to address $\underline{n}$ with the contents of Index Register 1 going to zero.

If I is specified by the programmer, the address searched is not M. Rather, the lowest 18 bits at address M become the address searched.

FORMAT



Function code

Optional modifier
Indirect addressing

Jump address if search is not satisfied.

Optional, base address. Zero if omitted.

Optional, bank term.

Current operand bank setting if omitted.

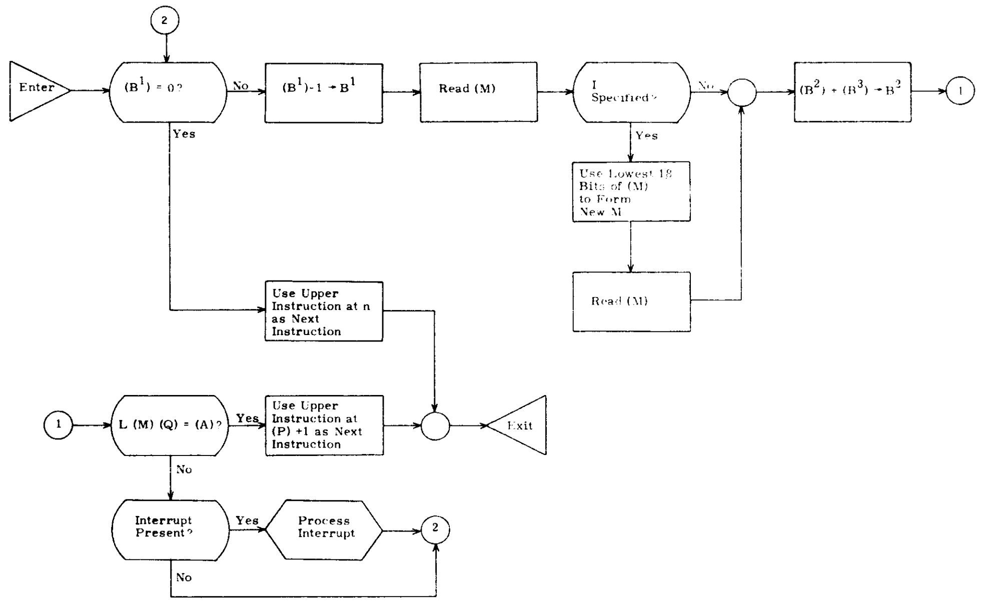
0-7, actual bank.

$, bank in which base address resides.

*, bank in which SMEQ resides.

FLOWCHART

SEARCH MASKED EQUALITY   FORMAT: SMEQ, I   (a) m, n $M = m + (B^2)$



2
Enter
$(B^1) = 0$?
No
$(B^1)-1 \rightarrow B^1$
Read (M)
I Specified?
No
$(B^2) + (B^3) \rightarrow B^2$
1
Yes
Use Lowest 18 Bits of (M) to Form New M
Read (M)
Yes
Use Upper Instruction at n as Next Instruction
1
L (M) (Q) = (A)?
Yes
Use Upper Instruction at (P) +1 as Next Instruction
Exit
No
Interrupt Present?
Yes
Process Interrupt
2
No

DESCRIPTION

SMEQ	1) (B^1) is number of addresses searched
	2) $m + (B^2)$ is first word address
	3) Addresses searched in increments of (B^3)
	4) Search for L (M) (Q) = (A)
	5) Jump to n if search not satisfied, otherwise full exit
SMEQ, I	Search indirectly

PROBLEM: Given: $(B^1) = 100$, $(B^2) = 0$, $(B^3) = 1$, $(Q) = 77$ octal

Search a table of 100 values starting at address PACK in Bank 1 for some value whose lowest six bits are equal to the contents of A. If the search is satisfied, continue the program. If not, jump to address ERROR.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SMEQ	(1)PACK,ERRØR	

PROBLEM: Given: $(B^1) = 100$, $(B^2) = 0$, $(B^3) = 2$, $(Q) = 770$——0

Search a table of 200 values by searching only every other value (actually 100 values). The first address is SORT in Bank 2. If there is some value whose highest six bits are equal to A, continue the program. If not, jump to address PIL.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SMEQ	(2)SØRT,PIL	

PROBLEM: Given: $(B^1) = 10$, $(B^2) = 0$, $(B^3) = 1$, $(Q) = 77777$ octal

Search a table of 10 values starting at address SETE in the bank where SETE resides for some value whose lowest 15 bits are equal to A. If the search is satisfied, continue the program. If not, jump to address FIX.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SMEQ	($)SETE,FIX	

PROBLEM: Given: $(B^1) = 100$, $(B^3) = 1$, $(Q) = 777770$——0

Search a table of 100 values starting at the address specified in Index Register 2 in Bank 3 for some value whose highest 15 bits are equal to A. If the search is satisfied, continue the program. If not, jump to address RECALL.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SMEQ	(3),RECALL	

The SEARCH WITHIN LIMITS Instruction

The SEARCH WITHIN LIMITS instruction is an instruction that searches memory for an operand that is greater than the contents of Q, but less than or equal to the contents of A. One memory reference results for each storage address searched.

Before this instruction is executed, three index registers are defined and must be set up as follows:.

1. (B^1) - number of storage words to search
2. (B^2) - modifier for address m
3. (B^3) - incrementer for (B^2)

The incrementer is set for 1 for searching sequential words, 2 for searching every other word, etc.

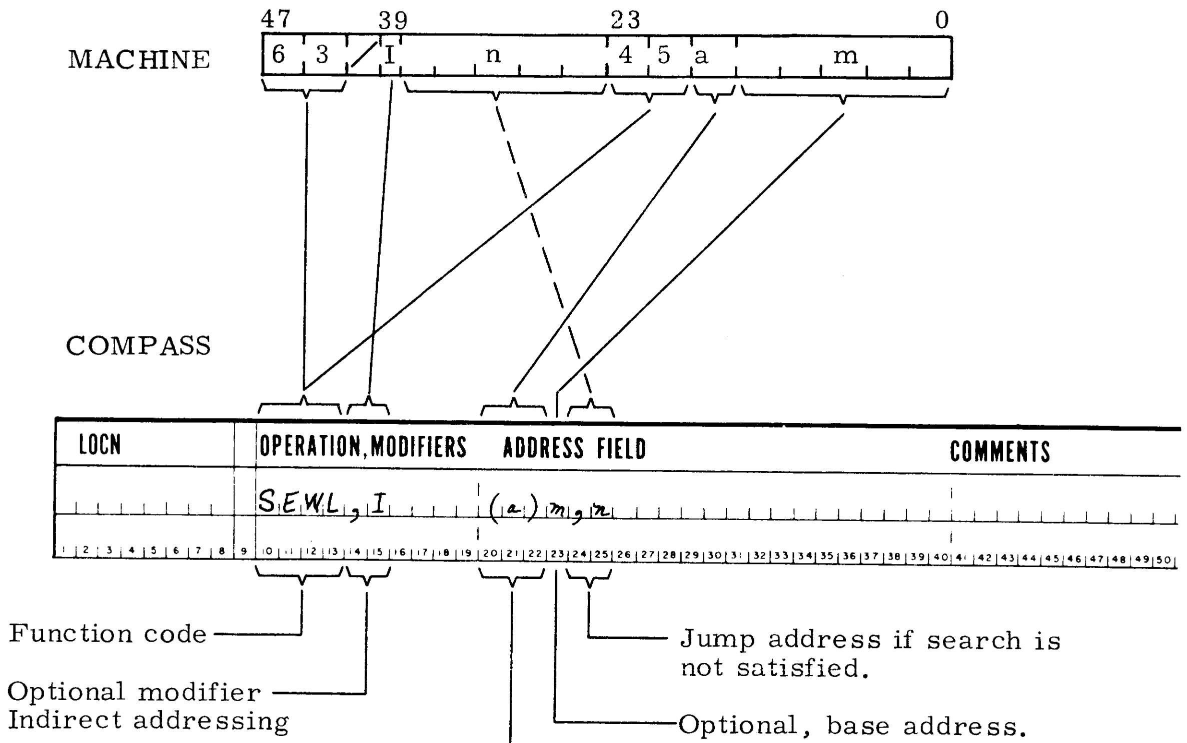
The bank term a allows the programmer to search in any bank starting with address M where $M = m + (B^2)$.

If the search is satisfied, a full exit is taken. The address of the operand that satisfies the search will be given by $m + (B^2) - (B^3)$.

If the search is not satisfied, program control will transfer to address n with the contents of Index Register 1 going to zero.

If I is specified by the programmer, the address searched is not M. Rather, the lowest 18 bits at address M become the address searched.

F O R M A T



Current operand bank setting if omitted.

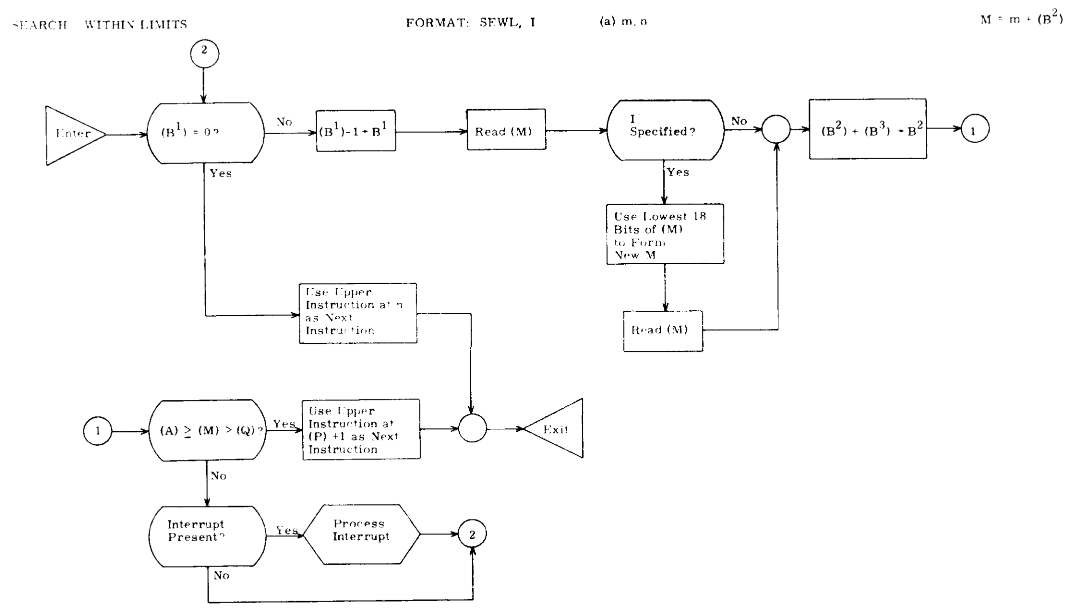
0-7, actual bank.

$, bank in which base address resides.

*, bank in which SEWL resides.

SEWL

FLOWCHART



DESCRIPTION

SEWL	1) (B^1) is number of addresses searched
	2) $n + (B^2)$ is first word address
	3) Addresses searched in increments of (B^3)
	4) Search for $(A) \geq (M) > (Q)$
	5) Jump to n if search not satisfied, otherwise full exit.
SEWL, I	Search indirectly

E X A M P L E S

PROBLEM: Given: $(B^1) = 100$, $(B^2) = 0$, $(B^3) = 1$

Search a table of 100 values starting at address FAB in Bank 1 for some value greater than Q, but less than or equal to the contents of A. If the search is satisfied, continue the program. If not, jump to address NOFIND.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SEWL	(1) FAB, NØFIND	

PROBLEM: Given: $(B^1) = 100$, $(B^2) = 0$, $(B^3) = 2$

Search a table of 200 values by searching only every other value (actually 100 values). The first address is MORT in Bank 2. If there is some value greater than Q, but less than or equal to A, continue the program. If not, jump to address NIL.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SEWL	(2) MØRT, NIL	

PROBLEM: Given: $(B^1) = 10$, $(B^2) = 0$, $(B^3) = 1$

Search a table of 10 values starting at address FETE in the bank where FETE resides for some value greater than Q, but less than or equal to A. If the search is satisfied, continue the program. If not, jump to address TIX.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SEWL	(Ø) FETE, TIX	

PROBLEM: Given: $(B^1) = 100$, $(B^3) = 1$

Search a table of 100 values starting at the address specified in Index Register 2 in Bank 3 for some value greater than Q, but less than or equal to A. If the search is satisfied, continue the program. If not, jump to address RECALL.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SEWL	(3), RECALL	

The SEARCH MAGNITUDE WITHIN LIMITS Instruction

The SEARCH MAGNITUDE WITHIN LIMITS instruction is an instruction that searches memory for an operand whose magnitude is greater than the contents of Q, but less than or equal to the contents of A. One memory reference results for each storage address searched.

Before this instruction is executed, three index registers are defined and must be set up as follows:

1. (B^1) - number of storage words to search
2. (B^2) - modifier for address m
3. (B^3) - incrementer for (B^2)

The incrementer is set for 1 for searching sequential words, 2 for searching every other word, etc.

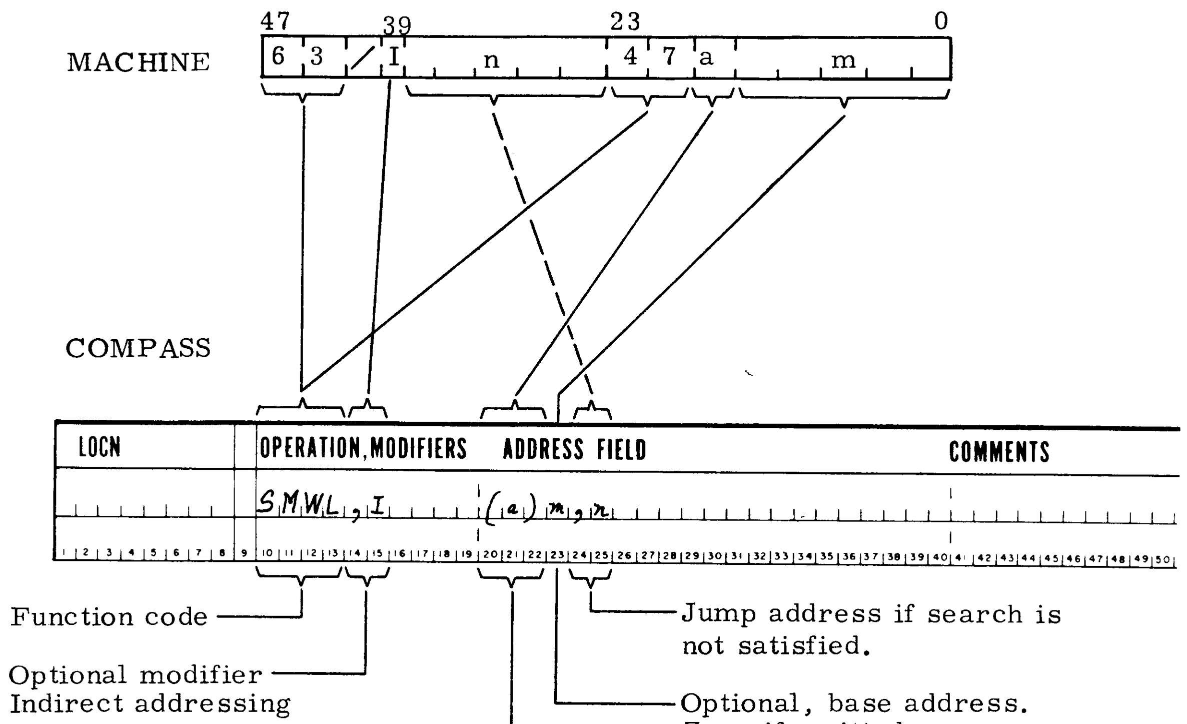
The bank term $\underline{a}$ allows the programmer to search in any bank starting with address $\underline{M}$ where $M = m + (B^2)$.

If the search is satisfied, a full exit is taken. The address of the operand that satisfied the search will be given by $m + (B^2) - (B^3)$.

If the search is not satisfied, program control will transfer to address $\underline{n}$ with the contents of Index Register 1 going to zero.

If I is specified by the programmer, the address searched is not M. Rather, the lowest 18 bits at address M become the address searched.

FORMAT



—Jump address if search is not satisfied.

—Optional, base address. Zero if omitted.

—Optional, bank term.

Current operand bank setting if omitted.

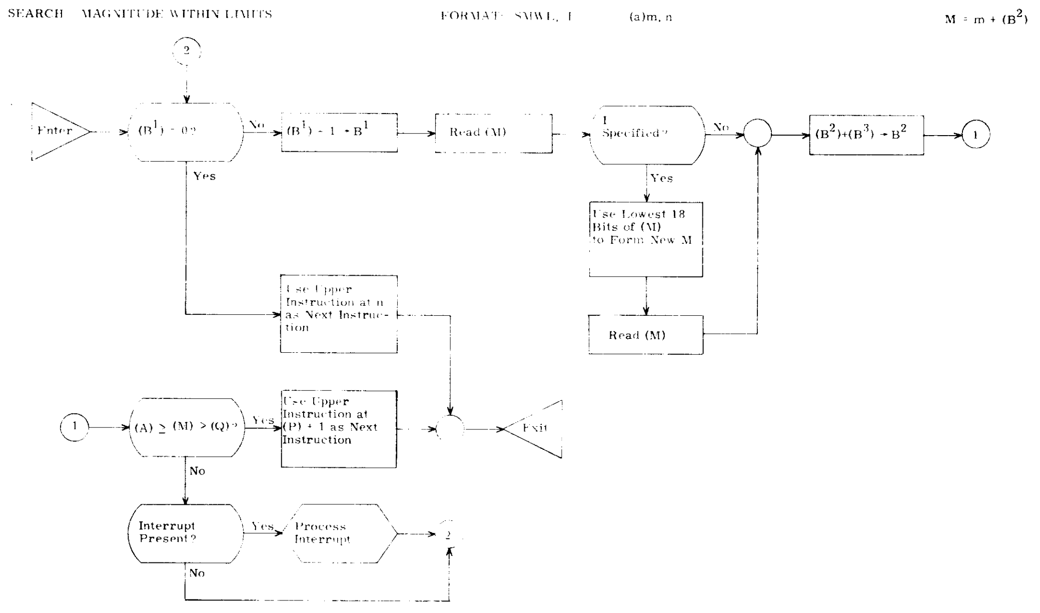
0-7, actual bank.

$, bank in which base address resides.

*, bank in which SMWL resides.

SMWL

FLOWCHART



DESCRIPTION

SMWL	1) (B^1) is number of addresses searched
	2) $m + (B^2)$ is first word address
	3) Addresses searched in increments of (B^3)
	4) Search for $(A) \geq \lvert (M) \rvert > (Q)$
	5) Jump to n if search not satisfied, otherwise full exit
SMWL, I	Search indirectly

PROBLEM: Given: $(B^1) = 100$, $(B^2) = 0$, $(B^3) = 1$

Search a table of 100 values starting at address TUB in Bank 1 for some value whose absolute value is greater than Q, but less than or equal to the contents of A. If the search is satisfied, continue the program. If not, jump to address SKIP.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SMWL	(1) TUB, SKIP	

PROBLEM: Given: $(B^1) = 100$, $(B^2) = 0$, $(B^3) = 2$

Search a table of 200 values by searching only every other value (actually 100 values). The first address is TORT in Bank 2. If there is some value whose absolute value is greater than Q, but less than or equal to A, continue the program. If not, jump to address SIL.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SMWL	(2) TØRT, SIL	

PROBLEM: Given: $(B^1) = 10$, $(B^2) = 0$, $(B^3) = 1$

Search a table of 10 values starting at address KATE in the bank where KATE resides for some value whose absolute value is greater than Q, but less than or equal to A. If the search is satisfied, continue the program. If not, jump to address SIX.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SMWL	($) KATE, SIX	

PROBLEM: Given: $(B^1) = 100$, $(B^3) = 1$

Search a table of 100 values starting at the address specified in Index Register 2 in Bank 3 for some value whose absolute value is greater than Q, but less than or equal to A. If the search is satisfied, continue the program. If not, jump to address PUNT.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SMWL	(3), PUNT	

The LOCATE LIST ELEMENT (UPPER) Instruction

The LOCATE LIST ELEMENT (UPPER) instruction is an instruction that reads from memory one element from a list of elements. No processing is done on the element with this instruction. This instruction, however, gives the needed information in order to process the element with future instructions.

An element is a word or a group of words usually in sequential order. A list is a set of elements. If a programmer's list of elements is in memory, he can call out any element of the list with this instruction. The machine does it by referencing the upper address portion of the beginning of each element in order to arrive at the next element of the list. At the end of the instruction the address of the element he wanted will be in the index register specified by $\underline{v}$ and the bank term will be contained in the operand bank setting. With this information he can process the element with future instructions.

The programmer calls out an element with this instruction by first entering the index register specified by $\underline{b}$ with the n^{th} element he wants minus 1; i.e., if he wants to work with the 5th element, he enters the index register with 4 since after 4 iterations in the hardware, he will have the address of the 5th element in Index Register $\underline{v}$. In Index Register $\underline{v}$ he enters initially the address of the first element.

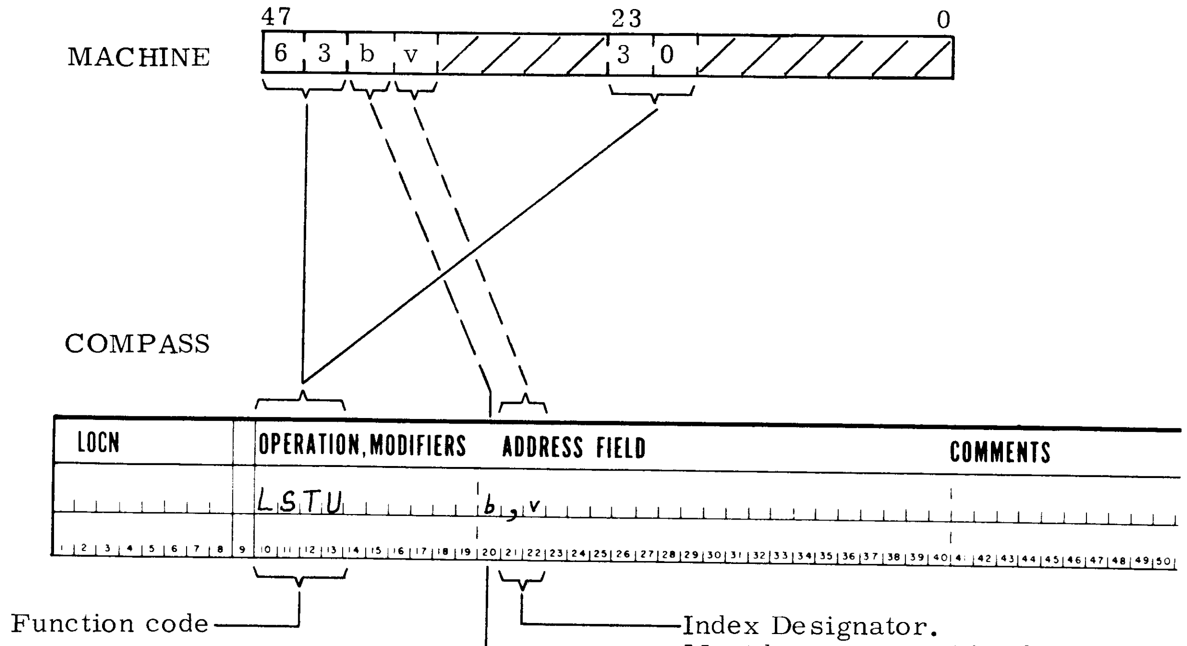
With these two specifications (both a must), the hardware iterates until the address of his element is in the index register specified by $\underline{v}$.

This instruction is especially important when a list is 200 elements or more. He can call the 247th element from the list without having to remember the address each time.

LOCATE LIST ELEMENT UPPER

LSTU

FORMAT

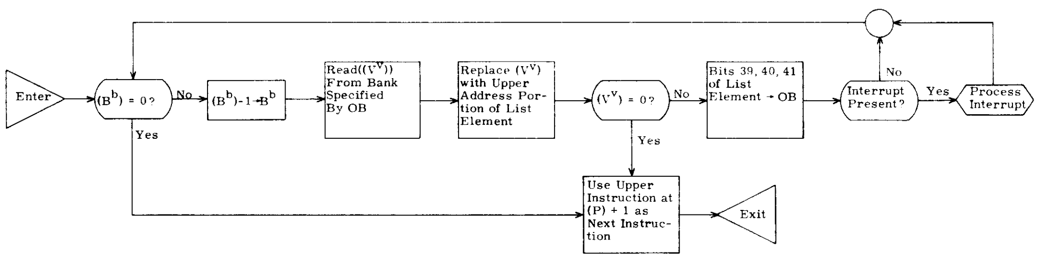


Index Designator.
Must be pre-set with address of first element of list.

Index designator
Must be pre-set with nth element minus one.

FLOWCHART

LOCATE LIST ELEMENT UPPER FORMAT: LSTU b, v


Enter
$(B^b) = 0?$
No
Yes
$(B^b)-1 \rightarrow B^b$
Read((V^v)) From Bank Specified By OB
Replace (V^v) with Upper Address Portion of List Element
$(V^v) = 0?$
No
Yes
Bits 39, 40, 41 of List Element → OB
Interrupt Present?
No
Yes
Process Interrupt
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

LSTU

1) (B^b) initially nth element -1
2) (V^v) initial address of first element
3) (V^v) final address of nth element

EXAMPLES

PROBLEM: Given: $(B^1) = 35$, (B^2) = START

Locate the 36th element of the list starting at address START.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LSTU	1,2	

PROBLEM: Given: $(B^2) = 498$, (B^3) = BEGIN

Locate the 499th element of the list starting at address BEGIN.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LSTU	2,3	

PROBLEM: Given: $(B^4) = 503$

Locate the 504th element of a list starting at the address specified in Index Register 5.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LSTU	4,5	

PROBLEM: Given: (B^6) = ENTER

Locate the nth element minus one (in Index Register 5) starting at address ENTER.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LSTU	5,6	

The LOCATE LIST ELEMENT (LOWER) Instruction

The LOCATE LIST ELEMENT (LOWER) instruction is an instruction that reads from memory one element from a list of elements. No processing is done on the element with this instruction. This instruction, however, gives the needed information in order to process the element with future instructions.

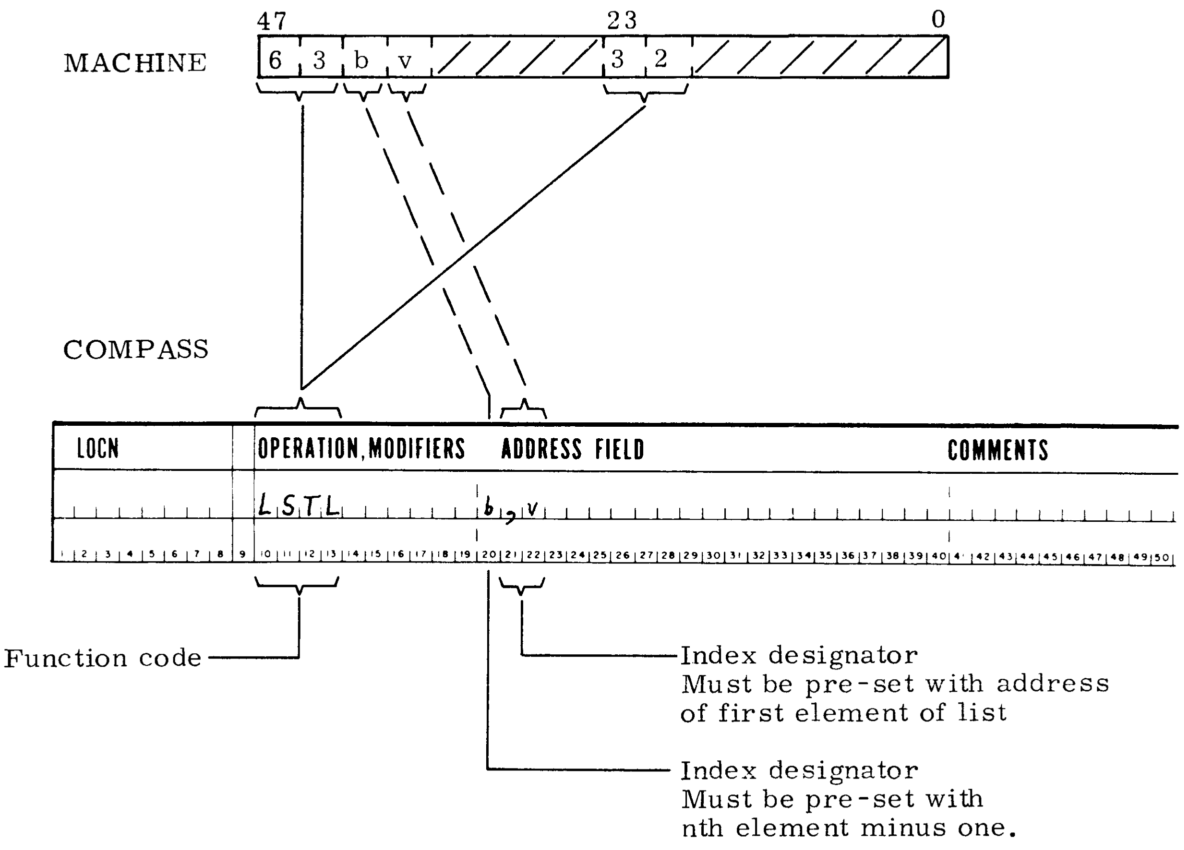
An element is a word or a group of words usually in sequential order. A list is a set of elements. If a programmer's list of elements is in memory, he can call out any element of the list with this instruction. The machine does it by referencing the lower address portion of the beginning of each element in order to arrive at the next element of the list. At the end of the instruction the address of the element he wanted will be in the index register specified by $\underline{v}$ and the bank term will be contained in the operand bank setting. With this information he can process the element with future instructions.

The programmer calls out an element with this instruction by first entering the index register specified by $\underline{b}$ with the n^{th} element he wants minus 1; i.e., if he wants to work with the 5th element, he enters the index register with 4 since after 4 iterations in the hardware, he will have the address of the 5th element in index register $\underline{v}$. In Index Register $\underline{v}$ he initially enters the address of the first element.

With these two specifications (both a must), the hardware iterates until the address of his element is in the index register specified by $\underline{v}$.

This instruction is especially important when a list is 200 elements or more. He can call the 247th element from the list without having to remember the address each time.

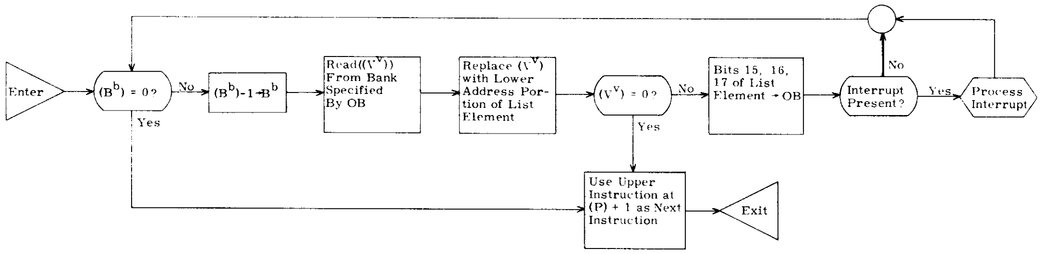
FORMAT


MACHINE
47
23
0
6
3
b
v
3
2
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
LSTL
b, v
Function code
Index designator
Must be pre-set with address
of first element of list
Index designator
Must be pre-set with
nth element minus one.

FLOWCHART

LOCATE LIST ELEMENT LOWER

FORMAT: LSTL b, v


Enter
$(B^b) = 0$?
No
Yes
$(B^b)-1 \rightarrow B^b$
Read$((V^v))$ From Bank Specified By OB
Replace (V^v) with Lower Address Portion of List Element
$(V^v) = 0$?
No
Yes
Bits 15, 16, 17 of List Element → OB
Interrupt Present?
No
Yes
Process Interrupt
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

LSTL

1) (B^b) initially nth element -1

2) (V^v) initial address of first element

3) (V^v) final address of nth element

PROBLEM: Given: $(B^1) = 39$, $(B^2) = \text{INIT}$.
Locate the 40th element of a list starting at address INIT.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LSTL	1,2	

PROBLEM: Given: $(B^3) = 5003$, $(B^4) = \text{GO}$.
Locate the 5004th element of a list starting at address GO.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LSTL	3,4	

PROBLEM: Given: $(B^5) = 496$
Locate the 497th element of a list starting at an address specified in Index Register 6.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LSTL	5,6	

PROBLEM: Given: $(B^6) = \text{COMMENCE}$
Locate the nth element minue one (in Index Register 4) of a list starting at address COMMENCE.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		LSTL	4,6	

The SCAN Instruction

The SCAN instruction is an instruction that compares "bytes" of memory words with a byte in the Q register. The comparisons made are for equality, greater than, less than, not equal, less than or equal, and greater than or equal. One and only one of these must be specified by the programmer.

Before the instruction is executed the A register must contain the byte count, i.e., the number of bytes that are to be scanned. Also the index register specified by $\underline{v}$ must contain the off-set designator for the memory word, i.e., the right-most bit position of the byte in memory where the scan is to start.

Within the instruction Qo must be specified where $\underline{o}$ is the off-set designator for Q, i.e., the right-most bit position for the byte in Q.

The modifier Ee represents the byte size, i.e., the width of the byte scanned. The letter $\underline{e}$ represents the number of bit positions. E6 would represent a 6-bit byte.

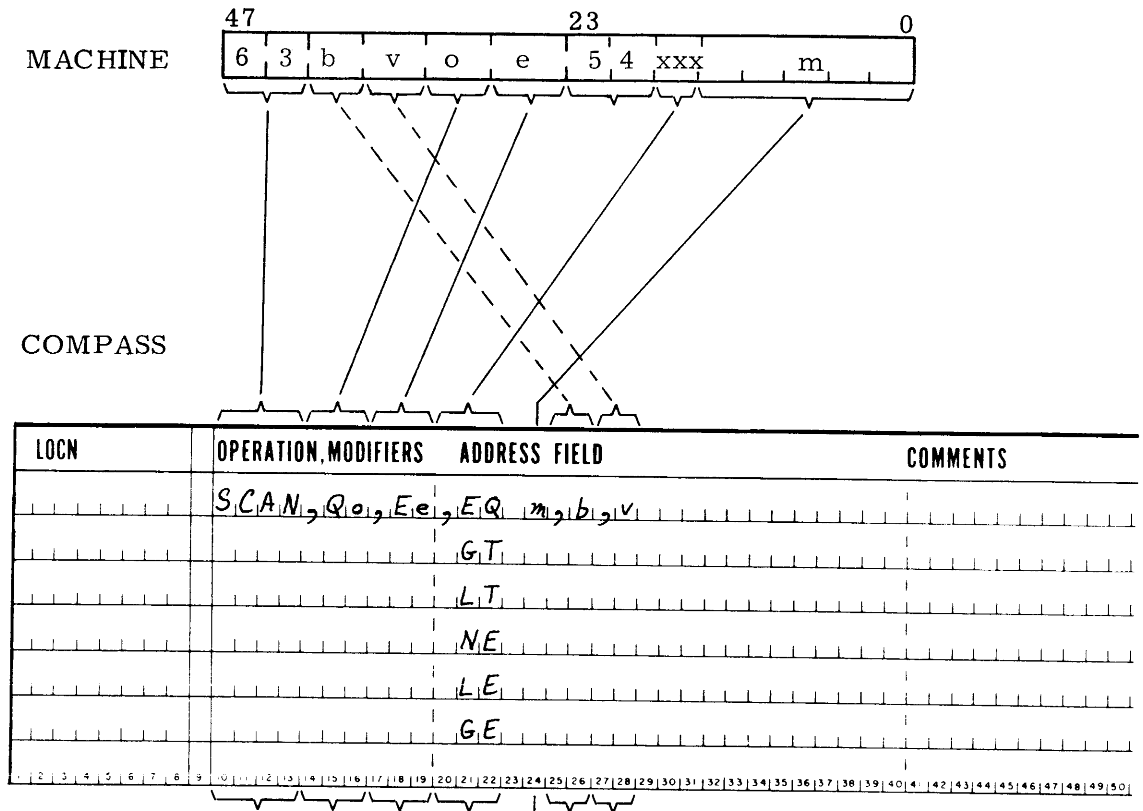
The first address searched is $\underline{M}$ where $M = m + (B^b)$. The scan continues through memory until the search is satisfied or until the byte count is exhausted.

If the search is satisfied, program control transfers to (P) + 2. If the search is not satisfied, program control transfers to (P) + 1.

FORMAT

MACHINE

COMPASS



Function code

Off-set designator for Q

Byte size

Modifier
EQ, M byte = Q byte
GT, M byte > Q byte
LT, M byte < Q byte
NE, M byte ≠ Q byte
LE, M byte ≤ Q byte
GE, M byte ≥ Q byte

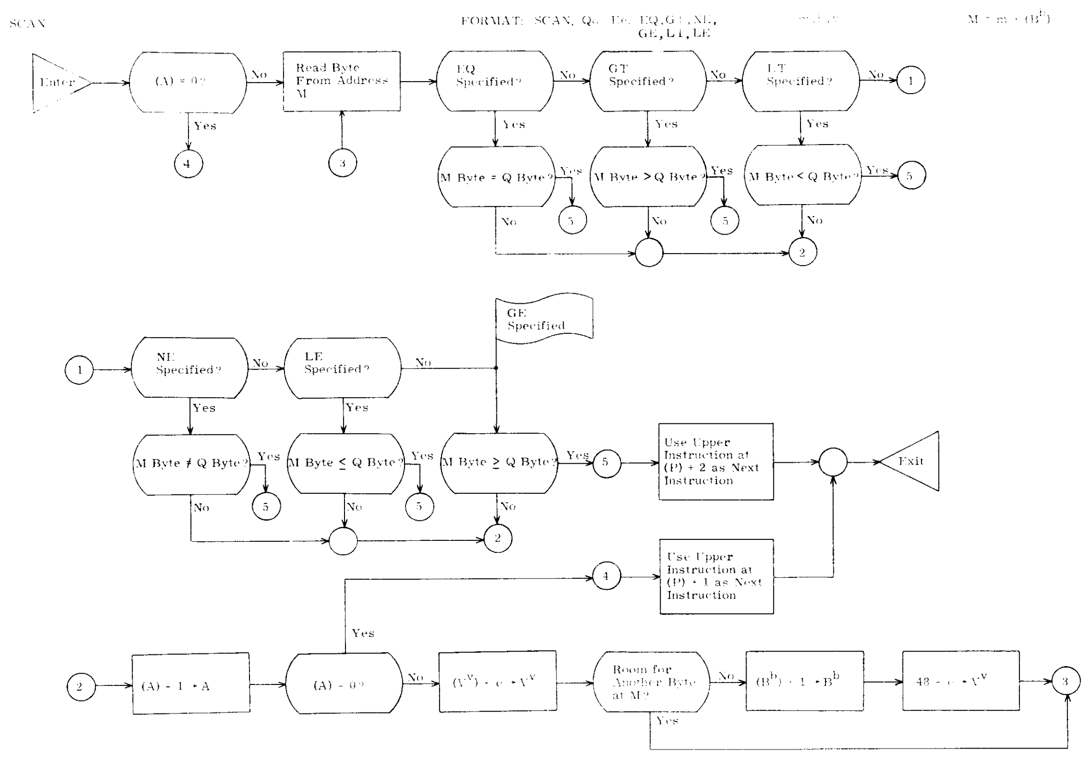
Index designator.
Zero if blank
Must be pre-set with off-set designator for memory.

Index designator.
Zero if omitted.
1-6, relative addressing.

Optional, base address.
Zero if omitted.

SCAN

FLOWCHART



DESCRIPTION

SCAN

1) Scan bytes of memory in succession until,

a. m byte = Q byte
b. m byte ≠ Q byte
c. m byte > Q byte
d. m byte < Q byte
e. m byte ≥ Q byte
f. m byte ≤ Q byte

2) Full exit if scan not satisfied

Skip exit if scan satisfied

PROBLEM: Given: (A) = 16, (B^1) = 0, (B^2) = 42, (Q) = 33 octal

Scan a set of 16 6-bit characters starting at address KIT for being equal to 33 octal.

SOLUTION:

LOCN	OPERATION,MODIFIERS ADDRESS FIELD	COMMENTS
	SCAN,Q0,E6,EQ KIT,1,2	

PROBLEM: Given: (A) = 50, (B^1) = 0, (B^2) = 24, (Q) = 0——04563 octal

Scan a set of 50 24-bit quantities starting at address KOKO for something less than or equal to 4563 octal.

SOLUTION:

LOCN	OPERATION,MODIFIERS ADDRESS FIELD	COMMENTS
	SCAN,Q0,E24,LE KØKØ,1,2	

PROBLEM: Given: (A) = 1000, (B^1) = 0, (B^2) = 24, (Q) = 12345

Scan a set of 1000 24-bit quantities starting at address PORT for something greater than or equal to 12345.

SOLUTION:

LOCN	OPERATION,MODIFIERS ADDRESS FIELD	COMMENTS
	SCAN,Q0,E24,GE PØRT,1,2	

PROBLEM: Given: (A) = 32, (B^1) = 0, (B^2) = 42, (Q) = XX0——0

Scan a set of 32 6-bit characters starting at address SORT for some character less than XX.

SOLUTION:

LOCN	OPERATION,MODIFIERS ADDRESS FIELD	COMMENTS
	SCAN,Q42,E6,LT SØRT,1,2	

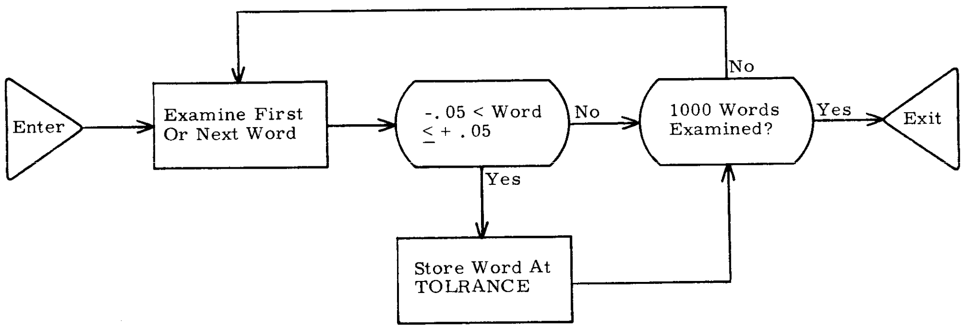
NEW CONCEPTS OF GROUP 15

The search instructions search a table of one or more memory words to find a word that satisfies a given condition. If a word is found, the search terminates and an exit is taken. The programmer can then determine what address satisfied the search. If a word is not found, the search terminates and a different exit is taken. The programmer can then record that the search was not satisfied.

Problem 15:

Assume a random set of 1000 floating point operands ranging from -100 to + 100 resides in memory starting at address TESTCASE. Write a subprogram that will store all values between -.05 and +.05 in a table starting at address TOLRANCE. Assume less than 10 of these.

Flowchart:



Problem 15 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	RESISTØR	
	ENTRY	RESISTØR	
CØNST	DEC	-.05,.05	
TØLRANCE	BSS	10	
RESISTØR	BSS	1	
	ENI	0,4	TØLRANCE PØINTER
	ENI	1000,1	NØ ØF WØRDS
	ENI	0,2	M+(B2) IS FWA
	ENI	1,3	SEQUENTIAL INCREMENTER
	LDA	CØNST+1	UPPER LIMIT
	LDQ	CØNST	LØWER LIMIT
CØNSRCH	SEWL	TESTCASE,RESISTØR	
	RXT	A,D	SAVE
	LDA	TESTCASE-1,2	SATISFIED ØPERAND
	STA	TØLRANCE,4	
	INI	1,4	BUMP PØINTER
	RXT	D,A	RESTØRE
	SLJ	CØNSRCH	
	END		

Somewhere within this subprogram would also be included the symbol TESTCASE in the location field with a declaration of the prestored data or area reserved.

Student Problem 15A:

A list of 1000 integer operands ranging from 1 - 100 reside in memory starting at address PEØPLE. Write a subprogram that will count the number of integers between 18 and 35. Store this count at address DRAFTABL.

Flowchart:

Problem 15A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 16

STORAGE TEST

GROUP 16

STORAGE TEST

1.	Storage Skip	SSK
2.	Storage Shift	SSH

This group of instructions checks the contents of a storage address for being positive or negative. If positive (false, no), a half exit is taken. If negative (true, yes), a full exit is taken. Because of this principle, these two instructions are upper instructions, and the computer assembler will force them to the upper position. Therefore the programmer should not modify these instructions with modifiers, bank terms, or second index designators since this overrides the assembler and makes the use of these instructions ineffective.

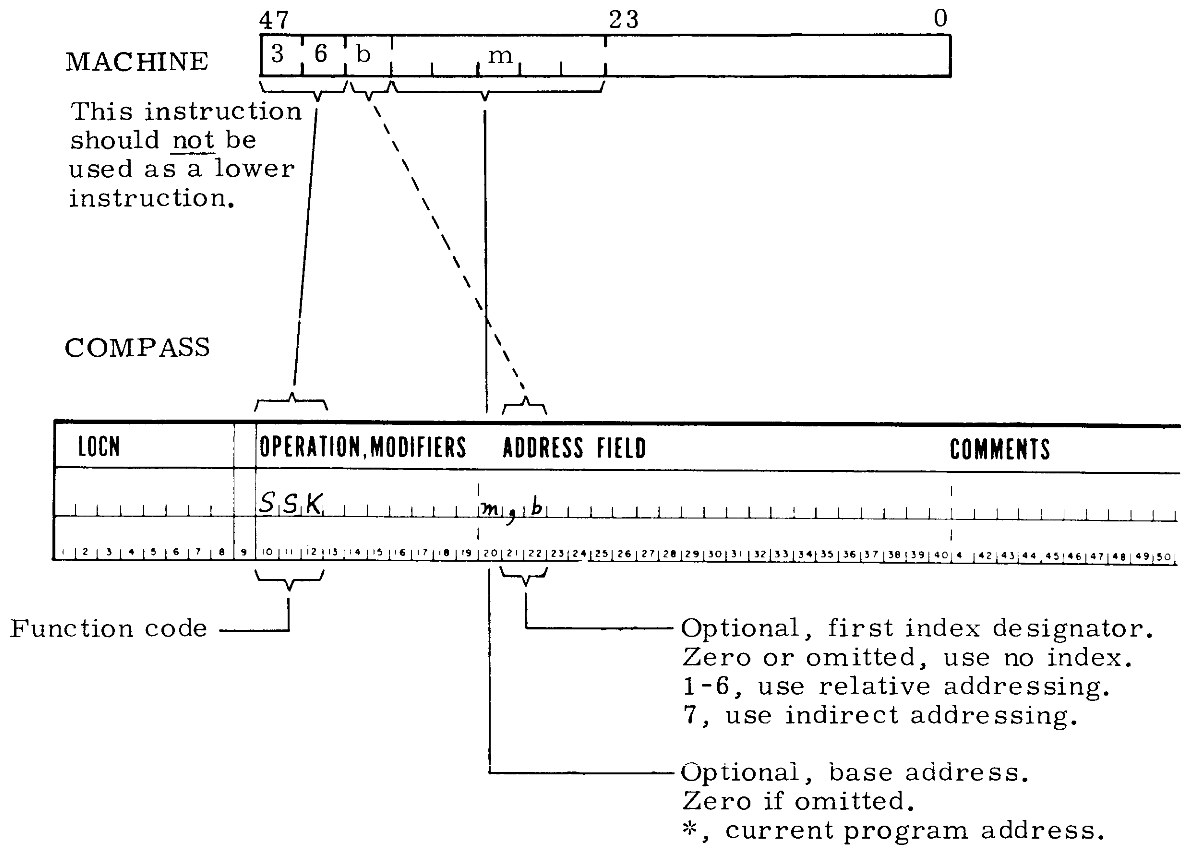
The second instruction is just like the first; however, the contents of the storage address are left shifted by one after the check.

The STORAGE SKIP Instruction

The STORAGE SKIP instruction is an instruction that checks the contents of a storage address for a negative quantity. This storage address is composed of a modified base address $\underline{M}$ where $M = m + (B^b)$. One memory reference is made.

If the contents of the storage address are negative, a full exit is taken. If the contents are positive, a half exit is taken. The instruction leaves the contents of the storage address unchanged.

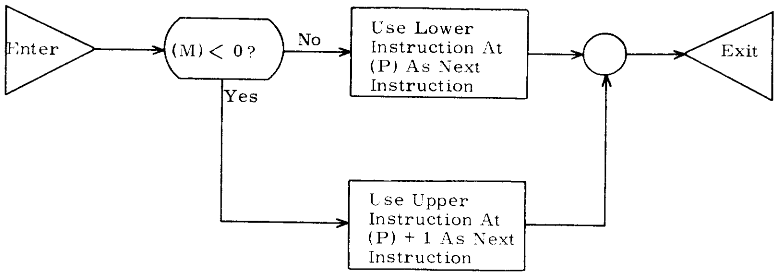
FORMAT


MACHINE
47
23
0
3
6
b
m
This instruction should not be used as a lower instruction.
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
SSK
m, b
Function code
Optional, first index designator. Zero or omitted, use no index. 1-6, use relative addressing. 7, use indirect addressing.
Optional, base address. Zero if omitted. *, current program address.

F L O W C H A R T

STORAGE SKIP*

FORMAT: SSK m, b $M = m + (B^b)$


Enter
(M) < 0 ?
No
Yes
Use Lower Instruction At (P) As Next Instruction
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

*Forced upper by the assembler

D E S C R I P T I O N

SSK Half exit if $(M) \geq 0$, otherwise full exit

EXAMPLES

PROBLEM:

Check the contents of address ANS. If negative, full exit. If positive, half exit.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SSK	ANS	

PROBLEM:

Check the contents of address BEATLE modified by Index Register 4. If negative, full exit. If positive, half exit.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SSK	BEATLE,4	

PROBLEM:

Check the contents of the address specified in Index Register 5. If negative, full exit. If positive, half exit.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SSK	,5	

PROBLEM:

Check the contents of address SLINK in the bank where the SSK resides. If negative, full exit. If positive, half exit.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SSK	(*)SLINK	

The STORAGE SHIFT Instruction

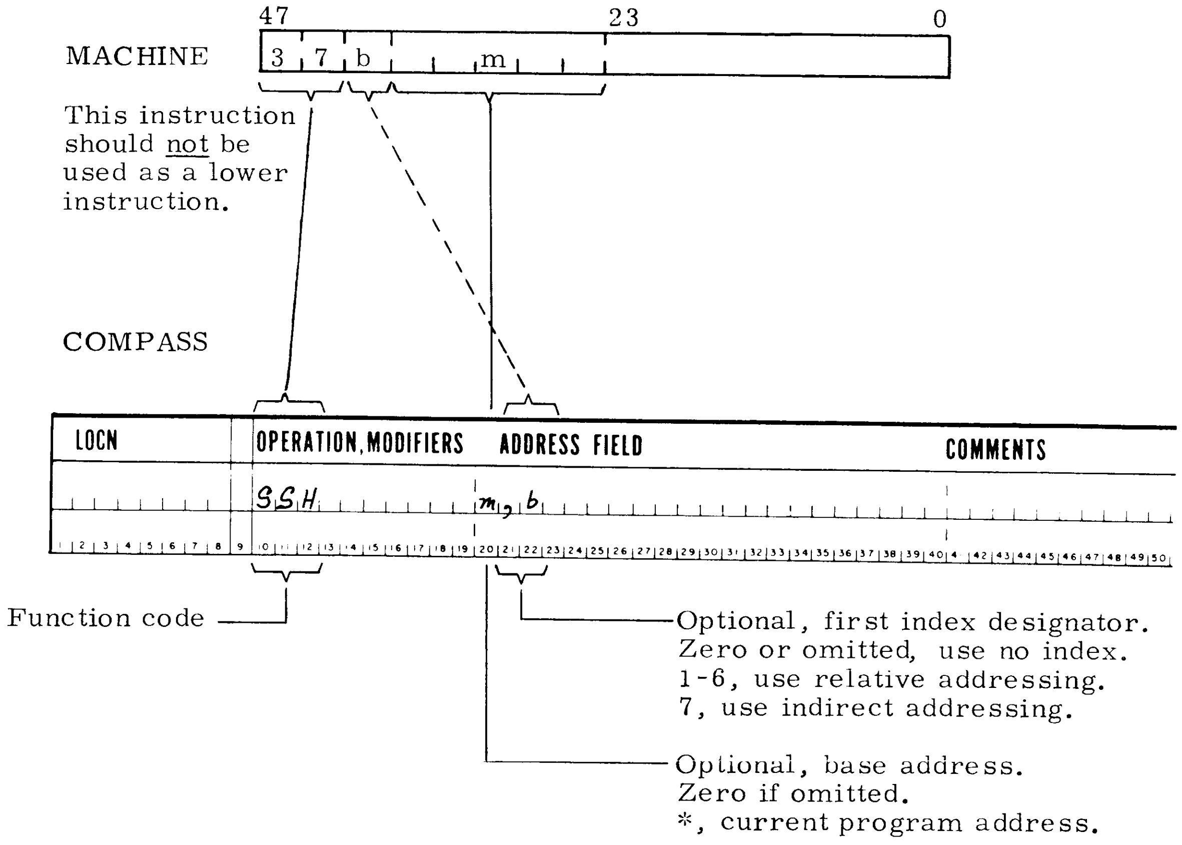
The STORAGE SHIFT instruction is an instruction that checks the contents of a storage address for a negative quantity. This storage address is composed of a modified base address $\underline{M}$ where $M = m + (B^b)$. One memory reference is made.

If the contents of the storage address are negative, a full exit is taken. If the contents are positive, a half exit is taken. In either case, the contents of the storage address are left-shifted one binary position, end around.

STORAGE SHIFT

SSH

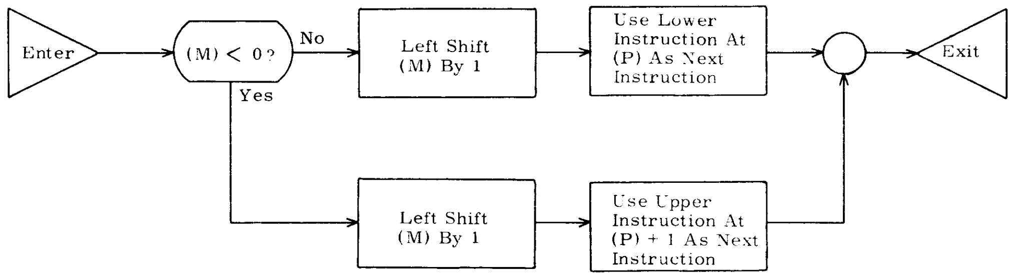
FORMAT


MACHINE
47
23
0
3
7
b
m
This instruction should not be used as a lower instruction.
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
SSH
m, b
Function code
Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
7, use indirect addressing.
Optional, base address.
Zero if omitted.
*, current program address.

SSH

F
L
O
W
C
H
A
R
T

STORAGE SHIFT* FORMAT: SSH m, b $M = m + (B^b)$


Enter
(M) < 0?
No
Yes
Left Shift (M) By 1
Use Lower Instruction At (P) As Next Instruction
Left Shift (M) By 1
Use Upper Instruction At (P) + 1 As Next Instruction
Exit

*Forced upper by the assembler

D
E
S
C
R
I
P
T
I
O
N

SSH

1) Half exit if $(M) \geq 0$, otherwise full exit
2) In either case left shift (M) by one

PROBLEM:

Examine the contents of address SWITCH. If negative, full exit. If positive, half exit. In either case, shift the contents.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SSH	SWITCH	

PROBLEM:

Examine the contents of address SWITCH modified by Index Registers 2 and 3. If negative, full exit. If positive, half exit. In either case, shift the contents.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SSH	SWITCH,2,3	

PROBLEM:

Examine the contents of the address specified in Index Register 4. If negative, full exit. If positive, half exit. In either case, shift the contents of the address.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SSH	,4	

PROBLEM:

Examine the contents of address PEACH in the bank where PEACH resides. If negative, full exit. If positive, half exit. In either case, shift left by 1 the contents of the address.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SSH	(#) PEACH	

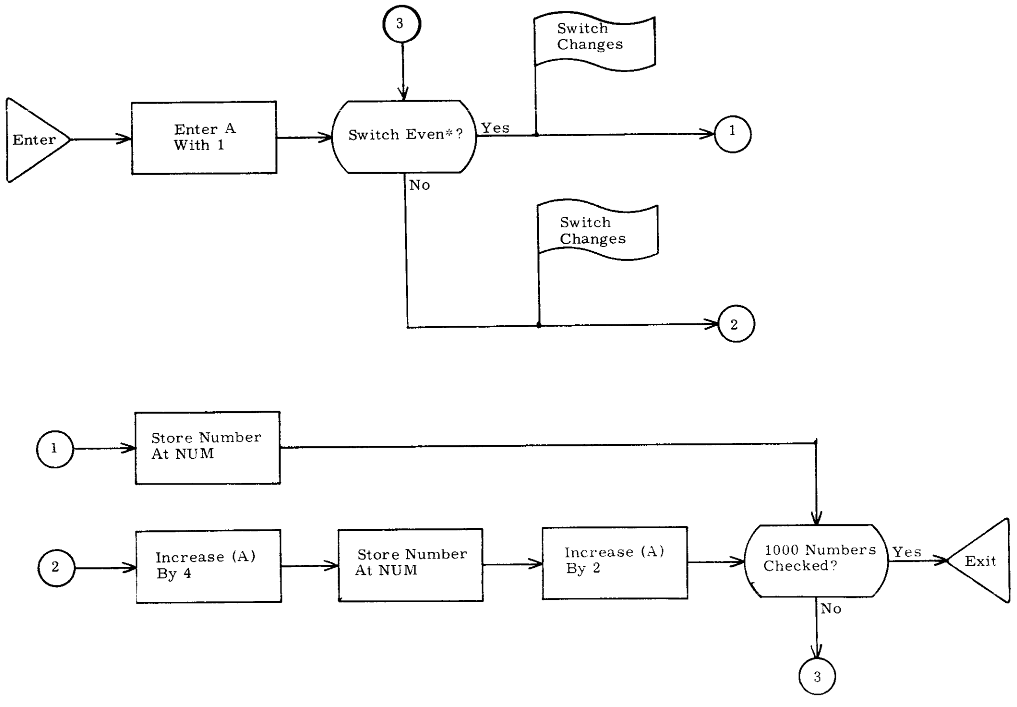
NEW CONCEPTS OF GROUP 16

No new concepts are presented here except that the programmer can use these instructions to branch to different points in his program. The second instruction automatically changes the transfer point at the discretion of the programmer.

Problem 16:

Generate starting at address NUM the first 1000 positive integers that are not divisible by 2 or 3.

Flowchart:



* Switch Starts Out Even

Problem 16 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	GENERATE	
	ENTRY	GENERATE	
NUM	BSS	1000	
SWITCH	ØCT	2525252525252525	
GENERATE	BSS	1	
	ENI	0,1	
	ENA	1	
CKSWITCH	SSH	SWITCH	
	SLJ	ST	
	INA	4	
	STA	NUM,1	
	INA	2	
CKCNT	ISK	999,1	
	SLJ	CKSWITCH	
	SLJ	GENERATE	
ST	STA	NUM,1	
	SLJ	CKCNT	
	END		

Student Problem 16A:

Generate starting at address NUM a table of integers with the following pattern:

1, -2, 5, -7, 9, -12, 13, ...

Flowchart:

Problem 16A would be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 17

VARIABLE DATA FIELD TRANSMISSION

GROUP 17

VARIABLE DATA FIELD TRANSMISSION

1.	Load Byte	LBYT
2.	Store Byte	SBYT

This group of instructions transmits a byte of a word either from memory to an operational register (A or Q), or from an operational register to memory. One memory reference is required. The byte size is variable as well as its position in the word.

If no indexing is specified, a full exit is taken after the transmission. If indexing is specified, the type of exit is determined by the amount of storage word left. If sufficient room is left for another byte, a skip exit is taken. If insufficient room is left, a full exit is taken whereupon the programmer has the opportunity to change the relative index counter and restore the memory off-set designator for the next word.

The LOAD BYTE Instruction

The LOAD BYTE instruction is an instruction that transmits a "byte" of a memory word from an 18-bit storage address to the A or Q register, whichever is specified. One memory reference is made.

The 18-bit address is composed of the operand bank setting and $\underline{M}$ where $M = m + (B^b)$. The operation leaves the contents of the storage address unchanged. The byte portion of the memory word replaces the byte portion of the specified register.

The modifier Ao or Qo, one of which must be specified, represents the destination register. The o following it is the off-set designator for the register, i.e., the right-most bit position of the byte in the register. For example, if the upper half of the A register is to be filled, the specification would be A24. The letter o ranges from 0 to 47.

The modifier Ee represents the byte size, i.e., the width of the byte transferred. The letter e represents the number of bit positions. For example, a byte size of 24 would be specified as E24. The letter e ranges from 1 to 48.

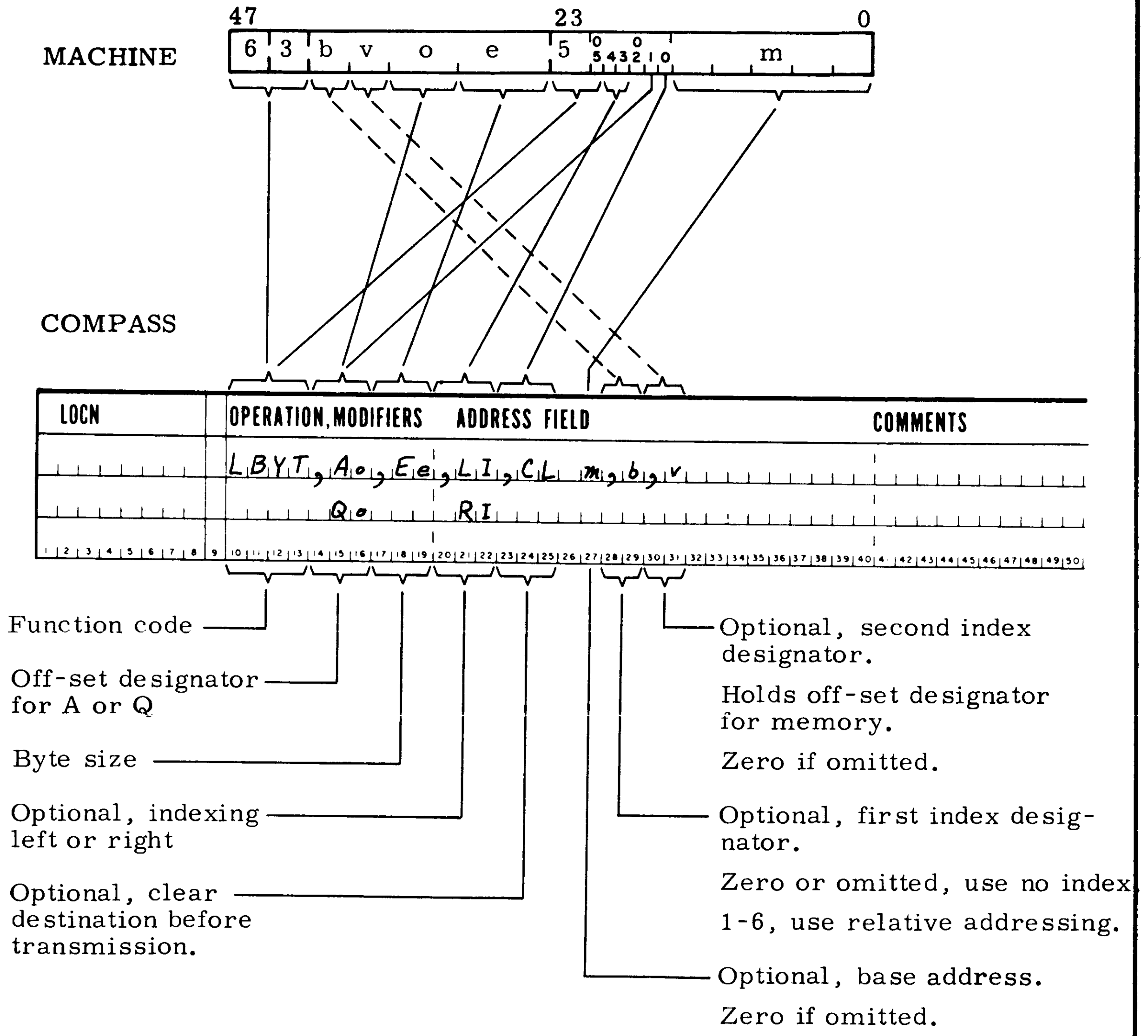
The contents of the index register specified by $\underline{v}$ represent the off-set designator for the memory word, i.e., the right-most bit position of the byte in the memory word. For example, if the byte is to come from the lower half of a memory word, the contents of the index register $\underline{v}$ would be zero.

If the programmer specifies CL, the <u>destination</u> will be cleared before the byte is inserted. In this instruction it would be either A or Q.

If the programmer specifies LI or RI, automatic indexing takes place with respect to the off-set designator for the memory word. This is used to unpack bytes in memory.

A full exit is taken if LI or RI is not specified. If either is specified, a full exit is taken when the off-set designator finishes with a memory word, and a skip exit, (P) + 2, is taken when a memory word is not finished.

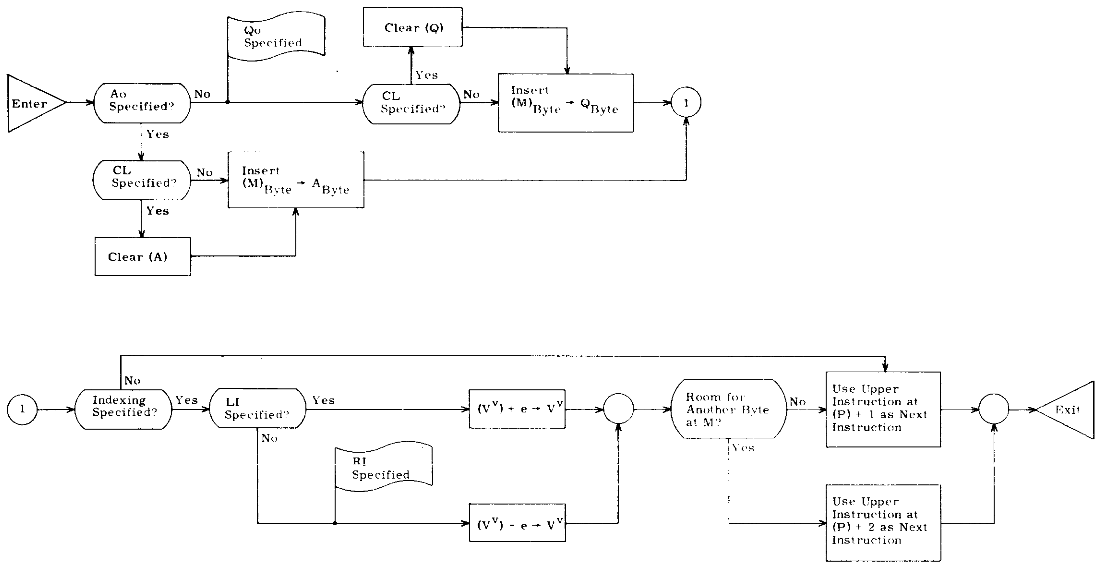
FORMAT


47
23
0
MACHINE
6 3 b v o e 5 0 5 4 3 2 1 0 0 m
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
LBYT, Ao, Ee, LI, CL m, b, v
Qo RI
Function code
Off-set designator for A or Q
Byte size
Optional, indexing left or right
Optional, clear destination before transmission.
Optional, second index designator.
Holds off-set designator for memory.
Zero if omitted.
Optional, first index designator.
Zero or omitted, use no index
1-6, use relative addressing.
Optional, base address.
Zero if omitted.

FLOWCHART

LOAD BYTE

FORMAT: LBYT, Ao, Ec, LI, CL m, b, v $M = m + (B^b)$
Qo RI



DESCRIPTION

LBYT

1) Transmit m byte → A or Q

2) Contents of M remain unchanged

E X A M P L E S

PROBLEM: Given: $(B^2) = 42$

Load the highest 6 bits from address SMOKEY into the lowest portion of A clearing the rest of A.

SOLUTION:

LOCN	OPERATION, MODIFIERS ADDRESS FIELD	COMMENTS
	LBYT,A0,E6,CL SMØKEY,,2	

PROBLEM: Given: $(B^2) = 24$

Load the upper address portion of the memory word at address SOAK into the lower address portion of the Q register without clearing the rest of Q.

SOLUTION:

LOCN	OPERATION, MODIFIERS ADDRESS FIELD	COMMENTS
	LBYT,Q0,E15 SØAK,,2	

PROBLEM:

Load the right-most BCD character at address POKE modified by Index Register 3 into the lowest portion of A without clearing the rest of A.

SOLUTION:

LOCN	OPERATION, MODIFIERS ADDRESS FIELD	COMMENTS
	LBYT,A0,E6 PØKE,3	

PROBLEM: Given: $(B^2) = 42$

Load the highest BCD character at address BUTT modified by Index Register 1 into the bottom part of A clearing the rest of A. Specify right indexing.

SOLUTION:

LOCN	OPERATION, MODIFIERS ADDRESS FIELD	COMMENTS
	LBYT,A0,E6,RI,CL BUTT,1,2	

The STORE BYTE Instruction

The STORE BYTE instruction is an instruction that transmits a "byte" of the A or Q register, whichever is specified, to a memory word at an 18-bit storage address. One memory reference is made.

The 18-bit address is composed of the operand bank setting and $\underline{M}$ where $M = m + (B^b)$. The operation leaves the contents of the register unchanged. The byte portion of the register replaces the byte portion of the memory word.

The modifier Ao or Qo, one of which must be specified, represents the source register. The $\underline{o}$ following it is the off-set designator for the register, i.e., the right-most bit position of the byte in the register. For example, if the upper half of the A register is the source byte, the specification would be A24. The letter $\underline{o}$ ranges from 0 to 47.

The modifier Ee represents the byte size, i.e., the width of the byte transferred. The letter $\underline{e}$ represents the number of bit positions. For example, a byte size of 24 would be specified as E24. The letter $\underline{e}$ ranges from 1 to 48.

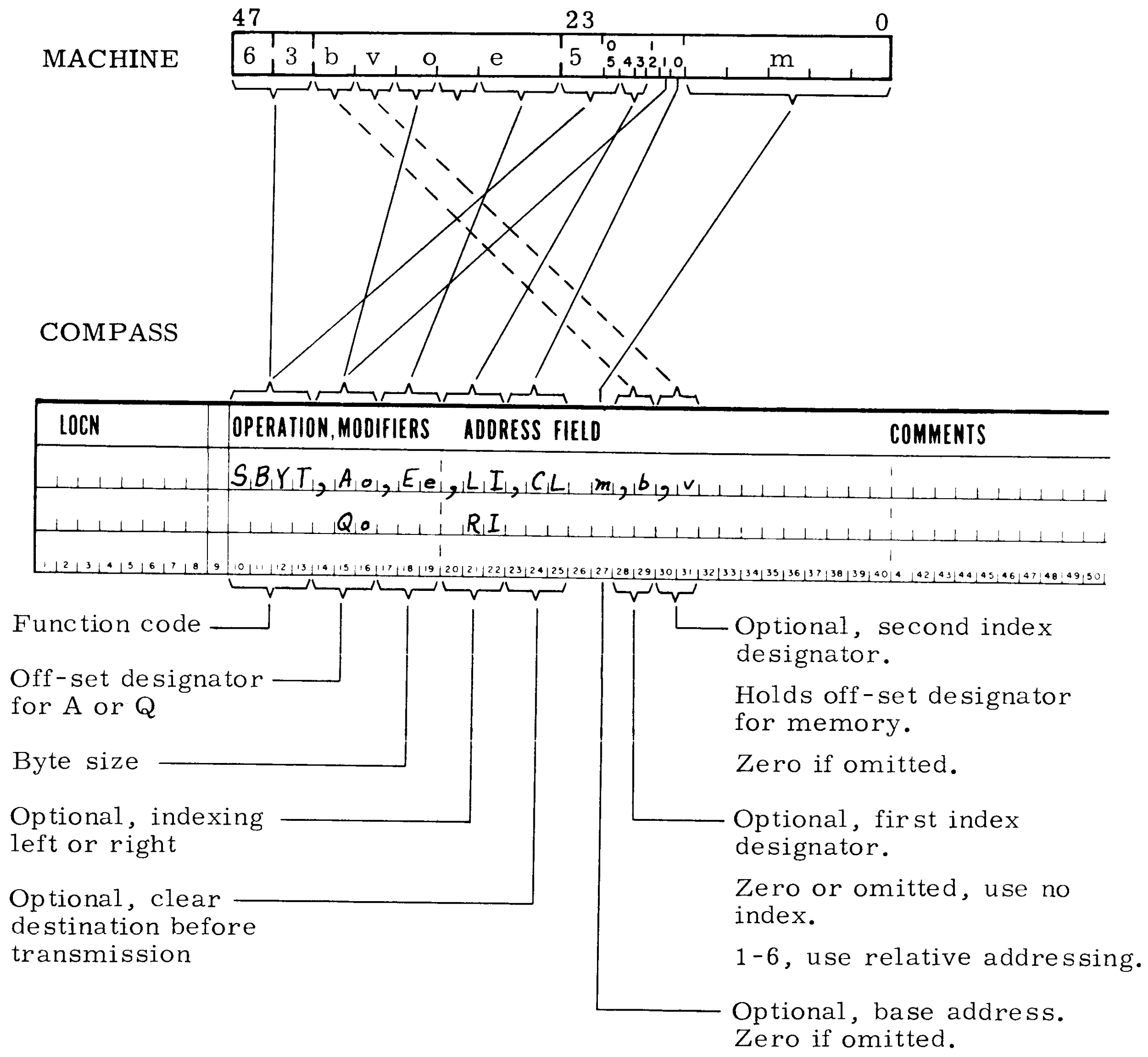
The contents of the index register specified by $\underline{v}$ represent the off-set designator for the memory word, i.e., the right-most bit position of the byte in the memory word. For example, if the byte is to be transferred to the lower half of a memory word, the contents of the index register $\underline{v}$ would be zero.

If the programmer specifies CL, the destination will be cleared before the byte is inserted. In this instruction it would be the memory word.

If the programmer specifies LI or RI, automatic indexing takes place with respect to the off-set designator for the memory word. This is used to pack bytes in memory.

A full exit is taken if LI or RI is not specified. If either is specified, a full exit is taken when the off-set designator finishes with a memory word, and a skip exit, (P) + 2 is taken when a memory word is not finished.

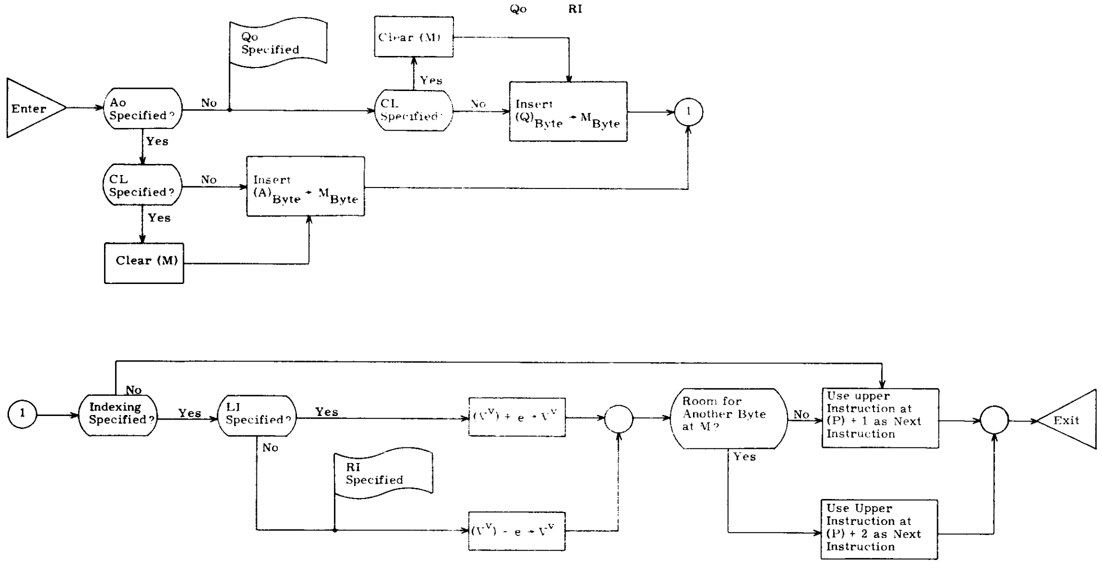
FORMAT


MACHINE
47
23
0
6
3
b
v
o
e
5
0
5 4 3 2 1 0
m
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
SBYT, Ao, Ee, LI, CL m, b, v
Qo RI
Function code
Off-set designator for A or Q
Byte size
Optional, indexing left or right
Optional, clear destination before transmission
Optional, second index designator.
Holds off-set designator for memory.
Zero if omitted.
Optional, first index designator.
Zero or omitted, use no index.
1-6, use relative addressing.
Optional, base address. Zero if omitted.

SBYT

FLOWCHART

STORE BYTE   FORMAT: SBYT, Ao, Fe, LI, CL   m, b, v $M = m + (B^b)$
Qo   RI



Enter
Ao Specified?
No
Yes
Qo Specified
CL Specified?
Yes
Clear (M)
No
Insert $(Q)_{Byte} \to M_{Byte}$
1
CL Specified?
No
Insert $(A)_{Byte} \to M_{Byte}$
Yes
Clear (M)
1
Indexing Specified?
No
Yes
LI Specified?
Yes
No
RI Specified
$(V^v) + e \to V^v$
$(V^v) - e \to V^v$
Room for Another Byte at M?
No
Yes
Use upper Instruction at (P) + 1 as Next Instruction
Use Upper Instruction at (P) + 2 as Next Instruction
Exit

DESCRIPTION

SBYT
1) Transmit A or Q byte → M
2) Contents of A or Q remain unchanged

PROBLEM: Given: $(B^2) = 42$

Store the lowest BCD character from A in the highest portion of the word at address SLINKY leaving the rest of the memory word untouched.

SOLUTION:

LOCN	OPERATION,MODIFIERS ADDRESS FIELD	COMMENTS
	SBYT,A0,E6 SLINKY,,2	

PROBLEM:

Store the upper address portion of Q in the lower address portion of the word at an address specified in Index Register 3 clearing the rest of the memory word.

SOLUTION:

LOCN	OPERATION,MODIFIERS ADDRESS FIELD	COMMENTS
	SBYT,Q24,E15,CL ,3	

PROBLEM: Given: $(B^2) = 24$

Store the lower address portion of Q in the upper address portion of the word at address SULKY without clearing the rest of the memory word. Specify right indexing.

SOLUTION:

LOCN	OPERATION,MODIFIERS ADDRESS FIELD	COMMENTS
	SBYT,Q0,E15,RI SULKY,,2	

PROBLEM: Given: $(B^4) = 0$

Store the lower address portion of Q in the lower address portion of the word at address SLIP modified by Index Register 2 without clearing the rest of the memory word. Specify left indexing.

SOLUTION:

LOCN	OPERATION,MODIFIERS ADDRESS FIELD	COMMENTS
	SBYT,Q0,E15,LI SLIP,2,4	

NEW CONCEPTS OF GROUP 17

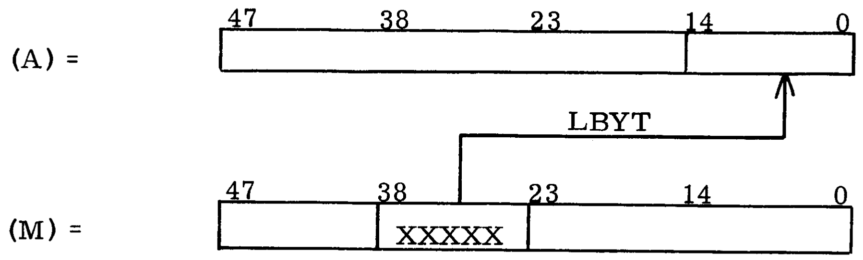
The principle of the LBYT and SBYT instructions should offer no more of a problem than the LDA and STA instructions. The BYTE instructions transmit one byte from memory to a register or from a register to memory, and exits. The only problem is the setting up of the instruction so that it does what the programmer wants it to do.

Since a portion of a word can be transmitted, there are several definitions which one must know. These apply to both instructions.

1. Index designator v where (V^V) represents the off-set designator i. e., the rightmost bit position of the byte of the memory word.
2. Ee where e represents the width of the byte (number of bits).
3. Ao or Qo where o represents the off-set designator for the A or Q register i. e., the rightmost bit position of the byte in A or Q (one of them must be specified).

With this information let's present a case, and then show the instruction (s) needed.

Suppose we wish to transmit the upper address portion of a memory word (address M) to the lower address portion of A. What is the instruction (s) needed?



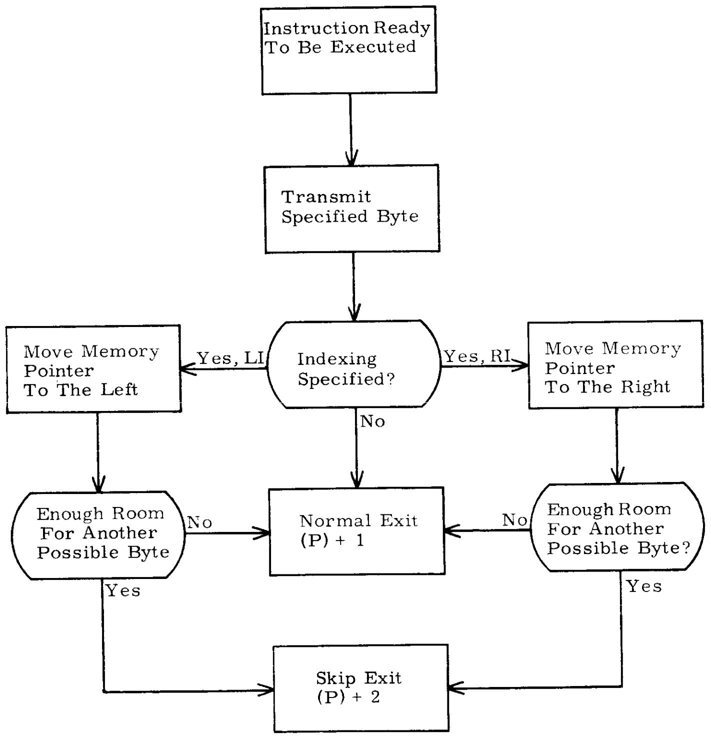
The LBYT instruction is needed here and the instruction set looks like the following:

```
ENI              24, 1
LBYT, A0, E15     M,, 1
```

The byte is inserted into the lower portion of A and the rest of A remains unchanged. If the programmer would like the rest of A cleared, he can specify CL in the operation field. CL will clear the destination word as the byte is transferring. After the LBYT, a full exit is taken to the next instruction.

How would you transmit the byte in the reverse direction? The only change would be SBYT instead of LBYT.

Now what about LEFT INDEX and RIGHT INDEXING? If LI or RI is specified, the memory off-set designator becomes a movable pointer and is automatically checked every time the instruction is executed. A general flowchart will help clear up this concept. One instruction will check all of the following:



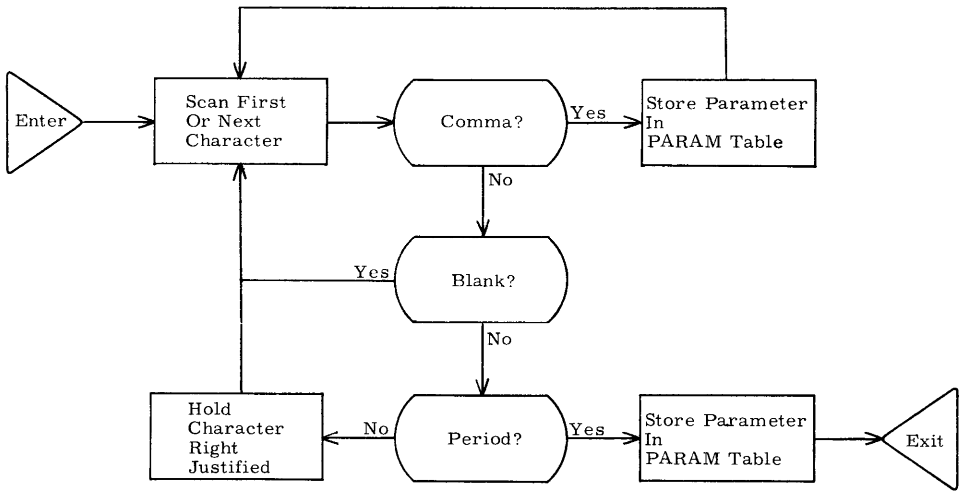
Instruction Ready To Be Executed
Transmit Specified Byte
Move Memory Pointer To The Left
Yes, LI
Indexing Specified?
Yes, RI
Move Memory Pointer To The Right
No
Enough Room For Another Possible Byte
No
Normal Exit (P) + 1
No
Enough Room For Another Possible Byte?
Yes
Yes
Skip Exit (P) + 2

Everything in the flowchart is checked for in the hardware. The reason for normal exiting or skip exiting when indexing is specified, is that the normal exit allows the programmer to re-establish the memory pointer (W) and to advance or decrement the relative address counter (B^b) before going on with the program.

Problem 17:

Assume a card image is in memory starting at address CARD. The card is in internal BCD and takes up 10 memory locations. Assume a set of parameters on the card, each 1-8 characters, each separated by a comma, and the set terminated by a period. Write a subprogram that will store each parameter, right justified with zero fill starting at address PARAM. Ignore blanks and assume less than 20 parameters. The internal BCD code for a blank is 60B, a comma is 73B, and a period is 33B.

Flowchart:


Enter
Scan First Or Next Character
Comma?
Yes
Store Parameter In PARAM Table
No
Blank?
Yes
No
Period?
No
Hold Character Right Justified
Yes
Store Parameter In PARAM Table
Exit

Problem 17 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	PARAMCNT	
	ENTRY	PARAMCNT	
PARAM	BSS	20	
PARAMCNT	BSS	1	
	ENA	0	
	ENI	0,1	
	ENI	0,2	
	ENI	42,3	
READCHAR	LBYT,Q0,EQ,RI,CL	CARD,2,3	
	ENI	42,3	EXECUTED ØNLY WHEN FINISHED
	INI	1,2	WITH A MEMØRY WØRD
	RGJP,EQ	Q,73B,CØMMA	CØMMA
	RGJP,EQ	Q,60B,*-3	BLANK, READ NEXT CHARACTER
	RGJP,EQ	Q,33B,PERIØD	PERIØD
	QLS	42	
	LLS	6	CHARACTER TØ A
	SLJ	READCHAR	
CØMMA	STA	PARAM,1	STØRE PARAMETER
	INI	1,1	
	ENA	0	
	SLJ	READCHAR	
PERIØD	STA	PARAM,1	
	SLJ	PARAMCNT	
	END		

Somewhere within this subprogram would also be included the symbol CARD in the location field with a declaration of the prestored data or area reserved.

Student Problem 17A:

Assume the same problem as just given except the card image was read in backwards. In other words column 1 is the last character at address CARD+9 and column 80 is the first character at address CARD.

Flowchart:

Problem 17A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 18

INPUT/OUTPUT

GROUP 18

INPUT/OUTPUT

1.	Connect	CONN
2.	External Function	EXTF
3.	Begin Read	BEGR
4.	Begin Write	BEGW
5.	Copy Status	COPY
6.	Clear Channel	CLCH
7.	Input to A	IPA

This group of instructions works with the I/O equipment in the system.

The first four instructions contain a "reject jump address" in case the operation cannot be performed. The first instruction opens a line from a data channel to a unit. The second instruction performs any one of a number of operations on the unit except reading or writing. The third and fourth instructions actually initiate the read or write operation which is completely buffered.

The fifth instruction interrogates the operation on a data channel and can do so at any time. The sixth instruction clears all channel control and terminates any operation being performed on the channel.

The last instruction is the only 24-bit instruction in the group. It inputs information from either the typewriter or card reader, whichever is selected (manual). The input is not buffered, i.e., program control will stop until the information is received.

The CONNECT Instruction

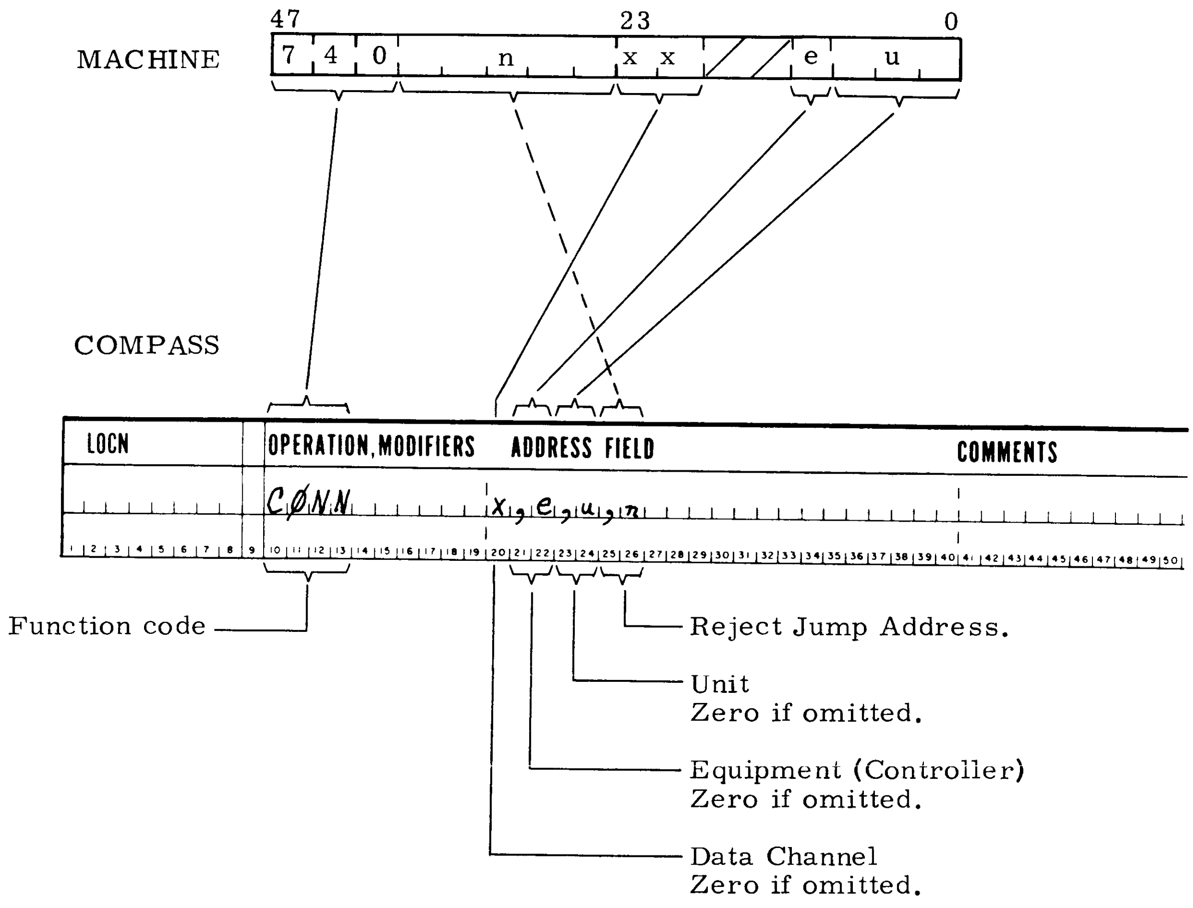
The CONNECT instruction is an instruction that connects one controller and one unit of that controller to a data channel.

The designator x specifies the number of the data channel. The designator e specifies the number of the equipment (controller). The designator u specifies the number of the unit. The designator n specifies the Reject Jump Address if the connection cannot be made.

If the connection can be made, it will be made and program control will go to the next instruction at (P) + 1.

If the channel or unit is busy, program control will transfer to address n.

If the channel or controller or unit is not present, an interrupt condition occurs. This is called the Internal Reject Interrupt. After going through the interrupt routine, program control will then go to address n.

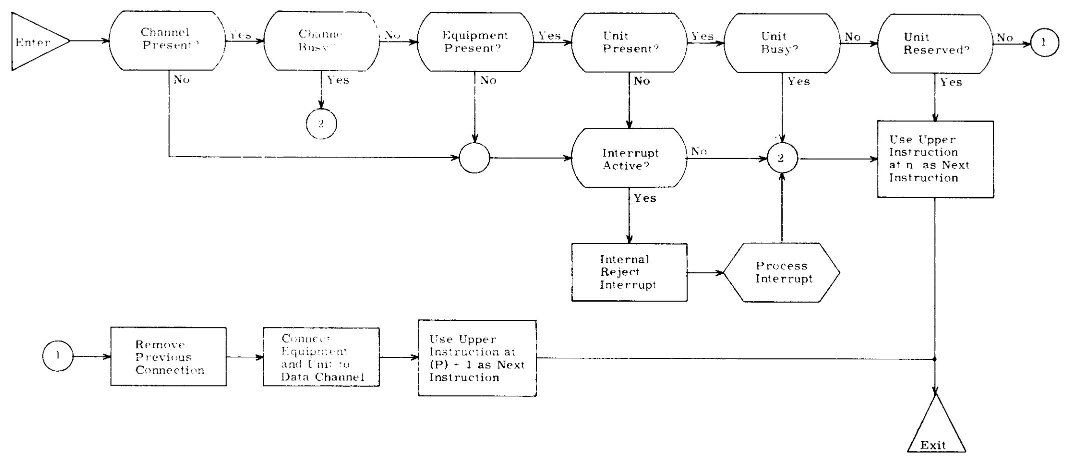


MACHINE
47
23
0
7 4 0
n
x x
e
u
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
CØNN
x, e, u, n
Function code
Reject Jump Address.
Unit
Zero if omitted.
Equipment (Controller)
Zero if omitted.
Data Channel
Zero if omitted.

FLOWCHART

CONNECT

FORMAT: CONN x, e, u n


Enter
Channel Present?
Yes
No
Channel Busy?
No
Yes
2
Equipment Present?
Yes
No
Unit Present?
Yes
No
Unit Busy?
No
Yes
Unit Reserved?
No
Yes
1
Interrupt Active?
No
Yes
2
Use Upper Instruction at n as Next Instruction
Internal Reject Interrupt
Process Interrupt
1
Remove Previous Connection
Connect Equipment and Unit to Data Channel
Use Upper Instruction at (P) + 1 as Next Instruction
Exit

DESCRIPTION

CONN Connect a unit to Data Channel X

PROBLEM:

Connect to Data Channel 2, equipment 3 and unit 5. If the connection cannot be made, jump to address REJ.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CØNN	2,3,5,REJ	

PROBLEM:

Connect to Data Channel 0, equipment 6 and unit 4. If the connection cannot be made, jump to address BUS.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CØNN	,6,4,BUS	

PROBLEM:

Connect to Data Channel 1, equipment 3. If the connection cannot be made, jump to address PETE.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CØNN	1,3,,PETE	

PROBLEM:

Connect to Data Channel 3, equipment 2 and unit 4. If the connection cannot be made, have the instruction reject upon itself.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CØNN	3,2,4,*	

The EXTERNAL FUNCTION Instruction

The EXTERNAL FUNCTION instruction is an instruction that performs some type of setting or operation on a unit. This includes everything that can be done to the unit except the transmission of data.

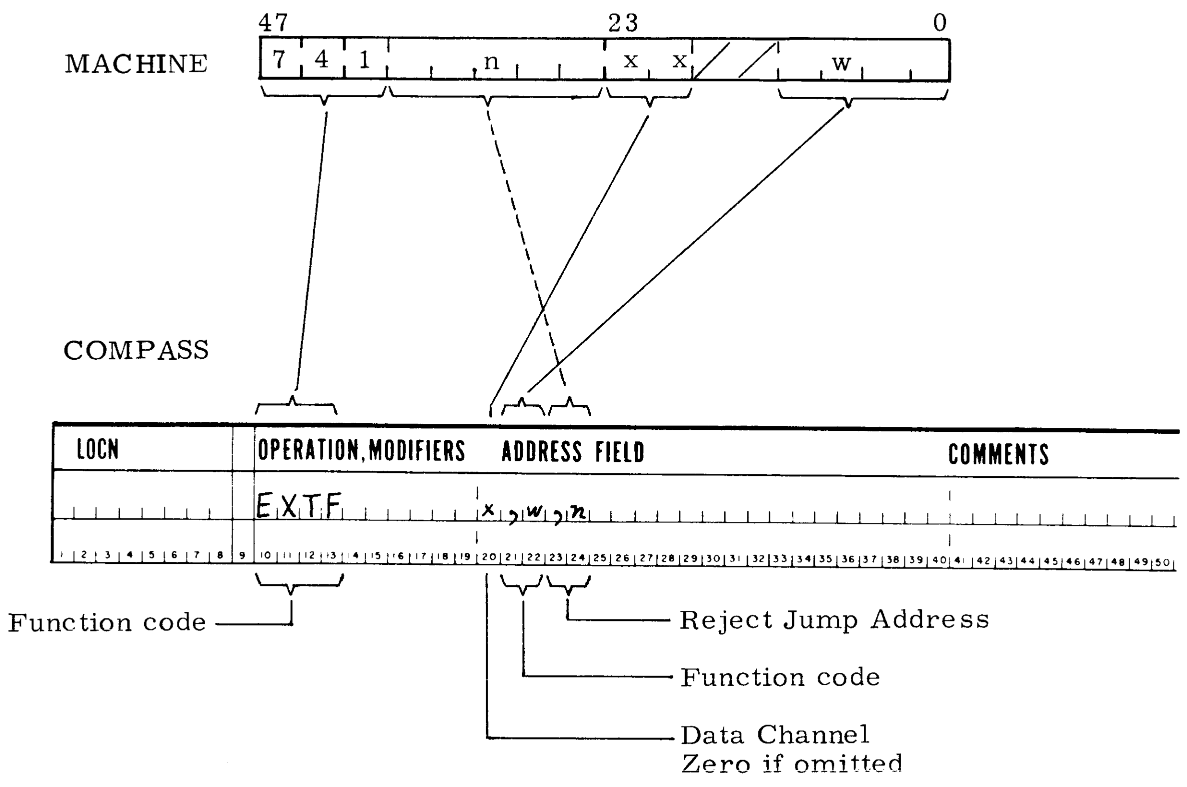
The designator w specifies the function code. For each operation there is a 12-bit function code which this instruction transmits to the unit. The function code specifying an operation for any particular unit can be found in its reference manual.

The designator x specifies the data channel through which the function code is to be transmitted to the unit.

The designator n specifies the Reject Jump Address if for some reason the function cannot take place.

If the channel or controller or unit is not present, an interrupt condition occurs. This is called the Internal Reject Interrupt. After going through the interrupt routine, program control will then go to address n.

If bit 23 is a "1" within the format of this instruction, the function code is transmitted only to the specified data channel. For these special function codes see the 3600 Computer System Reference Manual.

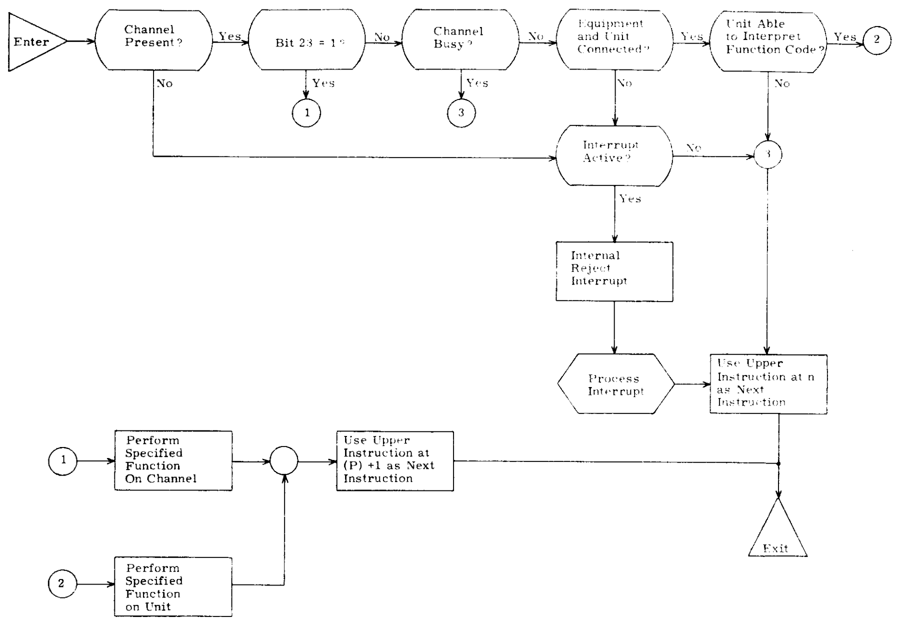

MACHINE
47
23
0
7
4
1
n
x
x
w
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
EXTF
x,w,n
Function code
Reject Jump Address
Function code
Data Channel
Zero if omitted

EXTF

FLOWCHART

EXTERNAL FUNCTION

FORMAT: EXTF x, w, n



Enter
Channel Present?
Yes
No
Bit 23 = 1?
No
Yes
1
Channel Busy?
No
Yes
3
Equipment and Unit Connected?
Yes
No
Unit Able to Interpret Function Code?
Yes
2
No
Interrupt Active?
No
3
Yes
Internal Reject Interrupt
Process Interrupt
Use Upper Instruction at n as Next Instruction
1
Perform Specified Function On Channel
Use Upper Instruction at (P) +1 as Next Instruction
2
Perform Specified Function on Unit
Exit

DESCRIPTION

EXTF — Perform some function on the unit connected to Data Channel X.

EXAMPLES

PROBLEM:

Assuming XXXX to be the octal function code rewinding tape to load point, execute the instruction that will do so if the unit is connected to Data Channel 3. If the operation cannot be performed, jump to address ZILCH.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EXTF	3,XXXXB,ZILCH	

PROBLEM:

A magnetic tape unit is connected to Data Channel 2. Set the unit to BCD mode if YYYY octal is the function code. If the operation cannot be performed, jump to address NIX.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EXTF	2,YYYYB,NIX	

PROBLEM:

A magnetic tape unit is connected to Data Channel 2. Set the unit to 556 density if VVVV octal is the function code. If the operation cannot be performed, jump to address JACK.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EXTF	2,VVVVB,JACK	

PROBLEM:

A magnetic tape unit is connected to Data Channel 2. Set the unit to Interrupt On Error if TTTT octal is the function code. If the operation cannot be performed, jump to address CONT.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EXTF	2,TTTTB,CØNT	

The BEGIN READ Instruction

The BEGIN READ instruction is an instruction that initiates transmitting of data words from the unit to memory.

The data will be read from the unit which is connected to the data channel specified by x.

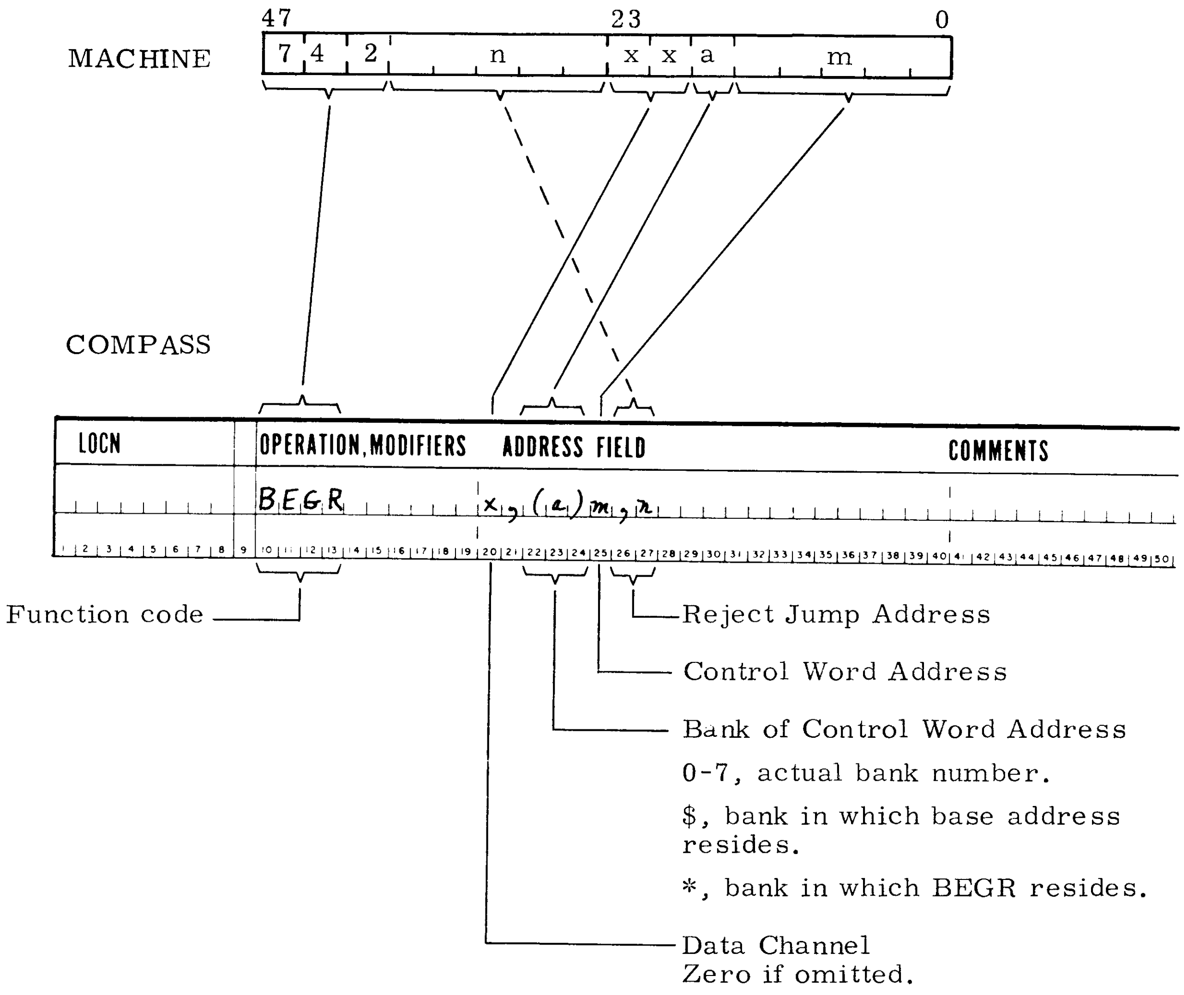
The designator (a) m specifies an 18-bit address called the Control Word Address. At this address is pre-stored a control word for the data channel. The control word is composed of an 18-bit starting address specifying where in memory the data is to be transmitted, and a word count telling how many 48-bit words are to be transmitted. A special feature of the control word allows the programmer to "skip" records and to read variable length records without having to know how long each one is. For a more detailed explanation of these special features see the 3600 Computer System Reference Manual.

If the equipment is capable of being read from, the read is initiated and program control goes to (P) + 1.

The designator n specifies the Reject Jump Address if for some reason the read cannot take place.

If the channel or controller or unit is not present, an interrupt condition occurs. This is called the Internal Reject Interrupt. After going through the interrupt routine, program control will then go to address n.

FORMAT

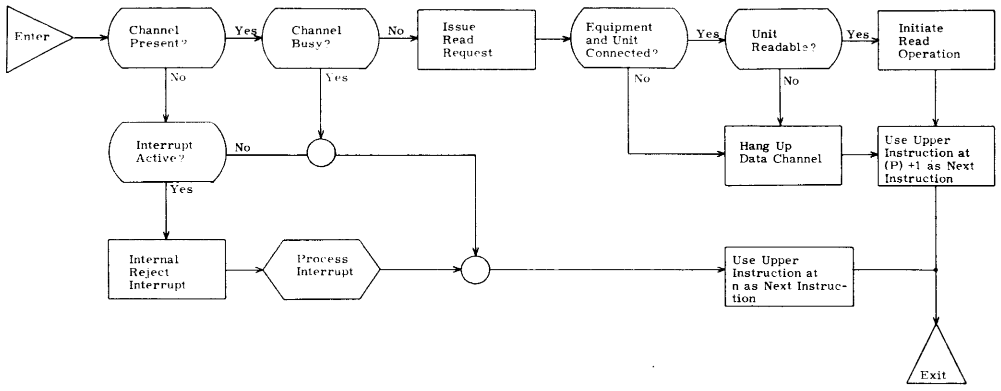


MACHINE
47
23
0
7 4 2
n
x x a
m
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
BEGR
x,(a)m,n
Function code
Reject Jump Address
Control Word Address
Bank of Control Word Address
0-7, actual bank number.
$, bank in which base address resides.
*, bank in which BEGR resides.
Data Channel
Zero if omitted.

BEGR

FLOWCHART

BEGIN READ FORMAT: BEGR x, (a) m, n



DESCRIPTION

BEGR Activate a read on Data Channel X

PROBLEM:

Activate a read on Data Channel 3 using control word address CWA1 from Bank 2. If the Read cannot take place, jump to address BUS.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BEGR	3,(2)CWA1,BUS	

PROBLEM:

Activate a read on Data Channel 0 using the control word from address CWA2 in the bank where CWA2 resides. If the Read cannot take place, jump to address REJ.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BEGR	,(#)CWA2,REJ	

PROBLEM:

Activate a read on Data Channel 1 using the control word from address CWA3 in the bank where the BEGR resides. If the Read cannot take place, jump to the address of the instruction.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BEGR	1,(*)CWA3,*	

PROBLEM:

In reference to the above problem, what would the control word at address CWA3 look like if we wanted to read the first record into address FWA3 in the bank where the BEGR resides? Assume first record not over 50 words.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IØTR	(*)FWA3,50	

The BEGIN WRITE Instruction

The BEGIN WRITE instruction is an instruction that initiates transmitting of data words from memory to the unit.

The data will be transmitted to the unit which is connected to the data channel specified by x.

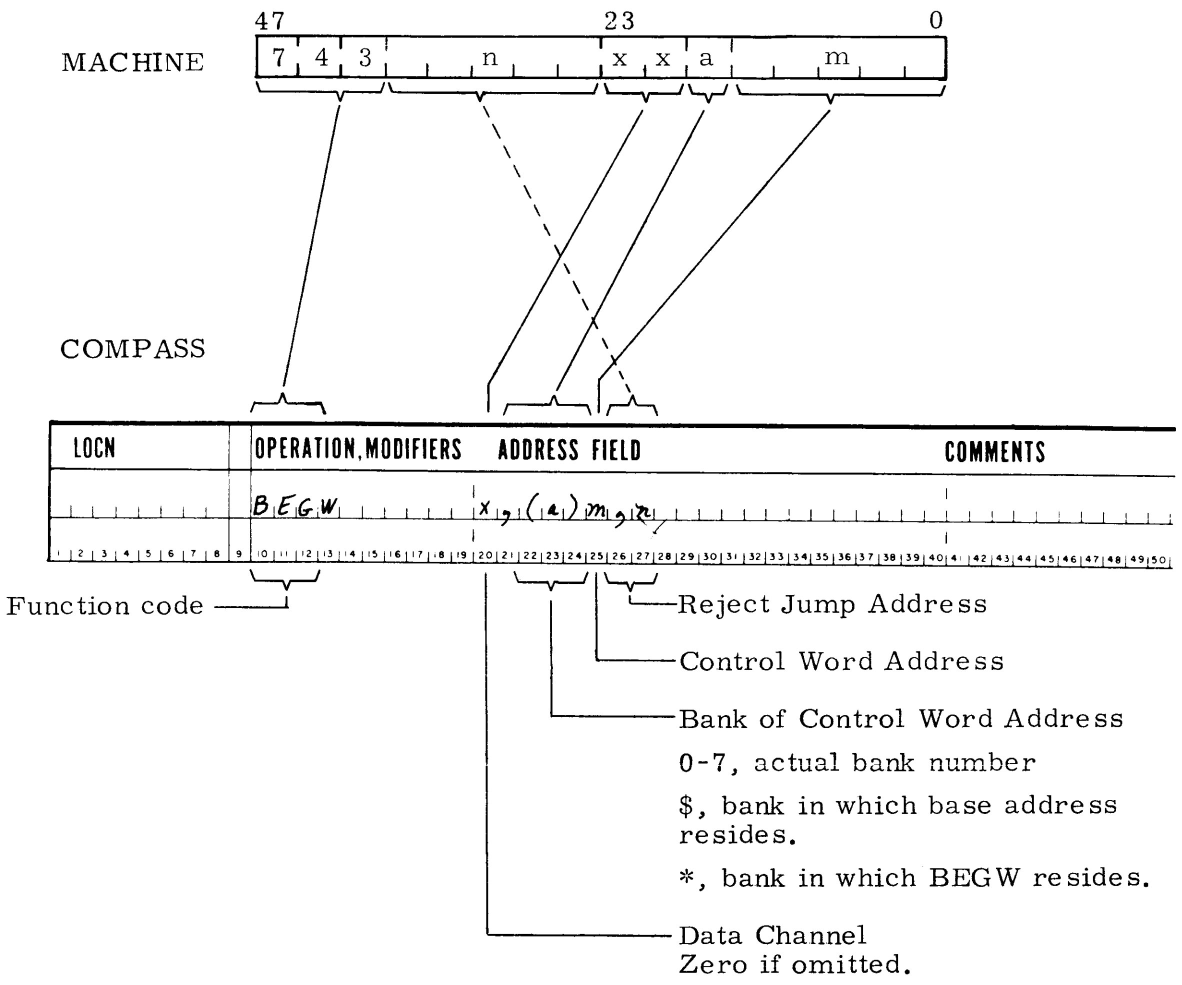
The designator (a) m specifies an 18-bit address called the Control Word Address. At this address is pre-stored a control word for the data channel. The control word is composed of an 18-bit starting address specifying where in memory the data is to be transmitted from, and a word count telling how many 48-bit words are to be transmitted.

If the equipment is capable of being written to, the write is initiated and program control goes to (P) + 1.

The designator n specifies the Reject Jump Address if for some reason the write cannot take place.

If the channel or controller or unit is not present, an interrupt condition occurs. This is called the Internal Reject Interrupt. After going through the interrupt routine, program control will then go to address n.

FORMAT

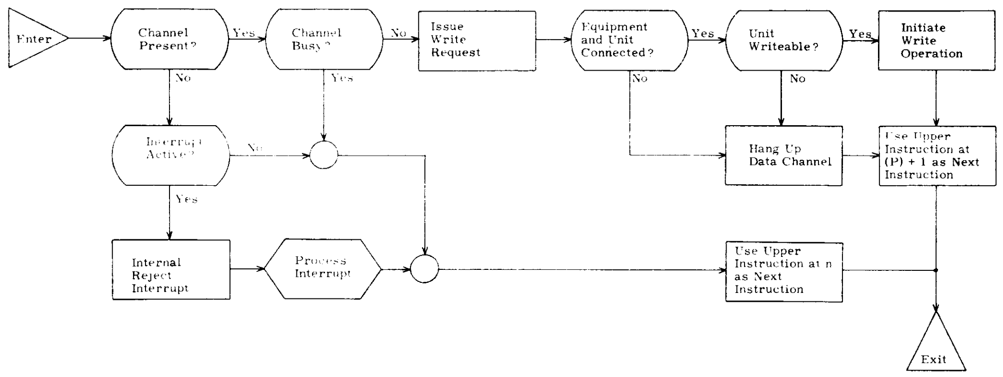


MACHINE
47
23
0
7 4 3
n
x x a
m
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
BEGW
x,(a)m,n
Function code
Reject Jump Address
Control Word Address
Bank of Control Word Address
0-7, actual bank number
$, bank in which base address resides.
*, bank in which BEGW resides.
Data Channel
Zero if omitted.

BEGW

FLOWCHART

BEGIN WRITE FORMAT: BEGW x, (a) m, n



DESCRIPTION

BEGW Activate a write on Data Channel X

EXAMPLES

PROBLEM:

Activate a Write on Data Channel 4 using control word address CWA1 from Bank 3. If the Write cannot take place, jump to address REJ.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	BEGW	4,(3) CWA1,REJ	

PROBLEM:

Activate a Write on Data Channel 2 using the control word at address CWA2 in the bank where CWA2 resides. If the Write cannot take place, jump to address MOD.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	BEGW	2,($) CWA2,MØD	

PROBLEM:

Activate a Write on Data Channel 1 using the control word at address CWA3 in the bank where the BEGW resides. If the Write cannot take place, jump to address NIX.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	BEGW	1,(*) CWA3,NIX	

PROBLEM:

In reference to the above problem, what would the control word at address CWA3 look like if we wanted to write 20 words onto tape from address FWA3 in the bank where BEGW resides?

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IØTW	(*) FWA3,20	

The COPY STATUS Instruction

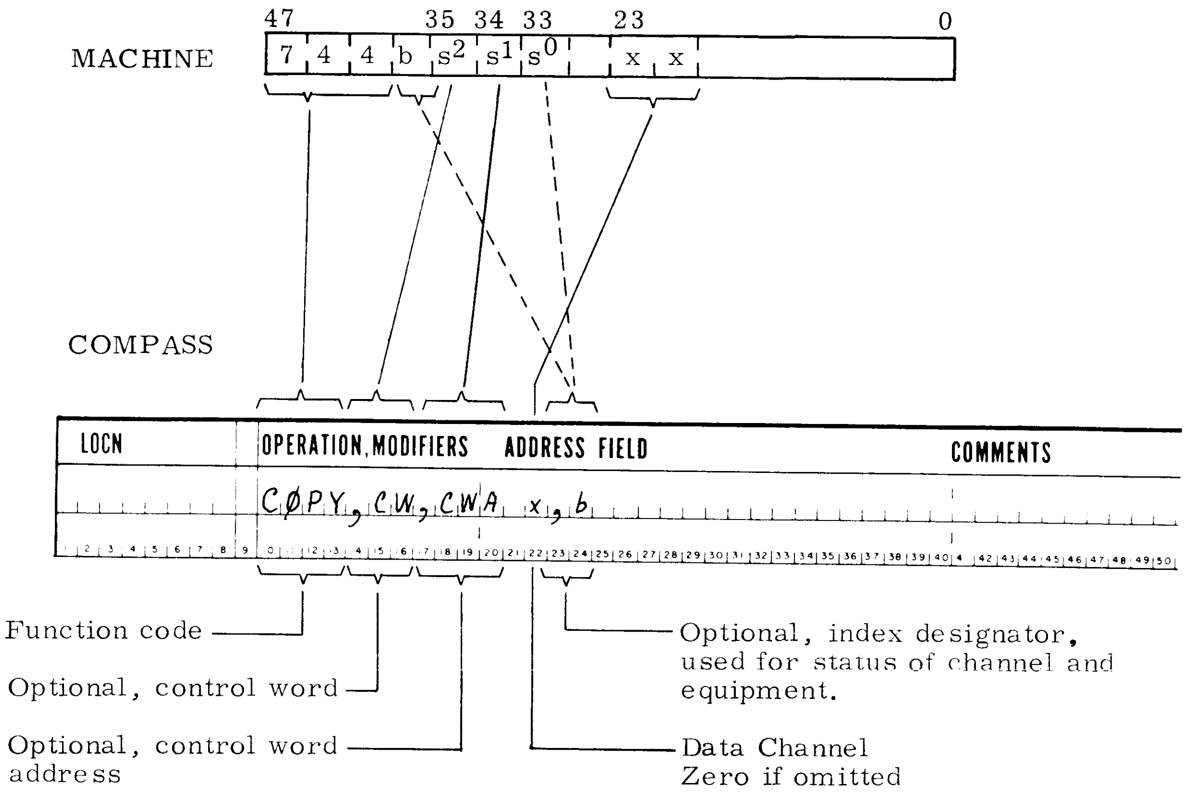
The COPY STATUS instruction is an instruction that probes the status of a data channel and the connected unit for various conditions.

The designator x specifies the data channel through which the status is to be probed.

If CW is specified by the programmer, the present Control Word will replace the contents of A with its present word count and its current word address. If CWA is specified by the programmer, the present Control Word Address will replace the contents of Q.

If b is specified by the programmer, the status of the equipment will replace the contents of that index register. To interpret this status the equipment reference manual must be referenced.

Program control always goes to (P) + 1 from this instruction since it is only a "probing" instruction.

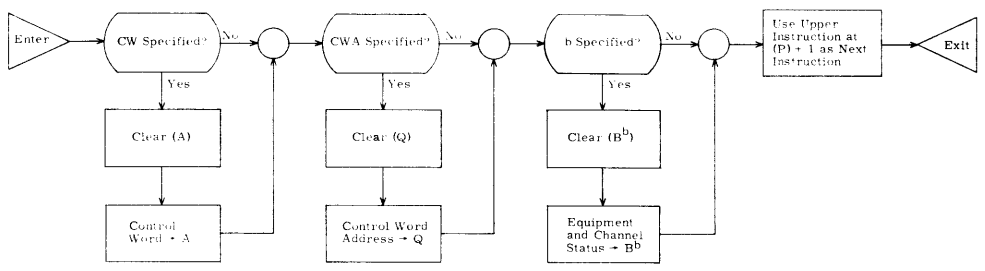


MACHINE
47
35
34
33
23
0
7
4
4
b
s2
s1
s0
x
x
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
CØPY, CW, CWA x, b
Function code
Optional, control word
Optional, control word address
Optional, index designator, used for status of channel and equipment.
Data Channel Zero if omitted

COPY

FLOWCHART

COPY STATUS FORMAT: COPY, CW, CWA x, b



DESCRIPTION

COPY Examine status

a. Control Word Address → Q

b. Control Word → A

c. Equipment Status → B^b

E
X
A
M
P
L
E
S

PROBLEM:

Probe the equipment status from Channel 2. Use Index Register 3.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		CØPY	2,3	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Probe the control word from Channel 1.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		CØPY,CW	1	
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Probe the control word address from Channel 0.

SOLUTION:

LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		CØPY,CWA		
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

PROBLEM:

Probe all possible status from Channel 3. Use Index Register 6 for equipment status.

SOLUTION:

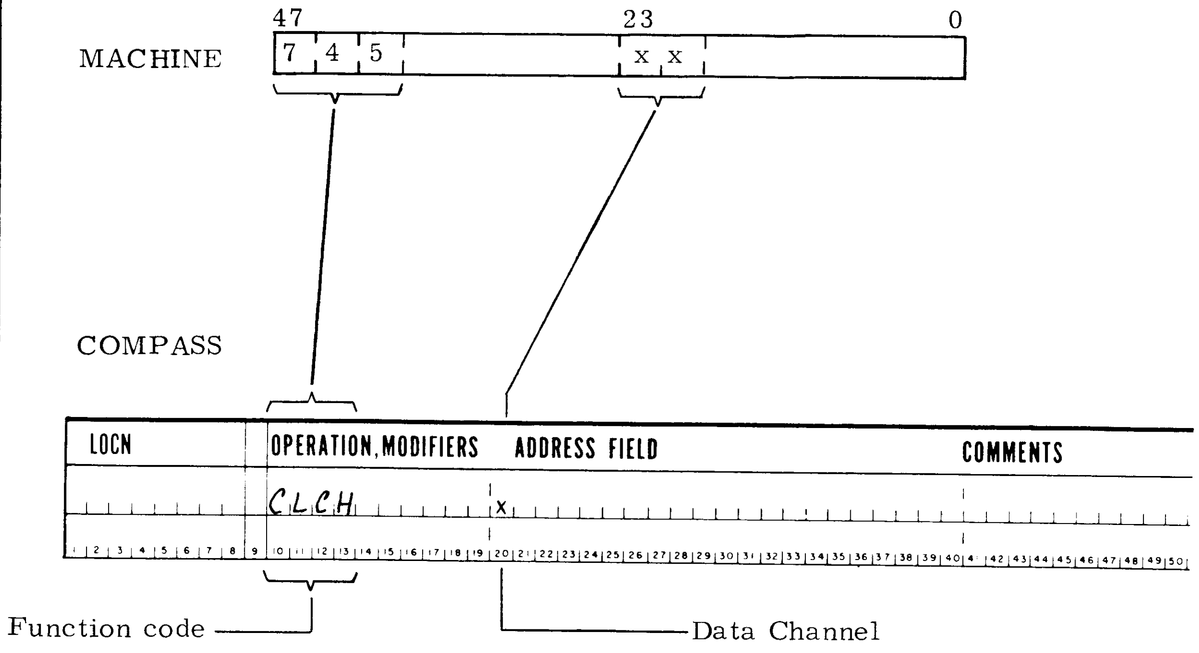
LOCN		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		CØPY,CW,CWA 3,6		
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50

The CLEAR CHANNEL Instruction

The CLEAR CHANNEL instruction is an instruction that clears the data channel specified by x. By so doing, all control is removed from the data channel. Any I/O transfer is terminated. The connection between the data channel and the unit is lost.

If the programmer wishes to use a unit on data channel x after using this instruction, he must re-connect the unit with the CONNECT instruction.

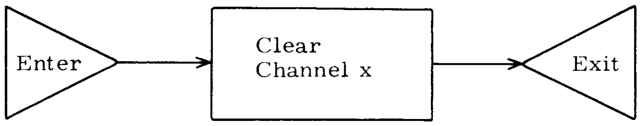
FORMAT


MACHINE
47
23
0
7
4
5
x x
COMPASS
LOCN
OPERATION, MODIFIERS
ADDRESS FIELD
COMMENTS
CLCH
x
Function code
Data Channel

FLOWCHART

CLEAR CHANNEL

FORMAT: CLCH x



DESCRIPTION

CLCH Clear channel X

a. Terminate activity

b. Remove control

c. Remove connection

E X A M P L E S

PROBLEM:

Clear Channel 0.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CLCH	0	

PROBLEM:

Clear Channel 1.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CLCH	1	

PROBLEM:

Clear Channel 2.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CLCH	2	

PROBLEM:

Clear Channel 3.

SOLUTION:

LOCN	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CLCH	3	

The INPUT TO A Instruction

The INPUT TO A instruction is an instruction that allows information to be transmitted to the A register via the typewriter or the card reader. No data channel is used by this instruction.

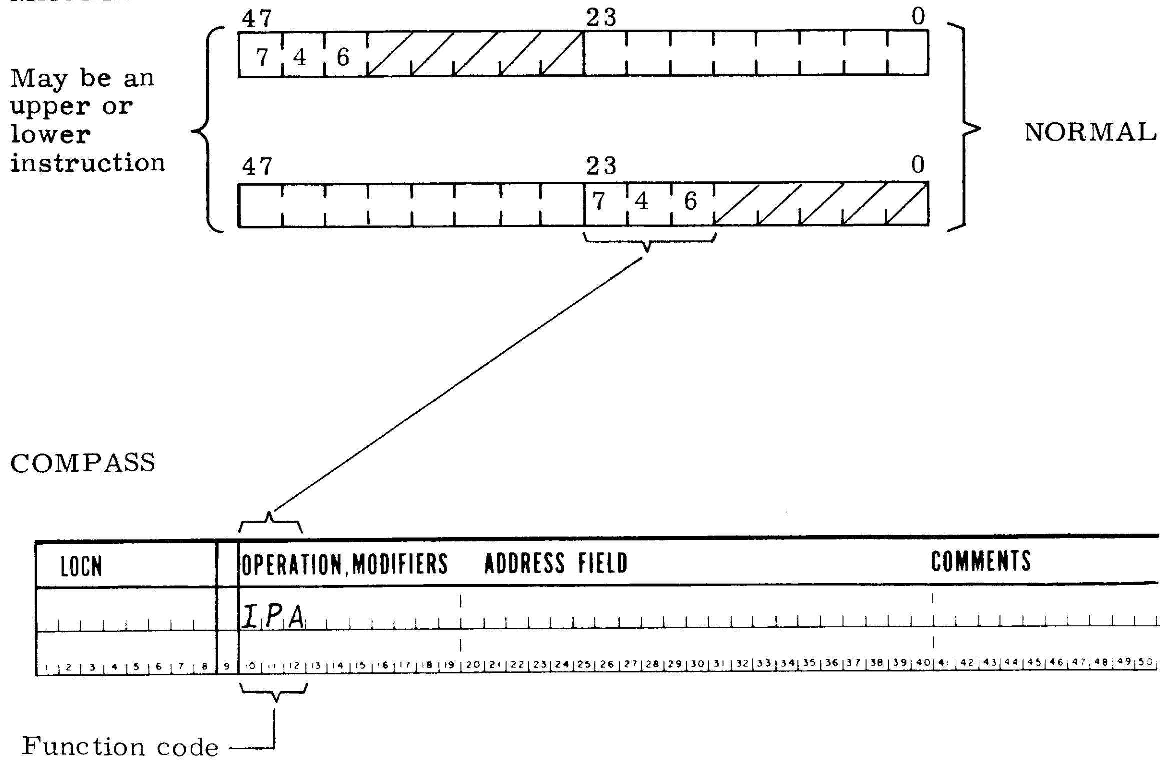
When this instruction is encountered, the computer will wait for the typewriter or card reader (selection on console) to issue one 6-bit character (typewriter) or one column (card reader). The information is transferred to the low order bits of A and program control continues to the next instruction.

INPUT TO A

IPA

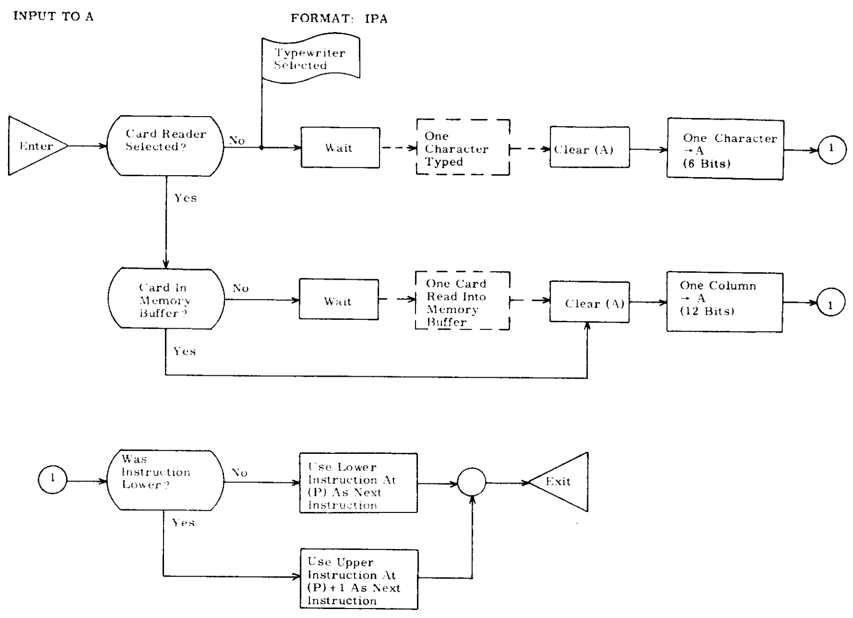
FORMAT

MACHINE



IPA

F L O W C H A R T



D E S C R I P T I O N

IPA

1) One character → A for typewriter
2) One column → A for card reader

PROBLEM:

Card reader is selected. Do an Input To A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IPA		

PROBLEM:

Typewriter is selected. Do an Input To A.

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IPA		

PROBLEM:

Which I/O instruction will hang up the computer?

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IPA		

PROBLEM:

Which I/O instruction is only 24 bits?

SOLUTION:

LOCN	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IPA		

NEW CONCEPTS OF GROUP 18

The concept of the I/O programming has already been introduced in Volume I, Section III with problem 19. There we included the concept with explanation of the CONN, EXTF, BEGR, and BEGW instructions. We also mentioned the control words needed with the BEGR and BEGW instructions.

At this time we want to give a deeper insight into I/O operations and discuss the interdependency between I/O and interrupt. How do you set up an operation? How do you monitor it? What do you do when it's finished? In order not to confuse the discussion we will assume a 362X magnetic tape controller and a 60X tape unit for the equipment, and the principle will be similar for all other equipment.

The function codes peculiar to magnetic tape are the following:

TAPE MOTION			
Rewind	0010	Search E.O.F. Backward	0014
Rewind Unload	0011	Write E.O.F.	0015
Backspace	0012	Skip Bad Spot	0016
Search E.O.F. forward	0013		
FORMAT			
Release	0000	Clear	0005
Binary	0001	800 BPI Density	0006
BCD	0002	Clear Reverse Read	0040
556 BPI Density	0003	Set Reverse Read	0041
200 BPI Density	0004		
INTERRUPT			
Interrupt On Ready And Not Busy	0020	Release Interrupt On End of Operation	0023
Release Interrupt On Ready And Not Busy	0021	Interrupt on Abnormal End of Operation	0024
Interrupt On End Of Operation	0022	Release Interrupt on Abnormal End of Operation	0025

These function codes are 12 bit codes generated by the central processor when the EXTF instruction is executed. All are in octal form.

None of these codes transmit data. The codes have the following meanings:

TAPE MOTION

1. Rewind (0010) -	This code rewinds tape at high speed to load point. It has no effect if the tape is already at load point.
2. Rewind Unload (0011) -	This code rewinds tape at high speed to load point. It then rewinds tape slowly until all of the tape is on the supply reel. The tape must be manually re-loaded in order for any further operations to occur on it.
3. Backspace (0012) -	This code backspaces the tape one record or until load point is detected. If Reverse Read (607 only) is also selected, the tape will forward space one record.
4. Search E. O. F. Forward(0013)-	This code advances tape until the next file mark is detected.
5. Search E. O. F. Backward(0014)-	This code backspaces tape until a file mark is detected.
6. Write E. O. F. (0015) -	This code advances the tape approximately 6 inches and writes a BCD 17_8. This does not affect the recording mode.
7. Skip Bad Spot (0016) -	This code erases approximately 6 inches of tape.

FORMAT

1. Release (0000) -	This code logically disconnects the connected unit and removes the reserve so that other data channels through a multi-channel controller may have access to it.
2. Binary (0001) -	This code causes all data read or written to be done so in binary notation.
3. BCD (0002) -	This code causes all data read or written to be done so in BCD notation.
4. 556BPI Density (0003) -	This code causes all data read or written to be done so at 556 frames per inch.
5. 200BPI Density (0004) -	This code causes all data read or written to be done so at 200 frames per inch.
6. Clear (0005) -	This code clears all reservations made through the data channel and logically disconnects the connected unit.
7. 800BPI Density (0006) -	This code causes all data read or written to be done so at 800 frames per inch. This is possible on the 607 only.
8. Clear Reverse Read (0040) -	This code clears the condition set by the Set Reverse Read code.
9. Set Reverse Read (0041) -	This code set up the condition that any tape read will occur in the reverse direction.

INTERRUPT

1. Interrupt on Read - And Not Busy (0020)	This code causes the controller to send an interrupt signal to the processor when the tape unit is in a Ready and Not Busy condition (all tape motion has ceased). This signal is cleared by a 0021 code.
2. Release Interrupt on Ready - And Not Busy (0021)	This code clears an Interrupt on Ready and Not Busy selection and signal if it is up.
3. Interrupt on End- Of Operation (0022)	This code causes the controller to send the interrupt signal when an End of Record is sensed.
4. Release Interrupt on End - Of Operation (0023)	This code clears the Interrupt on End of Operation selection and signal if it is up.
5. Interrupt on Abnormal - End of Operation (0024)	This code causes the controller to send an interrupt to the processor after an abnormal condition occurs For magnetic tape these conditions are: a. File Mark b. Load Point c. Vertical Parity Error d. Longitudinal Parity Error e. Lost Data f. Parity Error During Skip Bad Spot g. Connected Tape Unit Becoming Not Ready
6. Release Interrupt On - Abnormal End of Operation (0025)	This code clears the Interrupt on Abnormal End of Operation selection and signal if it is up.

The INTERRUPT function codes are common to all controllers with only the "abnormal conditions" listed being different. All codes listed are used to position tape and set operating mode before actual transfer of data takes place

The condition of the unit can be monitored at any time through the use of the COPY (status) instruction. The unit status is received in the lowest 12 bits of an index register, each bit representing a condition. A "1" bit means the condition is present as follows:

STATUS REPLIES			
Bit 0	Ready	Bit 6	Density (556 or 200BPI)
Bit 1	Channel, Control or Unit Busy	Bit 7	Density (800BPI)
Bit 2	Write Enable	Bit 8	Lost Data
Bit 3	File Mark	Bit 9	Longitudinal Parity Error
Bit 4	Load Point	Bit 10	Vertical Parity Error
Bit 5	End Of Tape	Bit 11	Reserve Reject

The status can be checked by the "Bit Sensing" instruction (NBJP or ZBJP). This instruction can examine any bit in any operational register.

On the COPY instruction the 12 bits are received in the lower portion of the index register. The uppermost three bits of the register will contain channel status as follows:

Bit 14 - "1", transmission parity error has occurred.
This is between memory and channel and between channel and controller only.

Bit 13 - "1", write operation is in progress.

Bit 12 - "1", read operation is in progress.

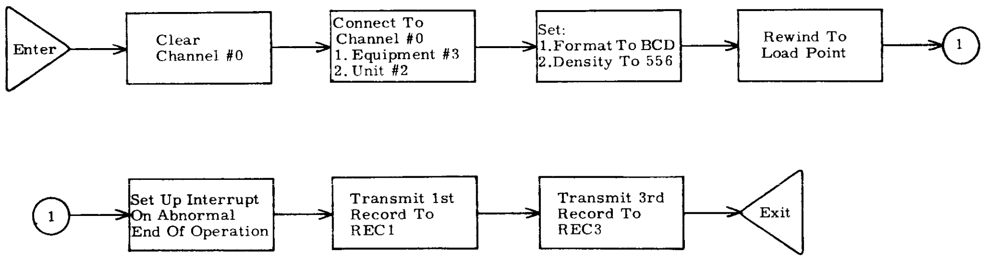
Problem 18:

Write a subprogram that will do the following:

1. Clear channel #0 .
2. Connect equipment #3, unit #2 (mag. tape) to channel #0.
3. Set format to BCD, density to 556.
4. Rewind to load point.
5. Set up Interrupt On Abnormal End of Operation .
6. Transmit the 1st record to address REC1.
7. Transmit the 3rd record to address REC3.
8. When finished, exit.

Assume the records are each less than 100 words.

Flowchart:


Enter
Clear Channel #0
Connect To Channel #0 1. Equipment #3 2. Unit #2
Set: 1.Format To BCD 2.Density To 556
Rewind To Load Point
1
1
Set Up Interrupt On Abnormal End Of Operation
Transmit 1st Record To REC1
Transmit 3rd Record To REC3
Exit

Problem 18 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	READINT	
	ENTRY	READINT	
REC1	BSS	100	
REC3	BSS	100	
CWAI	IØTR,C	REC1,100	
	IØSR,C	0,100	
	IØTR	REC3,100	
READINT	BSS	1	
	CLCH	0	
	CØNN	0,3,2,*	
	EXTF	0,2,*	SET FØRMAT TØ BCD
	EXTF	0,3,*	SET DENSITY TØ 556
	EXTF	0,10B,*	REWIND
	EXTF	0,24B,*	ABNØRMAL END ØF ØPERATIØN
	BEGR	0,CWAI,*	TRANSFER DATA
	CØPY	0,1	STATUS TØ B1
	WBJP	B1,12,*-1	LØØP IF STILL READING
	SLJ	READINT	FINISHED
	END		

Student Problem 18A:

Write a subprogram that will do the following:

1. Clear channel #2.
2. Correct equipment #4, unit #1 (mag. tape) to channel #2.
3. Set format to BIN, density to 200.
4. Rewind to load point.
5. Set up Interrupt On Abnormal End of Operation.
6. Write 20 words starting at CARDBUFF as one record.
7. When finished, exit.

Flowchart:

Problem 18A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

GROUP 19

LOCAL STORAGE ALLOCATION

GROUP 19

LOCAL STORAGE ALLOCATION

1.	Block Storage Reservation (Starting)	BSS
2.	Block Storage Reservation (Ending)	BES
3.	Octal Data Initialization	OCT
4.	Decimal Data Initialization	DEC
5.	Double Precision Data Initialization	DECD
6.	Binary-Coded-Decimal Data Initialization	BCD
7.	Type Character Initialization	TYPE
8.	Variable Field Data Initialization	VFD
9.	Literals	= MV
		= S

The first two instructions allow the programmer to declare a block of one or more words to be reserved before execution of the program. During the execution of the program, the programmer generates words and stores them in this area.

The next three instructions allow the programmer to prestore data before execution of the program. Constants are usually prestored with one of these instructions.

The BCD and TYPE statements allow the programmer to prestore messages that are to be output sometime during the execution of the program.

The VFD statement prestores data in variable bytes. The "string" of data can be of different modes.

The last item, "literals", is not a statement but a different method used to allocate storage words. Since this is a new concept, more is said on it further in this group.

The BSS Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
l	BSS	n	

This instruction reserves a block of consecutive 48-bit machine words. The value of the expression in the address field determines the number of words to be reserved. A location symbol is optional. If present, it is assigned to the first word reserved. In this case the programmer would use forward addressing.

The area reserved will not be zeroed out prior to execution. If the programmer so desires, he must do it within his program.

The BES Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
l	BES	n	

This instruction is the same as the BSS instruction, except that the symbol in the location field, if any, is assigned to the last word of the block. In this case the programmer would use backward addressing.

The OCT Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
ℓ	ØCT	C1,C2,C3,...	

This instruction is used to prestore octal constants. Constants are declared in the address field separated by commas. Each constant requires one 48-bit word.

The constants can be 1 to 16 octal digits, signed or unsigned. If a constant is less than 16 digits, it will be right justified in the word with the sign extended.

The character B̲ is not allowed. The assembler automatically assumes octal. A location symbol is optional. If present, it is assigned to the first word.

The following are valid examples of an OCT instruction:

```
CØNX          ØCT      1, 456, -7, 0, -0

(CØNX)        = 0———> 01
(CØNX +1)     = 0———>0456
(CØNX +2)     = 7———> 70
(CØNX +3)     = 0———> 0
(CONX +4)     = 7———> 7
```

The DEC Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
ℓ	DEC	C1, C2, C3, . . .	

This instruction is used to prestore decimal constants. Constants are declared in the address field separated by commas. Each constant requires one 48-bit word.

Constants are prestored in two different forms with this instruction: fixed point integer and floating point. If no decimal point is found in the constant, it is stored in fixed point integer form. If a decimal point is found in the constant, it is stored in floating point form. Care must be taken by the programmer that the forms are not mixed when using fixed point arithmetic instructions or floating point arithmetic instructions.

A location symbol is optional. If present, it is assigned to the first word.

The following constant specifications are legal:

```
PERCNT       DEC       5, 0, -8
```

(PERCNT) $= 0 \longrightarrow 005$
(PERCNT + 1) $= 0 \longrightarrow 000$
(PERCNT + 2) $= 7 \longrightarrow 767$

```
PARAM        DEC       10., 0., -. 5
```

(PARAM) $= 200450 \rightarrow 0$
(PARAM + 1) $= 0 \longrightarrow 0$
(PARAM + 2) $= 577737 \rightarrow 7$

Also legal as a constant specification is the form;

fDdBb

This specification allows constants to be formed in scientific notation as;

$$f \cdot 10^{d} \cdot 2^{b}$$

The _f_ specification determines the format as fixed or floating point. The _d_ and _b_ exponents may be signed or unsigned integers.

The following represent valid address field constants:

	Specification	Value
1.	85D4	850, 000; fixed point
2.	-27D3B2	-108, 000; fixed point
3.	7. 6D-6	. 0000076; floating point

The DECD Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
ℓ	DECD	C1, C2, C3, . . .	

This instruction converts double precision floating point constants to binary and stores them in consecutive pairs of machine words. The rules are the same as for the DEC instruction, except that more digits can be specified for precision and accuracy. Up to 25 digits may be specified.

The following are valid representations of constants:

```
        CØNX      DECD     - 01364295634574321697, 3.1415963214631794
```

CØNX and CØNX + 1 contain the first operand in floating point format.

CØNX + 2 and CONX + 3 contain the second operand in floating point format.

These operands are operated on with any of the following instructions which automatically reference two memory words:

1. DLDA
2. DSTA
3. DFAD
4. DFSB
5. DFMU
6. DFDV

The BCD Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
ℓ	BCD	n,8n characters	

This instruction stores internal BCD characters in consecutive machine words, 8 characters per word. The address field consists of a word count, n, followed by a comma and up to 8 characters following. The word count determines the number of characters scanned on the card. The characters may be any legal BCD characters. This includes blanks, periods, commas, etc. The address field is terminated by the 8n characters being satisfied. Any more characters, to column 73, are treated as remarks.

A location symbol is optional. If present, it is assigned to the first word.

Here is an example of how the BCD characters are prestored.

MESSAGE	BCD	2, NØ ERRØRS FØUND
(MESSAGE)		= NØ ERRØR
(MESSAGE + 1)		= S FØUND

The TYPE Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
ℓ	TYPE	n, 8n characters	

This instruction is very much like the BCD instruction except that the characters are in typewriter code.

For special settings on the typewriter, the following code is used:

Specification	Meaning
*R	Carriage Return
*U	Shift to Upper Case
*L	Shift to Lower Case
*B	Backspace
*T	Tab
*X	'
*A	'
*S	;

Even though each specification represents two columns on the card, it represents and counts as only one typewriter character.

Here is an example of pre-storing a typewriter message.

```
MESSAGE        TYPE      3, *RREMØVE TAPE REEL XX
(MESSAGE)                  = *RREMØVE
(MESSAGE + 1)              =  TAPE REE
(MESSAGE + 2)              =  L XX
```

The VFD Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
ℓ	VFD	mn/v,mn/v,...	

This instruction allows the programmer to pre-store variable types of data. The data can form a continuous string and can be of different modes. The instruction can convert octal and decimal constants, hollerith characters, typewriter characters and arithmetic expressions. The address field consists of one or more subfields separated by commas. Each subfield is in the form mn/v, where m specifies mode, n specifies number of bits, and v specifies value. The address field is terminated by the first blank which is not a part of a hollerith or typewriter field. A location symbol, if present, is assigned to the first word.

Five modes are allowed:

On/v	Octal constant -	Same rules as on OCT instruction except that n may range from 1-48.
Hn/v	Hollerith character code -	Same rules as on BCD instruction except n must be a multiple of 6 since n represents characters instead of words.
Tn/v	Typewriter character code -	Same rules as on TYPE instruction except n must be a multiple of 6 since n represents characters instead of words

Bn/v	Bank term -	the n term may be omitted as it is always assumed 3. The v term, when stored, must coincide with a bank designated position in a machine word.
An/v	Arithmetic expression of decimal constant -	represents a decimal fixed point constant or a relocatable expression if n = 15.

An example of how a "string" of data can be formed is shown here:

```
STRING    VFD   O24/-41, A24/32, T24/ABCD, H24WXYZ

(STRING)        = 7——>7360——>040
(STRING + 1)    = A B C D W X Y Z
```

LITERALS

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	Instrn.	=MV	
		=SNAME	
		=DSNAME	

Specifying literals in a subprogram gives the programmer an easy, fast, and efficient method of pre-storing data and reserving a block of memory. Consider the following set of instructions:

```
        LDA         CØN1
        MUI         CØN2
        STA         SAVE
```

If the program multiplies two constants, they must be specified somewhere, probably with a DEC card. Also, if the program stores the result, the address must be reserved, probably with a BSS card. If the constants are known, there is an easier method to perform the operation; using literals. By doing so, the programmer could have coded as follows:

```
        LDA         =D1564318
        ADD         =D-16741
        STA         =SSAVE
```

The execution time for either method is the same. But the second method needs no pseudo instructions to pre-store the data or reserve the area.

The Compass assembler automatically does so when the equal sign is scanned and the value is read.

A literal usually replaces the m subfield of an instruction and has the form:

1. =MV
2. =SNAME
3. =DSNAME

1. Literal Form: =MV

For this literal, M ≡ mode and V ≡ value. The mode can be any of the following:

D Decimal constant	The value can be any of those allowable on the DEC instruction.
O Octal constant	The value can be any of those allowable on the OCT instruction.
H Hollerith codes	The first 8 characters following the H specify the internal BCD value.
T Typewriter codes	The first 8 characters following the T specify the typewriter value.
DD Double Precision Decimal Constant	The value must be in floating point format. The result occupies two words.
DO Double Precision Octal Constant	The value must be octal digits. The result requires two words.

With the preceding literals the following instructions could be coded:

```
LDA    =D-.05
FMU    =D6.8
```

The programmer must make sure that the instruction and arithmetic format agree.

2. Literal Form: =SNAME

For this literal S ≡ Storage address. The "S" must be specified and is followed by the name. When the name is scanned by the assembler, a memory word is automatically reserved. This literal is closely related to the BSS instruction.

With this literal the programmer can code;

```
                    STA    =SHØLD
```

Here he is storing A at address HØLD, which does not have to be declared in the location field. The assembler generates its own address.

3. Literal Form: =DSNAME

   For this literal DS ≡ <u>double precision storage address</u>. The "DS" must be specified and is followed by the name. Two words are reserved at addresses:

   NAME and NAME + 1.

   This literal is usually used with double precision floating point arithmetic.

NEW CONCEPTS OF GROUP 19

The local storage allocation pseudo instructions are not machine instructions as we had presented in the last 18 groups (except for IDENT, ENTRY, BSS, OCT, DEC, and END instructions). Since these are not machine instructions, they normally go before the beginning, or after the end of the executable part of the program. The only time they could be placed within the executable part of the program is if the programmer jumps over the area, or instructions are placed in the area before the area is executed.

A block area reserved by the BSS instruction can make use of the ISK instruction to process or generate consecutive data. A block area reserved by the BES instruction can make use of the IJP instruction.

Literals usually should replace instructions having an m subfield. There is a difference between the instructions;

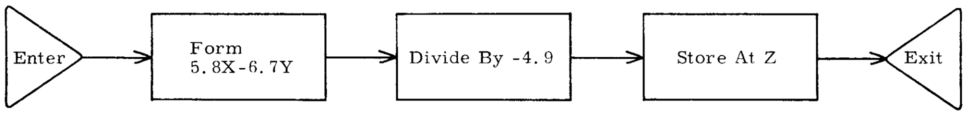
```
1. LDA =D691534
2. ENA =D691534
```

The first one loads A with the decimal integer 691534. The second one enters the assembler generated address into A with sign extended. This, in most cases, represents an undefined quantity.

Problem 19:

Use literals to solve the problem to solve $Z = \dfrac{5.8X - 6.7Y}{-4.9}$ if X and Y are given in floating point.

Flowchart:



Problem 19 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	EVAL	
	ENTRY	EVAL	
Z	BSS	1	
EVAL	BSS	1	
	LDA	Y	
	FMU	=D6.7	
	STA	-SHØLD	
	LDA	X	
	FMU	=D5.8	
	FSB	-SHØLD	5.8X-6.7Y
	FDV	=D-4.9	
	STA	Z	
	SLJ	EVAL	
	END		

Somewhere within this subprogram would also be included the symbols X and Y in the location field with a declaration of the pre-stored data or area reserved.

Student Problem 19A:

Using literals write a subprogram to evaluate $Q = \dfrac{-4.8(3R + 7.1S)}{8.2S - 71.6T}$ if R, S, and T are given in floating point.

Flowchart:

Problem 19A* could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

GROUP 20

SUBPROGRAM LINKAGE

GROUP 20

SUBPROGRAM LINKAGE

1.	Entry	ENTRY
2.	External	EXT

These are the instructions that allow communication between subprograms. Neither of them generates words or uses up memory core. When the symbols are extracted, they are entered into a symbol table for cross-referencing at load time.

The ENTRY Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ENTRY	m	

This instruction declares alphanumeric symbols within the subprogram as entry points which may be referenced by other subprograms. An entry point must be defined in the location field in the same subprogram that it is declared. Entry point symbols must be unique.

The EXT Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EXT	m	

This instruction declares symbols that are external to the subprogram in which the EXT instruction occurs. Symbols are declared like they are on the ENTRY instruction. At load time external symbols are assigned the value corresponding to the symbol in another subprogram.

For every external symbol, there must be a corresponding entry point. The converse is not true.

NEW CONCEPTS OF GROUP 20

The concepts given for this group may represent the most important information contained in this manual. Not only is the communication between subprograms discussed, but also communication between subprograms in a multi-bank system. This is especially important for the 3600/3800 systems programmers.

The discussion is divided into two parts:

1. A discussion of subprogram communication in a one-bank system. This applies to the 3400/3600/3800 systems.
2. A discussion of subprogram communication in a multi-bank system. This applies to the 3600/3800 systems.

1. Subprogram Communication in a One-Bank System

We already know that a subprogram consists of the instructions from IDENT through END. We have solved problems using this basic subprogram form. Now let's discuss subprograms further and introduce new ideas and definitions as we go along.

We speak of a subprogram as being a closed routine. By this we mean that all symbols are local to the routine. This means that, within a subprogram, any symbol in the address field must be declared in the location field, unless the symbol is declared external to the subprogram or in Common (discussed later). The symbol then becomes global (opposite of local).

Subprograms can be assembled together as such:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	A	
	ENTRY	A	
	•		
	•		
	•		
	END	A	
	IDENT	B	
	ENTRY	B	
	•		
	•		
	•		
	END		
	IDENT	C	
	ENTRY	C	
	•		
	•		
	•		
	END		

Each subprogram is assembled as a separate independent subprogram. The first subprogram is usually referred to as the main subprogram and the others as sub-subprograms.

If a subprogram is to be entered for execution, an entry point into the subprogram must be given. This is why you see at least one entry point for each subprogram.

How do we know which subprogram program control is initially going to enter? This is specified in the address field of one of the END instructions. This symbol is an address which is declared an entry point into one of the subprograms. This is called the "transfer address", the address to which program control will initially transfer to after the subprograms have been loaded and are ready to run. Note in the example above how subprogram A will initially be entered.

Program control actually starts at that address + 1. At the address is stored a 48-bit instruction that, when executed, will return program control to the monitor. This is why, at the end of a subprogram, an SLJ is made to the entry point, which executes the stored instruction.

Now how does the programmer transfer control from one subprogram to another? He does this by declaring a symbol external (EXT) and jumping to it. Of course, the symbol must be declared an entry point (ENTRY) in the subprogram jumped to. For a one-bank system the subprograms might look like this:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	A	
	•		
	•		
	•		
	EXT	SUB	
	RTJ	SUB	
	•		
	•		
	•		
	END		
	IDENT	SUB	
	ENTRY	SUB	
SUB	SLJ	**	
	•		
	•		
	•		
	SLJ	SUB	
	END		

Here, the second subprogram is treated as a subroutine to the main subprogram. Notice how the symbol SUB is declared external in the first subprogram and an entry point in the second subprogram.

When the "return" is made from the second subprogram to the first, it is not necessary to declare an external symbol or entry point. The return is taken care of automatically by the hardware. A SLJ is made to SUB which will return control to the first subprogram, to the point from which it left off. So we might say that EXT-ENTRY declarations are necessary going in the forward direction but not when returning. This concept is especially important when nesting subroutines.

EXT-ENTRY instructions can also be used to reference data outside of a subprogram. Consider the following:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	A	
	•		
	•		
	•		
	EXT	CØN1, X	
	LDA	CØN1	
	FMU	CØN1+1	
	STA	X	
	•		
	•		
	END		
	IDENT	B	
	ENTRY	B, CØN1, X	
CØN1	DEC	-6.8, 3.14159	
X	BSS	1	
	•		
	•		
	END		

Note how the data is read and stored outside the subprogram. In fact a whole table can be read or generated outside a subprogram by means of the EXT-ENTRY instruction and an index counter. By using the EXT-ENTRY instructions we make the symbol global instead of local to the subprogram.

2. Subprogram Communication in a Multi-Bank System

Assuming we know a little about subprograms existing in one bank, we now discuss subprograms existing in more than one bank (3600/3800 systems only). Here we have to be careful, because we have two bank registers to consider, one for instructions and one for operands, and it becomes easy to reference the right address but the wrong bank.

Subprograms in a multi-bank system may be interspersed through memory; a few in bank 0, a few in bank 1, etc. However, it is important to note that each subprogram must be wholly contained in one bank, i. e., one part may not be in one bank and the other part in another bank.

Unless the programmer knows for sure that subprograms are loaded into the same bank, he normally makes use of the bank designators given in the Compass instruction formats.

when transferring control from one subprogram to another or referencing data in another subprogram.

Let's assume a simple example. Suppose subprogram A is in bank 1 and subprogram B is in bank 2. Control can be transferred from subprogram A to subprogram B as follows:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	A	IN BANK 1
	•		
	•		
	•		
	EXT	B	
	BRTJ	(2)B,,2	
	•		
	•		
	•		
	END		
	IDENT	B	IN BANK 2
	ENTRY	B	
B	BSS	2	
	•		
	•		
	•		
	SLJ	B	
	END		

In this case the symbol is again declared external. In order to transfer control across banks a bank jump instruction is necessary, and in this case a BRTJ instruction is used. The second subprogram is then treated as a subroutine. It will process operands in bank 2.

Now let's suppose the same problem as above, except that in subprogram A there exists a table of 100 locations starting at TAB. When the return jump is made to subprogram B, it is to process the data beginning at TAB in subprogram A. Here is an example of what is needed.

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	A	IN BANK 1
	ENTRY	A,TAB	
TAB	BSS	100	
A	BSS	1	
	.		
	.		
	.		
	EXT	B	
	BRTJ	(2)B,,1	
	.		
	.		
	.		
	END		
	IDENT	B	IN BANK 2
	ENTRY	B	
	EXT	TAB	
B	BSS	1	
	LDA	TAB	
	.		
	.		
	.		
	SLJ	B	
	END		

Any time a symbol is referenced outside a subprogram, it must be defined as an external symbol and elsewhere as an entry point.

When the subroutine is entered, the Operand Bank register is 1 from the BRTJ. For this reason no bank term is necessary on the LDA instruction (LDA is added just as an illustration). It will load the first word at TAB. If indexing was used in the subroutine, the complete table at TAB could be processed.

The previous two examples dealt with subprograms that were each in a given bank. We could use numeric bank designators so that we made sure that we referenced not only the right address, but also the right bank. But what if two subprograms exist in two different banks and it is not known which bank they reside in? Can there be communication between the two subprograms?

Yes, there can be, and it requires some more definitions of bank designators. The bank designator may be one of the following symbols:

1.

Bank Designator	Meaning
$	The bank that the associated symbol resides in.

Example: LDA ($) NUM

Meaning: Load A from address NUM from the bank in which NUM resides. NUM would be declared an external symbol, and in its subprogram, an entry point. This instruction actually changes the Operand Bank control to the bank of NUM and will remain there until it is explicitly changed again.

2.

Bank Designator	Meaning
*	The bank that the instruction using this symbol resides in. In other words, the bank of the associated subprogram. This instruction actually changes the Operand Bank control to the bank of the present subprogram and remains until it is explicitly changed again.

Let's assume that in subprogram A we wish to form $Z= 3.5X-6.7$, but that 3.5 is in subprogram A and 6.7 is in subprogram B, both in floating point form. Here's how it could be done.

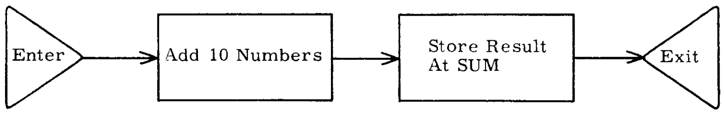
```
LOCATION  OPERATION,MODIFIERS  ADDRESS FIELD          COMMENTS
          IDENT                A                      IN ONE BANK
          ENTRY                A
          EXT                  CØN2
CØN1      DEC                  3.5
X         DEC                  700.14
Z         BSS                  1
A         BSS                  1
          LDA                  X
          FMU                  CØN1
          FSB                  (*)CØN2                SWITCH TØ ANØTHER BANK
          STA                  (*)Z                   RETURN TØ THIS BANK
          SLJ                  A
          END
          IDENT                B                      IN ANØTHER BANK
          ENTRY                B,CØN2
CØN2      DEC                  6.7
          .
          .
          END
```

This is how we can reference data in another subprogram, process it, and return answers in the initial subprogram.

Problem 20:

Write a subprogram that will add 10 fixed point integer numbers starting at address ELEMENTS and store the result at address SUM. The numbers, however, are in another subprogram.

Flowchart:



Problem 20 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	A	
	ENTRY	A	
	EXT	ELEMENTS	
SUM	BSS	1	
A	BSS	1	
	ENI	0,1	
	LDA	($)ELEMENTS,1	
NEXT	ADD	ELEMENTS+1,1	
	ISK	8,1	
	SLJ	NEXT	
	STA	(*)SUM	IN THIS BANK
	SLJ	A	
	END		

In another subprogram would be the table ELEMENTS and the symbol declared as an entry point.

Student Problem 20A:

Write a subprogram to calculate $S = .5MV - 3.5T^2$ in floating point if S is local and M, V, and T are external.

Problem 20A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

GROUP 21

SPECIAL INSTRUCTIONS

Group 21

SPECIAL INSTRUCTIONS

1.	Equivalence	EQU
2.	Bank	BANK
3.	Enter Operand Bank	ENO
4.	Call	CALL

These instructions do not fit in any group that we have discussed so far. They can be helpful to the programmer in order to perform various functions.

The first instruction equates a symbol to a numeric quantity or expression. When the programmer references the symbol, he is referencing the quantity.

The second instruction allows the programmer to define which bank each subprogram is to reside in. By doing so he can position his data better and keep the execution time of his program to a minimum.

The third instruction enters the Operand Bank register with any bank term so that future operand bank references will be taken from that bank.

The fourth instruction is a pseudo instruction that is placed in the executable portion of the program. The reason is that the assembler converts the CALL to the two instructions:

1. EXT
2. BRTJ ($)___________, , $

The first instruction defines the called name as being external, and the second instruction performs a bank return jump to that name, setting both the Instruction and Operand bank controls to the bank in which the name resides.

The EQU Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
l	EQU	m	

The EQU instruction equates a location field symbol with an address field expression. Any symbol in the address field must have been previously defined. The symbol in the location field will be assigned the value in the address field wherever it is found in the subprogram.

The BANK Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BANK	(a) NAME, (a) NAME, . . .	

The BANK instruction defines into which bank subprograms are to reside. The address field contains one or more bank terms, each followed by one or more names which represent entry points or common blocks.

The ENO Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ENØ	2	

The ENO instruction is a hardware instruction that changes the Operand Bank register to any one of the 8 possible banks. If this instruction changes the Operand bank register, it will remain so until it is explicitly changed again. The following two sets of coding will do the same thing and each set takes up 48 bits.

```
Set 1:          ENØ        2
                LDA        DICT       DICT IN BANK 2

Set 2:          LDA        (2)DICT    DICT IN BANK 2
```

The CALL Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CALL	NAME	

The CALL instruction defines the symbol appearing in the address field as an external symbol and assembles a bank return jump to the symbol. If NAME is a symbol, the assembly would take place as follows:

```
          CALL            NAME
                 becomes
          BRTJ            ($)NAME, , $
           EXT             NAME
```

The bank return jump is made to the bank in which name resides. The Operand bank register is also changed to that bank. This instruction is much like the Fortran CALL statement which can call a subroutine.

NEW CONCEPTS OF GROUP 21

The EQU instruction can be used to equate symbols to numbers. The programmer can refer to the symbol instead of the number, and can remember it easier since symbols are easier to remember than numbers. For example, checking for the symbol COMMA is easier than checking for 73B.

The BANK instruction will allow the programmer to place the most interactive sub-programs in the same bank so as to minimize the execution time.

The ENO instruction is simply a convenient method to change the Operand bank control.

The CALL instruction is also a convenient method to transfer program control to an external subroutine.

Problem 21:

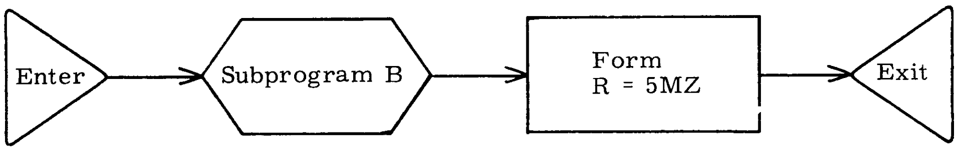
Given: Subprograms A and B

Subprogram B is a subroutine to subprogram A. Subprogram A is entered first.

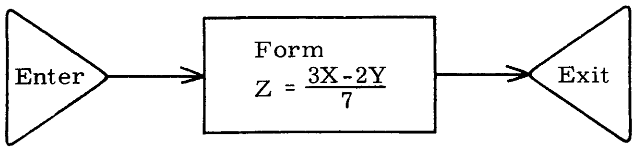
Write the subprograms, such that, subprogram B will calculate $Z = \frac{3X-2Y}{7}$ in floating point if X, Y, and Z are in subprogram A. Subprogram A should then form R = 5MZ where M is in subprogram B.

Flowchart:

SUBPROGRAM A



SUBPROGRAM B



Problem 21 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	A	
	ENTRY	A,X,Y,Z	
	EXT	M	
X	BSS	1	ASSUME FLØATING PØINT
Y	BSS	1	NUMBERS PRESTØRED
Z	BSS	1	
R	BSS	1	
A	BSS	1	
	CALL	B	GØ TØ ($)B. SWITCH ØB TØ THIS
	FMU	=D5.	BANK
	FMU	($)M	EXTERNAL
	STA	R	R=5MZ
	SLJ	A	
	END		
	IDENT	B	
	ENTRY	B,M	
	EXT	X,Y,Z	
M	BSS	1	
B	BSS	1	
	LDA	X	
	FMU	(*)=D3.	
	FSB	($)Y	
	FSB	Y	3X-2Y
	FDV	(*)=D7.	
	STA	($)Z	EXTERNAL
	SLJ	B	
	END		

Student Problem 21A:

Given: Subprograms A and B

Subprogram B is a subroutine to subprogram A. Subprogram A is entered first.

Write the subprograms, such that, subprogram B will compare floating operands at point X and Y, store the larger in the A register, and return. Subprogram A will then form Z = 3.(A)* 5.2X in floating point. Assume X, Y, and Z in subprogram A.

Flowchart:

Problem 21A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

GROUP 22

GLOBAL STORAGE ALLOCATION

GROUP 22

GLOBAL STORAGE ALLOCATION

1.	Block	BLOCK
2.	Common	COMMON
3.	Originate Relative Counter	ORGR

The first two instructions declare a block of storage which is not within a subprogram, but common to it. These instructions are comparable to the BSS instruction except that it is local and these are global.

The third instruction originates the relocatable counter to any position in the common area. The normal instructions used to prestore data (DEC, etc.) follow this instruction. The data is then assembled into the common area.

The instruction ORGR * returns the relocatable counter to the point from which it left. The instructions following this instruction would then be assembled within the subprogram.

The ORGR instruction can also be used to prestore data in the subprogram.

The BLOCK Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
l	BLØCK	m	

The BLOCK instruction defines a block of common. The name of the block is defined by the symbol in the location field. There are three possible types of common.

1. Labeled Common
2. Numbered Common
3. Blank Common

1. Labeled Common is declared if the location field contains a legal alphanumeric symbol.

2. Numbered Common is declared if the location field contains a number. If it does, all characters must be numbered.

3. Blank Common is declared if the location field is blank.

The definitions of these types of common are given later.

The value in the address field specifies the total maximum length of the common block. The value must be greater than or equal to the total length specified in subsequent COMMON instructions.

An address field containing zero or blank will set no limit on the size of the block and the size is determined by the amount of storage words declared by the subsequent COMMON statements. A block of common may not exceed one bank of memory and must be self-contained in one bank.

The COMMON Instruction

FORM:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	CØMMØN	A(I,J,...),B(I,J,...),	...

The COMMON instruction follows the BLOCK instruction and declares the arrays to be declared in the common block. The address field consists of one or more subfields, each of which defines an array to be included in the block.

An array is represented by A(i, j, k, . . . n) where the number of words reserved is the product of the dimensions.

The ORGR Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ØRGR	m	

The ORGR instruction may be used at any point in the subprogram to initiate a sequence of instructions or constants at a location different from the current program location. The address field contains an expression which must result in a program or labeled common relocatable value. Subsequent instructions or data words are assembled sequentially, beginning at the location specified by that value. This sequence continues until another ORGR instruction is encountered or until the end of the subprogram.

The number of words assembled into a labeled common block must not exceed the length of the block. The address field expression may not represent a location in numbered or blank common.

When the main subprogram storage assignment sequence is interrupted by an ORGR, the subprogram location counter is saved. An ORGR* will terminate the assembly of information into an area and return the location counter to the subprogram to the point from which it was interrupted.

NEW CONCEPTS OF GROUP 22

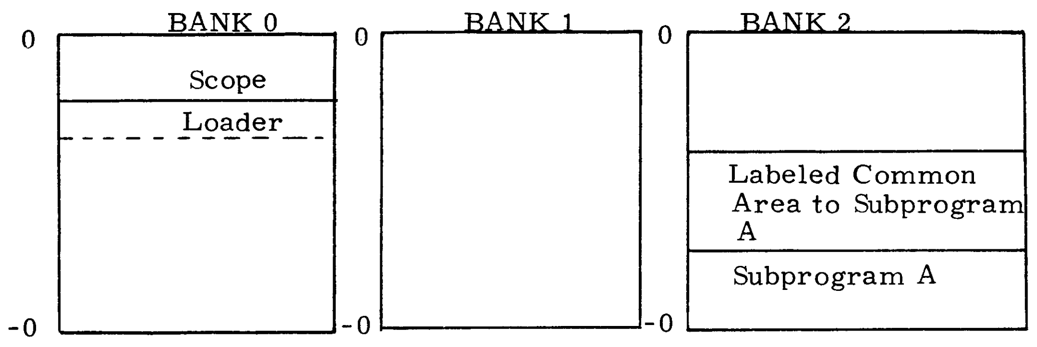
By having data stored in a common area, subprograms can reference the data without use of external symbols or entry points. The common area physically is a self-contained area but can be referenced quickly and easily if it is declared common to any subprogram. If more than one subprogram declares the same common area, any of them can reference it.

There are three types of common as defined on the BLOCK instruction.

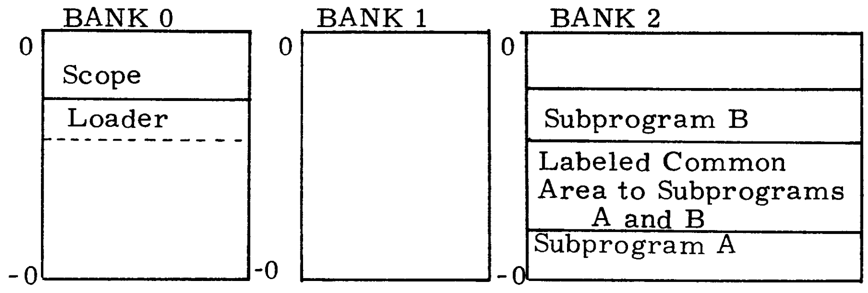
1. Labeled Common
2. Numbered Common
3. Blank Common

LABELED COMMON

Labeled common is a common area which is external to, but in common with the subprogram that declares it. Assume subprogram A declares a labeled common area. When the subprogram is loaded, it occupies highest core of the highest available bank. If a system consisted of three banks, the subprogram and common area would be loaded like this:



If subprogram B declares the same block of common by specifying the same symbol on its BLOCK instruction, the subprograms will be loaded as such:



From this, we can see that labeled common normally precedes the subprogram that defines it, and any other subprogram that defines it is simply loaded and has access to it.

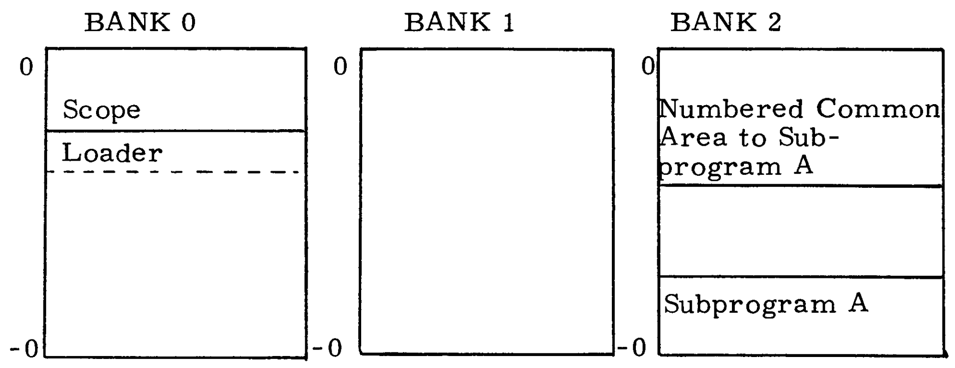
Labeled common may be prestored with data using the ORGR instruction. Usually the first subprogram will do it.

A common area may also reside in a different bank from the subprogram that declares it. Care must then be taken in the subprogram to specify the bank term when referencing the data.

NUMBERED COMMON

Numbered common is a common area which is external to, but in common with the subprogram that declares it. Assume subprogram A declares a numbered common area. When the subprogram is loaded, it occupies highest core of the highest available bank. The declared numbered common occupies the lowest available core of the same bank.

If a system consisted of three banks, the subprogram and common area would look like this:



Other subprograms can declare the same numbered common if the identical number is declared on its BLOCK instruction. Other subprograms would be loaded just previous to the given subprogram.

If a system has only one bank, bank 0, you will note that a problem arises with respect to the loading of numbered common. Scope cannot be destroyed or the monitoring of the subprogram will not take place. For this reason numbered common starts between Scope and the loader. Since the loader is needed during the loading operation, it is alright to reserve a numbered common area with the BLOCK and COMMON instructions, but it is illegal to prestore data in it since the prestored data would overlay (cover) the loader and wipe it out making loading impossible. For this reason it is illegal and an assembler diagnostic to attempt to prestore data in numbered common and blank common (as you will see later). It is legal always to prestore data in a labeled common area.

BLANK COMMON

Blank common is just like numbered common (in fact, many refer to blank common as numbered common), except that each blank common declaration overlays the previous blank declaration. This is because there can be no distinction in the location field of the BLOCK instruction since it is always blank. On numbered common, different numbers define different blocks and the distinction can be made. The same is true for labeled common.

Problem 22:

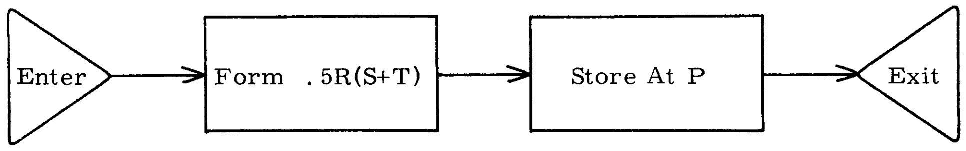
Given: Subprogram A

Labeled common block B1 containing address symbols P, R, S, T and the floating point constant . 5.

The subprogram and common areas are in different banks.

Write subprogram A so that this area is generated and then form P = . 5R(S+T).

Flowchart:



Problem 22 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	A	
	ENTRY	A	
B1	BLØCK	5	
	CØMMØN	P, R, S, T, CØN1	
	ØRGR	CØN1	
	DEC	.5	
	ØRGR	*	
A	BSS	1	
	LDA	(Ø)S	
	FAD	T	
	FMU	R	
	FMU	CØN1	
	STA	P	
	SLJ	A	
	END		

Student Problem 22A:

Given: Subprogram A

Subprogram B

Labeled common B1 containing addresses P, Q, R, S, T, M, and floating point constants .5 and 36.

Write subprogram A to generate this area and form $S = .5P(Q-R)$.

Then CALL subprogram B and form $M = 36 \dfrac{2S-T}{S+T}$

Then return to subprogram A and exit.

Assume the subprograms and the common area in different banks.

Flowchart:

Problem 22A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 23

PROGRAMMER MACROS

GROUP 23

PROGRAMMER MACROS

1. Macro	MACRO
2. End Macro	ENDM

The two instructions listed above are the beginning and ending instructions of a macro definition. The macro definition must come immediately after the IDENT instruction.

The MACRO Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
ℓ	MACRØ	(formal parameters)	

The MACRO instruction signifies the beginning of a macro definition. The location field contains the name of the macro. The name of the macro may be any legal alphanumeric symbol except IDENT, END, MACRO, IFT, IFF, ENDM, LIBM, or SCOPE.

The address field may be blank or may contain a set of formal parameters. The formal parameters are also placed in any field in the macro definition and are substituted by actual parameters during the assembly of the subprogram. The set of parameters should be enclosed in parentheses.

Formal parameters are separated by commas and must be legal alphanumeric symbols.

The ENDM Instruction

FORM:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
		ENDM		

The ENDM instruction terminates a macro definition.

NEW CONCEPTS OF GROUP 23

A programmer macro is an inline subroutine. The macro is defined and then it may be called at various places in the program with parameter substitutions. Each time it is called, the macro is inserted within the subprogram and the coding is assembled.

If a macro is defined, it must be done so immediately after the IDENT instruction.

Problem 23:

A two dimensional 3 x 3 array is shaped according to the matrix as shown here:

ARRAY

A11	A12	A13
A21	A22	A23
A31	A32	A33

The array is arranged in row-column fashion where A(1, 1) represents row 1-column 1, A(2, 1) represents row 2-column 1, etc.

The array exists in memory as one column as given here:

1. A11
2. A21
3. A31
4. A12
5. etc.

From this it is apparent that element A11 is in address A, element A21 is in address A + 1, etc.

Write a subprogram that will;

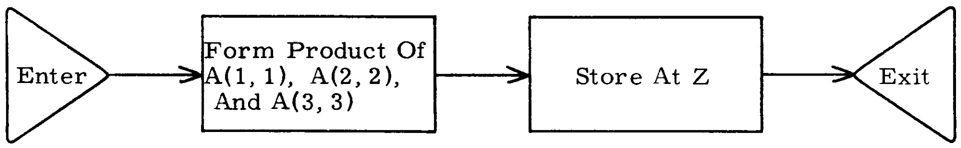
1. contain a macro that will return in the A register the contents of the memory address of an element specified by its coordinates.

2. multiply the diagonal elements A11, A22, and A33, in floating point and store the result at Z.

If A(I, J) represents a given array, an element A(i, j) of the array can be converted to a memory address by the formula;

$$A(i, j) = A + (i-1) + (j-1) * I$$

Flowchart:



Problem 23 could be solved by coding in the following manner;

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	MATMUL	
RETURN	MACRØ	(P1, P2)	
	LDA	A+P1+3*P2-4	CØNVERTED ADDRESS
	ENDM		
	ENTRY	MATMUL	
Z	BSS	1	
A	BSS	9	ARRAY
MATMUL	BSS	1	
	RETURN	(1, 1)	
	STA	Z	
	RETURN	(2, 2)	
	FMU	Z	
	STA	Z	
	RETURN	(3, 3)	
	FMU	Z	
	STA	Z	
	SLJ	MATMUL	
	END		

<u>Student Problem 23A:</u>

Assume problem 23 conditions except here we have a three dimensional, 9 x 9 x 9 array.

A111	A121.............	A191
A211	A221.............	A291
.	.	.
.	.	.
.	.	.
A911	A921.............	A991

The second through ninth planes are suffixed with the same numbers except the last digit is two through nine. Write a subprogram that will;

1. contain a macro that will return the value of an element to the A register if its coordinates are given in the macro call. If the memory layout is again like problem 23, the formula is;

   $A(i, j) = A + (i-1) + (j-1) *I + (k-1) *I *J$

2. form the product of the diagnal from one corner to its opposite. This would include elements A111, A222, A333, etc., all multiplied. Store the result at address Z.

Flowchart:

Problem 23A could be solved by coding in the following manner:

LOCATION		OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9	10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

GROUP 24

ECHO AND IF INSTRUCTIONS

GROUP 24

ECHO AND IF INSTRUCTIONS

1.	Echo	ECHO
2.	If Non-zero	IFN
3.	If Zero	IFZ
4.	If Inequalities	IF, s
5.	If Upper	IFU
6.	If Lower	IFL
7.	If True	IFT, s
8.	If False	IFF, s
9.	End If	ENDIF

The first instruction is a pseudo instruction that repeats a succeeding series of instructions or data. Each time the series is repeated different parameters can be substituted.

The other eight instructions are pseudo instructions that answer questions at assembly time. Should the assembler assemble a series of coding or should it bypass it? If yes, the succeeding coding will be assembled and entered as part of the program. If no, the succeeding coding is skipped.

The ECHO Instruction

FORM:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
ℓ	ECHO	m, n, (P1=A1, A2, . . ., P2=B1, B2, . . .)	

The ECHO instruction is a pseudo instruction that causes the succeeding m lines of coding to be assembled n times. Each time the m lines are assembled, actual parameters a, b, c, etc. may be substituted for the formal parameters p_1, p_2, p_3, etc. If no parameters are to be substituted, the column after the n are left blank.

The formal parameters must be alphanumeric symbols. The actual parameters may be any expressions which legally may appear where the formal parameters occur.

Location symbols within the range of an ECHO are assigned only in the first repetitions in order to prevent doubly defined symbols.

The IFN Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IFN	m,n	

The IFN instruction is a conditional pseudo instruction that checks the value of m for non-zero. If the value is non-zero, the next p lines of coding will be assembled. If the value is zero, the next p lines of coding are skipped and are not included in the assembly of the subprogram. Both m and p are evaluated as 15-bit quantities.

If p and the preceding comma are omitted, the range of the IFN is determined by an ENDIF pseudo instruction.

The IFZ Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IFZ	m, n	

The IFZ instruction is a pseudo instruction that checks for the opposite condition of the IFN instruction.

For this instruction, if the value _m_ is zero, the following _p_ lines of coding are assembled. If the value is non-zero, the following _p_ lines of coding are skipped.

The IF, s Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IF, S	m, n, p	

The IF, s instruction is a conditional pseudo instruction in which the operation field modifier s represents the condition compared. The s modifier may be replaced by any of the following:

Mnemonic	Meaning
EQ	$m = n$
NE	$m \neq n$
LT	$m < n$
LE	$m \leq n$
GT	$m > n$
GE	$m \geq n$

The next p lines of coding will be assembled if the relationship specified by the instruction modifier s exists between m and n. The next p lines are skipped, if the specified relationship does not exist.

The IFU Instruction

FORM:

LOCATION		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		IFU	p	

The IFU instruction is a conditional pseudo instruction that checks to see if the preceding instruction was assembled in the lower half of the memory word. If it was, the next p lines are assembled. If it wasn't the next p lines are skipped. If p is omitted, the range of the IFU is determined by the ENDIF pseudo instruction.

The IFL Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IFL	p	

The IFL instruction is a conditional pseudo instruction closely related to the IFU instruction except that, if the preceding instruction was an upper instruction, the next p lines are assembled. If the preceding instruction was not an upper instruction, the next p lines are skipped.

The IFT, s Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IFT, s	m, n, p	

The IFT instruction is a conditional pseudo instruction that compares character strings m and n to determine if coding lines which follow are to be assembled or skipped. This instruction may be used only within the range of an ECHO or MACRO instruction.

The modifier s specifies the condition compared and may be any one of the following:

Mnemonic	Meaning
EQ	$m = n$
NE	$m \neq n$
GT	$m > n$
GE	$m \geq n$
LT	$m < n$
LE	$m \leq n$
IN	m included in n; the character string n contains the characters in string m in sequence, but not necessarily consecutively.

The address field consists of two or three subfields. The m and n subfields must be present. Any actual parameter must be enclosed with slashes. Either or both subfields may be a single formal parameter as defined on the MACRO or ECHO instructions. Either may be a string of characters enclosed in slashes. If p and the preceding comma are omitted, the range of the IFT instruction is determined by the

ENDIF pseudo instruction.

The m and n subfields may include an optional modifier of the form (i, j) to define a portion of the actual parameter to be used in the comparison. This modifier may not contain formal parameters. The (i, j) modifier may be in one of four forms, in which X, Y and Z represent integers and K represents any non-blank BCD character except slash.

FORM	INTERPRETATION
(X, Y)	Y consecutive characters beginning with the xth character of the actual parameter.
(Z = K, Y)	Y consecutive characters following the Zth occurrence of the character K.
(X, Z = K)	Consecutive characters beginning with the Xth character up to, but not including, the Zth occurrence of the character K following the Xth character.
$(Z_1 = K_1, Z_2, = K_2)$	Consecutive characters following the Z_1th occurrence of character K_1 up to, but not including, the Z_2th occurrence of character K_2.

The IFF, s Instruction

FORM:

LOCATION		OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
		IFF, s	m, n, p	

The IFF instruction is a conditional pseudo instruction that is the same as the IFT instruction except that it tests for the opposite condition.

The ENDIF Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ENDIF	NAME	

The ENDIF instruction is a pseudo instruction which defines the limit to a conditional IF instruction not containing a line count. If lines are skipped because of a conditional IF, the associated ENDIF causes normal processing to resume.

The associated IF and ENDIF instructions are related by the location field symbol on the IF instruction and the address field symbol on the ENDIF instruction. If these symbols match, the IF is terminated. An ENDIF with a blank address field is associated with the last encountered unlabeled IF.

If the symbols do not match, the IF is not terminated and counts as a line of coding.

NEW CONCEPTS OF GROUP 24

The ECHO instruction is different from the IF instructions in that the ECHO instruction is not a conditional pseudo. The ECHO instruction can duplicate a series of coding with possible parameter substitution. If the programmer wishes to prestore alternating positive and negative zeros in core in order to perform some hardware test, he can do so simply in the assembly of the program by coding;

```
        ECHO                1, 500
         OCT                0, -0
```

The OCT instruction is assembled 500 times and each time it prestores positive and negative zero. The total amount of core used is 1000 locations.

The IF instructions are used to determine which sets of coding to assemble. If some condition is true, then assemble the coding. If it is not true, then skip the coding.

One example of how the IF instructions can be used is this: Assume an input/output program of approximately 1500 lines of coding. Assume that sometimes it is to be run with the drum as the primary input/output medium and at other times magnetic tape as the primary input/output medium. Now we know that the drum and magnetic tape units are not programmed the same way. Their function codes are different and the data is accessed differently. For each run must we physically remove the set of instructions that does not apply or could we just change one instruction?

To solve this problem we recall the EQU instruction (in group 21). With this instruction we can equate a symbol with a constant. Suppose we could enter either of the following instructions at the beginning of the program:

```
        1.  DRUMTAPE        EQU         1
        2.  DRUMTAPE        EQU         0
```

If DRUMTAPE = 1 means that the drum coding is to be assembled, and DRUMTAPE = 0 means that the tape coding is to be assembled, by inserting IFZ and IFN instructions at strategic points in the subprogram, only the applicable instructions and data will be assembled. Note how, in the following case, the drum instructions are assembled and the tape instructions are bypassed.

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	IØ	
	ENTRY	IØ	
DRUMTAPE	EQU	1	
	•		
	•		
	•		
	IFN	DRUMTAPE,15	
	(15 Lines	of drum coding)	
	•		
	•		
	•		
	IFZ	DRUMTAPE,10	
	(10 Lines	of tape coding)	
	•		
	•		
	•		
	END		

If we change the DRUMTAPE flag to a zero, note how the tape coding would be assembled, and the drum coding would be bypassed.

The IFU and IFL instructions can be used to remove NOP's when several conditionals are used. In this way the subprogram can be packed into core as tight as possible.

The IFT and IFF conditionals are within the range of ECHO or MACRO instructions and can be used to discard impertinent coding for each generation of the ECHO or MACRO. Problem 24 illustrates this use.

Problem 24:

Write a subprogram that contains a MACRO definition and a MACRO call. The MACRO definition makes use of IFT instructions that allows bypassing irrelevant coding for each individual macro call.

This particular macro definition is one that sets the operating mode of a tape unit on channel #1. It has two formal parameters, P1 and P2 which are defined as follows:

Formal Parameter	Meaning
1. P1	Format - the format may be BCD or binary.
2. P2	Density - the density may be 200, 556 or 800 BPI.

The name of the macro is ØPMØDE.

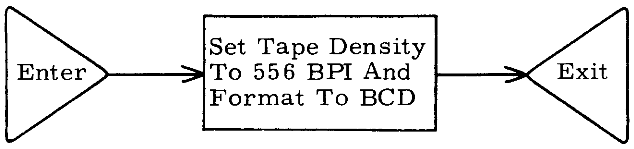
```
ØPMØDE          MACRØ       (P1, P2)
```

For the actual call of the macro any of the following calls are legal:

```
1.   ØPMØDE      (BCD, 200)
2.   ØPMØDE      (BCD, 556)
3.   ØPMØDE      (BCD, 800)
4.   ØPMØDE      (BIN, 200)
5.   ØPMØDE      (BIN, 556)
6.   ØPMØDE      (BIN, 800)
```

For any of the six possible calls, only the applicable coding should be assembled. Use the IFT instruction in the macro definition, and then, as an example, call ØPMØDE specifying BIN format at 556 BPI. The macro should set the tape unit to this operating mode (requires EXTF instructions).

Flowchart:



Problem 24 could be solved by coding in the following manner:

```
LOCATION  OPERATION,MODIFIERS  ADDRESS FIELD        COMMENTS
          IDENT      MØDE
ØPMØDE    MACRØ      (P1, P2)
          IFT,EQ     P1,/BCD/,1
          EXTF       1,2,*                 SET TØ BCD
          IFT,EQ     P1,/BIN/,
          EXTF       1,1,*                 SET TØ BIN
          IFT,EQ     P2,/200/,1
          EXTF       1,4,*                 200 BPI
          IFT,EQ     P2,/556/,1
          EXTF       1,3,*                 556 BPI
          IFT,EQ     P2,/800/,1
          EXTF       1,6,*                 800 BPI
          ENDM
          ENTRY      MØDE
MØDE      BSS        1
          ØPMØDE     (BIN,556)
          SLJ        MØDE
          END
```

<u>Student Problem 24A:</u>

This problem is similar to problem 24 except that, when the subprogram is entered, the A register contains a floating point number. Write a subprogram that contains a macro definition named VELOC with two parameters, P1 and P2 defined as such:

Parameter	Meaning
P1	Increase (A) or Decrease (A)
	INCRA or DECRA
P2	.05 or .1

The four possibilities for the macro call would be;

1.	VELØC	(INCRA, . 05)
2.	VELØC	(INCRA, . 1)
3.	VELØC	(DECRA, . 05)
4.	VELØC	(DECRA, . 1)

From the four choices we can increase (A) by 5 or 10% or decrease (A) by 5 or 10%. Set up the macro definition so that assembly of only the necessary lines will take place. Then decrease the velocity by 10% by calling the macro.

Flowchart:

Problem 24A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

GROUP 25

LIST CONTROL INSTRUCTIONS

GROUP 25

LIST CONTROL INSTRUCTIONS

1.	No List	NOLIST
2.	List	LIST
3.	Space	SPACE
4.	Eject	EJECT
5.	Remark	REM
6.	Title	TITLE
7.	Brief	BRIEF
8.	Detail	DETAIL

The instructions listed above do not affect the execution of the program. They are used only for the source listing.

The first two instructions allow sections of coding to be deleted. They could be sections that the programmer knows contains no errors and would simply cause unnecessary time and waste if they were listed.

The second two instructions make the listing easier to read. They incorporate spacings in the program which separate the various parts.

The TITLE instruction puts a title at the top of each listed page. This makes it easy to distinguish at a glance between subprograms and between various portions of a sub-program.

The last two instructions allow the deletions of repetitious and extraneous coding.

The NOLIST Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	NØLIST		

The NOLIST instruction is a pseudo instruction that suppresses the listing of the source subprogram until a LIST pseudo instruction is encountered. However, lines with error flags will still be listed.

The LIST Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LIST		

The LIST instruction is a pseudo instruction that resumes listing of the source sub-program. This instruction is meaningful only if a NOLIST instruction was encountered previously.

This instruction will have no effect if the Compass control card does not have the list option specified.

The SPACE Instruction

FORM:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	SPACE	n	

The SPACE instruction is a pseudo instruction that spaces the output listing the number of lines specified by n. If the spacing would cause an overflow at the bottom of a page, an eject takes place to the top of the next page.

The EJECT Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	EJECT		

The EJECT instruction is a pseudo instruction that causes the printer to eject the source listing to the top of the next page.

The REM Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	REM	any	

The REM instruction is a pseudo instruction that produces on the listing a line of printed remarks only. If the remarks exceed one line, they can be continued on the next line if a column 10 REM is repeated.

The TITLE Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	TITLE	any	

The TITLE instruction is a pseudo instruction which may appear anywhere in a subprogram after macro definitions and before the END instruction. The first TITLE instruction, no matter where it appears, will have columns 16-72 printed on the top of the first page of the subprogram listing and at the top of all subsequent pages until another TITLE instruction is encountered. Second and subsequent TITLE instructions will cause a page eject.

If TITLE is not used in a subprogram, the contents of the IDENT address field will be printed as the title.

The BRIEF Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	BRIEF		

The BRIEF instruction is a pseudo instruction that deletes the listing of the following:

1. the area set aside for literals.
2. all but the first full word generated by OCT, DEC, and DECD instructions.
3. the second half word and all subsequent words generated by TYPE, BCD, and VFD instructions.
4. location digits of second and subsequent array names of a COMMON pseudo instruction.

BRIEF may occur at any point in a program after macro definitions. It remains in effect until a DETAIL pseudo instruction is encountered.

The DETAIL Instruction

FORM:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	DETAIL		

The DETAIL instruction is a pseudo instruction that causes a return to the normal listing mode that was suppressed by the BRIEF pseudo instruction.

NEW CONCEPTS OF GROUP 25

It is important to note that the instructions in this group affect only the source listing. They do not control the output generated during the execution of the program.

We might say that these instructions add a helpful aid to anyone analyzing a large program. If spacings are allowed to separate important areas, if remarks are included at the beginning of each area, and if areas are entitled at the top, it makes it much easier to read and understand.

In order to show how these instructions affect the listing, problem 25 is a little different than the problems previously presented.

Problem 25:

Write a subprogram that will illustrate the use of the SPACE, REM, TITLE, NOLIST, and LIST pseudo instructions.

Problem 25 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	ILLUST	
	ENTRY	ILLUST	
	REM	THIS IS A DØ NØTHING	
	REM	PRØGRAM FØR ILLUSTRATIØN ØNLY.	
	SPACE	5	
	REM	WE HAVE JUST SPACED 5 LINES	
	REM	AND WE SPACE 5 MØRE IN	
	REM	ØRDER TØ SEPARATE THIS AREA.	
	SPACE	5	
	REM	NØW LET'S ENTITLE THIS PAGE.	
	REM	YØU SHØULD SEE NUTGØØDY	
	REM	AT THE TØP.	
	TITLE	NUTGØØDY	
	REM	NOW LET'S START	
	REM	ØUR PRØGRAM.	
ILLUST	BSS	1	
	LDA	CØN1	
	FAD	CØN2	
	SLJ	ILLUST	
	SPACE	5	
	REM	NOW LET'S ASSEMBLE	
	REM	THE DECIMAL CØNSTANTS	
	REM	BUT NØT LIST THEM.	
	NØLIST		
CØN1	DEC	5.6	
CØN2	DEC	-3.8	
	LIST		
	REM	THE CØNSTANTS ARE	
	REM	ACTUALLY ASSEMBLED.	
	REM	NØ ASSEMBLY ERRØR	
	REM	APPEARS.	
	SPACE	5	
	REM	THANKS FØR THE SHØW	
	REM	AND NØW WE SAY,	
THE	REM		
	END		

Problem 25A:

Write a subprogram that will illustrate the use of the EJECT, BRIEF, and DETAIL pseudo instructions.

Problem 25A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60166700 Rev. B

Litho in U.S.A.