IBM

Programmed Instruction Course

System/360 Assembler Language Coding
Appendix

Education Development — *Endicott, New York*

IBM

Programmed Instruction Course

System/360 Assembler Language Coding

Appendix

Education Development — *Endicott, New York*

ACKNOWLEDGEMENT

We wish to express our appreciation to the Field Engineering Division for providing most of the frames and illustrations used in this course.

In addition, we want to thank the Detroit and Los Angeles DP Education Centers for the frames and problem statements they provided.

Minor Revision (November 1968)

This publication is a reprint of Form R29-0233-2 incorporating minor editorial changes. The original publication is not obsoleted.

Copies of this publication can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM DPD Education Development, Education Center, Endicott, New York 13760

PREFACE

This is the Appendix to the ALC P.I. Course.

There is just one section in this volume. It deals with instructions related to those which were introduced in the previous volumes. As in the previous volumes, sometimes the instructions will be presented individually and sometimes they will be grouped because of their close logical resemblance.

The information presented in this volume is intended to complement and expand the instructions introduced in the first volumes to cover the complete Standard Instruction Set and the Decimal Feature Instructions of the System/360. The table of contents contains both the name of the instruction and its mnemonic to facilitate reference.

TABLE OF CONTENTS

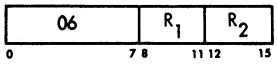
Branch On Count	BCT, BCTR	1
Branch on Index High	BXH	2
Branch on Index Low or Equal	BXLE	4
Store Multiple	STM	6
Add and Subtract Logical	AL, ALR, SL, SLR	7
Load, Special	LTR, LCR, LPR, LNR	10
Load Address	LA	12
Load Multiple	LM	13
Edit and Mark	EDMK	15
Compare Logical	CLR, CL, CLI, CLC	16
Translate	TR	21
Translate and Test	TRT	28
Insert and Store Character	IC, STC	31
Execute	EX	33
Shift Algebraic	SLA, SRA, SLDA, SRDA	36
Shift Logical	SLL, SRL, SLDL, SRDL	41

BRANCH ON COUNT INSTRUCTION

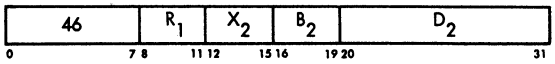
The "branch on count" is related to the "branch and link" instruction. It is used principally to control the number of times that a program loop is executed. Note particularly, that a branch will result each time the first operand has not been reduced to zero, and when the first operand has been reduced to zero, the next sequential instruction will be processed.

Branch On Count

BCTR R_1, R_2 [RR]

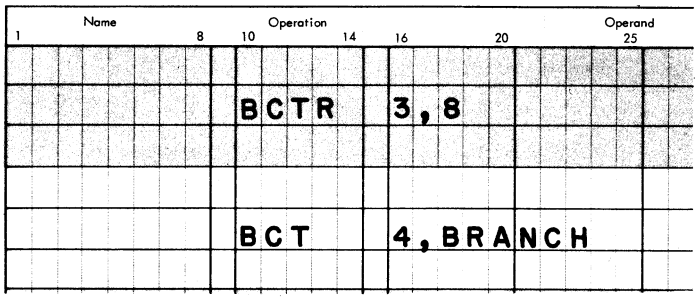


BCT $R_1, D_2(X_2, B_2)$ [RX]



- The digit one is subtracted from the register specified in the first operand (R1).
- The register is then tested for a result of zero.
- If the result is zero, no branch is taken and the next sequential instruction (nsi) is executed.
- If the result is non-zero, a branch is taken to the address specified in the second operand.

Condition Code: The code remains unchanged.
Program Interruptions: None



• • •

1. Just like the "branch and link" instruction, the "branch on count" instruction can be in two formats. List them:
_____, _____.

• • •

RR; RX

2. BCT is the mnemonic for the RX format of "branch on count". The mnemonic for the RR format is_____.

• • •

BCTR

3. Like the BALR instruction, the BCTR instruction will not result in a branch if the R2 field contains _____.

• • •

zero

4. The "branch on count" instruction (either BCT or BCTR) will always reduce the 1st operand (R1) by a value of_____.

• • •

one

5. The "branch on count" instruction will result in a branch if the 1st operand (R1)_____(has/has not) been reduced to zero.

• • •

has not

6. The reduction of the 1st operand occurs_____ (before/after) deciding whether to branch.

• • •

before

7. BCTR 7,3

Assuming that register 7 contains a value of +1, the above "branch on count" instruction_____ (will/will not) result in a branch.

• • •

will not; This is because the BCT instruction will reduce register 7 by 1 before deciding whether or not to branch. This will bring the contents of register 7 to zero.

8. BCTR 7,3

Assuming that register 7 contains a value of zero, the above "branch on count" instruction_____ (will/will not) result in a branch.

• • •

will; Since the register is reduced by 1 before testing for a branch, register 7 was reduced to a value of -1 and the branch did occur. In this case, the preceding instruction would have to be executed 2^{32} times before register 7 could be reduced to zero.

9. Examine the following program.

LOOP_____ _____
_____ _____
_____ _____
BCT 5, LOOP
nsi

Assuming that GR 5 is initialized with a value of 5. How many times will the routine be executed before the next sequential instruction (nsi) is processed?

• • •

5 times. Programming note: All program loops have 4 major characteristics. These are:
1. Initialize
2. Increment
3. Test
4. Branch

Notice that the "branch on count" instruction incorporates three of these characteristics within its own internal operation.
1. It increments by reducing the value of the general register by one.
2. It tests by testing for zero after incrementing.
3. It branches if a non-zero value is found in the register tested.

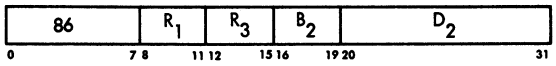
It must, however, be initialized externally.

BRANCH ON INDEX HIGH INSTRUCTION

The "branch on index high" instruction is similar to the "branch on count" instruction except that the increment and the limit (comparand) are explicitly initialized by the programmer.

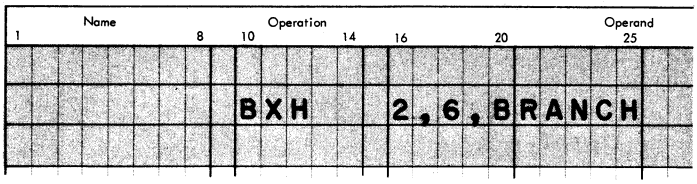
Branch On Index High

BXH $R_1, R_3, D_2(B_2)$ [RS]



- An increment amount is added to the first operand (R1).
- The sum (index) is placed in the first operand.
- The sum is compared algebraically to the comparand amount.
- If the sum is greater than the comparand, a branch is taken to the address specified in the second operand.
- If the sum is equal to or less than the comparand, the next sequential instruction (nsi) is executed.
- The increment is stored in the register specified by R_3.
- If R_3 is an even numbered register, the comparand will be located in the next higher register.
- If R_3 is an odd numbered register, the comparand will be located in the register specified by R_3 and is equal to the increment.

Condition Code: The code remains unchanged.
Program Interruptions: None.



In the above example,
Register 2 contains the index,
Register 6 contains the increment,
Register 7 contains the comparand (R3 is even),
BRANCH is the branch to address.

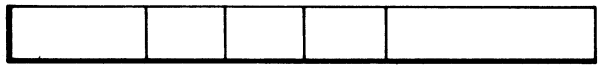
• • •

1. The "branch on index high" instruction has a mnemonic of _____.

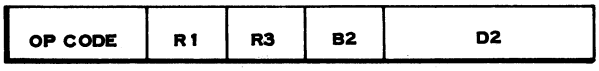
• • •

BXH

2. The BXH instruction uses the RS format. Label the fields of the RS format.



• • •



3. As with the other "branch" instructions you have learned, the generated storage address (B2 and D2 fields) is the_____.

• • •

"branch to" location

4. The R1 field in the BXH instruction is the address of the_____operand.

   Normally the number in an instruction field specifies which operand it is. For example, R1 specifies the 1st operand. However, in the case of the BXH instruction, the R3 field is used to specify the_____operand.

• • •

first; second

5. The second operand is the R3 field register.

   The third operand of a BXH instruction is also in a register. If the R3 field is even, the third operand is in the next odd-numbered register. That is, if the R3 field is 4, the second operand is in register _____and the third operand is in register_____.

• • •

4; 5

6. If the R3 field of a BXH instruction is odd, the second and third operands are in the same register. That is, if the R3 field is 5, the second operand is in register_____ and the third operand is also in register_____.

• • •

5; 5

7. Given the following "branch on index high" instruction, indicate the locations of the three operands.

   BXH 4, 6, BRANCH2

   1st operand is in register_____.
   2nd operand is in register_____.
   3rd operand is in register_____.

• • •

4; 6; 7

8. Given the following "branch on index high" instruction, indicate the locations of the three operands.

   BXH 3, 5, BRANCH2

   1st operand is in register_____.
   2nd operand is in register_____.
   3rd operand is in register_____.

• • •

3; 5; 5

9. BXH 7, 4, BRANCH2

   In the BXH instruction, the second operand is added to the 1st operand and the sum is algebraically compared to the 3rd operand. Given the above instruction, register_____will be added to register _____ and the sum will be compared algebraically to register_____.

• • •

4; 7; 5

10. In the BXH instruction, the resulting sum replaces the first operand after being compared with the _____(1st/2nd/3rd) operand.

• • •

3rd

11. Regardless of whether a branch does or does not occur, the sum of the 1st and 2nd operands always replaces the_____(1st/2nd/3rd) operand.

• • •

1st

12. Given the following, indicate (in hex) the contents of the registers after execution of the BXH instruction.

    BXH 4, 6, BRANCH2

    Everything in hex

	Before	After
Register 4	+16	_____
Register 6	-1	_____
Register 7	+8	_____

• • •

Register 4	+15
Register 6	Unchanged
Register 7	Unchanged

In the preceding problem, a value of -1 was added to a value of +16 and the sum of +15 replaced the 1st operand.

13. The sum of the 1st and 2nd operands is algebraically compared with the_____operand.

• • •

3rd

14. In an algebraic comparison, positive numbers are _____(lower/higher) than negative numbers.

• • •

higher

15. In the "branch on index high" instruction, the branch occurs if the sum is higher than the____ (1st/2nd/3rd) operand.

• • •

3rd

16. BXH 4, 6, BRANCH2

Register 4	00000000	
Register 6	00000001	In Hex
Register 7	00000010	

In the above BXH instruction, a branch_____ (will/will not) occur.

• • •

will not

17. BXH 4, 8, BRANCH2

Register 4	0
Register 8	+16
Register 9	+16

In the above BXH instruction, a branch_____ (will/will not) occur.

• • •

will not; This is because the sum is equal to but not higher than the third operand.

18. BXH 3, 6, BRANCH2

Register 3	+16
Register 6	+ 1
Register 7	+16

In the above BXH instruction, a branch_____ (will/will not) occur.

• • •

will

19. BXH 3, 8, BRANCH2

Register 3	-1	
Register 8	-1	In Hex
Register 9	+1	

In the above BXH instruction, a branch_____ (will/will not) occur.

• • •

will not; The sum of registers 8 and 3 is a value of -2. This is less than the +1 in register 9.

20. BXH 3, 5, BRANCH2

Register 3	+1	
Register 5	+1	In Hex
Register 6	+2	

In the above BXH instruction, a branch_____ (will/will not) occur.

• • •

will; Register 6 is not used in the preceding problem. The R3 field is odd. As a result, register 5 is used for both the 2nd and 3rd operands. The sum of registers 5 and 3 is a value of +2, which, of course, is higher than the contents of register 5.

21. BXH 3, 5, BRANCH2

Register 3	+16	
Register 5	- 1	In Hex
Register 6	+511	

In the above BXH instruction, a branch_____ (will/will not) occur.

• • •

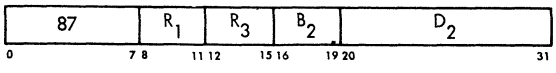
will; A 2nd operand of -1 is being added to a 1st operand value of +16 and the sum of +15 is high compared to the 3rd operand value of -1. (GR5 is the third operand)

BRANCH ON INDEX LOW OR EQUAL INSTRUCTION

The "branch on index low or equal" instruction is very similar to the "branch on index high" (BXH) instruction. Here, however, the branch is taken if the value of the first operand is less than or equal to the third operand (the comparand).

Branch On Index Low or Equal

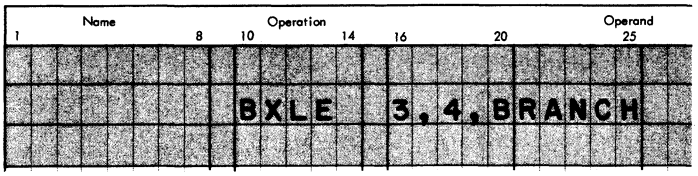
BXLE $R_1, R_3, D_2(B_2)$ **[RS]**



- An increment amount is added to the first operand (R1).
- The sum is placed in the first operand.
- The sum is compared algebraically to the comparand amount.
- If the sum is equal to or less than the comparand, a branch is taken to the address specified in the second operand.
- If the sum is greater than the comparand, the next sequential instruction (nsi) is executed.

- The increment is stored in the register specified by R_3.
- If R_3 is an even numbered register, the comparand will be located in the next higher register.
- If R_3 is an odd numbered register, the comparand will be located in the register specified by R_3 and is equal to the increment.

Condition Code: The code remains unchanged.
Program Interruptions: None.



In the above example,
Register 3 contains the index,
Register 4 contains the increment,
Register 5 contains the comparand (R3 is even),
BRANCH is the branch to address.

• • •

1. BXLE is the mnemonic for the "_____on_____ _____or_____" instruction.

• • •

"branch on index low or equal"

2. The BXLE instruction is similar to the BXH instruction in that the_____operand is added to the _____operand and the sum is algebraically compared to the_____operand.

• • •

2nd; 1st; 3rd

3. Indicate the location of the operands in the following BXLE instruction.

BXLE 3, 6, BRANCH2

1st operand is in register_____.
2nd operand is in register_____.
3rd operand is in register_____.

• • •

3; 6; 7

4. When the sum of 1st and 2nd operands is higher than the 3rd operand, the BXLE instruction differs from the BXH instruction in that a branch_____ (does/does not) occur.

• • •

does not

5. With the BXLE instruction, a branch only occurs when the sum of 1st and 2nd operands is_____ or_____compared with the 3rd operand.

• • •

low; equal

6. BXLE 4, 6, BRANCH2

Register 4 +8
Register 6 +1
Register 7 +16

In the above BXLE instruction, a branch_____ (will/will not) occur.

• • •

will; The sum is lower than the contents of register 7.

7. BXLE 5, 5, BRANCH2

Register 5 +1

When the same register is used for both the 1st and 3rd operands, the sum is compared with the original contents of the register. In the above BXLE instruction, a branch_____(will/will not) occur.

• • •

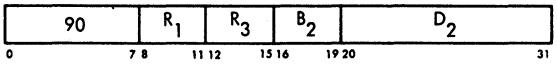
will not; In this case, the same register is used for all three operands. The 3rd operand is the original contents of reg 5. Obviously, then the System/360 will have to bring the contents of this register into ALU (Arithmetic and Logic Unit) and store it in some register so its original contents will not be lost when the 1st and 2nd operands are added together. If at a later time, this instruction is executed again, the sum from the first execution would be used as the 3rd operand.

STORE MULTIPLE INSTRUCTION

The "store multiple" instruction is similar to the "store" (ST) instruction except that more than one consecutive register may be stored in consecutive fullword storage locations.

Store Multiple

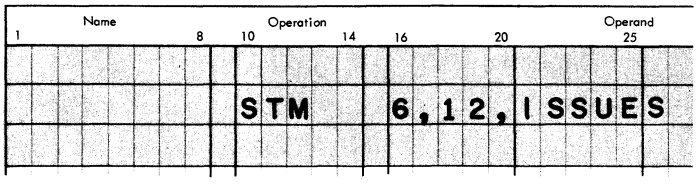
STM $R_1, R_3, D_2(B_2)$ ***[RS]***



The data in a set of general registers starting with the register specified by R_1 and ending with the register specified by R3 inclusive, are stored in a corresponding number of fullword storage locations beginning at the address specified by the second operand.

- The second operand must be on a fullword integral boundary.
- The general registers are stored in ascending order of their addresses starting with R_1.
- Register 0 follows register 15 as a "wraparound" condition is permitted.
- The contents of the general registers is not changed.

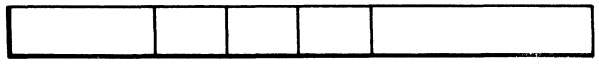
Condition Code: The code remains unchanged.
Program Interruptions:
Protection
Addressing
Specification



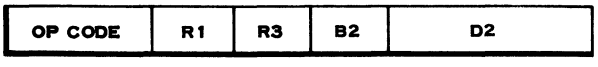
Seven full words will be stored starting at storage location ISSUES from general registers 6 through 12 respectively.

• • •

1. The STM instruction uses the RS format. Label the fields of the RS format.



• • •



2. Like the ST instruction, STM_____ (changes/does not change) the condition code.

• • •

does not change

3. STM 0, 15, 2000(0)

In the above STM instruction, register 0 through _____will be stored in byte locations 2000 through _____.

• • •

15; 2063

4. STM 0, 15, 2002(0)

The above STM instruction will result in a s_____exception.

• • •

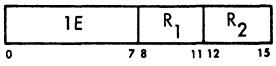
specification; Address 2002 is okay for halfwords but not for fullwords. The STM instruction uses the entire contents (fullword) of the registers.

ADD AND SUBTRACT LOGICAL INSTRUCTIONS

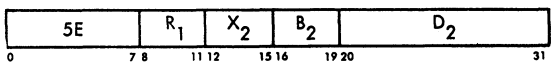
The "add and subtract logical" instructions are similar to the "add" (A) and "subtract" (S) instructions except that no overflow exception occurs (a "carry" is indicated by the condition code) and all 32 bits (unsigned) take part in the operation. This instruction may be used to perform binary addition and subtraction of numbers greater than 31 bits in length.

Add Logical

ALR R_1, R_2 ***[RR]***



AL $R_1, D_2(X_2, B_2)$ ***[RX]***



The fullword second operand is added to the first operand (R1) and the sum is placed in the first operand location.

AL and ALR:

- The first and second operands and the sum are 32 bit unsigned integers.
- This operation differs from the algebraic add in that no program interruption occurs on an overflow. Here an overflow is called a "carry".

AL only:

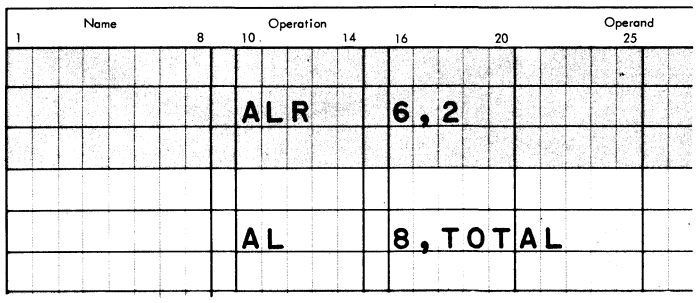
- The second operand must be on a fullword integral boundary.

Condition Code:

0 Sum is zero (no carry)
1 Sum is not zero (no carry)
2 Sum is zero (carry)
3 Sum is not zero (carry)

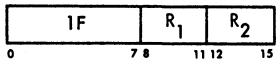
Program Interruptions:

Addressing (AL Only)
Specification (AL Only)

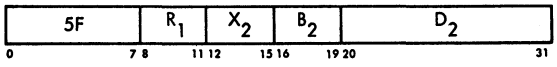


Subtract Logical

SLR ***R_1, R_2*** **[*RR*]**



SL ***R_1, $D_2(X_2, B_2)$*** **[*RX*]**



The fullword second operand is subtracted from the first operand (R1) and the difference is placed in the first operand location.

SL and SLR:

- The first and second operands and the difference are 32 bit unsigned integers.
- This operation differs from the algebraic subtract in that no program interruption occurs on an overflow. Here an overflow is called a "carry".

SL only:

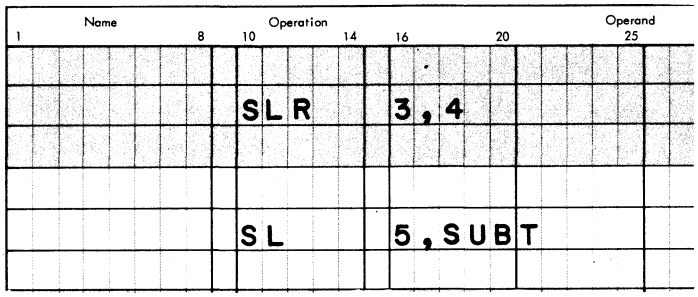
- The second operand must be on a fullword integral boundary.

Condition Code:

0 --
1 Difference is not zero (no carry)
2 Difference is zero (carry)
3 Difference is not zero (carry)

Program Interruptions:

Addressing (SL only)
Specification (SL only)



Logical Add and Subtract

Mnemonic	Hex Op Code	Data Flow
A L	5 E	Fullword storage to register
A L R	1 E	Fullword register to register
S L	5 F	Fullword storage from register
S L R	1 F	Fullword register from register

• • •

1. To differentiate the "logical add/subtract" instructions from the "algebraic add/subtract" instructions, which you previously learned, the logical instructions include the letter_____ in their mnemonics.

• • •

L

2. Just like the algebraic instructions, the logical instructions denote the RR format by the ending letter of _____.

• • •

R

3. The length of both operands in the "logical add/subtract" instructions is always a _____. Therefore, these instructions do not use the mnemonic, _____.

• • •

fullword; H

The instruction AR calls for an algebraic add (signed numbers) while the ALR instruction calls for a logical add (unsigned numbers). Actually, the arithmetic results are the same for both algebraic add/subtract and logical add/subtract.

Algebraic Add	Logical Add
0 1 1 0 1 1 0 1	0 1 1 0 1 1 0 1
+ 0 0 1 1 1 0 0 0	+ 0 0 1 1 1 0 0 0
1 0 1 0 0 1 0 1	1 0 1 0 0 1 0 1

4. Notice that the arithmetic results of the previous example are the same. The operands shown were 8 bits in length for purposes of simplicity. If the arithmetic results of algebraic and logical addition are the same, what is the difference between the two types of instructions?

• • •

The difference is in the consideration of the sign bit (Logical has none).

Algebraic Add	Logical Add
0 1 1 0 1 1 0 1	0 1 1 0 1 1 0 1
+ 0 0 1 1 1 0 0 0	+ 0 0 1 1 1 0 0 0
1 0 1 0 0 1 0 1	1 0 1 0 0 1 0 1
Condition Code = 3	Condition Code = 1

In the preceding example of an algebraic add, a fixed point overflow resulted because of a carry into the sign position without a carry out of it. This overflow was indicated by a condition code of 3.

In the case of the preceding logical add instruction, a fixed point overflow cannot possibly occur because there is no sign bit to consider. All that can be indicated is:

Condition Code	Meaning
0	No carry and zero result
1	No carry and a non-zero result
2	Carry and zero result
3	Carry and a non-zero result

5. The resulting condition code of the following "logical add" would be_____.

```
  1 0 0 1 1 1 0 0
+ 0 1 0 0 1 1 0 0
  1 1 1 0 1 0 0 0
```

• • •

1 (no carry)

6. The resulting condition code of the following "logical add" would be_____.

```
  1 0 0 1 0 0 0 1
+ 1 1 0 1 0 0 0 1
```

• • •

3 (carry)

7. In summary then:

   1. The arithmetic results of "logical" and "algebraic" addition of binary operands are _____(identical/different).
   2. The "logical add/subtract" instructions use _____(halfword/fullword) operand only.
   3. A "logical add/subtract" instruction_____ (can/cannot) result in a fixed point overflow.
   4. The "logical add/subtract" instructions use the letter_____in their mnemonic.
   5. The condition code settings and their meanings are as follows:

Condition Code	Algebraic	Logical
0	Zero Result	No carry, zero
1	Negative Result	No carry, non-zero
2	Positive Result	Carry, zero
3	Overflow	Carry, non-zero

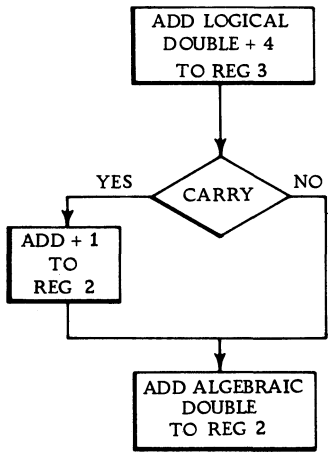
• • •

identical; fullword; cannot; L

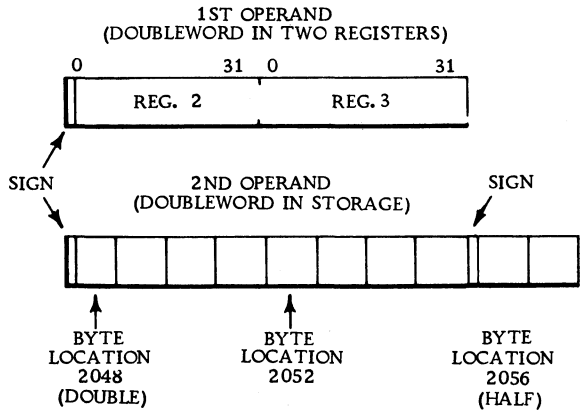
Before going on to more instructions, let's consider one use of the "logical add/subtract" instructions.

As you learned in the beginning of the "add" instruction section, only words and halfwords can be added. What happens when a programmer desires to add two doublewords? What he can do is place the high-order word of the 1st operand in one register and the low-order word in another. Then he can logically add the low-order word of the 2nd operand to the low-order word of the 1st operand. There is no fixed point overflow possible. He can then test the condition code for a carry. If a carry resulted, he can add a value of +1 to the high-order word of the 1st operand. In any case, the last step would be to algebraically add the high-order word of the 2nd operand to the high-order word of the 1st operand. The following flowchart and sample program should illustrate this more clearly.

FLOW CHART



OPERANDS



PROGRAM

Assume: HALF is the address of a halfword containing a value of +1.

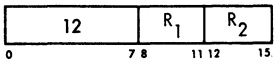
```
        AL      3,DOUBLE+4      Add logical DOUBLE+4 (location 2052) to Reg. 3.
        BC      12,NOCARRY      Branch if there is no carry to the algebraic add.
        AH      2,HALF          Add +1 to Reg. 2.
NOCARRY A       2,DOUBLE        Add algebraic DOUBLE (location 2048) to Reg. 2.
```

SPECIAL LOAD INSTRUCTIONS

In addition to the "load" instructions previously covered (LR, L, LH) there are several special purpose load instructions. These are special in the manner in which they affect the condition code and how they may also change the data as it is loaded. These instructions are all register to register operations.

Load and Test

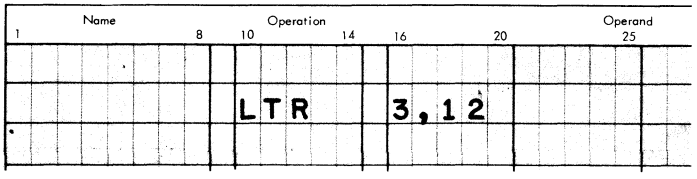
LTR R_1, R_2 [RR]



- The second operand is placed in the first operand location.
- The second operand is tested for sign and magnitude and the condition code is set accordingly.

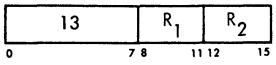
Condition Code:
- 0 R_2 is zero
- 1 R_2 is less than zero
- 2 R_2 is greater than zero
- 3 --

Program Interruptions:
None



Load Complement

LCR R_1, R_2 [RR]

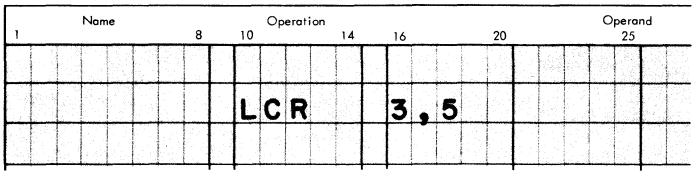


The two's complement of the second operand is placed in the first operand location.

- Positive numbers are made negative.
- Negative numbers are made positive.

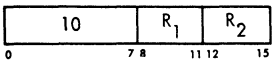
Condition Code: (The condition code is set after the operation is completed.)
- 0 Result is zero
- 1 Result is less than zero
- 2 Result is greater than zero
- 3 Overflow

Program Interruptions:
Fixed point overflow.



Load Positive

LPR R_1, R_2 [RR]

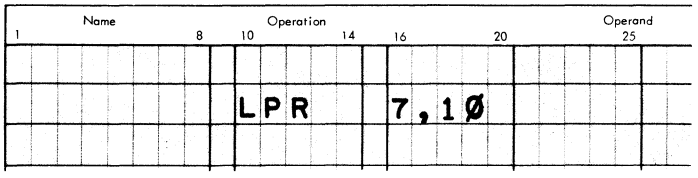


The absolute value of the second operand is placed in the first operand location.

- Positive numbers remain unchanged.
- Negative numbers are made positive.

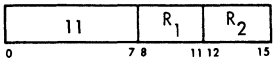
Condition Code: (The condition code is set after the operation is completed.)
- 0 Result is zero
- 1 --
- 2 Result is greater than zero
- 3 Overflow

Program Interruptions:
Fixed point overflow.



Load Negative

LNR R_1, R_2 [RR]

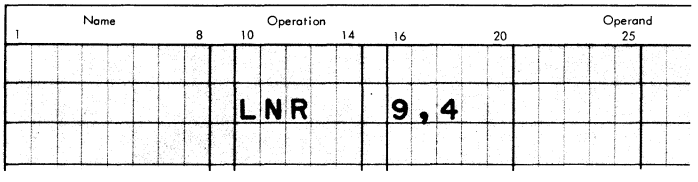


The two's complement of the absolute value of the second operand is placed in the first operand location.

- Positive numbers are made negative.
- Negative numbers remain unchanged.

Condition Code: (The condition code is set after the operation is completed.)
- 0 Result is zero
- 1 Result is less than zero
- 2 --
- 3 --

Program Interruptions:
None



Mnemonic	Hex Op Code	Data Flow
LTR	12	Load and test (only sets condition code)
LCR	13	Load complement (complements the data)
LPR	10	Load positive (complements negative data)
LNR	11	Load negative (complements positive data)

1. As indicated by the last letter of their mnemonics, the four instructions you just read about use the ____format. All four of these instructions can change the c_____c_____.

• • •

RR; condition code

2. The only difference between the LR instruction and the "load and test" (LTR) instruction is the effect on the_____ _____.

• • •

condition code; By specifying the same register in the R1 and R2 fields, the LTR instruction can be used to test the contents of a register.

3. The "Load Complement" (LCR) instruction will change the condition code and will also_____ data.

• • •

complement

4. With the LCR instruction, the condition code shows the status of the data_____(before/after) it was complemented.

• • •

after

5. The "Load Positive" (LPR) instruction only complements_____(positive/negative) numbers.

• • •

negative; The LPR instruction Loads Positive numbers into the register regardless of the original sign of the numbers.

6. The "Load Negative" (LNR) instruction complements_____numbers.

• • •

positive; The LNR instruction Loads Negative numbers into the register regardless of the original sign of the numbers.

7. The only positive number that cannot be complemented by either the LCR or the LNR instruction is z_____.

• • •

zero

8. Given the following list of mnemonics, indicate the effect (changed/unchanged) on the condition code and on the data as it is loaded.

Mnemonic	Condition Code	Data
LR	_____	_____
L	_____	_____
LH	_____	_____
LTR	_____	_____
LCR	_____	_____
LPR	_____	_____
LNR	_____	_____

• • •

Mnemonic	Condition Code	Data
LR	Unchanged	Unchanged
L	Unchanged	Unchanged
LH	Unchanged	Unchanged
LTR	Changed	Unchanged
LCR	Changed	*All data is complemented
LPR	Changed	Negative data is complemented
LNR	Changed	*Positive data is complemented

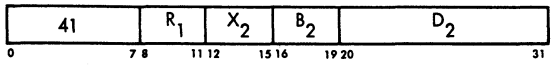
*With the Exception of Zero

LOAD ADDRESS INSTRUCTION

The "load address" instruction is another of the special purpose instructions. It is ordinarily used to load into a general register, for later usage or modification, the actual value of a symbolic address.

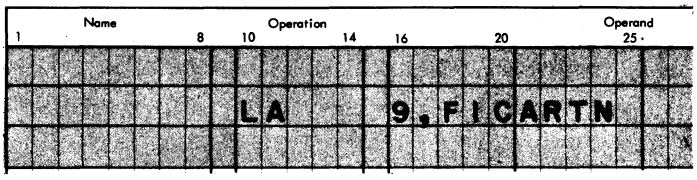
Load Address

LA $R_1, D_2(X_2, B_2)$ [RX]



- The address of the second operand is stored in bits 8 - 31 of the general register specified by the first operand.
- Bits 0 - 7 of the register are set at zero.

Condition Code:
The code remains unchanged.
Program Interruptions:
None.



* * *

1. LA is the mnemonic for the "_____ _____" instruction.

• • •

"load address"

2. The LA instruction uses the _____ format.

• • •

RX

3. The "load address" instruction will place in the specified register: (Circle one of the following.)
   a. A word from main storage.
   b. The generated storage address.

• • •

b

4. The generated 24-bit storage address will be placed in bits_____through_____of the specified register.

• • •

8; 31

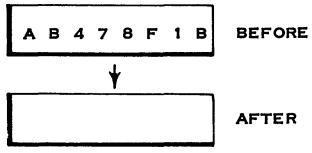
5. As a result of an LA instruction, bits 0-7 of the specified register: (Circle one of the following.)
   a. Remain unchanged.
   b. Are zeroed out.

• • •

b

6. Given the following "load address" instruction, show the resulting contents of the specified register.

LA 1,800(0,0)

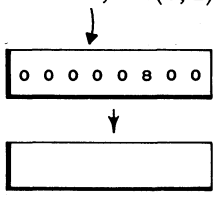


• • •

800 (decimal)

7. Given the following "load address" instruction, show the resulting contents of the register.

LA 1,800(0,1)



• • •

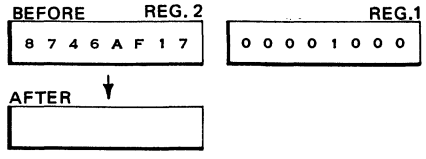
1600; In the preceding problem, the contents of register 1 were used as a base address in generating an effective storage address:

	Base Address	8 0 0
+	Displacement	8 0 0
	Effective Address	1 6 0 0

The effective address was then placed in register 1.

8. Given the following "load address" instruction, show the resulting contents of the register.

LA 2,0 (1,1)

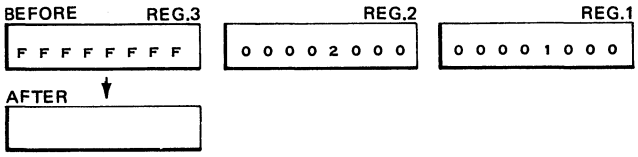


• • •

0 0 0 0 2 0 0 0

9. Given the following "load address" instruction, show the resulting contents of the register.

LA 3, 0(2, 1)



• • •

0 0 0 0 3 0 0 0

The preceding instructions show how successive base addresses could be loaded into general registers (although it is not usually done this way). Let's take another look at these instructions in a symbolic program.

```
LA    1, 2048 (0, 0)
LA    1, 2048 (0, 1)
LA    2, 0 (1, 1)
LA    3, 0 (2, 1)
```

NOTE: Refer to System/360 Reference Data Card (X20-1703) for meaning and sequence of symbolic notation. See the Operand column under Standard Instruction Set.

10. Examine the preceding program. Then indicate below (decimally) the base address that will be in each register at the completion of the program.

Register 1_____
Register 2_____
Register 3_____

• • •

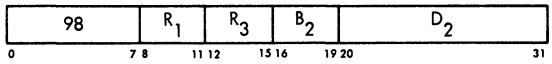
Reg. 1 4096
Reg. 2 8192
Reg. 3 12288

LOAD MULTIPLE INSTRUCTION

The "load multiple" instruction is closely associated with the "store multiple" instruction as well as the "load" (L) instruction. The "load multiple" instructions load more than one consecutive general registers with data from consecutive fullword storage locations.

Load Multiple

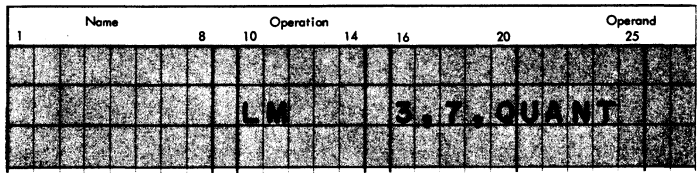
LM ***R_1, R_3, $D_2(B_2)$*** ***[RS]***



The data in a set of consecutive fullword storage locations starting at the address specified by the second operand, are placed in consecutive general registers starting at the register specified by R_1 and ending at the register specified by R_3.

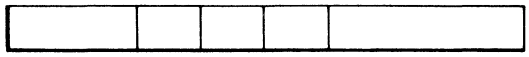
- The second operand must be on a fullword integral boundary.
- The general registers are loaded in ascending order of their addresses starting with R1.
- Register 0 follows register 15 as a "wraparound" condition is permitted.

Condition Code:
The code remains unchanged.
Program Interruptions:
Addressing
Specification

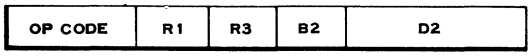


Note: The storage location specified by the second operand (QUANT in the above example) must be on a fullword integral boundary.

1. The LM instruction uses the RS format. Label the fields of the RS format.



• • •



2. Like the L instruction, the LM_____ (changes/does not change) the condition code.

• • •

does not change

3. LM 2,4,2004(0)

In the above LM instruction, byte locations 2004 through 2015 will be loaded into registers_____ through_____.

• • •

2; 4

4. LM 2,4,2004(0)

In the above LM instruction, register 3 will be loaded with the contents of byte locations_____ through_____.

• • •

2008; 2011

5. LM 0,15,2002(0)

The above LM instruction will result in a_____ exception.

• • •

specification; Address 2002 is okay for halfwords but not for fullwords. The LM instruction uses the entire contents (fullword) of the registers.

EDIT AND MARK INSTRUCTION

The "edit and mark" instruction is identical to the "edit" (ED) instruction except that provision is made in the "edit and mark" instruction for insertion of the dollar sign in a "floating dollar sign" operation. The "floating dollar sign" operation permits placing the dollar sign immediately to the left of the first significant digit in the edited result.

Edit and Mark

EDMK $D_1(L, B_1), D_2(B_2)$ **[SS]**

DF	L	B_1	D_1	B_2	D_2
0–7	8–15	16–19	20–31	32–35	36–47

The format of the second operand is changed from packed to zoned and edited under the control of the edit pattern (the first operand).

- The address of the first significant digit encountered prior to a significance-start character is stored in bits 8-31 of general register 1.
- The address is not inserted in GR1 when significance is forced by the significance-start character.
- Bits 0-7 of GR1 are not changed.
- This instruction facilitates the programming of the floating dollar sign.

Condition Code:

0 Edited result is zero
1 Edited result is less than zero
2 Edited result is greater than zero
3 --

Program Interruptions:

Operation (If decimal feature is not installed)
Protection
Addressing
Data

```
Name     Operation  Operand
         MVC   WORK,PATTRN      MOVE PATTERN INTO
*                               WORK
         LA    1,WORK+6         STORE ADDRESS OF
*                             SIGNIFICANCE-START + 1
         EDMK  WORK,DATA        EDIT DATA
*
         BCTR  1,Ø              REDUCE ADDRESS BY 1
         MVC   Ø(1,1),DOLLAR      MOVE $ SIGN TO
*                               ADDRESS
*
PATTRN   DC    X'4Ø2Ø6B2Ø2Ø214B2Ø2Ø4ØC3D9'
DOLLAR   DC    C'$'
```

1. ED is the mnemonic for the "edit" instruction while EDMK is the mnemonic for the "____ ____ ____" instruction.

• • •

"edit and mark"

2. Is there anything that the ED instruction can do that the EDMK instruction can't do?_____ (Yes/No)

• • •

No

3. What, then, is the difference between the ED and EDMK instructions?_____

• • •

The EDMK instruction causes the address of the 1st significant digit of the result to be placed in general register 1.

4. What happens on an EDMK instruction when significance is started by a significant start character?_____

• • •

No address is placed in register 1

5. Does the EDMK instruction insert the floating currency symbol?_____

• • •

no; the symbol must be inserted by subsequent instructions.

6. The address placed in register 1 is: (Circle one of the following.)
   a. The location where the currency symbol (such as $) should be inserted.
   b. The location +1 where the currency symbol should be inserted.

• • •

b; Register 1 has the address of the 1st significant digit. The currency symbol (such as $) should be placed just to the left of this digit.

7. What instruction can be used to reduce the address in register 1 by one?_____

• • •

Branch and Count; without a branch.

By using the RR format and an R2 field of zero, register 1 can be reduced. For example:

```
BCTR   1,0
```

After the BCTR instruction, the "move character" instruction (MVC) can use register 1 as a base register and move a dollar sign (currency symbol) into the desired location. (See example presented with the description of the EDMK instruction.)

COMPARE LOGICAL INSTRUCTIONS

You learned three "compare" instructions when you were studying the fixed point instructions. Their mnemonics are:

CR	Compare, RR format
C	Compare, RX format
CH	Compare Halfword, RX format

These three "compare" instructions compared on an algebraic basis. In other words, they treated the operands as signed binary integers. The operands were either positive or negative numbers. The "compare logical" instructions you will now learn also treat the operands as binary information. However, they will be considered as unsigned binary fields. For example, consider the comparison of the following binary fields on an algebraic basis.

	Sign	Integer	
Compare	0	0 0 0 0 0 0 0 1	1st Operand
Algebraic	1	1 1 1 1 1 1 1 1	2nd Operand

1. Because the 1st operand is a positive number (+1) and the 2nd operand is a negative number (-1), the 1st operand is high and the condition code would be set to____.

• • •

2

2. If the same fields were compared on a logical basis, they would be treated as unsigned integers and the absolute values would be compared as follows:

Compare	0 0 0 0 0 0 0 1	1st Operand
Logical	1 1 1 1 1 1 1 1	2nd Operand

In the above example, the 1st operand would compare low and the condition code would be set to_____. This occurs because an unsigned value of 1 is being compared with an unsigned value of 255.

• • •

1

The programmer must know what format his data is in before he can compare it. If his data consists of signed binary words or halfwords, he would use his three "algebraic" instructions: CR, C, CH. If his data consists of unsigned binary fields, he would use the "logical" instructions. As a point of interest, the EBCDIC code is so arranged that the special and alphameric characters will collate on a binary basis. That is, the "compare logical" instructions are used to compare EBCDIC characters.

Let's look at the coding for some EBCDIC characters.

"A"	11000001
"Z"	11101001

3. On a compare logical basis, which is low?_____ ("A" or "Z")

• • •

"A"

4.

"1"	11110001
"Z"	11101001

On a compare logical basis, which is low?_____ ("1" or "Z")

• • •

"Z"

5.

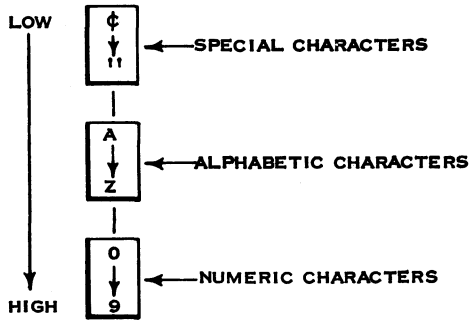
"#"	01111011
"A"	11000001

On a compare logical basis, which is low?_____ ("#" or "A")

• • •

"#"

The preceding examples should agree with the collating sequence you may be familiar with in your past experience with punched card equipment or equipment which used the standard BCD code (BA 8421). This is illustrated as follows:

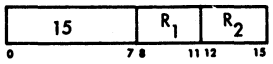


You now have an idea of the difference between algebraic and logical comparisons.

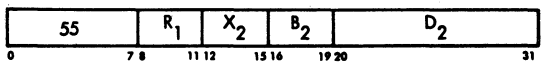
Mnemonic	Format	Comparison	Length of Operands
CLR	RR	Register vs. Register	Fullwords
CL	RX	Storage vs. Register	Fullwords
CLI	SI	Immediate vs. Storage	One byte
CLC	SS	Storage vs. Storage	1-256 bytes

Compare Logical

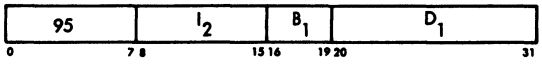
CLR R_1, R_2 [RR]



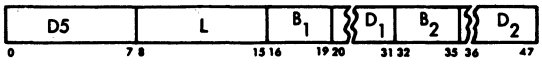
CL $R_1, D_2(X_2, B_2)$ [RX]



CLI $D_1(B_1), I_2$ [SI]



CLC $D_1(L, B_1), D_2(B_2)$ [SS]



All:
The first operand is compared with the second operand and the result is indicated in the condition code.

- Comparison is binary (that is bit by bit).
- Comparison proceeds from left to right.
- Comparison ends as soon as inequality is found.

CL only:

- The fullword second operand must be on a fullword integral boundary.

CLI only:

- One byte at the storage location specified by the first operand is compared with one byte of immediate data.

CLC only:

- The number of bytes to be compared is specified by the implicit or explicit length of the first operand.

Condition Code:

0 Operands are equal
1 First operand is low
2 First operand is high
3 --

Program Interruptions:

Addressing
Specification (CL only)

Name	Operation	Operand
	CLR	2,3
	CL	4,TEST
	CLI	CODE,C'E'
	CLC	NAME,=C'SMITH'

• • •

1. In both the CL and CLR instruction, the 1st operand is the register specified by the_____ field. The instructions cause a_____(logical/algebraic) comparison. As a result of the comparison, the _____ ____ is set.

• • •

R1; logical; condition code

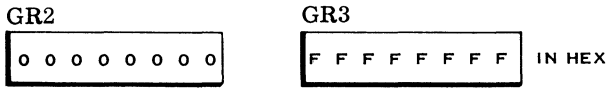
2. The condition code settings of 0, 1, 2 indicate that the_____(1st/2nd) operand is equal, low, or high compared to the_____(1st/2nd) operand. After a compare operation, it is impossible to have a condition code of _____.

• • •

1st; 2nd; 3

3. Given the following CLR instruction, indicate the resulting condition code bits in the PSW.

CLR 2,3



Condition Code =_____

• • •

1; if the CR (compare algebraic) instruction had been used, the 1st operand would have been high.

4. Given the following CL instruction, indicate the resulting condition code.

CL 2,800(0,0)

Condition Code =_____

• • •

2; By examining the four high-order bits, you can see that the 1st operand is high.

1st operand - 1000
2nd operand - 0111

5. Besides the CLR and CL instructions, System/360 can also compare logical using the SI and SS formats. CLI is the mnemonic for the " _____ _____ _____ " instruction.

• • •

"compare logical immediate"

6. The "compare logical immediate" instruction uses the SI format. In this format, the 1st operand is in _____ (main storage/the instruction).

• • •

main storage

7. The CLI instruction compares on a(n)_____ (algebraic/logical) basis. The comparison is between one byte in storage and one byte in the _____.

• • •

logical; instruction

8. CLI 2048(0),X'AF'

LOCATION 2048

In the above CLI instruction, the 1st operand is____ (low/high) and the resulting condition code is_____.

• • •

low; 1; In the SI format, the 1st operand is in main storage.

9. CLI 2048(0),X'07'

LOCATION 2048

In the above CLI instruction, the 1st operand is____ (low/high) and the resulting condition code is_____.

• • •

high; 2

10. The CLR and CL instructions compare one_____ (byte/word/halfword) of data with another.

The CLI instruction compares one_____ (byte/word/halfword) of data with another.

• • •

word; byte

11. The compare logical operation can also be done with the_____(SS/RS) format.

• • •

SS

12. CLC is the mnemonic for the "compare logical" instruction which uses the_____ format.

• • •

SS

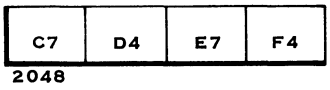
13. CLC means Compare Logical Characters. This instruction has an 8-bit length code and can compare up to_____characters.

• • •

256

14. The name of the CLC instruction indicates that characters are being compared. Actually, bytes are being compared on an unsigned binary (logical) basis. As was previously pointed out, however, the EBCDIC code assigned to characters is arranged so that they will collate on a binary basis.

CLC 2048(1, 0), 2050(0)



In the above CLC instruction,_____character(s) will be compared and the condition code will be set to_____.

• • •

two (one from each operand) 1

15. The coding of the byte at location 2048 above could represent the EBCDIC character "_____". Use your System/360 Reference Data Card (X20-1703) to answer these questions.

• • •

"G"; Because hex C7 equals the EBCDIC "G"

16. The coding of the byte at location 2049 could represent the EBCDIC character "_____ ".

• • •

"M"

17. The coding of the byte at location 2050 could represent the EBCDIC character "_____".

• • •

"X"

18. The coding of the byte at location 2051 could represent the character "_____".

• • •

"4"

19. CLC 2048(4, 0), 2052(0)

LOCATIONS 2048 – 2051 JOHN
LOCATIONS 2052 – 2055 LUKE

Given the above characters and CLC instruction, the condition code will be set to_____.

• • •

1; In the preceding problem, JOHN was the 1st operand and LUKE was the 2nd operand. The high-order character (J) of the 1st operand was lower than the high-order character (L) of the 2nd operand.

"J" - 11010001
"L" - 11010011

20. CLC 2048(8, 0), 3840(0)

LOCATIONS 2048 – 55 – JOHNSTON
LOCATIONS 3840 – 47 – JOHANSEN

Given the above characters and CLC instruction, the condition code will be set to_____.

• • •

2; On the first three high-order characters (JOH), both operands are equal. On the fourth character, the 1st operand will compare high as follows:

1st operand - "N" - 11010101
2nd operand - "A" - 11000001

21. You should now realize that the comparing is done for all practical purposes from_____(left to right/right to left).

• • •

left to right

22. As a result of comparing the bytes from left to right, it is not necessary to examine the entire field. The compare operation assumes that the fields are equal to begin with. In examining the bytes from left to right, the system can end the compare operation as soon as it finds an u_____condition.

• • •

unequal

23. List the mnemonics and the instruction formats of the four "compare logical" instructions.

Mnemonic	Instruction Format
_____	_____
_____	_____
_____	_____
_____	_____

• • •

Mnemonic	Instruction Format
CLR	RR
CL	RX
CLI	SI
CLC	SS

24. List the mnemonics and formats of the three "compare algebraic" instructions.

Mnemonic	Instruction Format
_____	_____
_____	_____
_____	_____

• • •

Mnemonic	Instruction Format
CR	RR
C	RX
CH	RX

25. What is the main difference between the CR and CLR instructions?

_____
_____
_____
_____

• • •

The CR instruction will treat the contents of a particular register as a signed integer (sign and 31 bits). The CLR instruction treats the contents of the same register as an unsigned 32-bit integer. As a result, the condition code setting may vary, depending on the instruction used.

26. Given the contents of the following two registers, indicate the resulting condition code for the instructions shown.

REG 2	REG 3
8 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1

CONDITION CODE

a. CR 2,3 _____

b. CLR 2,3 _____

• • •

a. 1 - Reg 2 has a negative number.
b. 2 - Reg 2 has a higher value.

27. Assume that a card record punched in standard card code has been read in main storage. The record contained alphabetic characters. To compare two fields in this record, which would you do?_____ _____ (compare logical/compare algebraic).

• • •

compare logical

TRANSLATE INSTRUCTION

Two "logical" instructions for you to study are the "translate" instruction and the "translate and test" instruction. If you do not have systems experience and are unfamiliar with the terms translate or table look up, these instructions may be among the most difficult of those you have encountered. Therefore, let's examine the concept of translating before reading the description of these instructions.

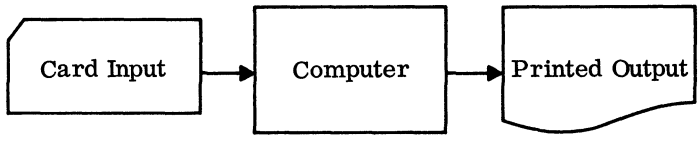
First of all, there is data to be translated. This data may be in any code form we wish. The only code you have studied in the System/360 is EBCDIC. There are, of course, other codes in use with computers. For instance, there is an 8-bit paper tape code and the 8-bit ASCII code. The "translate" instruction will allow us to translate data from one code to another, byte by byte.

1. The "translate" instruction will allow us to translate bytes of data: (Choose one of the following.)
   a. From one character code to any other character code.
   b. Only from EBCDIC to some other character code.
   c. Only to EBCDIC from some other character code.
   d. Only from EBCDIC to ASCII.

• • •

a; The bytes to be translated can be in <u>any</u> character code. These bytes can be translated to <u>any</u> other desired code.

Let's look at this concept of translating from a programmer's viewpoint and see how he would handle a <u>simplified</u> translating problem.



2. The basic job that is to be accomplished is the printing of a report. Input to the system is in the form of _____.

• • •

IBM punched cards

Assume that the cards were punched on a card punch that <u>did not</u> have special character keys. The machine could only punch numeric and alphabetic characters.

The operator who punched the cards used alphabetic symbols to represent the special characters. For example, the character P was used to represent a + sign.

Special Characters	Alphabetic Symbol
+	P
-	M
#	N
$	D
¢	C
&	A

The chart shows how each of the special characters was represented by an alphabetic character.

3. Given the following listings on a source document, indicate the characters that one operator actually punched in the cards.

Source Document	IBM Card
-79¢	_____
$120+	_____
E & F & # 3	_____

• • •

M79C; D120P; EAFAN3

4. The input cards that are used in our simplified application _____ (do/do not) contain special character punching.

• • •

do not

5. The output of this simplified application is to be in _____form. It is desired to have the listings on the printed report contain the special characters rather than the alphabetic symbols. Therefore, the computer must convert or t_____the input data before sending it to the printer.

• • •

printed; translate

Now, let's see how the programmer can use the "translate" instruction to solve the problem just discussed.

First, two tables must be established. They are the function table and the argument table.

6. The f_____table consists of the desired characters. In our application, the function table will consist of the_____(special/alphabetic) characters.

• • •

function; special

7. The a_____table consists of all the data that may have to be converted. In our application, the argument table will consist of the_____ symbols.

• • •

argument; alphabetic

8. In the next step, the programmer writes down all the possible data to be converted. Then he arranges it in binary bit sequence and forms the a_____table.

Argument Table

A		1100 0001
	US (Unused Symbol)	1100 0010
C		1100 0011
D		1100 0100
	US	1100 0101
	US	1100 0110
	US	1100 0111
	US	1100 1000
	US	1100 1001
	US	1101 0001
	US	1101 0010
	US	1101 0011
M		1101 0100
N		1101 0101
	US	1101 0110
P		1101 0111

• • •

argument

9. Now the programmer can make up the f_____ table. The table will indicate where the s_____ characters should be stored so that they can be easily located and used in place of the a_____ symbols (argument table).

• • •

function; special; alphabetic

10. Argument Table — Function Table (Table address is 6807)

Argument Bytes			Function Bytes	Storage Locations
A		1100 0001	&	7000
	US	1100 0010		7001
C		1100 0011	¢	7002
D		1100 0100	$	7003
	US	1100 0101		7004
	US	1100 0110		7005
	US	1100 0111		7006
	US	1100 1000		7007
	US	1100 1001		7008
	US	1101 0001		7016
	US	1101 0010		7017
	US	1101 0011		7018
M		1101 0100	-	7019
N		1101 0101	#	7020
	US	1101 0110		7021
P		1101 0111	+	7022

The function table is actually located in s_____.

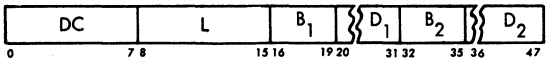
The argument table is made up on paper by the programmer. Its only use is to create the f_____ table in_____.

• • •

storage; function; storage

Translate

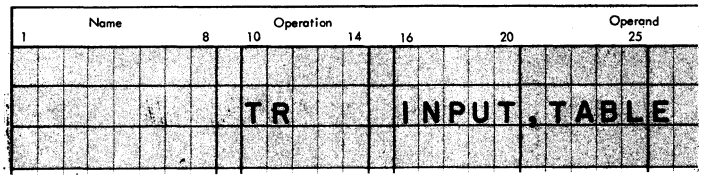
TR $D_1(L, B_1), D_2(B_2)$ [SS]



The value of the eight bit bytes of the first operand (arguments) are added to the address of the table specified by the second operand (functions) and the function byte at the effective address location replaces the corresponding argument byte.

- The argument bytes are translated one at a time.
- Translation proceeds from left to right.
- The number of bytes to be translated is determined by the implied or explicit length of the first operand.

Condition Code:
The code remains unchanged.
Program Interruptions:
Protection.
Addressing.



• • •

11. The byte or bytes in the first operand are the characters that are to be converted or t_____. They are called a_____bytes. In the simplified application that you just studied, a first operand argument byte could be a_____(D/$). Note that the bytes in the first operand are converted one byte at a time.

• • •

translated; argument; D

12. The second operand is the f_____table. In the simplified application, the address of the second operand would be_____.

• • •

function; 6807

13. The "translate" instruction does the following:
    1. Takes the binary bit value of an argument byte and adds it to the second operand's address.
    2. The resulting address is used to locate a function byte.
    3. The function byte replaces the argument byte (1st operand).

The binary bit value of the first argument byte (A) is_____in decimal.

• • •

193

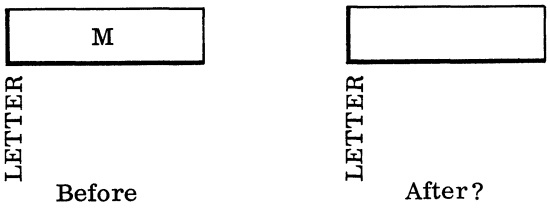
14. The addition of this value to the decimal value of the function table address (6807 in our example) results in the storage location address of the desired function byte. The address of the function byte corresponding with M is_____in decimal.

• • •

7019; M=11010100 binary=212 decimal
212+6807=7019

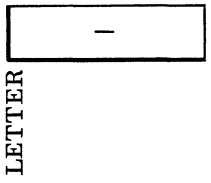
15. The function byte will then replace the argument byte in storage.

TR LETTER(1), FUNCTAB
If LETTER is the address of the letter M to be translated



What will the storage location LETTER contain after the execution of the instruction?

• • •



M=1101 0100 =212 decimal
FUNCTAB =6807 decimal
Function =7019 decimal
Function character=0110 0000 = —

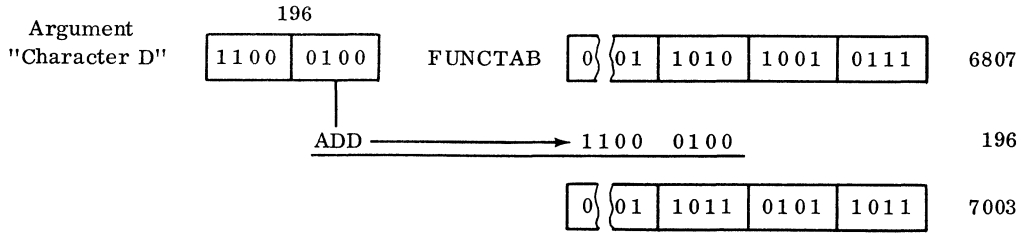
The execution of the TR instruction is completed when — replaces M at the storage location LETTER.

16. Note that more than one character may be translated by a single TR instruction but the characters will be handled one at a time. The number of characters may be explicit or implied. The instruction TR LETTERS(7), FUNCTAB will translate _____ characters each time it is executed.

• • •

7

```
TRANS     TR    ARGUMENT, FUNCTAB
-------OR-------------
TRANS     TR    300(1),6807 (Decimal Numbers)
```



Resulting Address of Function Byte

(Location in Decimal = 7003)

17. The above instruction will translate___byte(s). The character to be translated is a___. The instruction goes to location 7003 in the function table (refer to the preceding simplified function table) and finds a____character. It takes this character and puts it in location___where it replaces the D.

• • •

1; D; $; 0300

18. Now, let's go back and review the entire concept of translating and use a more typical application.

To translate, a table of the desired code must be available. For instance, assume that we wished to translate from EBCDIC to the 8-bit ASCII code. For simplicity we will only deal with the characters A-H. As a result our table will be only 8 bytes long.

Function Bytes	1 0 1 0 0 0 0 1	-A	Table of 8 Bit ASCII (A to H)
	1 0 1 0 0 0 1 0	-B	
	1 0 1 0 0 0 1 1	-C	
	1 0 1 0 0 1 0 0	-D	
	1 0 1 0 0 1 0 1	-E	
	1 0 1 0 0 1 1 0	-F	
	1 0 1 0 0 1 1 1	-G	
	1 0 1 0 1 0 0 0	-H	

The above table is located in main storage in 8 successive byte locations. As you can see, the bytes in the table are called_____bytes.

• • •

function

19. The function bytes represent: (Choose one of the following.)
    a. The bytes to be translated.
    b. The desired character code.

• • •

b; The desired character code.

20. Besides the table of function bytes, which represent the desired code, there must also be data bytes which need translation. The following is a five-character record which needs translating.

Argument Bytes	1 1 0 0 0 1 1 0	-F	Data to be Translated
	1 1 0 0 0 0 0 1	-A	
	1 1 0 0 0 1 0 0	-D	
	1 1 0 0 0 1 0 1	-E	
	1 1 0 0 0 1 0 0	-D	

The bytes to be translated are called_____bytes.

• • •

argument

21. The above record of five EBCDIC characters is to be translated by using a table of f_____bytes.

• • •

function

22. The "translate" instructions consist of replacing the characters to be translated with the characters of the desired code. In other words, the _____ bytes are replaced with the correct_____bytes.

• • •

argument; function

23. The "translate" instruction will replace all of the argument bytes with the desired characters from the function table.

Given the following function table and argument bytes, show the resulting contents of the argument field.

	Before	After
1 1 0 0 0 1 1 0		
1 1 0 0 0 0 0 1		
1 1 0 0 0 1 0 0		
1 1 0 0 0 1 0 1		
1 1 0 0 0 1 0 0		

Argument Field of Five EBCDIC Characters

1 0 1 0 0 0 0 1
1 0 1 0 0 0 1 0
1 0 1 0 0 0 1 1
1 0 1 0 0 1 0 0
1 0 1 0 0 1 0 1
1 0 1 0 0 1 1 0
1 0 1 0 0 1 1 1
1 0 1 0 1 0 0 0

Function Table of ASCII Bytes

• • •

1 0 1 0 0 1 1 0
1 0 1 0 0 0 0 1
1 0 1 0 0 1 0 0
1 0 1 0 0 1 0 1
1 0 1 0 0 1 0 0

24. You should now know what is meant by a function byte or an argument byte. You should also realize that the argument bytes are to be replaced by the desired function bytes. We can review the translating concept by asking ourselves "How does the machine know which function bytes to select?" The answer lies in the organization of the function table. This table must be arranged so that the desired characters match the binary sequence of the argument table. This is shown as follows:

EBCDIC	ASCII
1 1 0 0 0 0 0 1	1 0 1 0 0 0 0 1
1 1 0 0 0 0 1 0	1 0 1 0 0 0 1 0
1 1 0 0 0 0 1 1	1 0 1 0 0 0 1 1
1 1 1 1 0 1 1 1	0 1 0 1 0 1 1 1
1 1 1 1 1 0 0 0	0 1 0 1 1 0 0 0
1 1 1 1 1 0 0 1	0 1 0 1 1 0 0 1

Table of all possible argument bytes is arranged on paper, in binary bit sequence. The table is used to develop the correct sequence for the function table.

Table of function bytes is arranged to match the respective argument bytes.

The table of FUNCTION bytes in storage is arranged so that: (Choose one of the following.)

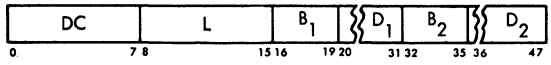
a. The function bytes are in binary sequence.
b. The binary sequence of the argument bytes determines the sequence of FUNCTION bytes.

• • •

b

Translate

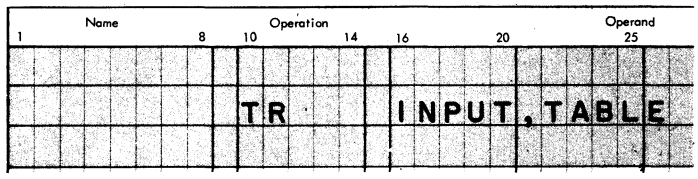
TR ***$D_1(L, B_1), D_2(B_2)$*** ***[SS]***



The value of the eight bit bytes of the first operand (arguments) are added to the address of the table specified by the second operand (functions) and the function byte at the effective address location replaces the corresponding argument byte.

- The argument bytes are translated one at a time.
- Translation proceeds from left to right.
- The number of bytes to be translated is determined by the implied or explicit length of the first operand.

Condition Code:
The code remains unchanged.
Program Interruptions:
Protection
Addressing



• • •

25. TR is the mnemonic for the "_____" instruction.

• • •

"translate"

26. The "translate" instruction uses the___ __format.

• • •

SS

27. The 1st operand of the TR instruction represents: (Choose one of the following.)
    a. The bytes to be translated.
    b. The desired coded bytes.

• • •

a

28. The 2nd operand of the TR instruction represents: (Choose one of the following.)
    a. The function bytes.
    b. The argument bytes.

• • •

a

29. The function table must be long enough to take care of all expected bit combinations of the argument bytes.

    The length code refers to: (Choose one of the following.)
    a. The argument bytes.
    b. The function bytes.
    c. Both argument and function bytes.

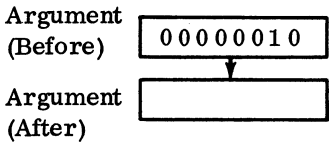
• • •

a

30. To find the desired character in the function table, the numeric value of the argument byte is added to the address at the beginning of the table.

    Given the following argument byte, what bit combination will replace it?

REPLCE1 TR ARGUMENT(1), FUNCTION



Argument located at $2048_{(10)}$
Function located at $3840_{(10)}$

Function	In Decimal
00101100	3840
10110001	3841
10101111	3842
11110011	3843
10100101	3844
11000001	3845
11101000	xxxx

• • •

10101111; The 1st operand's numeric value (decimal 2) was added to the 2nd operand's address (decimal 3840). The byte at location 3842 replaced the 1st operand.

31. You should now understand why the function table must be arranged according to the binary sequence of the argument bytes. This is because the argument byte is added to the initial table address. The coded character at that location then replaces the argument byte.

    REPLCE2 TR ARGUMENT(24), FUNCTION

    Given the above "translate" instruction, how many arguments bytes will be translated?_____

• • •

24

32. REPLCE5 TR ARGUMENT, FUNCTION

    Given the above "translate" instruction, how many bytes are in the function table?_____This question can be tricky, so answer carefully.

• • •

Unknown; the proper function byte is selected from the table by adding the argument byte to the starting address of the table. As a result, the table might contain a maximum of 256 bytes. This would depend on the total number of characters in the codes involved.

33. Argument Byte — 00110001 numeric value =_____

    REPLCE6 TR ARGUMENT, FUNCTION

    Given the above "translate" instruction, show the address of the character that will replace the argument byte.

    Value of argument byte = 49, Address of FUNCTION = 3840

• • •

49; 3889; as shown below

The numeric value of the byte is added to the starting address of the function table:

3840 - Table Address
  49 - Argument Byte Value
3889 - Address of function byte selected to replace the argument byte.

34. Function table address - 3840

    Argument Byte - 11001001 Numeric value = 201

    REPLCE7 TR ARGUMENT, FUNCTION

    Given the above "translate" instruction and one of the argument bytes, show the address of the function byte that will be selected.

• • •

4041; as shown below.

3840 - Table Address
201 - Argument Byte Value
4041 - Address of function byte selected to replace the argument byte

35. Argument Byte - 11110111 numeric value = _____

Function = 3840

REPLCE8 TR ARGUMENT, FUNCTION

Given the above, show the address of the selected function byte.

• • •

247; 4087

36. Given the following data, show the contents of the argument field after the "translate" instruction is executed.

REPLCE9 TR ARGUMENT,FUNCTION

	Argument Field	Argument Data Before	Argument Data After	Function Table
Locations	2048	F7	_____	3840 0A
	2049	F2	_____	
	2050	61	_____	3937 11
	2051	F2	_____	3938 00
	2052	F5	_____	
	2053	61	_____	
	2054	F3	_____	4080 0A
	2055	F2	_____	4081 01
				4082 02
				4083 03
				4084 04
				4085 05
			Locations	4086 06
				4087 07
				4088 08
				4089 09

in decimal — in hex — in decimal — in hex

• • •

Location	After
2048	07
2049	02
2050	11
2051	02
2052	05
2053	11
2054	03
2055	02

Argument Hex Value	Decimal Value	Effective Function Address	Function
F7 =	247	4087	07
F2 =	242	4082	02
61 =	97	3937	11
F2 =	242	4082	02
F5 =	245	4085	05
61 =	97	3937	11
F3 =	243	4083	03
F2 =	242	4082	02

The "translate" instruction can be summarized as follows:

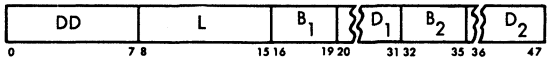
- The translation will be done by replacing an argument byte with a function byte from a table.
- The address of the 1st operand is the address of the argument bytes (those to be translated).
- The address of the 2nd operand is the address of the function table (those bytes which will be used to replace or translate the argument bytes).
- In order to obtain the proper function bytes, the table must be arranged according to the binary bit sequence of the argument byte.
- The argument byte is added to the function table address. The resulting address is used to select a byte from the function table and replace the argument byte with it.
- The "translate" instruction continues until all the argument bytes (determined by the length code) have been translated.

TRANSLATE AND TEST INSTRUCTION

The "translate and test" (TRT) instruction is similar to the "translate" (TR) instruction in the manner in which a function byte is located within a table from its corresponding argument byte. However, from there the operation changes. The "translate and test" instruction does not replace the argument byte with the function byte but tests the function byte for a non-zero condition and responds to the result of this test.

Translate and Test

TRT $D_1(L, B_1), D_2(B_2)$ [SS]



The value of the eight bit bytes of the first operand (arguments) are added to the address of the table specified in the second operand (functions) and the function bytes at the effective address location are tested for a non-zero value.

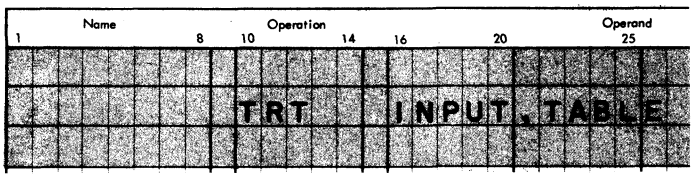
- If the function byte is zero, the operation continues to the next argument byte.
- If all the argument bytes result in a zero function byte, a condition code of zero is set.
- If a function byte is non-zero:
  a. The address of the related argument byte is stored in bits 8-31 of General Register 1. Bits 0-7 of GR1 are unchanged.
  b. The function byte is stored in bits 24-31 of General Register 2.
- If the non-zero function byte is not related to the last argument byte of the first operand, a condition code of 1 is set.
- If the non-zero function byte is related to the last argument byte of the first operand, a condition code of 2 is set.

Condition Code:

0 All function bytes are zero.
1 Non-zero function byte before first operand field is exhausted.
2 Last function byte is non-zero.
3 --

Program Interruptions:

Addressing



Upon finding a non-zero function in TABLE,
GR1 contains the address of the corresponding byte of the argument INPUT.
GR2 contains the non-zero function.
The Condition Code will be set to 1 or 2 as required.

• • •

1. In the "translate" instruction, the argument bytes are replaced with function bytes. In the "translate and test" instruction, the argument bytes _____ _____(are replaced with function bytes/remain unchanged).

• • •

remain unchanged

2. Is any translation actually done by the "translate and test" instruction?_____

• • •

no

3. The "translate and test" instruction tests the argument bytes by selecting the corresponding function bytes. The test results are recorded by changing the _____ _____.

• • •

condition code

4. How does the machine know which function bytes are to be selected?_____

• • •

It adds the numeric value of the argument byte to the starting address of the function table. The function byte at the resulting address is then tested.

5. The selected function byte is tested to see if it is_____.

• • •

zero

6. What happens if the function byte is zero?

• • •

The operation continues with the numeric value of the next argument byte being added to the table address and another function byte being selected.

7. If all of the function bytes selected by the argument bytes are zero, the operation is completed by setting the condition code to_____.

• • •

0

8. After a "translate and test" instruction, a condition code of 0 would indicate: (Choose one.)
   a. That one of the selected function bytes was zero.
   b. That all of the selected function bytes were zero.
   c. That none of the selected function bytes were zero.
   d. That all of the argument bytes were zero.

• • •

b; A condition code of 0 would indicate that all of the argument bytes has been used in selecting function bytes. It would also mean that all of the selected function bytes were zero. It does not mean that all of the function bytes in the table are zero. It means that the selected ones were zero.

The "translate and test" instruction is used to examine a data field (the argument bytes) for characters with special meaning. The function table would again be arranged (as in the "translate" instruction) according to the binary sequence of the data code.

For all characters that do not have a special meaning (nonsignificant characters), the function byte location would contain zero.

For all characters that do have a special meaning (significant characters), the function byte location would contain some non-zero bit configuration.

A resulting condition code of 0 would then indicate that the entire data field had been examined and that no significant characters were found. By significant characters, we mean those with special meaning in a data field.

9. If a character with special meaning (significant character) is found, the instruction is terminated. A significant character would be indicated by selecting a function byte that was_____(zero, non-zero).

• • •

non-zero

10. If a significant character is found before the entire data field is examined, the resulting code is 1 and the operation _____(continues/is terminated).

• • •

is terminated

11. After a TRT instruction, a condition code of 1 would mean: (Choose one of the following.)
   a. No significant character was found.
   b. All the argument bytes were used and a significant character was found.
   c. A significant character was found.
   d. One or more significant characters were found.

• • •

c; As soon as a significant character is found, the operation is terminated without testing any more bytes.

A condition of 1 then means that a significant character was found and some argument bytes haven't been tested. If the last argument byte is significant, the condition code is set to 2.

12. After a TRT instruction, a condition of 2 would mean: (Choose one of the following.)
   a. All argument bytes were used and none located a non-zero function.
   b. All of the argument bytes were not used. One of them was significant.
   c. The last argument byte located a non-zero function byte.
   d. All the argument bytes were used. One or more were significant.

• • •

c

13. After a "translate and test" instruction, which of the following condition codes would indicate that the entire field of argument bytes hasn't been examined? (Choose one of the following.)
   0
   1
   2

• • •

1

14. Which of the following condition codes would indicate that none of the argument bytes had special meaning? (Choose one of the following.)

0
1
2

• • •

0

15. Either a condition code of 1 or 2 will indicate that a significant character was found. Why then does the programmer need both settings?_____

_____

• • •

If the code were 1, the programmer would have to execute the TRT instruction again to see if the remaining argument bytes contained any characters with special meaning.

Decimal Data Field in Storage (Composed of Argument Bytes)

A	1	B	,	3	,	6	X

The purpose of the TRT instruction is to find significant characters in a data field. In the example above, the instruction could be used to find the location of commas in a decimal field. It would not make sense to know that there is a significant character without knowing where it is located. As a result, the TRT does more than just set the condition code. The address of the significant argument byte is placed in general register 1. The rest of general register 1 is left unchanged.

16. When a TRT instruction results in a condition code of 1 or 2, general register 1 will contain: (Choose one of the following.)

a. The address of a function byte.
b. A function byte.
c. The address of an argument byte.
d. An argument.

• • •

c; When an argument byte which contains a significant character is found, its address is placed in general register 1. If the condition code is 0, general register 1 is unchanged.

Besides placing the address of the significant argument byte in register 1, the TRT instruction will also place the non-zero function byte in bits 24-31 of general register 2. The rest of general register 2 remains unchanged.

17. Given the following TRT instructions, show the resulting condition code and the contents of general registers 1 and 2.

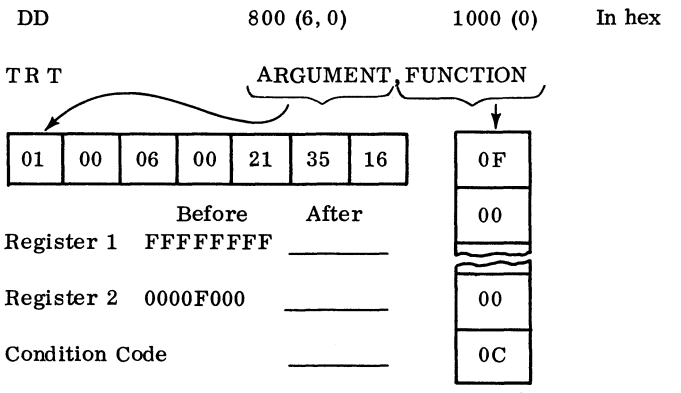


Table of Function Bytes

• • •

Register 1 FF000801; Register 2 0000F00F; Condition Code 1

In the preceding problem:

The 1st argument byte pointed to a zero function byte (the second byte in the function table).

The second argument byte pointed to a non-zero function byte (the first byte in the table).

The non-zero function byte is placed in the low order byte of register 2.

The address of the argument byte is placed in the low order 24 positions of register 1. The rest of registers 1 and 2 remains unchanged.

The length code indicates a total of 7 bytes. Since a significant character was detected prior to using all argument bytes, the condition code is 1.

The "translate and test" instruction can be summarized as follows:

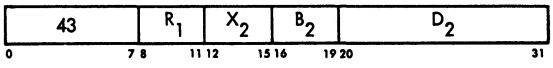
- The TRT instruction uses the SS format in which the length code gives the number of argument bytes less one.
- The 1st operand consists of the argument bytes (the field that is to be searched for characters that have special meaning).
- The 2nd operand consists of function bytes. These function bytes are pre-arranged according to the binary sequence of the argument bytes. The locations in this table that match the special meaning argument bytes have non-zero bit configurations.
- A numeric value of the argument byte is added to the starting address of the function bytes. The function byte at the resulting address is tested for a non-zero bit configuration. If it is non-zero, the operation is terminated. The address of the argument byte is put into register 1 and the corresponding non-zero function byte is placed in register 2. The condition code is set to 1 or 2, depending on whether or not the last argument byte has been translated.
- If all tested function bytes are zero, the operation is terminated by setting the condition code to 0. Registers 1 and 2 remain unchanged.

INSERT CHARACTER AND STORE CHARACTER INSTRUCTIONS

The "insert character" and "store character" are essentially one byte extensions of the fullword "load" (L) and "store" (ST) instructions.

Insert Character

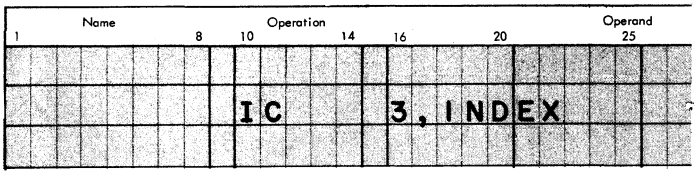
IC $R_1, D_2(X_2, B_2)$ [RX]



The eight bit byte at the second operand location is loaded into bits 24-31 of the general register specified in the first operand.

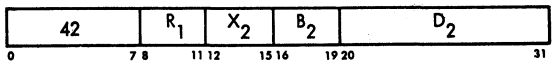
- Bits 0-23 of the register remain unchanged.

Condition Code:
The code remains unchanged.
Program Interruptions:
Addressing



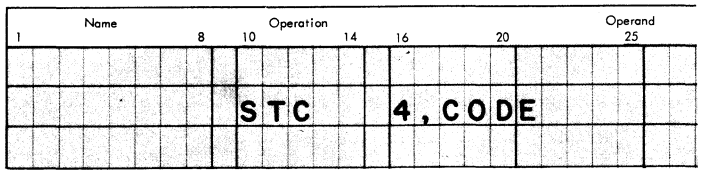
Store Character

STC $R_1, D_2(X_2, B_2)$ [RX]



Bits 24-31 of the register specified in the first operand are placed at the storage location specified in the second operand.

Condition Code:
The code remains unchanged.
Program Interruptions:
Protection
Addressing



• • •

1. IC is the mnemonic for the "_____ _____" instruction.

• • •

"insert character"

2. STC is the mnemonic for the "_____ _____" instruction.

• • •

"store character"

3. Both the IC and STC instructions use the__ __instruction format. These instructions _____ (change/do not change) the condition code.

• • •

RX; do not change

4. The "insert character" instruction will place the storage operand in the_____ _____byte of the specified register.

• • •

low order

5. The "insert character" instruction will place the storage operand in bits _____through_____ of the specified register.

• • •

24; 31; the low-order byte.

6. The remaining bits (0-23) of the specified register: (Choose one.)

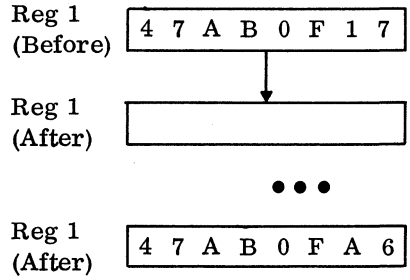
   a. Are zeroed out.
   b. Remain unchanged.

• • •

b

7. Given the following IC instruction, show the resulting contents of the specified register.

```
DEPOSIT     IC      1,MASK1     A6
                                MASK1
```



• • •

8. Was the condition code changed by the preceding instruction?_____

• • •

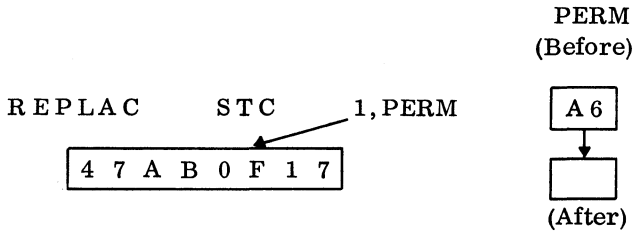
No

9. The "store character" instruction will place in the storage operand the contents of the _____ _____ byte from the specified register.

• • •

low order

10. Given the following STC instruction, show the resulting contents of the storage location.



• • •

17

11. If the address of the storage operand is not available on the particular installation, an_____ exception will be recognized.

• • •

addressing

12. Any instruction that changes the contents of main storage is subject to the storage protection feature. As a result, the_____(IC/STC) instruction can cause a protection exception.

• • •

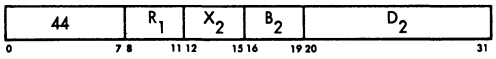
STC

EXECUTE INSTRUCTION

The "branch on condition, branch and link, branch on count, branch on index high, branch on index low or equal" instructions are the only actual "branch" instructions in the System/360. There is, however, another instruction called "execute" which does not change the instruction address in the PSW. However, it does cause one instruction in main storage to be executed out of sequence. That is, instead of branching from one routine to another, the "execute" instruction will cause one instruction in another routine to be executed without leaving the original routine.

Execute

EX $R_1, D_2(X_2, B_2)$ [RX]



This instruction operates in the following manner.

- Modify the instruction at the address specified in the second operand by the contents of the register specified in the first operand.
  a. OR bits 8-15 of the addressed instruction with bits 24-31 of the general register.
  b. The instruction in storage is not changed and the modification is effective only in the interpretation of the instruction.
- Execute the modified instruction addressed by the second operand.
- Return to and execute the next sequential instruction following the EX instruction.
- If the R1 field is zero, the instruction at the second operand location is executed without modification.

Condition Code:
The code may be set by the branch to instruction.
Program Interruptions:
Execute
Addressing
Specification

```
Name      Operation      Operand
1         8  10      14  16      20  25      30      35      40      45      50
                                                                              
          EX     5,MVCHAR     INSERT LENGTH CODE
*                             FROM REGISTER 5 IN THE
*                             MVC INSTRUCTION AT
*                             MVCHAR. EXECUTE THE
*                             MVC INSTRUCTION.
                                                                              
```

1. EX is the mnemonic for the "_____" instruction.

• • •

"execute"

2. Without branching, the "execute" instruction will cause another instruction to be executed. Given the following EX instruction, indicate the address of the instruction to be executed.

D0001 EX 0,4095

Address of instruction to be executed= _____.

• • •

4095

3. The instruction to be executed will be executed as it is if the R1 field of the EX instruction is_____.

• • •

zero

4. Assuming that the effective generated storage address from the "execute" instruction is location 8500, write the addresses of the instructions in the sequence in which they will be executed. (GR2 is the base register and contains the base address 6500.)

Location	Instruction	Actual Sequence
2048	LH 1,1000(0,2)	_____
2052	EX 0,2000(0,2)	_____
2056	STH 1,1002(0,2)	_____
8500	MVI 1025,X'00'	_____

• • •

2048; 2052; 8500; 2056

5. Was the instruction at location 8500 modified in any way prior to being executed?_____

• • •

No

6. Why not?_____

• • •

The R1 field of the "execute" instruction was zero.

7. Was the address of the instruction at location 8500 placed in the instruction address portion of the PSW?_____

• • •

No; The "execute" instruction is not a "branch" instruction. Instead, it causes an instruction to be executed that is not in the sequence presently being executed.

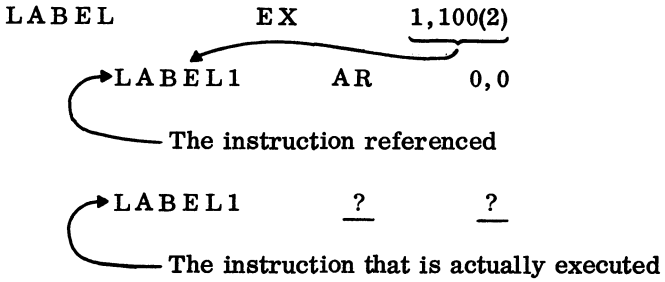
In the previous program example, the "execute" instruction at 2052 caused the instruction at 8500 to be executed. The normal sequence of instruction execution continued with the instruction at 2056.

8. If the R1 field of the "execute" instruction is other than zero, the low-order byte of the specified register will be ORed with the_____(1st/2nd) byte of the instruction to be executed.

• • •

2nd; The first byte of an instruction always contains the operation code.

9. Given the following, write the instruction that is actually executed.



Reg 1 contains 00 FF 00 FA (in hex)

• • •

LABEL1 AR 15,10; Only the second byte of the instruction | AR | R_1 | R_2 | was effectively modified.

10. The instruction that was actually executed causes the contents of register_____to be algebraically added to the contents of register_____.

• • •

10; 15

11. The ORing of the 2nd byte of the instruction with the low-order byte from the register is done in the ALU. As a result, the instruction in storage_____ _____(remains the same/is changed.)

• • •

remains the same

12. Given the following, write the instruction that will be executed in ALU. Remember that the bytes are ORed!

```
LABEL2      EX      1,100(2)
LABEL2      AR      2,3
```

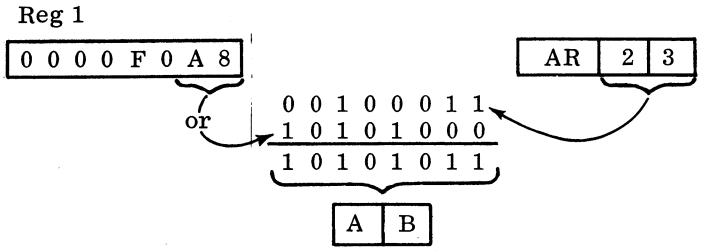
This instruction was actually executed.

Reg 1 contains 00 00 F0 A8

• • •

AR	A	B

Bits 24-31 of register 1 are ORed with bits 8-15 of the AR instruction as shown below.



13. As a result of the instruction shown in the previous example, the "execute" instruction would cause (via the "add" instruction) the contents of register_____ to be added to contents of register_____.

• • •

11; 10

14. Write the AR instruction that remains in storage as a result of the instruction in the preceding example.

AR		

• • •

AR	2	3

The instruction in storage is not changed.

There are three programming interrupts possible with an "execute" instruction.

- Specification exception
- Addressing exception
- Execute exception

15. A specification exception can occur on an "execute" instruction if the generated effective address of the instruction to be executed is_____(odd/even).

• • •

odd; Remember that all instructions must start on an even address.

16. An addressing exception can occur on an "execute" instruction if the generated effective address of the instruction to be executed_____(is/is not) available on the particular System/360 installation.

• • •

is not

17. An execute exception can occur on an "execute" instruction if the system is directed to another "_____" instruction.

• • •

execute

18.
```
LABEL       EX      1,X'EF1'
            AR      2,3
```

The above "execute" instruction will result in a(n) _____exception.

• • •

specification; The generated address is odd.

19.
```
LABEL       EX      1,X'FFA'
LABEL1      EX      1,0
```

The above "execute" instruction will result in a(n) _____exception.

• • •

execute

20.
```
            EX      0,0(1)
```

Reg 1

F F F F F F E

The above "execute" instruction will probably result in a(n) _____exception.

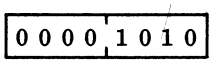
• • •

addressing

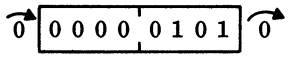
SHIFT INSTRUCTIONS, ALGEBRAIC

The "shift" instructions involve only the general registers and operate in the binary mode. Data in main storage cannot be shifted. The condition code is set after the operation.

What do we mean by shifting? Shifting basically is moving the contents of the register to the right or to the left. For instance, assuming we have a theoretical 8-bit register, shifting would take place as follows:



If this register were shifted one place to the right it would look like this:



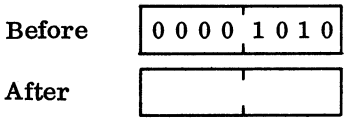
Notice that the low-order bit was shifted out. The resulting number (5) in the register is 1/2 the original number (10). Right shifting is similar to dividing by powers of 2.

1. A right shift of two places is similar to dividing by 4; a right shift of three places is similar to dividing by_____(6/8).

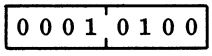
• • •

8

2. If the same theoretical 8-bit register shown below were shifted one place to the left, what would the resulting register look like?



• • •



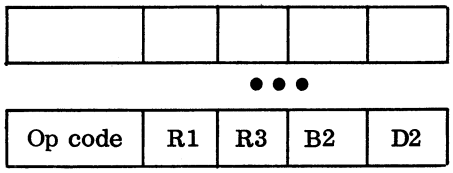
3. Notice that the result (20) of the preceding problem is twice that of the original number (10). Left shifting is similar to_____by the powers of two.

• • •

multiplying

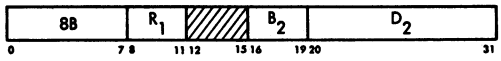
The System/360 can shift a register or a pair of registers either to the left or to the right. Furthermore, its "shift" instructions fall into two categories: algebraic and logical.

4. All of the "shift" instructions use the RS format. Label the fields of the RS format.



Shift Left Single

SLA ***R_1, $D_2(B_2)$*** ***[RS]***



Bits 1-31 of the register specified in the first operand are shifted to the left the number of positions specified by the low-order six bits of the second operand.

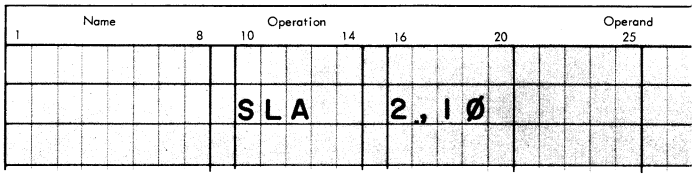
- Bit 0 is the sign bit and is not shifted.
- Zeros are supplied to fill the vacated low order positions.
- If a significant bit is shifted out of the register, an overflow occurs.

Condition code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 Overflow

Program Interruptions:

Fixed-point overflow



Shift Right Single

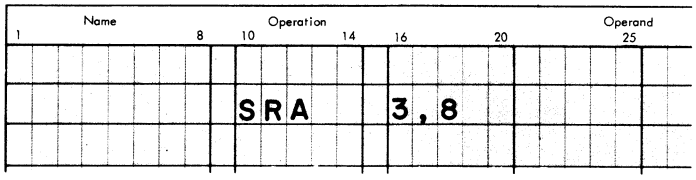
SRA $R_1, D_2(B_2)$ [RS]

Bits 1-31 of the register specified in the first operand are shifted to the right the number of positions specified by the low-order six bits of the second operand.

- Bit 0 is the sign bit and is not shifted.
- Bits equal to the sign bit are supplied to fill the vacated high-order positions.
- There is no overflow.

Condition code:

0 Result is zero
1 Result is less than zero
2 Result is greater than zero
3 --



Mnemonic	Hex Op Code	Data Flow
SLA	8B	Shift register to the left
SRA	8A	Shift register to the right

• • •

5. In the SLA instruction as in all "shift" instructions, the RS format is used but the_____ field is ignored. The register to be shifted by an SLA or SRA instruction is indicated by the _____ field.

• • •

R3; R1

6. The address generated by adding the base register contents and the displacement is used to_____ _____(address data/indicate number of positions to be shifted).

• • •

indicate number of positions to be shifted.

7. The number of places to shift the register is indicated by the_____low-order bits of the generated address.

• • •

8. The maximum number of positions that can be shifted is_____.

• • •

63; 111111 = 63

9. If the generated address is zero, the condition code will be set and the register_____ (will/will not) be shifted.

• • •

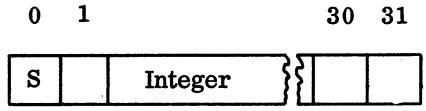
will not

10. The letter A in the mnemonics (SLA, SRA) indicates that the shift is_____ (algebraic/logical). In an algebraic shift, the sign bit _____(is/is not) shifted.

• • •

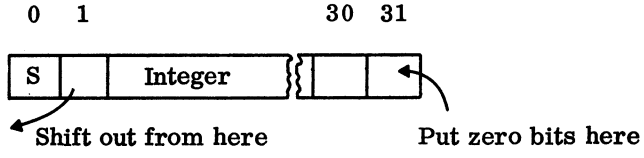
algebraic; is not

11. In the SLA instruction, the shifting is out of bit position_____(0/1).



• • •

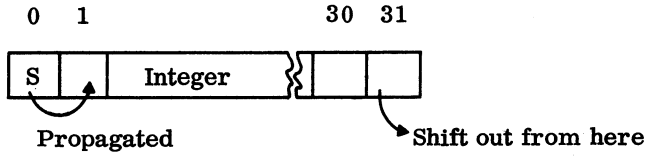
1; As shown below



12. In the SRA instruction, the sign bit is_____ (shifted/propagated) to the right.

• • •

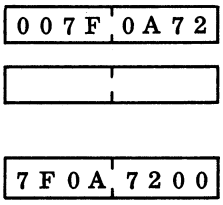
propagated; As shown below



13. Given the following SLA instruction, show (in hex) the contents of the shifted register.

SLA 2,8

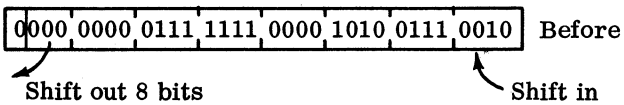
Reg 2



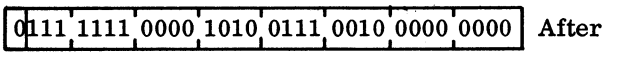
• • •

The generated address was 0008. As a result, register 2 was shifted eight places to the left. Let's take a look at the preceding example again. This time, we'll show the binary contents.

Reg 2



Reg 2



Notice that no significant bits were shifted out in the preceding example. If the register had been shifted 9 places, a significant bit would have been lost.

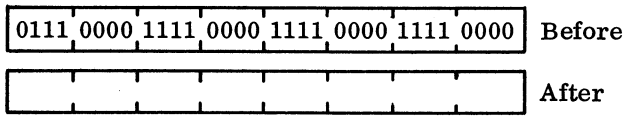
14. When a bit is shifted out (SLA only) that is different than the sign bit, a significant bit is lost. A _____ _____ _____ exception will result and a program interrupt may occur.

• • •

fixed point overflow; Notice that a program interrupt may occur. Remember that the fixed point overflow interrupt can be prevented by use of the program mask (bits 36-39 of the PSW).

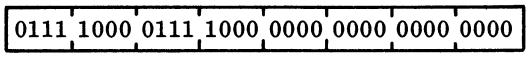
15. Given the following SLA instruction, indicate the contents of the shifted register and the condition code.

SLA 3,15




Condition code

• • •



Fixed point overflow

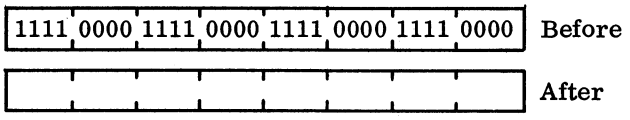
Notice that even though the fixed point overflow occurs with the 1st bit shifted, the entire shift of 15 places still occurs.

Let's move on to the "shift right algebraic" instruction.

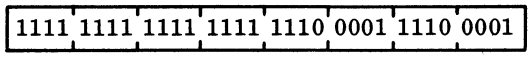
16. Given the following SRA instruction, show the contents of the shifted register.

SRA 3,15

Reg 3



• • •



Notice the propagation of the sign bit.

17. The condition code setting for the preceding problem would be _____.

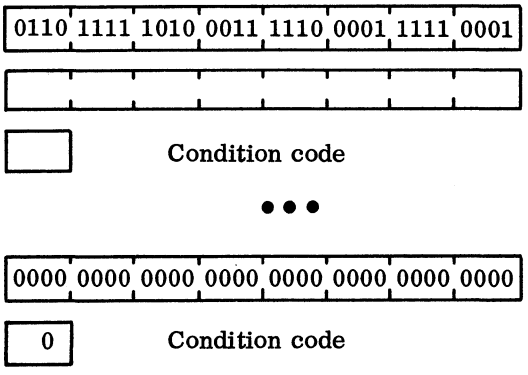
• • •

1; This condition code reflects a negative result. Notice that a fixed point overflow cannot occur on a right shift operation no matter what bits are shifted.

18. Given the following SRA instruction, show the contents of the shifted register and the condition code.

SRA 3,63

Reg 3

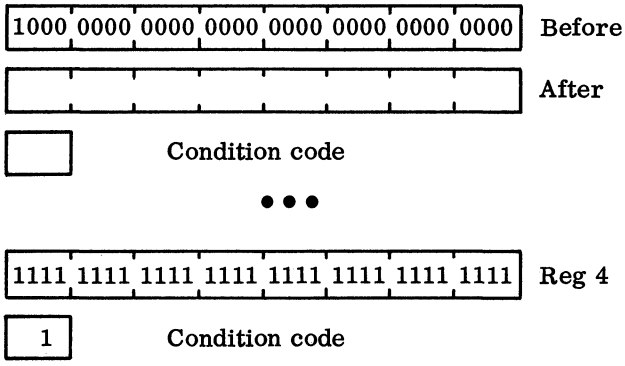


Notice that a right shift of 31 or greater of a positive number will zero out a register, because the sign bit of 0 is propagated to the right.

19. Given the following SRA instruction, show the contents of the shifted register and the resulting condition code.

SRA 4,63

Reg 4



Notice that a right shift of 31 or greater of a negative number will result in a -1, because the sign bit of 1 is propagated to the right.

Besides shifting a single register, the System/360 also has the ability to shift a doubleword that resides in an even-odd pair of registers (remember the doubleword product as a result of a multiply).

Shift Left Double

SLDA $R_1, D_2(B_2)$ [RS]

Bits 1-63 of an even-odd pair of general registers specified in the first operand are shifted to the left the number of positions specified by the low order six bits of the second operand.

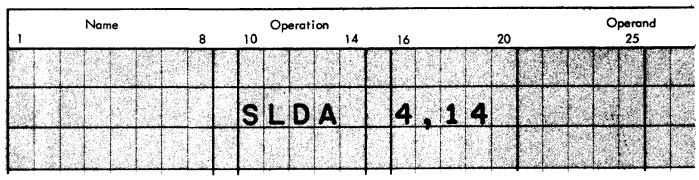
- Bit 0 of the even register is the sign bit and is not shifted.
- Zeros are supplied to fill the vacated low order positions.
- If a significant bit is shifted out of the register, an overflow occurs.
- The register specified in the first operand must be the even register of an even-odd pair.

Condition Code:

0	Result is zero
1	Result is less than zero
2	Result is greater than zero
3	Overflow

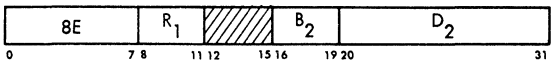
Program Interruptions:

Specification
Fixed-point overflow



Shift Right Double

SRDA $R_1, D_2(B_2)$ [RS]



Bits 1-63 of an even-odd pair of general registers specified in the first operand are shifted to the right the number of positions specified by the low order six bits of the second operand.

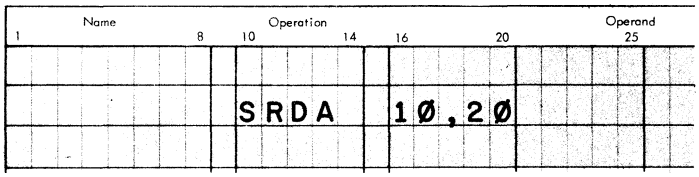
- Bit 0 of the even register is the sign bit and is not shifted.
- Bits equal to the sign bit are supplied to fill the vacated high order positions.
- There is no overflow.
- The register specified in the first operand must be the even register of an even-odd pair.

Condition Code:

0	Result is zero
1	Result is less than zero
2	Result is greater than zero
3	--

Program Interruptions:

Specification



Mnemonic	Hex Op Codes	Data Flow
SLDA	8F	Shift double reg to left
SRDA	8E	Shift double reg to right

• • •

20. The SLDA and SRDA instructions are also of the RS format. The SLDA and SRDA instructions are similar to the SLA and SRA instructions in that the _____ is ignored.

• • •

R3

21. The SLDA, SRDA, SLA and SRA are also similar in that the number of shifts is determined by _____ _____.

• • •

Only the low-order six bits of the generated address.

22. In both the SLDA and SRDA instructions, the R1 field must have the address of an _____ _____.

• • •

even-numbered register

23. SLDA 3,1

The above SLDA instruction would result in a _____ exception.

• • •

specification; Because the R1 field has an odd address.

24. SLDA 4,6

In the above SLDA instruction, registers _____ and _____ will be shifted together.

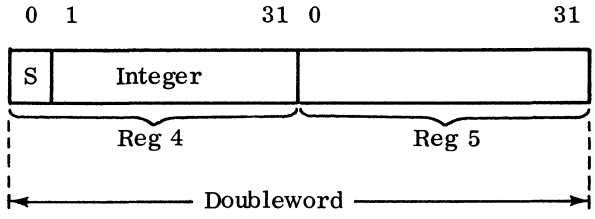
• • •

4; 5

25. In the preceding example the sign of the doubleword is in bit position _____ of register _____.

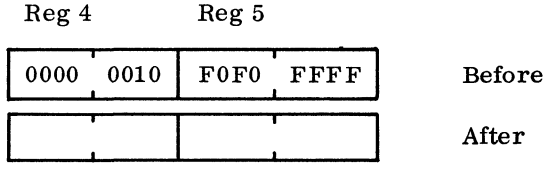
• • •

0; 4; as shown below

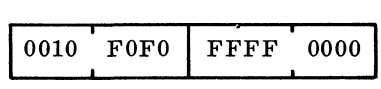


26. Given the following SLDA instruction, show (in hex) the contents of the shifted registers.

SLDA 4,16



• • •



A shift of 16 places was specified.

27. Given the following SRDA instruction, show (in hex) the contents of the shifted registers and the resulting condition code.

SRDA 4,16

Reg 4 Reg 5

0000	0010	F0F0	FFFF

Condition code

• • •

Reg 4 Reg 5

0000	0000	0010	F0F0

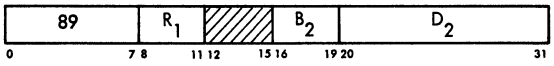
2

SHIFT INSTRUCTIONS - LOGICAL

You have finished the four "algebraic shift" instructions and are now ready to study the four "logical shift" instructions. The "logical shifts" differ from the "algebraic shifts" in that the entire register participates in the shift, the condition code is unchanged and a fixed point overflow cannot occur.

Shift Left Single

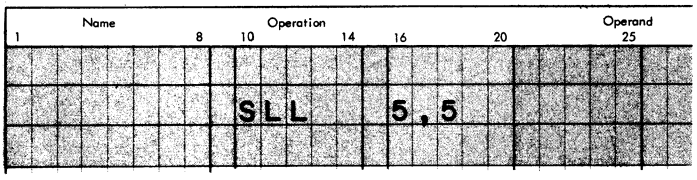
SLL $R_1, D_2(B_2)$ [RS]



All the bits of the general register specified in the first operand are shifted to the left the number of positions specified by the low-order six bits of the second operand.

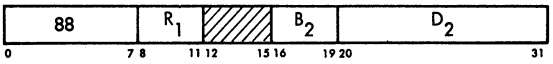
- Zeros are supplied to fill the vacated low-order positions.
- Significant bits which are shifted out are lost.

Condition Code:
The code remains unchanged.
Program Interruptions:
None



Shift Right Single

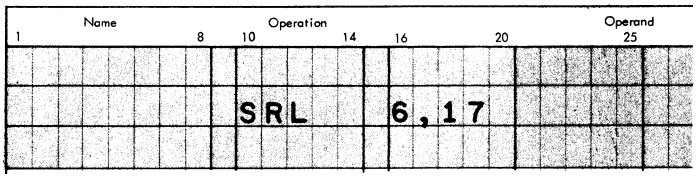
SRL $R_1, D_2(B_2)$ [RS]



All the bits of the general register specified in the first operand are shifted to the right the number of positions specified by the low-order six bits of the second operand.

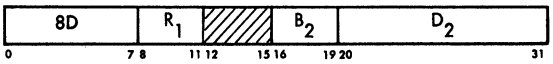
- Zeros are supplied to fill the vacated high-order positions.

Condition Code:
The code remains unchanged.
Program Interruptions:
None



Shift Left Double

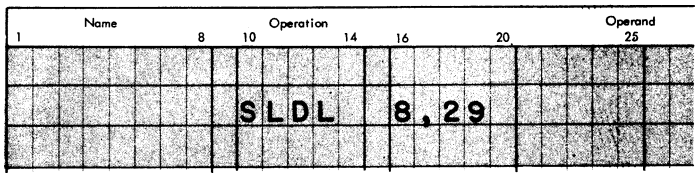
SLDL $R_1, D_2(B_2)$ [RS]



All the bits of an even-odd pair of general registers specified in the first operand are shifted to the left the number of positions specified by the low-order six bits of the second operand.

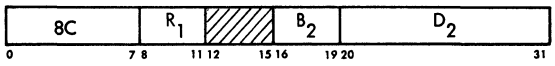
- Zeros are supplied to fill the vacated low-order positions.
- Significant bits which are shifted out are lost.
- The register specified in the first operand must be the even register of an even-odd pair.

Condition Code:
The code remains unchanged.
Program Interruptions:
Specification



Shift Right Double

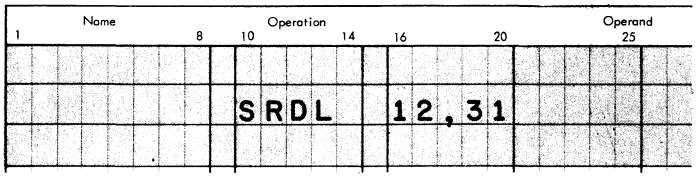
SRDL $R_1, D_2(B_2)$ [RS]



All the bits of an even-odd pair of general registers specified in the first operand are shifted to the right the number of positions specified in the low-order six bits of the second operand.

- Zeros are supplied to fill the created high-order positions.
- The register specified in the first operand must be the even register of an even-odd pair.

Condition Code:
The code remains unchanged.
Program Interruptions:
Specification



• • •

Mnemonic	Hex Op Code	Data Flow
SLL	89	Shift register left
SRL	88	Shift register right
SLDL	8D	Shift double reg left
SRDL	8C	Shift double reg right

1. The "shift logical" instructions are of the RS format and just like the "algebraic shifts", the "logical shift" instructions ignore the_____field.

• • •

R3

2. The number of logical shifts taken is determined by the_____.

• • •

Low-order six bits of the generated address.

3. Unlike the "algebraic shifts", the "logical shifts" _____(do/do not) change the condition code.

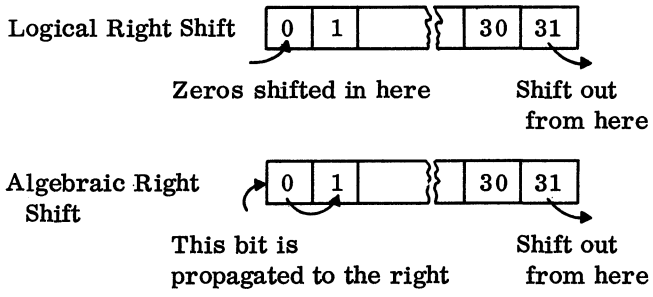
• • •

do not

4. In a "logical right shift", the sign bit is not propagated. Instead, it is shifted and zeroes are inserted in the bit position_____.

• • •

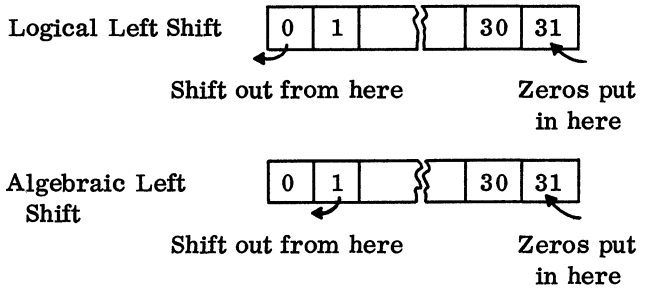
0 as shown below



5. In a "logical left shift" such as SLL, shifting is done out of bit position_____and zeros are inserted into bit position_____.

• • •

0; 31 as shown below



6. | 7FFF | FFFF | Reg Before

| 7FFF | FF00 | Reg After

Which of the following mnemonics_____(SLA, SRA, SLL, SRL) would have produced the results indicated above?

• • •

SLA; In this example, the condition code would have been set to 3 and a fixed point overflow occurs. If the SLL instruction had been used, shifting would have been done out of position 0 and the sign bit would have changed.

0 31

7. | A000 | F000 | Reg Before

| 0F00 | 0000 | Reg After

Which of the following mnemonics_____(SLA, SRA, SLL, SRL) would have produced the results indicated above?

• • •

SLL; In this example, bit position 0 is changed.

STUDENT'S COMMENT FORM

System/360 Assember Language Coding – Appendix
R29-0233-3

Your comments, as well as answers to the following questions, will help us design and administer programmed or self-study courses in a way that better suits your needs. If your answer to a question is "No", or needs further explanation, please use the space provided below. Comments and suggestions become the property of IBM.

- What is your occupation?
- Why did you take this course?

	Yes	No
• Did this course, in general, meet your needs?	☐	☐
• Did an IBM employee serve as your advisor?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Well illustrated?	☐	☐
• Did you feel that any particular topic should be added or emphasized?	☐	☐
• Did you feel that any particular topic should not have been included?	☐	☐

- We would appreciate your other comments; please give specific page and line references where appropriate.

Staple

Fold

Fold

FIRST CLASS
PERMIT NO. 10
ENDICOTT, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY....

IBM Corporation
1701 North St.
Endicott, N.Y. 13760

Attention: DP Education Development, Dept. 617

Cut Along Line

Fold

Fold

Additional Comments:

R29-0233-3

Printed in U.S.A.

R29-0233-3

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)