System/360 Assembler Language Coding Standard and Decimal Instructions Text

Programmed Instruction

System/360
Assembler Language Coding
Standard and Decimal Instructions
Text

Programmed Instruction

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.

Major Revision (October 1969)

This publication is a revision of Form R29-0232-5 incorporating changes made on pages 29, 30, 63, 66, 67, and 78. The original publication is obsoleted.

This material was produced for educational purposes only. The utmost care has been taken to ensure the accuracy of this publication. No responsibility is assumed for any inaccuracies that may occur. It should be understood, however, that changes may occur after this date (10/69) that may cause all or part of this publication to become obsolete.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM Branch Office serving your locality. Address comments concerning this publication to: DPD Education Development, IBM Education Center, 6 Roosevelt Avenue, Endicott, New York 13760.

© Copyright International Business Machines Corporation 1966

CONTENTS

SECTION I - INTRODUCTION TO SYMBOLIC CODING AND ASSEMBLY
Review and Terminology 1
Symbolic Coding and Assembly 4
Learning Objectives 4
Self-Evaluation Questions 4
Self-Study Text
Symbolic Addressing 6
Mnemonics and Operands 7
The Assembly Process 9

SECTION II - INTRODUCTION TO ASSEMBLER LANGUAGE CODING
Assembler Language Coding 12
Learning Objectives 12
Self-Study Text . . . . .
"Statements" on the Coding Sheet 13
Symbols 17
Base Register and Displacement Assignment 19
The Start and End Statements 22
Defining Storage Areas 23
Defining Data Constants 27
Self-Evaluation Questions 33

SECTION III - CODING SAMPLE PROGRAMS
Coding Sample Programs 37
Learning Objectives 37
Self-Study Text
Introduction to Ajax Sample Program #1 38
Examples from the Standard Instruction Set
Move Characters (MVC) 43
Move Immediate (MVI) 43
Skip Option on Move Instructions 44
End of Skip Option 46
Branch and Link Registers (BALR) 48
Branch and Link (BAL) 48
Skip Option on Branching Operations 49
End of Skip Option 51
Pack (PACK) 52
Convert to Binary (CVB) 53
Skip Option on Packing and Converting to Binary . . . 54
End of Skip Option 60
Store (ST) 60
Store Halfword (STH) 60
Skip Option on Store Instructions 61
End of Skip Option 62
Multiply (MR,M) 62
Multiply Halfword (MH) 62
Skip Option on Multiply Instructions 64
End of Skip Option 66
Divide (DR,D) 66
Skip Option on Divide Instructions 67
End of Skip Option 70
Add (AR,A) 70
Add Halfword (AH) 71

Skip Option on Add Instructions 71
End of Skip Option 74
Load (LR, L) 75
Load Halfword (LH) 75
Skip Option on Load Instructions 76
End of Skip Option 77
Convert to Decimal (CVD) 79
Skip Option on The Convert to Decimal Instruction . . 79
End of Skip Option 80
Subtract (SR, S) 80
Subtract Halfword 81
Skip Option on Subtract Instruction-Algebraic 82
End of Skip Option 83
Unpack (UNPK) 84
Skip Option on the Unpack Instruction 85
End of Skip Option 86
Move Zones (MVZ) 86
Branch on Condition (BCR, BC) 88
Skip Option on the Branch on Condition Instruction . . . 89
End of Skip Option 90
The Last Entry 91
Summary of Coding 91
Decimal Arithmetic on the System/360 91
Introduction to Ajax Sample Program #2 92
Examples from the Decimal Instruction Set
Zero and Add (ZAP) 94
Skip Option on Zero and Add 96
End of Skip Option 97
Multiply Decimal (MP) 97
Skip Option on Multiply Decimal 98
End of Skip Option 100
Divide Decimal (DP) 100
Skip Option on Divide Decimal 102
End of Skip Option 104
Add Decimal (AP) 104
Skip Option on Add Decimal 105
End of Skip Option 108
Move Numerics (MVN) 108
Skip Option on Move Numerics 109
End of Skip Option 111
Subtract Decimal (SP) 112
Skip Option on Subtract Decimal 113
End of Skip Option 114
Edit (ED) 115
The Source Field 116
The Pattern Field 116
The Fill Character 117
The Digit Select Character 118
The S Trigger 120
The Significance Start Character 121
The Field Separator Character 121
Examples 122
Setting Up Patterns in Storage 122
Introduction to Sample Program #3 128

Examples of Branching and Logic Instructions 128
Or Instructions (OR, O, OI, OC) 129
And Instructions (NR, N, NI, NC 130
Skip Option on "And, Or" Operations 132
End of Skip Option 135
Test Under Mask (TM) 135
Skip Option on Test Under Mask 137
End of Skip Option 139
Turning a Switch On and Off 139
Compare Decimal (CP) 145
Skip Option on Compare Decimal 146
End of Skip Option 148
Move with Offset (MVO) 148
Skip Option on Move with Offset 149
End of Skip Option 151
Exclusive OR (XR, X, XI,XC) 159
Skip Option on Exclusive OR 160
End of Skip Option 162
Compare (CR, C) 165
Skip Option on Compare 166
Compare Halfword (CH) 166
End of Skip Option 167
How to Prepare for the Test 169

SECTION I

INTRODUCTION TO SYMBOLIC CODING AND ASSEMBLY

Coding involves:

- Arranging for input and output of data.
- Establishing "work areas" in storage.
- Creating constants (e.g., values used in calculations, symbols used to set switches and punctuate output).
- Choosing and writing the instructions that move data, perform appropriate tests and calculations, handle exceptional conditions, and arrange data in a format specified for output.

Assembler language allows this to be done with symbolic notation, as you know.

The first section of this book treats symbolic coding in detail. You probably already know quite a bit about it. Read the following brief review, take the self-evaluation quiz, and read the indicated pages <u>only</u> if you have trouble answering the corresponding questions.

REVIEW AND TERMINOLOGY

Programming in a symbolic language offers a number of important advantages over programming in the actual language of the computer:

- Mnemonic operation codes are provided. For instance, the actual operation code for the instruction Store in hexadecimal is 50; in the assembly language we can write the mnemonic operation code ST. Most programmers never learn the actual codes.
- Addresses of data and instructions can be written in symbolic form, and in practice almost all addresses are so written. The programmer is thereby relieved of severe problems in the effective allocation of storage, and the resulting program is far easier to modify. Furthermore, the use of symbolic addresses reduces the clerical aspects of programming and eliminates many programming errors. If the symbols are chosen to be meaningful, the program is also much easier to read and understand than if written with numerical addresses.
- Constants may be introduced into the program structure, and space reserved for results, by the use of suitable assembler instructions. These are written in somewhat the same form as machine instructions but are treated quite differently by the assembly program.
- Many other assembler instructions direct the assembler in various other matters of concern. Among the most important of these are the techniques for letting the assembler assign base registers and compute displacements.

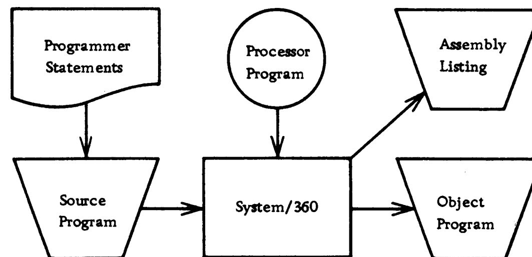
The sum effect of these advantages is so great that it is virtually out of the question to program in actual machine language, that is, to write actual operation codes and numerical addresses, and, in the case of the System/360, to write actual base register numbers and displacements.

An assembly language program is not directly executable by the computer. The mnemonic operation codes and symbolic addresses must be translated into the form the machine expects of instructions. This is the function of the processor program, also called the assembly program or simply the assembler.

```
                                      STATEMENT
Name      Operation  Operand                                   Comments
1       8 10     14 16   20   25   30   35   40   45   50   55
         TITLE  'ILLUSTRATIVE PROGRAM'
         START  256
BEGIN    BALR   11,Ø
         USING  *,11
         L      2,DATA             LOAD REGISTER 2
         A      2,TEN              ADD 1Ø
* THE FOLLOWING SHIFT HAS THE EFFECT OF MULTIPLYING BY 2
         SLA    2,1
         S      2,DATA+4           NOTE RELATIVE ADDRESSING
         ST     2,RESULT
         L      6,BIN1
         A      6,BIN2
         CVD    6,DEC              CONVERT TO DECIMAL
         EOJ                       END OF JOB
DATA     DC     F'25'
         DC     F'15'
TEN      DC     F'1Ø'
RESULT   DS     F
BIN1     DC     F'12'
BIN2     DC     F'78'
DEC      DS     D
         END    BEGIN
```

A program to illustrate assembly language concepts.

The assembly process begins with a source program written by the programmer. Ordinarily, a special coding form is used such as that above. Cards are punched from this form, one card for each line of coding, making up the source program deck. This source program deck becomes the primary input to the assembly process, as shown below.



Schematic presentation of the assembly process.

The assembly is done, in our case, by the System/360 under control of a processor program. The processor program is supplied by IBM; it consists of many thousands of machine instructions.

There are two outputs from the processor run. The first is an object program consisting of actual machine instructions corresponding to the source program statements written by the programmer. In many cases the object program is punched into cards; in other cases it is left on magnetic tape or magnetic disks. The second output is a program listing or assembly listing. This important document shows the original source program statements side by side with the object program instructions created from them. Many programmers work from the assembly program listing as soon as it is available, hardly ever referring to their coding sheets again. An example appears on the next page.

```
                                                           A
    C        B                              ┌──────────────────────────────────────┐
  ┌───┐  ┌───────┐                   1             PRINT NOGEN
                                     2 *           TITLE ILLUSTRATIVE PROGRAM
000100                               3             START 256
000100 05B0                          4 BEGIN       BALR  11,0
000102                               5             USING *,11
000102 5820 B022             00124   6             L     2,DATA          LOAD REGISTER 2
000106 5A20 B02A             0012C   7             A     2,TEN           ADD 10
                                     8 * THE FOLLOWING SHIFT HAS THE EFFECT OF MULTIPLYING BY 2
00010A 8B20 0001             00001   9             SLA   2,1
00010E 5B20 B026             00128  10             S     2,DATA+4        NOTE RELATIVE ADDRESSING
000112 5020 B02E             00130  11             ST    2,RESULT
000116 5860 B032             00134  12             L     6,BIN1
00011A 5A60 B036             00138  13             A     6,BIN2
00011E 4E60 B03E             00140  14             CVD   6,DEC           CONVERT TO DECIMAL
                                    15             EOJ                   END OF JOB
000124 00000019                     18 DATA        DC    F'25'
000128 0000000F                     19             DC    F'15'
00012C 0000000A                     20 TEN         DC    F'10'
000130                              21 RESULT      DS    F
000134 0000000C                     22 BIN1        DC    F'12'
000138 0000004E                     23 BIN2        DC    F'78'
000140                              24 DEC         DS    D
000100                              25             END   BEGIN
```

Assembly listing produced by the assembly of the program.

Proceeding from right to left in the example:

- The items listed under A should be exactly the same as the handwritten entries on the coding sheet. This provides a good check on the accuracy of the keypunching.
- The items under B are a representation, in hex, of the corresponding instructions and constants.
- C shows the addresses (in hex) of the instructions, constants, and areas of storage specified by the programmer.

SUMMARY:

- Programs consist of many instructions. Each instruction specifies some operation and the location of the data to be operated upon.
- The computer will accept only instructions written in the language the engineers designed for it. This machine language is the binary coding used to represent the instructions and data.
- Programmers usually write their programs in a near-English language called symbolic language.
- Symbolic programs are translated into machine language by a machine language program called a processor, assembly program, or assembler.
- A symbolic language and its associated processor is referred to as a programming system.
- The symbolic program that is input to the processor is called the source program.
- The object program is the name for the output data from the processor. The object program is the machine language equivalent for the source program.
- Along with the object program, the processor prints out a program listing which shows:
  a. Your coding, as keypunched into source cards.
  b. The hex equivalent of each instruction and constant that you have specified.
  c. The address, in hex, of each instruction, constant, and reserved storage area.
- Computers can't execute source programs. They can only execute machine language programs such as a processor program or an object program.

SYMBOLIC CODING AND ASSEMBLY

Learning Objectives

When you complete the following test, you will have demonstrated that you can:

- Identify the inputs to and outputs from the assembly process.
- Distinguish between labels and operation codes.
- State the rules for symbolic addressing and identify correct and incorrect examples.
- Describe how the Assembler assigns, and keeps track of, addresses for labels.

SELF-EVALUATION QUESTIONS — Reference Pages in Text

1. Which of the following is a program written in a symbolic language? — 6
   a. source program
   b. object program
   c. assembler program
   d. processor program

2. Which of the following does not mean a "symbolic address"? — 7
   a. label
   b. operand
   c. name
   d. symbol

3. The input data during assembly time is the — 9
   a. object deck
   b. coding sheet
   c. assembler program
   d. source deck

4. The symbolic representation of a machine language op code is called the — 7
   a. operand
   b. name
   c. mnemonic
   d. flag

5. Which of the following is generated by phase 1 of a two phase assembly? — 11
   a. symbol table
   b. program listing
   c. object deck
   d. location counter

6. Which of the following is used to assign addresses to the source statements? — 10
   a. symbol table
   b. location counter
   c. PSW
   d. none of the above

Reference Pages in Text

The following program is to be used for question 7.

NAME	OPERATION	OPERAND
BEGIN	MVC	FIELDA,FIELDA
	A	7,FIELDA
	ST	7,FIELDB
	BC	15,START
FIELDA	DC	F'4'
FIELDB	DC	F'8'

7. Which of the following symbolic addresses was not defined in the preceding program? — 7
   a. FIELDA
   b. FIELDB
   c. BEGIN
   d. START

8. The machine language program that results from translating a symbolic language program is called the — 11
   a. processor program
   b. source program
   c. assembler program
   d. object program

9. A processor program — 9
   a. is a machine language program.
   b. translates symbolic language programs into machine language.
   c. resides in main storage during assembly time.
   d. All of the above.

10. Which of the following is true? — 9
   a. During assembly time symbolic statements are converted to machine language and executed.
   b. An object program can be executed once it is loaded into main storage.
   c. The symbol table is a list of instruction mnemonics.
   d. Two phase assemblers require that a symbolic address be defined in a prior statement before it can appear as an operand.

ANSWERS

1.	a	6.	b
2.	b	7.	d
3.	d	8.	d
4.	c	9.	d
5.	a	10.	b

Go to Section II "Introduction to Assembler Language Coding"

SYMBOLIC ADDRESSING

The first item we encounter is the coding sheet. As part of your student materials you should have a pad of coding sheets (Form X28-6509). Tear off a sheet and refer to it when necessary during the next few frames. The coding sheet comes into play after the programmer has completed the preliminary steps of programming such as defining his problem and flow charting his solution. He is then ready to write the instructions of his program.

1. Which of the following is the function of the coding sheet? Choose the best answer.
   a. It is used as a direct means of input to the processor.
   b. It is used as an aid in manually keying the source program into the computer.
   c. It is used for keypunching the source program, which can then be read by the processor from a card reader.

• • •

c.

You know that the coding sheet is used by the programmer to code his symbolic program, and by the keypunch operator to punch the source deck. Of course, the programmer may have to become a part-time keypunch operator. Now let's see how you would write a symbolic statement on the coding sheet.

Each line of the coding sheet represents one symbolic statement. A symbolic statement is used to tell the processor to assemble a machine language instruction, a data constant, or to do something during assembly time. Approximately 24 statements can be written on each coding sheet. Normally programs contain hundreds of instructions. As a result, you would probably use a number of coding sheets to write a complete program. The first thing we will be concerned with is writing a statement to tell the processor to assemble a machine language instruction. All instructions have an address in main storage which they will occupy when the object program is being executed. Instructions also have an op code and usually the address of one or more data operands. For instance, take the case of an RR format instruction which adds the contents of one general register to those of another. This instruction would begin at some address in main storage, have an op code of hex 1A, and contain the addresses of two general registers. The address of an instruction, its op code, and the data addresses correspond respectively to the following fields on the coding sheet: Name, Operation, Operand. The entries on the coding sheet will be made symbolically, rather than in machine language. Let's discuss the concept of symbolic addresses first.

Go to the next frame.

Notice that the first eight columns of the coding sheet are called the name field. This field is used to give symbolic names to the locations referred to by your program. For instance, if your program contains a routine to handle fixed point overflows, it would be simpler if you did not have to remember the machine address of the routine. After assigning a name to the <u>first</u> instruction in this routine, you can then write a symbolic branch instruction referring to that name. The processor program, in converting your program to machine language, will take care of remembering the machine address of your symbolic name and will put it in your object program whenever you refer to it. Symbolic names are also referred to as either symbols or labels.

2. Which of the following is most correct?
   a. Symbols or labels should be assigned to any instruction to which you will branch or any data field on which you will operate, in your program.
   b. Symbols or labels should be given to every instruction or data field in your program.

• • •

a.

You understand that the reason for assigning labels is so you don't have to remember or compute the machine location of the instructions or data fields you want to refer to. As a result, you don't have to give labels to every instruction. When branching to some routine, only the first instruction of that routine needs to have a label. Another concept concerning labels is that the symbolic name should be meaningful. You can use any name you wish when assigning a label to an instruction or a data field. If the name isn't meaningful however, it will be difficult for you or anyone else reading your program to figure out what you were doing.

3. One of the fields of a data record contains the hours worked by an employee. Which of the following would be a more appropriate label for this field?
   a. FIELDA
   b. HRSWKD

• • •

b.

You <u>could</u> use the name FIELDA to refer to a field containing the hours worked. It certainly saves you the trouble of remembering or computing its address. However, if someone else had to examine your program, he would have difficulty in knowing what FIELDA refers to.

HRSWKD would be a better name for a field containing the number of hours worked. You may be wondering why we abbreviated as we did, rather than writing out both words for the symbolic name. The reason is that most processors put a limit on the length of a symbol, and do not allow the use of blanks or special characters. The coding sheet you have been looking at was designed for an assembler provided with an operating system. Up to eight characters can be used for a label.

4. Referring to the coding sheet you have been looking at, which of the following labels could be used to refer to a field containing an employee's serial number?
   a. MAN NO
   b. MANNO
   c. MANNUMBER
   d. EMERS*

•••

b.

MANNO would be a meaningful label for an employee's serial number. Unlike the others, it follows all the rules: It contains eight or less characters, with no blanks or special characters. Note that although the name field is eight columns long, the label can be shorter.

Note: $, #, and @ are not special characters in System/360.

Remember that the reason for assigning labels is so we can refer to them later in our program. The processor cannot remember or compute the machine address for a symbol if we haven't assigned the symbol to some location in our program.

5. Suppose the following instructions represent some program. If the last instruction is intended to cause an unconditional branch back to the add instruction, what must be done?

NAME	OPERATION	OPERAND
	ADD	FIELDA, FIELDB
	SUB	FIELDA, FIELDC
	MULT	FIELDA, FUDFAC
	BR	BEGIN

   a. Nothing since the processor will figure out that we want to branch back to the beginning.
   b. BEGIN must be written in the name field of the Add instruction.

•••

b.

If you use a symbol as the address of some instruction or data field, that symbol must appear in the name field on your coding sheet. In the example in Frame 5, the symbol BEGIN must be in the name column of the ADD instruction. Otherwise the processor program won't know what address to put in the Branch instruction. We will discuss how the processor keeps track of symbols and their associated machine addresses later in this book.

You now know the following concepts concerning the use of symbols in programs:

- Symbolic names are usually given to instructions or data fields referred to in programs.
- A symbol cannot be used in the operand field unless it also appears in the name field of one of your symbolic statements. That is, you can't refer to a symbol in your program unless you have used that symbol as the name of one of your instructions or data fields.
- Symbols are usually restricted in length and may not contain blanks or special characters.
- Within the preceding limitations, any symbol may be used. However, symbols should be as self-explanatory as possible.
- The following terms are used interchangeably: symbols, labels, names.

MNEMONICS AND OPERANDS

Each line on a coding sheet is one symbolic statement. A symbolic statement can be an instruction, a data field, or possibly just some information to the processor for use during the assembly process. By assembly process, we mean the time during which the processor translates your symbolic program into machine language. The operation field on your coding sheet tells the processor whether the symbolic statement is an instruction or something else. Although the name field of a symbolic statement may be left blank, the operation field must contain a "symbol" that is recognizable by the processor program. If the symbolic statement is an instruction, the "symbol" represents one of the computer's operation codes, and is called a mnemonic. The word "mnemonic" derives from a Greek word meaning "to remember". That is, it is easier to remember a symbolic op code than it is to remember its machine language equivalent.

1. If L is the mnemonic for the System/360 Load instruction while 58 is its machine language op code in hexadecimal notation, what would have to be put in the operation field of the following statement? The assumption is that you want the processor program to assemble an instruction that will load a fullword from MANNO into general register 4.

NAME	OPERATION	OPERAND
BEGIN		4, MANNO

   a. 58
   b. nothing
   c. L

●●●

c.

If nothing is put in the operation field of the symbolic statement, the processor won't know if you want an instruction assembled or a data field. If you want an instruction, you must tell the processor this by placing the mnemonic of that instruction in the operation field.

By writing an L you tell the processor to assemble a Load instruction. You place the mnemonic for the instruction in the operation field of the coding sheet. Each System/360 instruction has its own unique mnemonic. You could not use the mnemonic L to load a halfword into a general register. You would have to use the mnemonic LH which represents the Load Halfword instruction. The letter H in a System/360 mnemonic is used to distinguish the instructions that process halfwords from those that process fullwords.

2. If A is the mnemonic for the instruction that adds a fullword to a general register, which of the following will add a halfword to a general register?
   a. AH
   b. A
   c. H

●●●

a.

The mnemonic AH is the unique "symbol" for the Add Halfword instruction, just as the mnemonic A is the unique "symbol" for the Add instruction. You should now realize that just as each instruction has a particular machine language op code, it also has a particular meaningful mnemonic.

In the preceding frames, we discussed the use of mnemonics to tell the processor to assemble an instruction. By use of "symbols" other than machine instruction mnemonics, you can direct the processor to do other things such as to assemble some data constant for use in your program. However, the "symbol" would have to be something recognizable by the processor program. Since the processor "assembles" machine language object programs from symbolic statements, we will refer to it from here on as the assembler program. For now, take a look at your System/360 Reference card (form # X20-1703). Note that the card begins with a list of the instructions making up the System/360 standard instruction set. Next to the name of each instruction is its mnemonic.

The operand field on the coding sheet is used to write the remainder of the instruction. That is, the op code of an instruction is represented in the operation field by a mnemonic, while the location of the data to be operated upon is put in the operand field.

3. If you wanted to write an instruction to add the contents of FIELDA to the contents of general register 4, the symbol "FIELDA" would be written in the
   a. name field
   b. operation field
   c. operand field

●●●

c.

The operand field on the coding sheet is used to write the addresses of the data being operated upon. In our case, FIELDA is the symbolic address of some data we wanted to add into register 4. So you were right if you picked answer c. We can't go much further into discussing the operand field without bringing in the particulars of some specific symbolic language. So we'll hold off for now until you begin studying the assembler language later in this book.

At this point, you know that the programmer writes his symbolic statements on a coding sheet. Each line on the coding sheet represents one statement. If the statement is a machine instruction, you would put the mnemonic for that instruction in the operation field of the coding sheet. The addresses of the data to be operated on would be put in the operand field. These addresses could be either symbolic or actual. In the following example, MVC is the mnemonic for the Move Characters instruction, which is the SS format (Storage to Storage). HRSWKD is an example of a symbolic address, and decimal 2048 is an example of an actual address. Note that the term "actual" does not necessarily mean machine language, since the System/ 360 doesn't operate with decimal addresses.

NAME	OPERATION	OPERAND
LABELX	MVC	2048(6),HRSWKD

If you wanted to branch to this instruction from someplace else in your program, you would need an entry in the name field such as LABELX. This symbol could then appear in the operand field of a branch instruction.

4. Each of the statements on the coding sheet is punched into an IBM card. The deck of cards that results is known as the source program or source deck. Which of the following is the function of the source deck?
   a. serves as input data for the assembler program.
   b. is loaded into the computer to be executed.
   c. is translated into machine language by the assembler one card at a time and executed.

•••

a.

The sole function of the source deck is to serve as input data for the assembler program. None of the instructions in the source program are executed during the assembly (translation) process. The output data of the assembler will be a machine language program called the object deck. The object deck is your program converted into machine language. It can be loaded, either now or later, into the computer for execution. There is no need to reassemble your program each time it is executed. The object deck can be used over and over again until you make changes in the program. For a discussion of the assembly process proceed to the next topic.

THE ASSEMBLY PROCESS

This series of frames will help you acquire an understanding of the assembly process. In other words, you will gain an insight as to how source programs are converted to object programs.

To obtain an object (machine language) program from your source (symbolic) program, a processor (assembler) program must first be loaded into the computer's main storage. As the assembler is being executed, it will read in the cards of the source deck and convert them to the machine language program that will be the object deck. There are actually two outputs from the assembly process. One is the object program, while the other is a program listing.

As was mentioned earlier, the computer, while executing the assembler program, is acting as a super-clerk. One of the clerical tasks of the assembler is to assign machine addresses to symbolic names, and to remember these addresses and place them in the object program whenever the symbol is used in the operand of the source statement.

For instance, when the assembler encounters the following source statement, it must assign a machine address to the symbol BEGIN.

NAME	OPERATION	OPERAND
BEGIN	MVC	88(7),2048

The assembler must remember the address of BEGIN so that it can insert that address when it encounters the following branch instruction.

NAME	OPERATION	OPERAND
	BC	15,BEGIN

To be able to assign a machine address to a symbol, assemblers contain a program counter. This counter is called the Location Counter, and keeps track of the addresses in the source program, as it is being assembled. The Location Counter is incremented as each symbolic statement is processed. The length, in bytes, of main storage area required by each statement determines how much the Location Counter is incremented. For instance, assume that the Location Counter is set to decimal 1000 when the following symbolic statement is read by the assembler.

NAME	OPERATION	OPERAND
BEGIN	MVC	88(7),2048

MVC is the mnemonic for the Move Characters instruction, which uses the SS format.

1. When the assembler encounters the preceding statement it will
   a. assign the address of decimal 1000 to the symbol Begin, and step the Location Counter to decimal 1006.
   b. assign the address of decimal 1000 to the symbol Begin, and step the Location Counter to decimal 1001.
   c. it will place the MVC instruction in location 1000_{10}, and execute the instruction

●●●

a.

Whenever the assembler finds an entry in the name field, it assigns the setting of Location Counter to that name. It then increments the counter by the number of bytes required by the statement. The MVC instruction in our example is six bytes long, and the Location Counter was stepped from 1000 to 1006.

2. Which of the following statements is correct?
   a. the Location Counter is a register in the System/360 used to keep track of the instruction being executed.
   b. the Location Counter is a data field in the assembler used to keep track of the storage locations assigned to the source statements.

●●●

b.

The instruction address in the PSW (bit 40-63) keeps track of the instruction being executed.

The Location Counter is just a data area within the assembler, and is used as a counter. The assembler will give it some initial setting and step it as required during the assembly process. It's main function is to be able to assign an address to symbols as they are encountered in the source program. The object program, when loaded into the computer later on, might actually reside at locations different than those assigned at assembly time. This is known as program relocation, and will be discussed separately later in this book.

For now, you know that the assembler uses its Location Counter to assign addresses to symbols. However, the assembler needs to remember what address it assigned to a symbol. It can't use the Location Counter to remember since the counter is being incremented by each symbolic statement. So what is necessary is another data area within the assembler program. This area is referred to as the Symbol Table. When a symbol is encountered in the name field of a symbolic statement, that symbol as well as the Location Counter setting is placed in the Symbol Table. The area of storage used for the Symbol Table is limited. It is for this reason that assemblers put a limit on the length of symbols and how many symbols may be used in a program.

When the assembler reads the following symbolic statement, it must replace the symbol BEGIN with a machine address.

NAME	OPERATION	OPERAND
	BC	15,BEGIN

3. The assembler obtains the machine address for BEGIN from the:
   a. Symbol Table
   b. Location Counter

●●●

a.

Whenever the assembler finds a symbol in the operand field, it goes to the Symbol Table. When it locates the symbol in the Symbol Table it gets its machine address and places it in the assembled instruction. Of course, the symbol BEGIN must have appeared somewhere in the source program, in the name field.

This is a criterion frame. If you answer this frame correctly, you have mastered the relationship between symbols, Location Counter, and the Symbol Table.

Assume that the Location Counter is sitting at hexadecimal 128. Show the symbols and their hexadecimal addresses that will be put in the symbol table as a result of processing the following statements only. Note: You could use your Reference Data card to figure out instruction lengths, but that would take more time than it's worth in this case. We have shown the length (in halfwords) to the right of each instruction.

NAME	OPERATION	OPERAND	LENGTH
SANDY	CR	1,2	1
	BC	2,MVAB	2
NUMB2	CR	1,3	1
	BC	2,MVAC	2
NUMB3	CR	2,3	1
	BC	2,MVBC	2

●●●

SYMBOL TABLE	
SYMBOL	ADDRESS
SANDY	128
NUMB2	12E
NUMB3	134

As a result of processing the symbolic statements, the three symbols shown above were placed in the symbol table. Their corresponding addresses (shown in hex) are also put in the symbol table. Notice that the three symbols in the operand field (MVAB, MVAC, MVBC) are not in the answer. Actually, they would have to be put in the symbol table in order to assemble the symbolic program. They would be placed in the symbol table when the assembler processed the symbolic statements that had those symbols in the name field.

There are two types of assemblers used with IBM computers: single phase and two phase. The assemblers you'll be working with as you program the System/360 are two phase assemblers. Single phase assemblers can only process source programs in which symbols are defined in the name field prior to being used in the operand at a statement. Two phase assemblers can process statements which have symbols in the operand even though that symbol wasn't in the name field of an earlier statement. However, even with two phase assemblers, the symbol must appear in the name field somewhere in your source program. The first phase of a two phase assembler does not produce an object program. It is used to read the source program and buildup a complete symbol table. It does, however, have an intermediate output consisting of partially assembled statements. The intermediate output from the first phase is used as input data for the second phase. During this phase, the assembler program uses the symbol table to complete the assembly of the statements. The output of the second phase is the object program and program listing.

4. What type of assembler is needed to process the following source program?

NAME	OPERATION	OPERAND
	START	256
BEGIN	BALR	11,0
	USING	*,11
	L	3,OLDOH
	A	3,RECPT
	S	3,ISSUE
	ST	3,NEWOH
	EOJ	
OLDOH	DC	F'9'
RECPT	DC	F'4'
ISSUE	DC	F'6'
NEWOH	DS	F
	END	BEGIN

a. Single Phase Assembler
b. Two Phase Assembler

• • •

b.

A two phase assembler is needed to assemble the program shown in the preceding frame. In the first phase, the assembler would put the symbols BEGIN, OLDOH, RECPT, ISSUE, and NEWOH in the symbol table along with their machine addresses. In the second phase, it would assemble the object program by obtaining the addresses from the symbol table. For instance, when the statement with the mnemonic L is encountered in the second phase, the assembler would look up the symbol OLDOH. When the assembler finds it in the symbol table, it would take the machine address and insert it in the assembled instruction.

Let's summarize what we have been discussing about the assembly process.

- During assembly time, the assembler program is being executed using a source program as input data.
- The output data from the assembler consists of an object program and its program listing.
- A location counter in the assembler is used to keep track of the storage locations that will be used by the object program.
- When a source statement contains a name, the current setting of the location counter is given to the label.
- Each label and the address assigned to it is placed in the assembler's symbol table.
- System/360 assemblers are two phase assemblers.
- During phase 1, the source program is read, and the symbol table is generated.
- During phase 2, the symbol table is used to complete the assembly, and produce the object program with its program listing.

SECTION II

INTRODUCTION TO ASSEMBLER LANGUAGE CODING

There are certain fundamental coding practices that you should learn before you start reading examples and trying your hand at it. These include such items as:

- Using the coding sheet format.
- Directing the Assembler to establish constants and work areas.
- Assigning a general register as the base register.

Since these practices are assumed to be new to you, the Self-Evaluation Quiz is at the end of the section. You should read the entire section before taking the test, and review indicated pages, to correct any errors, before continuing with Section III.

ASSEMBLER LANGUAGE CODING

Learning Objectives

When you have completed this section and have taken the Self-Evaluation quiz, you will have demonstrated that you can:

- Tell the assembler where (in storage) to start assembling a program.
- Assign a register as the base register for a program segment.
- Define areas of storage to be used for input, output and work.
- Define constants to be operated on by instructions in a program.
- Insert comments, in a program, so that another programmer can easily relate groups of instructions to blocks in a program flowchart.

"STATEMENTS" ON THE CODING SHEET

Examine one of your coding forms. Notice that columns 1-71 are called the "statement". The "statement" is the portion examined by the assembler and used to produce the object deck and program listing. Column 72 is always left blank and columns 73-80 are only used to identify the program and put the source cards in sequence. This identification-sequence field (cols. 73-80) is not checked by the assembler, although it may appear in the program listing.

1. Look at your coding form and tell me how many fields are contained in a statement. ______________

•••

Four; cols. 1-71 contain the statement

2. As you can see a statement (cols 1-71) has four fields: name, operation, operand, and comments. All of these fields usually appear in a program listing. Three of them are used by the assembler to produce the object program. From what you know about symbolic programming, which field should have no effect on what is put in the object program?

•••

Comments

Comments are used for the program listing, and are an aid in de-bugging or analyzing a program. Note that the fields of the statement on your coding form are separated by a blank column. A blank is used as a de-limiting or separator character in our assembler language. That is why you are not allowed to use blanks in a label. The coding form is a free-form sheet. Although the operation field begins at column 10 of your form, it can actually begin in an earlier column as long as one blank column separates it from the name field.

3. If a four character field is used as a label, the operation field can begin in column ________ .

•••

6

As long as you leave one blank column, a statement field can begin in any column. However, for ease in reading the program listing, it is recommended that you use the area shown on the form.

4. You can get comments in your program listing by leaving at least one blank column between your comments and the _________ field.

•••

Operand

Besides leaving a blank column after the operand field, there is another way of getting comments in your listing. An asterisk (*) in column 1 of a statement identifies that entire statement as a comment field.

Note: For assembly, one card is punched for each statement on the coding sheet. Column 1 on the card corresponds to column 1 on the coding sheet, 2 to 2, etc.

5. No object coding is generated and the comment is printed on the listing. Figure 1 in your sample program book shows a listing in which comments have been included in both ways. Note that where there are two or more consecutive comment cards, each one requires an_________in column_____.

•••

Asterisk; 1

Except for comment cards (those with an * in column 1) all cards contain statements for use by the assembler. These statements fall in two main categories: machine instructions and assembler instructions. A machine instruction statement is used to tell the assembler to generate the object (machine language) coding for a System/360 instruction.

6. For instance, the following statement would tell the assembler to generate the machine coding for an unconditional branch instruction. As such, it is a(n) (machine/assembler) instruction.

```
RTN1        BC        15,BEGIN
```

•••

Machine

Any statement that tells the assembler to generate the object coding for a System/360 instruction is called a machine instruction. An assembler instruction on the other hand is any statement that tells the assembler to do something other than to generate a System/360 instruction.

7. The following (is/is not) an example of an assembler instruction.

RTN1 BC 15,BEGIN

• • •

Is not

There are many types of assembler instructions just as there are many machine instructions. Some assembler instructions cause coding to be generated for the object program and some do not. The following are some uses of assembler instructions:

- Generate data constants for the object program.
- Reserve storage locations within the object program for use as input-output areas or as work areas.
- Control the assembly process; such as setting the location counter to some value.
- Control the listing by telling the assembler to overflow to a new form.
- Telling the assembler when you intend to use a label that is defined in another program, to which you will link your program.

All statements, whether machine instruction or assembler instruction, require a mnemonic to be entered in the operation field.

8. Look at the list of instructions on your System/360 reference data card. Any entry in the operation field of a source statement that is not shown on the reference card may be the ____________ for some ____________________ instruction.

• • •

Mnemonic; Assembler

9. The three fields of a statement that can produce coding in the object program are the name, operation, and operand. Of these, the only field that must always be entered is the ____________.

• • •

Operation

A mnemonic in the operation field of a statement tells the assembler that it represents some specific machine or assembler instruction. The mnemonics representing machine instructions are shown on your System/360 reference cards. For assembler instruction mnemonics, you would have to refer to an Assembler Language manual.

10. Depending on what mnemonic appears in the operation field, entries may be required in the name and/or operand fields. Usually, entries are not required in the name field unless you want to define some ______________________________.

• • •

Label or equivalent (such as symbol)

11. All statements require entries in the operation fields. The name field usually doesn't require an entry unless you want to define some label so that you can refer to it elsewhere in your program. When a label has been defined by a name entry, the location of that statement can be used by writing the label in the ____________ field.

• • •

Operand

A label in the operand field of a statement represents the symbolic address of the statement which has that label in its name field. The operand field of a statement is the body of the machine or assembler instruction. Most statements require some kind of entry in the operand. The operand of many machine instruction statements contains the address of the 1st and 2nd operands. Note at this point, that the term "operand" has two meanings. With respect to System/360 instructions, it means the data that is to be operated upon. Most System/360 instructions have two operands. With respect to Assembler Language, an operand is one of the fields in a source statement.

Let's get back to machine instruction statements. The addresses of the data can be represented either by a symbol, an actual value, or a combination of the two.

12. The following machine instruction statement represents an RX format instruction. An ______ address is used for the general register and a ____________ address for the data in storage.

A 1,FIELDA

• • •

Actual; Symbolic

As was stated, the operand of a machine instruction statement can contain either actual or symbolic addresses. Your reference card shows you what to write for any machine instruction when you use actual addresses. Actual addresses are usually shown decimally. For instance, the following RR format instruction shows two actual addresses. Both are decimal rather than hexadecimal numbers.

```
AR      10,15
```

13. In the program listing, however, they would be shown in hex. Using the information on your reference card, show how the following machine language instruction would be coded in Assembler Language.

5A	F	2	A	010

The first thing you would have had to do is search the list of instructions for an op code of 5A. It is the second instruction down. You were then able to see its mnemonic was A. It uses the RX format, and the operand is written R1, D2(X2,B2). The instruction from the previous frame can now be written in this fashion. Note, however, the use of decimal rather than hexadecimal numbers.

● ● ●

```
A       15,16(2,10)
```

Looking at the operands in the list of standard instructions on your reference card, there are a couple of items to note. First, the sequence in which the fields are written is not the same as in machine language. For instance, the displacement field is written prior to the index or base registers. Second, a field that is ignored during instruction execution is not written in the symbolic sequence. For instance, note the Load PSW (mnemonic LPSW). This instruction takes a doubleword from main storage and makes it the current PSW. The LPSW instruction uses the SI format, but the I2 field (the second byte of the instruction) is ignored. Accordingly, it is not in the symbolic format on your reference card. It is written:

```
LPSW    D1(B1)
```

14. Here are a few assembler language instructions of various formats. Use the reference card to tell yourself what their machine language formats are.

ALR	6,11
AL	7,16(2,3)
NI	15(4),21
BXH	3,10,31(1)

● ● ●

1E 6 B; 5E 7 2 3 010; 94 15 4 00F;
86 3 A 1 01F

You have just noted the relationship of symbolically coded instructions to the RR, RX, RS, and SI formats. Next, note the SS format. This is a little different because of the length code. By referring to the Basic Instruction Formats chart in your System/360 reference card, you can see that the SS format can have either an 8-bit length code (L), or two 4-bit length codes (L1 and L2).

15. Here's an SS instruction. How would it appear in machine language?

NC 2(17,1),19(1) ____________________

● ● ●

D4 10 1 002 1 013; Op Code L B_1 D_1 B_2 D_2

(Don't forget that the L code is one less in machine language.)

When an actual address or number is used, we say that you are using a self-defining value or self-defining term. The self-defining values we've been using have been decimal terms such as 10 instead of hex A. In Assembler Language, a self-defining or actual value can be used in the operand field and expressed as either a decimal, hexadecimal or character value.

To express a self-defining value as something other than decimal, the value is enclosed in single quotation marks and preceded by a descriptive letter. For instance, to tell the assembler that a self-defining value is hexadecimal, we precede the value with the letter X. A character value is preceded by the letter C.

16. The following are some self-defining (actual) values. Indicate whether they are decimal, hexadecimal, or character values.

X'100' ____________________

100 ____________________

C'?' ____________________

● ● ●

Hexadecimal; Decimal; Character

The entries in the operand of your source statement will be symbolic, self-defining, or a combination of both. When using a self-defining value, you can express it as any of the three types: decimal, hexadecimal, or characters. Your usage of them will usually dictate which type of self-defining value will be used. For instance, to add the contents of general register 14 to those of general register 11, you could write:

```
AR      X'B',X'E'
```

However, you would probably prefer to write it as:

```
AR      11,14
```

Usually hexadecimal self-defining values are used when you, as a programmer, wish to manipulate individual bits. You won't need to do this in the coding problems for this course, so you needn't concern yourself with it at this time. How about character values? When would you use them?

Let's illustrate one case by using the Compare Logical instruction (SI format). This instruction can be used to compare EBCDIC information. Suppose that in processing a particular card file, you are only looking for records which have an asterisk punched in column 1 of the card. Assume the card input area is labeled CARD IN. The Compare Logical (SI Format) instruction could be used as follows:

```
CLI     CARDIN,C'*'     Does column 1 contain an
                        asterisk?
BC      X'8',CDROUT     If so, branch to process
                        card.
BC      15,READCD       If not, branch to read
                        another card.
```

17. What would you be doing if, instead of the first instruction shown above, you wrote:

```
CLI  CARDIN,C'C'
```

● ● ●

Looking for cards which have a C punched in column 1.

We have been discussing the coding sheet and its entries. The coding sheet can be summarized as follows:

- Columns 1-71 of the coding form are called a statement.
- The statement has four fields: name, operation, operand, and comments.
- Although there are specific areas on the coding sheet, the field can be written free-form by allowing one blank column between them.
- If there is an * in column 1, the entire statement is a comment.
- An entry in the name field is optional. A label (symbol) is put here if you wish to give it a symbolic address for reference.
- The operation field entry is mandatory (except for comment cards).
- The operation field is given a mnemonic representing either a machine instruction or an assembler instruction.
- The entries in the operand field depend on the specific type of statement.
- Machine instruction statements may have symbols or self-defining values as entries.
- Symbols entered in the operand field must be defined in the name field of a statement.
- Self-defining values may be decimal, hexadecimal or character values.
- 10 is a decimal self-defining value.
- X'10' is a hexadecimal self-defining value.
- C'#' is a character (EBCDIC) self-defining value.

SYMBOLS

We have been discussing the use of both symbols and self-defining values in the operand of a statement. As you know, a symbol is a symbolic address, and as a general rule must be defined as the name of some statement in your program. Since you will be using symbols in any program you write, you should know which symbols are valid and which are invalid. The rules for symbols are relatively simple.

- Symbols may contain from one to eight alphameric characters (A-Z, 0-9, $, #, and @).
- The first character must be alphabetic.
- Symbols may not contain any special characters or imbedded blanks.

1. Which of the following are not valid symbols?

a. ALPHA	f. 4F
b. Z	g. FIVER
c. FIELD1	h. RDACARD
d. RTN#1	i. RTNNO4
e. FLD A	j. 1DER

• • •

e, f, j

Here's how to determine the address assigned to each symbol:

The location counter is incremented after each statement is processed. The amount that is incremented depends on the number of bytes required by the object program for each statement. For example, a machine instruction using the RX format would cause the location to be increased by four.

If the statement is named, the setting of the location counter (before being incremented) is assigned to that symbol. Then the location counter is increased. The symbol and the assigned location are recorded in the assembler's symbol table. Whenever the symbol is used in the operand of a statement, the assembler will look it up in its symbol table and obtain the symbol's assigned address.

2. Assume that the location counter is set at a hexadecimal 1000. Given the following list of statements, state the hex address assigned to the symbols BEGIN and FINISH.

```
__________ BEGIN    MVC    FLDA,FLDB
                    L      1,FLDB
                    AR     1,1
                    STH    1,FLDB
__________ FINISH   BC     15,BEGIN
```

• • •

1000; 1010

Besides placing the address "attribute" of a symbol in the symbol table, the assembler also puts in its length "attribute". That is, when the assembler encounters the statement named BEGIN (see above), it will put the following in its symbol table:

- The symbol itself - BEGIN
- Its address "attribute", the current location counter setting - 1000
- Its length "attribute" - six bytes (because of SS format)

3. What is the length "attribute" of the symbol FINISH? ______________________________

• • •

4 bytes (because of RX format)

What happens now when the assembler finds a symbol in the operand of a statement? Well, it goes to the symbol table and gets its address. If the length of the field named by that symbol is necessary (SS format), it too can be obtained by the assembler program. There are two more concepts concerning symbolic addressing that you must master before proceeding in this course. They are: the significance of an * as a special symbol and the meaning of relative addressing. First, let's look at the asterisk. When the asterisk is used in the operand rather than a symbol or self-defining value, the assembler treats it as a special symbol whose value is the current setting of the location counter. For example:

```
          BC          15,*
```

The previous instruction is recognized by the assembler as a machine instruction. The asterisk is recognized as a special symbol meaning the current Location Counter setting. Since this setting will be the assumed location of the BC op code, the assembler will assemble an unconditional branch to itself in the object program.

To illustrate this point further, assume that the location counter is set to 2884, and the next instruction to be assembled is

```
BC        15,*
```

The location of the op code of this instruction is, therefore, 2884. Since the * assumes the value of the location counter, the instruction becomes, in effect,

```
BC        15,2884
```

This is an unconditional branch to location 2884 (op code BC), and thus the instruction is branching to itself. This will create an endless loop until some event, perhaps an interrupt, causes us to leave the problem state.

4. Assume the location counter is set to 1112. What is the equivalent form of

```
L         1,*
```

•••

L 1, 1112

5. What will this load instruction do?

•••

It will load itself into Reg. 1 (This example has little practical use. It is used only to illustrate a point.)

An asterisk in the operand of a statement is always assumed to represent the current setting of the Location Counter. For machine instructions, this always means the location of its op code.

6. What value will the * assume when this sequence of instructions is executed? Note that you are asked to assume that the program has been assembled and loaded back into the computer as an object program.

Current setting of Location Counter is X'100'.

```
LH        1,FLDA
AH        1,FLDB
L         2,*
```

•••

108

The Load instruction in the preceding frame will load itself into general register 2. Since the original setting of the location counter is 100 and the first two instructions are 4 bytes each, register 2 will be loaded with the data found at address 108, which is the load instruction itself.

The asterisk is often used with a plus or minus value such as:

```
BC   15,*+8        (RX format:  4 bytes)
A    1,FIELDA      (RX format:  4 bytes)
ST   1,FIELDB      (RX format:  4 bytes)
```

7. If an asterisk refers to the value in the location counter (the op code of the BC instruction), *+8 would refer to a value eight bytes higher. The BC instruction in the above example would cause an unconditional branch to the instruction whose mnemonic is ____________ .

•••

ST

The use of a plus or minus value with an asterisk or symbol is called relative addressing. Relative addressing reduces the number of symbols that you must have in a program. The assembler program has a limit (depending on storage availability) on the number of symbols used. The use of symbols in a program also increases assembly time because of the need to look up the symbols in the symbol table. There is also the time required to write it out at the end of phase 1. Symbols are still needed to refer to locations that are quite far apart in your program. But where you want to refer to a location just a few bytes away, relative addressing can save you labels.

8. Write the necessary operand to cause the BC instruction to branch back to the MVC instruction without the use of a defined symbol.

```
MVC       FLDA,FLDB
L         1,FLDB
AR        1,1
STH       1,FLDB
BC        15,________
```

•••

*-16

Relative addressing can be used with symbols as well as with the asterisk.

9. With the use of symbol BEGIN and relative addressing, write the necessary operand for the BC instruction to branch to the L instruction.

```
BEGIN    MVC    FLDA,FLDB
         L      1,FLDB
         AR     1,1
         STH    1,FLDB
         BC     15,________
```

• • •

BEGIN+6

Let's summarize our discussion of symbols:

- Symbols are defined in the name field using from one to eight alphameric characters (A-Z, 0-9 and $, # and @).
- The first character of a symbol begins in column 1 and must be an alphabetic character.
- Symbols may not contain special characters or imbedded blanks.
- When a symbol is defined, the setting of the location counter becomes its address value.
- Symbols are recorded in the assembler's symbol table with their address and length values.
- Undefined symbols cannot be used.
- The asterisk is used as a special symbol whose value is the current setting of the Location counter.
- Relative addressing can be used with either the asterisk or symbols.
- FIELDA+8 is an example of relative addressing.
- Relative addressing reduces the number of symbols used in a program.

BASE REGISTER AND DISPLACEMENT ASSIGNMENT

In the System Review text, you learned that one of the important features of System/360 was program relocatability. This, in turn, was due to the fact that the address of each instruction is composed of a base address and a displacement, and the base address is contained in a general register. In order to locate a program in a different area of storage from that assigned at assembly time, all we need to do is start it off with a different base address.

In this section you will learn how a base address is specified, and how a general register is assigned as the base register for the program.

The first instruction that you need to learn about is an assembler instruction called USING. The second operand of this instruction gives the number of a general register, and the instruction tells the assembler that you are using that register as the base register.

The other operand tells the assembler what base address to use as the contents of the specified register. Thus, the assembler can calculate displacements and assign addresses to the symbols in the symbol table, as it assembles the program.

For the moment, we will leave the first operand of the USING instruction blank, and only show the one that specifies the base register:

```
         USING          ,4
```

1. This instruction tells the assembler that you are using general register 4 as the base register. Which register is specified by the following (incomplete) assembler instruction:

```
         USING          ,11
```

• • •

General register 11

Now for the first operand of the USING instruction. If we specify it (e.g., X'800' or 2048) by a self-defining value, it is the address that the assembler will assume to be the base address: Every label will be given an address based on this fixed value (plus a displacement) and our program will not be relocatable.

We need a way to specify a base address that is not a fixed value, but rather depends on the location of the program in storage when it is loaded at object time: This value is the setting of the location counter, specified by *.

If we write USING *, 7 we will be telling the assembler to assume that the value of the location counter setting is to be the base address, and that the general register 7 is to be the base register.

So far, so good. Regardless of where in storage our program is loaded (located), we have arranged for the setting of the location counter (the beginning of the program) to be the base address and have specified the register that holds it. Incidentally, since USING is an assembler instruction, it does not become a part of the object program, and, therefore, does not take up any room in storage.

The USING instruction merely tells the assembler that a certain register is the base register and to assume that it contains a certain value (such as the setting of the location counter).

We haven't actually arranged for the setting of the location counter to be placed in a register. We do it with a machine instruction, called "Branch and Link" (registers), whose mnemonic is BALR.

The effect of BALR is to store the address of the next machine instruction in one register (specified by the R1 operand) and branch to an address contained in another register (specified by the R2 operand).

But if the R2 operand is 0, no branch occurs: the effect of the BALR in that case is simply to store the address of the next machine instruction in the register specified by R1.

For example, BALR 2,0 stores the address of the next machine instruction in general register 2, with no branching.

2. What would BALR 5,0 do?

•••

Store the address of the next machine instruction in general register 5 and would not branch.

Note: When the BALR instruction is being assembled the address of the next machine instruction is the setting of the location counter. Thus, if we follow BALR 5, 0 with USING *,5 we place the setting of the location counter in register 5, and we tell the assembler that the setting is the base address and that we are using register 5 as the base register.

Here is another example:

```
         BALR    2,0
         USING   *,2
BEGIN    L       4,FIELDA
         A       4,FIELDB
         ST      4,FIELDC
         BC      15,BEGIN
```

3. The BALR places the address of the next machine instruction in general register 2. Since USING is an assembler instruction, not a machine instruction, which instruction's address will become the base address for the program?

•••

The address of the L 4,FIELDA instruction labelled BEGIN.

In the above example, the address of the Load instruction at BEGIN is placed in register 2, no branch is taken, and the computer will execute the Load instruction next. This occurs at object time. At assembly time, the USING statement tells the assembler to assume that register 2 will contain the current setting of the Location counter.

4. Suppose that the location counter is initially set at X'1000'; the assembly of the BALR instruction, itself, will cause the location counter to step up 2 bytes, and the setting will be __________ .

•••

X'1002'

5. Since the location counter was increased by two after processing the BALR statement, it is sitting at X'1002' when the USING statement is being processed. Since the USING statement doesn't affect the location counter, the symbol BEGIN is assigned an address value of X'______'.

•••

X'1002'

Note that we said that the assembler was told to assume that X'1002' would be in register 2. Depending on where the object program is subsequently loaded, the value that is put in register 2 by the BALR instruction may vary. Regardless, however, of what actual value is loaded into register 2 by the BALR, it will always be the address of the next instruction. Since displacement value is always relative to the base address, it doesn't make any difference what actual value is placed in the base register. The use of the BALR instruction will always allow us to relocate programs without having to change displacement values.

Since displacement values are always positive numbers, we can't expect the assembler to compute the displacement for any symbol whose storage address is less than the assumed base address. For instance, you can't do this:

```
BEGIN    BALR     4,0
         USING    *,4
         L        1,FIELDA
         A        1,FIELDB
         ST       1,FIELDC
         BC       15,BEGIN
```

You can't use the symbol BEGIN in the BC instruction because BEGIN has an address value that is two less than the base address you told the assembler you were using. Of course, there shouldn't be any reason for ever wanting to branch back to BEGIN. The base address only needs to be loaded once.

Sometimes you may be involved with a program that uses more than 4096 bytes. In this case, you will need more than one base register. With several USING statements and relative addressing, you can tell the assembler which registers to use and what will be their contents.

For instance:

Assume the location counter is initially set to X'1000'.

```
         BALR     4,0
         USING    *,4
         USING    *+X'1000',5
BEGIN    L        1,FIELDA
```

6. The first USING statement tells the assembler to assume the address of the Load instruction will be (at object time) in register________ . The second USING statement tells the assembler that register 5 will contain an address of X'____'.

• • •

4; X'2002'

Since the location counter was initially set to X'1000', it will be at X'1002' as the two USING statements are processed. The first one says that register 4 will contain X'1002'. The second USING statement says that register 5 will contain *+X'1000' (X'1002'+X'1000'). Notice that only register 4 will be loaded at object time in the previous example. Register 5 could be loaded with an address 4096 bytes higher than that in register 4 by the following:

```
BALR     4,0
USING    *,4
USING    *+X'1000',5
```

Register 5 would be loaded with a value of X'800' by this instruction:

```
LA       5, 2048
```

Register 5 would be loaded with a value of X'2002' by this instruction: (Assume registers 4 and 5 contain X'1002' and X'800' respectively, prior to the instruction. Note that calculation of the final address involves adding the D2, X2, and B2 components of the instruction.)

```
LA       5, 2048(4,5)
```

Let's summarize our discussion of base register and displacement assignment:

- The assembler cannot assign base registers and compute the necessary displacement values for your symbolic addresses unless you provide the necessary information.
- The USING statement provides the assembler with the address of your base register and the value of the base address.
- The following statement tells the assembler that the address of the next sequential byte in the object program will be in register 7, which can be used as a base register.

```
USING    *,7
```

- It is up to you to see to it that the base register is loaded at object time.
- The BALR instruction is normally used to load the base register.
- The following statements will provide the necessary information at assembly time and take the necessary action at object time:

BALR	7,0	Loads base register at object time.
USING	*,7	Tells assembler at assembly time.

You will be seeing the USING statement in every program listing throughout this course. As a result, you will become quite familiar with it.

THE START AND END STATEMENTS

Two more statements that you should understand before you can write a program or analyze a listing are the START and END statements. Both of these are assembler instructions and produce no coding in the object program. Up to this time, we've been telling you to assume that the location counter has a particular value. Now you will see how the location counter is initially set.

Let's take a look at the first of these assembler instructions: the START statement. This card is usually the first card in the source deck. Besides indicating the beginning of the source deck, the START statement card is used to:

- Give a name (symbol) to the program.
- Provide the initial setting to the location counter.

If the operand of a START statement is blank, the location counter is initially set to zero by the assembler. Since the lower storage locations are permanently assigned locations for such things as old and new PSW's, it is necessary to set the Location counter to a higher value (See the Permanent Storage Assignment chart on your System/360 reference card).

* * *

1. Given the following, the assembler will assume that register 7 will contain a base address value of X'________ .

```
PROGA    START    X'F00'
         BALR     7,0
         USING    *,7
         L        4,FIELDA
```

• • •

X'F02'

2. Write the necessary statements to:
   1. Initially set the location counter to decimal 2048.
   2. Tell the assembler to use register 4 as a base register.
   3. Load the base register at object time.

____

____

____

• • •

Either of the following could be the correct answer.

```
START    2048              START    X'800'
BALR     4,0               BALR     4,0
USING    *,4               USING    *,4
```

Another reason for using the START statement is to give your program a name. There are two reasons for this:

- To provide a means of branching to the beginning of your object program after it has been loaded.
- To provide a way for another program or program segment to link to your program.

Just as the START statement is the first card of your source program, the END statement will be the last card. The mnemonic of END tells the assembler that the assembly is finished.

The operand of the END statement usually contains the address of the first instruction you wish executed. It may be left blank. In this case, control will pass to the first byte in your program.

For instance:

```
BEGIN    START    X'1000'
         BALR     4,0
         USING    *,4
         L        2,FIELDA
           ~        ~
         END      BEGIN
```

3. Since BEGIN will be the beginning of this program, did the END statement require an operand? Yes or No.

• • •

No

4. You have not yet seen how data fields or data constants are defined. However, it is entirely possible that you might want to put some data ahead of the instructions of your program. In this case, would you want to leave the operand of the END statement blank? Yes or No.

• • •

No

If you have data preceding the instructions in your program, you would want to define a symbol as the name of the first instruction you wanted executed after load time. That symbol should then be written as the operand of your END statement.

5. What should be entered as the operand of the END statement? Choose one:
   a. BEGIN
   b. LETSGO
   c. COMEON
   d. Left blank

```
BEGIN     START    2048
          DS       CL100    This reserves
LETSGO    BALR     4,0       100 bytes of
          USING    *,4      storage.
COMEON    L        2,FIELDA
          {        {
          END ___________
```

• • •

LETSGO

You haven't studied the DS statement. But, the comment above stated that 100 bytes of storage were reserved. In other words, BEGIN has an address value of decimal 2048 and LETSGO will have an address value of decimal 2148. If you had left the operand blank or had written BEGIN, the loader would have passed control to your object program at the first byte of the storage area. I'm sure that's not where you wanted to be. If you had written COMEON, the loader would have put you at the Load instruction. This is okay except you would have missed loading your base register.

The use of the START and END statements is fairly simple. Let's summarize.

- The START statement is the first card in your source deck.
- The START statement is used to provide the initial setting of the location counter.
- If the operand of the START statement is left blank, the location counter will be initially set to zero.
- The START statement should be named to provide an entry point from some other program.
- If the END statement contains an operand entry, the loader will pass control to that location in your program.
- If the END statement is left blank, the loader will pass control to the first byte of your object program. This should necessarily be the first instruction you wanted executed.
- As a general rule, you should play it safe by putting in the operand of the END statement the symbol of the first instruction you want executed.
- The END statement terminates the assembly operation.

DEFINING STORAGE AREAS

You have learned a good deal about the fundamentals of the Assembler Language. You have examined the coding sheet and the entries you must make on it. The use of symbols and how to write valid ones has been explored. You've seen that the operation field must always contain the mnemonic for either a machine instruction or an assembler instruction. By using the System/360 reference card, you can code machine instructions using actual (self-defining) values. By means of the USING statement, you were able to give the assembler the necessary information for automatically computing base register and displacements. This allowed you to use symbolic addresses instead of actual values. By means of the START statement, you were able to initially set the value of the assembler's location counter. With the END statement, you were able to provide for a branch to the correct starting point in your program after load time.

However, there is one more topic to be discussed before you can begin to code source programs or analyze a program listing. You know that a program consists of more than just machine instructions. Instructions are no good unless they have data to operate upon.

There are three kinds of data used in a program. There are (1) the constants which remain relatively unchanged in storage (2) the data which is brought in from an I/O device and placed in an input-output area in main storage, and (3) intermediate results of calculations or logical operations. For instance, in a payroll job the withholding rates may be constants in main storage while the employee's records would probably be read into a storage area from some I/O device.

First let's see how data areas are defined. A data area is not necessarily an input-output area. It could for instance, be a work area. The number and types of data areas required depends on the complexity of the particular program. For instance, in a simple job, a record might be read into an input area, processed in that area, and written out from the same area. In a more complicated program, we might want to use two input areas so we could overlap processing with input.

For instance, if we have two input areas for a given file, we could read into input area 1, and then process that record as the next record from the file is being read into input area 2. To allow output operation to be overlapped with processing and input operations, it is also desirable to have two output areas. Figure 2 in your sample program book illustrates this point.

Sometimes several data records are written on magnetic tape as one tape record. We call this "blocking records". When our input file consists of blocked records from tape, each data record can be moved from its input area to a separate work area for processing. After the data record is processed, it can be moved to one of the output areas. Figure 3 illustrates this type of processing.

We are not attempting to teach you the various methods of input-output data processing. We are trying to point out the need for defining storage areas in our program for use as input areas, output areas, and work areas. We need to reserve some storage location for these areas and we would like to be able to refer to them with symbolic addresses, such as INAREA or HRSWKD. We can't define the value of the data because it is unknown and will be coming from an input file or will be the result of our processing. We can, however, define the length of the storage area. To define these areas, an assembler instruction with a mnemonic of DS (define storage) is used.

* * *

1. The following DS statement will define a storage area of 80 bytes whose symbolic address is ___________ .

```
INAREA          DS          CL80
```

• • •

INAREA

2. You are already familiar with the use of the letter C to indicate a character value such as C'ABC'. With the DS you can't define the character value, but you can state the number of characters for which you want storage reserved. This is done with the letter L followed by some decimal value. The following DS statement will reserve a storage area of _________ bytes.

```
OUTPUT          DS          CL100
```

• • •

100

3. Assume that the location counter is at a hexadecimal 1000. What addresses (in hex) will be assigned by the assembler to the symbols OUTPUT and INAREA respectively? __________ , ____________ . What will be their length values (in decimal)? __________ , ___________ .

```
OUTPUT      DS      CL16
INAREA      DS      CL80
```

• • •

1000; 1010; 16; 80

It should be noted that the DS statement does not cause any data to be generated. It simply reserves storage. During assembly time, it causes the following action:

- The location counter setting is assigned as the value of the symbol in the name field.
- The length of the storage area (given in the operand of the DS) is assigned as the length value of the symbol. This value is also used to step the location counter.

The length value given in a DS may be defined by means of a decimal number or may be calculated by the assembler. The latter occurs when no length is specified, but a constant <u>is</u> described. In this case, the assembler determines the length of the field and reserves the appropriate amount of storage, but does not assemble the constant.

4. Which of the following is/are valid DS statement(s)?

```
a.  INAREA      DS      C80
b.  OUTPUT      DS      CL800
c.  WKAREA      DS      C'123456'
d.  AREA1       DS      CLX'80'
```

• • •

b;c

Answer <u>a</u> is invalid because L was left out. Answer <u>d</u> did not have a decimal number for the length value.

In the preceding examples, a C was used to describe the area as storage for characters of data.
It is also possible to write a DS with an X, describing the area as storage for hexadecimal data.

For example:

```
HEXNUM          DS          XL15
```

This instruction assigns an address and a length attribute (of 15 bytes) to the label HEXNUM. It also causes the location counter to be advanced by 15.

5. Does the instruction cause data to be stored at location HEXNUM?

• • •

No. A DS only reserves storage.

You have seen how storage can be reserved for one data field. It is also possible to reserve more than one identical storage area (useful for blocked records). For instance, if 5 data records of 80 characters are written as one tape record, the following DS statement will reserve the necessary storage area for input purposes.

```
INAREA          DS          5CL80
```

The DS statement can also be used to reserve fixed length storage areas such as halfwords, fullwords, or doublewords. In these cases, the length is fixed and no length value is written. The basic format of the DS operand is d t Ln where:

- d is the number of consecutive storage areas.
- t is the <u>type</u> of area such as C for characters, H for halfwords, F for fullwords and D for doublewords.
- Ln is the length of character fields in decimal.

The following would reserve 4 words of main storage. FIELDA would be the name of the first word and would have a length value of 4 bytes.

```
FIELDA          DS          4F
```

6. Assume the location counter is initially at hexadecimal 1000. Show the location counter settings (in hex) for each of the following:

LOCATION CTR.	NAME	OPERATION	OPERAND
1000	A	DS	2H
____	B	DS	3F
____	C	DS	D
____	D	DS	2CL16
____	E	DS	CL80

• • •

1004; 1010; 1018; 1038

Notice that when one storage area is wanted, the number 1 does not need to be written.

Sometimes you only want to define a symbol without reserving storage. This can be done by specifying zero areas such as:

```
INAREA          DS          0CL80
```

In this case, the location counter setting will be given to the symbol INAREA and the counter <u>will not</u> be stepped. The INAREA will be assigned a Length attribute of 80. You are probably wondering why you would want to do this. The following should explain it.

Many data records contain descriptive fields which are not used in processing. For example, a payroll record would contain both the employee's name and his number. Since two employees might have the same name, the employee number is used for processing. The name is carried along for descriptive information so it can be printed. In such a case, you might want to give a name to the entire record area (such as INAREA) and to the fields within that record to which you might refer (such as HRSWKD). To do this you would want to define the area with a symbol (INAREA) and state that it consists of 80 characters (CL80). However, you don't want to step the location counter since you want to name some of the fields within the record. So you write the operand as 0CL80. Since the location counter didn't step, you can now name the necessary fields and state their length.

7. Given the following, check the address values of the symbols (in hex). Which one is wrong? Assume the location counter is initially at hexadecimal 1000.

LOCATION CTR.	NAME	OPERATION	OPERAND
1000	RD	DS	0CL80
	AREA	DS	CL20
1014	MANNO	DS	CL6
101A	HRSWKD	DS	CL4
101E	DATE	DS	0CL6
101F	DAY	DS	CL2
1020	MONTH	DS	CL2
1022	YEAR	DS	CL2
		DS	CL10
102E	GROSS	DS	CL8
1036	FEDTAX	DS	CL8
		DS	CL18

• • •

DAY should be at 101E, since the instruction DATE DS 0CL6 would not change the location counter

There is a reason you are being asked to use hexadecimal rather than decimal values when computing addresses:
The location counter values on your program listings will be shown in hexadecimal. By using hexadecimal now, you will find it much easier to work with these listings.

You have seen how you can reserve storage areas for use in your program. Note that the DS only reserves storage. It does <u>not</u> clear the storage areas. That will be up to you. You have also seen that by using a duplication factor of zero (e.g. 0CL80), you can name a storage area and give the symbol a length value without stepping the location counter. This allowed you to define fields within the storage area.

There is another use for a duplication factor of zero which has nothing to do with reserving storage. It has to do with boundary alignment. When fixed length fields (such as doublewords or instructions) are used, they must reside in storage on an address that is divisible by the number of bytes in the field. For instance, the leftmost byte of a doubleword must have an address divisible by eight such as 0018, 0F0, F00. Instructions are considered halfwords. An instruction's address must be divisible by two, even though instructions are two, four, and six bytes in length. To ensure

boundary alignment, the DS statement with a zero duplication factor should be used with the type of alignment desired. For instance, to step the location counter so that it will be at an even address, do the following:

DS 0H

Now I know you were told that the location counter didn't step when zero duplication factor was used. This was true when the zero duplication factor was used for a character area such as 0CL80. When the type of field is fixed length such as 0H, 0F, 0D, the location counter will step if it isn't on the correct boundary.

8. Assume the location counter is set at 1001. Where will it be after the following?

   DS 0H

   • • •

   1002

The location counter stepped to the next even address. Now try this one.

9. The location counter is sitting at hex 1001. Where will it be after the following?

   DS 0D

   • • •

   1008

Notice that the location counter stepped to the next address that was divisible by eight. The location counter only steps to get on the correct boundary. If the location counter is already on the right boundary, it doesn't step. Try this one:

10. Assume the location counter is sitting at hex 1000. Where will it be after the following?

   DS 0H

   • • •

   1000

Right again. Hex 1000 is divisible by two. As a result, the location counter did not have a step to get on the correct boundary.

Note, again, that 0H simply allows us to name a halfword of storage that begins on a halfword integral boundary.

An area of storage can be named and reserved and the location counter can be aligned on the appropriate boundary, by using an assumed duplication factor of 1:

FLDA DS H reserves a 2 byte area of storage named FLDA and aligns it on a halfword integral boundary.

FLDB DS F reserves a labelled fullword of storage, aligned on a fullword integral boundary.

FLDC DS D reserves an 8 byte area of storage named FLDC and aligns it on a doubleword integral boundary.

So much for the DS statement. Let's wrap it up!

- To reserve storage without defining the data that will be in it, the DS statement is used.
- The symbol used to name the DS statement will have an address value equal to the setting of the location counter.
- The length value associated with the name of the DS statement is affected by the type of defined storage area.
- The DS statement can be used to reserve storage for a variable number of characters or for the following fixed length fields: halfwords, words, doublewords.
- The operand format of the DS statement is d t Ln where:
  - d is the duplication factor.
  - t is the type of data that the area will be used for such as:
    - C = characters
    - X = hexadecimal
    - H = halfword
    - F = fullword
    - D = doubleword
  - Ln is the number of bytes reserved.
- A duplication factor of zero is used to align the location counter (when t is H, F, or D) or to name a storage area and give the symbol a length value (when t is C). For example:

	DS	0D	aligns the location counter on a doubleword boundary.
INAREA	DS	0CL80	defines the symbol INAREA and gives it a length value of 80, without stepping the location counter.

Just a reminder:

a. A "halfword integral boundary" is a storage address that is divisible by 2.
b. A "fullword integral boundary" is a storage address that is divisible by 4.
c. A "doubleword integral boundary" is a storage address that is divisible by 8.

If you are trying to determine which type of boundary is represented by an address in hex, convert the last hex digit to binary, and apply the following rule:

a. If the binary number has 1 low order zero, the address lies on a halfword boundary.
b. If the binary number has 2 low order zeros, the address lies on a fullword boundary.
c. If the binary number has 3 low order zeros, the address lies on a doubleword boundary.

DEFINING DATA CONSTANTS

You have just seen how to reserve storage for your input-output operations, as well as for work areas. Your program will probably also use some constants. A constant by definition is a fixed data value which comes into storage as part of your program. Constants are used for many purposes. Sometimes you might want a constant as a means of incrementing a programmed counter. At other times constants are used in the actual processing of data. One example would be the use of withholding rates (constants) when computing the amount of withholding tax for each employee's record.

The assembler instruction with a mnemonic of DC is used for defining constants. The operand format of a DC statement is quite similar to that of a DS statement. It is dtLn'c' where:

- d — is the duplication factor (number of identical constants).
- t — is the type of data such as:
  - C for characters
  - H for halfword binary data
  - F for fullword binary data
  - X for hexadecimal data
- Ln — is the length value
- 'c' — is the constant itself

For instance, suppose that an area of your program is to be used for an operator message. If the beginning of the message is constant, you might want to define it and reserve storage for the rest as follows:

```
MESAGE    DS    0CL22
          DC    C'MOUNT INFILE ON'
          DC    C'TAPE#'
MES1      DS    CL1
```

Note the following concerning the DC statement as used in the preceding example.

- duplication factor can be left out if you want only one constant.
- the length value can be left out. The number of characters, including blanks, will become the length of the field.
- the character constant is enclosed with single quotation marks.

* * *

1. Write the statement necessary to define the following constant.

   THEbANSWERbISb ←—— b is a blank space

   ____

●●●

If you wrote it correctly, it would look like this:

```
DC      C'THEbANSWERbISb'
```

When characters are defined as constants, they will be included in your object program as bytes of EBCDIC information. See your Reference Data card for the coding of any character. Notice that most of the 256 possible characters do not have a printer graphic. In this case, you would show the actual machine language (bit coding) using hexadecimal notation. Each character would be represented by two hexadecimal digits.

For instance, this constant:

```
DC      C'DObITbNOW'
```

could also be defined this way:

```
DC      X'C4D640C9E340D5D6E6'
           ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
           D O b I T b N O W
```

2. Write the DC statement that will define THINK as a hexadecimal constant.

• • •

By using your card, you should have been able to define THINK as:

DC X'E3C8C9D5D2'

In the examples so far, a length value wasn't used. The maximum number of characters that can be defined with either C or X, and no length value, is 120. Because it takes twice as much space on the coding sheet to write characters in the form of hexadecimal digits, the maximum is 60 using hex digits. If a length value is used and it is smaller than the number of characters defined, the leftmost characters (for X constants) or the rightmost characters (for C constants) will be dropped. For example:

DC CL4'THINK' will generate THIN
DC XL4'E3C8C9D5D2' will generate HINK

I don't know why anyone would want to specify a length that is less than what he defined for constant. But, if he did, the constant would be shortened as shown above.

Note: The maximum length that may be specified for a "C" type or "X" type constant is 256 bytes.

However it is feasible that you, as a programmer, would want to specify a length greater than the defined constant.

You would do this when you want the assembler to pad the constant (extend it with either blanks or zeros). Character ("C" type) constants are padded with blanks (hex 40 is used) in the rightmost bytes. Hexadecimal ("X" type) constants are padded with zeros (00) in the leftmost bytes. For example:

DC CL8'THINK' generates THINKbbb
DC XL5'478341' generates 0000478341

(hex digits, two per byte)

3. Given the following:
1. Show the location counter setting for each statement (in hex).
2. Show the machine language (2 hex digits for each byte). Use hex 40 for blanks.

Assume the location counter is initially at hex 1000.

LOC. CTR.	MACHINE		
______	______	DC	XL4'42'
______	______	DC	CL5'IN'
______	______	DS	0D

• • •

LOC. CTR.	MACHINE		
1000	00000042	DC	XL4'42'
1004	C9D5404040	DC	CL5'IN'
1010	Nothing is generated	DS	0D

Note that besides the padding, the location counter was also aligned on a doubleword boundary. The second DC statement stepped the location counter from 1004 to 1009. 1009 is not divisible by eight. As a result, the DS statement caused the assembler to step the location counter 7 positions to location 1010 (in hex).

Because of the difference in the ways character and hexadecimal constants are shortened or extended, you may have guessed that the hexadecimal constant is used for purposes other than just defining unprintable EBCDIC characters. Often times you may want to define a stream of bits for use in your program. Since hexadecimal notation is a shorthand method of writing a long stream of bits, the hexadecimal constant is used in these situations. The hexadecimal constant is used for such purposes as mask fields, pattern fields for use in editing and in defining constants for use in the PSW.

We have been using the DC statement to write either character or hexadecimal constants. You have seen that these constants could have either an implicit length or an explicit length. When there is no length value (C'ABC' rather than CL4'ABC'), we say there is an implicit length which is equal to the number of bytes required by the written constant. An explicit length has the effect of either extending or shortening the constant. This occurs when the written length value is different from the implied length of the constant. For instance, the following DC statement has an explicit length of four and an implied length of three:

DC CL4'ABC'

Since explicit values always override implied values, the preceding constant would be extended with a blank in the rightmost of the four bytes.

4. What characters would be included in a program as a result of the following:

   DC CL 2'THINK'

   • • •

   TH

Two more types of constants that can be defined in the Assembler Language are halfword binary operands and fullword binary operands. The codes used for these constants are H and F respectively.

The following will define a storage constant consisting of a halfword binary operand equal to a decimal value of 7.

   DC H'7'

Since halfwords and fullwords define binary information, they should not be given an explicit length value. These two constants are very useful in that you can define binary operands without having to convert decimal values to binary. The following will generate a fullword binary operand equal to a decimal value of 103. Hexadecimal notation is used to show the generated machine language.

   source statement DC F'103'

Object coding generated by the above DC statement is:

00	00	00	67

This is a 32 bit signed binary operand equal to a decimal value of +103.

5. Write a statement named CON1 that will generate a fullword signed binary operand equal to a decimal value of 2400.

   ____

   • • •

   An unsigned value is assumed to be plus. You could have written the preceding with or without a plus sign as follows:

   CON1 DC F'2400'

   or

   CON1 DC F'+2400'

6. Write a statement that gives you a halfword binary operand equal to a value of -2.

   • • •

   Negative operands must always be signed. The preceding would be written:

   DC H'-2'

   Note that these binary operands are always written using the decimal value in the operand. The preceding is a negative binary operand. Negative binary operands in the System/360 are in complement form. The machine language generated by the preceding DC statement is:

   FFFE (hexadecimal notation)

7. The following are some DC constants. Write the generated machine language for each constant using hexadecimal notation. Use the hexadecimal-decimal conversion chart on your Reference Data card.

   ________________ DC F'0'
   ________________ DC F'2400'
   ________________ DC H'+15'
   ________________ DC H'100'
   ________________ DC H'-15'

   • • •

   00000000; 00000960; 000F; 0064; FFF1

So far you have seen four types of constants that can be generated with a DC statement: characters, hexadecimal or unsigned binary constants, halfword signed binary operands and fullword signed binary operands. These were defined using the codes C, X, H, and F respectively. More types of constants can be defined in Assembler Language. Two of these types of constants (codes E and D) have to do with floating point operands and we will not be concerned with them. The third (code A) has to do with defining an address for a constant. In this case, a symbolic address with or without relative addressing is usually used. Constants of this type are called address constants or "adcons". Adcons differ from other constants in that the address constant is enclosed in parentheses rather than in single quotation marks. The following are examples of adcons.

   DC A(FIELDA)
   DC A(INAREA+16)

Adcons can be rather tricky if you have had no past experience using them. It is very easy to confuse data with the address of the data. So, let's take a look at an adcon in usage.

Suppose that somewhere in your program, you want to branch unconditionally to an instruction named RTNE1. This could be written as:

```
BC      15,RTNE1
```

From the preceding statement, the assembler would produce a Branch on Condition instruction using the RX format. Assuming that RTNE1 has a displacement value of hex 100 and register 4 is the base register, the object language instruction would look like this:

```
47 F 0 4 100
```

The RR format can also be used for the Branch on Condition instruction. In this case, the register specified by the R2 field must contain the address you wish to branch to. For instance, if register 7 contained the address of RTNE1, you could write your unconditional branch like this:

```
BCR     15,7
```

Well, now, how do you get the address of RTNE1 into register 7? Can you do it this way? Take your time in answering this.

```
L       7,RTNE1
BCR     15,7
```

Yes	Go to the next paragraph.
No	Skip the next two paragraphs.

You said yes. You are wrong, but don't feel upset about it. This is a very common error. Let's see why the preceding example was wrong.

```
L       7,RTNE1
BCR     15,7
```

RTNE1 is the symbolic address of an instruction. What was placed in register 7 by the preceding Load instruction?

a.	The address of an instruction	Go to the next paragraph.
b.	The instruction itself	Skip the next paragraph.

The Load instruction puts in a register the data which is located at some address. In the preceding paragraph there is an instruction which is located at some address (RTNE1). It is the instruction itself which is placed in register 7 rather than its address. This is not what you wanted to do. You wanted to load the address of the instruction. Can you do it with this?

```
LA      7,RTNE1
```

Yes, this instruction (Load Address) would load the address of the instruction at RTNE1 into register 7. However, there is another way to do it, using an adcon.

You know that the following statement would not load the address of some instruction into register 7.

```
L       7, RTNE1
```

It was the instruction itself that was loaded and not its address. We could load the address by using the Load Address instruction as follows:

```
LA      7,RTNE1
```

We can also load the address of the instruction at RTNE1 by using an adcon. Since adcons are what we are interested in at the present time, let's see how one can be used.

8. First, let's define the address constant. Write the statement that will generate the address (RTNE1) as a data constant.

• • •

The address of the data at RTNE1 (which we are assuming is an instruction) can be generated as a constant in this fashion.

```
DC      A(RTNE1)
```

A constant isn't of any value unless we can refer to it. Let's give it a name.

```
ADCON1    DC      A(RTNE1)
```

We will now have a constant located at ADCON1. This constant will be an address. It will be the address of an instruction located at RTNE1.

9. With this in mind, write the instruction that will load this address into register 7.

____

● ● ●

Okay, let's see the entire picture including the correct answer to the preceding question. The correct answer is the first of the following statements:

NAME	OPERATION	OPERAND	
	L	7,ADCON1	
	BCR	15,7	
RTNE1	LH	4,FIELDA	This is the instruction we wanted to branch to.
	AH	4,FIELDB	
	STH	4,FIELDC	
FIELDA	DC	H'100'	
FIELDB	DC	H'200'	
FIELDC	DC	H'0'	
ADCON1	DC	A(RTNE1)	

The preceding is not necessarily the best way to use adcons. However, it serves our purpose of explaining an adcon and how it can be used.

10. Suppose that instead of branching to the LH instruction at RTNE1, you wanted to branch to the AH instruction that follows it. How would you write the necessary adcon? Hint: use relative addressing.

NAME	OPERATION	OPERAND
_____	________	________

● ● ●

If you wanted to branch to the instruction following the one at RTNE1, you could do it with relative addressing like this:

```
ADCON        DC           A(RTNE1+4)
```

Let's summarize our discussion of defined constants.

- A constant is data which is read into and resides in storage along with the instructions of your program.
- The assembler instruction with a mnemonic of DC is used to define constants.
- The operand format of a DC statement is dtLn'c' where
  - d is the duplication factor (number of identical constants)
  - t is the type of constant, such as characters, binary operands written hexadecimally, halfword and fullword signed binary operands, and address constants.
  - Ln is the explicit length value of the constant.
  - 'c' is the constant itself. Adcons are enclosed by parentheses.
- An explicit length value (Ln) is optional and is used to override the implied lengths of character, hexadecimal, and address constants.
- The following codes are examples used to define the type (t) of constant.
  - C - characters
  - X - hexadecimal
  - H - halfword signed binary operand
  - F - fullword signed binary operand
  - A - address constant (adcon)

  CL3'TIME' becomes TIM
  CL5'TIME' becomes TIMEb

  XL3'47FA4D0F' becomes FA4D0F
  XL5'47FA4D0F' becomes 0047FA4D0F
- The halfword and fullword constants are useful for generating signed binary operands while writing the decimal value.

  H'17' becomes 0011
  F'-4' becomes FFFFFFFC
- Adcons are useful for defining the address of some instruction or data as a data constant.
  Given the following:

  ```
  FIELDA          DC        F'103'
  ```
  The DC statement below will generate the address of the above constant:

  ```
  ADCON           DC        A(FIELDA)
  ```
- Adcons have an implied length of one fullword (4 bytes). If an explicit length is used with an adcon, the leftmost bits are dropped.

  ```
  ADCON1          DC        AL2(FIELDA)
  ```
  In this adcon, the rightmost 16 bits of the address are generated as a constant. This is useful on models of System/360 which have 64K bytes or less of main storage. Any decimal location from 0000-65,535 can be addressed with 16 bits.

So far, you have learned how:

- to code machine instructions using the System/360 reference card.
- to reserve storage areas (DS statement) for use in your program.
- to define constants (DC statement) for use in your program.
- to load the base register at object time so that the program is relocatable (BALR instruction).
- to tell the assembler what will be the base register and its contents relative to your program (USING statement).
- to initially set the location counter at the beginning of assembly time (START statement).
- to tell the assembler which instruction you intend to execute first (END statement).

We have finished our discussion on the fundamentals of the basic assembler language. You have not learned all there is to know about this language. There are a number of assembler statements which haven't yet been discussed. All of these points will be brought out in the following sections as you start to analyze some programs as well as develop some of your own programming.

However, you have learned enough about the basic assembler language so that you can evaluate a source program with instructions, constants, data areas, and the necessary assembler instructions for a successful assembly. This is your opportunity to test your progress in these areas.

Attempt to answer as many of the following questions as possible, without using reference material except the System/360 reference card. Answer the remaining questions by referring back to the indicated pages.

SELF-EVALUATION QUESTIONS

Reference Pages in Text

Use the following program to answer the questions.

```
              START     X'190'
BEGIN         BALR      11,0
              USING     *,11
              MVC       A,B
              L         1,C
              A         1,A
              ST        1,D
              BC        15,*-4
A             DS        CL4
B             DC        F'10'
C             DC        F'4'
D             DS        4C
              END       BEGIN
```

1. At assembly time, the asterisk in the USING statement has a hexadecimal value of — 19
   a. 190
   b. 192
   c. 194
   d. none of the above

2. What is the mnemonic of the statement which loads the base register? — 20
   a. START
   b. BALR
   c. USING
   d. END

3. What is the effect of the BC statement at object time? — 17
   a. causes an unconditional branch to itself.
   b. causes an unconditional branch to the ST instruction.
   c. causes a branch if the PSW condition code is set to 15.
   d. none of the above.

4. What is the effect of the DS statement named A? — 23
   a. generates a field of four blanks.
   b. generates a field of four zeros.
   c. reserves four storage bytes without generating any coding.
   d. none of the above.

5. Which of the following is the generated object coding (two hex digits per byte) of the DC statement named B? — 26
   a. 0000000A
   b. 00000010
   c. 00000002
   d. F1F0

	Reference Pages in Text
6. What is the net effect of the END statement? a. tells the assembler that this is the last source card. b. tells the loader to pass control to the instruction identified by its operand. c. causes a branch to the BALR instruction at the end of load time. d. all of the above.	22-23
7. Which of the following is not put in the symbol table when the BALR is processed by the assembler? a. the symbol BEGIN b. an address value of hexadecimal 190 c. a length value of 2 d. the mnemonic BALR	17
8. Given the instruction AR X'B',10. The first operand entry is in a. machine language b. symbolic language c. neither a. or b. d. both a. and b.	15
9. The first operand entry in the instruction in question 8 represents which sort of value: a. hexadecimal b. decimal c. character d. none of the above	15
10. An asterisk in the operand of an instruction stands for a. The setting of the location counter as of this point in the assembly. b. The address of the instruction in which it appears. c. neither a. or b. d. both a. and b.	17
11. What will be set up by the following instruction: DS 5CL80: a. An area of storage 400 bytes long. b. 5 areas of storage each of which is 80 bytes long. c. 80 areas of storage each of which is 5 bytes long. d. none of the above.	26

Reference Pages in Text

12. We might use a series of instructions such as the following: — 25

```
INAREA      DS   0CL80
            DS    CL10
BRANCHNO    DS    CL3
DEPARTNO    DS    CL5
EMPLOYNO    DS    CL7
DATE        DS   0CL6
MONTH       DS    CL2
DAY         DS    CL2
YEAR        DS    CL2
            DS    CL49
```

a. To define areas in storage which, in all, take up 166 bytes.
b. To identify ten fields for use by the program.
c. To identify fields within fields.
d. None of the above.

13. The instruction DS 0D always causes the location counter to be set to an address that is a — 25
a. halfword boundary
b. word boundary
c. three halfword boundary
d. doubleword boundary

14. What constant does DC XL4'1492' generate? — 27
a. 1492
b. 001492
c. 0001492
d. 00001492

ANSWERS

	Answer	Reference Pages in Text
1.	b	19
2.	b	20
3.	b	17
4.	c	23
5.	a	26
6.	d	22
7.	d	17
8.	c	15
9.	a	15
10.	d	17
11.	b	26
12.	c	25
13.	d	25
14.	d	27

SECTION III

CODING SAMPLE PROGRAMS

Now we begin the payoff on all that you have learned so far. As you are shown sample programs, you will be brought gradually into the activity of coding instructions for them.

CODING SAMPLE PROGRAMS

Learning Objectives

When you complete this course, you will have demonstrated that you can:

- Given a problem statement, a program flowchart, and instructions for linking your program to existing I/O routines,
  a. Identify the specific requirements for linking to the I/O routines and the constraints that they place on your program.
  b. Find and use information describing the functions of Assembler Language instructions. You should not try to memorize them.
  c. Code an Assembler Language source program for the solution of the problem.

The problem statement for the first coding demonstration is in Figure 4 of your sample problem book, and the flowchart is on the facing page.

Read the problem statement and scan the flowchart, before you answer the following questions.

1. How many lines of output data will we print for each transaction card?

   • • •

   one line of data per card

2. How many fields are there in each output line?

   • • •

   6

3. Are we punctuating the output fields with $ or decimal points?

   • • •

   no

4. The output fields are identified by fields in the ______ lines.

   • • •

   heading

5. Do we read the heading fields from heading cards?

   • • •

   no

6. We will need to establish the words for the heading fields as__________in storage.

   • • •

   constants

7. Is the formula printed alongside block E2 correct, according to the problem statement?

   • • •

   yes

8. When we perform the operations required to calculate monthly interest, how many (implied) decimal positions will the result contain? (Work it out, on scratch paper.)

   • • •

   6 decimal positions (The rule is the product will have a number of decimal positions equal to the sum of the decimal positions in the two factors. Principal has two decimal positions; Rate has four. 2+4=6.)

If you have performed the operation called "half adjusting", in a computer program or other series of calculations, skip the following two paragraphs.

When an arithmetic operation results in a number with more decimal positions than necessary, we must "round it off". We remove the unwanted positions. But we want our rounded figure to be accurate; we can't just knock digits off of the low order end of the field. For example, we often round off to the nearest cent:

a. 1734.506 is 1734.51, to the nearest cent, not 1734.50.
b. 84.002 is 84.00, to the nearest cent, not 84.01. In this case, our rounding operation must produce the same result as if we had simply removed the right-most digit.

Here is the rule: Add 5 (followed by an appropriate number of zeros) to the digit position to the <u>right</u> of the one that you want to keep. Then truncate (knock off unwanted positions).

a. Half-adjust and truncate 17.5683 to the nearest cent.

17.5683	
+ 50	half-adjusting
17.5733	
17.57	rounded answer

b. Half-adjust and truncate 387.764001 to three decimal positions.

387.764001	
+ 500	
387.764501	half-adjusted
387.764	rounded answer

9. Now you try one. Half-adjust and truncate 18.068621 to the nearest cent.

• • •

half-adjusting

```
 18.068621
+     5000
 18.073621
```

rounded answer   18.07

10. Assuming an interest field of XX.XXXXXX at the end of the calculation, into which position will we add 5 (to half-adjust to the nearest cent)?

• • •

```
XX.XXXXXX
    ↑
```

11. Remember that the decimal point is implied. In order to add 5 to this position, we must add (50/500/5000)______to the field.

• • •

5000

```
 XX.XXXXXX
+     5000
```

12. The rest of the arithmetic on the flowchart is self-evident. The only other point is that we need to "assemble a line" by bringing together six ________ fields, before we can print a transaction.

• • •

output

That's all there is to the basic analysis of the problem statement and flowchart. Now we turn to the coding considerations.

The coding sheets for this problem are reproduced on the next three pages of your sample program book. Leaf through them and then turn back to Figure 6. Note the page numbers at the upper right of each coding sheet, as we will refer to the sheets both by Figure numbers and page numbers.

Did you notice that the programmer used asterisks, with no comments, on some of the lines? This will produce corresponding spaces in the program printout and, by isolating instructions into functional groups, makes the program's logic much easier to follow.

The first thing you need to learn is which instructions to ignore in these sample programs. They are the ones which tell the Assembler to produce routines that perform input, output, and end of job operations.

The program that you write at the end of the course will be able to be linked to pre-written routines that will perform the I/O for you. All that you need is information on how to make the system use those routines, at appropriate points in your program.

IBM

IBM System/360 Assembler Coding Form

PROGRAM | PUNCHING INSTRUCTIONS | GRAPHIC | PAGE OF

PROGRAMMER | DATE | PUNCH | CARD ELECTRO NUMBER

```
         START 256
*
CARDIN   DTFCD                                                          C
               DEVADDR=SYSRDR,                                          C
               EOFADDR=EOJ,                                             C
               IOAREA1=INPUT
*
         CDMOD
*
ALINE    DTFPR                                                          C
               BLKSIZE=132,                                             C
               DEVADDR=SYSLST,                                          C
               IOAREA1=OUTPUT
*
         PRMOD
*
BEGIN    BALR  11,Ø
         USING *,11
         OPEN  CARDIN,ALINE
         BC    15,START
*
EOJ      CLOSE CARDIN,ALINE            END OF JOB ROUTINE
         EOJ
*
READ     GET   CARDIN                  READ SUBROUTINE
         BCR   15,1Ø
*
WRITE    PUT   ALINE                   WRITE SUBROUTINE
         BCR   15,1Ø
```

(A) brackets the DTFCD entries; (B) brackets the DTFPR entries; (C) brackets CLOSE CARDIN,ALINE and EOJ.

Let's suppose, for a moment, that this is the case with programming example 1: Someone else has coded the I/O subroutines and the other instructions shown at the left, and you are to code the rest.

A brief description of what these instructions do will show you what to look for, when you begin your part of the coding.

- DTF means "define the file", and the DTFCD macro (at A) describes the card input file to the Assembler. It also tells the Assembler that you are going to define a main storage area called "input". (IOAREA1=INPUT). Only this latter information affects your coding.
- CDMOD tells the assembler to produce a card input module (routine) using the file description provided by DTFCD.
- By now you may have guessed that DTFPR (at B) describes the printer output file and promises the Assembler that you are going to define a main storage area called OUTPUT. This affects your coding.
- PRMOD tells the Assembler to produce a printer output routine.
- OPEN tells the Assembler to produce a routine that makes the files available for processing. Notice that its operands, CARDIN and ALINE are the labels for the file definitions.
- GET and PUT generate instructions to use the input and output routines, respectively. Their labels tell the Assembler that you will use READ or WRITE, as an operand in your program, when you want to link up with one or the other of the I/O subroutines. The BCR instruction, which follows each of these, is a branch on condition instruction with a mask field value of 15. If you remember what you read about this instruction in System Review you know that this mask field will cause a branch under all conditions. The R2 operand is register 10, so the branch will be to some address, in the main part of the program, which was placed in register 10 when the branch to the READ or WRITE subroutine was issued.
- EOJ (at C) is a label for a routine to which the program will branch, when there are no more cards to be read. It will:
  a. Close the files (make the card reader and the printer unavailable to the system.)
  b. Signal to the Supervisor program that the job has been completed so that it can initiate a transition to the next job.

So much for I/O subroutines. The assembler instructions should be familiar to you by now:

- With his first instruction, the writer told the Assembler to start putting these routines together at location 256 (hex 100).
- BALR and USING established register 11 as the base register, and set the location counter to the address of the next instruction, which is the first instruction of the OPEN routine.

At this point we need to emphasize a distinction between events at assembly time and those at the actual execution of the program:

a. As we have said, the macro OPEN tells the Assembler to produce a routine that makes the input and output files available. This routine is produced at assembly time.
b. The machine instruction BC 15,START, which follows, is assembled as a link from the OPEN routine to the main section of the program.
c. When the program is executed, the system starts at the instruction labelled BEGIN. When it reaches the first instruction produced by the OPEN macro, it branches to the routine that makes the card reader and printer available for I/O operations. Now data processing can begin, so the system executes the branch (under all conditions) specified by the BC 15,START.

It goes directly to an instruction named START, which would be the first instruction in your part of the program.

Refer to the coding sheet pages to review the preceding information, before continuing.

Now that we have gotten to a part of the program like the one you, eventually, will be coding, let's summarize what you would need to keep in mind:

a. You will define storage areas called INPUT and OUTPUT.
b. You will use READ or WRITE, as an operand, when you want to read a card or print a line of output.
c. Register 11 has been specified as the base register, so you must not use it in any of your instructions.
d. You will use register 10 to hold the address to which the system will branch, after it reads a card or prints a line.
e. You will label your first machine instruction with START.

Now we'll go through the coding process as it was actually done, explaining each instruction as we come to it. You will have an opportunity to practice using the information about each instruction, both in and out of the context of the sample problem.

The first instructions that we code are not machine instructions. We know that we must define input and output areas, and it is good to do this first. Therefore, the coding sheet that we start with turns out to be page #5 by the time we're finished coding. Turn to Figure 10 in your sample program book.

First we define our input area. From what you have already learned about this, you should easily understand the following statements:

a. We define INPUT as a storage area 80 bytes long, one for each column in a transaction card.
b. The leftmost 0, in the operand 0CL80, allows us to name, and use, each of the input card fields (independently).
c. Now we start specifying each of the fields, according to the information given in the problem statement:
   (1) Account Number, columns 1-6, is labelled ACCTNO and is defined as a 6 byte field with DS CL6.
   (2) Principal, columns 7-13, is labelled PRIN and is defined as a 7 byte field with DS CL7. You should be able to relate the other input fields to the problem statement.
d. We aren't interested in using any data that might be punched in card columns 22-80, but we must account for the bytes it takes up. Therefore, we write an unlabelled DS CL59.

Now we do something a bit different. Thinking ahead to the time that we want to clear any leftover data from our output area, we specify a constant that is one blank byte. Note the DC that does it. Shortly, you will see how we will use that blank.

13. After an asterisk for spacing we continue, by defining our output area. Our printer prints a 132 character line, so our area has____bytes.

• • •

132

14. After writing the DS for the total area, we put in a DS that simply creates space for a left hand margin on our report. Which instruction does this?

• • •

DS CL33

Now we label the area which will contain all of our output fields. In order to determine its length, we:

a. Add up the lengths of the fields as shown on the problem statement.
b. Add a number of bytes for spaces between the fields, so that they will be positioned properly with respect to the heading field.

The total comes to 56. Note how the six fields within this area were labelled and specified.

15. We still have a few bytes to account for in our entire output area. They will simply provide space to the right of each line. We define this area with the instruction________.

• • •

DS CL43

16. Now we are ready to write our first machine instruction. Remembering what we noted earlier, when we were examining the I/O coding, we label this instruction______.

• • •

START

Turn to Figure 7 (coding page 2). Since you are not concerned with I/O coding, you would start coding on a new sheet. However, the person who coded program 1 simply continued after he was done with I/O.

17. According to the comment alongside the instruction labelled START, our first task is to____________.

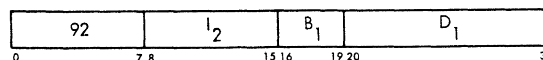
• • •

clear the output area

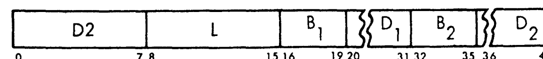
Read the following information on the move instruction. Also, locate the group of move instructions, that includes MVI and MVC, on your Reference Data card.

Move

MVI $D_1(B_1), I_2$ **[SI]**



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



In the execution of either MVC or MVI, the second operand is placed in the first operand location.

MVC (Move Characters)
- The bytes are moved one at a time in each field.
- Movement is left to right.
- The number of bytes moved is determined by the implicit or explicit length of the first operand.

MVI (Move Immediate)
- One byte of immediate data is stored in the first operand location. Immediate data is data supplied by the instruction itself, and in this case is the second operand.

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

Programming Note:

It is possible to propagate one character through an entire field by having the first operand field start one character to the right of the second operand field.

EXAMPLES:

Name (1–8)	Operation (10–14)	Operand (16–25)
	MVI	SWITCH,X'Ø1'
	MVC	OLD,NEW

18. Both MVI and MVC are called "________________" instructions.

•••

move

The effect of the first example would be to move a hex 01 into a storage area named SWITCH.

19. The effect of the second example would be to move the contents of an area called______into an area called OLD.

•••

NEW

20. Reread the programming note, and then look at the operands of our first MVC. Is OUTPUT one character (byte) to the right of OUTPUT-1?

•••

yes

Here is a reprint of a section of the coding sheet on which we defined storage.

Name	Operation	Operand
PAY	DS	CL4
	DS	CL59
*		
	DC	CL1' '
*		
OUTPUT	DS	ØCL132
	DS	CL33
HEADER	DS	ØCL56

21. As you probably remember, the * only affects the assembly listing; it does not become a part of the program in storage. This means that the field we have called OUTPUT starts one byte to the right of a character which we defined as a______.

•••

Blank

22. According to the programming note you read a moment ago, the effect of our MVC OUTPUT, OUTPUT-1 instruction would be to ______________ (your own words) ______________________________.

•••

propagate (reproduce) the blank throughout the field named OUTPUT

Neat, isn't it? Instead of defining constants with a total of 132 blanks and moving them into OUTPUT, we can clear the area by using one blank byte, next door. That's why we put it there.

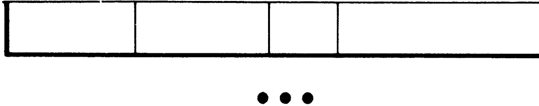
Now you, as a reader, come to a branching point. If you doubt that you can predict the result of executing either an MVI or an MVC instruction, regardless of the operands, you should study the frames on the next three pages. If you don't feel doubtful, skip the frames and we'll continue with our first programming example.

Hereafter, we'll refer to these branch points as "Skip Options".

MOVE INSTRUCTIONS

Mnemonic	Op Code	Format	Descriptive Title
MVI	92	SI	Move Immediate
MVC	D2	SS	Move Characters

1. The "move immediate" instruction uses the SI format. Label the fields of the SI format.



• • •

2. Operands that are carried in the instruction itself are called__________operands.

• • •

immediate

3. In the MVI instruction, the immediate operand is one_____long and is the________(1st/2nd) operand. The instruction will move the byte of immediate data to main_________.

• • •

byte; 2nd; storage

4. The MVI instruction will cause the byte in main storage to be replaced by a byte from the __________. The main storage address ________ (does/does not) have to be even.

• • •

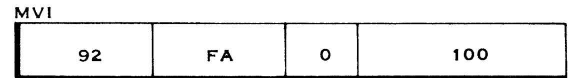
instruction; does not; Any byte in main storage can be addressed.

5. The MVI instruction can move ____________ (more than one/only one) byte of data.

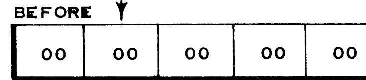
• • •

only one; Only the immediate byte in the instruction.

Given the following (in hex) show the contents of main storage after the MVI instruction is executed.



LOCATION SPECIFIED BY 1ST OPERAND



6. AFTER MAIN STORAGE

• • •

7. MVI is the mnemonic for the move immediate. MVC is the mnemonic for__________ __________.

• • •

move characters

8. The MVC instruction uses the SS format because both operands are __________(fixed/variable) length fields in main storage.

• • •

variable

9. The MVC instruction moves bytes (characters) from one area of main storage to another. The number of characters is determined by the 2nd byte of the MVC instruction. This byte is called the____field.

• • •

length or L

10. The length code byte can represent a count of from 0 to 255. Since the number of bytes in a field is equal to the length code +1, a length code of zero would mean_______byte.

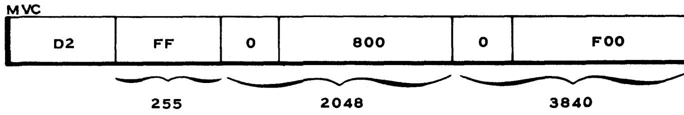
• • •

one

11. A length code of 255 in the MVC instruction would cause _____ bytes to be moved.

•••

256



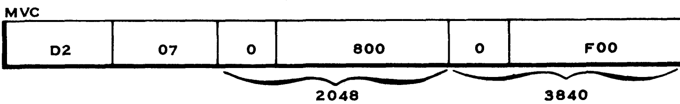
As in most System/360 operations, the 1st operand receives the results.

12. In the above MVC instruction, _____ bytes would be moved from location 3840 to location _____.

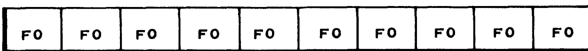
•••

256; 2048

13. Given the following, show the contents of the indicated storage area after the MVC instruction is executed. Everything is shown in hex.



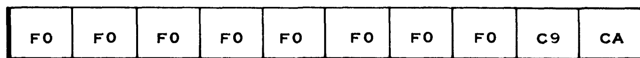
2048 (before)



3840 (before)

2048 (after)

•••



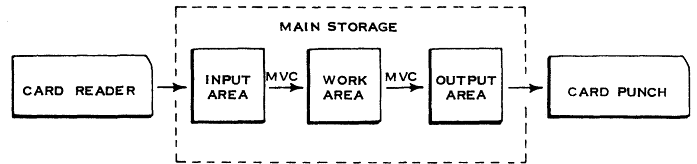
14. The bytes moved by the MVC instruction are not checked for any particular data format. Therefore they: (choose one)

    a. Must be packed decimal data.
    b. Must be in the halfword signed binary format.
    c. Can be in any format.
    d. Must be EBCDIC characters.

•••

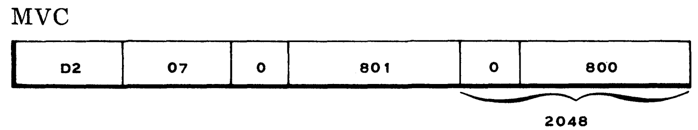
c; The bytes that are moved are not checked for coding. The title of the instruction (move characters) implies, however, that this instruction could be used to move EBCDIC characters from one area of storage to another. For instance, data could be moved from an input area to a work area without being changed. After the data has been processed in a work area, it could be moved to an output area.

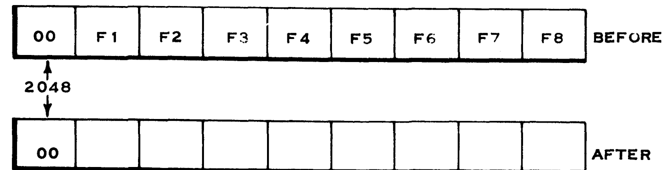
The following drawing illustrates this point.



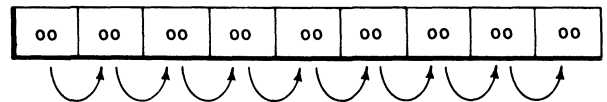
15. Because the bytes are moved <u>one at a time</u>, in a left to right direction, the MVC instruction can also be used to propagate one character throughout an area. Given the following, show the resulting storage contents.

MVC





•••



In the preceding problem, the following occurred:

1st; The instruction looks at the first location (2048), finds the 00 and moves it to 2049 where it replaces the F1.

2nd; The instruction looks at the next location (2049), finds the 00 and moves it to 2050.

3rd; The leftmost byte continues to be moved (propagated) to the right.

END OF SKIP OPTION

Now for the second machine instruction. Our output area has been cleared, and the comment shows that we want to move the first header (line) to the output area.

1. From what you know of MVC, what will be the result of the second move instruction?

●●●

The contents of a field named HDR1 will be moved into the part of the output area that we have called HEADER

But we haven't defined HDR1 yet, so we had better take care of it. Turn to Figure 11 in your sample program book.

Again we start a new coding sheet, to follow the one with our input and output area DS instructions.

2. Our first DC is for HDR1, the first header line. Since this entire constant will be moved into the area called HEADER in one operation, how long must it be? (If necessary, check the DS for HEADER.)

●●●

66 bytes (or characters)

When you first learned to define constants, you were told that you can define a constant containing as many as 120 characters (assuming no duplication factor and no length value specified). You didn't see any long constants then, but you do now.

The key to defining a constant of this size is the C in column 72. This signals the Assembler that the next source card is for a line of coding that continues to specify the same constant.

Only one continuation line is allowed for any given DC.

If you're wondering how we could define a constant 120 characters long, here is an example (remember that the Assembler coding sheet allows free form coding):

IBM IBM System/360 Assembler Coding Form

```
         DC    C'THIS°°°°°°IS°°°°°°A°°°°°°CONSTANT°°°°°°THAT°°°°°°IS°°°°°ONE°°°°°C
               °HUNDRED°°°°°°AND°°°°°°TWENTY°°°°°°CHARACTERS°°°°°°LONG'
```

Note that the characters on the continuation line must begin in column 16.

Now back to our program. Although we have been concerned only with HDR1, you should know that the author of the program wrote the DC for HDR2, also, before continuing to code the machine instructions. He did this because he wanted to set up the vertical relationship between the words, at this time.

The arrows in the following illustration show that he positioned ACCOUNT directly over NUMBER, but that he indented OLD and NEW (in HDR1) one space, compared to their counterparts in HDR2.

IBM IBM System/360 Assembler Coding Form

```
*
HDR1     DC    C'ACCOUNT     OLD              NEW         MONTHLY   MONTHLY         AMOC
               UNT           '
HDR2     DC    C'NUMBER     PRINCIPAL     PRINCIPAL     PAYMENT     INTEREST     APPC
               LIED TO PRIN'
```

This is just a matter of style: You could line up the first letter of each word in HDR1 with the first letter of the corresponding word in HDR2, or you could indent more, whatever you wish. The important consideration is the spacing of words with respect to the numerical fields which will be printed below them, later. Also (including blanks) each header line must be 66 characters long.

Now back to Figure 7. Having moved the first header line into the output area, we want to print it. Remember that there is an output routine to which we will branch, for printing, by using the operand WRITE in a branch instruction. After the output routine is over, we want to get back to the next instruction, after our branch, in this part of the program.

Remember also, that the output routine already ends with a branch to the address held in register 10.

3. We have to write a branch instruction which will also store the address of the _____ __________ in register ___.

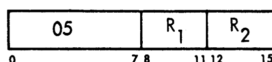
● ● ●

next instruction; 10

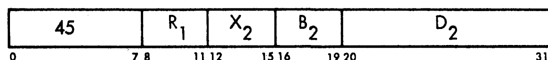
Read the following information on branch instructions. Also, locate the group of branch instructions, which include BALR and BAL, on your Reference Data card.

Branch and Link

BALR R_1, R_2 **[RR]**



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



The address of the next sequential instruction (nsi) is stored in the first operand and a branch is taken to the address stored in the second operand.

BALR (Branch and Link)
- If the second operand is 0 (zero), no branch is taken.

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

Examples:

Name (1–8)	Operation (10–14)	Operand (16–)
BEGIN	BALR	11,0
SIDEROAD	BAL	9,ROUTINE
NEXTDATA	MVC	OLD,NEW

The first of the examples shows our old friend BALR. In fact it shows the instruction that was written, in our current programming example, to store the program's base address in register 11.

If you look back at Figure 6 for a moment, and remember what you learned about the effect of the USING instruction, you can note that register 11 will contain the address of the first instruction produced by the OPEN macro.

4. In the second example printed above, BAL is used to branch to an instruction named ROUTINE. What is the instruction whose address will be stored in register 9?

● ● ●

The next sequential instruction: the MVC called NEXTDATA

5. Back to our current program, in Figure 7. What will be the effect of the BAL 10, WRITE instruction?

● ● ●

The address of the next sequential instruction (MVC HEADER, HDR2) is placed in register 10. The address of WRITE is placed in the PSW, so that the system can branch to that routine.

6. Look at the WRITE routine in Figure 7. What instruction at the end of the output routine, will cause a branch back to the address of the MVC HEADER, HDR2 instruction in the main part of the program?

• • •

The "conditional" branch BCR 15,10 which will branch to the address stored in register 10, regardless of the condition code in the PSW.

SKIP OPTION

If you doubt that you can predict the result of a BAL or BALR instruction, read the frames on the following two pages. If you feel sure about the use of these instructions, skip over those frames and we'll continue with the programming example.

BRANCHING OPERATIONS

You have previously learned how the PSW is used to control the sequence of instruction fetching and execution. You have also learned how the normal sequence of instruction fetching can be changed by (a) an interrupt, (b) the "load PSW" instruction and, (c) a "branch" instruction. You have also studied one of the "branch" instructions: "branch on condition."

1. The address of the next instruction to be fetched is contained in the _____ _____ _____.

• • •

PSW

2. After an instruction has been fetched, information in its Op code is used to update the instruction address portion of the PSW. Updating this instruction address portion of the PSW consists of increasing it by_____, _____, _____.

• • •

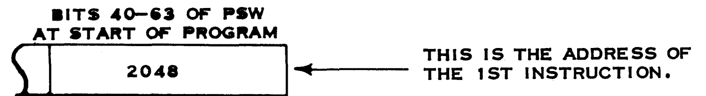
2; 4; 6

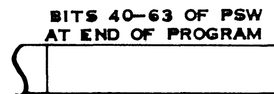
3. If the information in the "current" instruction's Op code indicates an RR format, the instruction address is increased by _____.

• • •

2

4. Given the following symbolic program, indicate (decimally) the contents of the address portion of the PSW after the program is executed. All of these instructions use the RX format and are four bytes long.





L	3,0(0,1)
AH	3,4(0,1)
MH	3,6(0,1)
S	3,8(0,1)
ST	3,12(0,1)

• • •

2068; Each instruction increased the address by 4.

5. The sequential manner of instruction fetching can be changed by means of a "branch" instruction. When a branch is taken, the address of the "branch to" location replaces _______________.

• • •

The instruction address (bits 40-63) in the PSW.

BRANCH AND LINK INSTRUCTION

1. The "branch and link" instruction can be either of the RR format or the RX format. The mnemonic used for the RR "branch and link" is ________.

•••

BALR

2. The BAL instruction always results in a branch. The BALR instruction will not result in a branch if the R2 field is _____.

•••

zero

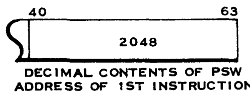
3. In all cases (even when the R2 field is zero), the address information in the PSW is stored in the register specified by the____ field.

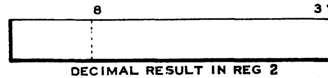
•••

R1

The address that is stored by a "branch and link" instruction is the address of the instruction that would have been executed if the branch were not taken.

4. Given the following symbolic program, show decimally what address will be put in register 2 when the BALR instruction is executed.






```
LH      0,0(0,7)
AH      0,2(0,7)
BALR    2,3
STH     0,4(0,7)
```

NOTE: Refer to System/360 Reference Data Card (X20-1703)

•••

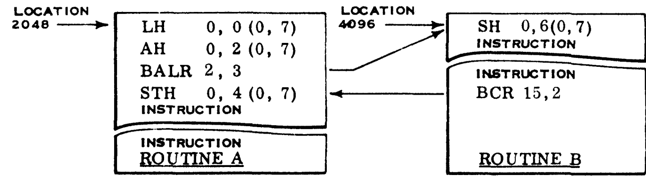
Reg 2 = 2058

5. Write the mnemonic of the instruction whose address was stored in register 2 in the previous problem___.

•••

STH

The reason for storing the address of the next sequential instruction during a "branch and link" operation is to provide a linkage between routines. This is illustrated as follows:



As you can see above, the "branch and link" instruction will:

- Cause the address of the STH instruction of routine A to be stored in register 2.
- A branch will be taken to the SH instruction in routine B (assumed as the contents of register 3).
- The last instruction in routine B is an unconditional branch (because the mask field contains 15) back to the STH instruction in routine A.
- The address of the STH instruction was obtained from register 2 where it was stored from the preceding BALR instruction.

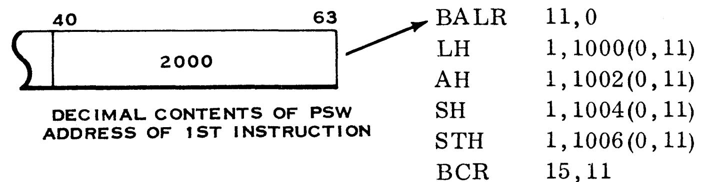
6. BALR 1,0

   In the above example, the BALR instruction will cause the address information in the PSW to be stored in register_____, and a branch ________ (is/is not) taken.

•••

1; is not

The BALR instruction, with an R2 field of zero, may be used to load a base address into a general register. Examine the following program; then read the following frames.



7. Assuming that 2000 is the address of the BALR instruction, what address will be placed in register 11?_______

•••

2002

8. Because the R2 field of the BALR instruction is zero, a branch ______(will/will not) be taken.

• • •

will not

9. The second operand of the LH instruction will have a base address of_____ and a displacement of_____.

• • •

2002; 1000

10. The second operands of the second through the fifth instructions will all have a base address of______.

• • •

2002

11. The last instruction (BCR) will cause an "unconditional branch to" location ______.

• • •

2002

12. Write the mnemonic of the instruction at location 2002. ______

• • •

LH; Actually this program will never end because of the "unconditional branch back to" the LH instruction.

13. Thus far you have seen two main uses for the "branch and link" instruction. It can be used to:

    1. Branch to some routine and automatically provide the programmer with an ______ to branch back to. Hence, the name of the instruction: "branch and link."
    2. Provide the programmer with a way to load his initial base address into a g______ r______.

• • •

address; general register

END OF SKIP OPTION

1. Now we continue with programming example 1. We have printed the first header line, so now we need to ____.

• • •

Move the second header line into the output area

2. Which instruction accomplishes this?

• • •

MVC HEADER, HDR2

3. What happens and what MVC instruction does the system branch back to, as a result of executing the second BAL 10, WRITE?

• • •

The address of the next sequential instruction is stored in register 10, the system branches to the WRITE routine, prints the second header line, and branches back to the address of MVC OUTPUT, OUTPUT-1.

4. Now we want to clear the words of the second header line from the output area, to ready it for our numerical output fields. How does the MVC instruction accomplish this?

• • •

It propagates a blank in byte after byte until the entire output area contains blanks.

5. We see a comment on the coding form which tells us that we will read the transaction cards. For each line of output, how many cards will be read?

• • •

One

6. How does the BAL instruction accomplish this?

• • •

It places the address of the next instruction, PACK, in register 10 and causes a branch to the first instruction of the routine called READ. After one card is read, the system branches back to the PACK instruction.

FORMAT OF INPUT DATA

1. Let's see if you can recall what you read about input operations and data format in System·Review: Numerical data, read in from a punched card, is in the _________ _____________ format.

•••

zoned decimal (or unpacked)

2. When the computer performs an arithmetic operation, however, we want to save time and storage space; we _________ the data before beginning calculations.

•••

pack

3. Let's look ahead a bit: Since we are going to use standard instructions, not decimal, to perform the calculations in this first program, we must both pack the data and convert it to _________.

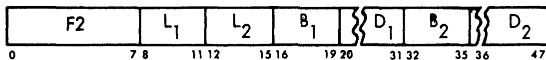
•••

binary

Read the following description of the PACK instruction, and locate it on your Reference Data card.

Pack

PACK $D_1(L_1, B_1), D_2(L_2, B_2)$ **[SS]**



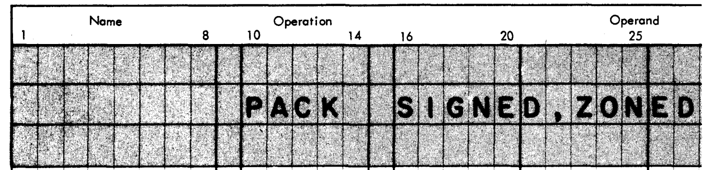
The signed or unsigned number in the zoned format at the second operand location is changed to packed format and stored in the first operand location.

- The fields are processed one byte at a time, from right to left. They are not checked for valid sign or digit combinations.
- If the first operand field is too long, it will be filled with high order zeros.
- If the first operand field is too short, any remaining high order digits in the second operand will be ignored.
- The maximum size of the second operand (zoned field) is 16 bytes.

Condition Code:
The code remains unchanged.

Program Interruptions:
Protection
Addressing

EXAMPLE:



This instruction places the contents of a storage area called "ZONED" into a storage area called "SIGNED". The result in SIGNED is in packed format and the rightmost 4 bits represent the sign of the low order digit. The sign of the low order digit in the zoned field will be given to the packed field. An unsigned zoned field (containing hex Fs in the zone portions) will be assumed to be positive.

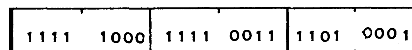
Refer to the description of the instruction, if necessary while answering the following question about the example:

4. Would ZONED be cleared, after execution of PACK, or would the same data still be there?

•••

The description says nothing about clearing: The same data would still be there.

5. Here is a zoned decimal field:



1. Its sign is___________ .
2. In order to pack it, we need to specify an area of storage with at least __ bytes.

•••

minus (1101); 2

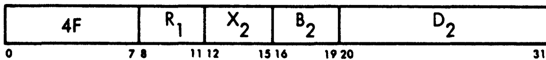
This is a good point to show you the convert to binary instruction also (CVB), since we will use it with our packed data.

You will remember that unless we are using instructions from the decimal set, we cannot perform arithmetic operations on packed data. We must convert packed data to binary, since we are using arithmetic instructions from the standard set.

Locate it on your Reference Data card, then read the following information.

Convert to Binary

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



A doubleword of packed decimal data at the second operand location is converted to a 31 bit binary number and sign in the register specified by the first operand.

- The second operand must be on a doubleword integral boundary.
- The data is right aligned and signed in both locations.
- The maximum number that may be converted is +2,147,483,647.
- The minimum number that may be converted is -2,147,483,648.
- Exceeding these values will result in a fixed point divide exception and a program interrupt follows.
- During execution, the packed data is checked for valid sign and digit codes. An invalid code causes a data exception.

Condition Code:
 The code remains unchanged.
Program Interruptions:
 Addressing
 Specification
 Data
 Fixed point divide

EXAMPLE:

Name	Operation	Operand
	CVB	3,PDEC

6. Use the descriptive information to answer the following questions about the example:
   1. Briefly, what does CVB 3,PDEC do?
   2. What must be true of the address of PDEC?

• • •

1. It changes the format of the data in an area called PDEC to binary, and puts it in register 3.
2. It must be on a doubleword boundary.

7. Look at the first PACK instruction in Figure 7. The area in storage into which we will pack PRIN (the Principal amount) is called __________.

• • •

PPRIN

Now look at Figure 11.

8. We specify the storage area called PPRIN with a DS D. Will the doubleword so specified be large enough to hold PRIN after it is packed?

• • •

Yes: Doubleword=8 bytes. Prin=7 bytes; when packed, it requires 4 bytes (7 digits and a sign).

In fact, a doubleword is twice as long as necessary; you may be wondering why we used a DS D.

Look at the other PACK instructions in Figure 7. Note that RATE and PAY will also be packed, so that they can be converted to binary for the calculations.

9. Now think back to the information on defining storage, earlier in this volume: The DS D instruction defines a doubleword, and it also steps the location counter so that the doubleword is aligned (where?) __________.

• • •

On an integral doubleword boundary.

10. If necessary, look back, in this text, to the note below the example of a Convert to Binary instruction. Then use your own words to tell why we used DS D instructions to specify PPRIN, PRATE, and PPAY, even though doublewords are much larger than necessary for the packed fields.

• • •

DS D automatically starts each of these fields on an integral doubleword boundary. This is where each of these fields must be located, if we are to convert them to binary.

11. Figure 7 shows that we continue, after the PACK instructions, by converting ________ and ________ to binary.

• • •

The contents of PPRIN.
The contents of PRATE.

12. Where will the binary equivalents of these two fields be placed?

• • •

PPRIN into general register 3.
PRATE into general register 7.

13. When PRIN was packed, the high order 4 bytes of the doubleword PPRIN were filled with ________.

• • •

zeros

14. Which bytes from PPRIN are converted to binary and placed in general register 3 as a result of the CVB instruction?

• • •

The low order (rightmost) 4 bytes

15. What happens to the zeros in the high order 4 bytes of PPRIN?

• • •

They are checked and, since they are not significant, they are ignored.

SKIP OPTION

If you doubt that you can state the reasons for, and the results of, the PACK and CVB instructions, read the frames on the following five pages.

Otherwise, skip to the point where we resume discussion of our first sample program.

CONVERTING DATA TO BINARY

You have demonstrated a knowledge of binary data formats. You know that positive numbers are represented in true form and that negative numbers are represented in complement form. You also know that these binary numbers appear in main storage as halfwords or fullwords. You are probably wondering, however, how data from a punched card gets into main storage in these binary formats. You should know the standard card code (hollerith). So let's start at that point.

1. To punch the decimal number 1234 in an IBM card would require______ columns.

• • •

four

2. Each column of a card read into a System/360 usually occupies one b_____ of main storage.

• • •

byte

3. Data from a card reader is usually represented in main storage in the Extended ______ _______ ______ Interchange Code.

• • •

Binary Coded Decimal (BCD)

4. The Extended Binary Coded Decimal Interchange Code is usually called _______. The code uses ____ bits to represent a card column.

• • •

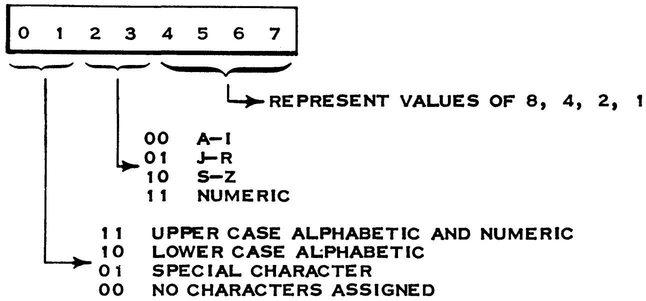
EBCDIC; 8

5. The 8 bits of EBCDIC have two parts: zone and numeric. The zone part consists of bits ________ and the numeric part consists of bits _________ .

• • •

0-3; 4-7

Bit positions in the EBCDIC byte



6. As can be seen on the EBCDIC chart on your Reference Data card, a numeric "1" punch would be represented by a combination of 8 bits in EBCDIC as ____________.

• • •

11110001

7. The letter A (12 and 1 hole punches) would be represented as __________.

• • •

11000001

8. The character "J" is represented on a card by an ____ zone punch and a __ digit punch. It is represented in main storage as the following byte: ______________ .

• • •

11; 1; 11010001

9. A lower case "j" would be represented in a card by a digit punch of 1 and zone punches of____ and ____. It would be represented in storage as __________.

• • •

12; 11; 10010001

10. The special character % would be represented by a ____ zone punch and digit punches of ___ and ____. It would be represented in storage as __________.

• • •

0; 8; 4; 01101100

11. To get the bit combination 01101100 into storage would require a zone punch of___ and digit punches of ___ and ____.

• • •

0; 8; 4

12. A blank column on a card ("no punches" under Punched Card Code) would be represented in storage as ____________ .

• • •

01000000

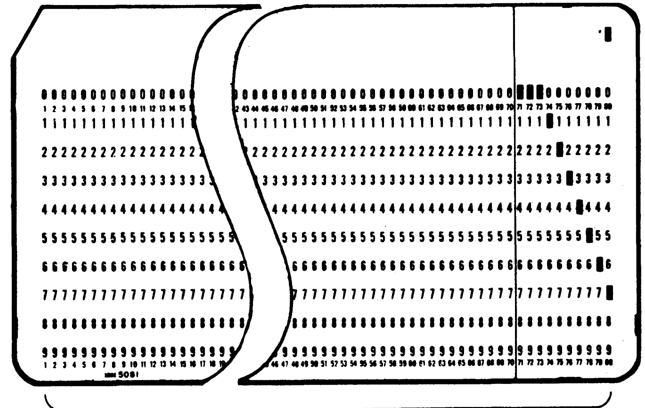
13. To get a bit combination of 00000000 would require zone punches of ____ , ____ and digit punches of ____, ____, ____.

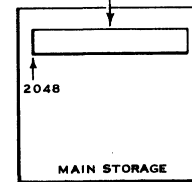
• • •

12; 0; 9; 8; 1

Usually cards are punched in the standard hollerith card code. That is, only decimal and alphabetic information is punched in the card. Then after the data is brought into storage, it can be converted (via instructions) to binary and processed with the fixed point instructions.

14. Given the following card record:





The card contains + 0001234567 in columns 71-80. Assuming that the entire card record has been read into main storage starting at location 2048, columns 71-80 will be in byte locations __________ through ________.

• • •

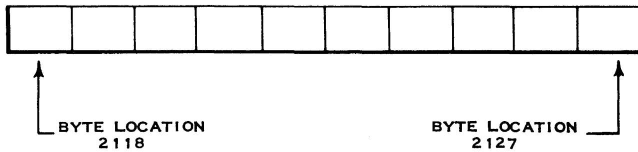
2118, 2127; locations 2048-2117 would contain card columns 1-70

15. Numeric fields in EBCDIC are said to be in the __________ (zoned/packed) decimal format.

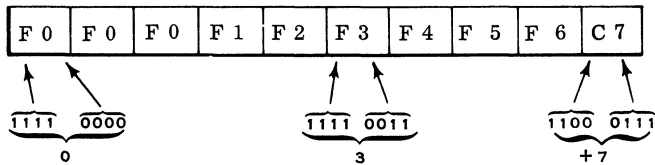
•••

zoned

16. Show (in hex) the zoned decimal data from columns 71-80 of the card.



•••



17. The sign of a zoned decimal data field is in bits ___ of the _____ (low/high) order byte.

•••

0-3; low

PACK INSTRUCTION

1. Decimal data must be in the packed format before it can be converted to binary. Zoned decimal fields can be changed to the packed decimal format by an instruction called "_______".

•••

"pack"

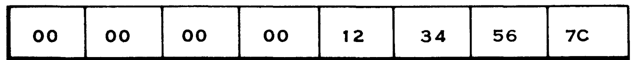
2. Packed decimal data consists of two______ per byte with the low-order byte containing one digit and the______. The sign of a packed decimal field is in bits_____ of the low-order byte.

•••

digits; sign; 4-7

3. Show (in hex) how the data in columns 71-80 would look if it were packed into eight bytes.

•••



4. The instruction "pack" is of the SS format. Label the fields.

•••

5. Reread the description of the "pack" instruction. In the "pack" instruction, the 2nd operand contains the ______ (packed/zoned) decimal data.

•••

zoned

6. The low-order byte of the 1st operand receives the low-order byte from the zoned data field. The zone bits of this byte are________________________ (assumed to be the sign/ignored).

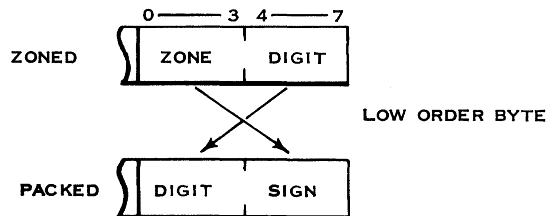
•••

assumed to be the sign

7. The zone and digits bits from the low-order byte of the zoned decimal field are__________ before being placed in the 1st operand.

•••

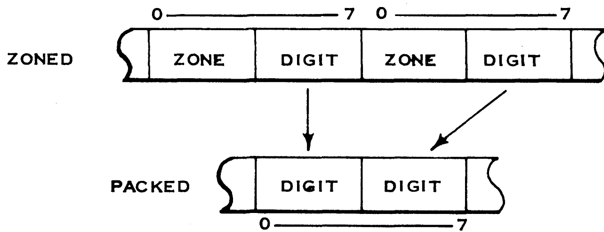
reversed or swapped as shown below



8. Each remaining byte of the 1st operand receives the______(zone/digit) bits from two successive bytes of the zoned decimal field.

• • •

digit; As shown below



9. The zone bits from the 2nd operand in the preceding example are_______.

• • •

ignored

10. The bytes from the zoned decimal field (2nd operand)_______(are/are not) checked for valid sign or digit combinations.

• • •

are not

11. The zoned decimal field (2nd operand) and the resulting packed decimal field (1st operand) ______(can/cannot) be of different lengths.

• • •

can

12. The 2nd byte of the "pack" instruction contains the _________ codes of the two operands. The number in the length code is _____________(equal to/one less than) the number of bytes in the operand.

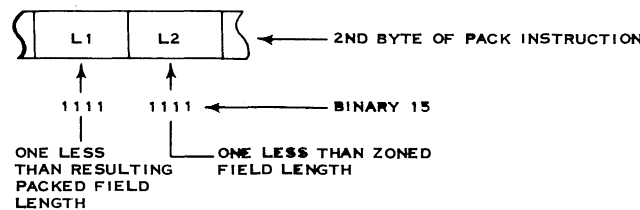
• • •

length; one less than

13. The maximum number of bytes in either operand of a "pack" instruction is_______.

• • •

16; As shown below



14. If the length codes are such that the 1st operand is long (compared to the 2nd operand), the packed decimal field will be extended with high-order _______________.

• • •

zeroes

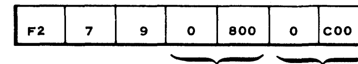
15. If the length codes are such that the 1st operand cannot contain all the digits from the zoned field, the remaining digits are _____________.

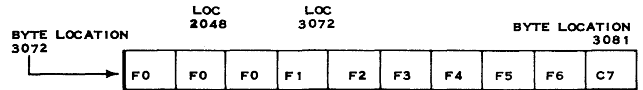
• • •

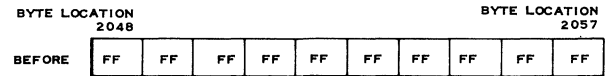
ignored

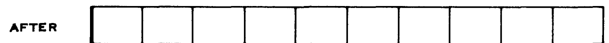
16. Given the following "pack" instruction, show the resulting packed decimal field. Instructions and data are shown in hex.

PACK

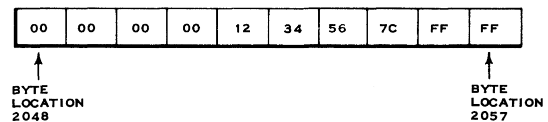








• • •



17. Just as with the previous instructions dealing with fixed length operands, the operands of the "pack" instruction are addressed by their __________ (high/low) order byte location.

• • •

high

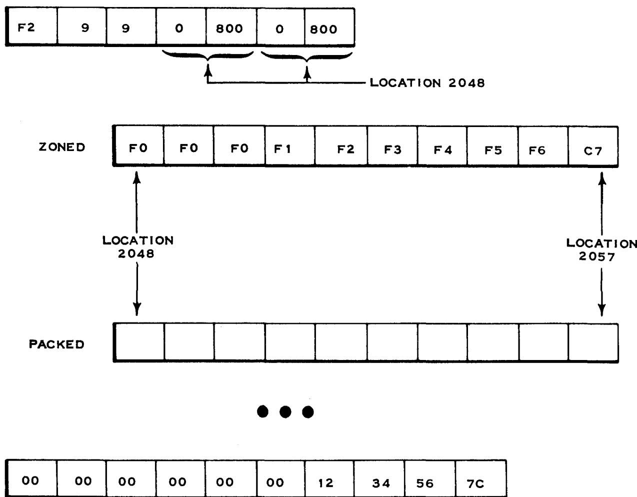
18. The address of the low-order byte of either operand in the "pack" instruction can be determined by adding its_______code to its generated effective address.

• • •

length

19. Given the following "pack" instruction, show the resulting packed decimal field. Everything is shown in hex.

PACK



Notice that the original zoned data field was used to contain the resulting packed decimal field.

CONVERT TO BINARY INSTRUCTION

You have now seen how the "pack" instruction can change a zoned decimal field to a packed decimal field. The packed decimal data can now be changed to a word of binary data by use of the instruction: "convert to binary." This instruction will not only convert the data to binary, it will also load it into a general register. Reread the description of the "convert to binary" instruction.

1. In the CVB ("convert to binary") instruction, the 2nd operand contains a ________________ (zoned decimal/packed decimal/binary) data field.

•••

packed decimal

2. To use the CVB instruction, the packed decimal field must consist of _______ bytes. The specified address of the high-order byte must be divisible by _____ or a _______________ exception will occur.

•••

eight; 8; specification

3. The results of the CVB instruction will be a binary word and will be loaded into a _______ _________.

•••

general register

4. The data in the packed decimal field is checked for valid sign and digit codes. If any codes are improper, a _______ exception will be recognized.

•••

data

5. 0000-1001 are valid digit codes. If any of the digits of the packed decimal field are coded from 1010-1111, a ________ exception will be recognized.

•••

data

6. Valid sign codes are 1010-1111. If the sign of the packed decimal field (low-order four bits) contains any of the valid digit codes, a _______ exception will be recognized.

•••

data

7. Since a twelve hole punch is used to indicate a plus field on a card, the usual EBCDIC plus sign will be_______. (Refer to the EBCDIC chart on your card opposite 12-0, 12-1, 12-2, 12-3, etc.).

•••

1100; These are the zone bits for the letters A-I

8. Since an eleven hole punch is used to indicate a minus field on a card, the usual EBCDIC minus sign will be_________.

•••

1101; These are the zone bits for the letters J-R

9. Sometimes plus fields in a card do not have a twelve hole punch. In these cases, the expected EBCDIC plus sign would be________.

•••

1111; These are the zone bits for the numbers 0-9

10. Either 1101 or 1011 are acceptable as minus signs. All other bit combinations of 1010-1111 are acceptable as_________signs.

•••

plus

11. If the sign of the packed decimal field is plus, the binary equivalent of the field will be loaded into a register in ________ (true/complement) form.

• • •

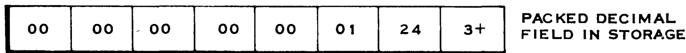
true

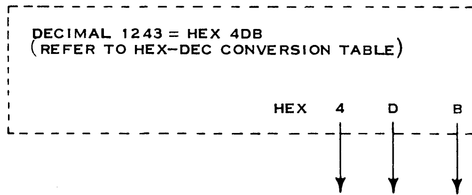
12. If the sign of the packed decimal field is minus, the binary equivalent of the field will be loaded into a register in __________ (true/complement) form.

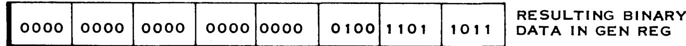
• • •

complement

Example of conversion from packed decimal format to binary format.

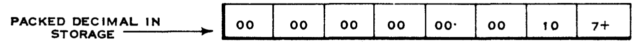




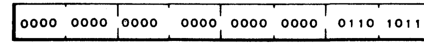


In the preceding example, the conversion was made by first changing the decimal data to hex data and then the hex data to binary data. We go through the hex step simply because it makes decimal to binary conversion easier. Of course, the System/360 does not go through this hex step. It converts directly from decimal to binary.

13. Given the following packed decimal field, describe the converted results in binary bits in the general register.



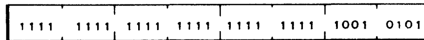
• • •



The binary value of 107 is loaded, and the high order bit is set to 0 to denote a positive number.

14. Given the following packed decimal field, describe the binary results in the general register.

• • •



The -107 is loaded as the complement of the value 107, and the high order bit is set to 1, to denote a negative number.

END OF SKIP OPTION

1. We have almost arrived at the point, in our first programming example, where we can perform the first arithmetic operation. According to the flowchart in Figure 5, this will be the calculation of ________ _________ .

●●●

monthly interest

2. According to the comment alongside the block for this calculation, we will first ________ Principal and Rate, then we will ______ the result by 12.

●●●

multiply; divide

We are going to arrange for this first calculation in an unusual way, and you deserve a note of explanation.

Assembler language instructions can be grouped by type of function: For example, your Reference Data card shows eight different branch instructions, in the Standard Set, but they all have the same basic function - altering the sequence of execution of instructions in a program. Similarly, there are seven different compare instructions, but they all share the function of setting a condition code.

In determining the content of the sample programs for this course, the authors discarded two unworkable approaches:

- Trying to design one data processing problem (or two, or three) which would demonstrate all of the instructions.
- Designing a large number of small routines - with two or three instructions in each - to encompass the entire instruction set.

Both of these approaches were discarded for the same reason: The sample programs would not represent anything like the real-world of data processing with which you will be dealing.

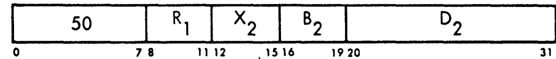
We decided to design sample programs which, if they did not use every instruction, would use at least one instruction of each type. Through learning to use, for example, one of the compare instructions, you will have learned most of what you need for coding the others. The remainder of what you need to learn can be found in the Appendix to this course.

We started with realistic data processing problems and found, when we had the first three coded, that we had used every type of instruction but one: STORE. Accordingly, we found a way to include it in the first sample program, so you would have a chance to see how this type of instruction works.

Read the following information, and locate the group of store instructions on your Reference Data card.

Store

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



32 bits from the general register specified in the first operand are stored at the second operand fullword location.

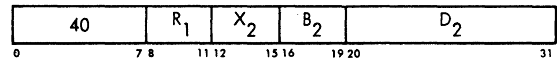
- The second operand must be on fullword integral boundary.

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

Name (1–8)	Operation (10–14)	Operand (16–)
	ST	6,NEWLOC

Store Halfword

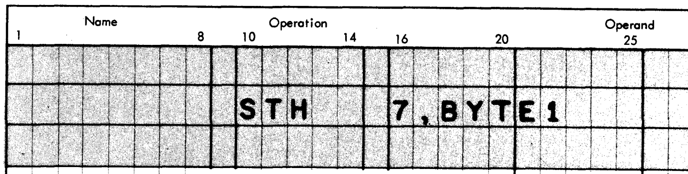
STH $R_1, D_2(X_2, B_2)$ [RX]



16 low order bits (bits 16-31) from the general register specified in the first operand are stored at the second operand halfword location.

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

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



Note: The four store type instructions are unusual in that the direction of activity is from the first operand to the second.

You should make a note of this, on your Reference Data card, by showing an arrow ⟶ to the left of the TYPE (RX, or RS) designation for each of these instructions.

We thought we might as well show you two of the store instructions, since they differ, mainly, only in the amount of data stored. The other point which we need to note is that ST (which might be called "Store Fullword") requires a storage location that begins on an integral fullword boundary.

3. In our sample program, we are going to store the binary equivalent of RATE (which is in general register 7) in an area called BRATE. How do you know, from Figure 11, that BRATE is aligned on the proper boundary?

●●●

It was specified by a DS F, which both reserves a fullword of storage and aligns it on a fullword boundary.

SKIP OPTION

If you doubt that you can predict the results of a store instruction, read the following frames. Otherwise, skip to the next page.

STORE INSTRUCTIONS

Besides the ability to put data into the registers, System/360 also needs the ability to put data from the registers back into main storage.

1. The last type of operation is accomplished by a "________" instruction.

●●●

"store"

Mnemonic	Hex Op Code	Data Flow
ST	50	Fullword, register to storage
STH	40	Halfword, low-order of register to storage

2. You have learned that, as a general rule, most instructions cause the results to replace the ______ (1st/2nd) operand. The "store" instructions are an exception to the preceding rule. In the ST and STH instructions, the ______ (1st/2nd) operand replaces the ____ (1st/2nd) operand.

●●●

1st; 1st; 2nd

3. In the case of the STH instruction, the 2nd operand in main storage is replaced by bits ____ through ____ of the general register.

●●●

16; 31

4. The ST and STH instructions ______ __________ (change/do not affect) the condition code.

●●●

do not affect

5. Write the instruction that will store the contents of general register 7 in a location called FULLWORD.

●●●

ST 7, FULLWORD

END OF SKIP OPTION

The group of store type instructions is a small one. The only ones remaining are:

Name of Instruction	Mnemonic	Basic Function
Store Character	STC	Places the low-order byte, from a specified register, into a specified storage location.
Store Multiple Register	STM	Places the contents of a specified series of registers into a specified storage location.

Self-Study material on these instructions can be found in the Appendix to this course.

1. Figure 8 shows the calculation steps in the program. The first instruction is the store that we've been talking about. The next operation that we must perform is the calculation of annual interest. We will do that, by multiplying __________ by________.

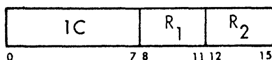
• • •

Principal; Rate

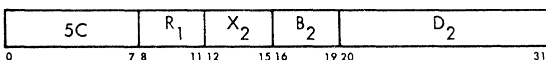
Read the following descriptions of Multiply and Multiply Halfword. Also, locate them on your Reference Data card.

Multiply

MR R₁, R₂ **[RR]**



M R₁, D₂(X₂, B₂) **[RX]**



M and MR:
The first operand (multiplicand) is multiplied by the second operand (multiplier) and the product replaces the multiplicand.

- The multiplier and the multiplicand are 32 bit signed integers.
- The resulting product is a 64 bit signed integer.
- The first operand must refer to the even register of an even-odd register pair.
- The multiplicand is right aligned in the odd register of the even-odd pair.
- The product is right aligned in the even-odd register pair.

MR only:
- The even register may contain the multiplier.

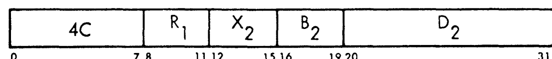
M only:
- The multiplier (second operand) is a fullword right aligned signed integer.
- The multiplier must be on a fullword integral boundary.
- The even register is used only in developing the product.

Condition Code:
The code remains unchanged.
Program Interruptions:
Addressing (M only)
Specification

Name (1–8)	Operation (10–14)	Operand (16–25)
	MR	2,5
	M	4,NUMBER

Multiply Halfword

MH R₁, D₂(X₂, B₂) **[RX]**



The first operand (multiplicand) is multiplied by the halfword second operand (multiplier) and the product replaces the multiplicand.

- The multiplicand is a 32 bit signed integer.
- The multiplier is a 16 bit signed integer.
- The resulting product is a 32 bit signed integer.
- The multiplicand (first operand) may be any register. The product is developed in that register.
- The multiplier (second operand) is a halfword right aligned signed integer.
- The multiplier must be on a halfword integral boundary.

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

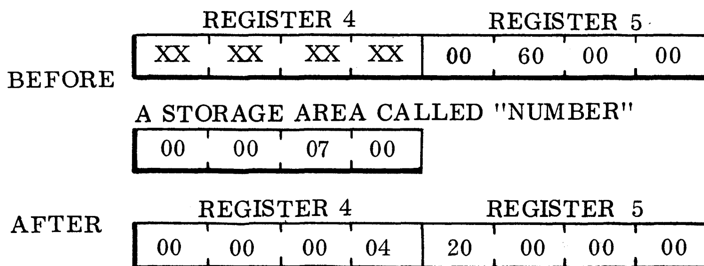
Name	Operation	Operand
	MH	9,HALF

There are two points about the M instruction that need to be emphasized:

- The product of the multiplication operation is placed in our even-odd register pair. The even numbered register of this pair is R_1 (the first operand) in the instruction, but the multiplicand must be in the odd numbered register.
- The multiplier (the second operand in the instruction) must be aligned on a fullword boundary.

Thus, in the example M 4,NUMBER, the contents of register 5 will be multiplied by the contents of a field called NUMBER, and the product will be placed in registers 4 and 5.

Picture it this way (fields are shown in decimal):



We don't care what was in register 4 before the operation, because all of the positions to the left of the first significant digit (in the product) will be filled with zeros.

Suppose that register 9 contains the binary equivalent of 555, and a fullword named MULT contains 2.

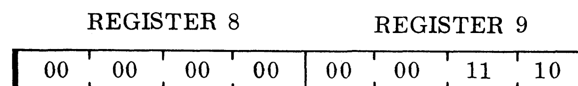
2. What would you write if you wanted to multiply them?

•••

M 8,MULT

3. What would be the contents of the even-odd register pair, 8 and 9, after the operation? Show it in decimal.

•••



Now back to our sample program. Remember that the contents of register 3 are the binary equivalent of PRIN, a 7 digit number XXXXX.XX with the decimal point implied; BRATE contains the binary equivalent of RATE, a four digit number .XXXX with the decimal point implied.

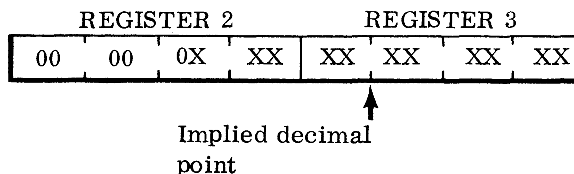
4. Which registers will hold the product of M 2,BRATE?

•••

Registers 2 and 3

5. Using Xs, show their contents (as if they were decimal digits) and show where the implied decimal point would be.

•••



SKIP OPTION

If you doubt that you can predict the results of a multiply operation, read the frames on the following three pages. Otherwise, skip to the point where we resume discussion of our sample program.

MULTIPLY INSTRUCTIONS

1. The "multiply halfword" instruction, like all instructions involving halfwords in main storage, has a mnemonic which ends with the letter _____.

• • •

H

2. In the MH instruction, the multiplicand is in a general register while a halfword in main storage is the ___________. The halfword from storage is expanded to a ________ before the multiplication.

• • •

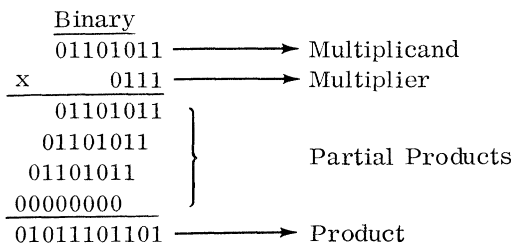
multiplier; fullword

3. In the MH instruction, the multiplicand is _____ bits long.

• • •

32; The entire register is multiplied by the multiplier. Normally, the register will only be holding a halfword and therefore the register's 16 high-order positions will not affect the product. The net result is that a halfword will be multiplied by a halfword.

Binary multiplication can be quite lengthy if done by hand. The following is an example of an 8-bit multiplicand being multiplied by a 4-bit multiplier.



Judging from the preceding example you can see that binary multiplication is quite lengthy. If it is necessary to determine the results of a "multiply" instruction, you should convert the numbers to decimal and then multiply.

4. The MH instruction follows the rules of algebra. That is, if both operands are plus the product will be a ____________ (positive/negative) number.

• • •

positive

5. If both operands are negative, the product will again be a ____________ (positive/negative) number.

• • •

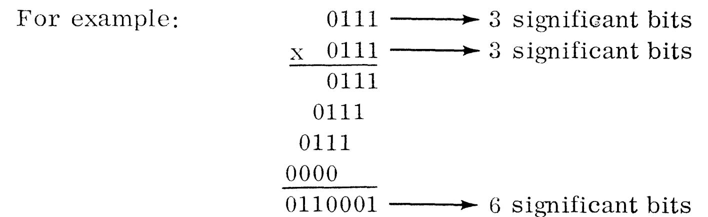
positive; multiplication of like signs always results in a positive answer. For example: (+2) x (+7) = +14; (-2) x (-7) = +14

6. If multiplication of like signs results in a positive product, multiplication of unlike signs should result in a ____________ (positive/negative) product.

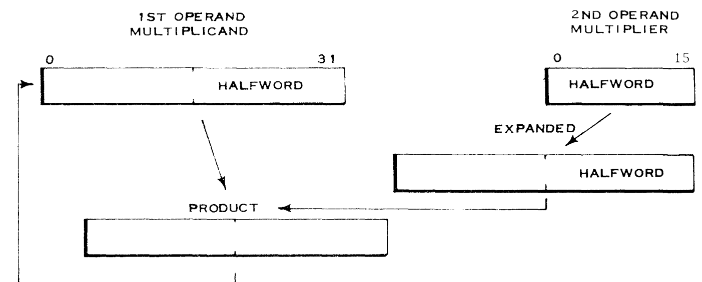
• • •

negative

Another rule of multiplication is that the maximum number of significant bits in the product is equal to the total number of significant bits in multiplicand and multiplier.



MH



7. The preceding example shows that the MH instruction is normally used to multiply one h_______ (1st operand) by another h__________(2nd operand.) The maximum product that could result would be a f__________and would replace the contents of the 1st operand r____________.

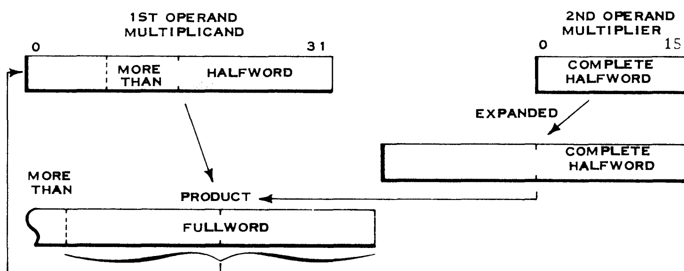
● ● ●

halfword; halfword; fullword; register

8. If the MH instruction is used with a multiplicand that has 15 significant bits and a multiplier that has 15 significant bits, the product would contain _______ significant bits and would replace the m__________.

● ● ●

30; multiplicand



9. The example above shows that the 1st operand register contains more than a _____________. The 2nd operand contains a complete (16 significant bits) ________________.

● ● ●

halfword; halfword

10. If the MH (multiply halfword) instruction is used, the product will be more than ______ bits long. The entire product will not fit in the 1st operand r______.

● ● ●

32; register

11. In the preceding example, the resulting product in the 1st operand register contained only the _____ low-order bits of the actual product. The high-order bits of the actual product were _________.

● ● ●

32; lost

12. The preceding example ________(is/is not) a normal application of the MH instruction.

● ● ●

is not

13. The product of the 32-bit multiplicand and the 16-bit multiplier may exceed 32 bits but only the low-order ______ bits of the product replace the 1st operand.

● ● ●

32

14. Although the register containing the 1st operand may not contain the entire product, a fixed point overflow will not occur and the condition code remains _____.

● ● ●

unchanged.

In summary, the MH instruction multiplies the contents of a general register by a halfword from main storage. The low-order 32 bits of the product replace the multiplicand. No fixed point overflow is possible and the condition code remains unchanged.

15. The mnemonic MR denotes a multiply instruction of the___ ___ format. The instructions MH, M, and MR cause the _______(1st/2nd) operand to be multiplied by the______(1st/2nd) operand. The product of the MH, MR, or M instruction replaces the_______(1st/2nd) operand.

● ● ●

RR; 1st; 2nd; 1st

16. The R1 field in both the M and MR instructions must contain the address of an____(even/odd) numbered register. If the R1 field of an M or MR instruction has an odd address, a program interrupt will be caused by a ___________ exception.

● ● ●

even; specification

17. A specification exception of an M instruction can also be caused by a 2nd operand address that is not divisible by ________.

● ● ●

4; The 2nd operand must be a fullword in storage.

18. Although the R1 field contains the address of an even-numbered register, the 1st operand (multiplicand) is actually in an _____ (even/odd) numbered register.

•••

odd

19. If the R1 field of an M instruction contains a 4, the contents of register 4 are ignored and the multiplicand is brought out of register ______.

•••

5

20. MR

In the above MR instruction, the multiplicand is in register ____ and the multiplier in register ____.

•••

5; 7

21. MR

In the above MR instruction, the multiplicand is in register _____ and the multiplier is in register _____.

•••

5; 4

22. In the preceding MR instruction, both the multiplicand and the multiplier were wiped out by the product which is placed in register ____ and ____.

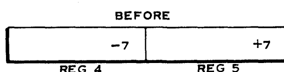
•••

4; 5

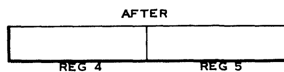
23. Show the register contents (expressed decimally) after the following MR instruction is executed.

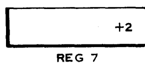
MR

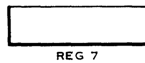
BEFORE



AFTER



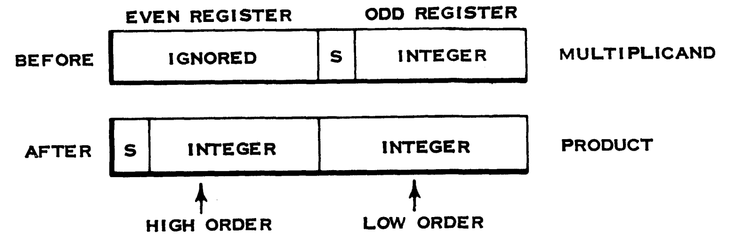




•••

Reg 4 = Zero; Reg 5 = +14; Reg 7 = +2

Notice that in the preceding instruction, register 4 was zeroed out even though the product was small enough to be fitted into reg 5.



24. The example above shows that the product of an MR or M instruction is always developed as a doubleword with the high-order in the ____ register and the low-order in the _____ register.

•••

even; odd

END OF SKIP OPTION

1. Now back to our program.
   At the end of the multiplication step, you noted that registers 2 and 3 contained yearly interest. According to the flowchart, what will we do next ?

•••

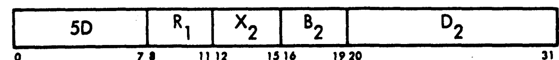
divide by 12

Read the following information about the divide instruction, and locate it on your Reference Data card.

Divide

DR R_1, R_2 [RR]

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



D and DR:
The dividend (first operand) is divided by the divisor (second operand) and the quotient and the remainder replace the dividend.

- The dividend (first operand) is a 64 bit signed integer.
- The dividend occupies an even-odd pair of general registers.

- The first operand refers to the even register of an even-odd pair.
- The divisor (second operand) is a 32 bit signed integer.
- All factors are right aligned signed integers.
- After the division, the quotient is in the odd register of the even-odd pair as a 32 bit signed integer.
- The remainder is in the even register of the even-odd pair as a 32 bit signed integer.

D only:
- The divisor (second operand) is a fullword right aligned signed integer.
- The divisor must be on a fullword integral boundary.

Condition Code:
    The code remains unchanged.
Program Interruption:
    Addressing (D only)
    Specification
    Fixed-point divide

Now we have everything set for our calculation of monthly interest.

At this point, we will introduce a special sort of self-defining value that you haven't seen before: It is called a "literal".

You specify it, in an operand, by using an equal sign, a type symbol (for halfword, fullword, or doubleword) and the value that you want (in quotes). For example: = H '100' would specify a halfword binary operand with a value of 100, and would place it on an integral halfword boundary. This has the same effect as if you had defined a constant and labelled it, but this way saves time and eliminates one label.

1. What would = F '50' specify?

●●●

A fullword binary operand, with a value of 50, starting on an integral fullword boundary.

From the preceding, answer the following questions about the divide instruction D 2,=F'12' in Figure 8.

2. What is the dividend?

●●●

The dividend is the 64 bit signed binary integer that represents yearly interest.
It is in registers 2 and 3.

3. What is the divisor?

●●●

The divisor is a fullword binary number with a value of 12.

4. What is the form of the quotient, what does it represent, and where is it after the instruction has been carried out?

●●●

The quotient is a 32 bit signed integer (a fullword) which represents monthly interest. It is in register 3.

5. How many implied decimal positions does it have?

●●●

6, same as yearly interest.

SKIP OPTION

If you doubt that you can predict the result of a divide operation, read the frames on the following two pages. Otherwise, skip to the point where we resume discussion of our sample program.

DIVIDE INSTRUCTIONS

Let's consider the instructions that will <u>divide</u> fixed length binary numbers. But first, let's review some information concerning division.

```
      120
12 ) 1440
```

1. The problem above shows a division of decimal numbers. The number 12 is called the __________ and the number 1440 is the __________ . The answer is called the __________ .

●●●

divisor; dividend; quotient

```
12 ) 1443
```

2. The divide problem above has a __________ of 120 and a __________ of 3.

●●●

quotient; remainder

3. The sign of the quotient follows the rules of algebra. If both the divisor and dividend have plus signs, the quotient will also have a ________ sign.

• • •

plus

4. If both the divisor and dividend have negative signs, the quotient will have a ________ sign.

• • •

plus

To illustrate the preceding rules, consider the following:

$$-12 \overline{\smash{)}\,-144}^{\;+12}$$

To check, multiply the quotient and divisor.

$$+12 \times -12 = -144$$

5. If the divisor and dividend have opposite signs, the quotient will have a ________ sign.

• • •

minus

To illustrate the preceding rule, consider the following:

$$-12 \overline{\smash{)}\,+144}^{\;-12}$$

To check, multiply the quotient and divisor.

$$-12 \times -12 = +144$$

6. What about the sign of the remainder? By definition, the remainder is what is left from the dividend. As a result, the sign of the remainder should be ____________ (the same as/different from) that of the dividend.

• • •

the same as

To illustrate the preceding rule, consider the following:

$$-12 \overline{\smash{)}\,-1443}^{\;+120} \quad \text{with a remainder of } -3$$

To check the above, multiply the quotient and divisor and add the remainder.

$$-12 \times +120 = -1440$$
$$-1440 + (-3) = -1443$$

7. Show the quotient and remainder.

$$-12 \overline{\smash{)}\,+1443}$$

• • •

$$-12 \overline{\smash{)}\,+1443}^{\;-120} \quad \text{with a remainder of } +3$$

To check: $-12 \times -120 = +1440$
$+1440 + (+3) = +1443$

8. There are two binary divide instructions. Their mnemonics are _____ and _____.

• • •

D, DR; Notice that there is no DH instruction.

9. The R1 field of the D and DR instructions must contain the address of an ________(odd/even) register or a program interrupt will be caused by a ________ exception.

• • •

even; specification

10. The even-odd pair of registers addressed by the R1 field of the divide instruction contains a doubleword that is the ____________ (divisor/dividend).

• • •

dividend

11. In the DR instruction, the R2 field has the address of the register containing the ____________.

• • •

divisor

12. In the D instruction, the divisor is a word from __________ __________.

• • •

main storage

13. The quotient and remainder from a D or DR instruction replaces the ______(dividend/divisor).

• • •

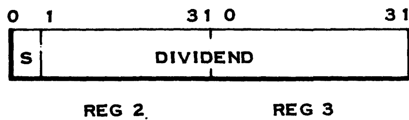
dividend

14. DR

In the above DR instruction, the dividend is in ______ ______________________________.

• • •

Registers 2 and 3 as shown below

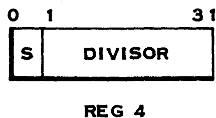


15. DR

In the above DR instruction, the divisor is in ______ ______________________________.

• • •

Register 4 as shown below

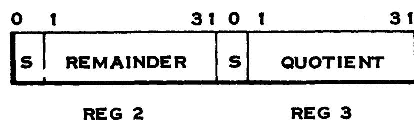


16. DR

In the above instruction, the quotient will be in ____________ and the remainder will be in _________ .

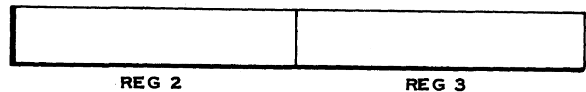
• • •

Reg 3
Reg 2 as shown below

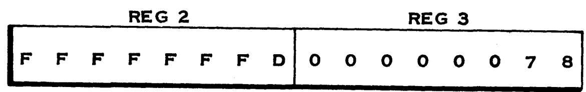


17. Given the following DR instruction, show the contents (in hex) of registers 2 and 3 after the instruction has been executed. Assume the dividend is -1443 and the divisor is -12. Identify the location of the quotient and remainder.

DR



• • •



You have already learned some of the exceptions that can cause program interrupts. They are as follows:

- Fixed point overflow
- Specification
- Addressing

An additional exception is fixed point divide. A fixed point divide occurs any time the quotient cannot be contained as a 32-bit signed integer.

18. When the divisor is zero, a program interrupt will be caused by a ______ ______ ______exception.

• • •

fixed point divide

19. No division takes place and the dividend is left undisturbed any time the System/360 recognizes a _______ ________ _________ exception.

• • •

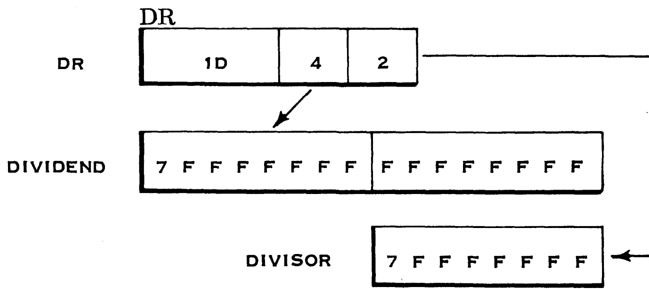
fixed point divide

20. The System/360 would recognize a divisor of ________ as a fixed point divide exception.

• • •

zero

21.



In the above problem:
Will a fixed point divide be recognized? ________
Will the contents of registers 4 and 5 be changed? ________

• • •

Yes. The quotient cannot be contained in reg 5 because it is too large.
No. A fixed point divide exception will occur instead.

22. If that portion of the dividend that is in the even register is equal to or greater than the divisor, the system ______(will/will not) recognize a fixed point divide exception.

• • •

will

END OF SKIP OPTION

1. Back to our sample program. We have calculated the monthly interest to six decimal positions. In order to have it represent an amount of money to the nearest cent, we need to ____ _____ and truncate.

• • •

half-adjust

2. Earlier in this volume, you answered a frame by saying that the programmer would add 5000 to the field to half-adjust. Have we discussed the DC that would specify this constant?

• • •

No.

3. Look at the DC called HAFADJ in Figure 11. Note the F before the specification of the value. What will be placed in storage?

• • •

A fullword binary operand with a value of 5000

Since this type of DC automatically provides the needed constant in binary form, there is no need to convert: It is ready to be used in calculations.

4. Having arranged for the constant, we turn to the coding sheet shown in Figure 8. We will round-off our monthly rate by (adding/subtracting) ________ HAFADJ.

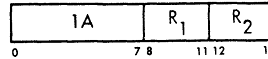
• • •

adding

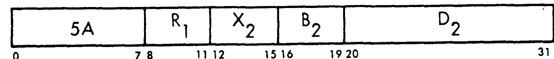
Read the following descriptions of Add and Add Halfword, and locate them on your Reference Data card.

Add

AR R_1, R_2 **[RR]**



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



A and AR:
The second operand is added to the first operand and the sum is placed in the first operand.

- Both operands and the sum are 32 bit signed integers.

A only:
- The second operand is a fullword 32 bit signed integer.
- The second operand must be on a fullword integral boundary.

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

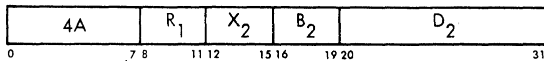
Program Interruptions:
- Addressing (A only)
- Specification (A only)
- Fixed-point overflow

EXAMPLES:

Name	Operation	Operand
	AR	5,8
	A	7,AMOUNT

Add Halfword

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



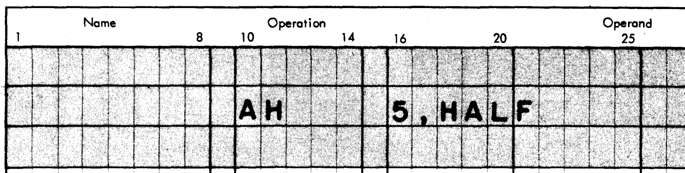
The halfword second operand is added to the first operand and the sum is placed in the first operand.

- The second operand is a 16 bit signed integer.
- The second operand must be on a halfword integral boundary.
- The first operand is a 32 bit signed integer.
- The sum is a 32 bit signed integer.

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

Program Interruptions:
- Addressing
- Specification
- Fixed-point overflow

EXAMPLE:



5. The examples show that we can add either register-to-register or storage-to-register. Which are we going to do in our sample program?

• • •

storage-to-register

6. Where will the sum be?

• • •

in register 3

7. How do we know that HAFADJ is on an integral fullword boundary?

• • •

It was defined by a DC with an F operand.

This not only reserves a fullword of storage but also aligns it on an integral fullword boundary.

SKIP OPTION

If you doubt that you can predict the results of an add instruction, read the frames on the following four pages. Otherwise, skip to the point where we resume discussion of our sample program.

ADD INSTRUCTIONS - ALGEBRAIC

Shown below are three instructions which can be used to add the binary data formats that you have learned.

Mnemonic	Hex Op Code	Data Flow
AH	4A	Halfword storage to register
A	5A	Fullword storage to register
AR	1A	Fullword register to register

It is assumed that you know that a mnemonic is a symbolic method of representing an Op code. Notice that the letter "A" is used to indicate an add instruction. An ending letter of "H" is used to indicate a halfword operand length while an ending letter of "R" is used to indicate an RR type instruction.

In each of the above instructions, the 2nd operand is added to the 1st operand and the sum replaces the 1st operand.

That is, there is no need to analyze the signs and then to complement one of the operands if they were different. This is because negative operands are already in complement form.

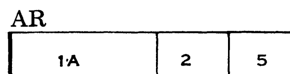
For example, the addition of a value of -1 to a value of + 7 should produce a sum of + 6.

0 0 0 7	+ 7
+ F F F F	-1 (already in complement form)
0 0 0 6	sum of + 6

1. Write (using "hex") the complete instruction to add field A to field B. Assume field A is in register 5 and field B is in register 2.

• • •

AR



Notice that since we are adding to field B, field B (reg 2) is implied to be the 1st operand.

2. Show the contents of registers 2 and 5 as a result of the preceding instruction.

Reg 2   0 0 4 8 7 A 0 1 ________

Reg 5   F F F F A A A A ________

• • •

Reg 2   0 0 4 8 2 4 A B

Reg 5   F F F F A A A A

Notice that the 2nd operand is unchanged by the addition. The 1st operand (in reg 2) is replaced by the sum.

Example of how the System/360 executes the instruction using the actual binary operands:

Reg 2 = 0000 0000 0100 1000 0111 1010 0000 0001

Reg 5 = 1111 1111 1111 1111 1010 1010 1010 1010

0000 0000 0100 1000 0010 0100 1010 1011

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

0 0 4 8 2 4 A B

3. In the preceding example, reg 2 contained a ________ (positive/negative) number and reg 5 contained a ________ (positive/negative) number.

• • •

positive; negative

4. As a result of adding the above numbers, the sum was ________ (positive/negative).

• • •

positive

After an "add" instruction, the condition code is set to indicate one of the four possible arithmetic results. See page 5 of your System/360 Reference Data Card (X20-1703).

5. Indicate the condition code setting for each of the following arithmetic results.

Result	
Zero	____
< Zero or Negative	____
> Zero or Positive	____
Overflow	____

• • •

0; 1; 2; 3

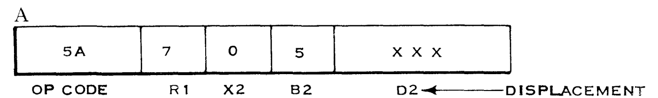
6. The mnemonic "A" is used to indicate a fullword add of storage to register. This instruction, whose Op code is a hex 5A, is of the ___ ___ format.

• • •

RX

7. Assuming that the base address is in reg 5 and that there is no index address, write the instruction that would add a fullword in storage to a fullword in reg 7.

• • •



8. Given the following, show the contents after execution of the preceding instruction.

	Before	After
Reg 7	0 F 0 F 0 F 0 F	________
Storage	F F F F F F F F	________

• • •

Reg 7	0 F 0 F 0 F 0 E
Storage	F F F F F F F F

The resulting sum which replaces the original operand in reg 7 can be determined either by converting the operands to binary and then adding, or simply by adding the hex numbers. Of course, as far as the System/360 is concerned, these are binary operands.

	Hex Addition	Binary Addition
	F F F F F F F F	1111 1111 1111 1111 1111 1111 1111 1111
+	0 F 0 F 0 F 0 F	0000 1111 0000 1111 0000 1111 0000 1111
	0 F 0 F 0 F 0 E	0000 1111 0000 1111 0000 1111 0000 1110
		↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
		0 F 0 F 0 F 0 E

9. The preceding instruction ________ (did/did not) result in a fixed point overflow.

• • •

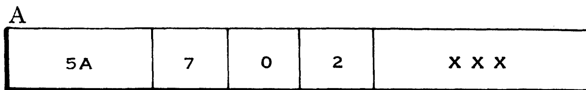
did not

10. The condition code setting as a result of the above instruction would be ______.

• • •

2; The final sum was positive or greater than zero.

11. Using the following instruction, show the contents of reg 7 and storage after instruction execution.



	Before	After
Storage	F 0 F 0 F 0 F 0	______
Reg 7	F F F F F F F F	______

• • •

Storage Unchanged
Reg 7 F0F0F0EF

F F F F F F F F 1111 1111 1111 1111 1111 1111 1111 1111
F 0 F 0 F 0 F 0 1111 0000 1111 0000 1111 0000 1111 0000
F 0 F 0 F 0 E F 1111 0000 1111 0000 1111 0000 1110 1111

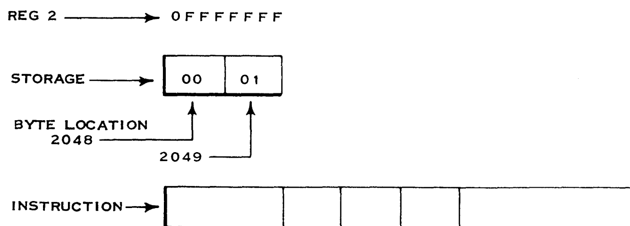
12. Notice that the final sum was negative and as such is in "twos" complement form. The condition code setting will be ______.

• • •

1

If the operand in storage is a halfword, the Op code "4A" (mnemonic: AH) can be used. It also is of the RX format.

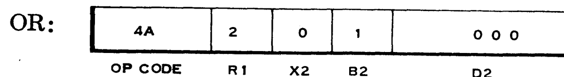
13. Write the instruction that will add the following binary operands. Assume reg 1 has a base address of 2048.



• • •

AH 2,0(0,1)

OR:



In the add halfword instruction, the entire register contents are used. The halfword from storage is expanded to a fullword by propagating the sign bit to the left. The operands are then added and the result goes back into the register.

14. Show (in hex) the contents of reg 2 after adding the indicated halfword. Don't forget to propagate the sign of the halfword.

	Before	After
Reg 2	8 0 0 0 0 0 0 0	______
Storage	F F F F	

• • •

Reg 2 7FFFFFFF

15. In the preceding problem, the condition code would be set to ____ indicating a _____ _____ ________.

• • •

3; fixed point overflow

Remember now, that in the AH instruction, only the storage operand is considered to be a halfword. It is expanded to a fullword by sign bit propagation before being added to the fullword in the register.

16. In review then, there are three instructions to algebraically add binary operands. List their mnemonics. _____ _____ _____

• • •

AR; A; AH

17. A mnemonic which ends in the letter R (such as AR) indicates an instruction of the ___ ___ format. If the mnemonic (of the RX type) ends in the letter H (such as AH), it indicates that the second operand is a ______________ .

• • •

RR; halfword

18. Given the following, what would be the contents of reg 0 after the instruction is executed? What would be the condition code?

Reg 0	Before	0 A 4 3 F 8 7 6
Reg 8	Before	0 0 0 3 2 1 F 9
Reg 0	After	______
PSW Condition Code		____

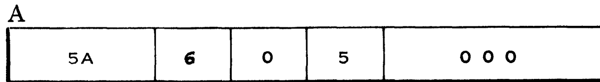
• • •

Reg 0 = 0 A 4 7 1 A 6 F; Condition Code = 2

19. The hexadecimal Op code for the mnemonic A is 5A. Assuming that the 1st operand is in reg 6 and that the 2nd operand has a displacement of zero, with a base address in reg 5 and no index factor, write the instruction (in hex) that would add these operands.

• • •

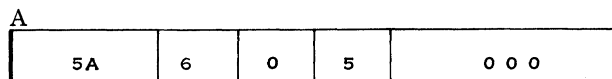
A



MNEMONIC IS A   R1   X2   B2   D2

20. Referring to the preceding instruction and given the following, what will be the contents of reg 6 after the instruction is executed? What will be the condition code?

A



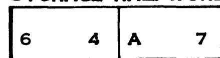
1st Operand	Before	A 0 8 7 F A 7 6
2nd Operand	Before	0 7 4 A 0 2 3 7
Reg 6	After	________
PSW Condition Code		______

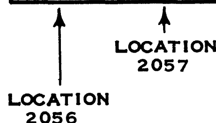
• • •

Reg 6 = A 7 D 1 F C A D;   Condition Code = 1

To save main storage space, smaller binary numbers can be kept in main storage as halfwords. The mnemonic to add a halfword in storage to a fullword in a register is AH. The hexadecimal Op code is 4A.

21. Assuming that reg 12 has a base address of 2048, write the instruction that will add the following halfword to the word in reg 5.

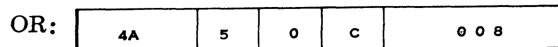




• • •

AH 5,8(0,12)

OR:

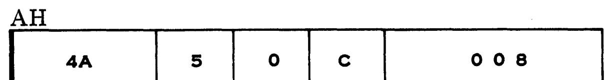


BASE ADDRESS IS IN REG 12

Notice the displacement of 008. This is because location 2056 is eight bytes away from location 2048.

22. Given the following, show the contents of reg 5 and the condition code after the instruction is executed.

AH



1st Operand	Before	0 7 4 A A 4 3 F
2nd Operand	Before	6 4 A 7
Reg 5	After	__________
PSW Condition Code		____

• • •

Reg 5 = 074B08E6;   Condition Code 2

END OF SKIP OPTION

Other add instructions, not illustrated by the sample programs, are:

Name of Instruction	Mnemonic	Basic Function
Add Logical	ALR	The contents of a specified register are added to the contents of another specified register.
	AL	The contents of a specified storage location are added to the contents of a specified register.

Self-study material on these instructions can be found in the Appendix.

1. The half adjusted quotient that was produced by our first divide operation becomes the (dividend/divisor) __________ when we want to set the decimal point.

• • •

dividend

But, according to the information on the divide instruction, the dividend is a 64 bit signed integer. This means that the computer considers the contents of both registers, in the even-odd pair, to be the dividend.

2. Our intended dividend is in register 3: We have to get rid of the __________ that is still sitting in register 2.

● ● ●

remainder (from the first divide)

According to Figure 8, after we "calculate monthly interest" with our first divide instruction, and half-adjust, we "clear the even G(eneral) R(egister) for division".

3. What instruction does this?

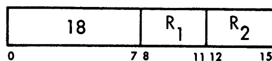
● ● ●

L 2,ZEROS

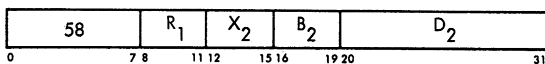
Read the following information and locate the group of load instructions on your Reference Data card.

Load

LR R_1, R_2 **[RR]**



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



LR and L:
The fullword second operand data is placed in the register specified by the first operand.

L only:

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

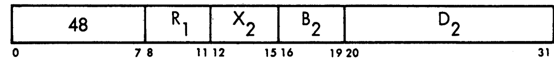
Condition Code:
The code remains unchanged.
Program Interruptions:
Addressing (L only)
Specification (L only)

EXAMPLES:

Name	Operation	Operand
	LR	3,9
	L	7,NUMBERS

Load Halfword

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



The halfword second operand data is placed in bit locations 16-31 of the register specified in the first operand.

- Bit locations 0-15 of the first operand contain the same bit value as bit 16. (The sign bit is propagated through the high order bit positions.)
- The second operand must be on a halfword integral boundary.

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

EXAMPLE:

Name	Operation	Operand
	LH	11,HALF

Since Load and Load Halfword are so similar, we decided to give you the information on both of them.

4. In the second example under Load, suppose that NUMBERS is a fullword with a value of + 15. What will the leftmost bit position in register 7 contain, as a result of the load operation?

● ● ●

0 (signifying a positive number)

5. We want the sign bit in the even-numbered register (of our even-odd pair) to indicate a positive number, but we do not want any of the other bit positions in that register to have a numerical value. We will load register 2 with __________.

● ● ●

zeros

6. Have we defined a constant for these zeros yet?

● ● ●

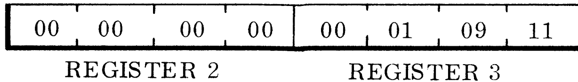
No

7. Write the constant on a coding sheet, label it ZEROS, and check the way it is shown on Figure 11. We have defined a fullword with a value of zero, thus all of its bit positions will be set to zero. Your answer should be equivalent.

If this is register 3,

what will registers 2 and 3 look like after the load operation?

• • •



SKIP OPTION

If you doubt that you can predict the result of a load operation, read the following frames on this page. Otherwise, skip to the point where we resume discussion of our sample problem.

LOAD INSTRUCTIONS

As you know, all input data must come into main storage before it can be processed. In turn, processed data must be in main storage before it can be sent to an output unit. As a result, there must be instructions to take data out of main storage and place it in a general register and later to put the processed binary data back in storage. These instructions are the "load" and "store" instructions. "Load" instructions put data in a register, while "store" instructions put data back in main storage.

There are three "load" instructions which do no more than place data in a general register. These instructions have no effect on the PSW condition code and do not change the 2nd operand.

Mnemonic	Hex Op Code	Data Flow
LR	18	Fullword register to register
L	58	Fullword storage to register
LH	48	Halfword storage to register

1. A "load" operation is specified by the letter ____ in its mnemonic. Just like the "add/subtract" instructions, the mnemonics of the "load" instructions to denote RR format or halfword use ending letters of ______ or ______.

• • •

L; R; H

2. The condition code in the PSW______(is/is not) changed by the LR, L, or LH instructions.

• • •

is not

3. The L and LH instructions load a register with data from main storage. The LR instruction loads a register from a register. Write the instruction (in hex) that will load reg 1 from reg 5.

• • •

LR

4. The LH instruction loads a halfword from storage into bits____ through ____ of a general register.

• • •

16; 31

5. As a result of the LH instruction, bits 0-15 of the register are __________ (changed/unchanged).

• • •

changed

6. The following halfword is placed in a register by use of the LH instruction. Show (in hex) the resulting contents of the register.

Storage A 7 B 6

Register after execution of LH instruction ____________________

• • •

F F F F A 7 B 6; The halfword is expanded to a fullword by propagating the sign bit to the left.

7. In the preceding example, the result in the register is a __________ (positive/negative) number.

●●●

negative: As a reminder, don't forget that negative binary numbers are carried in their complement form.

8. The two programming errors that are possible when using the L and LH instructions are ______________ and _________ exceptions.

●●●

specification; addressing

END OF SKIP OPTION

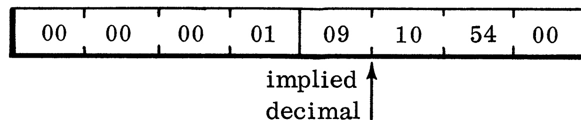
Other instructions of this type, not illustrated by the sample programs are:

Name of Instruction	Mnemonic	Basic Function
Load and Test	LTR	The contents of a specified register are placed in another specified register. A condition code is set, indicating that the contents are zero, negative (< 0), or positive (>0).
Load Complement	LCR	The complement of the contents of a specified register is placed in another specified register. A condition code is set, as in LTR (above).
Load Positive	LPR	The absolute value of the contents of a specified register is placed in another specified register. If the contents of the former are negative, they are complemented before being placed in the latter register.
Load Negative	LNR	The complement of the absolute value of the contents of a specified register is placed in another specified register.
Load Address	LA	The address of a specified storage location is placed in a specified register.
Load Multiple	LM	A specified series of registers is loaded with the contents of a specified storage location.

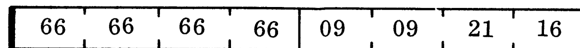
Self-study material on these instructions can be found in the Appendix.

Back to our program:
At this point, it's necessary to picture what the contents of registers 2 and 3 look like.

Suppose that, as a result of the multiply operation, the following decimal amount is in registers 2 and 3 (decimal point implied):



After we divide by 12, we have:



After we half-adjust, we have:

66 66 66 66 09 09 71 16

After we load zeros, we have:

00 00 00 00 09 09 71 16

We have to truncate the contents of the registers so that they contain an implied 9.09.

All along, we have kept track of the decimal; the computer hasn't. So far as it is concerned, the value of the contents is 9,099,116. We must divide this by another whole number, some power of 10, to get a quotient which (to the computer) is 909.

Name		Operation	Operand
		DR	4,7
		D	6,TEN

Suppose that, in the statement D 6,TEN registers 6 and 7 contain 1377045, and the content of TEN is a binary fullword with a value of 10.

1. What would be in each of the registers as a result of the divide operation?

•••

REGISTER 6

00	00	00	05

REGISTER 7

00	13	77	04

2. The quotient shows that division by 10 effectively removes one low-order digit. If the divisor were 100, it would remove ________________.

•••

two low order digits

3. In our sample program, we want to remove four low order digits. We will divide by ________ .

•••

10000

4. Don't look at Figure 11, for a moment. Have we set up a constant to use as a'divisor?

•••

Not yet

5. We want to define a constant which will be a binary operand. What about its location (alignment) if it is to be used with the divide instruction?

•••

It must be located on an integral fullword boundary.

6. Write the type of DC that will give us this operand (use a line on a coding form).

•••

	DC	F'10000'

7. Now look at Figure 11 and see what we're going to call this constant.

•••

DECPT

8. Now that we have the divisor, we can perform the divide operation. What (in terms of quotient and remainder) will be in which registers after our D 2,DECPT?

•••

The quotient will be in register 3 and the remainder will be in register 2.

9. Monthly Interest Amount, half-adjusted and truncated is represented by the contents of __________.

•••

register 3

10. Do we want to print it as output?

•••

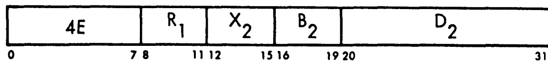
yes

In order for us to print data, it must be in decimal form, in storage. We will take the first step in this direction, with the Interest field, at this point.

Read the following information about the convert to decimal instruction, and locate it on your Reference Data card.

Convert to Decimal

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

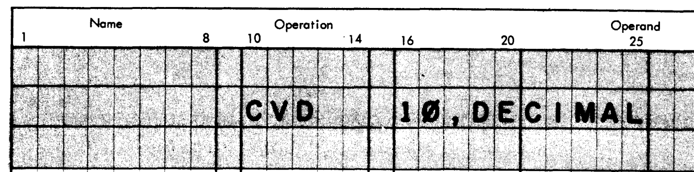


The binary data stored in the register specified by the first operand is changed to a packed decimal signed integer and stored in the second operand.

- The second operand is a right aligned doubleword packed decimal signed integer.
- The second operand must be on a doubleword integral boundary.

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

EXAMPLE:



Note: Since this is another instruction for which the direction of activity is from the first operand to the second, show an arrow ⟶ in the space on the left of its type designation (RX) on your Reference Data card.

6. The register contains a fullword; the second operand must be a doubleword. In the example above, what is done to the data in register 10, and where is it placed?

•••

It is converted to a signed integer in packed decimal form and is right-aligned (placed in the rightmost bytes) of the doubleword called DECIMAL.

7. In our sample program, the instruction CVD 3,PINT puts the monthly rate into an area of storage, in packed decimal form. Is PINT the right sort of storage area to receive it? Check Figure 11, if necessary.

•••

Yes; PINT is a doubleword on an integral doubleword boundary.

SKIP OPTION

If you doubt that you can predict the result of a convert to decimal instruction, read the frames on the following two pages. Otherwise, skip to the point where we resume discussion of our sample program.

CONVERT TO DECIMAL INSTRUCTION

After the data has been processed, it may be desirable to change it back to the zoned decimal format (EBCDIC). This would be necessary if we wished to print the data out in recognizable form or punch the data out in standard card code. This can be done by use of two instructions. The "convert to decimal" instruction will convert the contents of a general register to the packed decimal format and place it in main storage. This packed decimal field can then be changed to the zoned format by use of the "unpack" instruction.

1. The first step in changing a binary result to EBCDIC is to use the CVD instruction. This instruction will change the binary word to a doubleword of __________ decimal data.

•••

packed

2. If the address of the 2nd operand (packed decimal result) in the CVD instruction is not on a doubleword boundary, a __________ exception will be recognized.

•••

specification

3. The coding of the sign bits of the packed decimal result will depend on the sign of the binary word and bit position 12 of the PSW. If bit 12 of the PSW is 0, the EBCDIC plus sign of ______ or minus sign of ______ will be generated. (Refer to EBCDIC chart.)

• • •

1100; 1101

4. If bit 12 of the PSW is set to 1, the standard EBCDIC signs will not be generated. Instead, the generated signs will be those of the extended __________ code.

• • •

ASCII

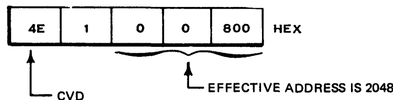
5. The generated sign is placed in the _____ (low/high) order four bits of the doubleword in storage. The remaining bits of the doubleword will contain a total of _____ BCD digits.

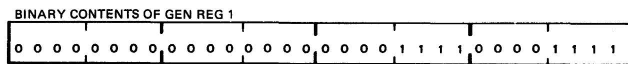
• • •

low; 15

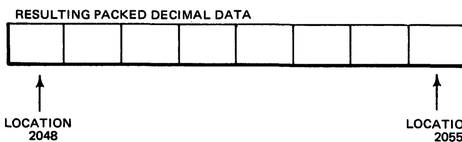
6. Given the following CVD instruction, show the resulting packed decimal field.

CVD

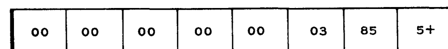




(1ST CONVERT THE BINARY TO HEX. USE THE HEX-DEC CONVERSION TABLE TO FIND THE DECIMAL RESULT.)



• • •



END OF SKIP OPTION

1. According to the flowchart for the program, the next arithmetic operation after calculating, and rounding interest is to calculate ______________ ____________________ (your own words).

• • •

the amount paid on the principal

2. We find this amount, by subtracting ____________ from __________.

• • •

interest; payment

3. Look at Figure 7. Did we convert the field called PAY (Payment) to binary after we packed it into PPAY?

• • •

No

4. As other arithmetic operations do, the subtract operation in the standard instruction set uses binary fields. Which instruction in Figure 8 puts PPAY in the correct format, and where is the result placed?

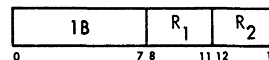
• • •

CVB 2,PPAY; general register 2

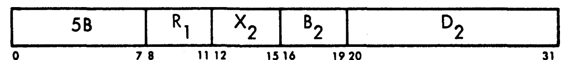
Read the following information on the subtract and subtract halfword instructions, and locate them on your Reference Data card.

Subtract

SR R_1, R_2 **[RR]**



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



S and SR:
The second operand is subtracted from the first operand and the difference is placed in the first operand location.

- Both operands and the difference are 32 bit signed integers.

S only:

- The second operand is a fullword 32 bit signed integer.
- The second operand must be on a fullword integral boundary.

Condition Code:

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

Program Interruptions:

Addressing (S only)
Specifications (S only)
Fixed-point overflow

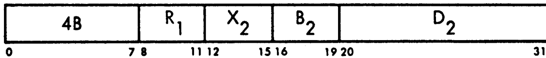
EXAMPLE:

```
         SR    5,10

         S     5,DISCNT
```

Subtract Halfword

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



The halfword second operand is subtracted from the register specified in the first operand and the difference is placed in the register.

- The second operand is a 16 bit signed integer.
- The second operand must be on a halfword integral boundary.
- The first operand is a 32 bit signed integer.
- The difference is a 32 bit signed integer.

Condition Code:

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

Program Interruptions:

Addressing
Specification
Fixed-point overflow

EXAMPLE:

```
         SH    10,HALF
```

When we earlier converted our monthly interest payment to packed decimal form, we did not destroy the contents of register 3. The same field is still sitting there as a 32 bit signed integer.

5. We will use an (SR/S/SH) _____ to subtract it from payment (in register 2).

   •••

   SR

6. Try to write the instruction to calculate the amount paid on the principal, before checking with Figure 8.

   •••

   SR 2,3

7. Where was the difference placed?

   •••

   register 2

8. This wiped out the binary fullword representing _______________.

   •••

   Payment (or PPAY)

SKIP OPTION

If you have any doubt that you can predict the results of a subtract instruction, read the following 14 frames. Otherwise, skip to the point where we resume our sample program.

SUBTRACT INSTRUCTIONS - ALGEBRAIC

Just as there are three Op codes for algebraic addition of binary operands, there are three Op codes for algebraic subtraction.

Algebraic Subtraction

Mnemonic	Hex Op Code	Data Flow
SH	4 B	Halfword storage from register
S	5 B	Fullword storage from register
SR	1 B	Fullword register from register

1. "A" is the mnemonic for add while "S" is the mnemonic for ____________.

• • •

subtract

2. A mnemonic ending in "H" (such as AH or SH) indicates that the second operand is a ________.

• • •

halfword

3. A mnemonic ending in "R" (such as AR or SR) indicates that the instruction is of the _ _format.

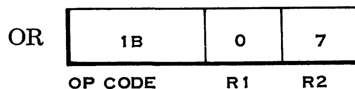
• • •

RR

4. Write the complete instruction that will subtract field B from field A. Both fields are binary operands. Field A is in register 0 and field B is in register 7.

• • •

SR 0,7

OR 

Notice that since we are subtracting field B (reg 7) from field A (reg 0), register 0 contains the 1st operand. Also note that register 0 can be used as an accumulator. As you have previously seen, register 0 could not be used as a base or an index register.

5. Because the preceding instruction says to subtract binary operands, the 2nd operand will be complemented and then________to the 1st operand.

• • •

added

6. SR

In the SR instruction above, the register that will have its contents complemented is register______.

• • •

7

7. In the preceding instruction, the complementing of the 2nd operand (reg 7) during a binary subtract operation ________(does/does not) change the contents of the 2nd operand (reg 7).

• • •

does not; In other words, the 2nd operand will be brought out to the ALU without changing the register. In ALU, the 2nd operand is complemented and added to the 1st operand which has also been brought out to ALU. The resulting answer is then put back in the location of the 1st operand. The actual mechanics of how the ALU does the complementing or adding may vary from one model of System/360 to another. Such topics will not be covered here.

8. SR

The SR instruction will subtract the contents of one register from another. It can also be used to subtract the contents of a register from itself. In the instruction above, the contents of register 6 after instruction execution will be_________.

• • •

zero; The preceding instruction is a good example of how a register may be cleared out.

9. In the preceding example, the condition code was set to __________.

• • •

0

10. The SR instruction used the RR format. The S and SH instruction use the ___ ___ format. These S and SH instructions are identical to the A and AH instructions with the following exception. In the S and SH instructions, the 2nd operand (main storage) is added to the 1st operand after it (2nd operand) has been ________________________.

• • •

RX; complemented

11. Just as in the A and AH instructions, the main storage operands specified by the S and SH instructions must reside on the correct fixed length boundaries. If not, a program interrupt will result and a __________ exception will be indicated in the "__________" PSW.

• • •

specification; "old"

12. If the address of the main storage operand is not available on the particular System/360 (such as address 16,000 on an 8K machine), an addressing exception will cause a __________ __________.

• • •

program interrupt

13. A fixed point overflow can cause a _______ _______.

• • •

program interrupt

14. For the following mnemonics, indicate the instruction formats and the length of the 2nd operand.

Mnemonic	Format	Length of 2nd Operand
SR	________	____________________
S	________	____________________
SH	________	____________________

• • •

Mnemonic	Format	Length of 2nd Operand
SR	RR	Fullword
S	RX	Fullword
SH	RX	Halfword

In all the above instructions, the 1st operand is a word in length.

END OF SKIP OPTION

Other subtract instructions, not illustrated by the sample programs, are:

Instruction Name	Mnemonic	Basic Function
Subtract Logical	SLR	The contents of a specified register are subtracted from the contents of another specified register.
	SL	The content of a specified location in storage is subtracted from the content of a specified register.

Self-study material on these instructions can be found in the Appendix.

1. Now back to our sample program. Do we want the amount applied to the principal to be printed as an output field?

• • •

Yes

2. What location in storage did we set up for it, in packed form? Check Figure 11, if necessary.

• • •

a doubleword called PAMT

3. What happens as the result of the instruction following the first subtraction instruction, in Figure 8?

• • •

The 32 bit signed integer representing the amount paid on the principal is converted to a signed, packed decimal integer and placed in the doubleword called PAMT.

4. We converted the packed decimal equivalent of the Principal Amount to binary once (CVB 3,PPRIN) in order to calculate the annual interest. What happened to the binary word in register 3?

• • •

It was destroyed, when the result of the multiplication operation was placed in registers 2 and 3.

Now we want to subtract the amount applied to the principal from the principal, to calculate the new principal amount.

5. Is the packed version of the original principal amount still in the area called PPRIN?

●●●

Yes. We have done nothing to destroy PPRIN.

6. With the CVB 3,PPRIN instruction, we __________ ______________________________ (in your own words).

●●●

Convert the packed decimal field PPRIN to binary, again, and place it in general register 3.

7. Could we have specified a different general register, if we wanted to?

●●●

Yes, except for GR 2, which contains the binary equivalent of the amount to be applied to (subtracted from) the principal.

8. What does the next instruction in Figure 8 do?

●●●

It calculates the new Principal Balance by subtracting the amount applied to the Principal from the Principal, and by placing the difference in register 3.

9. In what form, and where, is the new principal stored by the CVD instruction that follows?

●●●

It is stored in packed decimal form in a doubleword called PNEWPR.

Let's see where we stand, in terms of our output fields.

We want data in the following output fields:	We have the data in the following fields (P means in packed decimal format):
ACTNUM (Account Number)	ACCTNO
OLDPRI (Old Principal Balance)	PRIN
NEWPRI (New Principal Balance)	PNEWPR
MONPAY (Monthly Payment)	PAY
MONINT (Monthly Interest)	PINT
MONAMT (Amount Applied to Principal)	PAMT

10. Review the coding sheets to check on the preceding. All but three of the desired fields are in zoned decimal form, and need only to be __________ to the respective output field areas to be ready for printing.

●●●

moved

11. The three fields that are in packed decimal form are __________, __________, and __________.

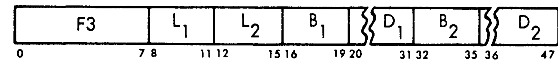
●●●

PNEWPR; PINT; PAMT

Read the following information on the unpack instruction, and locate it on your Reference Data card.

Unpack

UNPK $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]

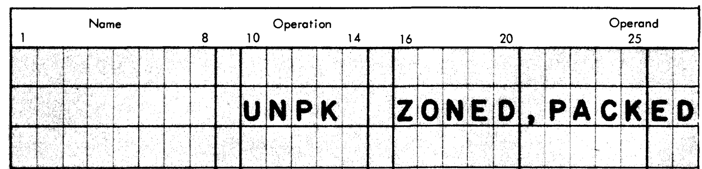


The packed format second operand location is changed to signed zoned format and is placed in the first operand location.

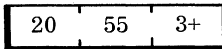
- The fields are processed from right to left.
- If the first operand field is too long, it will be filled with high order zeros.
- If the first operand field is too short, any remaining high order digits will be ignored.
- The maximum size of the first operand (zoned field) is 16 bytes.
- A standard plus (1100) or minus (1101) sign will be attached to the low order digit of the first operand, depending upon the sign of the packed field.

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

EXAMPLE:



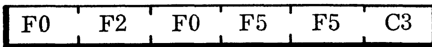
12. Assuming that the three byte field called PACKED looks like this:



20	55	3+

How would the six byte field called ZONED look, after the instruction is executed? Use hex.

• • •



F0	F2	F0	F5	F5	C3

Note that a plus sign of 1111 (hex F) was placed in the zone portion of every byte except the low order byte.

Now you have all the information you need to understand the instructions under the comment "Assemble and Print a Line" in Figure 9.

Let's run through them:

13. Account Number and Principal Amount are moved to their respective output areas. New Principal is unpacked and moved to its output area by the instruction ______________________.

• • •

UNPK NEWPRI, PNEWPR

14. Next, Monthly Payment is ________ to its output area.

• • •

moved

15. Both Monthly Interest and Amount Applied to Principal are ________________________________ ________________________________ (your own words).

• • •

unpacked and moved to their respective output areas

SKIP OPTION

If you doubt that you can predict the results of an unpack instruction, read the following frames on this page. Otherwise, skip to the point where we resume discussion of our sample program.

UNPACK INSTRUCTION

Now that the binary results of the processed data have been placed back in main storage as packed decimal data, the "unpack" instruction can be used to change the data to the zoned decimal format.

1. The 2nd operand of the "unpack" instruction is assumed to be in the ________(zoned/packed) format.

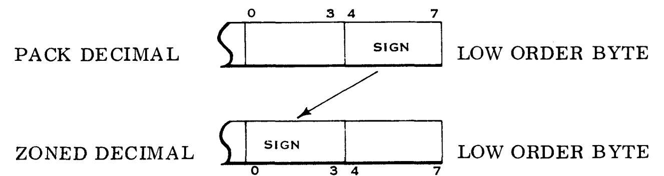
• • •

packed

2. Bits 4-7 of the 2nd operand's low-order byte are placed unchanged in bits ______ of the 1st operand's low-order byte.

• • •

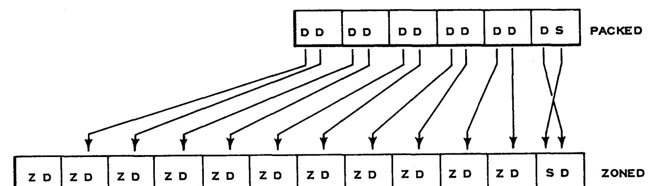
0-3; These bits represent the sign as shown below.



3. The remaining bits of the packed decimal field represent __________. These digits are placed in bits_____ of the bytes of the 1st operand.

• • •

digits; 4-7; As shown below.



Bits 0-3 of the zoned decimal field represents the zone

In review, then, once data has been processed with the fixed point instructions, it can be converted back to the zoned decimal format. This would allow the data to be punched out in the standard card code or printed out in readily readable form. To convert binary data to zoned decimal data requires using the "convert to decimal" instruction and the "unpack" instruction. The "convert to decimal" instruction will take the binary contents of a general register and place it in main storage as a doubleword of packed decimal data. The "unpack" instruction will change the packed decimal data to zoned decimal data.

END OF SKIP OPTION

1. Back to our sample program. Our output fields are all lined up, but the comment by the MVZ instructions says that we must remove ____________ before we can print.

• • •

signs from numbers

When the data was read in originally, every character was given the sign value 1111 (hex F) in the zone portion of each byte. Those fields which were simply moved from their input areas to the corresponding output areas, (Account Number, Principal, and Payment) still carry those zones. They would print as numerical fields.

But the three fields that had to be unpacked to get them to the output area in zoned decimal form would not print properly: The zone portion of each low order byte contains a plus sign (1100) because the packed fields were positive, and the unpack operation places the sign in the zone portion.

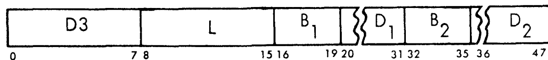
Because of the sign, the low order digit in each of these fields (New Principal, Monthly Interest, and Amount Applied to Principal) would be printed as some other character. Or it might not print at all.

We have to (effectively) remove the signs given to those bytes, by changing their zone portions to 1111.

Read the following information on the move zone instruction, and locate the group of instructions that includes it, on your Reference Data card.

Move Zones

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

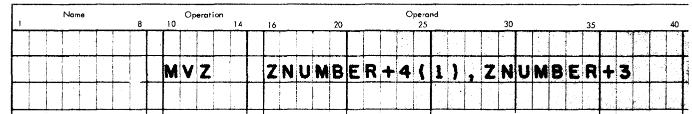


The high order 4 bits (zone portion) of each byte of the second operand are placed in the high order 4 bits of the corresponding bytes in the first operand field.

- Movement is from left to right.
- Movement is one byte at a time.
- The number of zone portions moved is determined by the implicit or explicit length of the first operand.

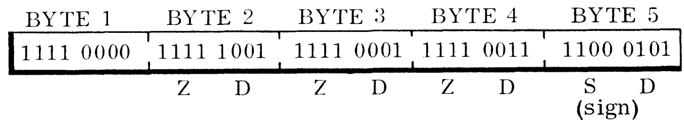
Condition Code:
The code remains unchanged.
Program Interruptions:
Protection
Addressing

EXAMPLE:



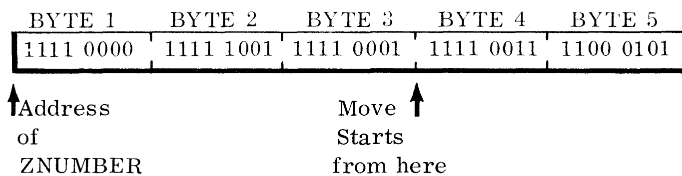
In the example, ZNUMBER is the label of a five byte field. Assume that it was just unpacked and that it carries a positive sign.

The field is shown below, in binary. The zone, digit, and sign bits are indicated.



The way that we replace the sign bits (1100 in the rightmost byte) with 1111, is to move a zone in from one of the other bytes.

Look at the example again. The second operand (from which the zone bits will be moved) has been specified with an address adjustment of +3. This means that the move operation will begin at an address three bytes to the right of ZNUMBER, as shown below.



2. Look again at the example and notice how the first operand was specified. Not only was its address adjusted with a +4, but it was also defined as having an explicit length of ________ (how many?) byte(s).

• • •

One

3. The information on the move instruction stated that the number of zone portions moved is determined by the implicit or explicit length of the first operand. In the example, ________ (how many?) zone portion(s) will be moved.

• • •

One

4. MVZ ZNUMBER+ 4(1), ZNUMBER+ 3 will move the 1111 zone bits from ________________________ to ________________________ (your own words).

• • •

the zone portion of the next-to-last byte; the zone portion of the last byte.

5. In our sample program, what will

MVZ NEWPRI+ 6(1), NEWPRI+ 5

do?

• • •

It will move the zone portion of the next-to-last byte of NEWPRI (NEWPRI+ 5) into the zone portion of the last byte of NEWPRI

6. Will the New Principal Amount field then be ready to be printed?

• • •

Yes

7. The next two MVZ instructions do the same thing for the Monthly Interest and ________________ (full name of field, not the label).

• • •

Amount Applied to Principal

8. If Amount Applied to Principal were a five byte zoned decimal field, instead of a four byte one, how would you write the MVZ instruction? Use the labels used in the program.

• • •

MVZ MONAMT+ 4(1), MONAMT+ 3

NOTE: Our use of an explicit length of 1 byte, for each of the 1st operands, was extremely important and needs to be emphasized.

The description of the MVZ instruction noted that the number of zone portions moved is determined by the implicit or explicit length of the first operand.

Let's see what would happen if we mistakenly left the (1) off the first MVZ instruction, giving:

MVZ NEWPRI+ 6, NEWPRI+ 5

The implicit (sometimes called "implied") length of an operand is the number of bytes in the storage area that was defined for that operand. Thus, the computer would look up NEWPRI in the symbol table and assume that it occupies 7 bytes of storage.

The fact that we used an address adjustment of + 6 would not alter the number of bytes affected by the MVZ operation: The symbol table says 7 bytes, so the computer moves the zone portion of NEWPRI+ 5 into NEWPRI+ 6, then from NEWPRI+ 6 (which is the last byte of NEWPRI) into the first byte of the adjacent storage area, and so on, until it completes seven moves.

This kind of thing can really mess up data in storage.

There are other instructions that have a length value implied in their first operand. One of them, MVC, you have already seen. If you glance down the operand column on your Reference Data Card, looking for an L in operand 1 only, you will see the others.

Some other instructions show an L1 in the 1st operand and an L2 in the 2nd. Depending on the instruction, the number of bytes affected by the operation is controlled by L1 or L2.

Although you haven't used most of these instructions yet, you should be able to complete the following rule:

9. "When using an instruction whose execution depends on the length of one of the operands, if you do not want the implied number of bytes affected, you must (use your own words to state how you must code that operand) ________________________".

• • •

code the operand with an explicit length expressed as a number (of bytes) in parentheses.

You should note that the explicit length (coded in parentheses) is an entirely different thing than address adjustment (coded by a plus sign). Explicit length specifies the number of bytes to be used in the operation, while address adjustment specifies where in storage the operation should begin.

Let's review the points just made, about address adjustment and explicit length, before we continue.

If an operand is coded as FIELDA+ 4(3), we know that we want the operation (whatever it is):

a. to begin with the 5th byte of FIELDA.
b. to affect three bytes of data.

10. What can we tell about an operation if an operand is coded FIELDX+ 7(5) ?

●●●

a. We want the operation to begin with the 8th byte of FIELDX.
b. We want the operation to affect 5 bytes of data.

11. We're ready to print a line ! Briefly, what does the BAL instruction do ?

●●●

It stores the address of the next instruction in register 10 and branches to a routine called WRITE.

12. After a line is printed, the computer branches back to (which instruction ?)________________.

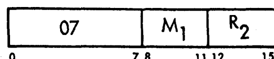
●●●

BC 15,NEXT

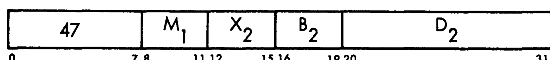
Read the following material.

Branch On Condition

BCR M_1, R_2 **[*RR*]**



BC $M_1, D_2(X_2, B_2)$ **[*RX*]**



A branch to the address specified in the second operand is taken whenever the condition code matches a condition specified in the first operand (M1).

To code this instruction.

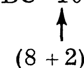
- Place the mask value corresponding to the desired condition code in the first operand.

Condition Code	0	1	2	3
Mask Value	8	4	2	1

Example:
A. Desired Condition Code is 1.
B. Coding is BC 4,BRANCH.

- To test for more than one condition code, place the sum of the mask values corresponding to the desired condition codes in the first operand.

Example:
A. Desired Condition Codes are 0 and 2.
B. Coding is BC 10,BRANCH.

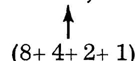


(8 + 2)

Note: Either condition code 0 or condition code 2 will cause a branch to be taken.

- When the first operand is 15, the branch is always taken (unconditional branch).

Example:
A. BC 15,BRANCH



(8+ 4+ 2+ 1)

B. Any condition code (0, 1, 2, or 3) will cause the branch to be taken.

- When the first operand is 0, no branch is taken (a no-operation or "no-op" equivalent).

Example:
A. BC 0,BRANCH
B. None of the condition codes will cause a branch to be taken.

Name (1–8)	Operation (10–14)	Operand (16–)
	BCR	13,2
	BC	15,ALWAYS

13. For which condition code will the BCR example not cause a branch ?

●●●

Condition code 2

14. What will happen when the BC example BC 15,ALWAYS is executed?

●●●

Regardless of which condition code is set, the computer will branch to the address labelled ALWAYS.

15. In our sample program, what will be the result of the execution of the branch instruction BC 15,NEXT ?

●●●

The computer will branch to the instruction labelled NEXT, which is a BAL to the input routine called READ.

And so the execution of the program goes, until there are no more cards to be read.

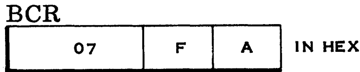
SKIP OPTION

If you doubt that you can predict the results of a BC instruction, for any condition code setting, read the frames on the following two pages. Otherwise, skip to the point where we continue discussion of our program.

BRANCH ON CONDITION INSTRUCTION - REVIEW

1. The second operand fields in all "branch" instructions indicate the "branch to" location. Given the following BCR instruction, the address portion of the PSW will be replaced by bits 8-31 of register _______.

BCR

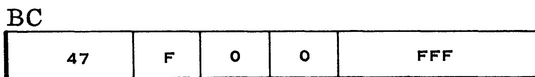


• • •

10

2. Given the following BC instruction, the address portion of the PSW will be replaced by __________ __________________ (the effective generated address/the contents of the storage area at location FFF).

BC



• • •

the effective generated address

3. In the "branch on condition" instruction, the condition code is tested against the R1 field or (as it is referred to) the ______________ field.

• • •

M1 or Mask

4. Each bit of the mask field (bits 8-11 is used to test for a specific setting of the ________ ______.

• • •

condition code

5. The high order bit position is used to test for a condition code setting of _______ .

• • •

0

6. The next bit position to the right tests for a condition code setting of _______ .

• • •

1

7. The 3rd position to the right tests for a condition setting of _____ and the low order position is used to test for a condition code setting of _______.

• • •

2; 3

8. More than one condition code setting can be tested for at the same time by setting the appropriate bits of the mask field. Show the mask field bits that will test for a condition code setting of 2 or 3.

• • •

0011

9. Show the mask field bits that are necessary to branch on an equal or high indication after a "compare" instruction.

• • •

1010

10. The "branch on condition" instruction can be used as an "unconditional branch" instruction. Show the mask field bits that would accomplish this.

• • •

1111 (expressed hexadecimally as F)

11. The "branch on condition" instruction will never result in a branch if the mask field contains ______.

• • •

0000

12. If the R2 field of a BCR (not BC) instruction is 0, a branch __________ (can/cannot) occur.

• • •

cannot

13. Which of the following "branch" instructions will <u>not</u> result in a branch? (Circle one or more.)

a. BCR

07	1	1

b. BC

47	0	0	1	000

c. BCR

07	1	0

d. BC

47	1	0	0	000

• • •

b. because the mask field is zero;
c. because the R2 field is zero
Answers a and d above will result in a branch if the condition code is 3.

END OF SKIP OPTION

Other branch instructions, not illustrated in these sample programs are:

Name of Instruction	Mnemonic	Basic Function
Branch on Count	BCT	A "one" is subtracted from the contents of a specified register. If the result is not zero, the computer branches to a specified address in storage.
	BCTR	Same as above, except that the branch-to address is contained in another specified register.
Branch on Index High	BXH	The contents of a specified register are increased by a certain amount. This sum is compared to the contents of another specified register. If the sum has a higher value, the computer branches to a specified address.
Branch on Index Low or Equal	BXLE	Same operation as above, but the branch occurs if the sum is equal to, or lower than, the contents of a specified register.

Self-Study material on these instructions can be found in the Appendix.

THE LAST ENTRY

1. There is just one more instruction to be mentioned. It is an assembler instruction, written at the end of the program. Can you guess what it is, before checking Figure 11?

●●●

END BEGIN

2. Do you remember what this instruction tells the Assembler?

●●●

It notifies the assembler that the last source card, for this program, has been read.

SUMMARY OF CODING

There has been a lot of information presented, and it is possible that you may have lost sight of how the programmer proceeded, in coding this program.

Look back at Figures 4 and 5, and we'll review the steps.

- The programmer wrote DTF's and I/O macros and established a starting point, a base register, and a linking register.
- He saw, from the description of the job in Figure 4, what the input and output areas must be like, and he wrote the DS's for them.
- After arranging for the output area to be cleared, he wrote the DC's for his header lines (on a separate coding sheet).
- Returning to the coding sheet for the machine instructions, he arranged to have his header lines moved in and printed and for the output area to be cleared.
- Next he caused a card to be read, data required for calculations to be packed and converted to binary, and all of the calculations to be performed. He interrupted the coding of machine instructions, to define work areas and constants used in the calculations, whenever necessary. The use of separate sheets for coding machine instructions, constants, and storage areas helped in this.
- He converted the results of his calculations to decimal, unpacked them, moved them to the output area and put them in the correct form for printing.
- He arranged to have a line of output printed and another card read.

GO ON TO THE SECOND SAMPLE PROGRAM.

DECIMAL ARITHMETIC ON THE SYSTEM/360

1. The coding you have just studied for the Ajax Company problem uses the System/360 Standard instruction set. Numeric data being processed by the Standard instruction set must be (packed/unpacked) __________ and (does/does not) __________ have to be converted to binary before arithmetic operations are performed on it.

●●●

packed; does

2. In packed form, a byte will contain (how many) __________ digit(s).

●●●

Two

There is available for the System/360 a set of instructions which will perform arithmetic operations on data in packed decimal form, without first converting it into binary form. To have these instructions the System/360 must include a feature known as the decimal feature.

3. Numeric data processed by instructions provided by the decimal feature will be in (packed/unpacked) __________ decimal form. It (does/does not) __________ have to be converted to binary before arithmetic operations are performed on it.

●●●

packed; does not

A System/360 which has both the standard instruction set and the decimal feature is said to have a Commercial instruction set.

4. Will the Commercial instruction set process data in both packed decimal and binary forms? __________

●●●

Yes. The standard set will pack numeric data, convert it to binary, and process it in that form. The decimal feature permits processing numeric data in packed decimal form, without converting it to binary.

The standard instruction set uses instructions of the RR and RX formats. This means that data processed by the standard instruction set must be in fixed-length format, occupying a 32-bit word or a 16-bit halfword. Also, these words and/or halfwords are located on integral storage boundaries.

By contrast, the decimal feature provides instructions that are of the SS format. Data processed by these instructions may be in fields of varying lengths, starting at any address in storage. (That is, not aligned to integral storage boundaries.)

5. The lengths in bytes of data fields used in SS (storage-to-storage) operations are indicated where? (Your own words.) ________________ ________________________________.

• • •

In the instructions that process the data

Read the following statements carefully:

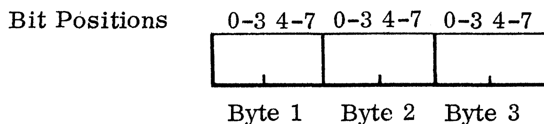
- In packed decimal format two decimal digits are placed in each byte.
- Data is right-aligned in its field. High order bytes not containing significant digits will contain zeros.
- Each field has a sign in the four low order bit positions (bits 4-7) of the right-most byte.
- Processing takes place right-to-left between main storage data fields.

6. Assume a four byte field contains packed data on which arithmetic operations are being performed. What is the maximum number of digits the field can hold?

• • •

7

7. If you answered 8, you forgot that packed decimal fields have a sign in the four low-order bit positions of the right-most byte. Where is the sign in the packed decimal field below?

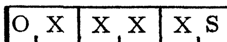


• • •

Bit positions 4-7 of byte 3

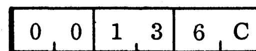
8. A field of 4 decimal digits is packed into a three byte field. Keeping in mind that packed data always is right-aligned, show the contents of the field.

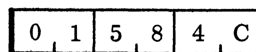
• • •



Don't forget that every unused high order position of a packed field will contain a zero. If a field of 3 digits plus sign (2 bytes) is packed into one that can hold five digits plus sign (3 bytes), zeros will automatically be inserted in the high order byte.

9. If the fields below are to be added, the first digits to be operated on are the 4 and the 6. The next two are the 8 and the 3. You can see that SS processing takes place (right-to-left, left-to-right) ________________.





• • •

right-to-left

The program flowchart and problem statement for the Ajax Company Mortgage Payment Job are shown in Figure 12 and Figure 13 respectively.

Take a coding sheet from the pad, and define the input areas. They are the same as they were for the first program:

The fields within the main area called INPUT are labelled ACCTNO, PRIN, RATE, and PAY. Find the lengths of the fields from the problem statement.

After you have accounted for all of the bytes in INPUT, define the blank byte that will be used to clear the output area. (You may use the asterisk in column 1 to separate entries on your coding sheet as you wish.)

Next define the 132 byte output area, define an unlabelled 33 bytes of storage for spacing, and define a 66 byte HEADER area. Later you will see how the fields within the HEADER area have been labelled.

This is about all you can define at the moment, because the editing of individual output fields, in this program, requires that some be longer than they were in the first program. We'll come to that, later.

Check your coding sheet against the one in Figure 14.

10. We shall now consider the coding for the Ajax Company's problem, using the Commercial instruction set. The Commercial instruction set is composed of (your own words) ________________________________.

• • •

The standard instruction set plus the decimal feature.

Remember that the decimal feature doesn't replace any standard instructions. It merely adds some new ones which increase the power of the system by enabling it to work with packed decimal data.

11. Decimal feature instructions are listed on page 2 of your Reference Data card. There are (how many) ________ decimal feature instructions?

    • • •

    8

12. Two of the decimal feature instructions, the Edit and the Edit and Mark instructions, are used to prepare data for printing. From your Reference Data card you can tell that the other six decimal feature instructions are used for (what kind) ____________ operations.

    • • •

    arithmetic and compare

We will use these decimal feature instructions to perform the arithmetic processing steps in the Ajax Company problem. For the other steps we will use the same standard set instructions as before.

13. Look at the flowchart for the Ajax Company problem. (Fig. 12). At which block do you think we will start using the decimal feature instructions? ______________.

    • • •

    E2

E2 is the first block that calls for arithmetic processing. We will use decimal set instructions for the arithmetic processing called for at E2 and certain other blocks, but we will use standard set instructions for all other processing.

You have learned the standard set instructions used for the other blocks; now you will have an opportunity to use them. As you read the following frames, you will write the instructions for blocks A2, B2, C2, and D2. You will need a coding sheet and your Reference Card.

Read the frames carefully - they will guide and advise you. Don't look back at the coding you already have studied. Don't look at the answers to the frames until you have made an honest effort to write the desired instruction. And don't be afraid to make mistakes. You will learn from your mistakes; you will not learn by copying the answers.

Take another coding sheet.

Our first step is to clear the print area. You will recall that this was done by moving the contents of the byte just preceding the print area into the first position of the print area, using an instruction with operands that will not only move the contents of the byte, but also will propagate them through the entire print area. You will recall that the byte just preceding the print area contained a blank; thus the print area is cleared to blanks by this instruction.

14. Write this instruction. Give it the name START. The symbolic name for the print area is OUTPUT.

    • • •

    START MVC OUTPUT,OUTPUT-1

15. Your next instruction will move the first of two header lines to the print area. You will recall that these header lines are defined as constants, using the DC assembler instruction. Find the correct constant in your list of DC's and DS's in Figure 15 and, using its name, write the instruction that moves it to the print area. You have defined the portion of the print area into which the header line goes on the other coding sheet.

    • • •

    MVC HEADER,HDR1

We will not ask you to code the DC's for header lines or the DS's for work areas (but you can try it on your own). It is most important for you to recognize that you need them, and, at appropriate points in our discussion of the program, check Figure 15.

16. Having placed the first header line in the print area, it must be printed. This calls for a branch to a print routine called WRITE. This routine, plus a similar one used for reading in input records, are shown in Figure 16. They are the same routines you used in the previous version of the Ajax Company problem. It is necessary only to branch to the routine, using the correct register to establish the linkage. The routine branches back to the correct point in your routine automatically. General Register 10 is used in this operation. Write the instruction.

    • • •

    BAL 10,WRITE

17. Now write the instruction that will move the next header line into the print area.

    • • •

    MVC HEADER,HDR2

18. Write the instruction that will branch to an I/O routine, to print this line.

• • •

BAL 10,WRITE

19. Write the instruction that will move the character in the byte immediately preceding the area called OUTPUT into the first byte of OUTPUT and propagate it throughout the area, thus clearing the OUTPUT area.

• • •

MVC OUTPUT,OUTPUT-1

20. You have coded blocks A2 and B2 of the Ajax problem flowchart. The next block, C2, calls for us to get an input record. The procedure is the same as for writing the header lines, except that the special routine is called READ, and brings data into the computer instead of putting it out. Write the instruction and give it the name NEXT.

• • •

NEXT BAL 10,READ

21. According to the flowchart, what is the first thing we must do to get our input data ready for the arithmetic operations? ______________

• • •

Pack it

In our previous coding for the Ajax problem, using only the standard instruction set, we first changed the input data to packed decimal, then converted it to binary format. This was necessary to get the data into processable form.

22. The decimal feature gives us instructions that make it unnecessary for us to do both of these operations. Which is unnecessary? ____________________

• • •

Conversion to binary

23. We do not have to convert our data to binary format but we still have to pack it. Write the instructions that will pack the principal, interest rate, and payment amount into the areas reserved for this purpose. See your list of DS's (Figure 15) for the symbolic names of these areas.

• • •

```
PACK    PPRIN,PRIN
PACK    PRATE,RATE
PACK    PPAY,PAY
```

24. We are now ready to use the first of the decimal set instructions. Decimal set instructions are of the (RX, SS, RR) ____________ format.

• • •

SS

Since instructions of the SS format do not use general registers, we sometimes find it necessary to set up special areas in storage to act as "accumulators". In these "accumulators" we do our arithmetic operations. We can add and subtract factors in the accumulator, and perform multiplication and division on its contents.

We use a regular DS declarative to reserve the accumulator area, making sure it is of the right size to handle the factors with which we are working.

25. For our problem we have set aside an area called PINT, (Figure 15), to be used as an accumulator. According to the list of DS's for this problem, PINT is (how many) ___________ bytes long.

• • •

7

26. Where is the area called PINT located? ___________
____

• • •

In main (or primary) storage

27. What is the purpose of the area called PINT? ______
____

• • •

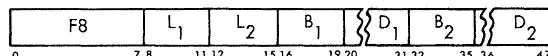
It is used as an accumulator for arithmetic operations performed by decimal set instructions.

We want to get one of the factors in our arithmetic operation into the storage accumulator called PINT. But before we use a storage accumulator we must be sure that it is cleared of data. Our first decimal set instruction is designed to clear the accumulator and put in the first factor in our arithmetic operation.

Read about this instruction in the following material.

Zero and Add

ZAP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The storage location specified by the first operand is cleared to zero and then the second operand data (packed format) is added to the first operand.

- If the first operand field is too short to contain all the significant digits of the second operand, overflow will occur.

Condition Code:
  0 Result is zero
  1 Result is less than zero
  2 Result is greater than zero
  3 Overflow

Program Interruptions:
  Operation
  Protection
  Addressing
  Data
  Decimal Overflow

EXAMPLE:

Name (1–8)	Operation (10–14)	Operand (16–25)
	ZAP	WORK,PDATA

Block E2 of the Ajax problem flowchart calls for calculation of the monthly interest by multiplying the principal by the interest rate. We want to get one of the two factors involved in this multiplication into the accumulator called PINT. We will move the principal into the accumulator, and designate it as the multiplicand.

28. Write the Zero and Add instruction that begins the arithmetic processing in our Ajax problem.

• • •

ZAP PINT,PPRIN

If your answer wasn't correct, cross it out and write it correctly.

29. What is the result of this instruction? ____________

• • •

The area called PINT is set to zero and the contents of the field called PPRIN are placed in PINT.

30. Where are the digits making up the amount of PPRIN located in PINT. ____________

• • •

They are right aligned.

31. Is there a sign in PINT after the operation, and if so, where is it located?

• • •

Yes. In bit positions 4-7 of the right-most byte of PINT.

32. If the number of significant digits in PPRIN do not occupy all the available bytes in PINT, what will be in the high-order bytes of PINT? ____________

• • •

zeros

SKIP OPTION

If you have any doubt about your ability to predict the results of a ZAP instruction, regardless of the symbolic names and field lengths of the operands, you should read the frames on the following pages. Otherwise, you may skip to the point where we resume discussion of our program.

ZERO AND ADD INSTRUCTION

Refer to the description of this instruction, which you read a few frames back.

1. The ZAP instruction will replace the ____________ (1st/2nd) operand.

• • •

1st

2. Does the data in the 1st operand location affect the ZAP instruction? __________

• • •

No; it is ignored.

3. Does the 2nd operand need to be in the packed decimal format or can any type of data be moved by the ZAP instruction? ____________________

• • •

The 2nd operand must be valid packed decimal data or a data exception will be recognized and cause a program interrupt.

4. Do both operands on a ZAP have to be of equal length? ________

• • •

No.

5. What happens to the extra bytes of the 1st operand when the 1st operand is longer than the 2nd operand? ____________________

• • •

They are zeroed out.

6. If the 1st operand is too short to contain all of the significant digits from the 2nd operand, a ________ ________________ will be recognized.

• • •

decimal overflow

7. Given the following ZAP instruction, show the resulting contents of the 1st operand and the condition code.

WUNCMOR ZAP SET1,SET2

SET1 (Before) | 99 | 88 | 7D | SET2 | 78 | 42 | 9C |

SET1 (After) | | | | Condition Code ____

• • •

SET1(After); Condition Code 2

| 78 | 42 | 9C |

8. Given the following ZAP instruction, show the resulting contents of the 1st operand and the condition code.

MORE1 ZAP ONE,TWO

ONE (Before) | 98 | 76 | 54 | 32 | 1C | TWO | 50 | 07 | 6D |

ONE (After) | | | | | | Condition Code ____

• • •

ONE(After);

| 00 | 00 | 50 | 07 | 6D | Condition Code 1

9. Given the following ZAP instruction, show the resulting contents of the 1st operand and the condition code.

MORE2 ZAP DATA1,DATA2

DATA1 (Before) | 17 | 88 | 9C | DATA2 | 00 | 00 | 23 | 71 | 0C |

DATA1 (After) | | | | Condition Code ____

• • •

DATA1(After);

| 23 | 71 | 0C | Condition Code 2

10. In the previous problem, the 2nd operand was longer than the 1st operand. Why wasn't a decimal overflow indicated in the condition code? (If you are uncertain about what causes a decimal overflow, re-read the material about the ZAP instruction.)

____

• • •

All significant digits from the 2nd operand were able to fit in the 1st operand.

11. Given the following ZAP instruction, show the resulting contents of the 1st operand and the condition code.

MORE3 ZAP SET1,SET2

SET1 (Before) | 77 | 77 | 7D | SET2 | 98 | 76 | 54 | 3C |

SET1 (After) | | | | Condition Code ____

• • •

SET1 (After);

| 76 | 54 | 3C | Condition Code 3

12. Will a program interrupt occur after the preceding instruction is executed? ______

• • •

Yes, because the first operand field is too short to contain all significant digits of the second operand. This causes a decimal overflow, resulting in a program interruption.

Remember that for all decimal feature instructions, a program interrupt is caused by exceptional operation codes, operand designations, data or results.

13. Besides the data and decimal overflow exceptions, the ZAP instruction is subject to other exceptions. They are: ______________________,

______________, and ______________.

• • •

Operation (if the Decimal feature is not installed); Addressing; Protection

You will learn more about decimal arithmetic exceptions later.

END OF SKIP OPTION

1. Returning to our Ajax Company problem, we find we have the principal in the accumulator called PINT. The next step in calculating the monthly interest is to multiply the principal by the interest rate. From your Reference Data card you can deduce that the decimal instruction we will use is ______________.

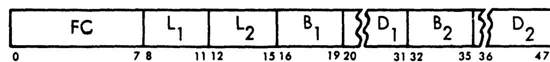
• • •

Multiply Decimal (MP)

Read about this instruction in the following material.

Multiply Decimal

MP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The first operand (multiplicand) is multiplied by the second operand (multiplier) and the signed product is placed in the first operand location.

- Both the multiplicand and the multiplier must be in packed format.
- The product is in packed format.
- The length of the first operand in bytes, must be equal to or greater than the number of bytes required to contain all the significant digits of the multiplicand plus the total number of bytes in the multiplier (second operand) field.

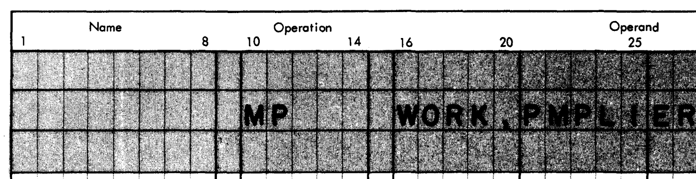
  Example:

  Multiplier = xxxx | 0x | xx | xS | (3 bytes)
  Largest multiplicand = xxxxxxx | xx | xx | xx | xS | (4 bytes)
  The product field length must then be 7 bytes (3+4) or larger in length. | xx | xx | xx | xx | xx | xx | xS |

- The multiplier may not exceed 15 digits and sign (8 bytes) in length.
- The maximum product size is 31 digits and sign (16 bytes).

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

EXAMPLE:



2. Now write the instruction that will multiply the principal (placed in the accumulator called PINT by the ZAP instruction) times the interest rate.

•••

MP PINT, PRATE

If necessary, correct your answer.

3. Where will the product be located after the operation? ____________________

•••

In PINT, right aligned.

4. The product (will/will not) ________ be signed.

•••

will

5. How will the sign be arrived at, and where will it be located? ____________________

•••

By the rules of algebra. In bit positions 4-7 of the right-most byte of PINT.

6. Where will the original multiplicand (PPRIN) be located in PINT? ____________________

•••

Nowhere. It will have been lost in the multiply operation.

7. The problem statement for the Ajax problem gives the size of the input data fields and the number of decimal places, if any, in each. The product in PINT will have (how many) ______ decimal places.

•••

6

SKIP OPTION

If you have any doubts about your ability to predict the results of a decimal multiply operation, regardless of the symbolic names, field lengths, and signs of the multiplier and multiplicand, you should read the frames on the following pages. Otherwise, you may skip to the point where we resume discussion of our program.

MULTIPLY DECIMAL INSTRUCTION

Re-read the description of this instruction; then let's take a few simple examples.

1. In the MP instruction, the 1st operand is the multiplicand. The 2nd operand is the multipler. The product will replace the ____________________ (multiplicand/multiplier)

•••

multiplicand; It is the 1st operand.

2. Given the following MP instruction, identify the multiplier and multiplicand.

AGAIN1 MP FLD1,FLD2

FLD1 | 00 | 00 | 11 | 2C | FLD2 | 01 | 0C |

A. ________________ B. ________________

•••

A. multiplicand; B. multiplier

3. For the preceding MP instruction, show the resulting contents of the multiplicand:

FLD1 | | | | |

•••

| 00 | 01 | 12 | 0C |

The contents of FLD1 are 112, with a plus sign.
The contents of FLD2 are 10, with a plus sign.

The product is

```
   112  +
x   10  +
  ----
  1120  +
```

4. Show the resulting product for the following MP instruction.

AGAIN2 MP SET1,SET2

SET1(Before)

00	09	9D

SET2

9D

SET1(After)

• • •

SET1;

00	89	1C

AGAIN3 MP TOT1,TOT2

5. If the amounts in TOT 1 and TOT 2 both have positive signs, will the product be positive or negative ? ________________

• • •

Positive. Like signs give a positive product.

TOT1 (Before)

01	07	32	1D

TOT2

01	23	4D

TOT1(After)

6. Can the resulting product for the above MP instruction fit into the multiplicand field? ________________

• • •

No; the rule of thumb is that the number of digits in the product is equal to the sum of the number of significant digits in both operands.

To prevent the product from overflowing the multiplicand field on a "multiply decimal", the System/360 has the following restriction on the multiplicand.

The number of high-order bytes containing zeroes in the multiplicand must be at least equal to the number of bytes containing significant digits in the multiplier. For example:

If the multiplier is

01	23	4D

, there must be 3 high-order bytes with zeroes in the multiplicand such as:

00	00	00	01	07	32	1D

AGAIN4 MP SETA,SETB

SETA

01	07	32	1D

SETB

01	23	4D

7. The above MP instruction will result in a ________ exception and cause a program interrupt.

• • •

data; because the number of high-order zeroes in the multiplicand is less than the size of the multiplier.

8. What must be done to prevent a specification exception on an MP instruction? ______________
____

• • •

The multiplier must be shorter than the multiplicand and cannot have a length code greater than 7 (15 digits and a sign).

END OF SKIP OPTION

1. Returning to the Ajax Company problem, we see that we have calculated the yearly interest by multiplying the principal times the yearly interest rate. The result is in the "accumulator" called PINT. To get the monthly interest, we must divide the yearly interest amount in PINT by 12. From your Reference Card, select the decimal instruction that will perform this operation.

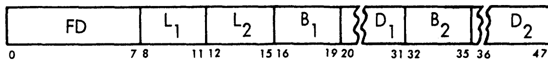
• • •

Divide Decimal (DP)

Read about this instruction in the following material.

Divide Decimal

DP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The first operand (dividend) is divided by the second operand (divisor) and quotient and remainder are placed in the first operand.

- The dividend and the divisor must be in packed format.
- The quotient and the remainder are in packed format.
- The dividend field (first operand) length in bytes must be equal to or greater than the total length in bytes of the divisor (second operand) plus the length in bytes of largest quotient expected in the problem.
- The remainder will be a signed integer right aligned in the right-most portion of the first operand field and has a length in bytes equal to the length of the divisor. The sign of the remainder is the same as that of the dividend.
- The quotient will be a signed integer and will occupy the remaining left-most bytes of the first operand field. The sign of the quotient is determined by algebraic rules from dividend and divisor signs.

Example:
If the divisor is 4 bytes long and the dividend is 6 bytes long, after the divide operation, the remainder will occupy the right-most 4 bytes of the dividend field and the quotient will occupy the rest of the dividend field.

Dividend	XX	XX	XX	XX	XX	XS	(6 bytes)
Divisor			OX	XX	XX	XS	(4 bytes)
Result (In dividend field)	XX	XS	OX	XX	XX	XS	
	Quotient		Remainder				

- The maximum size of the divisor (second operand) is 15 digits and sign (8 bytes).
- The maximum size of the dividend (first operand) is 31 digits and sign (16 bytes).

Condition Code:
The code remains unchanged.

Program Interruptions:
Operation
Protection
Addressing
Specification
Data
Decimal Divide

Programming Note:

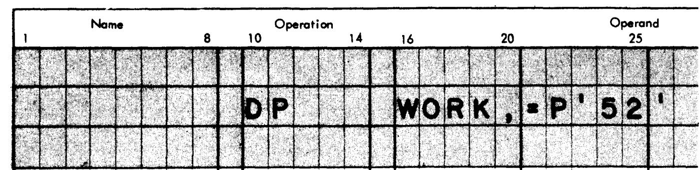
It is often desirable to obtain a quotient of more decimal places than are represented by the dividend. The result may be readily obtained by multiplying the dividend, prior to the divide operation, by the proper power of 10, e.g., 10, 100 or 1000 for 1, 2 or 3 more decimal places respectively, in the results. The Multiply effectively shifts the dividend the desired number of places to the left.

For example, assume that the sum of 7 numbers is 53 and that we wish to find the average of these numbers to 2 decimal places. Dividing 53 by 7 using the DP instruction would result in a quotient of 7 and a remainder of 4. However, if the dividend is multiplied by 100 before dividing by 7, the quotient (with decimal point inserted for clarity) will be 7.57 and a remainder of 1 after the DP operation.

In defining the length of the dividend field, one additional byte must be added to the calculated length for every two additional decimal places or fraction thereof.

For example, if the length of the dividend field was calculated to be 8 bytes and an additional 4 decimal places were required in the quotient, two more bytes would have to be added to the calculated field length giving a total of 10 bytes. Note that because this is a packed field, 3 additional decimal places would also require two more bytes.

EXAMPLE:



During the first sample program you were introduced to the use of a literal as an operand. As you may recall, it was in the instruction:

```
D      2,=F'12'
```

During assembly, this instruction would establish a binary value of 12 in a fullword (as required for the divide operation).

In this second sample program we will again use a literal to divide yearly interest by 12 but, since we are working with packed decimal data, we will specify the literal accordingly.

Look at the coded example above. The contents of WORK will be divided by 52, when this instruction is executed. We tell the Assembler to establish the literal as a packed decimal value, by using the type code P.

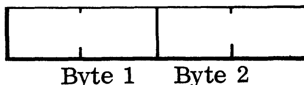
2. With this in mind, write the instruction that will divide the yearly interest in PINT by a constant of 12 to give monthly interest: ________________

•••

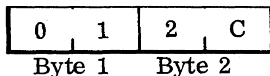
```
DP   PINT,=P'12'
```

Correct your answer, if necessary.

3. The operand =P'12' causes the number 12 to be established in storage as a constant in packed decimal form. The constant will have a plus sign. Show how the constant will look in storage.



•••



4. The second operand of a divide instruction is the (divisor/dividend) ________________ .

•••

divisor

5. The divisor in our instruction is (your own words) ________________

•••

A constant of 12 in packed decimal form

6. As a result of the divide operation, where will the remainder, if any, be positioned? ________________

•••

It will be right justified in PINT

7. The remainder (will/will not) ______ have a sign.

•••

will

8. What will be the size of the remainder? ________________

•••

2 bytes. Remainder area is always equal in size to the size of the divisor.

9. Where will the quotient be located? ________________

•••

In PINT, to the left of the remainder.

10. Is there any space in PINT between quotient and remainder? ______

•••

No.

11. Is the quotient signed, and if so, where is the sign located? ________________

•••

Yes. Quotient sign is in bits 4-7 of the right-most byte of the quotient area in PINT.

12. In a decimal divide operation, how is the minimum size of the dividend field determined? ________________

•••

By adding the length, in bytes, of the divisor to the length, in bytes, of the largest quotient expected in the problem.

13. If you wanted the quotient to 5 additional decimal places, ________ additional bytes would be required in the dividend field.

•••

3

14. Now, multiply the dividend by ________ to obtain the desired result in the quotient.

•••

100,000

SKIP OPTION

If you have any doubts about your ability to predict the results of a decimal divide operation, regardless of the symbolic names, field lengths, and signs of the factors involved, you should study the frames, in the following two pages. Otherwise, you may skip to the point where we resume discussion of our program.

DIVIDE DECIMAL INSTRUCTION

LABEL	DP	D1(L1,B1),	D2(L2,B2)
	Op code	location of dividend	location of divisor

1. As you can see above, the dividend is the _______ (1st/2nd) operand and the divisor is the _______ operand.

•••

1st; 2nd

2. As in the other instructions you have studied, the generated effective storage addresses refer to the ________ (high/low) order byte of the data fields.

•••

high

3. The mnemonic for the "divide decimal" instruction is DP. This indicates that the divide instruction operates on _______ (packed/zoned) decimal data.

•••

packed

DP SET1(16),SET2(7)

4. The above DP instruction has a dividend that is ______ bytes in length.

•••

16

5. Sixteen bytes of packed decimal data can contain ________ digits and a sign.

•••

31

FIGURE DP SUBTOT(4),SUBTOT+4(2)

SUBTOT	00	02	56	0C	01	6C

6. Given the above DP instruction, a value of _______ will be divided by a value of _________.

•••

+2560; +16

The DP instruction will have as a result both a quotient and a remainder. These two results will be in the packed decimal format and will replace the dividend. The quotient will replace the high order and the remainder will replace the low order of the dividend. The following example will illustrate this:

PROCES DP SET(4),SET+4(2)

SET (before)	00	02	56	0C	01	6C
SET (after)	16	0C	00	0C	01	6C
	quotient		remainder		unchanged	

Note: The remainder is always the same size as the divisor!

7. Given the following "divide decimal" instruction, show the resulting contents of the dividend field.

AGAIN DP SET(4),SET+4(2)

SET (before)	00	01	44	0C	01	2C
SET (after)					01	2C

Location 2048

•••

SET	12	0C	00	0C

8. The remainder is placed in the low order of the dividend field and always contains the same number of bytes as the ______________________.

•••

divisor or 2nd operand

9. The quotient is placed in the dividend field just to the left of the ____________.

•••

remainder

Re-read the description of the "divide decimal" instruction, which you read earlier in this series of frames.

10. The address of the quotient of a DP instruction will be the same as the original ____________________ .

• • •

dividend or 1st operand

11. The size of the quotient will be equal to the dividend size minus the __________ size.

• • •

divisor

12. If the quotient cannot be fitted into its area, a __________ __________ exception will be recognized.

• • •

decimal divide

13. When a decimal divide exception is recognized, the dividend field will _____________ (remain unchanged/ contain part of the quotient).

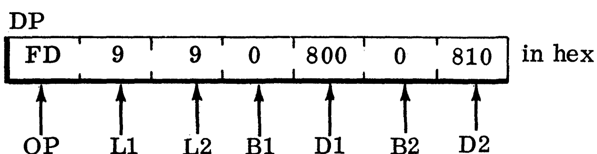
• • •

remain unchanged

Another programming rule that applies to the "divide decimal" instruction is this:

The divisor must be shorter than the dividend and cannot exceed eight bytes. That is, $L2 < L1$ and $L2 \leq 8$.

DP

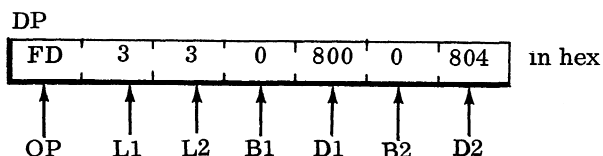


14. The above DP instruction will result in a specification error because the divisor's length code is greater than _________ .

• • •

7

DP



15. The preceding DP instruction will result in a specification error because ______________________.

• • •

The divisor is not shorter than the dividend.

Let's summarize what we have learned about the divide decimal operation.

- The Dividend
  a. The 1st operand is the dividend.
  b. The dividend has a maximum size of 31 digits and a sign.
  c. The dividend will be replaced by the quotient and remainder.
  d. The dividend must have at least one high-order zero digit.
- The Divisor
  a. The 2nd operand is the divisor.
  b. The divisor has a maximum size of 15 digits and a sign.
  c. In all cases, the divisor must be shorter than the dividend.
- The Remainder
  a. The remainder replaces the low order bytes of the dividend field.
  b. The remainder has the same length as the divisor.
  c. The sign of the remainder is the same as the sign of the original dividend.
- The Quotient
  a. The quotient replaces the high order bytes of the dividend field.
  b. The size of the quotient is equal to the dividend size minus divisor size (L1-L2).
  c. Since the quotient is placed in the high order bytes of the dividend field, its address will be the same as the dividend's.
  d. The sign of the quotient follows the rules of algebra:
     (1) Like signs = +
     (2) Unlike signs = -
- Decimal Divide Exception
  a. This exception indicates that the quotient would be too large to be fitted into its allotted field.
  b. The decimal divide exception is recognized prior to any division. The dividend field is left unchanged and a program interrupt is taken.
- Specification Exception
  This exception is recognized on a "divide decimal" instruction whenever:
  a. The divisor is longer than eight bytes.
  b. The dividend is not longer than the divisor.

END OF SKIP OPTION

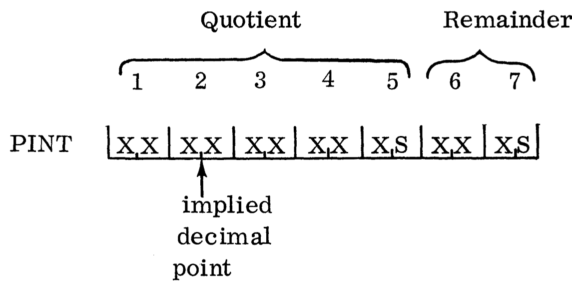
Returning to the Ajax Company problem, we find that the monthly interest rate is in the 5 leftmost bytes of our 7 - byte storage "accumulator" called PINT. (The two rightmost bytes of PINT contain the remainder of the divide operation; we are not interested in them.)

1. Since our dividend (the yearly interest) contained six decimal positions and the divisor (constant of 12) contained no decimal positions, the quotient contains (how many) __________ decimal positions.

• • •

6

Here is a picture of how PINT looks at this point:



2. We will be working with the (quotient/remainder) __________ which is in the (how many?) _______ leftmost bytes of PINT.

• • •

quotient; 5

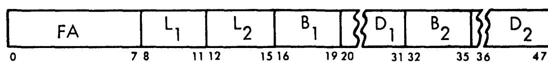
3. We want to round off our monthly interest figure to two decimal positions. We will half adjust by adding a 5 into the thousandths position of the quotient in PINT. Where is this? (your own words) __________
____
From your Reference Card you can determine that the instruction we will use is __________________ .

• • •

The rightmost bit positions (4-7) of the 3rd byte of PINT. ; Add Decimal (AP)

Add Decimal

AP $D_1(L_1, B_1), D_2(L_2, B_2)$ **[SS]**



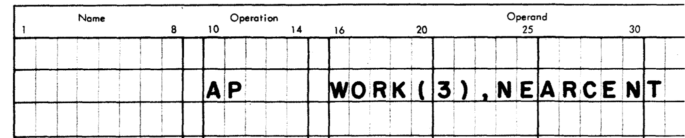
The second operand is added to the first operand and the sum is placed in the first operand storage location.

- Both operands must be in packed format.
- The sum is in packed format.
- If the first operand is too short to contain all the significant digits of the sum, overflow occurs.
- If the second operand is shorter than the first operand, addition will take place normally.
- A field may be added to itself.

Condition Code:
0 Sum is zero
1 Sum is less than zero
2 Sum is greater than zero
3 Overflow

Program Interruptions:
Operation
Protection
Addressing
Data
Decimal overflow

EXAMPLE:



In the example shown above, a constant called NEARCENT is added to WORK. The explicit length parameter (3) in the instruction limits the addition operation to the 3 leftmost bytes of WORK.

4. Write the instruction that will add a constant called HAFADJ to the monthly interest amount in the 5 leftmost bytes of PINT.

• • •

AP PINT(5),HAFADJ

Correct your answer, if necessary.

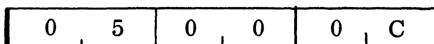
5. The symbolic name HAFADJ can be found in the list of DC's for this version of the problem. (Figure 15). It refers to a signed constant in (packed/unpacked) __________ form, whose value is __________ .

• • •

packed; 5000

6. Show how this packed decimal constant of 5000 looks in storage.

• • •



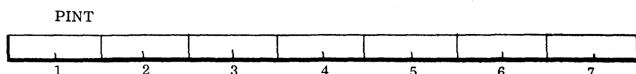
7. The add decimal instruction rounds off the monthly interest amount by adding the constant of 5000 at the appropriate place in PINT. Study the add decimal instruction carefully. For this step the length in bytes of PINT will be ________ .

• • •

5

PINT will be 5 bytes long for this step because of the explicit length indicator (the digit 5 in parentheses) in the add instruction. These are the 5 leftmost bytes of the original 7 making up PINT.

8. Remembering that in SS instructions, processing takes place from right to left, the constant of 5000 will be added to bytes ___ ___ ___ of the 5 bytes we are working with on this step.



• • •

5, 4, 3

9. The digit 5 from the constant will be added to the (tenths, hundredths, thousandths) ______________ decimal position.

• • •

thousandths

10. Adding the constant of 5000 into the appropriate place in PINT, half-adjusts the monthly interest amount. What will the condition code be, after this operation? ______

• • •

2(the sum is positive)

SKIP OPTION

If you have any doubt about your ability to predict the results of an ADD Decimal instruction, regardless of the symbolic names, field lengths, (implicit or explicit) and algebraic signs of the factors involved, you should read the frames on the following four pages. Otherwise, you may skip to the point where we resume discussion of our program.

ADD DECIMAL INSTRUCTION

Refer to the description of this instruction which you read earlier in this series of frames.

1. AP is the mnemonic for the " ____ ___________ ".

• • •

"add decimal"; the ending letter of P tells us that both operands must be in the packed decimal format.

2. In the "add decimal" instruction, the sum of the 1st and 2nd operands replaces the _______ operand.

• • •

1st

3. Since all packed decimal numbers are in true form, the signs of the field must be analyzed prior to any addition. If the signs are different (one plus and one minus), the 2nd operand is _____________ (added to/subtracted from) the 1st operand.

• • •

subtracted from

4. Subtraction on a computer is done by means of ______________ addition.

• • •

complement

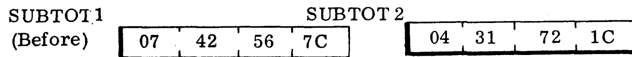
5. Given the following AP instruction, show the resulting contents of the 1st operand.

```
GO        AP    SUBTOT1,SUBTOT2

SUBTOT2   DS    PL4

SUBTOT1   DS    PL4
```

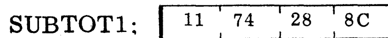
SUBTOT1 (Before)



SUBTOT1 (After)

• • •

SUBTOT1:



Notice that the sign bits weren't added.

6. The 2nd operand ________ (was/was not) changed by the preceding problem.

• • •

was not

7. In the preceding problem, two positive numbers were added together. The resulting positive sum would set the condition code to ______ .

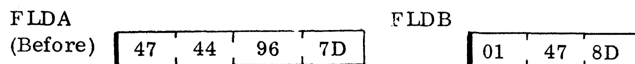
• • •

2

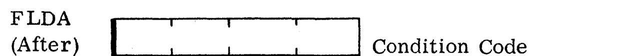
8. Given the following AP instruction, show the resulting 1st operand field and the condition code.

```
GOAGIN   AP   FLDA,FLDB
```

FLDA (Before)



FLDA (After)



Condition Code________

• • •

FLDA (After);

Condition Code 1

9. In the preceding problem, two ________ (positive/negative) numbers were added together and the sum was ______________ (positive/negative).

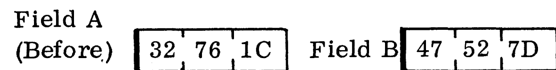
• • •

negative; negative

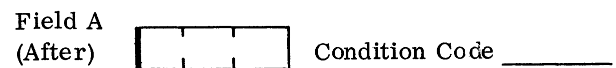
10. If the signs of the two operands are different, the 2nd operand is effectively subtracted from the 1st operand. Given the following AP instruction, show the resulting 1st operand and condition code.

```
GO1   AP   FLDA,FLDB
```

Field A (Before)



Field A (After)



Condition Code ________

• • •

Field A; (After)

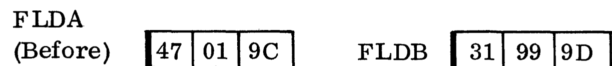
Condition Code 1

As you previously learned, subtraction in a computer is usually done by means of complement addition.

11. Given the following AP instruction, show the resulting 1st operand and condition code.

```
GO2   AP   FLDA,FLDB
```

FLDA (Before)



FLDA (After)

Condition Code________

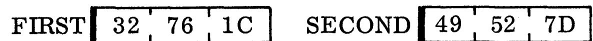
• • •

FLDA: (After)

Condition Code 2

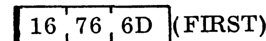
12. Given the following "add decimal" instruction with operands that have different signs, find the final result.

```
GO4   AP   FIRST,SECOND
```



Final Result (FIRST)

• • •



(FIRST)

13. Since decimal data is always carried in true form, the signs must be analyzed for arithmetic operations. Given the following signs, indicate whether the fields are true or complement added. The instruction is "add decimal". (Choose the correct answer for each set of signs.)

	1st Operand	2nd Operand	
a.	+	+	True add/Complement add
b.	+	-	True add/Complement add
c.	-	-	True add/Complement add
d.	-	+	True add/Complement add

• • •

a. True add; b. Complement add; c. True add; d. Complement add

14. Given the following AP instruction, show the resulting contents of the 1st operand and the condition code.

GO5 AP FIRST,SECOND

FIRST(Before) | 00 | 67 | 9D | SECOND | 67 | 9C |

FIRST(After) | | | | Condition Code______

• • •

FIRST(After) | 00 | 00 | 0C | Condition Code 0

In the previous problem, there are two equal values with different signs. Since one quantity would be subtracted from the other, the result would be zero as indicated by the condition code setting. A zero result is always plus. That is the reason for changing the sign of the 1st operand from minus to plus.

15. The original length of the 1st operand will never be exceeded regardless of the result. Carries beyond the 1st operand's high order byte are lost. When there is a high-order carry, the condition code is set to __________.

• • •

3

16. A carry out of the high order during an AP instruction is called a ________ ____________.

• • •

decimal overflow

17. A decimal overflow _____ (can/cannot) cause a program interrupt.

• • •

can

18. What else can cause a decimal overflow besides a high-order carry? ______________________________

____

• • •

The number of significant digits in the 2nd operand exceeding the length of the 1st operand.

19. Which of the following can cause a decimal overflow on an AP instruction? (Select one of the following.)

	1st Operand	2nd Operand
a.	47 9C	52 0C
b.	98 1C	22 7D
c.	47 2C	00 37 6C
d.	None of the above.	

• • •

d

20. Which of the following can cause a decimal overflow on an AP instruction? (Select one of the following.)

	1st Operand	2nd Operand
a.	22 7C	00 90 7C
b.	50 0D	50 0D
c.	04 7C	01 00 1C
d.	All of the above.	

• • •

d

Besides a decimal overflow, there are other programming exceptions that can occur on an "add decimal" instruction. They are:

- Operation--If the decimal feature is not installed on a system, any of the eight decimal instructions are considered illegal.
- Protection--Since the result of the instruction replaces the contents of main storage, this instruction is subject to a storage protection violation.
- Addressing--Any instruction which addresses main storage for an operand is subject to an addressing exception. This exception occurs when the address is not available on a particular system (such as an address 16000 on an 8K system).
- Data--All packed decimal operands are checked for valid digits and sign. All of the digit positions must be coded from 0000-1001. The sign position must be coded from 1010-1111.

21. What would happen if the "add decimal" instruction were used to add two zoned decimal fields? _______

____

• • •

A data exception would be recognized and a program interrupt would occur.

22. The decimal feature is optional on models 30 and 40 of System/360. What would happen if an "add decimal" instruction was fetched on a model 30 which doesn't have the decimal feature installed? __

____

• • •

An operation exception would be recognized and a program interrupt would occur.

END OF SKIP OPTION

1. Returning to our Ajax Company problem, we see that the monthly interest amount, half adjusted with two decimal positions, is in the leftmost 2 1/2 bytes of our accumulator PINT. We will want to extract this figure from the accumulator and use it in further arithmetic operations. But we know that decimal operations require factors to be signed. So we see that we must attach a sign to the monthly interest amount. Where do you think the sign should be placed? ______________________________

• • •

In bit positions 4-7 of the third byte of PINT.

2. The monthly interest amount was developed as the quotient of a divide operation; thus its sign is the sign of the quotient. Where is the sign of the original quotient? ______________________

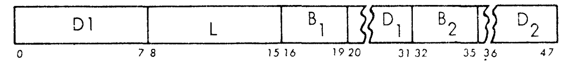
• • •

In bit positions 4-7 of the fifth byte of PINT.

The instruction we will use to move the quotient sign from byte 5 to byte 3 is the Move Numerics instruction. Read about this instruction in the following material.

Move Numerics

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



The numeric portion (low order four bits) of each byte in the second operand are placed in the numeric portion of the corresponding bytes in the first operand.

- The number of bytes in the operation is determined by the implicit or explicit length of the first operand.
- Movement is from left to right through each field.
- Movement is one byte at a time.
- Zones remain unchanged.

Condition Code:
The code remains unchanged.

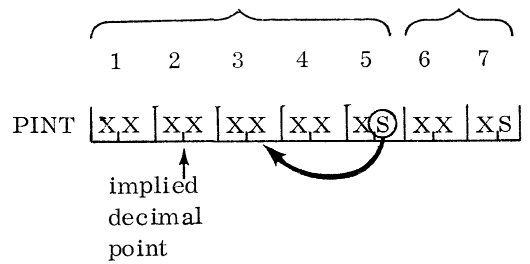
Program Interruptions:
Protection
Addressing

Name	Operation	Operand
	MVN	PINT+2(1),PINT+4

The instruction shown above will cause our monthly interest amount to have the proper sign, in the proper place.

Copy this instruction onto your coding sheet.

Here's how this instruction works:



3. As a result of the Move Numerics instruction, the original quotient sign is moved from bit positions 4 - 7 of the fifth byte in PINT to bit positions 4 - 7 of the third byte in PINT. How does the first operand of the move numerics instruction specify the address of the third byte in PINT? ____________
____

• • •

The address of PINT (leftmost byte address) is increased by 2 to get the address of the third byte.

4. The field length of the first operand is __________ .

• • •

1

5. The field length of the first operand is specified by
__________________________.

• • •

The digit 1 in parentheses.

6. The address of the fifth byte in PINT, where the original quotient sign is located, is specified by
____ .

• • •

adding 4 to the address of PINT.

7. Why aren't the contents of the entire fifth byte transferred by this instruction? ______________
____ .

• • •

The move numerics instruction operates only on bits 4-7 of the designated byte(s).

SKIP OPTION

If you are sure you can predict the results of a move numerics instruction regardless of the symbolic names, field lengths (implicit or explicit) and relative addressing involved, you may skip the following two pages. If you are not sure you can predict the results of move numeric instructions, you should read the following frames.

MOVE NUMERICS INSTRUCTION

Refer to the description of the "move numerics" instruction, which you already have read in this series of frames.

1. MVN is the mnemonic for the "____ _______" instruction.

• • •

"move numerics"

The MVC instruction moved the entire byte from the 2nd operand.

2. The MVN instruction only moves bits ________ (0-3/4-7).

• • •

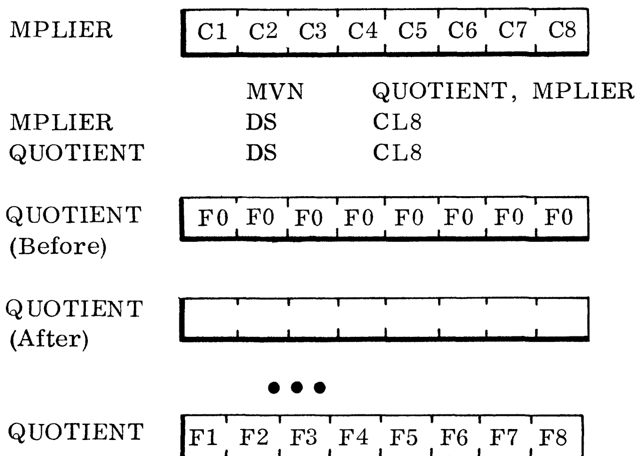
4-7

3. The MVN instruction moves bits 4-7 of each byte from the ________ (1st/2nd) operand to bits 4-7 of the ________ (1st/2nd) operand.

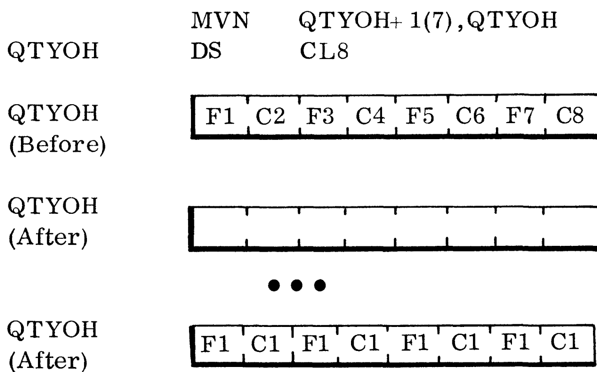
• • •

2nd; 1st

4. Given the following, show the resulting contents of the 1st operand.



5. Given the following, show the resulting contents of the main storage area.



The numeric portion of the leftmost byte was moved (propagated) to the right, byte by byte.

One problem that is often encountered after a multiply operation is the placement of the decimal point. For instance, 0001120C multiplied by 00010C equals 0011200C. However, suppose these numbers represented dollars and cents, such as:

```
$    11.20
       .10
     -----
      0000
     1120
     -----
$   1.1200
```

As you can see, the multiplication resulted in a product with 4 decimal places.

What is usually necessary is shifting the product to the right in order to re-establish the proper place for the decimal point. There are no "shift" instructions for the storage-to-storage operations. However, the "move" instructions can be used to effectively shift storage data.

In our previous example, the product had to be shifted two places to the right in order to maintain the decimal point. For instance:

```
    00011.20C
x       0.10C
    ---------
    001.1200C
```

Assuming we are not interested in the third and fourth decimal places, the above product should look like this: 00001.12C. This can be accomplished by use of the "move numerics" (MVN) instruction followed by a "zero and add" (ZAP). See the example below.

Program to Multiply and to Position the Decimal Point

```
MORE        MP     TOTAL, MPLR
            MVN    TOTAL+2(1),TOTAL+3
            ZAP    TOTAL(4),TOTAL(3)

TOTAL       DS     PL4
MPLR        DS     PL2
```

Storage Contents Before Execution of the Above Instructions

TOTAL	00	01	12	0C
MPLR	01	0C		

Storage Contents

TOTAL (After MP)	00	11	20	0C
TOTAL (After MVN)	00	11	2C	0C
TOTAL (After ZAP)	00	00	11	2C

Any time a packed decimal field is to be shifted an even number of places to the right, the MVN instruction can be used to place the sign next to the low-order digit. As shown previously, the packed decimal field can then be shifted to the right by use of the ZAP instruction.

END OF SKIP OPTION

Returning to the Ajax problem, we see that we have calculated the monthly interest amount, half adjusted it, and given it the proper sign in the proper location.

1. The next block in the flowchart, G2, tells us to calculate the amount of the payment which is to be applied to the principal. We know the amount of the principal and the amount of the payment, and we know how much of the payment goes for interest. We can determine how much of the payment is left to be applied to the principal by (your own words) ____________________

•••

subtracting the monthly interest amount from the payment amount. The remainder is the amount paid on the principal.

The first thing to be done in setting up a subtract operation is to get the number being subtracted from (the minuend) into an accumulator.

2. We could use PINT as the accumulator for the subtract operation and ZAP the minuend into it. However, PINT still contains the monthly interest amount. What happens to the contents of a field when we ZAP another quantity into that field?

____

•••

They are destroyed.

We need the monthly interest amount which still is in PINT. Therefore, if we use PINT as our accumulator we must first store its contents somewhere else. As we will see later, this would require an extra step which is not really necessary.

3. The most efficient way to perform this step is to set up another accumulator. In the list of DC's for the problem you will find a storage area called PAMT. (Figure 15.) This area has been reserved for use as an accumulator. It is (how many) ______ bytes long.

•••

3

4. We will perform our subtract operation in a 3 byte accumulator called ____________ .

•••

PAMT

5. The first step in subtracting is to get the minuend into an accumulator. We are subtracting the monthly interest amount from the payment amount. The ________ ________ is the minuend.

•••

payment amount

6. The symbolic name of the payment amount (in packed form) is _______ . (Consult Figure 15, if necessary.)

•••

PPAY

7. We want to get the payment amount, PPAY, into the accumulator called __________ .

•••

PAMT

8. For this operation, we should use a (ZAP/AP/MVC) ____________ instruction. Why? ______

____

•••

ZAP; To clear the accumulator of old data before putting in the new data.

9. Write the instruction that will put the minuend into the accumulator. ____________________

•••

ZAP PAMT,PPAY

10. The payment amount, whose symbolic name is __________ , is (where) ____________________ after the instruction is executed.

•••

PPAY; in an accumulator called PAMT

11. Our next step is to actually subtract the monthly interest amount from the payment amount, PPAY, which is in the accumulator called PAMT. Consult your Reference Data card and select an instruction that will do this. The instruction is __________ .

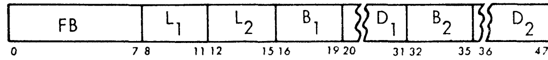
•••

Subtract Decimal (SP)

Read about this instruction in the following material.

Subtract Decimal

SP $D_1(L_1, B_1), D_2(L_2, B_2)$ [SS]



The second operand is subtracted from the first operand and the difference is placed in the first operand.

- Both operands must be in packed format.
- The difference is in packed format.
- If the first operand is too short to contain all the significant digits of the difference, overflow occurs.
- If the second operand is shorter than the first, subtraction will take place normally.
- A field may be subtracted from itself.

Condition Code:

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

Program Interruptions:

Operation
Protection
Addressing
Data
Decimal Overflow

Name (1–8)	Operation (10–14)	Operand (16–25)
	SP	PAMT,PINT(3)

The instruction above will perform the subtract operation for the Ajax problem.

Copy this instruction on your coding sheet.

12. The field length of the second operand is stated explicitly. The second operand is (how many) _____ bytes long.

• • •

3

13. The second operand comes from the accumulator called __________.

• • •

PINT

14. The first three bytes of PINT contain what quantity?

____

• • •

The monthly interest amount.

15. We have subtracted the monthly interest amount from the monthly payment. The difference is in the accumulator called ________.

• • •

PAMT

16. The quantity in PAMT is the amount of the monthly payment that is applied to the (principal/interest) ______________ .

• • •

principal

17. Since the signs of both factors in this operation were positive and the absolute value of the subtrahend was smaller than the absolute value of the minuend, the sign of the result is (positive/negative) _________ .

• • •

positive

18. The condition code will be set to ________ .

• • •

2

SKIP OPTION

If you have any doubts about your ability to predict the results of Subtract Decimal operations, regardless of the symbolic names, field lengths (implicit or explicit) and algebraic signs of the factors involved, you should read the frames on this page. Otherwise, you may skip to the point where we continue with our program.

SUBTRACT DECIMAL INSTRUCTION

Refer to the description of the SP instruction, which you read earlier in this series of frames.

1. The mnemonic for the "subtract decimal" instruction is ___________

• • •

SP

The operation of "subtract decimal" instruction is similar in all respects to the "add decimal" instruction. The only difference is that the AP instruction adds and the SP instruction subtracts.

You may want to refer to pages 108-112 in this volume, if you have any difficulty answering the next few frames.

2. Given the following signs, indicate whether the fields will be true or complement added on an AP instruction. (Select the answers.)

	1st Operand	2nd Operand
a.	+	- True add/Complement add
b.	+	+ True add/Complement add
c.	-	- True add/Complement add
d.	-	+ True add/Complement add

• • •

a. Complement add; b. True add; c. True add; d. Complement add

3. Given the following signs, indicate whether the operands will be true or complement added on an SP instruction. (Select the answers.)

	1st operand	2nd operand
a.	+	+ True add/Complement add
b.	+	- True add/Complement add
c.	-	+ True add/Complement add
d.	-	- True add/Complement add

• • •

a. Complement add; b. True add; c. True add; d. Complement add

4. One use of the "subtract decimal" instruction is to zero out a packed decimal field. Show the resulting contents of the 1st operand for the following SP instruction.

AGAIN1 SP STORE,STORE

STORE(Before) | 42 | 10 | 7D |

STORE(After) | | | |

• • •

STORE | 00 | 00 | 0C | ; Notice that a zero difference results in a plus sign.

5. The SP instruction can also be used to zero out the low order bytes of a field. Show the result of the following instruction.

AGAIN2 SP SET,SET+1(3)

SET (Before) | 41 | 67 | 42 | 7D |

SET (After) | | | | |

• • •

SET | 41 | 00 | 00 | 0D | ; Notice that since only part of the field was zeroed out, the sign remained minus.

6. What would happen on the following instruction?

AGAIN3 SP LUMP,LUMP(3)

LUMP | 65 | 43 | 21 | 0C |

Your response: ______________________________

• • •

A data exception would be recognized and a program interrupt would occur. This occurs because the 2nd operand's low-order byte contains 21. Bits 4-7 of this byte would be recognized as an invalid sign code.

1st operand

LUMP | 65 | 43 | 21 | 0C |

2nd operand

| 65 | 43 | 21 |

END OF SKIP OPTION

Earlier we said that we could have used PINT as our accumulator for the preceding subtract operation, but that it would have required an extra step. We should avoid unnecessary program steps. Each instruction requires a certain amount of time, however small, for execution. The sum of the execution times for a large number of unnecessary instructions can be significant, costing valuable computer time.

To use PINT as the accumulator we would have to:

- Move the monthly interest amount from PINT to another location in storage.
- ZAP the monthly payment amount into PINT.
- Subtract the monthly interest amount from PINT.

By setting up another accumulator we eliminated the first step above. Our program would be:

- ZAP monthly payment amount into a different accumulator.
- Perform the subtraction.

1. Returning to the Ajax Company problem we find we have, in the accumulator called PAMT, the amount of the payment that is to be applied to the principal. Our next step is indicated in block H2 of the flowchart. It is to ____________________.

 • • •

 calculate the new principal.

2. To calculate the new principal we must subtract from the old principal the amount of the monthly payment that is to be applied to the old principal. Where is this amount? ____________________________

 • • •

 In accumulator PAMT

3. PAMT, in this case, contains the __________ (minuend/subtrahend) of the subtract operation we must perform.

 • • •

 subtrahend

4. The (subtrahend/minuend) ___________ goes into the accumulator first.

 • • •

 minuend

5. The minuend in this operation will be (symbolic name) _____________.

 • • •

 PPRIN

6. To perform this subtraction it will be most efficient to (choose one):
 1. Store the contents of PAMT in another location and use PAMT as the accumulator.
 2. Set up another accumulator.

 • • •

 2

7. Your list of DS's in Figure 15 shows a storage area called PNEWPR which we will use for an accumulator. Write an instruction that will place the minuend of this subtraction into PNEWPR.

 • • •

 ZAP PNEWPR,PPRIN

8. Now write the instruction that will subtract the subtrahend. __________________

 • • •

 SP PNEWPR,PAMT

9. We have completed the arithmetic operations in the Ajax problem. The next block in the flowchart, J2 tells us to ______________________________ .

 • • •

 assemble a line

10. The format of the printed line is not quite the same as in the previous version of the problem. Look at the DS statements in Figure 14 that specify the output area. The first item to be printed in the line is ______ ________ , for which the symbolic name is _____________ .

 • • •

 account number; ACTNUM

11. This involves a direct move of a data item from the input area to the output area. The data item has the symbolic name ___________ in the input area.

● ● ●

ACCTNO

12. Write an instruction that will move this data item from the input area to the output area. If necessary, consult your Reference Data card for the correct instruction.

● ● ●

MVC ACTNUM, ACCTNO

13. The next item in the printed line is called ________ .

● ● ●

OLDPRI

OLDPRI is the symbolic name of the field in the output area from which the original principal will print. It is a quantitative figure that we want printed with appropriate punctuation. But numeric quantities in storage do not carry any punctuation - the punctuation is only implied. So we must supply it at the time a quantity is printed.

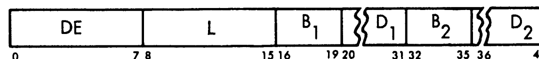
To simplify the punctuation of data that is to be printed, a decimal instruction called Edit is used. Before you read about this instruction you should know the following terms:

- Source - the data being edited.
- Pattern - an arrangement of characters in a field in storage, that determines what will be done with each character in the data being edited. It also specifies what punctuation will be inserted in the data, and where.

Now read about the Edit instruction in the following material. In addition, read the frames that discuss the Edit instruction further.

Edit

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



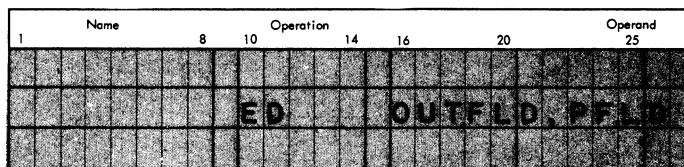
The format of the second operand (source) is changed from packed to zoned and is edited into the pattern in the first operand.

- The second operand (source) must be in packed format.
- Editing proceeds left to right one character at a time.
- The edited result replaces the pattern.

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

Program Interruptions:
- Operation
- Protection
- Addressing
- Data

EXAMPLE:



EDIT INSTRUCTION

The purpose of edit operations is to produce easy-to-read documents by inserting the proper punctuation into a data record. The data to be edited is called the source field and must be in the packed decimal format. Consider the following source field

00	12	49	07	10	7C

Source Field

In its present format, the preceding field cannot be printed. The data must be in zoned format before being printed.

1. One of the functions of the edit operation is to change a source field from the _______ format to the __________ decimal format.

•••

packed; zoned

If changing from the packed to the zoned format were all that was necessary to produce a meaningful report, the "edit" instruction wouldn't be necessary. The "unpack" instruction, which you previously studied, would be sufficient. For instance, if the previous packed decimal operand were changed to the zoned format, it would look like this:

Packed	00	12	49	07	10	7C					
Zoned	F0	F0	F1	F2	F4	F9	F0	F7	F1	F0	C7

If the above zoned decimal field were printed, it would look like this:

0 0 1 2 4 9 0 7 1 0 G

By examining the printed document, you could tell by looking at the low-order character (G) that it was a positive number with a low-order digit of 7. However, the printed document is still not too meaningful. Perhaps the number represents money. It would be better if it could look like this:

$1,249,071.07

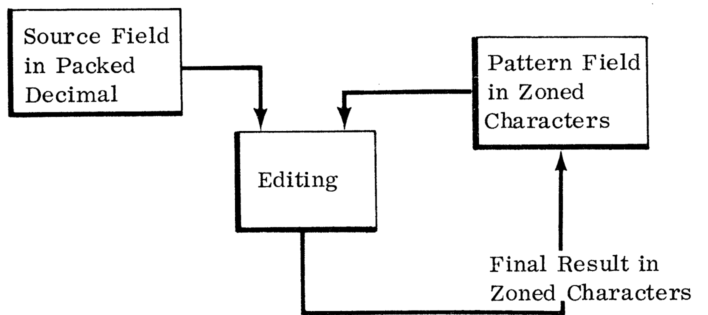
Along with other changes to the data, this would require inserting the commas and decimal points in the right places. This is another function of the edit operation.

2. The edit operation will change a packed decimal field which is called the _________ field, into the zoned format and insert the necessary punctuation characters.

•••

source

The edit operation consists of moving the source field (the data to be edited) into a pattern field. The pattern field will be made up of ZONED characters that will control the editing. The final edited result will replace the PATTERN field.



3. During an edit operation, the _______ field is edited under control of the ___________ field.

•••

source; pattern

4. The source field contains packed decimal data while the pattern field contains _________ characters.

•••

zoned

5. The edited result will replace the __________ field.

•••

pattern

The pattern to be used normally is set up as a hexadecimal constant by a DC statement. It is given a symbolic name and is kept in storage. If the pattern field is to be used more than once, it must be moved to a storage work area before each use. The Move Characters instruction can be used for this purpose.

The reason the pattern field must be moved to a work area before each use is that it is destroyed during the edit operation.

6. Although the pattern field is destroyed by the edited result, the original pattern is retained as a constant in another location of ______________ .

•••

storage

7. For edit operations the pattern field should be moved to a _______ area if it is to be used more than once.

•••

work

8. The MVC instruction can be used to move the __________ ______ to a work area before the edit operation begins.

•••

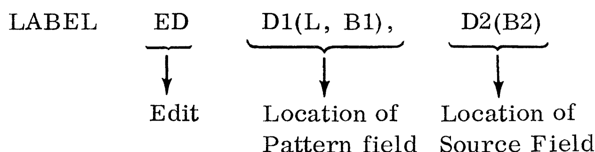
pattern field

9. The pattern field should be moved to a work area by the MVC instruction prior to doing the actual edit operation. This is because the pattern field is _________________ by the edit operation.

•••

destroyed

The edit instruction uses the SS format as shown below:



10. As you can see above, the source field is the _________ (1st/2nd) operand.

•••

2nd

11. Like most instructions, the results of the edit operation replace the 1st operand, which is the ________ field.

•••

pattern

12. Prior to the edit operation, the pattern field should be moved into a work area. This work area then becomes the (1st/2nd) _________ operand in the edit instruction.

•••

1st

The pattern field, which is the first operand of the edit instruction, is made up of certain characters. Each of these characters serves a specific purpose in the edit operation. You will learn later what these characters are and the purpose of each.

As the edit operation proceeds, each of the characters in the pattern is examined. Depending on what the character is, it will be treated in one of three ways. The first of these three ways is this:

> The pattern character may be replaced by a special pattern character called the fill character. The fill character is located in the high order (leftmost) byte of the pattern field.

13. A pattern character can be treated in one of (how many) ________ ways.

•••

three

14. One of these is replacement by the _____ character.

•••

fill

The fill character is located in the high order (leftmost) byte of the pattern. Replacement by the fill character is one of three ways a pattern character can be treated during the edit operation.

15. The fill character (is/is not) _____ located in the pattern itself.

•••

is

16. The fill character is located in the ______________ byte of the pattern.

•••

high-order/leftmost

The characters in the pattern field determine the editing that will take place. The high-order (leftmost) character in the pattern field is called the fill character. The fill character will be substituted in the pattern for any source or pattern character we don't want to print.

17. The fill character is in the _____________ byte of the ___________ field.

•••

leftmost (high-order); pattern

18. Suppose there are some high-order zeroes in the source field that we don't want to print. The high-order byte in the pattern field contains a blank. We will get (blanks/zeroes) _______ in the printed output.

●●●

blanks

For many edit operations the blank is used as a fill character. The bit structure for a blank is 0100 0000. This also is the bit structure of the hexadecimal number 40.

19. Any of the 256 possible characters can be used as a fill character. However, in many operations the fill character is the hexadecimal number ______ , whose bit structure is ___________ .

●●●

40; 0100 0000

20. When the fill character in an edit pattern is the hexadecimal number 40 (bit pattern 0100 0000), a __________ will be substituted in the pattern field for any source or pattern character we don't want to print.

●●●

blank

By using a blank (hex 40) as a fill character, high-order zeroes in the source field can be blanked out and will not print.

21. As mentioned previously, any character can be used as a fill character. The asterisk (*) often is used as a fill character, to afford check protection. If the asterisk is used, all high-order zeroes in the source field will be replaced by __________ .

●●●

asterisks

Asterisks commonly are printed on checks in place of high-order zeroes. This makes the amount of the check easy to read and at the same time prevents the insertion of high-order digits on the check to change its value.

In the following frames we will represent a blank in an edit pattern by the hexadecimal number 40. Other characters in the pattern also will be represented by their hexadecimal number equivalents.

22. The hexadecimal number 40 in an edit pattern represents a __________ .

●●●

blank

Besides the fill character, there are three more characters used in edit patterns. Each has a special meaning. These characters are:

- The Digit Select character.
- The Significance Start character.
- The Field Separator character.

These three characters can appear anywhere in the pattern field.

The digit select character has a bit structure of 0010 0000. This is the bit structure of the hexadecimal number 20.

23. A hex number 20 (bit structure 0010 0000) in an edit pattern represents the d____ s_____ character.

●●●

digit select

24. In the following frames we will represent the digit select character by the hexadecimal number 20. The hexadecimal number 20 and the digit select character have (the same/different) ________ bit structure(s).

●●●

the same

25. An edit pattern containing the hexadecimal number 40 in its leftmost byte and the hexadecimal number 20 in its remaining bytes is composed of what pattern characters?

●●●

the fill character; the digit select character

Digit select characters in the pattern are always replaced. They will be replaced by either the fill character or by a digit from the source field.

26. Assume a given pattern contains six digit select characters. After the edit operation (how many) ________ digit select characters will remain in the pattern.

●●●

no

Digit select characters are replaced either by the fill character or a digit from the source field.

For each digit select character in the pattern field there is a corresponding digit in the source field. When, during the edit operation, a digit select character is encountered, it will be replaced by the corresponding source field digit, unless we do not want the source digit to print.

27. If the source digit is not to print, the digit select character is replaced by ___ ___ __________.

•••

the fill character

28. If the source digit is to print, the digit select character is replaced by ___ _____ ________.

•••

the source digit

29. What two characters can replace a digit select character? ________________ ________________

•••

a digit from the source field; the fill character from the pattern

Replacement by the source digit is the second of three ways in which a pattern character can be treated during the edit operation. The first is replacement by the fill character.

30. Suppose the fill character is a blank and the source field has three high-order zeroes that we want to suppress; that is, we do not want them to print. Following the fill character in the pattern there are three digit select characters, so that for each high-order zero in the source field there is a corresponding digit select character in the pattern. These digit select characters will be replaced by ___ ___ _______ and the printed output will have _________ in the three high-order positions.

•••

the fill character; blanks

31. Suppose there are two high order zeroes followed by two significant digits in the source field. The pattern fill character is a blank and there are four digit select characters immediately following it. The first two digit select characters will be replaced by _____ and the next two by __________ ____________________.

•••

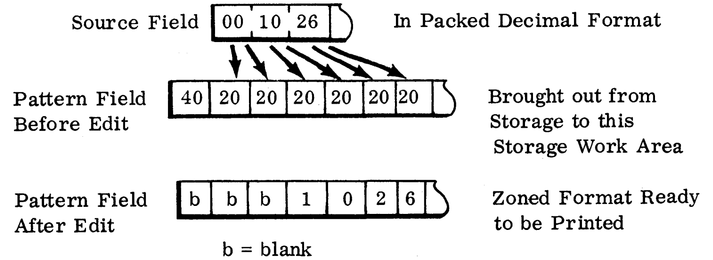
blanks; significant digits from the source field.

32. Name two ways a digit select character in a pattern field can be treated during an edit operation. ______ ________________________________

•••

It can be replaced by the fill character. It can be replaced by a digit from the source field.

Let's look at an example of the use of the fill character and the digit select character.



33. Prior to the edit instruction, the pattern was brought out of the constant storage area and put in the p________ f________ .

•••

pattern field

34. The leftmost position of the pattern field contains the fill character which in this example is a _______. The remaining positions of the pattern field contain _________ _____________ characters.

•••

blank; digit select

35. The source field contains high-order zeroes which must be edited out. This field is in the (packed/zoned) ____________ format.

•••

packed

36. The edit operation begins by examining the fill character. After determining what the fill character is, the edit operation leaves the fill character in the pattern and moves to the second pattern character. The second pattern character in our example is a ________ __________ character.

•••

digit select

37. Since the first digit in the source field is a nonsignificant (zero) digit, the first digit select character in the pattern will be replaced by _______ ____

• • •

the fill character (a blank)

38. Next, the edit instruction looks at the third position of the pattern field. The previous operation is repeated. The third position of the pattern field ends up with a (blank/digit) ____________ .

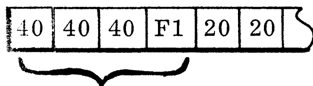
• • •

blank

39. The edit instruction looks at the pattern field's fourth position, finds another _______ _______ character, which tells it to look at the source field. A significant digit is found and moved into the pattern field.

• • •

digit select



The first four bytes of the pattern field now contain blank - blank - blank - 1. The first blank is the original fill character, still in place. The next two blanks are fill characters that replaced two digit select characters. The 1 is a significant digit from the source field that replaced a digit select character in the pattern.

Note that digit select characters in the pattern have been replaced by both the fill character and by a significant digit from the source. This demonstrates the two ways in which a digit select character can be replaced.

The fact that a significant digit was found must be remembered by the system so that all remaining digits in the source field, including zeroes, can be put into the pattern field.

We know that digit select characters are replaced by either a source digit or the fill character. We also know that after the first significant digit has been encountered in the source field we normally want all remaining source digits to print. This includes zeroes.

Obviously, the system needs some way of knowing which to choose: the source digit or the fill character.

This is determined by a remembering device called the S trigger.

The S trigger is part of the system circuitry.

The S trigger can be set to one of two states: the zero (0) state or the one (1) state.

40. The S trigger tells the system to supply either the _____ _____________ or the ________ _________ .

• • •

fill character; source digit

When set to 1, the S trigger indicates that all unexamined digits from the source field are significant. As a result, the digit select characters in the pattern field are replaced with the digits from the source field.

At the beginning of the edit operation, the S trigger is set to 0. As long as the S trigger is 0, the digit select characters in the pattern field are replaced with the fill character.

41. What determines whether a digit select character is replaced with a source digit or with the fill character? _____ __ ___________

• • •

The S Trigger

42. At the beginning of the edit operation, the S trigger is set to _________ (1/0).

• • •

0

43. When the S trigger is set to 0, a digit select character in the pattern field is replaced with _____ _____ __________ .

• • •

the fill character

44. When the S trigger is set to 1, a digit select character in the pattern field is replaced with _____ ____ .

• • •

the digit from the source field

Both the source field and the pattern field are processed left to right, a character or a digit at a time. Each time the digit from the source field replaces a digit select character, the 4-bit digit has the proper zone bits inserted.

The S trigger is set to 0 at the beginning of the edit operation. It is set to 1 by one of two methods.

- A significant (non-zero) digit in the source field.
- A significance start character in the pattern field.

The significance start character has a bit pattern of 00100001 (hex 21). This bit pattern has no character symbol. In the following frames we will use the hexadecimal number 21 to represent the significance start character in edit patterns.

45. Which hexadecimal number is used to represent each of the following?
    Blank = _____
    Significance Start Character = ____
    Digit Select Character = ____

• • •

Blank = 40; Significance Start Character = 21;
Digit Select Character = 20

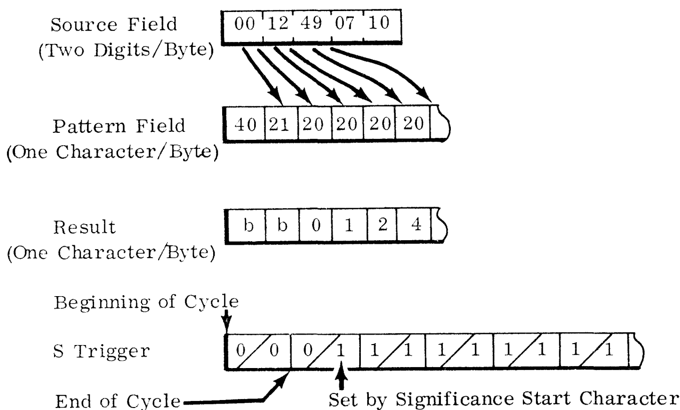
46. What two characters can set the S trigger to 1?
    1. ____
    2. ____

• • •

1. A non-zero digit from the source field.
2. A significance start character in the pattern field.

A significance start character is replaced (as was the digit select character) by either a digit from the source field or the fill character.

For example:



The edit operation begins by examining the fill character, which is left in place in the pattern field. Then the next pattern character is examined. In the previous example, this was a significance start character. The high-order source digit is then examined. Because the source digit is zero and the S trigger is 0 (at this time), the significance start character does set the S trigger to 1 so that all subsequent source digits are significant. The remaining pattern characters in our example are digit select characters which are replaced with source digits.

47. Given the first few characters of a source and pattern field below, show the resulting contents of the pattern field after editing.

00	12	49	07	10	7C	Source Field
40	20	20	20	20	20	Pattern Field
						Result

• • •

b b b 1 2 4

In the previous problem, there was no significance start character. As a result, the two high-order zeroes from the source field did not go into the pattern field. The fill character was used instead. Once significance was started, the remaining pattern characters were replaced by source digits.

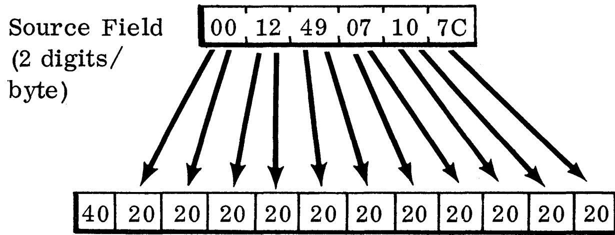
Once significance is started, the S trigger will remain on until one of two things happen:

- The sign of the source field is examined and is plus.
- A field separator character (00100010) is recognized. This is the bit structure of the hexadecimal number 22.

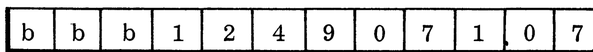
The field separator character is used when two or more packed decimal source fields are to be edited into a pattern with one instruction field. We'll examine this later. For now, let's discuss the handling of the sign.

Since the sign is in the right half of the source field's low-order byte, it can be examined at the same time as the low-order digit is examined. The sign itself is skipped over but, if plus, the S trigger is set to zero. The following will illustrate this:

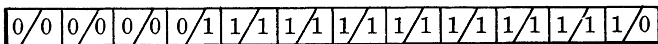
Source Field
(2 digits/
byte)



Pattern Field (1 Character/Byte)



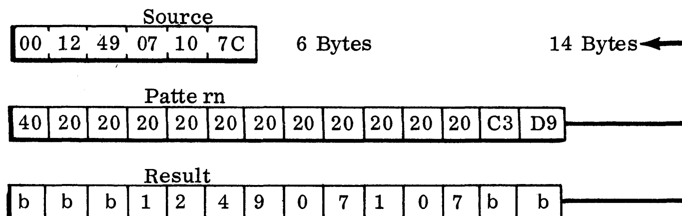
Result



S Trigger

When a pattern character is examined and is not one of the three special control characters, the source field is not examined and the pattern character is left in place if the S trigger is 1. Otherwise, it is replaced by the fill character.

For example, a common method of indicating a negative quantity in a printed report is with the letters "CR". If we take another look at the previous example and add the CR symbol (hex numbers C3 and D9), this would be the result:



Because the plus sign sets the S trigger to 0, the remaining pattern characters (CR) were replaced by the fill character. If the sign of the source field had been minus, the "CR" would have been left in the pattern field.

Let's take the following source field and produce the edited result.

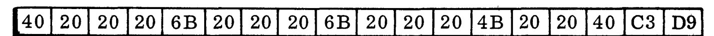
Source 00 12 49 07 10 7D

Edited Result b b b 1, 2 4 9, 0 7 1. 0 7 b C R

Note that the pattern which produces this edited result must have commas and a decimal point, as well as the letters CR and the special control characters.

The original pattern would look like this:

Pattern:



48. You know that each of the numbers in an edit pattern is a hexadecimal number whose bit structure represents one of the pattern characters. Compare the pattern above with the edited results it produced in the previous frame. Which of the hexadecimal numbers represents the comma? _____ The period? ______

• • •

6B; 4B

Notice that the commas, decimal point, and the CR were left in the pattern and not replaced by source characters. This occurred because the S trigger was set to 1 and remained there.

Previously we said there were three ways a pattern character could be treated during the edit operation. Two of them were replacement by the fill character and replacement by a source digit. You have just seen the third. It is: the pattern character is left undisturbed in the pattern to print as itself.

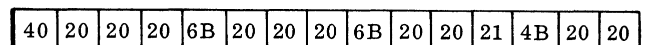
49. Name the three ways a pattern character can be treated during an edit operation. _______________

____

• • •

replacement by a source digit; replacement by the fill character; left undisturbed to print as itself.

50. The hexadecimal number used to represent the comma in edit patterns is 6B. The period is represented by 4B. The following pattern is used to edit a dollar and cents field. It will have (how many) ___commas and (how many) _____decimal points in the edited result. (Assume no high-order zeroes in the source field.)



• • •

2; 1

51. The comma is represented by the hex number ____.
The period by the hex number _____ .

●●●

6B; 4B

52. Given the following, show the edited result.

Source	00	14	71	3C									
Pattern	40	20	20	6B	20	20	20	4B	20	20	40	C3	D9
Result	- - - - - - - - - - -												

●●●

b b b b 1 4 7 . 1 3 b b b

Of course, the blanks in the previous answer won't print in the final printed report which would look like this:

147.13

53. Let's review the settings of the S trigger during edit operations. When the trigger is set to zero, high-order zeroes in the source field (will/will not) __________ be printed.

●●●

will not

54. If the S trigger is off, the first significant (non-zero) character encountered in the source field (will/will not) ________ turn the trigger on.

●●●

will

55. If a significance start character is encountered in the pattern before the first non-zero character is encountered in the source, the S trigger (will/will not) ________ be turned on.

●●●

will

56. Assume a plus amount is being edited into a pattern which includes the characters CR. The CR, you recall, prints if the source is minus. The S trigger will be (on/off) _____ at the time the CR is encountered in the pattern.

●●●

off

57. Assuming that, in the previous example, the S trigger was turned on by either a significance start character in the pattern or a significant character in the source, what turned it off before the CR was reached?

____

●●●

The + sign of the source

58. A + sign in the source field turns the S trigger off to prevent printing the CR. The CR normally prints for ______________________________ amounts.

●●●

negative, minus, (-)

59. Assume two contiguous source fields are to be edited for printing. A single pattern can be set up and a single edit instruction used to perform the operation. The two parts of the pattern would be separated by a (field separator/digit selector) ______ __________ character.

●●●

field separator

60. Since the S trigger should be set to zero (off) for each source field being edited, you can deduce that the field separator character will set the S trigger (on/off) ________ .

●●●

off

The field separator character has the bit structure 0010 0010. This is also the bit structure for the hexadecimal number 22. There is no character for this bit pattern on any of the System/360 printers. The field separator character will be replaced by the fill character whenever it is encountered in a pattern.

61. In the following pattern, which is the field separator character?

40	20	20	20	4B	20	20	22	40	20	20	6B	20	20	20	4B	20	20

●●●

22

62. Which pattern character replaces the field separator character? ___ _____ ______________ .

●●●

the fill character

63. Which of the following can set the S trigger to 1 (on)? ______ To 0 (off)? ______
   a. Digit Select Character
   b. Significance Start Character
   c. Plus sign in source field
   d. 1st non-zero character in source field
   e. Field Separator Character

• • •

b, d; c, e

64. A digit select character in the pattern is replaced by the _____ ___________ if the S trigger is 0 or by a ________ _______ if the S trigger is 1.

• • •

fill character; source digit

65. A significance start character in the pattern is replaced by the ___ __________ if the S trigger is 0 or by a ______ _____ if the S trigger is 1.

• • •

fill character; source digit

66. A field separator character always sets the S trigger to 0 and is replaced by the _____ ______________ .

• • •

fill character

67. Characters in the pattern other than the control characters are either replaced by the ___ ________ if the S trigger is 0 or are ____ __ _________ .

• • •

fill character; left in place

68. Show the results of the following "edit" instruction.

EDIT1 ED PATTERN(13),SOURCE

PATTERN | 40 | 20 | 20 | 6B | 20 | 20 | 21 | 4B | 20 | 20 | 40 | C3 | D9 | (characters)

SOURCE | 0 7 | 9 4 | 7 6 | 9 C | (digits and sign)

PATTERN (after) ____________________

• • •

b b 7 , 9 4 7 . 6 9 b b b

69. Show the result of the following "edit" instruction.

EDIT2 ED PATTERN(13),SOURCE

PATTERN | 40 | 20 | 20 | 6B | 20 | 20 | 21 | 4B | 20 | 20 | 40 | C3 | D9 | (characters)

SOURCE | 0 0 | 0 0 | 0 6 | 9 D | (digits and signs)

PATTERN (after) ______________

• • •

b b b b b b b . 6 9 b C R

70. Referring to the previous problem, show the result for the following pattern.

Pattern | 40 | 20 | 20 | 6B | 20 | 20 | 20 | 4B | 20 | 20 | 40 | C3 | D9 |

Pattern (after) ________________

• • •

b b b b b b b b 6 9 b C R; Note that a significance start character should be in the pattern to protect the decimal point in case the amount is less than a dollar.

71. Show the result of the following "edit" instruction. Note that 5C is the hex number whose bit structure is the same as an asterisk.

EDIT3 ED PATTERN(13),SOURCE

Pattern | 5C | 20 | 20 | 6B | 20 | 20 | 21 | 4B | 20 | 20 | 40 | C3 | D9 |

Source | 0 0 | 0 0 | 0 6 | 9 C |

Pattern (after) __________________

• • •

* * * * * * * . 6 9 * * *; Note that the use of an asterisk as the fill character will provide asterisk check protection.

72. You will remember that the bit structure of the hex number 22 is used as the field separator character in edit patterns. The following edit instruction will edit multiple adjacent source fields. Show the result.

EDIT4 ED PATTERN(20),SOURCE

Pattern | 40 | 20 | 20 | 21 | 4B | 20 | 20 | 40 | C3 | D9 | 22 | 20 | 20 | 20 | 4B | 20 | 20 | 40 | C3 | D9 |

Source | 0 1 | 7 7 | 6 C | 0 0 | 0 0 | 0 D |

Pattern (after) ________________

•••

b b 1 7 . 7 6 b b b b b b b b b b b b b
Note that the field separator character again sets the S trigger to zero. No significant digits were found in the 2nd source field. As a result, the pattern characters were replaced by the fill character (blanks).

Up to this point you have seen that the "edit" instruction can be used to:

- Eliminate high-order zeroes.
- Provide asterisk protection.
- Handle sign control (CR).
- Provide punctuation.
- Blank out an all-zero field.
- Edit multiple adjacent fields via the field separator character.
- Protect the decimal point by use of the significance start character. This character can also be used to retain the high-order zeroes when desired.

We have seen that every character in an edit pattern can be expressed as a hexadecimal number. For example, the bit structure for a blank is 0100 0000; this is also the hexadecimal number 40. The bit structure for a comma is 0110 1011; this is also the bit structure for the hex number 6B.

73. You can deduce that a convenient way to set up edit patterns in storage is as (character/hexadecimal/packed decimal) ________________ constants.

•••

hexadecimal

There is one other instruction of this type, but it is not illustrated by the sample programs. It is:

Instruction Name	Mnemonic	Basic Function
Edit and Mark	EDMK	Same as for Edit. In addition, the address of the first significant digit is placed in register 1.

Self-study material on this instruction can be found in the Appendix.

74. In Figure 15 there are two DC statements that set up hexadecimal constants for use as edit patterns. These statements are (define storage (DS)/define constant (DC)) ____________________ statements.

•••

define constant (DC)

75. The DC statements that set up hexadecimal constants have a series of hexadecimal numbers in their operands. These numbers are enclosed in (parentheses/single quote marks) ______________________ .

•••

single quote marks

76. A hexadecimal constant is set by a DC statement having as its operand a series of hexadecimal numbers enclosed in single quote marks. These numbers are preceded by the letter _____ .

•••

X

77. A hexadecimal constant is set up by a _____ statement having as its operand a series of __________ numbers, enclosed in ____________________ and preceded by the letter ____ .

•••

DC; hexadecimal; single quote marks; X

40 = blank	4B = period
20 = digit select character	5C = asterisk
21 = significance start character	C3 = C
6B = comma	D9 = R

78. Using the above hexadecimal numbers, write the statement that will set up the edit pattern that produced the following edited result from a source field of 11 digit positions and sign:

    b b b b b 1 4 8 , 9 6 3 . 4 2

Assume the source data will never be negative. Any position in the source field may contain a significant digit. The fill character in the edit pattern should be a blank. The edit pattern should force the printing of the decimal point and the two following digits, even when they are zeroes. No dollar sign should be printed. ______________________

• • •

DC X'40 20 20 20 6B 20 20 20 6B 20 20 21 4B 20 20'

79. Write the statement that will set up the edit pattern for this dollar and cents field, which can have a negative sign:

    0 0 0 2 2 4 7

The fill character should be a blank. We will not print a dollar sign. Print the decimal point and the last two digits of the source.

• • •

DC X'40 20 20 6B 20 20 21 4B 20 20 40 C3 D9'

80. Write the statement that will set up the edit pattern for the following dollar and cents field:

    0 0 0 0 4 2 9 6 5

Use an asterisk as the fill character. The field will not be negative. Show the edited result as it will print. We will not print a dollar sign.

• • •

DC X'5C 20 20 20 20 6B 20 20 21 4B 20 20'
* * * * * * 4 2 9 . 6 5

In the list of DC's in Figure 15 you will find a hexadecimal constant called PATRN1. This constant is the pattern we will use to control the editing of the old principal, PPRIN, into its place in the printed line.

81. The Edit instruction moves the source into the pattern. If we edit our source, PPRIN, into the pattern where the pattern now is stored, what will happen to the pattern? ______________________

• • •

It will be replaced by the edited result.

82. We do not want to lose the pattern because we will use it for every record that is processed by this program. What should we do to preserve the pattern in its original form? ________________

• • •

Move it to another area and do the editing in that area.

83. Which of the following is the more efficient?
    1. Move the pattern to another area.
    2. Edit the source data into that area.
    3. Move the edited results to the output line.

or:

    1. Move the pattern to the field in the output area from which the edited data is to print.
    2. Edit the source data into the pattern in this field.

• • •

The second alternative is the more efficient.

84. The field in the output area, from which the old principal will print, is called OLDPRI. Write the instruction that will move the pattern into this field in the output area.

• • •

```
MVC    OLDPRI,PATRN1
```

85. The pattern now is in the output area, in the field called OLDPRI. Write the instruction that will edit the old principal into the pattern in the output area.

• • •

```
ED    OLDPRI,PPRIN
```

Correct your answer, if necessary.

86. Assuming that PPRIN contains the following quantity:

    0 3 1 7 5 2 5 C

show the edited result. ________________

• • •

b b 3 , 1 7 5 . 2 5

87. If PPRIN contained all zeroes, the edited result would be ______________________________ .

• • •

b b b b b b b . 0 0

The old principal is now in the output area, correctly punctuated for printing. Note that once we have moved the edit pattern (PATRN1) into the output field called OLDPRI we must use the name OLDPRI in the edit instruction. Even though OLDPRI contains data which is elsewhere in storage with its own unique name, if we want to use that data while it is stored in OLDPRI we must use the name OLDPRI. If we were to use the name PATRN1 we would address the pattern in its original location and it would be destroyed. We must always address fields in storage by the names assigned to them when they were defined.

88. Incidentally, the old principal is in storage in another location, the input area, where it has the symbolic name PRIN. Why could we not have used this field as the source data? ________________

____

• • •

Data being edited must be in packed form. PRIN is not packed.

89. The next field in the output area is called _________ (Refer to the output area DS statement.)

• • •

NEWPRI

90. You can guess that the amount being printed from this field is the _____ ________________ .

• • •

new principal

91. Where is the new principal stored? (Refer to previous instructions if you have forgotten.)

____

• • •

In the accumulator called PNEWPR

92. The old and new principal amounts are both dollar and cents quantities. You can guess that they will be punctuated (the same/differently) ___________ and that we will use (the same/a different) ____________ pattern.

• • •

the same; the same

93. Write the instructions that will move PATRN1 into the output field for the new principal, and edit the new principal amount into it.

• • •

```
MVC    NEWPRI,PATRN1
ED     NEWPRI,PNEWPR
```

Each of the remaining data items that will print from the output area are to be edited using another pattern. The symbolic name of this pattern is PATRN2. We will follow the same procedure as before, which is to move the pattern into the output field and edit the source data into it.

94. The next output field is called ________________ .

• • •

MONPAY

95. Write the instruction that will put PATRN2 in MONPAY.

• • •

```
MVC    MONPAY,PATRN2
```

96. MONPAY is the symbolic name for monthly payment, which is in storage under another symbolic name. Locate this field and write the instruction that will edit it into the output field.

• • •

```
ED     MONPAY,PPAY
```

97. Assume the amount of the monthly payment is $49.95. Show the edited result after the previous instruction has been executed.

• • •

b b 4 9 . 9 5

98. The two remaining fields in the output area are MONINT (monthly interest amount) and MONAMT (the amount of the payment applied to the principal). Write the instructions necessary to move PATRN2 into these fields and edit the appropriate data items into them.

• • •

```
MVC    MONINT,PATRN2;    ED    MONINT,PINT;
MVC    MONAMT,PATRN2;    ED    MONAMT,PAMT
```

99. We have completed all the processing steps for the Ajax problem and have only to write the printed record out. Write the instruction that does this.

• • •

```
BAL    10,WRITE
```

100. Now the flowchart tells us to branch back to the instruction that obtains another input record. This is an unconditional branch. Write this instruction.

• • •

```
BC    15,NEXT
```

Let's review briefly what you have done in coding the Ajax Company problem, using the decimal instructions. Refer to your coding sheets, as we go along.

- You learned that data can be processed in packed format, using decimal feature instructions. Such data need not be converted into binary after being packed.
- You defined the input and output areas required, and named the data fields within those areas.
- You wrote the move instructions (MVC), which you already had learned, to move header lines into the output area. You wrote the branch instructions that sent the program to the write routine to print the header lines.
- You wrote instructions to pack the input data.
- You wrote assembler instructions to set up work areas (for use as accumulators) as needed.
- You learned about specific decimal feature instructions that operate on data in packed decimal form, and were shown how the appropriate instruction operated on the data for the Ajax Company problem. These instructions included:
  - ZAP - Zero and Add
  - MP - Multiply Decimal
  - DP - Divide Decimal
  - AP - Add Decimal
  - SP - Subtract Decimal
  - ED - Edit
- You learned that good housekeeping in coding is important. Machine instructions should be written sequentially on one set of coding sheets; assembler instructions on another set.
- It was demonstrated to you that the coding you write must follow the program flowchart exactly. The specific operations called for by the flowchart, in the sequence it specifies, must be reflected in the instructions you write.

GO ON TO THE THIRD SAMPLE PROGRAM

SOME ADDITIONAL BRANCHING AND LOGIC INSTRUCTIONS IN SYSTEM/360 ASSEMBLER LANGUAGE

You have learned and used many of the instructions in the System/360 commercial instruction set. In the next series of frames you will learn some additional instructions in the standard set, and you will use some of the instructions you already have learned.

Read the problem statement in Figure 17 of the Sample Program Book, scan the I/O Coding in Figure 18, and study the program flowchart in Figure 19.

Look back at Figure 18. This group of assembler and machine instructions precedes those you are going to write. The macro instructions DTFCD and DTFPR are declaratives that describe the input and output files.

1. The name of the input file is _________ . Data in this file is recorded on (tape/disk/cards) ________ .

   ● ● ●

   CARDIN; cards

2. The output file is (printed/punched) __________ . Its symbolic name is ____________ .

   ● ● ●

   printed; ALINE

3. The machine instructions starting with BEGIN establish general register _______ as the base register for the program.

   ● ● ●

   11

4. After the files are opened, the program branches to an instruction named ____________ .

   ● ● ●

   START

5. Your first instruction will be called START. According to the flowchart in Figure 19, the function of the instruction is to (your own words) ____________________ .

●●●

Turn on a switch.

6. You will recall that a computer program may have both mechanical and programmed switches. Mechanical switches require manual action to set them. Since we will use an instruction to set the switch, it will be a ____________ switch.

●●●

programmed

7. Programmed switches are reserved storage locations whose contents are changed according to the setting desired, and then tested at the appropriate points in the program. To be sure that the switch location will not be used for any other purpose, we must (your own words) ______________ the location.

●●●

reserve

8. You know that storage areas may be reserved by writing a DS, which simply reserves the storage, or a DC, which not only reserves it but establishes its initial contents. On a separate coding sheet, write a DC or a DS, whichever you choose, to set aside a one byte location. The symbolic name for the switch is in the list of work areas and constants in the problem statement.

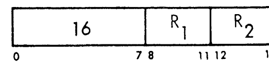
●●●

SW DS CL1 or SW DC CL1' '
Note that if a DC is used, the initial switch content will be whatever character you enclose in single quote marks in the operand.

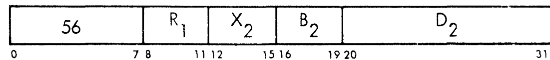
We have reserved a switch location called SW. The instruction we will use to set the switch has the mnemonic operation code OI. It is one of several similar instructions some of which use "AND" logic, and some of which use "OR" logic. Read about the OI instruction in the following material and locate the group that contains it on your Reference Data card.

OR

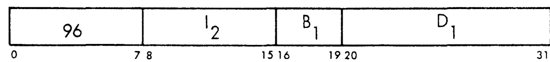
OR R_1, R_2 **[RR]**



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



OI $D_1(B_1), I_2$ **[SI]**



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

All of above:
The first and second operands are examined on a corresponding bit by bit basis.

- If either or both of the corresponding bits are ones, the result is a one and replaces the bit in the first operand.
- If both bits are zeros, the result is a zero and replaces the bit in the first operand.

O only:
- The second operand is a fullword and must be on a fullword integral boundary.

OI only:
- The second operand is one byte (8 bits) of immediate data which operates with one byte of data at the first operand storage location.

OC only:
- The number of bytes taking part in the operation is determined by the implicit or explicit length of the first operand.

Condition Code:
0 Result is zero
1 Result is not zero
2 --
3 --

Program Interruptions:
Protection (OI and OC only)
Addressing (O, OI and OC only)
Specification (O only)

Programming Note:
The OR may be used to set a bit to one.

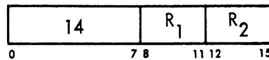
EXAMPLES:

Name	Operation	Operand
	OR	9,2
	O	3,MASK
	OI	SW,X'Ø1'
	OC	ONE,TWO

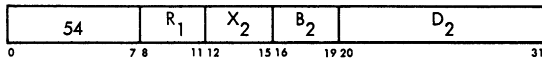
You have just read about the OR instructions. Now you will read about the AND instructions in the following material.

AND

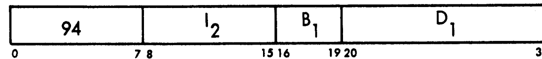
NR R_1, R_2 **[*RR*]**



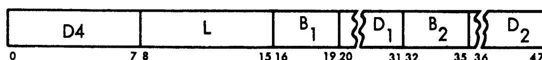
N $R_1, D_2(X_2, B_2)$ **[*RX*]**



NI $D_1(B_1), I_2$ **[*SI*]**



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



All of above:
The first and second operands are examined on a corresponding bit by bit basis.

- If both of the corresponding bits are ones, the result is a one and replaces the bit in the first operand.
- If either or both of the corresponding bits are zeros, the result is a zero and replaces the bit in the first operand.

N only:

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

NI only:

- The second operand is one byte (8 bits) of immediate data which operates with one byte of data, at the first operand location.

NC only:

- The number of bytes taking part in the operation is determined by the implicit or explicit length of the first operand.

Condition Code:
- 0 Result is zero
- 1 Result is not zero
- 2 --
- 3 --

Program Interruptions:
- Protection (NI and NC only)
- Addressing (N, NI and NC only)
- Specification (N only)

Programming Note:
The AND may be used to set a bit to zero.

EXAMPLES:

Name	Operation	Operand
	NR	4,5
	N	3,SET
	NI	SW,X'ØØ'
	NC	FIELD,PATTRN

Here is the instruction we will use to set the switch:

OI SW,X'01'

Copy this instruction onto a separate coding sheet. Don't use the one you are using for assembler instructions.

9. Consult your Reference Data card. The OI instruction is of the (RR/SI/SS) _______ format.

• • •

SI

10. The second operand of this instruction is (your own words) ______________________________ .

• • •

the hexadecimal number 01.

11. Note that the second operand in the instruction is not preceded by an equal sign. This means that when the instruction is assembled, the second operand will be (choose one)
   1. The address of a hexadecimal constant of 01.
   2. The hexadecimal number 01, incorporated into the instruction.

• • •

2.

12. SI instructions have, as their second operands, the actual data to be used in the operation. The actual data to be used in this operation is ______________ .

• • •

The hex number 01.

13. The hex number 01 has what bit structure? (If necessary, consult your Reference Data card)

______________________ .

• • •

0000 0001

14. The second operand of your OI instruction will have what bit structure? ______________________ .

• • •

0000 0001

15. When the immediate data in the instruction (the second operand) is ORed with the contents of our switch, SW, the value of bit number 7 in the switch will be (0/1) ________ .

• • •

1

16. Bit #7 will be a 1 bit because if either the specified bit of the first operand, or the corresponding bit of the second operand, or both, is a one, the result is a one and is stored in the first operand. Which operand do we know has a one bit in position 7? ______ __________________ Why? __________________ .

• • •

The second operand. It is the hex number 01, whose bit structure is 0000 0001.

17. As you will see later where we consider the instruction that tests the switch, bit position 7 actually is the switch. We do not use the other bit positions in SW. For experience in predicting the results of the OI instruction, however, perform this exercise:

SW before OI instruction executed:	0011 1110
Second operand of OI instruction:	0000 0001
SW after OI instruction executed:	

• • •

0011 1111

SKIP OPTION

If you are sure of your ability to predict the results of OR and AND instructions, you may skip to the point at which we resume discussion of the sample program. If you are not sure, you should read the frames on the following four pages.

AND, OR OPERATIONS

The "and" instruction is used to mix two operands on a logical AND basis. The definition of an AND condition is this: If both bits are 1, the resulting bit is 1. Otherwise, it is zero.

The following will illustrate the result of ANDing two bytes together.

	BIT POSITIONS
	0 1 2 3 4 5 6 7
1st Operand	1 0 1 0 1 0 1 0
2nd Operand	1 0 0 1 1 1 0 0
Result	1 0 0 0 1 0 0 0

Notice that only in bit positions 0 and 4 were both bits set to 1. As a result, only bits 0 and 4 of the result are 1. As in most System/360 operations, the result will replace the 1st Operand.

1. Given the following two bytes, show the result after they are ANDed together.

1st Operand	0 1 1 1 0 1 1 0
2nd Operand	1 1 0 0 1 1 0 0
Result	

• • •

0 1 0 0 0 1 0 0 ; Notice again that the operands are ANDed together on a bit-by-bit basis. There is no connection (carry) from one bit position to another.

2. Show the result of the following AND operation.

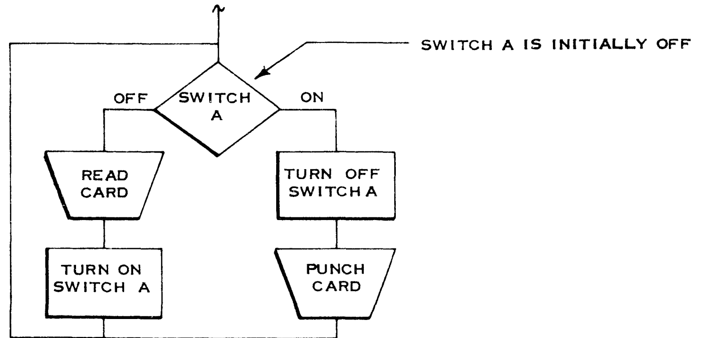
1st Operand	1 1 0 0 0 0 1 1
2nd Operand	1 0 0 0 0 0 0 1
Result	

• • •

1 0 0 0 0 0 0 1

In a typical user's program, the programmer will occasionally use data as a programmable switch. That is, in a flowchart, a branch decision will occasionally be based on whether a switch is on or off.

For example:



Notice that switch A is used to determine whether to read a card or punch a card. Also, notice that the flowchart assumes that there is a method of turning the switch on and off.

One use of the "and" instruction is to turn off program switches.

0 1 0 0 0 0 1 1

3. Assume that bit position 7 of the byte shown above represents a program switch. In order to turn off this program switch, bit position 7 of the second operand must be ______ (1/0).

• • •

0; See below.

1st Operand	0 1 0 0 0 0 1 1
2nd Operand	1 1 1 1 1 1 1 0
Result	0 1 0 0 0 0 1 0

4. Since a byte contains 8 bits it can hold ____ program switches.

• • •

8

5. It is desired to turn off one program switch in a byte without affecting the other switches. Show the 2nd operand necessary to turn off only the switch in bit position 6.

1st Operand	1 0 1 1 0 1 1 0
2nd Operand	__________
Result	1 0 1 1 0 1 0 0

• • •

1 1 1 1 1 1 0 1 ; Notice that only bit position 6 was changed. This 2nd operand would work with any 1st operand and still turn off only position 6.

6. It is desired at the beginning of the program to be sure that all of the program switches are off. Show the necessary 2nd operand.

1st Operand	0 1 1 0 0 0 1 0	8 program switches
2nd Operand	________	

• • •

0 0 0 0 0 0 0 0

Now that you can turn off program switches, how about turning them on?

The "and" instructions can be used to turn off program switches. The "or" instructions can be used to turn on program switches. The definition of an OR condition is this: If either bit is 1, the resulting bit is 1. Otherwise, it is zero.

The following will illustrate the result of ORing two bytes together.

1st Operand	1 0 1 0 1 0 1 0
2nd Operand	1 0 0 1 1 1 0 0
Result	1 0 1 1 1 1 1 0

Notice that only in bit positions 1 and 7 neither bit was set to 1. Consequently, only bits 1 and 7 of the result are set to 0. The remaining bits contain a 1.

7. Given the following operands, show the result if they are ORed together.

1st Operand	0 1 1 1 0 1 1 0
2nd Operand	1 1 0 0 1 1 0 0
Result	

• • •

1 1 1 1 1 1 1 0

8. Show the result of the following OR operation.

1st Operand	1 1 0 0 0 0 1 1
2nd Operand	1 0 0 0 0 0 0 1
Result	

• • •

1 1 0 0 0 0 1 1

9. [0 1 0 0 0 0 1 0]

Assume that bit position 7 of the byte shown above is a program switch. In order to turn on this program switch, bit position 7 of the 2nd operand must be ______ (0/1).

• • •

1; See below.

1st Operand	0 1 0 0 0 0 1 0
2nd Operand	0 0 0 0 0 0 0 1
Result	0 1 0 0 0 0 1 1

10. It is desired to turn on one program switch in a byte without affecting the others. Show the necessary 2nd operand to turn on only the switch in bit position 2.

1st Operand	1 1 0 1 0 1 1 0
2nd Operand	________

• • •

0 0 1 0 0 0 0 0

11. Show the result of ORing the previous answer with the 1st operand.

1st Operand	1 1 0 1 0 1 1 0
2nd Operand	0 0 1 0 0 0 0 0
Result	

• • •

1 1 1 1 0 1 1 0

12. It is desired at the beginning of a program to be sure that all of the program switches are initially on. Show the necessary 2nd operand.

1st Operand	0 1 1 0 0 0 1 0	8 program switches
2nd Operand	________	

• • •

1 1 1 1 1 1 1 1

AND INSTRUCTION - OR INSTRUCTION

1. The mnemonic of the "and" instruction uses the letter ________ .

• • •

N

2. The mnemonic of the "or" instruction uses the letter ________ .

• • •

O

3. Both the "and" and "or" instructions can use four formats. Use the following mnemonics and indicate the instruction format and whether it is an "and" or "or" instruction.

Mnemonic	Format	Instruction
NR	__ __	________
NC	__ __	________
OI	__ __	________
N	__ __	________
OR	__ __	________

• • •

Mnemonic	Format	Instruction
NR	RR	"and"
NC	SS	"and"
OI	SI	"or"
N	RX	"and"
OR	RR	"or"

4. After executing an "and" or "or" instruction, the condition code can be set to one of ____ possible settings.

• • •

two

5. A condition code of 00 would indicate a result of __________ (all zeroes/all ones).

• • •

all zeroes

6. If the result is not zero, the condition code will be set to _______ 0/1.

• • •

1

7. The two possible settings of the condition code after an "and" or "or" instruction are: _____ _______

• • •

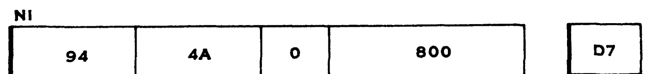
0; 1

8. The "branch on condition" instruction can be used after an "and" or "or" instruction to: (Circle one of the following.)
   a. Check if all program switches are set to 0.
   b. Check if one specific switch is on or off.

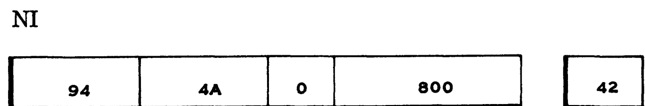
• • •

a; The condition code reflects the status of the entire result. It can either be zero or non-zero.

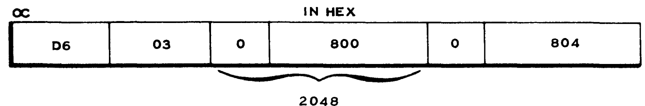
9. Given the following "and" instruction, show the contents of the instruction and the storage byte after the instruction is executed.



• • •



10. Given the following, show the storage contents after the instruction is executed.

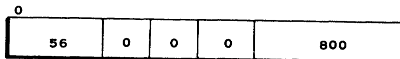


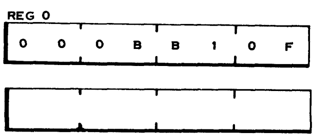
Location	Before	After
2048	DE	____
2049	A0	____
2050	7F	____
2051	8B	____
2052	9E	____
2053	01	____
2054	72	____
2055	F1	____

• • •

Location	After
2048	DE
2049	A1
2050	7F
2051	FB
2052	9E
2053	01
2054	72
2055	F1

11. Given the following "or" instruction, show the contents of register 0 and the condition code after the instruction is executed.



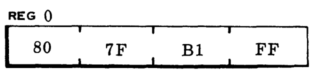


IN HEX

LOCATION	CONTENTS
2048	80
2049	7E
2050	01
2051	F0

Condition Code = _____

• • •



Condition Code = 1

END OF SKIP OPTION

Returning to our problem we see that the next step on the flowchart is to read a card. At this point in coding a program, we usually write the assembler instructions that reserve the input record area and indicate the sizes and symbolic names of the data fields within this area.

1. The problem statement, Figure 17, gives the symbolic names you will use, and indicates the sizes of the data items that will read into the input area. Using this information, write the assembler instructions that will reserve a storage area for the input card, and define the data fields within the area.

• • •

Name (1–8)	Operation (10–14)	Operand (16–)
INPUT	DS	ØCL8Ø
ENAME	DS	CL15
EMPNO	DS	CL6
TAXCL	DS	CL2
YTDGRS	DS	CL7
YTDWH	DS	CL6
YTDFICA	DS	CL5
GROSS	DS	CL6
	DS	CL33

2. Now that we have defined our input area we are ready to read a card. You will recall that we can do this by branching to a pre-coded routine that actually performs the read operation, and then branches back to your routine. The name of the routine is READ. Using register 10 as the linking register, write the instruction that branches to the READ routine. Name your instruction READCD.

• • •

READCD BAL 10,READ

3. According to the flowchart, the next step is to test the switch. If it is OFF, we check the sequence of the card just read; if ON, we turn it OFF and bypass the sequence check. When would we not want to check sequence? ____________________
Why? ____________________

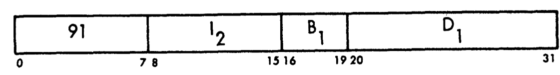
• • •

When the first card is read. There is no preceding card to check it against.

To test the switch we will use an instruction called Test Under Mask. Its mnemonic is TM. Read about this instruction in the following material and locate the group that contains it on your Reference Data card.

Test Under Mask

TM $D_1(B_1), I_2$ [*SI*]



The state of the first operand bits selected by a mask (second operand) is used to set the condition code.

- 1 - 8 bits may be tested.
- The mask is one byte (8 bits) of immediate data (second operand).
- A mask bit of one indicates that the corresponding storage bit is to be tested.
- A mask bit of zero indicates that the corresponding storage bit is to be ignored.

Condition Code:
- 0 Selected bits or mask, all-zero
- 1 Selected bits mixed zero and one
- 2 --
- 3 Selected bits all-one

Program Interruptions:
- Addressing

EXAMPLE:

Name	Operation	Operand
	TM	SW2,X'03'
	BC	1,OFF

4. In the above example, the mask byte is 0000 0011 (as determined by the hex operand X'03'). Suppose that the first operand, SW2, contains 0000 0001. After the TM instruction is executed what condition code will be set, and why?

• • •

1 (selected bits mixed 0 and 1) because positions 6 and 7 of SW2 are tested; 6 contains a 0 and 7 contains a 1.

5. With this condition code, what will be the result of the instruction BC 1,OFF?

• • •

The program will not branch. If necessary, check your Reference Data card. A first operand of 4 is needed in the BC instruction in order to test for a condition code of 1.

6. Now back to the sample program: You will recall from our discussion of the OI instruction that if the second operand of the source language instruction is an X'01', the operand of the assembled instruction will have the bit structure ________________

• • •

0000 0001

7. A byte with bit structure 0000 0001 has the one bit in the (position number)________ position.

• • •

7

8. A TM instruction whose second operand has the bit structure 0000 0001 will test the (position number) ________ bit in the byte being tested.

• • •

7

9. Bit position 7 in SW functions as (your own words) ______________________________ in our program.

• • •

A programmed switch.

The second operand of the TM instruction is called a mask. If a mask bit is one, the corresponding bit in the character specified by the first operand will be tested.

10. If the mask of our TM instruction has the bit structure 0000 0001, which bits in SW will be tested?_______ Why? ____________________

• • •

the low-order bit. Only mask bits of one cause testing. Mask bits of zero do not.

11. What is the setting of the low-order bit in SW?

____

• • •

It is a 1.

12. Write the TM instruction that will test only the low-order bit in SW. _______________

• • •

TM SW,X'01'

Change your answer, if necessary.

13. The setting of the condition code as a result of the TM instruction will be (0/1/2/3)______.

• • •

3

The condition code will be 3 because the selected bits in the position under test are all-one. In this case only a single position was tested, but since it contained a one-bit, the all-one condition is met and the condition code is 3.

14. In the following example, which bit positions will be tested? ___________

	0 ... 7
TEST MASK	0 0 1 1 0 1 0 1
BYTE UNDER TEST	1 1 0 1 1 0 0 1

• • •

2, 3, 5, 7

15. The condition code will be (0/1/2/3) ________.

• • •

1

SKIP OPTION

If you are sure you can predict the results of Test Under Mask instructions, regardless of the bit structures involved, you may skip to the point at which we resume discussion of our sample program. If you are not sure, you should read the frames on the following three pages.

Note that in these frames, the Mask is referred to as I2 (second operand) while the storage address of the character under test is given in hexadecimal notation. Instruction operation codes, and bit structure of masks and test characters, are given in hex numbers. Consult your Reference Data card for the necessary conversion values.

TEST UNDER MASK INSTRUCTION

So far, you can turn on, or change a program switch by use of "logical" instructions. However, you still can't test them. The "branch on condition" instruction is not sufficient. This instruction will only let you find out if all switches are either on or off. To be able to test a specific switch (or some but not all switches) will require another instruction. This instruction is known as "test under mask". The "test under mask" instruction will let you examine specific bits (program switches) and set the condition code accordingly. Then the "branch on condition" instruction can be used effectively.

Reread the description of the "test under mask" instruction.

1. TM is the mnemonic for "_____ ______ ________".

•••

"test under mask"

2. The TM instruction uses the ___ ___ format. The instruction can be used to test program switches. These switches must be in ______________ (main storage/general registers).

•••

SI; main storage

3. The "test under mask" instruction will test one _________ (byte/halfword/word).

•••

byte

4. A byte can contain 8 program switches. The TM instruction can test _________ (only one/all) of them at one time.

•••

all

5. The TM instruction can, if desired, test as few as ________ program switch(es) at a time.

•••

one

6. The bits (program switches) to be tested with the TM instruction are determined by the instruction's _____ field.

•••

I2

7. The I2 field corresponds bit-by-bit with the main storage byte to be tested. If all bits in the main storage byte are to be tested, the I2 field must contain _______ (in hex).

•••

FF

8. To test only bit position 0, the I2 field of the TM instruction must contain ______ (in hex).

•••

80

9. The function of the TM instruction is to set the condition code. A condition code of 0 would indicate that all of the selected bits are zero. Assume that the I2 field of a TM instruction is FF. A condition code of 0 would indicate that the storage byte contains _______ (in hex).

•••

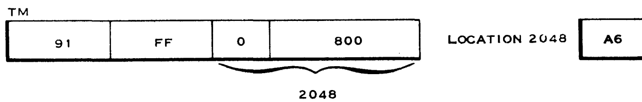
00; An I2 field of FF would test all bits. A condition code of 0 would then indicate that all bits of the byte are zero.

10. If a TM instruction results in a condition code of 3, it would indicate that all of the selected bits are one. If the I2 field of the instruction used contained FF, it would mean that the storage byte contains _________ (in hex).

•••

FF

11. A condition code of 1 is also possible after executing a TM instruction. This condition code (1) would indicate that some but not all of the selected bits contain a one. Given the following, what would the resulting condition code be?



Condition Code = ________

• • •

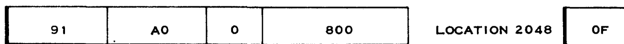
1

12. After executing a TM instruction, it is not possible to have a condition code of ________ .

• • •

2

13. TM

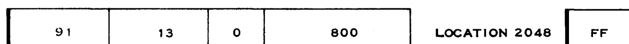


The above "test under mask" instruction would result in a condition code of _____ .

• • •

0

14. TM

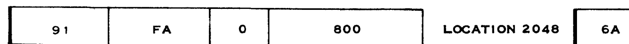


The above TM instruction would result in a condition code of _____ .

• • •

3

15. TM

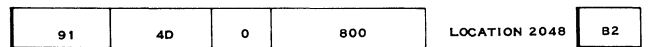


The above "test under mask" instruction would result in a condition code of _______ .

• • •

1

16. TM

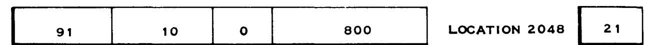


The above TM instruction would result in a condition code of ______ .

• • •

0

17. TM



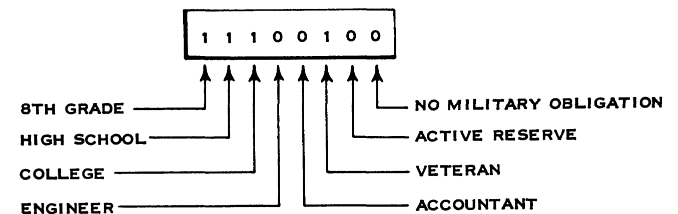
The above TM instruction would result in a condition code of _______ .

• • •

0

At this point, you can turn program switches on and off. You can also test them. The instructions that you have been using can be used for purposes other than program switches.

Logical instructions can also be used to examine records for specific requirements. Consider the case of a program where it is desired to find all employees whose qualifications fit a particular job description. The following byte will show the minimum requirements for the job.

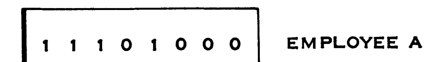


18. According to the preceding requirements, the employee must have a _______ degree. However, he does not need to be an ___________ or an ___________________ .

• • •

college; engineer; accountant

Assume that a card column in each employee's record is punched to show his qualifications. The following byte shows the qualifications of one such employee.



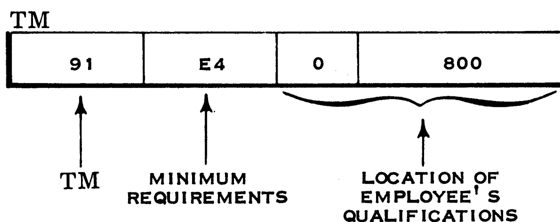
19. Examine the preceding employee's qualifications and refer back to the requirements for the job. Employee A is not qualified because he is not a ____________ .

• • •

veteran

The question now is: How can logical instructions be used to determine whether an employee meets the requirements?

To determine whether an employee meets the minimum requirements, the "test under mask" instruction could be used. This is shown as follows:



20. With the TM instruction, the condition code is set according to the status of the selected bits. To meet the minimum requirements, the selected bits in the employee's qualifications must be all ______ (ones/zeroes).

• • •

ones

21. If the selected bits are all ones, the TM instruction would cause the condition code to be set to ____ (0/3).

• • •

3

END OF SKIP OPTION

Returning to our sample problem we find we have tested the switch by using the TM instruction, and as a result of this test the condition code in the PSW is set to 3.

1. In setting the switch ON, we put a one in bit position 7 of the SW, using the OI instruction. The TM instruction tested that bit position of SW, and put a 3 in the condition code of the PSW. Therefore, a condition code of 3 means the switch was (OFF/ON). ________

• • •

ON

2. You previously learned an instruction that will test the PSW for one or more possible condition codes, and branch if any one of those tested for exists. Write an instruction that will branch to a location called OFF if the condition code is 3. ____________

• • •

BC 1,OFF

3. The conditional branch instruction you just wrote will branch to OFF if our programmed switch is ON. (Condition code is 3). From the flowchart you can deduce that OFF is the name of the first instruction in block (D3/E2) ______ .

• • •

D3

The instructions for D3 and E3 are said to be "out-of-line", that is, the program branches out of its primary sequence of instructions, executes the out of line instructions, and then rejoins the primary sequence at some point further on.

4. The block of instructions that will end with a branch back to the primary sequence is (E2/E3) ______ .

• • •

E3

You will find it convenient to code "out-of-line" instructions on a separate coding sheet. This makes the final source program easier to follow. We will write the instructions for block D3 and E3 on a separate sheet.

We used logical instructions to turn ON and test the switch; we will also use a logical instruction to turn it OFF. The instruction we will use employs AND logic. Its mnemonic is NI. You read about the NI instruction previously, along with other AND logic instructions. If you are in doubt about how the NI instruction works you should review that material.

Here is the instruction that will turn the switch OFF:

```
NI   SW,X'00'
```

Copy this instruction onto a separate coding sheet.

5. From the previous instruction you wrote you can determine the name of this instruction. Write this name in the name field. _________

• • •

OFF

6. Let's consider the result of this NI instruction. It is of the (RR/RX/SI/SS) _________ format.

• • •

SI

7. The second operand of an SI instruction is the actual data to be used in the instruction. The second operand of our NI instruction will be ___________ .

• • •

a hexadecimal zero

8. The bit structure of a hex zero is _________ .

• • •

0000 0000

In an NI operation if only one, or neither, of the bits is a one, the result is a zero and is stored in the first operand.

9. The bit structure in SW, when the switch is ON, is _________________ .

• • •

0000 0001

10. The result in SW after the NI operation, will be ________________ .

• • •

0000 0000

We have set our switch off. Returning to the flowchart we see that the next block in the out of line routine is to store the employee number from the first card, so it can be used for sequence checking.

There are several ways we could set up the employee numbers for the sequence check. We could convert them to binary form and compare storage-to-storage, register-to-register, or register to indexed storage. Or we could pack them and compare storage-to-storage.

11. To give you the widest possible experience, we will convert to binary, and compare register-to-register. Our first step then will be to (pack/unpack) ________ the employee number from the first card.

• • •

pack

To pack a data item we must provide a field in storage that is at least half as long in bytes as the data item. We will call the field NUM.

12. Write an assembler instruction that will reserve a work area called NUM into which we can pack the employee number. The statement that defines the input field indicates how long NUM should be. Remember that the employee number will be converted to binary format after it is packed.

• • •

NUM DS D (Remember that the second operand of a CVB must be a doubleword.)

13. Write the instructions that will pack the employee number and convert it to binary, using work area NUM and register number 2.

• • •

PACK NUM,EMPNO; CVB 2,NUM

14. You have coded the out of line blocks in the flowchart. To return to the primary sequence requires a (conditional/unconditional) _______________ branch.

• • •

unconditional

15. Write the instruction to branch back to an instruction called PACK in the primary sequence.

• • •

BC 15,PACK

The next instructions you will write will be in the primary sequence, so use the appropriate coding sheet.

16. The flowchart block to which the program returned from the out of line instruction is ___________ .

●●●

F3

You will code the instructions for F3 now. But since one block preceding F3 has not been coded, you should leave space for those instructions on the coding sheet. Since you don't know exactly how many instructions will be needed, you will have to estimate the number. There is no sure way of doing this. Leave 7 or 8 lines blank.

17. The first instruction of the next block is the one you branched to from the out of line instructions. Its name is ________ .

●●●

PACK

18. We are going to do all the arithmetic steps in the payroll using data in packed decimal form. There are several fields in the input area containing data that must be packed. From the description of the required computations, in the problem statement, you know that these fields are (symbolic names)

____

●●●

TAXCL, YTDGRS, YTDWH, GROSS, YTDFICA

19. Normally when it is necessary to pack data, the data is packed into a special work area set up for the purpose. However, when storage space is limited the data can be packed right back into the field from which it came. To give you experience in this, we will pack our data back into its own locations. Write the instruction to pack the input fields you previously selected into their own locations. Remember that the first of these instructions has a name that you already have identified. Be sure you have left 7 or 8 coding lines blank.

●●●

```
PACK    PACK TAXCL,TAXCL
        PACK YTDGRS,YTDGRS
        PACK YTDWH,YTDWH
        PACK GROSS,GROSS
        PACK YTDFICA,YTDFICA
```

Note that when data is packed back into its own location it is right-justified; that is, the packed digits occupy the rightmost (low order) bytes of the field. Note also that the rightmost half byte holds the sign.

UNPACKED	F 1	F 2	F 3	F 4	F 5	F 6	F 7
PACKED	0 0	0 0	0 0	1 2	3 4	5 6	7 S

20. Block F3 of the flowchart tells us to ______________ ______________________

●●●

Calculate new YTD Gross.

21. We have the YTD gross and the current gross in packed decimal form in storage. New YTD gross equals YTD gross and current gross. Write the instruction. _________________

●●●

```
AP    YTDGRS,GROSS
```

If you had your operands reversed in the instruction you would develop a new YTD gross in the gross field. Since we packed these fields back into their own locations, gross would be lost for good. As the problem statement indicated, we will use gross in further calculations. Thus the operands should be as shown.

22. The instruction you just wrote resulted in (choose one):
    1. Two YTD gross amounts, one larger than the other by the amount of gross.
    2. An updated YTD gross amount and a gross amount, both in separate fields.
    3. No change in the YTD gross and gross fields.

●●●

2.

23. To show the required payroll calculations in greater detail, we have developed more detailed versions of certain sections of the flowchart. One of these is in Figure 20. It shows the detailed steps required to compute ___________________________________

●●●

Current and YTD Withholding tax.

24. As the problem statement points out, a number of calculations are involved in computing withholding tax. Each will develop a specific figure to be included in the output. Since our input data is in packed decimal form, we can work on it using decimal instructions. This means we must reserve some storage areas to serve as accumulators and _________ areas.

• • •

work

One of the amounts we must calculate is the current withholding tax. The formula for computing it is opposite block G3. We will need a work area in which to develop the current withholding figure.

In setting up work areas to serve as accumulators for decimal arithmetic operations, it is good practice to follow this rule:

> Set up the area as a constant of packed zeroes, sufficiently long to accommodate the results of any decimal arithmetic operation you plan to do in the area.

Observance of this rule will yield two desirable results:

- The area is at zero the first time you use it; hence there is no danger of including data left in the area from a previous job.
- The area has a sign in the four low-order bits of the rightmost byte. This is essential, since all decimal instructions require that both operand fields be signed.

25. Check the sizes of the factors involved in computing current withholding tax, and the kind of arithmetic operations required. Now write an assembler instruction that will reserve a packed work area large enough to handle the operation. Name the area CURWH. ____________________

• • •

```
CURWH   DC   PL6'0'
```

CURWH now is correctly initialized for decimal arithmetic operations. However, as you will see when the coding for this problem is complete, it will be convenient to do some additional initializing on this particular accumulator. The following frames explain what needs to be done to CURWH.

26. From the other formulas on the flowchart you can tell that we will do a(n) (add/divide/multiply/subtract) _______________ in computing withholding tax.

• • •

multiply

27. The product of the multiplication will have (how many) _______ decimal positions.

• • •

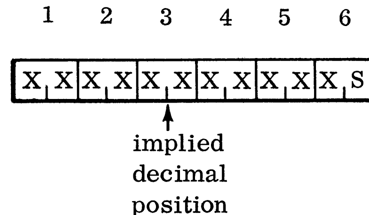
4

28. A product with more than two decimal points usually is _________________ before printing.

• • •

rounded, half-adjusted

You will recall that the results of decimal arithmetic operations are always signed, and that when the product of a multiply operation has been half-adjusted it is necessary to move the sign to the correct location in the product field:



29. Assume the above is a product to be half-adjusted. We want two decimal positions in the final result. In what position do we half-adjust? _______________

• • •

The low-order half of byte 4.

30. To which position will we move the sign? __________

• • •

The low-order half of byte 4.

As you will see, there will be an operation performed in your program in which it will be desirable to have the sign of the data in CURWH located in the rightmost half of byte 5.

31. By using a packed decimal literal with a value of zero as the second operand of a ZAP instruction, we can put a sign into any byte in CURWH. The literal will be kept in storage in a single byte, with the zero being in the four leftmost bits and the sign in the four rightmost bits. The ZAP instruction will use the literal each time it (the ZAP instruction) is executed. Note that an explicit length is necessary in the ZAP instruction to put the literal into the desired byte. Write the instruction. ____________________

• • •

ZAP CURWH(5),=P'0'

We will see later why it is beneficial to do this operation now.

32. The first thing to be done in computing the current and new YTD withholding tax is ______________ (See block B2, Figure 20.)

• • •

Compute exempt amount.

The exempt amount to be computed is that amount of earnings which is not taxable. The formula for this computation is printed beside block B2 on the flowchart. Since this is an arithmetic operation we will need a work area to serve as an accumulator in which the exempt amount can be developed.

33. Check the sizes of the factors that will be involved, and the number of bytes required for each in packed format. Then write an assembler instruction to reserve a packed field large enough to hold the exempt amount and initialize it with zeroes. Name it EXAMT.

• • •

EXAMT DC PL5'0'

34. Refer to the formula beside block B2 on the flowchart and write the instructions that will compute the exempt amount. Note that you will need a packed decimal constant of $28.00.

• • •

ZAP EXAMT,TAXCL
MP EXAMT,=P'2800'

Now we know how much of the employee's earnings he doesn't have to pay tax on. The next step is to find out how much he does have to pay tax on. The formula for this is beside block C2 in the flowchart (Figure 20).

35. We find we need another accumulator. Determine the size of the factors (in terms of packed, signed fields) and write the assembler instruction to reserve the storage for one. You will find the symbolic name in the problem statement list of work areas you will need. This accumulator needs no initializing value.

• • •

TXBLGR DS PL4

36. Write the instructions to calculate the taxable gross amount, using the correct factors.

• • •

ZAP TXBLGR,GROSS
SP TXBLGR,EXAMT

Now we have, in TXBLGR, the amount of earnings on which the employee owes tax. But it's possible he owes no tax. If the exempt amount is equal to or greater than his gross, the tax calculations are bypassed (see flowchart).

37. As a result of every add or subtract instruction, the c__________ c________ is set in the PSW.

• • •

condition code

The condition code will reflect whether the result of a subtract operation is positive, minus, or zero.

38. The flowchart in Figure 20 indicates that the program is to bypass the income tax calculations if the results of our calculations are ____ __ ____.

• • •

zero or minus

39. Use your Reference Data card to find the conditions, and write an instruction that will test for these two conditions and branch to an instruction named A30.

• • •

BC 12,A30

The block of instructions to which the program branches has not been coded. You will code it later. For now, place a mark, such as a check or asterisk, opposite the branch instruction at the edge of the coding sheet. Later on, when you check your program for completeness, these marks will signal any unfinished coding.

40. If the result of the previous subtract operation is plus, the program (does/does not) ______ branch and withholding tax (is/is not) _____ calculated.

● ● ●

does not; is

41. We already have set up a storage accumulator in which to calculate current withholding tax. Its symbolic name is ______________ .

● ● ●

CURWH

42. Write the instructions that will compute current withholding tax in CURWH and half-adjust the result to two decimal positions. The formula in Figure 20 indicates the value of one of the constants you will need. The other will be used in half-adjusting. You should be able to establish its exact value. It may help you to draw a picture of CURWH to establish the bytes you want to work with.

● ● ●

ZAP CURWH,TXBLGR; MP CURWH,=P'14'; AP CURWH,=P'50'

43. We want two decimal positions in the result. Move the sign of the product to the proper position. Remember the instruction that moves only the rightmost half of a byte.

● ● ●

MVN CURWH + 4(1), CURWH + 5

44. You have calculated the current withholding tax. Write an instruction to update YTD withholding tax. Remember you want only part of the contents of CURWH.

● ● ●

AP YTDWH, CURWH(5)

45. You have coded blocks E3, F3 and G3 in the flowchart in Figure 20. Since H2 is the block to which you branched when there was no taxable gross, you know the first instruction of this block will be named ______ . (Look for check marks or asterisks along the margins of your coding sheets.)

● ● ●

A30

46. A30 is the name of the first instruction of a group of instructions whose function is to ____________ __________________ . (Consult Figure 20.)

● ● ●

Compute new YTD FICA

47. The processing steps required to compute new YTD FICA are shown in the detailed flowchart in Figure 21. The first step in this series of calculations is ____________________________ .

● ● ●

Zero Current FICA

48. You have written the instructions to calculate current withholding tax, the exempt amount, and the taxable gross. In each case it was necessary to write an assembler instruction. What do those assembler instructions accomplish (your own words)? ______________________________
____

● ● ●

They set up storage accumulators in which the desired amounts are developed.

Before you set up this accumulator, there is one thing you should consider. The FICA percentage at this time is 4.4, and in packed form requires only 2 bytes.

0 4	4 S

49. In the future, however, the number of decimal positions in the FICA percentage may increase. If the percentage was, say, 4.455, it would require (how many?) __________ bytes.

● ● ●

3

04	45	5S

The FICA percentage has included as many as three decimal positions in the past, and may become that size again. It is good planning, therefore, to allow for this expansion now and set up our storage accumulators and constants accordingly. For this reason, we will assume the FICA percentage to have three decimal positions and one integer.

Note that when we perform the calculation we change the FICA percentage to a decimal fraction. Thus, 4.455% becomes .04455.

50. The accumulator for a packed decimal multiply operation in which the factors are XXXX.XX and .0XXXX must be (how many) ________ bytes long?

• • •

7

51. Sometimes it is possible to use an accumulator that was set up for another purpose, providing, of course, that we no longer need the data it contains. In this case we do need the data in the accumulators we have set up. The next step, then, is to (your own words) ________________________________ .

• • •

Set up another storage accumulator.

52. For what purpose will this accumulator be used? ________________________________ .

• • •

To compute and store current FICA.

53. Consult the formula for computing current FICA and the sizes of the factors. Don't forget that we are allowing for future expansion of one of these factors. Set up an accumulator of the correct size, with an initial value of zero. Name the accumulator CURFICA.

• • •

CURFICA DC PL7'0'

54. Although the accumulator is initialized to zero, it will contain some figure each time it is used in a FICA calculation. However, under certain conditions the FICA calculations are not made, and we will want the accumulator to contain zeroes again. At this point in the program, then, we need an instruction that will replace the contents of this accumulator with zeroes. Write the instruction.

• • •

ZAP CURFICA,=P'0'

Later you will see the reason for putting zeroes in the accumulator at this particular point in the program. Just now you are ready to write the instructions that will perform the FICA calculations.

At this point you may want to review the explanation of FICA calculations in the problem statement. They will help you to understand the reasons for the next group of instructions you will write.

55. Block B2 in Figure 21 is a decision block. Two factors are compared and the result determines (your own words) ________________________________ .

• • •

whether or not we compute FICA.

56. The decision block calls for instructions that compare year-to-date FICA with a figure of $290.40. What does the figure of $290.40 represent? ________________________________ .

• • •

The maximum FICA that any employee can pay.

57. So we see we must compare YTDFICA with the maximum FICA. Since both quantities to be compared are in packed decimal form, we will use a decimal feature instruction for the compare operation. From your Reference Data card select a decimal instruction that we can use ________________________________ .

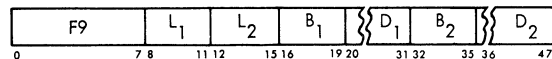
• • •

The instruction is Compare Decimal. Its mnemonic is CP.

Read about this instruction in the following material.

Compare Decimal

CP $D_1(L_1, B_1), D_2(L_2, B_2)$ **[SS]**



The first operand is compared algebraically with the second operand and the result determines the setting of the condition code.

- Both operands are in packed decimal format.
- Comparison proceeds from right to left taking into account the sign as well as all the digits of each field.
- Fields of unequal length can be compared.
- The shorter field in effect will be extended with high-order zeroes.
- Plus zero and minus zero compare equal.

Condition Code:

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

Program Interruptions:
Operation
Addressing
Data
Note: If both fields are not packed, a data exception will occur.

EXAMPLE:

Name	Operation	Operand
	CP	YTDFICA,=P'29040'
	BC	4,LIMITOK

The CP instruction shown above will compare YTDFICA with maximum unpaid FICA.

Copy this instruction onto your coding sheet.

The setting of the condition code in the PSW, after a compare operation, is determined by the relationship of the first operand to the second. Note that the Reference Data card designates first operands of compare instructions as "A", second operands as "B". Resulting condition codes thus reflect the relationship of A to B, on the Reference Data card.

58. Fill in the right-hand column below:

Operand Relationship	Resulting Condition Code
YTDFICA = 290.40	______
YTDFICA > 290.40	______
YTDFICA < 290.40	______

• • •

0; 2; 1

59. Assume FIELDA is three bytes long and FIELDB is five bytes long. Can the fields be compared with the CP instruction? ________

• • •

yes

60. What happens to the shorter field?

• • •

It is extended with high-order zeroes.

61. FIELDA has a plus sign, FIELDB a minus. Both are of the same length. FIELDB has a larger absolute value than FIELDA. The resulting condition code will be (0/1/2) ________ .

• • •

2

SKIP OPTION

If you are sure you can predict the results of compare decimal operations you may skip to the part at which we resume discussion of the sample problem. If you are not sure, you should read the frames on the following two pages.

COMPARE DECIMAL INSTRUCTION

The Compare Decimal instruction makes an algebraic comparison of two packed decimal fields. It does not compare alphameric information. Re-read the description of the Compare Decimal instruction.

1. If the fields addressed by a CP instruction are not in the packed decimal format, a __________ exception will be recognized.

• • •

data

2. The result of the comparison is recorded into the __________ __________ .

• • •

condition code

3. A condition code of 0 would indicate that the operands were ________ .

• • •

equal

4. A condition code of 1 would indicate that the ______ (1st/2nd) operand was low.

• • •

1st

5. A condition code of 2 would indicate that the ______ (1st/2nd) operand was high.

• • •

1st

6. The "compare decimal" instruction _______ (does/does not) change the operands.

• • •

does not

7. Show the resulting condition code for the following "compare decimal" instruction.

   GO1 CP FLDA,FLDB

   FLDA | 79 | 18 | 2C | FLDB | 79 | 18 | 2C |

   Condition Code _______

   • • •

   0; Both operands were equal.

8. Show the resulting condition code for the following CP instruction.

   GO2 CP SETA,SETB

   SETA | 98 | 76 | 5C | SETB | 98 | 76 | 4D |

   Condition Code _______

   • • •

   2; Since the 1st operand is positive, it is high.

9. Show the resulting condition code for the following "compare decimal" instruction.

   GO3 CP FLD1,FLD2

   FLD1 | 00 | 79 | 84 | 7C | FLD2 | 80 | 06 | 1C |

   Condition Code ______

   • • •

   1; Even though the 1st operand is longer, its algebraic value is less than that of the 2nd operand.

10. Show the resulting condition code for the following "compare decimal" instruction.

    GO4 CP SET1,SET2

    SET1 | 98 | 22 | 57 | 18 | 9D | SET2 | 99 | 99 | 99 | 9D |

    Condition Code _____

    • • •

    1; The numeric value of the 1st operand is greater; however, both operands are negative. Algebraically, a small negative number is greater than a large negative number as shown below.

    Low ------------------------ High

    Minus Plus

    5 4 3 2 1 0 1 2 3 4 5

11. Show the resulting condition code for the following "compare decimal" instruction.

    GO5 CP DATA1,DATA2

    Data 1 | 01 | 23 | 4D | Data 2 | 98 | 76 | 5D |

    Condition Code _____

    • • •

    2

END OF SKIP OPTION

1. Returning to the sample problem we see that we have compared YTDFICA with maximum FICA. As a result of this compare operation the condition code in the PSW has been set. The setting of the condition code will determine the path taken by the program; therefore we must test the condition code. Write the instruction that will test the condition code and branch to an instruction called B if YTDFICA is greater than or equal to $290.40.

•••

```
BC    10,B
```

Since this instruction branches the program to a routine not yet coded, place a mark beside it at the edge of the coding sheet.

2. Block C2 in Figure 21 tells us to compute current FICA. Consult the formula, determine the size of the factors involved, and, using the accumulator you set up for this purpose, write the instructions to compute current FICA and half-adjust it to two decimal places. Remember that we are allowing for future expansion of the number of decimal places in the FICA percentage -- this will influence the size of your constants.

•••

```
ZAP   CURFICA, GROSS
MP    CURFICA, =P'4400'
AP    CURFICA, =P'50000'
```

3. The product of this multiplication has 7 decimal places. We are interested in only two, and half-adjusted accordingly. What must we do next to prepare the desired result for further use?

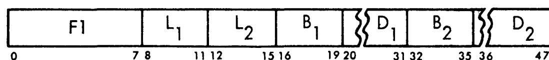
•••

Place the sign in the correct position.

Previously you used the Move Numerics instruction to reposition the sign after a multiply operation. But there are other ways to achieve the same effect, one of which you will learn now. The instruction we will use is the Move with Offset. Read about the instruction in the following material.

Move with Offset

MVO $D_1(L_1, B_1), D_2(L_2, B_2)$ **[SS]**



The second operand is placed in the first operand location, to the left of and adjacent to the low-order four bits of the first operand.

- The fields are processed right to left.
- If the second operand field is shorter than the first operand, it is extended with high-order zeroes.
- If the first operand field is shorter than the second operand, the remaining information is ignored.

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

NOTE: Any type of data can be moved with this instruction. The instruction does not check the type.

Programming Note:

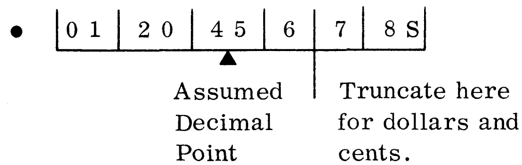
A very common usage of the move numeric (MVN) and move with offset (MVO) instructions, is the correct positioning of the sign in decimal rounding.

Rule:

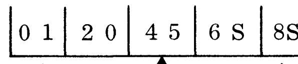
- If the packed decimal number is to be truncated after an ODD number of digits (thus ending in the middle of a byte), move the sign to the number with a move numeric (MVN) instruction.
- If the packed decimal number is to be truncated after an EVEN number of digits (thus ending on a complete byte), move the number to the sign with a move with offset (MVO) instruction.

Examples:

Assume that we are working with dollars and cents. If the assumed decimal point is as shown, the operation would be as follows:

- 

  01 20 4▲56 contains an ODD number of digits and the sign would be moved to the number with a MVN with the following result:

  This is the desired result in valid packed decimal format and would represent an amount of $1204.56.

- | 01 | 23 | 45 | 67 | 8S |

Assumed Decimal Point (between 23 and 45) | Truncate here for dollars and cents (between 45 and 67).

01 23▲45 contains an EVEN number of digits and the number would be moved to the sign with a MVO with the following result:

00	00	12	34	5S

This is the desired result in valid packed decimal form and would represent an amount of $123.45.

EXAMPLE:

Name	Operation	Operand
	MVO	CURFICA,CURFICA(4)

The instruction shown above will sign the half-adjusted product in CURFICA.

Copy this instruction onto your coding sheet.

4. You can see that instead of moving the sign to the data we have moved the data to the sign, getting rid of any unwanted decimal positions in the process. The data we wanted was effectively shifted (left/right) _________ .

• • •

right

5. Show the results of this instruction

MVO COUNT,COUNT(4)

COUNT (before) | 00 | 08 | 37 | 19 | 83 | 7C |

COUNT (after) | | | | | | |

• • •

00	00	00	83	71	9C

SKIP OPTION

If you are sure you can predict the results of Move with Offset instructions, you may skip to the point at which we resume discussion of our sample program. If you are not sure, you should read the frames on the following two pages.

MOVE WITH OFFSET INSTRUCTION

Re-read the first material on the MVO instruction.

Here is an example of moving with offset:

The following packed decimal field represents a product with seven decimal positions:

FIELDA | 3 4 | 7 5 | 6 8 | 1 7 | 9 3 | 4 S |

(implied decimal point between 7 5 and 6 8)

Assume we want only two decimal positions in the final result. To eliminate the five unwanted decimal positions and to position the remaining data next to the sign, the following MVO instruction could be used:

MVO FIELDA,FIELDA(3)

Note that the first operand consists of the entire 6 bytes of FIELDA, whereas the second consists of only the first three bytes of FIELDA plus as many high-order zeroes as may be required. The zeroes are supplied automatically. The following shows the results of the instruction:

FIELDA(3) | 0 0 0 0 0 0 3 4 7 5 6 8 |

FIELDA | 3 4 7 5 6 8 1 7 9 3 4 S |

FIELDA (after MVO) | 0 0 0 0 0 3 4 7 5 6 8 S |

Note the offset. The second operand is placed to the left of and adjacent to the low-order four bits of the first operand.

This effectively eliminates the unwanted decimal positions and places the wanted data next to the sign.

The MVO instruction is normally used to align or shift a packed decimal number to a sign.

DOLLARS	24	68	05	79	3S

1. Assume that the amount we wish to preserve in the field called DOLLARS is $2468.05. The symbolic address for this amount is _________ .

•••

DOLLARS(3)

2. Since we wish to align the amount to the sign, the symbolic address of this field is ______________ .

•••

DOLLARS; the implied length of DOLLARS is 5 bytes so that no explicit length need be specified.

3. The complete instruction is ________________ .

•••

MVO DOLLARS,DOLLARS(3)

4. The field DOLLARS(3) has a length of ___________ bytes and after the execution of the MVO instruction will be _______ half a byte to the left and placed adjacent to the sign.

•••

three; offset

5. The second operand is ________ (shorter/longer) than the first operand and the resulting field (first operand) will contain ________ in the high positions after the operation.

•••

shorter; zeroes

6. Show the contents of the field DOLLARS after the MVO operation.

DOLLARS (before)	24	68	05	79	3S
DOLLARS (after)					

•••

00	02	46	80	5S

7. Given the following MVO instruction, show the resulting contents of the 1st operand.

	MVO	HASH(3),HASH(2)
HASH	DS	PL4

HASH (before)	00	54	10	7C
HASH (after)				

•••

HASH (after)	00	05	40	7C

8. Is the data moved by the MVO instruction checked to see if it is valid packed decimal data? ________

•••

No. You have seen it being used to right shift packed decimal data an odd number of places. However, any type of data can be moved by this instruction.

END OF SKIP OPTION

Returning to the payroll problem, we see that the next block to be coded is E2 (Figure 21).

See if you can code this block without detailed prompting.

One hint: if you need a symbolic name for a storage accumulator check the list in the problem statement.

You should have written an assembler instruction to reserve a storage accumulator. Either of these would do:

```
UNPDFICA    DC  PL3'0'

UNPDFICA    DS  CL3
```

The instructions to calculate unpaid FICA are:

```
ZAP     UNPDFICA,=P'29040'
SP      UNPDFICA,YTDFICA
```

Note that although we are writing decimal instructions, it is possible to set up the accumulator with a DS statement. The accumulator is not initialized to a particular value, nor is it given a sign. The ZAP instruction, however, checks only its second operand data for a sign; thus it is not necessary for the accumulator to be signed.

1. The unpaid FICA you have just calculated is (select one):
   1. The total FICA the employee must pay in one year.
   2. The FICA he must pay this period.
   3. The remaining FICA he must pay this year, if any.

•••

3.

2. Having calculated the unpaid FICA, if any, we now must determine ____________________
______________________________.
(See Figure 21)

•••

If current FICA is greater than the unpaid FICA.

3. If current FICA <u>is</u> greater than the unpaid FICA, the employee owes (how much) ____________
____________________________ FICA.

•••

The difference between YTD FICA and $290.40. This is the unpaid FICA.

4. If current FICA is not greater than the unpaid FICA, the employee owes (how much) __________
____________________________ FICA.

•••

The full amount of FICA previously calculated on his current earning. This is current FICA.

So we see that we must compare current FICA to unpaid FICA. The result of this comparison will determine exactly how much FICA the employee will pay this time.

5. Write the instruction that will compare current FICA with unpaid FICA. ____________________

•••

```
CP      CURFICA,UNPDFICA
```

6. As a result of this compare operation, the condition code in the PSW has been set. We do not at this moment know what the setting is. Since the path the program takes depends on the condition code, the next instruction must (your own words) ________
____
______________________________.

•••

Test the condition code and, depending on its setting, go to the appropriate routine.

After the condition code in the PSW has been set, we can test it in one of two ways:

- We can test for one or more of the possible settings, or
- We can test for all possible alternate settings.

For example:

In the last compare instruction the setting would be 2 if the first operand was high. It would be 1 or 0 if the first operand was not high.

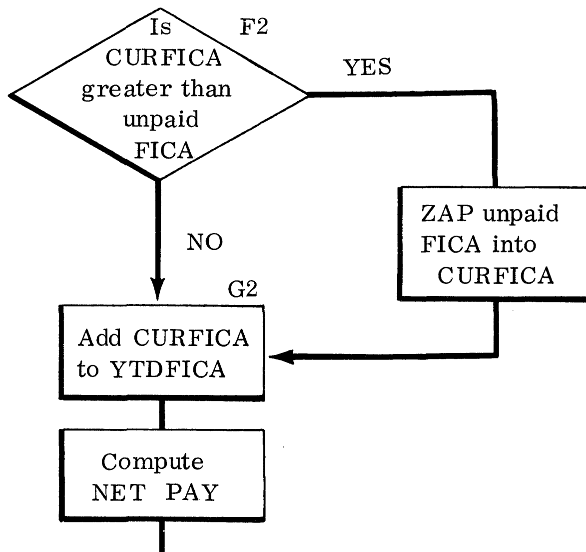
So we could test for 2, or we could test for the alternatives to 2, which are 1 and 0.

7. Write an instruction that will test for the alternatives to the condition code setting of 2, as a result of the previous compare instruction. Go to an instruction named A50 if the alternative conditions exist. ____________________

•••

```
BC      12, A50
```

In the next few frames you will see an example of how imaginative programming can save steps. The following flowchart is a modification of blocks G2 and G3, in Figure 21.



8. The above flowchart has (how many?) ______ blocks in which a quantity is added to YTDFICA.

• • •

1 (G2)

9. Look at Figure 21. There are (how many?) ________ blocks in which a quantity is added to YTDFICA.

• • •

2 (G2 and G3)

At first glance it may appear from the above flowchart that we will always add current FICA to YTDFICA.

But consider this: CURFICA is actually just the symbolic name of an area in storage. True, its name is an abbreviation of current FICA, but there is no reason why we can't put other quantities into it.

10. If we do put something other than current FICA into the area called CURFICA, what name must we use to address the data in that area? ____________.

• • •

CURFICA

We must always address data by the symbolic name of the area in which it is stored.

11. Suppose we put the employee's net pay into CURFICA. To work with the net pay in that area we would use the name ____________ .

• • •

CURFICA

12. If YTD gross were stored in EMPNAME, we would address it as ________ in future instructions.

• • •

EMPNAME

So, although the names of storage areas often indicate their contents, we are not restricted as to what data can be placed in those areas.

Look at the previous flowchart. The key to making a single add instruction do the job of updating YTDFICA is to place the proper amount in CURFICA before the add instruction is executed.

13. If the program takes the NO path from F2 what data will be in CURFICA when block G2 is executed?

• • •

current FICA

14. If the program takes the YES path from F2, what data will be in CURFICA when block G2 is executed?

• • •

unpaid FICA

When we branch (take the YES path) we simply ZAP unpaid FICA into the area called CURFICA, then add CURFICA to YTDFICA.

15. You can see that in either case, a single add instruction will add the proper amount into year to date FICA. See if you can write the instructions to do this. One will be named A50.

• • •

```
        ZAP   CURFICA,UNPDFICA
A50     AP    YTDFICA,CURFICA
```

If you had difficulty in understanding this logic, you should re-read the past few frames. This is illustrative of the kinds of programming that yields the greatest efficiency.

16. The formula for computing net pay is in Figure 19. Set up an accumulator and write the instructions.

• • •

```
NETPAY   DS    CL4
         ZAP   NETPAY,GROSS
         SP    NETPAY,CURFICA
         SP    NETPAY,CURWH(5)
```

17. Figure 21 indicates that a previous instruction branches to the first instruction in the routine that computes net pay. Locate the previous instruction on your coding sheet and assign the correct name to the first instruction in the net pay routine.

• • •

```
B     ZAP     NETPAY,GROSS
```

Now we're ready to print a line on the payroll report. However, before we do, there are a couple of loose ends to be tied up.

18. You remember that we wrote an instruction that zeros the field called CURWH and puts a sign in the right most half of byte 5. This instruction precedes the current withholding tax calculations. Look at your coding sheet and find the instruction. It is ________________________.

• • •

```
ZAP     CURWH(5),=P'0'
```

We said we would see later why it is beneficial to do the operation at this particular point in the program. The reason is this: not all employees will pay income tax. To determine whether or not the employee owes any tax, the program subtracts his exempt amount from his taxable gross; if the result is zero or minus, no tax is due.

19. The instruction that tests for the "no tax due" condition is ______________ .

• • •

```
BC     12,A30
```

If no tax is due, the program bypasses the instructions that calculate the amount of the tax. This means that CURWH is not used and its contents, whatever they are, will be undisturbed.

20. The contents of CURWH, if no tax is due, will be ________________ .

• • •

all zeros, with a sign in the rightmost half of byte 5.

21. Looking at Fig. 26, we see that when we compute net pay we subtract the five high-order bytes of CURWH from NETPAY. Is this calculation ever bypassed?

• • •

No, it is executed for all employees, regardless of whether or not any income tax or FICA is due.

22. Perhaps you can begin to see why we prepare CURWH beforehand. If we didn't, it would have in it the current withholding tax amount for the preceding employee. Now, if the employee we currently are processing has no withholding tax due, and if CURWH contains the tax amount for the preceding employee, what will happen when we compute net pay?

• • •

The current withholding tax amount for the preceding employee will be subtracted from the gross pay of the current employee, <u>who has no tax due</u>.

Foreseeing the possibility of this, we simply zero out CURWH and arrange for a sign to be placed in the proper position; then, if we don't calculate withholding tax for the employee, we still can go through the Net Pay calculations just as if withholding were due. Otherwise we would have to modify the Net Pay routine, to prevent developing incorrect data.

23. Why do we need a sign in the rightmost half of byte five?

• • •

Because we use only the first five bytes of CURWH in computing Net Pay. The computation involves a subtract packed (SP) operation, for which both factors must be in packed decimal format. This means each must be signed.

24. At another place in the program we wrote an instruction to zero out the field called CURFICA before starting FICA calculations. Look at the coding in Fig. 26 and try to determine why we did this.

• • •

The reason is exactly the same as for CURWH. If the employee owes no FICA, nothing is stored in CURFICA. But, because CURFICA was zeroed out in advance (and signed in the rightmost half-byte) the Net Pay calculations can proceed normally.

Look at the flowchart in Figure 19.

Having calculated net pay the next step is to print a line. This means that we must set aside an output area in which all the data that will print on the line can be positioned before the line is printed.

25. Refer to the problem statement for the symbolic name and size of the output area. The name is _______________ and the size in bytes is ________ .

• • •

OUTPUT; 132

26. Write an assembler instruction that will give the name OUTPUT to a storage area of 132 consecutive bytes.

• • •

```
OUTPUT     DS 0CL132
```

Remember that the statement you have just written does not actually reserve the storage area. Further assembler instructions are necessary to reserve and name the individual fields that make up the output area.

The names and sizes of these individual data fields in the output area will be indicated by assembler instructions that follow the one you just wrote.

There are several considerations in setting up data fields in an output area. Some important ones are:

- Where in the output area is the field to go?
- How long must the field be in bytes?
- How many bytes should there be, if any, between this field and the adjacent ones?
- Will the data in the field be edited?
- What punctuation is required, if any?
- If a character constant is shorter or longer than the length specified for it, the right end of the constant will be truncated (dropped) or extended with blanks.
- If a hexadecimal constant is shorter or longer than the length specified for it, the left end of the constant will be truncated or extended with zeros.

The next frames will show you how to deal with these considerations in setting up data fields in the output area.

27. The output from this program will be in (the recording medium) __________ form.

• • •

printed

Look at Figure 22.

28. This is the printer spacing chart for the output from our problem. A printer spacing chart is a planning aid that shows the sizes of the data fields, the punctuation required for specific data items, the location of the fields on the report, the spacing between adjacent fields, and other valuable information. You can deduce that a printer spacing chart should be prepared (before/after) ___________ the output area is set up in the source program.

• • •

before

The printer spacing chart should be carefully laid out and checked to insure that it accurately reflects the distribution and appearance of the output data items on the report form.

We will not discuss the techniques of laying out a printer spacing chart. They are fairly self-evident. However, as you will see, the spacing chart provides valuable information that assists you in determining the lengths of the fields in the output area.

29. Figure 22 shows the p_____ s_____ c______ for our problem.

• • •

printer spacing chart

Look at Figure 22.

30. How many data items on the report do not require editing? ____________ How many do require editing? __________

• • •

2; 5

As you will see, the considerations for setting up data fields in the output area, for edited data, are different in several ways from those for unedited data. We will consider first how to set up fields for unedited data.

31. The leftmost field to be printed on the report is called __________ . It requires (how many) printing positions. _______

• • •

employee number; 6

32. Employee number requires six printing positions; therefore it consists of six digits. Since this data was never packed, it occupies (how many) _______ bytes in storage.

• • •

6

33. To place employee number in the output area we will simply move it from its present location in storage to the appropriate field in the output area. You can deduce that the output area field must be (how many) _______ bytes long.

• • •

6

34. There are 132 printing positions on the printer. The first digit of employee number prints in the ______ printing position. (Consult the printer spacing chart.)

• • •

first

35. Since the first digit in employee number prints in the first printing position, and since employee number is six digits long, the data field for employee number will occupy which positions in the output area? ____ ______ _____

• • •

the first six

36. Write an assembler instruction that will reserve the first six digits of the output area. Consult the problem statement for the symbolic name to be assigned to this field. Remember that this instruction must follow the one that named the entire output area.

• • •

LNO DS CL6

37. The spacing chart in Figure 22 specifies (how many) ______ spaces (printing positions) between the first and second fields on the report.

• • •

2

38. The field that prints immediately to the right of employee number is (edited/unedited) ____________.

• • •

unedited

To obtain the correct spacing between adjacent unedited fields on a report, the data that will print in those fields must be positioned in the output area so that the desired number of spaces exists between the data items. This means that we must define not only the data fields in the output area, but the space areas between them in the output area.

39. We know that we want two spaces between the first and second data fields on the report. Write an assembler instruction that will reserve an unnamed field of two bytes immediately following the field called LNO. ______________

• • •

DS CL2

40. What effect will this field have on the printed report? (your own words) ____________________ ____.

• • •

It will cause the first and second fields to be separated by two printing positions (spaces).

You have reserved and named an output data field for employee number, and have provided for the spacing between it and the next field.

41. The next field will contain (edited/unedited) _______ data.

• • •

unedited

42. Write the assembler instruction that will reserve the next (second from left) data field in the output area. The name for this field is in the problem statement. ______________________

• • •

LNAME DS CL15

Look at Figure 22.

43. The next data field on the report (third from left) will contain (edited/unedited) _________ data.

• • •

edited

There are several things that influence the lengths of output data fields for edited data. Each must be considered before the assembler instructions defining such fields can be written.

44. With which printing position does the fill character in the edit pattern correspond? (Your own words) ______________________________________ .

• • •

The space immediately preceding the YTD Gross field on the report.

45. The spacing chart indicates that two spaces are wanted between the employee name and YTD Gross fields. From the above frame you can deduce that one of these two spaces will be provided by ________ ____________________________________ .

• • •

the fill character in the edit pattern

46. You have learned that when the fill character is a blank the edit pattern for a data field must be at least one byte longer than the number of significant digits and punctuation marks in the edit result. This extra byte is occupied by ________________ .

• • •

the fill character

47. You have also learned that if the fill character is a blank it will provide a ________ immediately preceding the printed data on the report.

• • •

space

You may wonder what would happen if the fill character were other than a blank. Would it still provide a space, that is, would the printing position immediately preceding the data on the report print nothing? The answer depends on whether or not the fill character is printable; that is, does the printer have a graphic character that will print in response to the presence of the fill character in the pattern.

48. Your Reference Data card shows many bit configurations for which no graphics (printable characters) exist. These "characters" if used as fill characters, (would/would not) ______ produce a blank on the report.

• • •

would

49. The asterisk (*) often is used as a fill character. It (will/will not) _____________ produce a blank.

• • •

will not

50. The asterisk will not produce a blank when used as a fill character, because an asterisk is printable. For a fill character to provide a space beside the printed data it (the fill character) must (your own words) ______________________________________ .

• • •

NOT be printable

51. Look at the second and third fields to be printed on the report. (Spacing chart, Figure 22.) There are (how many) _______ spaces between these fields on the report.

• • •

2

We know that the third field on the report (YTD Gross) will contain edited data. Therefore, to determine the exact number of positions that we must reserve for this data in the output area, we must determine exactly how many characters will print on the report.

52. By counting the punctuation marks and the X's you can determine that the YTD Gross field will print a maximum of _________ characters.

• • •

9

53. The first step in editing data is to move a pattern into the output data field. The pattern for YTD Gross must provide for (how many) ____________ punctuation marks.

• • •

2 (comma and period)

54. It must provide for (how many) _____ significant digits. (Count the X's on the spacing chart.)

• • •

7

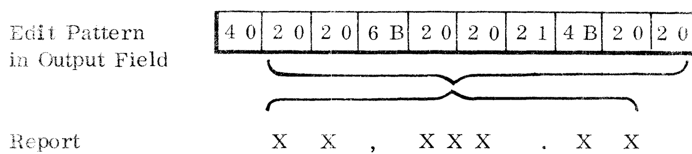
55. It must provide for one extra character, the fill character, which in this case will be a blank. The total number of characters in the edit pattern will be ________________ .

• • •

10 (7+2+1)

The edit pattern will be 10 bytes long. But a maximum of 9 characters (digits and punctuation) is wanted on the printed report.

The nine rightmost bytes in the pattern correspond with the nine positions in which YTD Gross will print:



You have learned that if the fill character is a blank we must define a field in the output area one byte longer than the number of positions in the printed data. Also you have learned that the blank fill character will provide a space immediately preceding the printed data on the report.

Note: It is possible to use a digit select character or a significance start character as a fill character. This gives the effect of having no fill character at all, because if the first digit in the source field is a significant digit, it is placed in the fill character location and printed.

There is another factor that will influence the size of the data field in the output area from which the edited data will print. It is the size of the source field.

Assume the two following fields are laid out on a spacing chart:

X X , X X X . X X

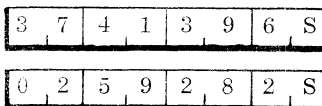
X , X X X . X X

Sample data for these two fields might be:

3 7 4 1 3 9 6

2 5 9 2 8 2

Since data being edited must be in packed format these amounts would look like this in storage:

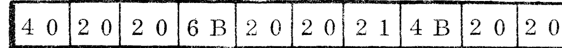


Zero supplied automatically during pack operation

Note that the smaller amount has a high order zero that was automatically supplied during the PACK operation, to fill out the extra half byte in the packed field.

The larger quantity does not need a fill-in zero since the quantity itself fills all bytes in the field.

Now let's build the edit pattern for these two fields. Using a blank as the fill character the pattern for the first quantity looks like this:



56. Look back at the field layouts just shown you. This pattern contains (how many)_______ more characters than the maximum number that will print on the report.

• • •

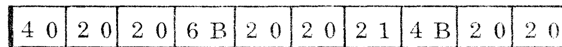
one

57. The extra character in the pattern is ___________ .

• • •

the fill character

Here is the edit pattern for the second field:



58. This pattern contains (how many)______ more characters than the maximum that will print on the report.

• • •

two

59. The extra pattern characters are the ______ ________________ and the ______________________ .

• • •

fill character; first digit select character

60. An additional pattern character is required to edit the high order zero in the source field of the second data item. This zero (was/was not) _______ in the source field before the field was packed.

• • •

was not

61. The first source field has an (odd/even) ______ number of digits. It (does/does not) __________ require a fill-in zero when it is packed.

• • •

odd; does not

62. The second source field has an (odd/even) ______ number of digits. It (does/does not) ____ require a fill-in zero when it is packed.

• • •

even; does

63. If a source field has an even number of digits its edit pattern will have (how many) _______ characters more than the maximum that will print on the report.

• • •

two

64. If a source field has an odd number of digits its edit pattern will have (how many) _____ characters more than the maximum that will print on the report.

• • •

one

When the source field has an odd number of digits there is no need for a fill-in zero from the pack operation because the data occupies all positions in the field. Hence the edit pattern contains only one extra character (the fill character) and provides one space immediately preceding the printed data.

65. When the source field has an even number of digits a fill-in zero is supplied during the PACK operation because the data does not occupy the leftmost half byte in the field. Hence the edit pattern contains (how many) ____ extra character(s) that provide(s) (how many) ______ extra space(s) immediately preceding the printed data.

• • •

two; two

So you can see that before we can define the fields in the output area we must know the sizes of the edit patterns.

To determine the sizes of the edit patterns we must know how many bytes there are in the source fields, how much punctuation is required and the specific character to be used as a fill character.

To determine the number of spaces for which we must write assembler instructions we must know whether the fill character in the edit pattern will provide a space, and whether there is an additional digit select character in the pattern to compensate for a fill-in zero in the source field, which will provide an additional space.

Let's determine the size of the data field in the output area from which the next field on the report will print. This is the YTD Gross field.

66. Look at Figure 22. The maximum number of digits and punctuation marks that will print in the YTD Gross field is _______ .

• • •

9

67. Consult the problem statement (Figure 17). The YTD Gross amount in the input record contains an (even/odd) __________ number of digits.

• • •

odd

68. When the YTD Gross amount was packed a fill-in zero (was/was not) ___________ inserted.

• • •

was not

69. The edit pattern will consist of (how many) _______ characters, including the fill character.

• • •

10

70. There should be two spaces between the employee name field and the YTD Gross field on the report. How will the one preceding YTD Gross be supplied? ___________________________________

• • •

By the fill character in the YTD Gross edit pattern.

71. How will we provide the other space? ___________ ____

• • •

By writing an assembler instruction that will reserve one byte immediately following the employee name field in the output area.

72. Write an unnamed assembler instruction that will reserve one byte immediately following the employee name data field in the output area. ______

• • •

DS CL1

73. Now write an assembler instruction that will reserve a data field in the output area that will hold the edit pattern for the YTD Gross amount. The name of this field is given in the problem statement.

● ● ●

```
LGROSS   DS   CL10
```

There are four more amounts for which data fields must be set up in the output area. Each of these four will be edited, therefore the data fields in the output area must be large enough to hold the edit patterns for those amounts. Also, each printed field is spaced a certain number of positions from the adjacent fields.

74. Write the assembler instructions that will reserve the data fields in the output area for the remaining amounts that are to print on the report. Also, write the assembler instructions that are required to provide the correct amount of spaces between adjacent fields.

Do not develop the edit patterns at this time, just determine the number of characters each must have. Check each source field to see if a fill-in zero was supplied during the PACK operation - this will call for an extra pattern character (a digit select) that will in turn influence the spacing between fields.

● ● ●

```
            DS      CL1
LFEDWH      DS      CL10
            DS      CL1
LFICA       DS      CL7
            DS      CL1
LCURGR      DS      CL10
            DS      CL1
LNPAY       DS      CL7
```

75. Now add up the total of bytes you have reserved in the output area and write an additional assembler instruction that will reserve the remainder of the 132 bytes allocated to the area.

● ● ●

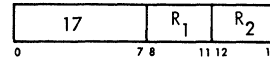
```
DS        CL60
```

Now that we have defined the output area and the individual fields that make it up, we are ready to put it to use. The first thing that should be done prior to using an output area is to clear it of previous data.

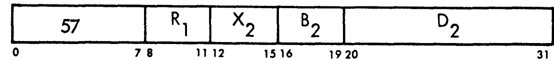
The instruction that you will use this time to clear the output area before putting in new data is called Exclusive OR. Read about this instruction in the following material.

Exclusive OR

XR R_1, R_2 **[RR]**

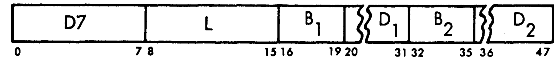


X $R_1, D_2(X_2, B_2)$ **[RX]**



XI $D_1(B_1), I_2$ **[SI]**

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



The first and second operands are examined on a corresponding bit by bit basis.

All of above:

- If either, but not both, of the corresponding bits is a one, the result is a one and replaces the bit in the first operand.
- If both bits are ones or if both bits are zeroes, the result is a zero and replaces the bit in the first operand.

X only:

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

XI only:

- The second operand is one byte (8 bits) of immediate data which operates with one byte of data at the first operand storage location.

XC only:

- The number of bytes taking part in the operation is determined by the implicit or explicit length of the first operand.

Condition Code:

0	Result is zero
1	Result is not zero
2	--
3	--

Program Interruptions:
Protection (XI and XC only)
Addressing (X, XI and XC only)
Specification (X only)

EXAMPLES:

Name	Operation	Operand
	XR	7,1
	X	4,SEARCH
	XI	SWITCH,X'72'
	XC	OUTPUT,OUTPUT

Here is the instruction you will use to clear the print area of old data:

```
XC    OUTPUT,OUTPUT
```

Copy this instruction on your coding sheet.

76. Assuming that a given bit in the first operand is a one and the corresponding bit of the second operand is also a one, the resulting bit in the first operand will be a (0/1) _______ .

• • •

0

77. Give the results of the following combinations:

1st Operand bit	2nd Operand bit	Resulting 1st Operand bit
1	0	_____
0	0	_____
0	1	_____

• • •

1; 0; 1

If we execute the XC instruction with both operands the same, corresponding bit positions in the two operands are always identical. Since, if neither bit is a one, or if both bits are ones, the resulting bit is zero, the operand is effectively zeroed out.

SKIP OPTION

If you are sure you can predict the results of XC instructions you may skip to the point at which we resume discussion of the sample program. If you are not sure, you should read the frames on this and the following page.

EXCLUSIVE OR INSTRUCTION

So far you have had a good look at the "and" and "or" instructions. You have seen how they can be used to turn on and turn off program switches. Another instruction that can be used to alternately turn on and turn off a program switch is the "exclusive or" instruction. The definition of an Exclusive OR condition is this:

If one and only one of the bits is 1, the result is 1. Otherwise, it is zero.

The following will illustrate the result of Exclusive ORing two bytes.

1st Operand	1 0 1 0 1 0 1 0
2nd Operand	1 0 0 1 1 1 0 0
Result	0 0 1 1 0 1 1 0

Notice that in bit positions 2, 3, 5 and 6 one and only one of the bits was a 1. So only these positions of the result have a 1. In bit position 0, both bits were 1 and the result was 0. In bit position 1, both bits were 0 and the result was 0.

1. Given the following operands, show the result of Exclusive ORing them.

1st Operand	0 1 1 1 0 1 1 0
2nd Operand	1 1 0 0 1 1 0 0
Result	

• • •

1 0 1 1 1 0 1 0

2. Show the result of the following Exclusive OR operation.

1st Operand	1 1 0 0 0 0 1 1
2nd Operand	1 0 0 0 0 0 0 1
Result	

• • •

0 1 0 0 0 0 1 0

3. It is desired to change only one program switch in a byte without affecting the others. Show the 2nd operand necessary to change only the switch in bit position 3.

1st Operand	1 1 0 1 0 0 1 1
2nd Operand	____________

• • •

0 0 0 1 0 0 0 0

4. It is desired to change all of the program switches in a byte. Show the necessary 2nd operand and the expected result.

1st Operand	1 0 1 1 0 1 1 1
2nd Operand	____________
Result	____________

• • •

2nd Operand	1 1 1 1 1 1 1 1
Result	0 1 0 0 1 0 0 0

5. Assume that bit position 7 of a byte is a program switch. In order to change the setting of this program switch, bit position 7 of the 2nd operand must be ______ (0/1).

• • •

1; See below.

1st Operand	0 1 0 0 0 0 1 1
2nd Operand	0 0 0 0 0 0 0 1
Result	0 1 0 0 0 0 1 0

6. The letter X is used for the mnemonic of an "e____________ o____" instruction.

• • •

"exclusive or"

Like the "and" and "or" instructions, the "exclusive or" instruction uses four formats.

7. Given the mnemonics, indicate the formats of the four "exclusive or" instructions.

Mnemonic	Format
XR	__ __
X	__ __
XI	__ __
XC	__ __

• • •

Mnemonic	Format
XR	RR
X	RX
XI	SI
XC	SS

8. Just like the "and" and "or" instructions, the "exclusive or" instruction will cause the condition code to be set to ______ (0/1/2/3) for an all-zero result.

• • •

0

9. For a non-zero result, the "exclusive or" instruction will set the condition code to ______ .

• • •

1

END OF SKIP OPTION

Returning to the sample problem we find that we are ready to move the required data items into their respective fields in the output area. This involves instructions you already have learned.

1. Which output fields do not require editing? (Symbolic Names). ____________

•••

LNO; LNAME

2. Move the proper data items to these output fields. If you have forgotten where these data items are located, check your list of assembler instructions (DC's and DS's).

•••

```
MVC   LNO,EMPNO
MVC   LNAME,ENAME
```

3. All the other fields to be printed will contain quantitative data items that are more readable if they contain punctuation. Therefore, they must be ________ as they are moved into the output area.

•••

edited

4. How are punctuation and suppression of high-order zeroes accomplished in an edit pattern (your own words)? ____________

•••

By an edit mask called a pattern.

5. A pattern is a special edit mask that punctuates and suppresses high-order zeroes in a data field to be printed. Which is true?

   1. The pattern field is moved into the data field.
   2. The data field is moved into the pattern field.

•••

The data field is moved into the pattern field.

6. Moving the data into the pattern field (does/does not) ________ destroy the pattern.

•••

does

7. How do we keep this from happening, since obviously we want to use the pattern many times (your own words)? ____________

____

•••

Move the pattern into a work area and edit the data into that area.

8. Using the following hexadecimal numbers to provide the desired characters, set up the pattern for editing the year-to-date gross figure into LGROSS in the output area. Call it PATRN1. Don't forget to check the source field for a fill-in zero that is supplied during the PACK operation if the number of source digits is even.

Fill Character	hexadecimal 40
Digit Select Character	hexadecimal 20
Significance Start Character	hexadecimal 21
Comma	hexadecimal 6B
Period	hexadecimal 4B

The significance start character should precede the decimal point. The edited data will never be negative.

•••

```
PATRN1  DC  X'40 20 20 6B 20 20 21 4B 20 20'
```

9. You will recall that you made the output area fields large enough to hold the edit patterns. You can deduce that, to preserve the edit patterns we will do the editing in ____ ________ ______ .

•••

the output fields.

10. Write the instruction to move PATRN1 to LGROSS in the output area.

•••

```
MVC   LGROSS,PATRN1
```

The next step is to edit the year-to-date gross amount into the output area.

Before you write the edit instruction, remember these points:

- Year-to-date gross was packed back into its original field.
- We are concerned with only that part of the original field that contained the packed data.
- Address adjustments will be necessary to address the byte in which the first digit(s) of the packed data are located.

- Data packed into its original location will be right-justified; that is, it will occupy the rightmost bytes in the field.
- The high-order (leftmost) byte that contains packed data will have a fill-in zero in its leftmost four bit positions if the original amount contained an even number of digits. It will not contain a fill-in zero if the original amount contained an odd number of digits.

11. Write the edit instruction that will edit year-to-date gross into the pattern in the output area.

●●●

ED LGROSS, YTDGRS+3

If you had trouble determining the correct amount of address adjustment to use in addressing YTDGRS, consider the following:

YTDGRS is 7 bytes long. The input data field is 7 bytes long also. After data is read in, YTDGRS looks like this:

FX	FX	FX	FX	FX	FX	FX

We packed the contents of YTDGRS back into YTDGRS. After packing, YTDGRS looked like this:

00	00	00	XX	XX	XX	XS

None of the subsequent operations on YTDGRS changed its format.

LGROSS, the output field, is 10 bytes long. PATRN1 also is 10 bytes long. LGROSS looked like this after the pattern was moved in:

40	20	20	6B	20	20	21	4B	20	20

By addressing YTDGRS+3 we got this result:

```
YTDGRS    00  00  00  XX  XX  XX  XS
LGROSS        b  X  X  ,  X  X  X  .  X  X
              ↑
              Fill Character
```

12. The next field to be printed is ________________ .

●●●

The employee's Federal Withholding Tax (LFEDWH)

13. Is LFEDWH a numerical field? ______ Will it require editing? ________ .

●●●

yes; yes

14. The problem statement has the edited format of all output data fields. Is LFEDWH punctuated the same as LGROSS or differently? ________________

●●●

The same. (They both have a comma and a decimal point.)

15. The YTD Federal withholding tax, which will print in LFEDWH, contains an (even/odd) ________ number of digits?

●●●

even

16. Develop the pattern necessary to edit the YTD Federal withholding tax amount into the output area, keeping in mind the fill-in zero in the source field.

●●●

40	20	20	6B	20	20	21	4B	20	20

17. Compare the pattern you have just developed with PATRN1. They are (the same/different) ______ .

●●●

the same

18. Since they are the same, you can use PATRN1 to edit YTD Federal withholding tax into the output area. Can you explain why the edit patterns are the same even though the YTD Federal withholding tax field is one digit shorter than the YTD Gross field?

●●●

PATRN1 contains two digit select characters preceding the comma, to edit the two high order digits in the source field. The pattern for YTD Federal withholding tax must also contain two digit select characters preceding the comma; one to edit the fill-in zero and one for the first significant digit in the source.

You will remember that patterns containing a fill character, and a digit select character to edit a fill-in zero, will result in two spaces preceding the printed data on the report.

Look at the printer spacing chart.

19. How many spaces do we want between YTD Gross and YTD Federal Withholding tax? ________

●●●

3

20. We arranged for one of these spaces by writing an assembler instruction to reserve a byte in the output area. The other two are obtained from (your own words) ______________________________

• • •

the fill character and the first digit select character in PATRN1

You can see that by spacing the printed fields appropriately we can use the same edit pattern for more than one field.

21. Write the instructions to edit the employee's Federal Withholding Tax into the correct location in the output area.

• • •

```
MVC    LFEDWH, PATRN1
ED     LFEDWH, YTDWH+2
```

22. By looking at the output area DS, you can see that there are three more fields into which data is to be edited. They are (symbolic names) ______________ ________________________

• • •

LFICA, LCURGR, LNPAY

23. See if you can write the instructions that will edit the correct data into these fields. Consult the problem statement for the names of accumulators in which edit data is stored. If you need additional edit patterns, create them. Don't look at the answers to this question until you have written all the instructions you believe are necessary.

• • •

You need one additional edit pattern:

```
PATRN2  DC  X' 40 20 20 21 4B 20 20'
```

These six instructions will accomplish the remaining editing:

```
MVC    LFICA, PATRN2
ED     LFICA, YTDFICA+2
MVC    LCURGR, PATRN1
ED     LCURGR, GROSS+2
MVC    LNPAY, PATRN2
ED     LNPAY, NETPAY+1
```

24. Now you are ready to write the line you have just set up. Write the instruction. Remember that register 10 is the linking register.

• • •

```
BAL    10,WRITE
```

25. Check Figure 19 to determine the next step, then write the appropriate instruction.

• • •

```
BC     15,READCD
```

26. You have written the bulk of the instructions for the sample problem. However, there are still a couple of short subroutines to be coded. You will recall that you left some lines blank on your coding sheet. You must now write the coding for (block number 5, Figure 19) _________ on these lines.

• • •

E2

27. The instruction preceding the first line you skipped is ______________ .

• • •

```
BC     1,OFF
```

You will recall that this instruction branches to an out of line routine that turns off the switch. We said the purpose of the switch is to bypass the sequence check on the first card.

For all cards following the first card the program will not branch, because it never repeats the instruction that turns the switch ON.

28. Look at the instructions you wrote for block E3, Figure 19. What do they do? ____________________

• • •

Pack and convert to binary the employee number from the input record, and place it in general register 2.

29. Why did we do this?

• • •

To set up the employee number so it can be compared to the employee number from the next card.

Now that we have processed the first card and have read the next one, we are ready to sequence check. You will recall that we said we will do this by comparing the employee numbers register-to-register.

30. What must be done to the employee number from the second card to prepare it for register-to-register comparison?

• • •

It must be packed, converted to binary, and stored in a general register.

31. Write the instructions to ready the employee number from card two for comparing. You may use the same work area you used to pack the employee number from the first card. Put the number in general register 3.

• • •

PACK NUM,EMPNO; CVB 3,NUM

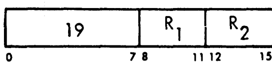
We are now ready to compare the contents of registers 2 and 3, to check the sequence of the input file.

The instruction you will use is called Compare.

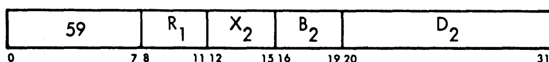
The compare instruction differs from the previous compare instruction you learned, in that the fields to be compared are in binary form rather than packed decimal. Read about the instruction in the following material.

Compare

CR R_1, R_2 [RR]



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



CR and C:
The first operand is compared algebraically with the second operand and the result determines the setting of the condition code.

- Both operands are 32 bit signed integers.

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

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

Program Interruptions:
Addressing (C only)
Specification (C only)

EXAMPLES:

Name (1–8)	Operation (10–14)	Operand (16–)
	CR	4,6
	BC	13,SEQ
	C	11,B

32. Write the instruction that will compare the employee number from the second card to that from the first card.

• • •

CR 3,2

33. If the cards are in sequence (second employee number greater than first employee number) the condition code setting is (0/1/2) ________ .

• • •

2

34. If we had reversed the order of the operands in the compare instruction and the cards are in sequence, the condition code setting would be (0/1/2) ______.

• • •

1

Assume the following values in registers 2 and 3

Reg 2 14987+
Reg 3 14693-

35. What is the condition code setting if this instruction is executed?

CR 3,2

• • •

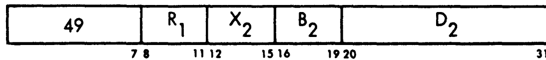
1

SKIP OPTION

If you are sure you can predict the results of compare register instructions regardless of the absolute values and algebraic signs of the factors involved, you may skip to the point at which we resume discussion of our program. If you are not sure, you should read the following frames. A related instruction, Compare Halfword, also is discussed here.

Compare Halfword

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



The first operand is compared algebraically with the halfword second operand and the result determines the setting of the condition code.

- The first operand is a 32 bit signed integer.
- The second operand is treated as a 32 bit signed integer by propagating the sign value through the 16 high-order positions (16 zeroes for a positive number and 16 ones for a negative number).
- The second operand must be on halfword integral boundary.

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

Program Interruptions:
Addressing
Specification

EXAMPLE:

Name (1–8)	Operation (10–14)	Operand (16–)
	CH	7,LIMIT

COMPARE INSTRUCTIONS

1. To indicate a "compare" instruction, the mnemonic uses the letter _____. To compare a halfword in storage to the contents of a general register you would use the mnemonic ______ . To compare the contents of one register to another, you would use the mnemonic ______ .

●●●

C; CH; CR

2. The 1st and 2nd operands are __________ (changed/unchanged) by the compare operation. The operation is used to set the PSW _______ ______.

●●●

unchanged; condition code

3. A "compare" instruction would usually be followed by the instruction " _______ __ ______ ."

●●●

"branch on condition"

4. If a compare operation shows that both operands are equal, the condition code would be set to ___ .

●●●

0

5. A condition code of 1 indicates a low compare. In other words, the _______ (1st/2nd) operand is less than the ______ (1st/2nd) operand.

●●●

1st; 2nd

6. A condition code of 3 is impossible after a compare but a code of 2 would indicate that the _________ (1st/2nd) operand is high.

●●●

1st

7. The comparison is algebraic. In other words, the operands are considered as signed integers. A negative operand would be __________ (less/greater) than a positive integer.

•••

less

8. Given the following CR instruction, indicate the condition code setting.

CR

19	4	7

Reg 4   A 0 F 1 0 F F F

Reg 7   7 F F F F F F F

Condition Code ______

•••

1; The 1st operand (reg 4) is low because it is a negative number which is algebraically less than a positive number.

9. Given the following CH instruction, indicate the condition code setting.

CH

49	4	0	1	00F

Reg 4   7 F F F 7 F 7 0

Main Storage   7 F F F

Condition Code ______

•••

2; The halfword is expanded to a fullword by sign propagation. Then the two fullword operands are algebraically compared.

END OF SKIP OPTION

Returning to the sample problem, we see that, according to Figure 19, we want to branch out of the primary sequence and print a message if the input records are out of sequence.

1. To be in sequence, the employee number from the second card must be greater than the one from the first card. What other relationships are possible?

•••

The second card employee number could be equal to or less than the first card employee number.

2. You will recall that we wrote an instruction that, rather than test for one set of conditions, tests for the alternatives. Write an instruction that will test for the alternatives to the second employee number being greater than the first. Branch to an instruction called SEQ if either of the alternatives exist.

•••

BC 12,SEQ

SEQ is the name of the first instruction in a routine that prints a message indicating an out of sequence condition. You will code that routine shortly.

3. Meanwhile assume that card number 2, the one just read, is in sequence. It will be processed, and then another card, number 3, will be read. Card number 3 must be sequence checked. It should be checked against card number (1/2) ______

•••

2

Each card must be checked against the card immediately preceding it. Therefore, we must set up register 2 and 3 so the employee numbers will be correctly compared.

4. For any given comparison, the number from the preceding card is in register (2/3) _____ . The number from the card just read is in register (2/3)

• • •

2; 3

5. To set up for the next comparison, then, we must move the employee number from each card, as it is read and found to be in sequence, from (register 3 to register 2/register 2 to register 3) ____________________________

• • •

register 3 to register 2.

6. Write an instruction that will set up the registers correctly for the next comparison. _______________

• • •

LR 2,3

As each card is read, the employee number is read into register 3 for comparison. After the comparison, the employee number is then placed in register 2, so the next card read can be compared against it.

One other routine remains to be coded. If a card is out of sequence, we want to bypass processing it and print a message identifying the card and stating that it is out of sequence. (Block E1, Figure 19.)

7. The first instruction of this routine will be named _______ . (See the instruction that tests the condition code after the compare operation.)

• • •

SEQ

8. What is the first thing to be done in preparing for a print operation? ______________________________

• • •

Clear the print area.

Remember that this is an out-of-line routine. It should be coded on a separate coding sheet, not in the primary sequence of instructions.

9. Write an instruction that will clear the output area. You previously wrote this same instruction elsewhere in the program. Give this instruction the correct name.

• • •

SEQ XC OUTPUT,OUTPUT

10. The out of sequence card must be identified. Select the data item from the card, that serves to identify it, and move it to its position in the output area.

• • •

MVC LNO,EMPNO

Having identified the out of sequence card, we must print a message indicating the out of sequence condition. This message could be of any length that would fit into the remaining output area. For convenience, however, we can make it fit into one of the existing output fields.

11. Write an assembler instruction that will set up a constant that says OUT OF SEQUENCE. Name the constant SEQERR.

• • •

SEQERR DC C'OUT OF SEQUENCE'

12. Determine which output field will hold the message, and move the message into the field.

• • •

MVC LNAME,SEQERR

13. Write the instructions that will cause a branch to a routine to print the information and then branch back to the appropriate point in the program.

• • •

BAL 10,WRITE; BC 15,READCD

You have coded the payroll program. The only remaining thing to do is to check the margins of all coding sheets to determine if there are any uncoded out-of-line routines, or branch instructions which have NOT yet had the branch address included. Since there are none in your program, you may erase all margin checks.

Now that you have completed the coding for the Ajax payroll problem you will want to check your coding sheets against a set on which the coding is known to be accurate. Figures 23 through 30 show the coding as you should have written it. Compare your coding sheets against these illustrations. If any serious discrepancies exist, consult your advisor.

Note that Figure 23 shows the I-O coding for the problem, which you were not required to write. You have no coding sheet with entries that match Figure 23.

You have completed the System/360 Assembler Language self-study course. You should now contact your advisor and make arrangements to take the final examination.

The final examination is a coding exercise that asks you to demonstrate your knowledge of Assembler Language by coding portions of a problem that already has been flowcharted. You will not be required to program the I-O routines, and for the most part you will be told what coding strategy to use and the nature of the instructions that are to be coded.

HOW TO PREPARE FOR THE TEST

If you are uncertain of your understanding of any of the topics and instructions you have studied it would be wise to review them before taking the examination. In addition, you should review these topics:

- The definition of input and output areas, with emphasis on the space that must be allowed when an output field is edited.
- The methods of specifying "immediate data" in the second operand of an SI type instruction. Be sure that you know how to code it as a self-defining constant (a "literal") in hexadecimal, in binary or decimal, and as a character.
- The methods that you used for clearing storage.
- The setting on, testing, and turning off of a switch.
- Rounding, with emphasis on address adjustment and control of a move type operation by the use of explicit operand lengths.

You need not, and should not, try to memorize the functions of machine instructions. This information will be provided for you, for every instruction illustrated by the sample programs, when you take the test. You will also have free use of your Reference Data card, with <u>your</u> notes on it.

Overall, the best way to prepare for the test is to review the kinds of operations on data that the problems required. Use those operations as reminders of the machine instructions that performed them.

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601
(USA only)

IBM World Trade Corporation
821 United Nations Plaza, New York, N.Y. 10017
(International)

R29-0232-6

System/360 Assembler Language Coding Standard and Decimal Instructions Printed in USA R29-0232-6