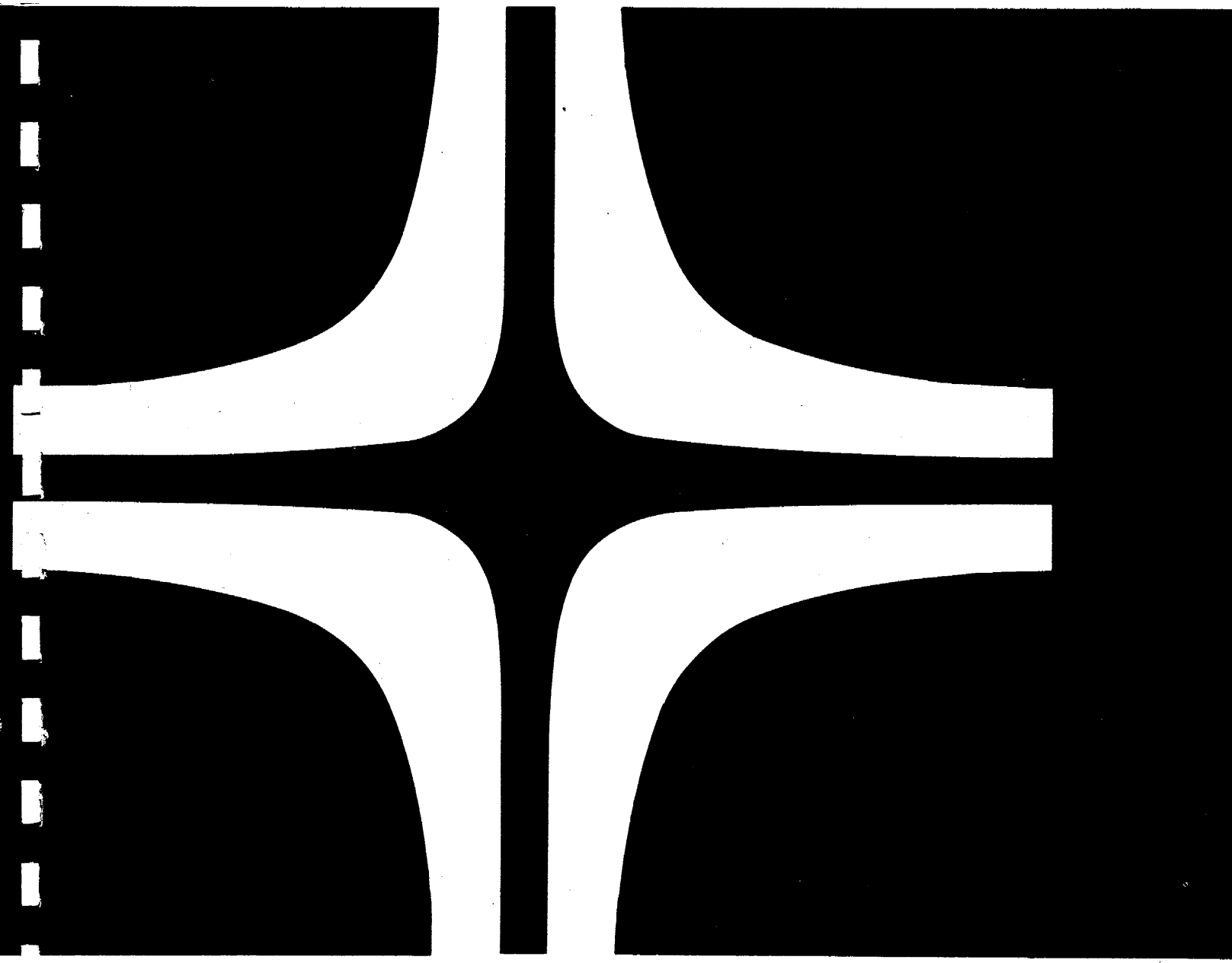
UNIVAC 9000 CARD ASSEMBLER

Programmed Instruction Course

Book 2 - Assembler Language



SPERRY UNIVAC
COMPUTER SYSTEMS

EDUCATION CENTER

UE-686.2

UNIVAC 9000 CARD ASSEMBLER PROGRAMMED INSTRUCTION COURSE

ASSEMBLER LANGUAGE

Book 2

UE-686.2

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

INTRODUCTION

This text is Book 2 of a series of programmed instruction manuals designed to teach 9000 Series Card Assembler programming. Successful completion of Book 1 (UE-686.1) and the self-test evaluation are prerequisites for starting Book 2.

In this text, the novice acquires the basic Card Assembler programming knowledge and coding skill required to successfully complete the diagnostic exercises and terminal problem in Book 3.

1. PREVIEW

The Univac 9200/9300 Assembler Programming System helps the user prepare programs by means of Software Programming Libraries provided by Univac.

The acquisition of input data from peripheral devices such as the card reader or magnetic tape and the transfer of the processed output data to peripheral devices such as the printer, card punch, or magnetic tape are important aspects of programming. To provide for input/output functions on the Univac 9200/9300 systems, the programmer can use a prewritten input/output programming system known as the Input/Output Control System (IOCS). This system, developed by Univac, simplifies control of input/output devices and reduces the programmer's task of providing for input/output procedures.

This section of the course will discuss information the programmer must provide in his coding when he wishes to use the Input/Output Control System to perform input/output functions. IOCS consists of two parts: the input/output macro routines and the macro instructions specified by the user program to communicate with input/output routines. The macro instructions used to communicate with the input/output routines are Imperative Macro Instructions; those used to generate the input/output routines are Declarative Macro Instructions.

The macro instructions used to communicate with the input/output routines are _____ Macro Instructions.

Imperative

The system provided by Univac to reduce the programmer's I/O commands is the _____ .

IOCS

The macro instructions used to generate the I/O routines are the _____ Macro Instructions.

Declarative

2. The user is provided with a complete set of routines for controlling all input/output operations that may be required by the system. Since not every source program will require every routine or its variable functions, Univac provides a Macro Generator program which is capable of specializing each input/output routine according to the particular user's stated requirements.

   The Macro Generator reads Definition Statements (Declarative Macro Instructions) made by the user describing the input/output operations required by the application and punches them into cards in the Assembler Source Language format. They may then be assembled as part of the user's source program, or assembled separately and linked with the user program at execution time.

   These Imperative Macro Instructions are related to the input/output routine to which they refer by means of a file name. At Assembly time, the IOCS routines become part of the user program.

   At execution time, when an input/output function is desired in the program, control is passed from the user-logic portion of the program to the IOCS portion of the program.

   Check the following statements as true (T) or false (F):

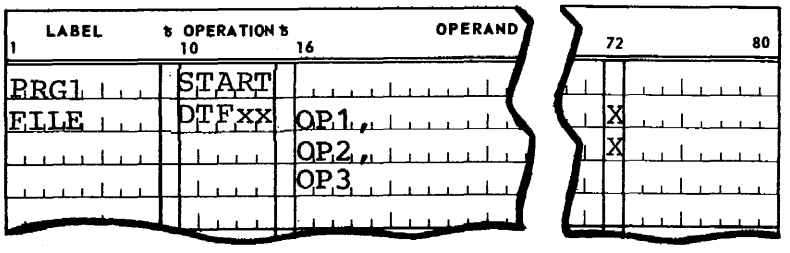
T	F		
☐	☐	Univac provides a Macro Generator program which is capable of specializing each input/output routine according to the particular user's stated requirements.	True
☐	☐	The Macro generator reads the user's Declarative Macro Instructions and punches them into cards in the Assembler Source Language format.	True
☐	☐	Data files are brought in and processed at assembly time.	False
☐	☐	IOCS routines may become part of the user program at assembly time.	True

3. A file is a collection of logically related records that are to be brought in by an input device or written out to an output device. To use the IOCS system, the programmer must supply certain key information about each input and output file used in the program. This file information must be defined in a Declarative Macro Instruction called a Define-The-File statement (DTF). Match the following: A. DTF _____ Input/output routines. B. IOCS _____ A Declarative Macro Instruction. _____ A statement coded by the programmer to supply file information. _____ A Univac supplied macro program.	 B A A B
4. The DTF statement is a macro instruction used to describe one file. A DTF statement indicates the device to be used and the type of processing to be performed on the file. It also specifies the memory areas that are to be allocated for the input or output data. Whenever data is brought into memory from a peripheral device, an assigned area of storage must be available for that data. This area is called an input area. Similarly, when data is written to a peripheral device, it must be sent from an area of storage known as the output area. How many DTF statements are required to perform the following sequences? _____ ● Read from a card file. ● Perform calculations. ● Produce a printed report file. How many peripheral devices are required? _____ How many input areas of storage are required? _____ How many output areas of storage are required? _____	 Two Two One One

5. The START directive defines the program name and tentative starting location. It must precede all other program statements in the source deck except comments.

   All DTF statements must be placed directly after the START instruction and must precede the main coding.

   The general format for coding this macro is as follows:



   Each file must be identified by a unique filename not more than four characters in length.

   The operation field must contain the Assembler mnemonic for the Declarative macro which is_____.

   DTF

   Entries in the operand field are selected by the programmer.

   Each operand (except the last) must be followed by a comma. A character must be inserted in column 72 of the coding form to indicate continuation.

   The maximum length of a filename is _____ characters.

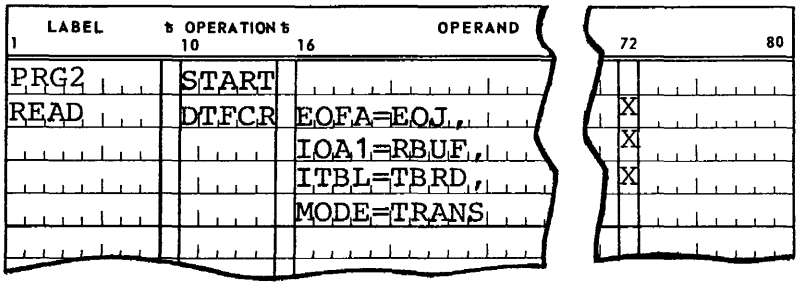
   four

   All DTF statements must be placed directly after the _____ instruction.

   START

6. The DTFCR Declarative Macro Instruction is used to define the file for the card reader. The rules for coding the DTFCR statement are described in Panel 1 on page 2-115.

   Examine the DTFCR statements below and answer the questions that follow.



What device will this file use? _____ Card Reader

What is the name of this file? _____ READ

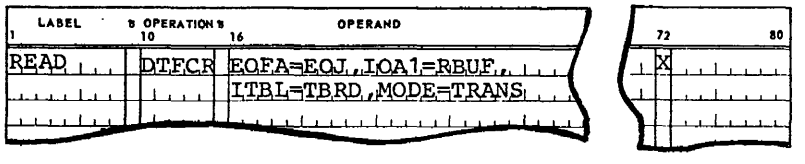
What is the name of the subroutine the programmer will want completed after all data cards have been read? _____ EOJ

What is the name of the first input area? _____ RBUF

What is the name of the translate table? _____ TBRD

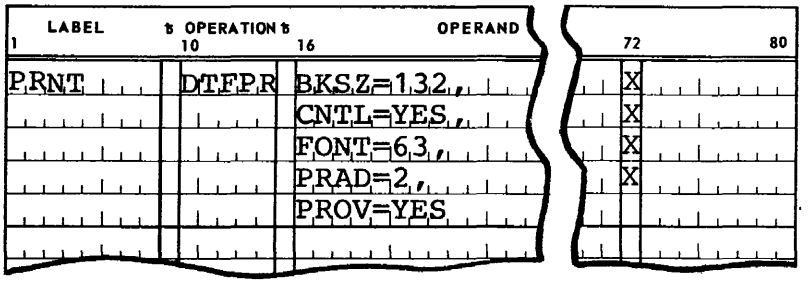
What keyword entry specifies that a translate table is to be used? _____ MODE

The above DTFCR may also be coded serially as follows:



7. The DTFPR Declarative Macro Instruction is used to define the file for the Printer. The rules for coding the DTFPR statement are described in Panel 2 on page 2-116.

   Examine the DTFPR statements below and answer the questions that follow:



What device will this file use? _____ Printer

What is the name of this file? _____ PRNT

How many print positions per line are specified?

_____ BKSZ=132

Will there be additional spacing? _____ CNTL=YES

Will a 48-character or 63-character print bar be used?

_____ FONT=63

Will there be single spacing or double spacing? _____ double (PRAD=2)

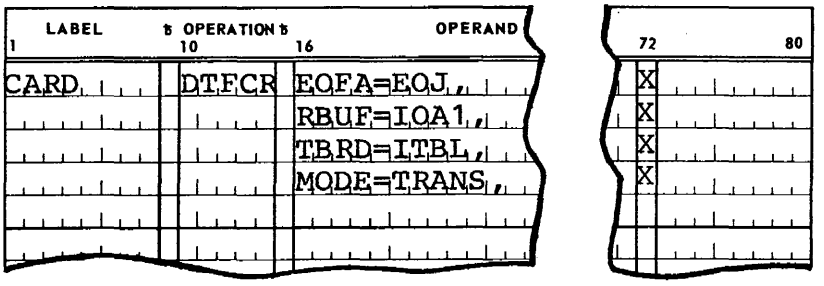
Is there provision for control of an overflow condition?

_____ PROV=YES

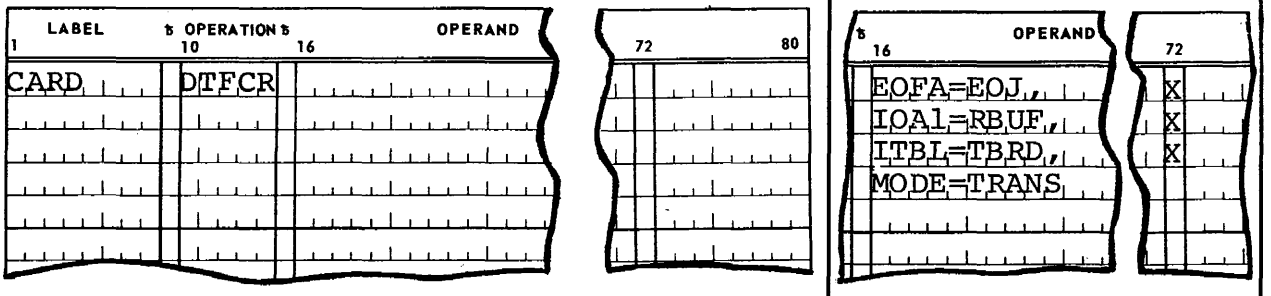
8. Match the following Keywords and corresponding response functions:

1. BKSZ	_____	Specifies symbolic address of input translation table.	8
2. CNTL	_____	Specifies symbolic address of output translation table.	10
3. FONT	_____	Provides for handling printer overflow.	5
4. PRAD	_____	Provides for line spacing or skipping control.	2
5. PROV	_____	Specifies number of print positions.	1
6. EOFA	_____	Specifies spacing advance after printing.	4
7. IOA1	_____	Specifies either 48 or 63 print bar character set.	3
8. ITBL	_____	Specifies that card codes require translation.	9
9. MODE	_____	Specifies symbolic address of input area.	7
10. OTBL	_____	Specifies symbolic address of end-of-file routine.	6

9.



Correct the above DTFCR example. Use the coding form below.



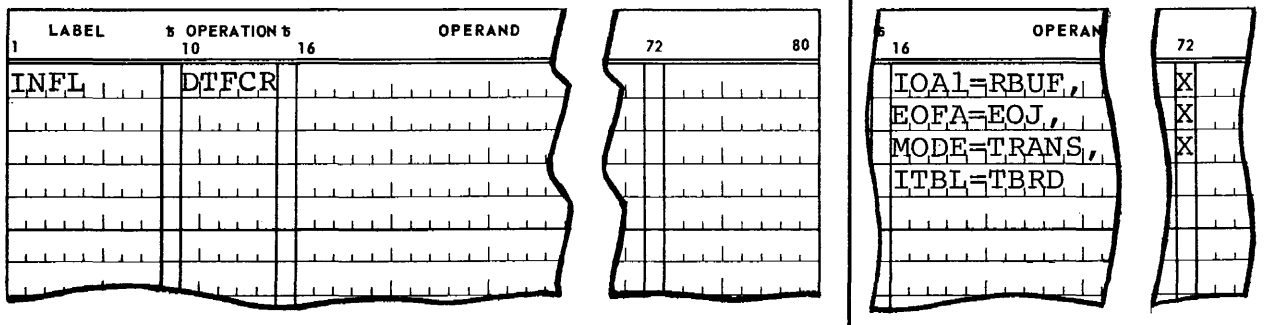
10. Check the appropriate column for each of the following keywords:

KEYWORD	DTFCR	DTFPR
PRAD		
EOFA		
BKSZ		
IOA1		
ITBL		
CNTL		
MODE		
FONT		
PROV		

DTFCR	DTFPR
	✓
✓	
	✓
✓	
✓	
	✓
✓	
	✓
	✓

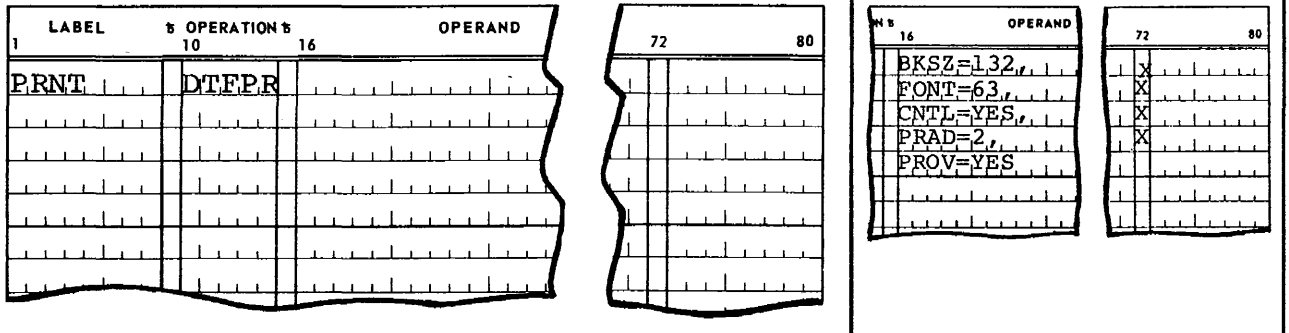
11. Complete the definition statement for a card file labelled INFL. Include the following specifications: (any sequence acceptable)

The I/O area label is RBUF.
The end-of-file routine is labelled EOJ.
The card codes require translation.
The input table label is TBRD.



Complete the definition statement for a printer file labelled PRNT. Include the following specifications:

The print-line requirement is 132 print positions.
The specified print bar has 63 characters.
Spacing and skipping is user controlled.
The printer will advance two lines after printing.
The printer will automatically advance paper to the first line on the next page when a page is completed.

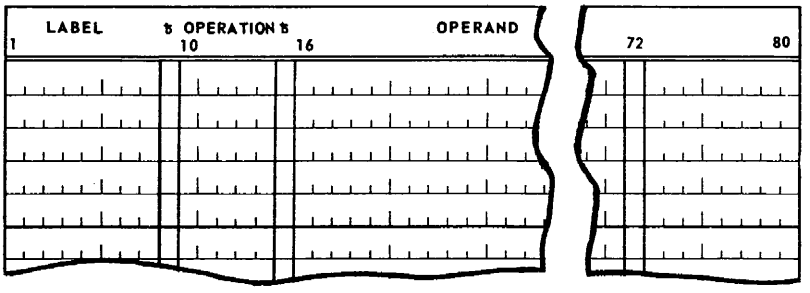


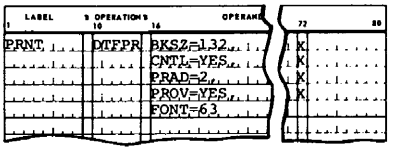
12. When defining a printer file, a programmer uses keyword entries that are unique to the printer. These entries relate to line spacing, page changing and overflow.

    The BKSZ = 132 entry provides for the number of print positions per line (max. up to 132). The printer image area, included in the first 260 bytes of memory, is the buffer area for the printer. When PUT, the Imperative Macro Instruction to print a line, is encountered in the program, the data in the printer work area is transferred to the printer image area. Printing is executed using the data in the printer image area.

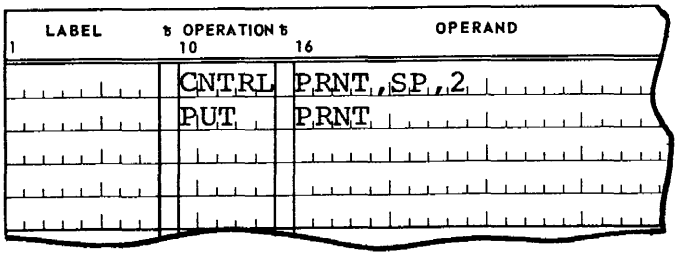
    The CNTL = YES entry signals IOCS that the programmer will control the paper advance and test for bottom of page condition by writing a CNTRL macro instruction within the main coding of the program.

    Write a complete DTFPR statement for a printer file. Assume that paper control will be directed by macro instructions written in the main coding of the program. Also, assume that there will be double spacing. Printing is to begin at the top of each form.

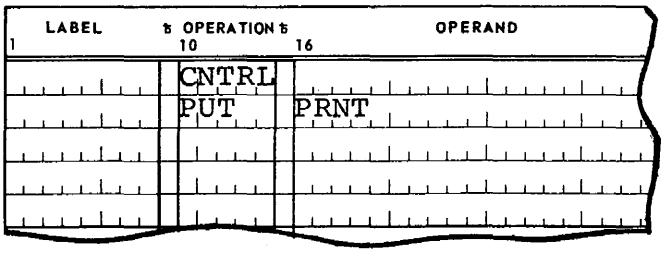


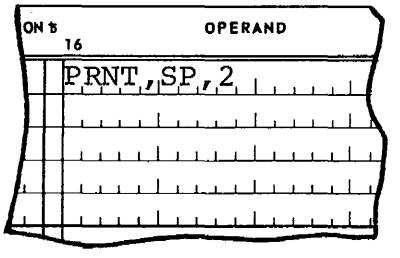


13. When the programmer uses the CNTL = YES entry in the printer's DTFPR, he must specify the format of a printout by using the CNTRL instruction in the logic or process portion of the program. Usually, it is written immediately before a PUT command. For example, assume we are producing a report named PRNT. To advance two line spaces before the next line is printed, the programmer must write the two instructions as follows:



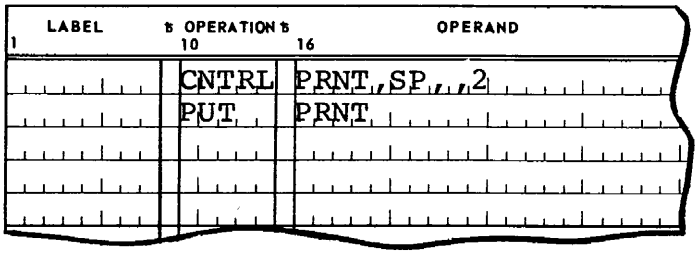
Complete the macro that will cause the paper in the printer to advance two lines before the next line is printed.





14. The CNTRL macro provides the programmer with the ability to advance the paper before or after the line is printed.

    To delay the paper advancement until after the next line is printed, place two commas in the second operand of the CNTRL instruction.



    The above code will cause the paper in the printer to advance two lines:

    ☐ before the line is printed.

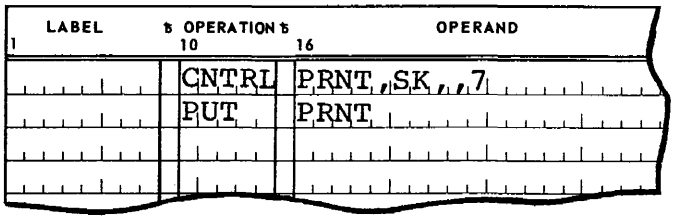
    ☐ after the line is printed.

    The CNTRL macro (spacing) can advance the paper up to 2 lines with 1 instruction.

after the line is printed.

15. Single spacing is automatically provided each time a PUT is issued. Therefore, when a report is to be single spaced, the CNTRL macro is not needed for line spacing. However, CNTRL must be used to advance (skip) the paper to the top of the next page (home paper position) or to the bottom of the current page.

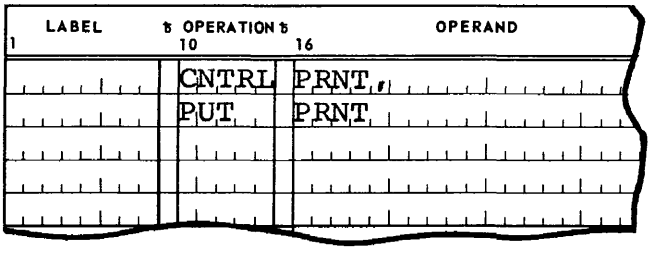
    Example:

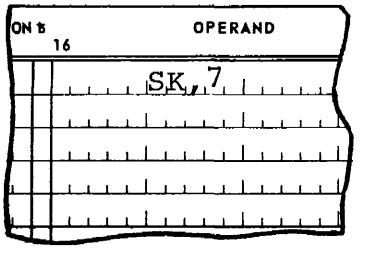


    When used with a 7 as shown above, the SK parameter skips to the top of the next page. To skip to the bottom of the current page, a 1 is written in the SK parameter.

    When the above CNTRL macro is executed, the paper in the printer advances to the top of the next page after the line is printed.

    Complete the statement below so as to cause the paper to skip to the top of the next page before the line is printed.





16. REVIEW FRAME

T	F		
☐	☐	IOCS routines called for by the programmer are assembled with the user program.	True
☐	☐	Data files are brought into memory and processed at assembly time.	False
☐	☐	The programmer supplies information about the files in the DTF.	True
☐	☐	A program with two input files and two output files requires five DTF statements.	False
☐	☐	A file name must not exceed four characters in length.	True
☐	☐	I/O areas must be defined and named in another area of the program.	True
☐	☐	The response to the keyword IOA1 = must be the symbolic name assigned to the I/O area of memory.	True
☐	☐	The keywords must be listed in a prescribed sequence.	False
☐	☐	The last entry is always followed by a comma.	False
☐	☐	The CNTL = entry is only applicable to printer files.	True
☐	☐	To cause the paper in the printer to advance two lines before printing, the programmer must write: CNTRL filename, SP, 2.	True
☐	☐	If a report is to be single spaced, the programmer must use the CNTRL macro.	False
☐	☐	CNTRL filename, SK, 1 advances the paper to the top of the next page before the line is printed.	False

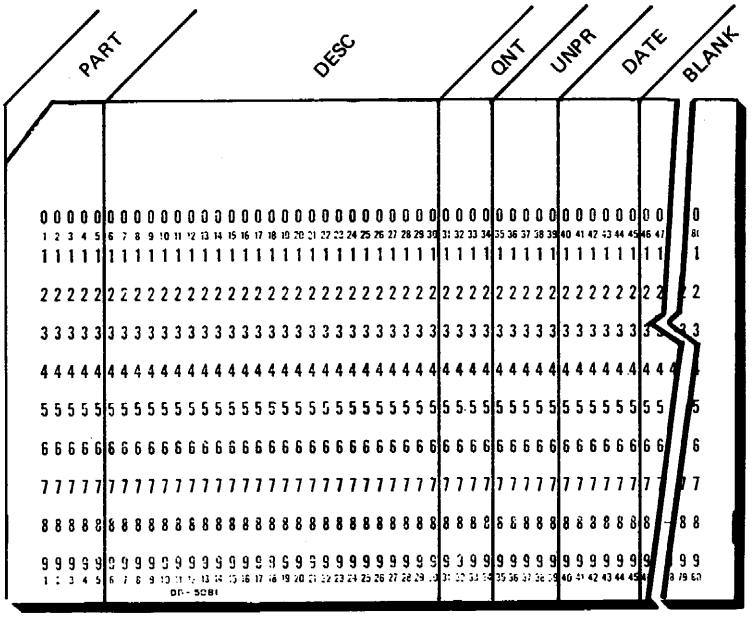
17. PREVIEW

In the DTF coding, the programmer names the input/output areas that will be needed. The Define Storage instruction reserves memory for the I/O areas.

In the following frames you will learn how to code the DS instructions to allocate memory for the input/output areas needed to process a simplified inventory problem. The input in this problem is a punched-card file. The output will be a printed report.

18. Assume that the punched-card file records have the following format:

Item	Columns	Length	Field Name
Part Number	1-5	5 cols.	PART
Description	6-30	25 cols.	DESC
Quantity on hand	31-34	4 cols.	QNT
Unit price	35-39	5 cols.	UNPR
Date	40-45	6 cols.	DATE
(unused field)	46-80	35 cols.	(blank)



How many bytes of storage must be reserved as an input area to receive the above data? _____

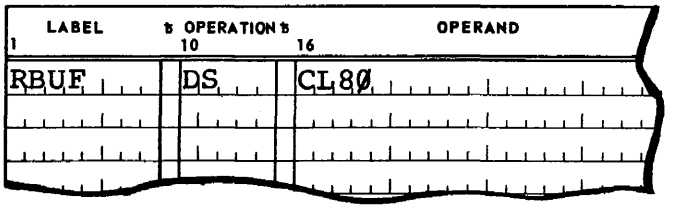
80 bytes

Assume that the input area has been previously identified in the DTFXX coding as RBUF. How many bytes of memory must be allocated for the area named RBUF?

- ☐ 45 bytes
- ☐ 80 bytes

80 bytes

19. The following DS statement allocates memory for the input area named RBUF.



Answer Yes or No to the following questions concerning above coding:

Does the tag begin in column 1? _____ Yes

Is the tag within the four-character limit provided by the Label field? _____ Yes

Is there an embedded (blank) space within the tag? _____ No

Does the tag start with an alphabetic character? _____ Yes

Does the mnemonic DS start in column 10? _____ Yes

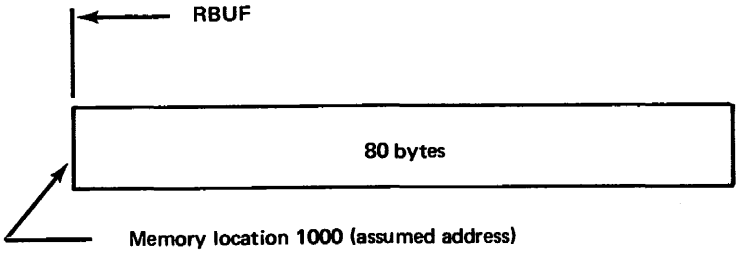
Does the operand specify the number of bytes to be reserved? _____ Yes

Are there any embedded blanks within the operand field? _____ No

Does the operand start in column 16 of the operand field? _____ Yes

Read the DS coding specifications on Panel 3 of page 2-117.

20.



What is the memory address of the first byte of the input area named RBUF?

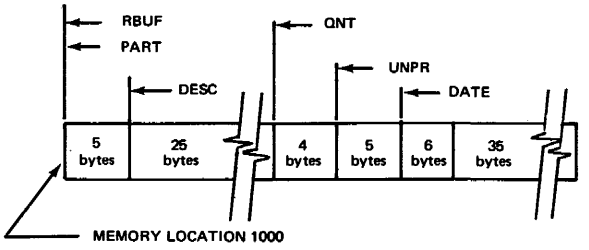
1000

What is the address of the last byte?

- ☐ 1044
- ☐ 1079

1079

21.



The above illustration shows the input area as redefined to include the field subdivisions. Complete the DS coding below to allocate memory for the records illustrated above.

	LABEL	OPERATION	OP
	1	10	16
1000	RBUF	DS	CL8Ø
		ORG	RBUF
1000	PART	DS	
1005	DESC	DS	
1030	QNT	DS	
1034	UNPR	DS	
1039	DATE	DS	
1045		DS	

OPERATION	OP
10	16
	CL5
	CL25
	CL4
	CL5
	CL6
	CL35

The above coding directs the Assembler location counter to allocate 80 bytes starting with location 1000. The ORG code then directs the Assembler to decrement the count to the starting address represented by RBUF.

What address is symbolized by the tag RBUF? _____ 1000

The input area RBUF can also be redefined by means of the duplication factor as shown below:

	LABEL	OPERATION	OP
	1	10	16
1000	RBUF	DS	ØCL8Ø
1000	PART	DS	CL5
1005	DESC	DS	CL25
1030	QNT	DS	CL4
1034	UNPR	DS	CL5
1039	DATE	DS	CL6
1045		DS	CL35

22. The ORG instruction resets the Assembler location counter to the starting address of the tag in the Operand field. (Read the ORG instruction specifications described in Panel 4 page 2-118).

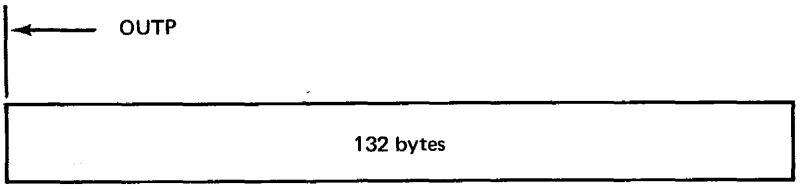
When the program is assembled what address is assigned to each of the following tags?

RBUF _____	1000
PART _____	1000
DESC _____	1005
QNT _____	1030
UNPR _____	1034
DATE _____	1039

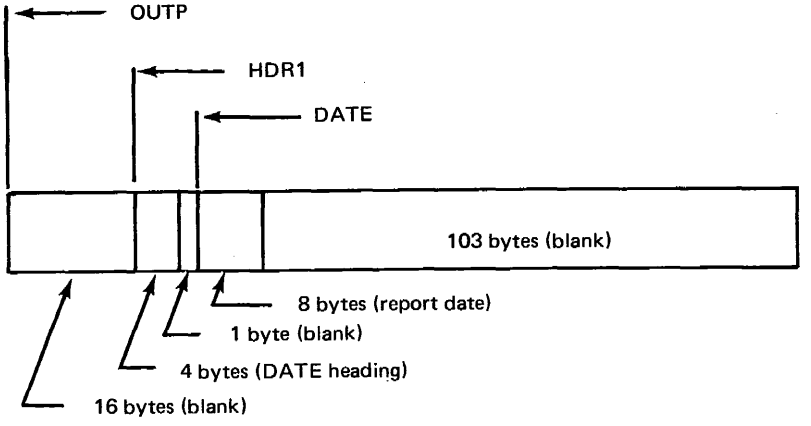
Note:

In the printed Assembly listing, addresses and location counter references are represented as hexadecimal values. However, since we have not yet studied the hexadecimal numbering system, we will represent addresses as decimal values.

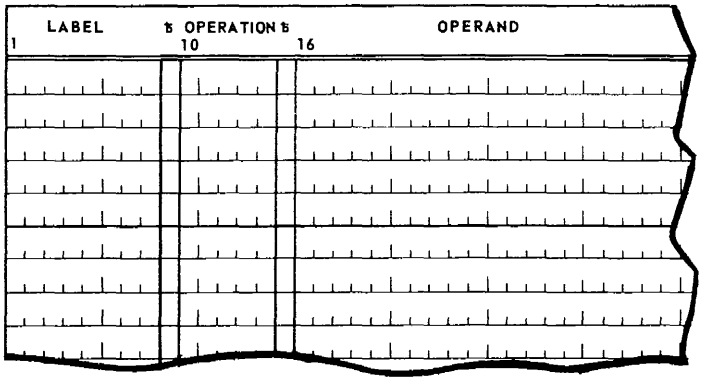
23. The inventory report is to be printed in the format illustrated in Panel 5 on page 2-119. This requires the assignment of an output area in memory that will receive the heading and data fields of each line to be sequentially printed. We will assign the address tag OUTP to a 132-byte area as illustrated below.



The first printline will contain the heading, DATE, and the date of the report. These fields are defined in the output area as illustrated below:

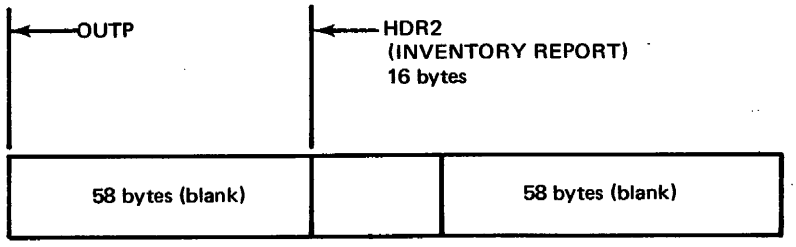


Write the DS coding to allocate memory for the fields illustrated above. Since the blank fields will not be addressed in the program, they require no address tags. However, a blank field that precedes a heading or data field must be defined to increment the location counter to the first byte location of the heading and data fields to be addressed. DS coding to define the final 103-byte blank field is not required in actual programming but should be included in this introductory problem.

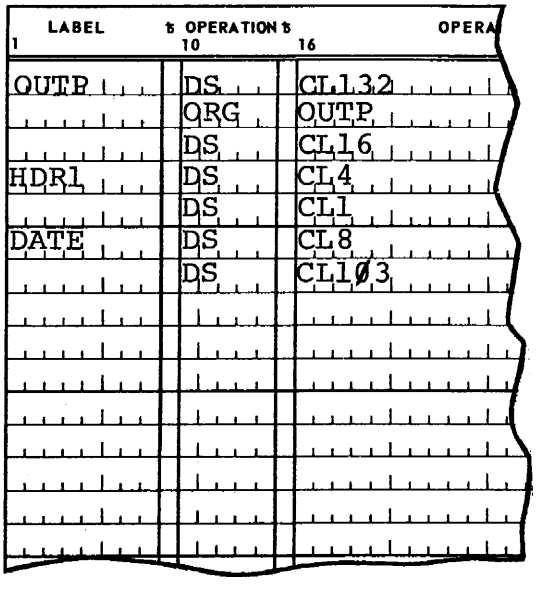


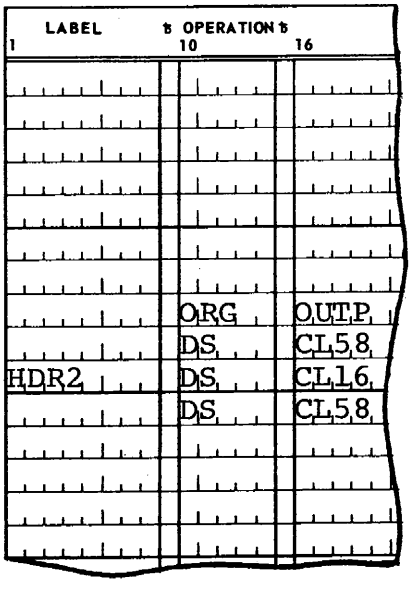
LABEL	OPERATION	
OUTP	DS	CL132
	ORG	OUTP
	DS	CL16
HDR1	DS	CL4
	DS	CL1
DATE	DS	CL8
	DS	CL103

24. The second line to be printed will contain the report title. Thus, the output area is redefined as illustrated below:

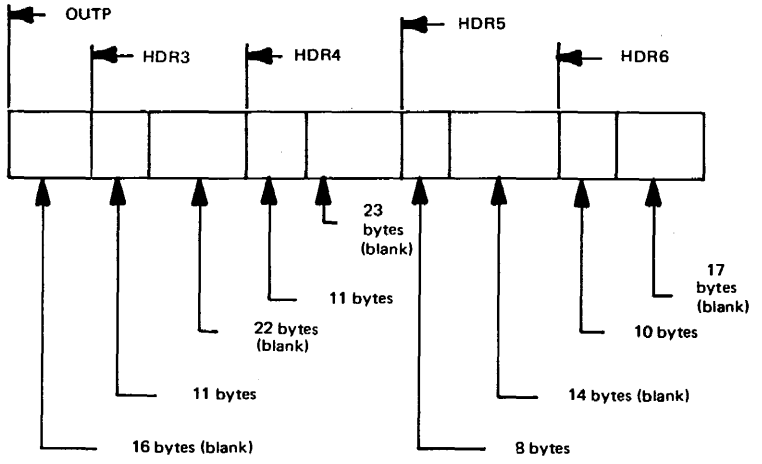


Continue the DS coding to redefine the storage area for the fields illustrated above, for the second line header.





25. The third printline will contain the column headings. Thus the output area is redefined as illustrated below:

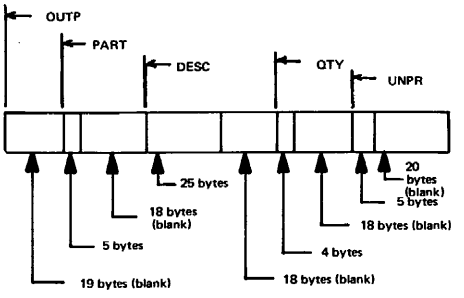


Continue the DS coding to redefine the storage area for the fields illustrated above for header positions 3, 4, 5 and 6.

LABEL	OPERATION	OPERAND
OUTP	DS	CL132
	ORG	OUTP
	DS	CL16
HDR1	DS	CL4
	DS	CL1
DATE	DS	CL8
	DS	CL103
	ORG	OUTP
	DS	CL58
HRD2	DS	CL16
	DS	CL58

LABEL	OPERATION	OPERAND
	ORG	OUTP
	DS	CL16
HDR3	DS	CL11
	DS	CL22
HDR4	DS	CL11
	DS	CL23
HDR5	DS	CL8
	DS	CL14
HDR6	DS	CL10
	DS	CL17

26. The fourth printline will contain the first line of data. The output area is redefined as illustrated below. Since the fields in the remaining printlines will have the same format as those below, further redefinition of the memory area is not required.



Continue the DS coding to redefine the storage area for the data fields illustrated above for data on the 4th print line.

```
LABEL     OPERATION     OPERAND
1         10            16
OUTP      DS            CL132
          ORG           OUTP
          DS            CL16
HDR1      DS            CL4
          DS            CL1
DATE      DS            CL8
          DS            CL1Ø3
          ORG           OUTP
          DS            CL58
HDR2      DS            CL16
          DS            CL58
          ORG           OUTP
          DS            CL16
HDR3      DS            CL11
          DS            CL22
HDR4      DS            CL11
          DS            CL23
HDR5      DS            CL8
          DS            CL14
HDR6      DS            CL1Ø
          DS            CL17
          ORG           OUTP
          DS            CL19
PART      DS            CL5
          DS            CL18
DESC      DS            CL25
          DS            CL18
QTY       DS            CL4
          DS            CL18
UNPR      DS            CL5
          DS            CL2Ø
```

27. REVIEW

Check the following statements as True (T) or False (F):

T	F		
☐	☐	The tag in a DS statement represents the symbolic address of the first byte of the defined field.	True
☐	☐	An unused field defined by a DS statement must be named.	False
☐	☐	The ORG instruction is an Assembler-directing instruction.	True
☐	☐	The ORG instruction resets the location counter to permit a storage area to be redefined.	True
☐	☐	A tag can be used more than once in the Label field of the coding form.	False
☐	☐	A DS statement can be used to define storage for an I/O area only.	False
☐	☐	Printline storage requires an allocation of at least 132 bytes of memory.	True

28. PREVIEW

The programmer frequently uses constant values that must be loaded into the program at object time. A constant may be a column heading that is to be printed or it may be a numeric value that will be used in an arithmetic operation. The programmer defines a constant and specifies the required storage by coding a Define Constant (DC) instruction.

The 9200/9300 instruction set permits considerable flexibility in defining constant values. Two commonly used types of constants are discussed in the following section:

Character constants

Hexadecimal constants

Because hexadecimal notation is used in representing constant values, a brief introduction to the hexadecimal numeric system precedes the discussion of constants.

29. In the hexadecimal (hex) numbering system, 16 symbols are used to represent numeric values. Digits 0 through 9 and alphabetic characters A through F represent numeric values 0 through 15 as shown in the comparative table below:

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

A hex symbol represents how many binary digits?

Four

Binary 0100 0000 represents hex 40. What hex value represents binary 1111 1111?

FF

What is the binary representation of the hex value C1?

1100 0001

30. As in the decimal and binary numbering systems, the value of a hexadecimal numeric symbol is determined by its position. Hex positional values correspond to powers of 16 and increase by the progression of the power of 16. This is illustrated below using the hexadecimal value 1111 as an example:

	POSITIONAL VALUE			
POWER OF 16	16^3	16^2	16^1	16^0
HEX VALUE	1	1	1	1
DECIMAL VALUE	4096	256	16	1

Write the hex equivalents of the following decimal values:

4096 _____ 1000

256 _____ 100

16 _____ 10

1 _____ 1

What is the decimal equivalent of hex value 1111? _____ 4369

The decimal equivalent of the hex value FF can be calculated as follows:

F F

$15 \times 16^0 = 15$

$15 \times 16^1 = 240$

255

What is the decimal representation of the hex value F1?

_____ 241

What is the EBCDIC representation of the hex value F1?

_____ 1111 0001

31. Internally, the computer represents all printable characters in memory as in EBCDIC (as shown in Panel 6, page 2–120). Externally, in the Assembly listing, the computer converts EBCDIC values into hex values for the reading convenience of the programmer. Thus any printable character can be represented as either an EBCDIC value or a hex value.

Write the EBCDIC and hex byte representations of the printer graphic symbols ABC:

EBCDIC 

1100 0001 1100 0010 1100 0011

HEX 

C 1 C 2 C 3

Convert the printer graphic symbols 7 3 4 to EBCDIC and hex values:

EBCDIC 

1111 0111 1111 0011 1111 0100

HEX 

F 7 F 3 F 4

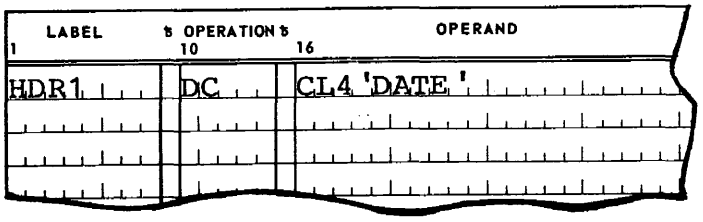
Which form of representation is easier for the programmer to code and read?

☐ EBCDIC

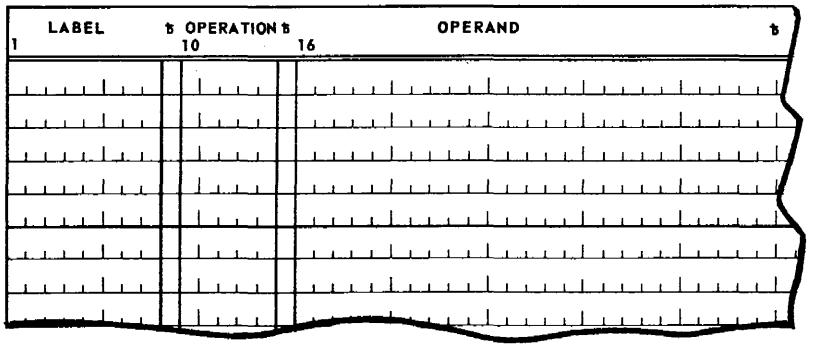
☐ Hexadecimal

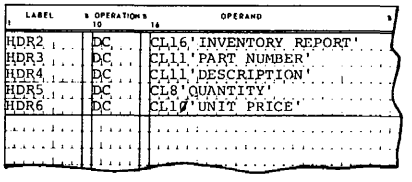
Hexadecimal

32. In the preceding section we discussed the allocation of storage for data to be printed in a monthly inventory report. Provision also must be made by the programmer to print the page headings on each page of the report. These headings can be defined as character constants and coded as shown in the following example:



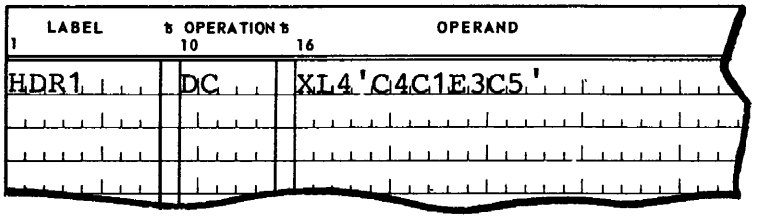
Read the rules for coding character constants in Panel 7 on page 2-121 then write the DC coding for the report title and column headings of the inventory report shown on Panel 5. Use the tags HDR2, HDR3, HDR4, HDR5, and HDR6.





33. Any constant defined as a character constant can also be defined as a hex constant. (Read the rules for coding hexadecimal constants in panel 8 on page 2-122.)

Example:



What is the printer graphic representation of the constant defined in the above DC coding (see Panel 6)? _____

DATE

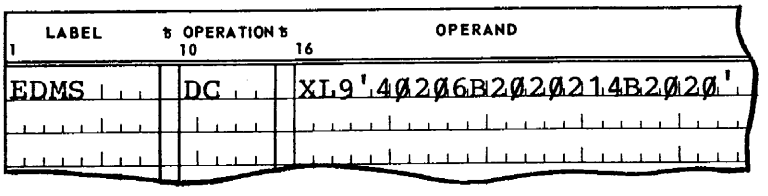
Each byte in a hexadecimal constant contains:

- ☐ one hex character
- ☐ two hex characters

two hex characters

When one or more characters in a constant cannot be represented by a printer graphic character, the constant must be defined as a hex constant.

Example:



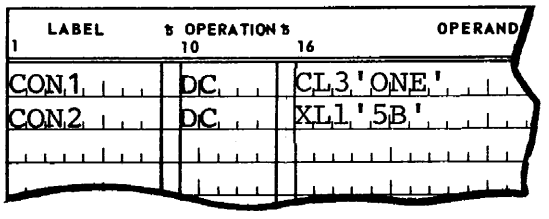
Can the above constant be defined as a character constant?

_____

No

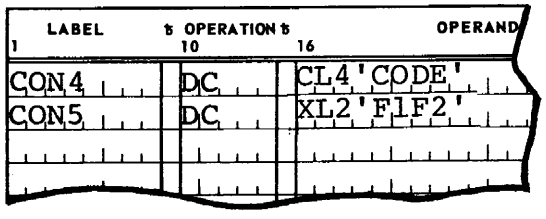
34. The Assembler converts constants in memory and represents them as hex values in the object code listing.

Examples:



LABEL	OPERATION	OPERAND	OBJECT CODE
CON1	DC	CL3'ONE'	D6D5C5
CON2	DC	XL1'5B'	5B

Refer to Panel 6 and write the object code hex representation generated by the following defined constants.



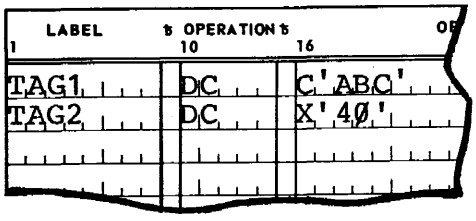
LABEL	OPERATION	OPERAND	OBJECT CODE
CON4	DC	CL4'CODE'	_____
CON5	DC	XL2'F1F2'	_____

OBJECT CODE

C3D6C4C5
F1F2

35. When the explicit length factor is omitted, the implied length of a constant is the number of characters stated within the enclosing quotation marks.

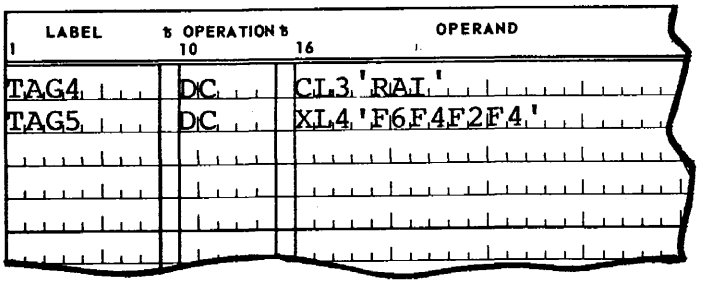
Examples:

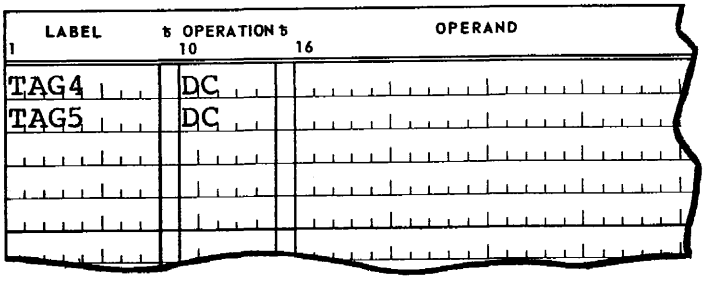


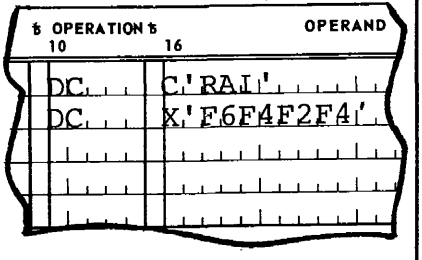
IMPLIED LENGTH

3 bytes
1 byte

Rewrite the DC coding for each of the following constants. Omit the explicit length factor:

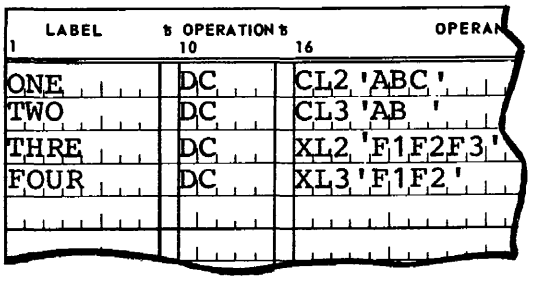






36. When DC coding specifies an explicit length that is greater or less than the implied length of a constant, the specified explicit length overrides the implied length and padding or truncation occurs as described in the coding specifications in Panels 7 and 8.

Examples:



LABEL (1)	OPERATION (10)	OPERAN (16)	OBJECT CODE
ONE	DC	CL2 'ABC'	C1C2
TWO	DC	CL3 'AB '	C1C240
THRE	DC	XL2 'F1F2F3'	F2F3
FOUR	DC	XL3 'F1F2'	00F1F2

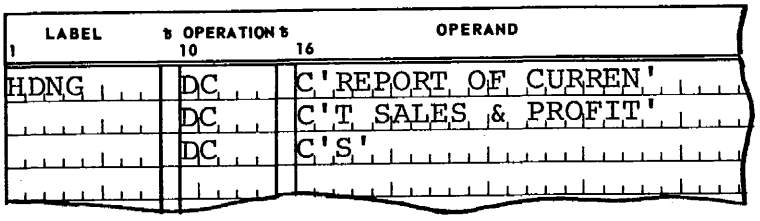
Match one of the following conditions to each of the constants coded above:

		LABEL	
A.	Truncated on left side	_____ ONE	B
B.	Truncated on right side	_____ TWO	D
C.	Padded on left side	_____ THRE	A
D.	Padded on right side	_____ FOUR	C

37. A defined constant cannot exceed 16 bytes as specified by a single DC statement. When the constant length exceeds 16 bytes, one or more additional DC statements must be specified. For example, the heading constant:

'REPORT OF CURRENT SALES & PROFITS'

is defined as follows:



How many DC statements are required to define the above constant ? _____

3 DC statements

What is the maximum length that can be specified in a single DC statement ? _____

16 characters

38. Match each constant with the corresponding object code (refer to Panel 6):

CONSTANT		OBJECT CODE
C'A'	_____	C1
X'A'	_____	0A
XL4'F2F3'	_____	5B
XL2'F2F3'	_____	40
XL1'F2F3'	_____	F2F3
X'40'	_____	0000F2F3
C'$'	_____	F3

C'A'

X'A'

C'$'

X'40'

XL2'F2F3'

XL4'F2F3'

XL1'F2F3

39. REVIEW

Check the following statements as true (T) or false (F):

T	F		
☐	☐	One hex digit represents eight binary digits.	False
☐	☐	Two hex digits can represent the contents of one byte of memory.	True
☐	☐	The memory representation of a constant in the Assembly object code listing is in hexadecimal.	True
☐	☐	Character constants can be used to define printer graphic characters only.	True
☐	☐	Hexadecimal constants can be used to define printer graphic characters only.	False
☐	☐	When defining a constant, the explicit length must be specified.	False
☐	☐	The operand field of a DC statement contains a constant value which must be enclosed within single quotation marks.	True
☐	☐	The implied length of a constant is indicated within the enclosing quotation marks.	True

40. PREVIEW

The 9200/9300 has eight general purpose registers as well as eight psuedo registers that can be used in base and displacement addressing.* The USING directive provides the Assembler the information it needs to assign the base register a value for the base register table. These psuedo registers are assumed to contain values that are multiples of 4096.

Psuedo-register	Value
0	0
1	4096
2	8192
3	12288
4	_____
5	_____
6	_____
7	28672

Psuedo register 4 in the above table contains what assumed value?

_____

16384

Psuedo register 5 contains what assumed value?

_____

20480

Psuedo register 6 contains what assumed value?

_____

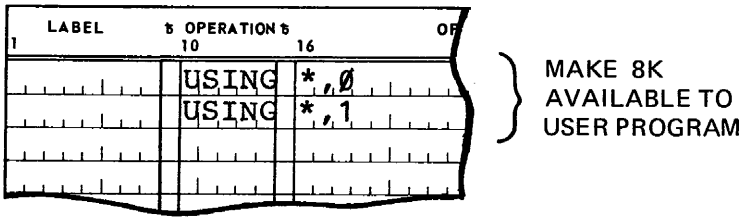
24576

*General purpose registers are numbered 8 to 15. Psuedo registers are numbered 0 to 7.

41. USING

The USING directive informs the Assembler that a specified register is available for base register assignment and that it contains a specified value. Each USING statement provides for 4096 bytes of memory. When the USING directive is used to specify pseudo registers, it provides for direct addressing by listing the modules of 4096 bytes that are available for the program being assembled. Since it may not be known in advance how many locations will be utilized, it is better to use the full capacity of the computer unless the program is small or the programmer has knowledge of the size of the assembled program.

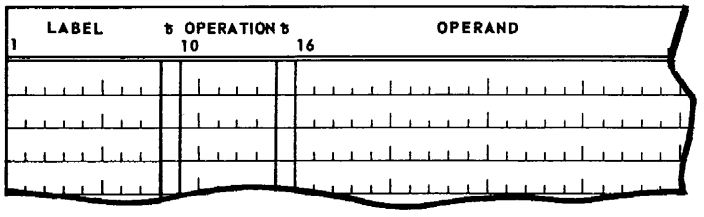
42. The format of the USING instruction is illustrated below.

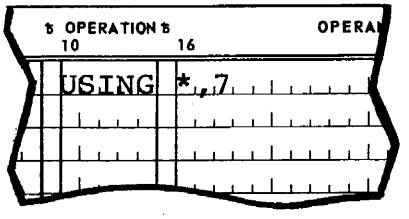


- The * (asterisk) indicates that the current value of the location counter is to be used for generating displacements.
- The Number 1 above specifies that psuedo-register 1 is to be used as a base register.

If the programmer knew that the program would need less than 4000 bytes of memory, only the first USING directive would be used. For a program requiring 8000 bytes of memory, the first two USING directives would be used.

Write a USING instruction to notify the Assembler that general register 7 will be used as a base register. The current value of the location counter will be used for generating displacements.





The USING instruction:

T	F		
☐	☐	computes base displacement addresses.	False
☐	☐	tells the Assembler which register will be used as the base register.	True
☐	☐	provides the Assembler with the information it needs to assign base registers a value for the base register table.	True
☐	☐	does not appear in the Object Program.	True
☐	☐	is an Assembler directive.	True

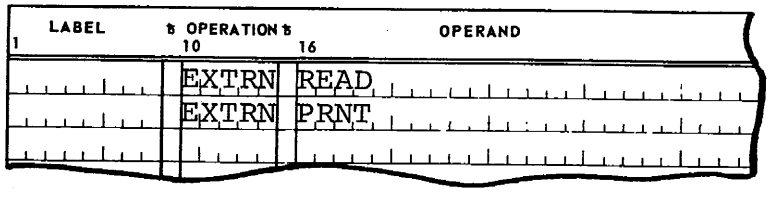
43. PREVIEW

When a job consists of more than one subprogram, the elements, which are the output of separate Assembler runs, must be combined before they can be loaded as an executable object program. This combining, or linking, is done by a utility program called the linker. The linker inserts the storage addresses for references made from one element to another and modifies addresses if an element is relocated.

44. EXTRN DIRECTIVE (Externally Referenced Symbol Declaration)

EXTRN notifies the assembler that a symbol (label) referenced in one separately assembled program will be defined in another program.

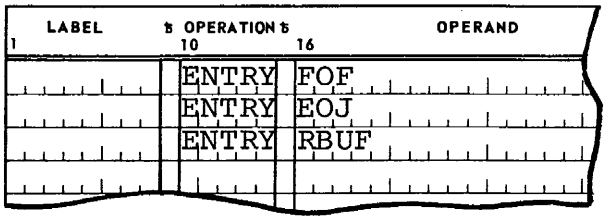
In the example below the subprograms READ and PRNT have been assembled separately. The User program, by means of the EXTRN directive, notifies the Assembler that the symbols, READ and PRNT, will be assigned address values at a later time by the Declarative Macro Instruction subprogram.



45. ENTRY DIRECTIVE (Externally Defined Symbol Declaration)

An ENTRY notifes the Assembler that a symbol (label) in one program will be referenced by another separately assembled program.

In the example below the subprograms FOF and EOJ are referenced in a subprogram that contains Declarative Macro Instructions DTFPR and DTFCR but are defined in the User program. The RBUF area is referenced in the Declarative Macro Instruction subprogram and defined in the User program. To provide the linkage between the subprogram and the User program ENTRY directives with externally referenced symbols (labels) are written into the User program. The labels FOF and EOJ are entry points into the User program from another program. The subprogram is executed after control is transferred to it from a subprogram.

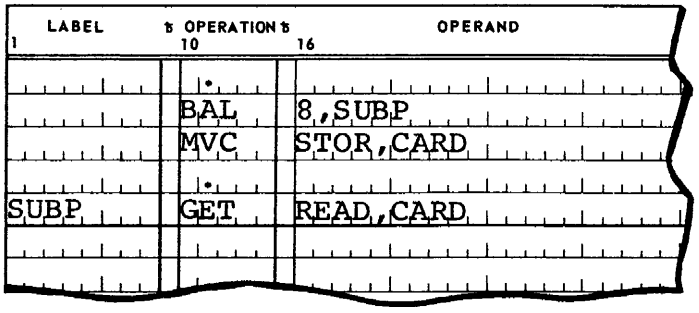


The subprogram would have EXTRN directives corresponding to the ENTRY directives in the main User program. Subprogram READ, for example, contains EXTRNS for EOJ and RBUF. The subprogram PRNT contains an EXTRN for FOF.

For every ENTRY in an element, there is an EXTRN in one or more other elements and, for every EXTRN in an element, there is one ENTRY in another element.

46. BRANCH AND LINK (BAL)

The Branch and Link instruction provides an unconditional branch to the address specified in OP2 while storing the address of the next executable instruction in the register specified by OP1.



In the above program when the BAL instruction is read the program skips to SUBP.

What address is placed in register 8? _____

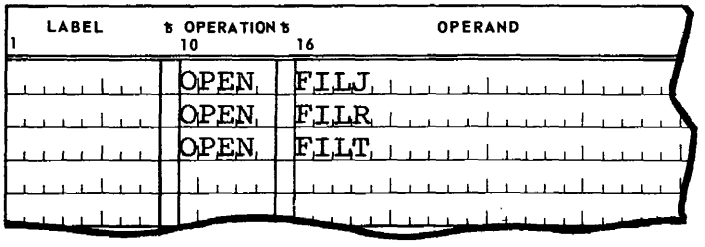
MVC

47. PREVIEW

To read data into the processor or write data out of the processor, it is necessary to have contact with the peripheral equipment (card reader, printer, etc). The OPEN macro instruction provides access between the processor and the peripheral devices. It prepares the peripheral device to be ready to send data upon receipt of a GET macro instruction. The CLOSE macro instruction terminates the communication link.

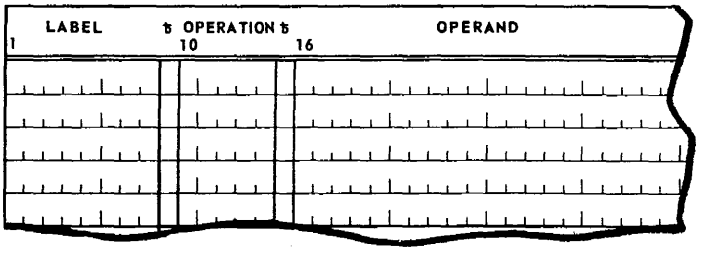
The GET macro instruction causes data to flow across the communication link from the input peripheral device to the processor. The PUT macro instruction causes data to flow from the processor to the output peripheral device.

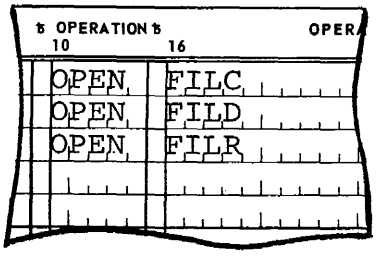
48. EXAMPLE:



In the above instructions, FILJ is opened, then FILR is opened, and finally FILT is opened.

Write three OPEN macro instructions to activate three files named FILC, FILD, and FILR.





Each filename in the operand must be identical to the filename of a DTF statement.

Before data is made available from a file, the file must be _____.

opened (activated)

Each DTF name must agree with the _____ name in the operand of the OPEN macro instruction.

file

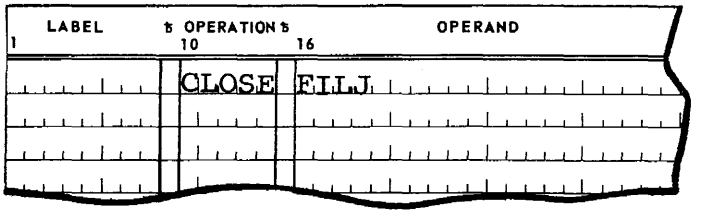
49. The OPEN macro instruction is usually written after the USING instruction, at the beginning of the program.

Check the following statements as true (T) or false (F):

T	F		
☐	☐	Only one OPEN macro can be included in a User's program.	False
☐	☐	A file must be opened before it is accessed.	True
☐	☐	A separate OPEN macro must be written for each DTF.	True
☐	☐	All files must be opened immediately after the USING operation code at the beginning of the program.	False
☐	☐	In a preceding frame it was stated that ENTRY statements are necessary for FOF, EOJ and RBUF because they are referenced by another program.	True
☐	☐	EXTRN statements are necessary for subprograms externally defined.	True

50. The CLOSE macro instruction is similar to the OPEN macro instruction but performs the reverse function. This allows the operator to assign the peripheral device to another program.

Example:



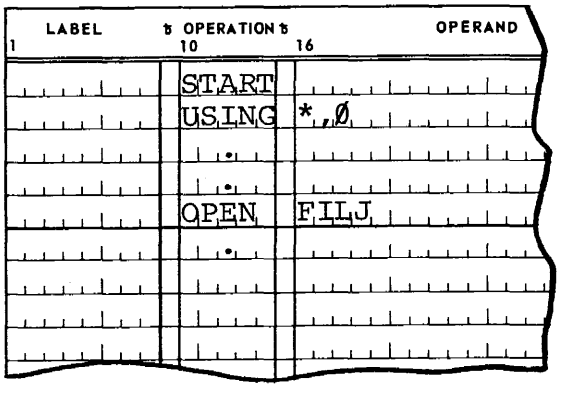
Rules:

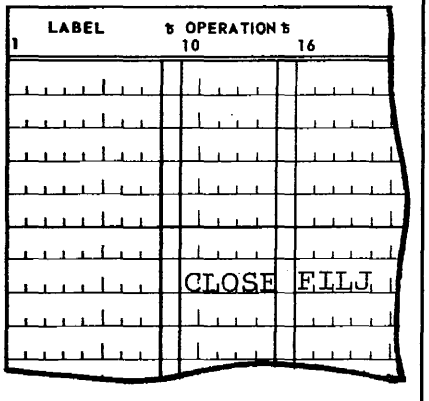
- Any entry in the label field is optional.
- The Operation code is CLOSE.
- The Operand contains the name of the file to be closed. This is the filename of a DTF statement.

Match the following:

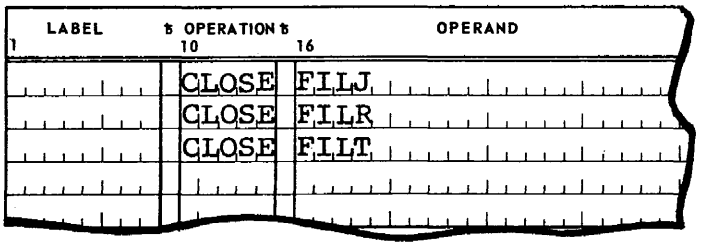
A. CLOSE	_____ Label field	C
B. Filename (FILJ)	_____ Operation field	A
C. Entry is optional	_____ Operand field	B

Write a CLOSE macro instruction to deactivate a filename FILJ.



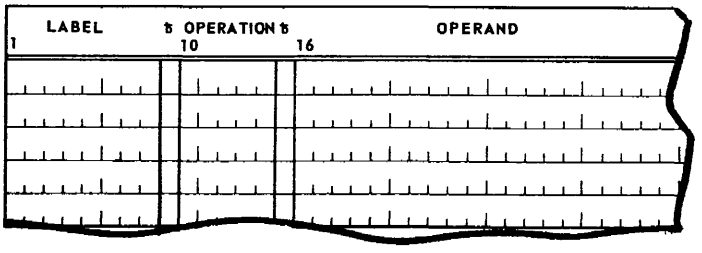


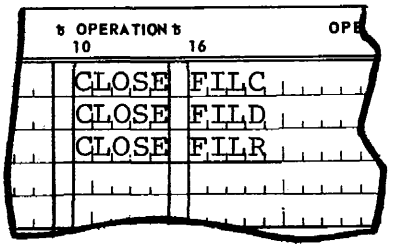
51. The files to be closed are listed in the operand.



In the above instructions FILJ is closed first, then FILR, then FILT.

Write CLOSE macro instructions to deactivate three DTF files named FILC, FILD, and FILR.





Each filename in the operand must agree with the name of a DTF statement and must agree with a filename in the OPEN macro instruction.

A file must be _____ before it can be _____ .

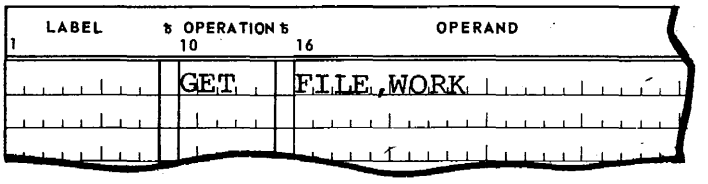
opened (activated)
closed (deactivated)

When a program is finished with a file, the file should be _____ .

closed

52. The GET macro instruction causes the next consecutive record to be read into the processor from the opened peripheral device.

The following is a typical GET macro instruction that places the next logical record into the previously defined input area.



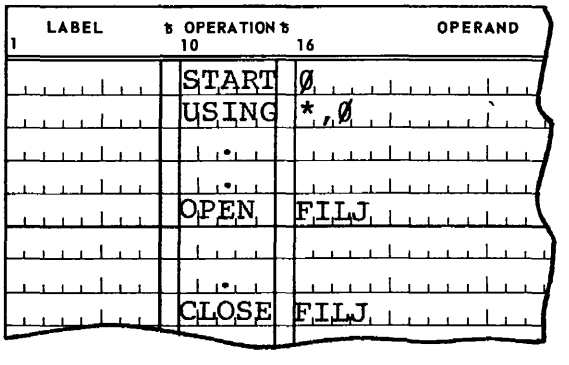
Rules:

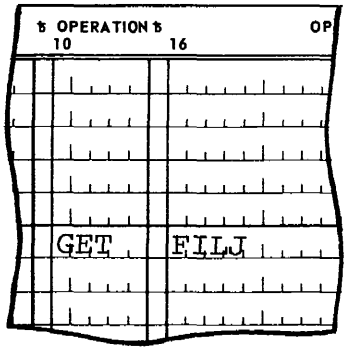
- An entry in the Label field is optional.
- The Operation code is GET.
- Operand 1 specifies the name of a file that identifies the peripheral device from which the record is to be retrieved. This filename is addressed by a label in a DTF statement. Operand 2 specifies the name of the area which is to receive data.

Match the following:

A. GET	_____ Label field	B
B. Optional entry	_____ Operation field	A
C. Filename (FILE)	_____ Operand field	C

Write a GET macro instruction to read the next record from FILJ into the processor.

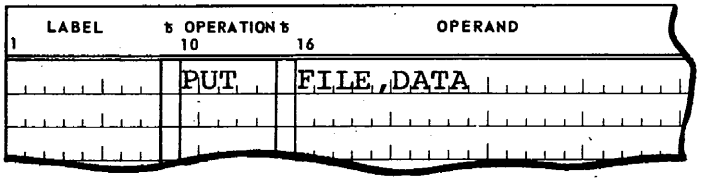




53. The PUT macro instruction causes the next logical record to be written from the processor to the opened peripheral device.

The PUT macro instruction directs a peripheral device to write, punch, or display logical records that are in the output area of memory.

The following PUT macro instruction sends a completed record to the output device identified as the filename of the DTF statement.



Rules:

- An entry in the Label field is optional.
- The operation code is PUT.
- Operand 1 specifies the filename that identifies the peripheral device that is to receive the record. The filename must be identical to the label of the DTF statement. Operand 2 specifies the symbolic address of the data that is to be transferred to the peripheral device.

The PUT macro instruction above processes data records serially.

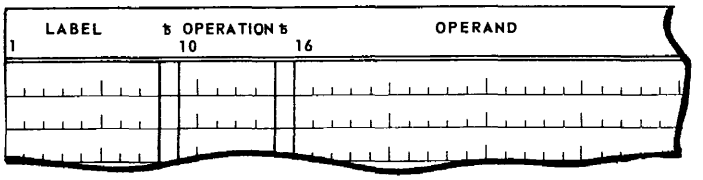
The filename in operand 1 must agree with the Label in the _____ statement.

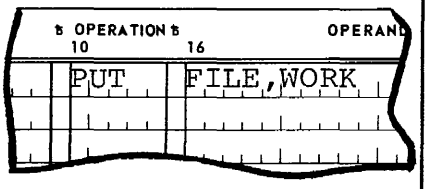
DTF

The location where the record is to be written is called _____.

FILE

54. Write a PUT macro instruction that sends the next record from a work area (WORK) to a printer (FILE).





55. If we have moved all data into the output area (labelled OUT) to be printed, what output device would be specified in Operand 1? _____

Printer

This device would be specified in Operand 1 as PRNT.

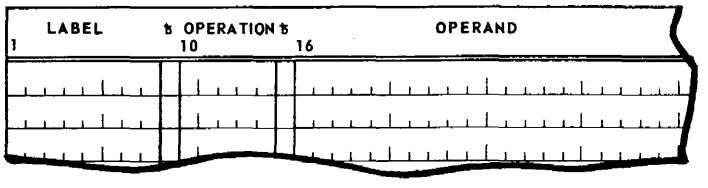
Suppose we wanted the data in the output area to be punched instead of printed, what device would be specified in Operand 1? _____

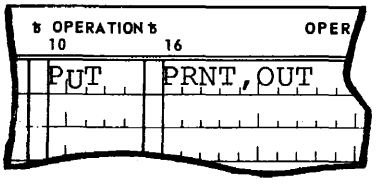
Punch

This device would be specified in Operand 1 as PNCH.

Operand 2 would contain the label of the output area.

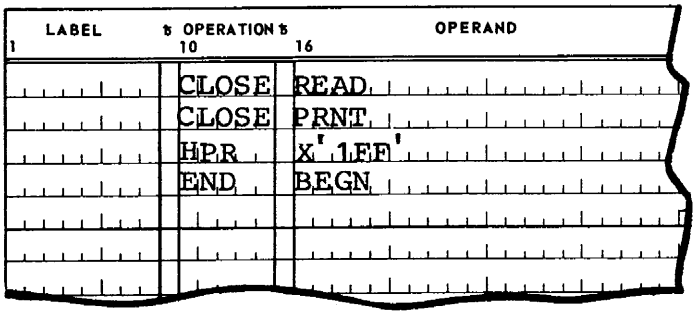
If we were moving data from OUT to PRNT, how would the instruction to cause printing be coded? Write the instruction below.





56. END

The END statement directs the Assembler to terminate the program being assembled.



The operand in the END statement is the symbolic address of the first executable instruction of the Object program.

The last instruction in an Assembly program is the:

- ☐ CLOSE macro instruction
- ☐ END statement
- ☐ TERM instruction

END statement

57. PREVIEW

Most of the data processing steps illustrated in a process flowchart are performed within the main memory of the computer when the Object Program is executed.

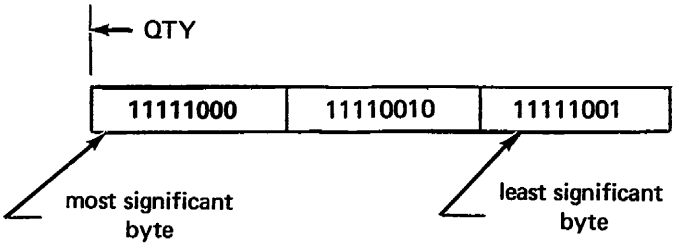
In this section, you will learn more about applications and rules governing instructions that perform calculations within the main memory. Specifically, the Add, Subtract, and Multiply instructions.

58. FOOTNOTE FRAME

In the Introduction to 9200/9300 you learned that numeric data must be stored in packed format before arithmetic calculations can be performed. At that time, we introduced EBCDIC code and showed illustrations of data in packed and unpacked formats. The concept of packed and unpacked data is reviewed in this section.

59. Assume that we are writing a stock-control program that reads in punched cards. The cards contain the number of items added to or removed from stock. One field in this transaction card is called QTY (quantity).

After we read a card with the value 829 in the QTY field, memory would appear as:



The four leftmost bits of each byte contain:

- ☐ all zeros.
- ☐ all ones.
- ☐ a combination of ones and zeros.

all ones

When the four leftmost bits of the least significant byte are all ones, the field is considered to be positive. In the illustration above, the value 829 is:

- ☐ positive (+).
- ☐ negative (-).

positive (+)

The sign of the field is indicated by the four leftmost bits of the:

- ☐ most significant byte.
- ☐ least significant byte.

least significant byte

60. When numeric data is read in from punched cards, it appears in memory in unpacked format. Character representation of decimal data is unpacked. This means that the leftmost four bits of the least significant byte in the field always designate the sign. The leftmost four bits of the other bytes in the field are called zone bits. The illustration below shows a three-byte, unpacked decimal field.

Zone	Numeric	Zone	Numeric	Sign	Numeric

Using the positive value 829, memory contents could be illustrated as:

Z	8	Z	2	+	9

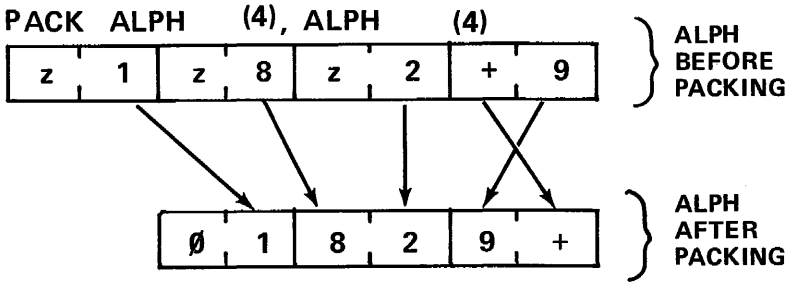
The four leftmost bits of the bytes containing the digits 8 and 2 are called _____ bits.

zone

The four leftmost bits of the byte containing the 9 designate the _____ of the field.

sign

61. The zone bits have no decimal value. Therefore we can replace them with numeric information by packing two decimal digits into an eight-bit byte. A single Pack instruction can convert a field from unpacked format to packed format. The instruction preserves the sign of the unpacked field by reversing the positions of the sign and the numeric portion of the least significant byte in the field as shown below:



To calculate the minimum length of the receiving field, halve the length of the unpacked field and add one.

After packing, the sign of the field is located in the:

- ☐ rightmost position of the field.
- ☐ leftmost position of the last byte.

rightmost position of the field

An unpacked numeric field that is three bytes in length can be packed into_____ bytes (fractions are not counted).

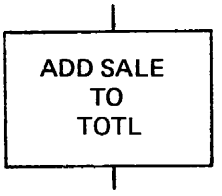
two

An unpacked numeric field that has a length of nine bytes can be packed into a field having a length of:

- ☐ eight bytes.
- ☐ five bytes.
- ☐ four bytes.

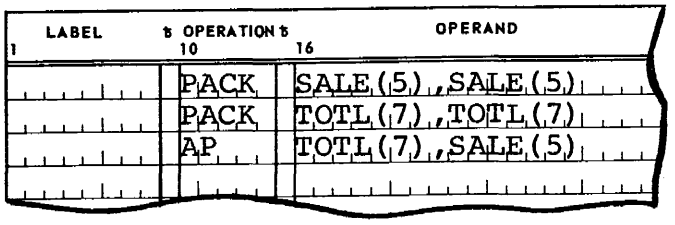
five bytes

62.



This block of a flowchart specifies that the data in the area of memory named SALE is to be added to the data in the area named TOTL. If the data is read in from cards, it is in unpacked format. The data must be packed before it is used in calculations. This function is not illustrated in the flowchart.

The following instructions cause the data to be packed and then added.



How many instructions must be written to pack the two data items? _____

two

How many bytes of memory does the field named TOTL occupy before it is packed? _____

seven

How many bytes of memory does the field named TOTL occupy after it is packed? _____

seven

63. In the first card read into memory, the field SALE contains the value 45613 and the field TOTL contains 0071460. Show each field in memory before and after packing. (Refer to panel 1 in the back of this manual.)

SALE Before

F4	F5	F6	F1	F3

TOTL Before

FØ	FØ	F7	F1	F4	F6	FØ

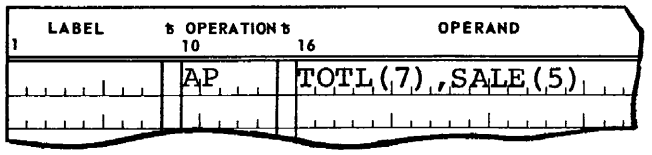
SALE After

ØØ	ØØ	45	61	3+

TOTL After

ØØ	ØØ	ØØ	ØØ	71	46	Ø+

64.



This Add Packed Decimal (AP) instruction adds the packed data in SALE to the packed data in TOTL. The result is placed in TOTL (destroying the previous contents of TOTL). The data in SALE is not destroyed by the operation.

After an AP instruction is executed at Object Program execution, the sum is located in the memory area indicated by:

- ☐ the first operand.
- ☐ the second operand.

the first operand

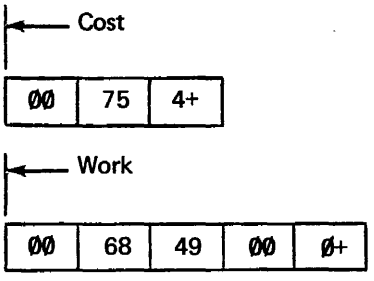
How does the control unit of the computer know the number of bytes of data in each field to be added?

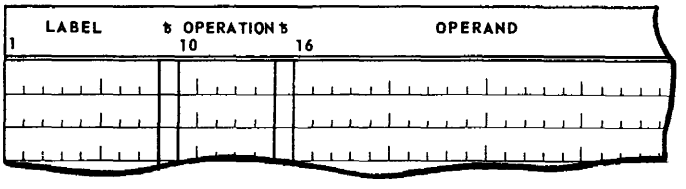
- ☐ All data fields used in arithmetic instructions are exactly five bytes long.
- ☐ The programmer has included two length factors in the instruction.

The programmer has included two length factors

65. All of the decimal arithmetic instructions permit the operands to be of different lengths. Each operand can be up to 16 bytes in length. The first operand, however, must be the longer because this field will contain the result after execution of the instruction. If the first operand is shorter than the second operand, significant digits in the high-order bytes of the second operand will not enter into the result.

Write an instruction that will add the following numeric data items and place the result in the larger area.

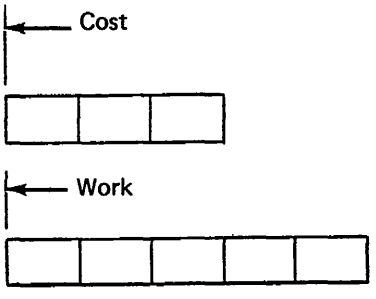




OPERATION 10 | OPERAND 16

AP | WORK(5),COST(3)

Show the two areas of memory *after* the instruction is executed.



ØØ | 75 | 4+

ØØ | 68 | 49 | 75 | 4+

66. The maximum length of the operands in the Pack (PACK), Add Decimal (AP), and Subtract Decimal (SP) instructions is:

- ☐ 7 bytes.
- ☐ 16 bytes. — 16 bytes
- ☐ 40 bytes.
- ☐ 256 bytes.

When a decimal arithmetic instruction is executed, the result is found in the memory location identified by the:

- ☐ first operand. — first operand
- ☐ second operand.

67. Examine the following code and answer the questions.

```
LABEL      OPERATION      OPERAND
1          10             16
           PACK           SALE(6),SALE(6)
           PACK           ABLE(4),BAKE(7)
           AP             SALE(6),ABLE(4)
```

What is the length of the field named SALE after the packing operation?_____ bytes. — six

Before packing, the data in BAKE was seven bytes long. What is the length of ABLE after packing?_____ bytes. — four

In the Add instruction, the four bytes of data at ABLE are added to the numeric data in the field named _____. — SALE

The sum produced by this addition will be in the field named _____. — SALE

68. The PACK instruction permits the programmer to specify a length for both operands. The packing operation can be performed in place (as in our earlier examples) or into another memory area. When the packing is performed in place, high-order zeros are generated to fill out the field. When the packing operation is to occur in a separate area of memory, this second area does not have to be cleared or initialized in any way as high-order zeros are generated.

Write the instructions that will pack the contents of AMT into an area named WORK, and then add WORK to TOTL. Examine the data before writing the instructions.

AMT before packing:

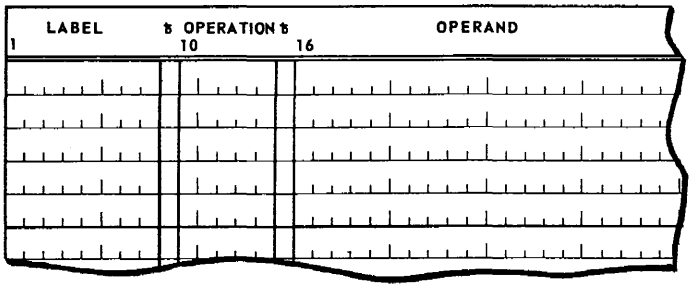
F 7	F 5	F Ø

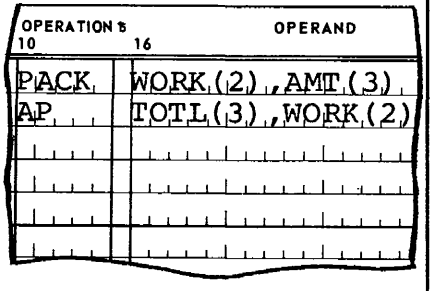
WORK after packing:

7 5	Ø +

TOTL before addition:

Ø 9	1 2	9 +





69.

PRCE		
96	87	1+

COST		
Ø7	67	2+

Adding fields of equal length can result in an overflow condition.

Suppose AP PRCE (3), COST (3) is executed when the data in the fields are as shown above. What is the *true* total of the two values?_____

104,543

What is the name of the result field?_____

PRCE

Can the result fit into the result field?

☐ Yes

☐ No

No

The computer would try to get the result, 104543+ into PRCE in the same way you perform arithmetic calculations, by starting at the right and working to the left.

What would appear in PRCE as a result of the addition?

Ø4	54	3+

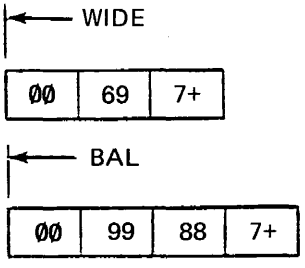
70. **FOOTNOTE FRAME**

The situation described in the preceding frame is called overflow because the result overflows the field in which it is to be stored. It can occur even when the result field is longer than the other field; for example, adding 1 to the five-character field 99,99,9+. It is the programmer's responsibility to provide adequate storage for the expected result. Remember, the sign is always included.

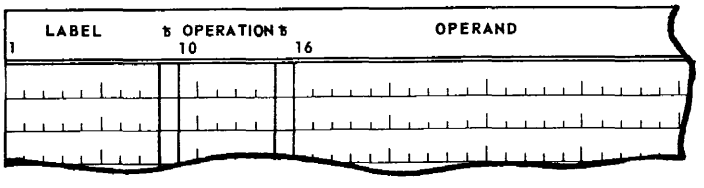
The decimal Add and Subtract instructions set an internal indicator called the *condition code,* following the execution of each instruction. We may test this condition code and branch to another place in the program, or not branch, depending on the setting of the internal indicator. The indicator may be set to one of four conditions, which are designated by the values 0, 1, 2, and 3. Condition code 3 indicates that overflow has occurred.

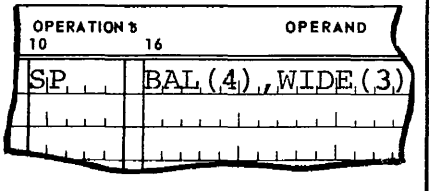
You will learn how to write statements to test for overflow later in the course.

71. The Subtract Packed Decimal (SP) instruction has the same format as the Add Packed Decimal (AP) instruction. The first operand contains the result (difference) following execution.



Write an instruction that will subtract the quantity in WIDE from the quantity in BAL. The result is to appear in BAL.





72. Write the instructions needed to pack and then subtract the two data items from a field named TOTL.

TOTL

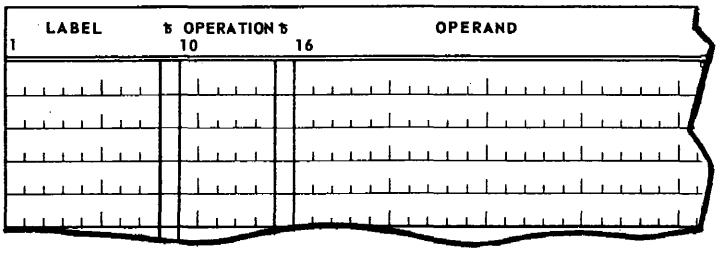
ØØ	96	25	47	6+

WORK

F2	F6	F7	F3

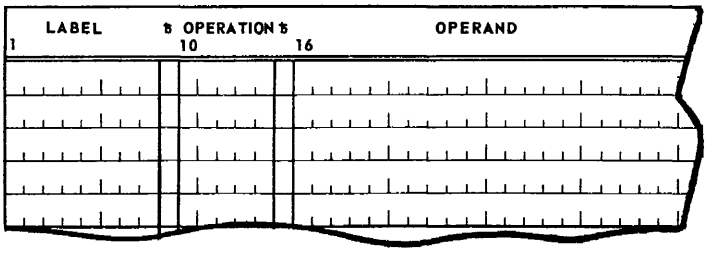
COST

F1	F3	F4



OPERATION	OPERAND
PACK	WORK(4),WORK(4)
PACK	COST(3),COST(3)
SP	TOTL(5),WORK(4)
SP	TOTL(5),COST(3)

Now write code that will produce the same result by packing and then adding WORK and COST and subtracting the sum from TOTL.



OPERATION	OPERAND
PACK	WORK(4),WORK(4)
PACK	COST(3),COST(3)
AP	WORK(4),COST(3)
SP	TOTL(5),WORK(4)

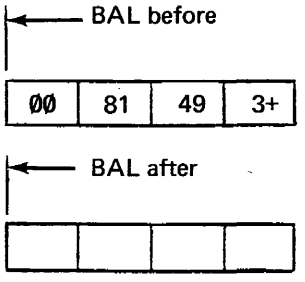
Show the contents of the area named TOTL after the instructions are executed.

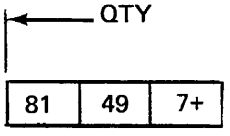
ØØ	96	22	66	9+

73. The packed numeric data specified by the second operand is algebraically added to (or subtracted from) the data specified by the first operand.

All of the examples that have been used thus far in the explanation of the Add Decimal and Subtract Decimal instructions have had a positive result. But positive results are not always the case. Whenever a larger positive value is subtracted from a smaller one the result is negative. In some cases when the operands have different signs the results are negative. The conventional rules of algebra apply.

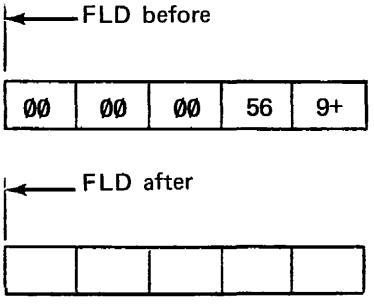
Show the result obtained by the instruction SP BAL(4), QTY(3).

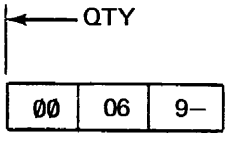




ØØ | ØØ | ØØ | 4–

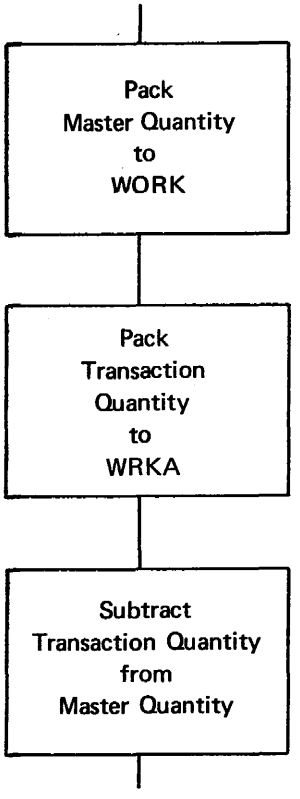
Because of the algebraic rules, the addition of a negative number decreases the first operand. Show the results of AP FLD(5), QTY(3).

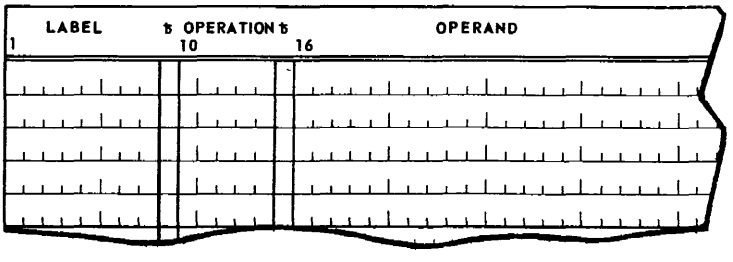




ØØ | ØØ | ØØ | 5Ø | Ø+

74. Assume the master quantity (MQTY) and transaction quantity (TQTY) fields contain a maximum of five decimal characters. The work areas (WORK and WRKA) have been defined as three-byte fields. Write the code to perform the functions contained in the following flowchart.



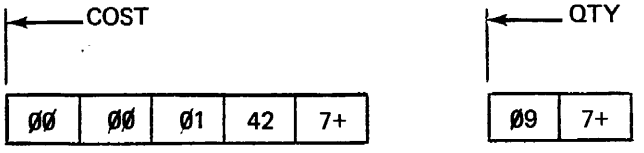


OPERATION	OPERAND
PACK	WORK(3),MQTY(5)
PACK	WRKA(3),TQTY(5)
SP	WORK(3),WRKA(3)

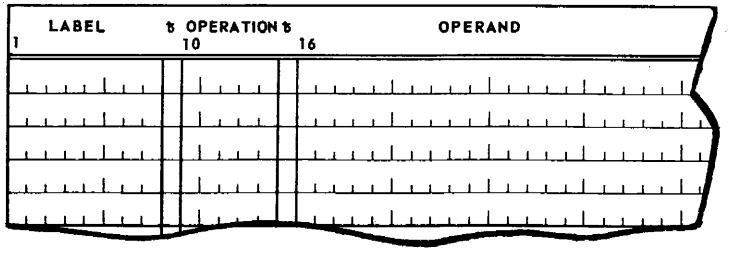
75. The Multiply Packed Decimal instruction has the same format as the Add Decimal and Subtract Decimal instructions; the result field is specified by the first operand. The operation code is MP. The data in the location specified by the first operand is multiplied by the data in the location specified by the second operand. The result field (the product) must be large enough to contain all of the significant digits resulting from the multiplication. The following rules ensure that the first operand is large enough to receive the product.

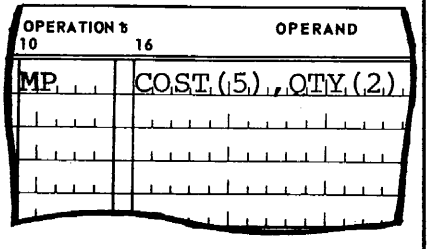
*The second operand (multiplier) must be shorter than the first operand and must not exceed eight bytes in length.

*The first operand (multiplicand) must have high-order zero bytes equal to the number of bytes in the multiplier field.



Write a Multiply instruction using the data above.

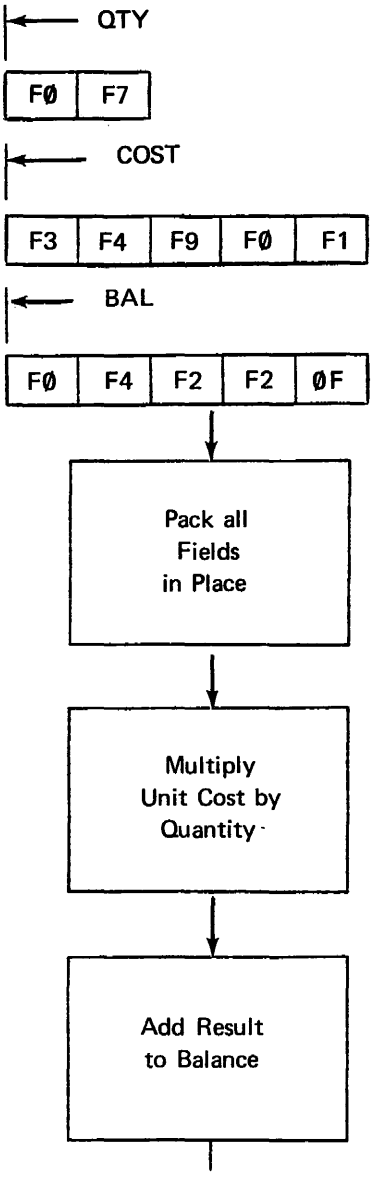




76. Check the following MP statements as true (T) or false (F):

T	F		
☐	☐	High-order zero bytes in the first operand must be equal to the length of the second operand.	True
☐	☐	The first operand must not exceed eight bytes.	False
☐	☐	The second operand is the multiplier.	True
☐	☐	The product is located in the first operand.	True
☐	☐	The multiplier must contain leading zeros.	False
☐	☐	The Multiply instruction has the same format as the Add and Subtract instructions.	True
☐	☐	The result field is specified by the first operand.	True

77. Write the instructions that perform the functions indicated by the flowchart. Assume the following data is read in from punched cards.

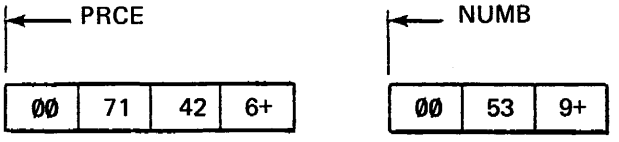


LABEL	OPERATION	OPERAND

OPERATION	OPERAND
PACK	QTY(2),QTY(2)
PACK	COST(5),COST(5)
PACK	BAL(5),BAL(5)
MP	COST(5),QTY(2)
AP	BAL(5),COST(5)

78. A rule for multiplication states that the first operand must have high-order zero bytes equal to the number of bytes in the second operand. Packing the unit cost field (COST) in the previous frame produced the two bytes of leading zeros required. Would it be possible to multiply the following two fields?_____

No



Check the reasons for your answer.

- ☐ Neither field is large enough to store the result.

  Neither field is large enough

- ☐ At least three bytes of leading zeros are required in the first operand.

  Three bytes of leading zeros are required

- ☐ The result could be stored in PRCE.

- ☐ Two bytes of leading zeros are present.

79. Another decimal arithmetic instruction performs the dual function of zero-filling a work area and moving a packed field into the work area. The Zero and Add Packed instruction performs this function. The operation code is ZAP and the format is the same as the AP, SP, and MP instructions.

We were unable to multiply the PRCE and NUMB fields in the previous frame. However, we can allocate a seven-byte work area called WORK with the statement WORK DS CL7 and write an instruction that will ZAP the fields named WORK and PRCE. The result of this ZAP instruction is a field that is large enough to receive the result of the multiplication. Write this instruction on the line above the MP instruction.

```
LABEL      OPERATION      OPERAND
1          10             16
           MP             WORK(7),NUMB(3)
```

```
OPERATION      OPERAND
10             16
ZAP            WORK(7),PRCE(4)
```

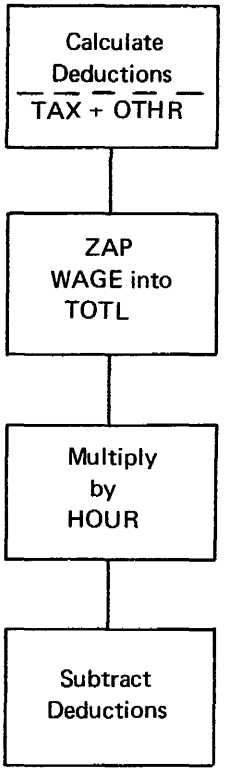
After execution of the Multiply instruction, the result is located in the field named_____.

WORK

80. Assume that the DS statements defining the names used in the flowchart below are:

LABEL	OPERATION	OPERAND
WAGE	DS	CL3
TAX	DS	CL4
HOUR	DS	CL2
TOTL	DS	CL5
OTHR	DS	CL2

All data is in packed format. WAGE may contain a maximum of five decimal numbers and a sign. Complete the following instructions indicated by the flow chart.

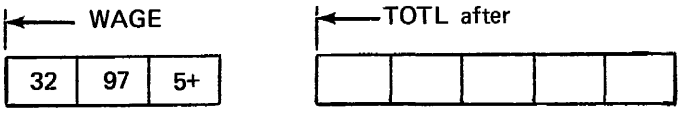


LABEL	OPERATION	OPERAND
	AP	
	ZAP	TOTL(5),
	MP	
	SP	

OPERAND
TAX(4),OTHR(2)
WAGE(3)
TOTL(5),HOUR(2)
TOTL(5),TAX(4)

81. After execution of the instruction below, show the memory contents of the field named TOTL if the value in WAGE is as shown.

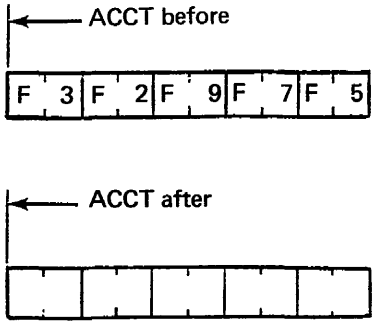
ZAP TOTL(5), WAGE (3)



ØØ	ØØ	32	97	5+

Show the memory contents of the field named ACCT after execution of the PACK instruction.

PACK ACCT (5), ACCT(5)



ØØ	ØØ	32	97	5+

82. FOOTNOTE FRAME

To this point in the text you have studied the PACK, AP, SP, MP, and ZAP instructions. A thorough understanding of these instructions enables you to perform all of the calculations required in the course.

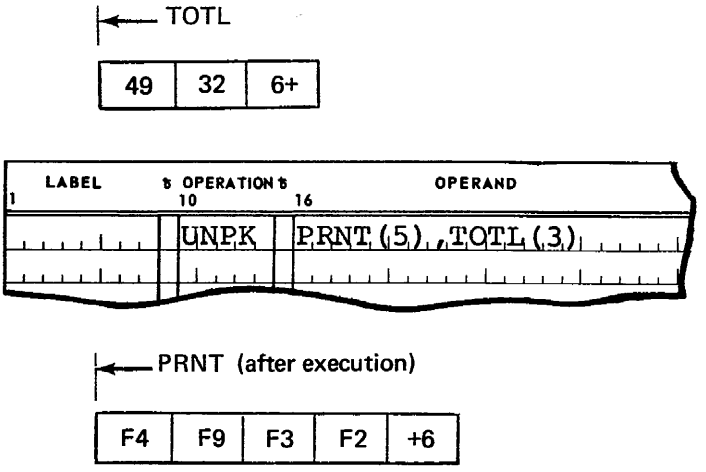
83. Many programs are designed to produce a written report as an end product. In a stock control problem the report may contain a listing of the items that are currently in stock. The report may include the quantity on hand, the unit price, and many other items. A banking report may contain information about mortgage, checking or savings transactions, etc. Each of these programs will require calculations on packed fields of data. Before the results of the calculations can be displayed on a printed report or punched into cards, the data must be converted to unpacked format. Unpack is a decimal arithmetic instruction with the operation code UNPK. The instruction format is the same as the PACK, AP, SP, MP, and ZAP instructions.

Review the following Unpack instruction and check the succeeding statements as true or false:

```
UNPK    WORK(5), SALE(3)
```

T	F		
☐	☐	The data in the second operand is unpacked into the area identified by the first operand.	True
☐	☐	After execution, the data in WORK will be in packed format.	False
☐	☐	Data must be unpacked before it is printed or punched into cards.	True
☐	☐	The Unpack instruction has the same format as the Pack instruction.	True
☐	☐	UNPK is a decimal arithmetic instruction.	True
☐	☐	The field into which we are unpacking the data must be larger than the packed field.	True

84.



As in the packing operation, the half-bytes of the rightmost byte are reversed. The unpacked field must be approximately:

- ☐ twice as large as the packed field.
- ☐ half the size of the packed field.

twice as large as the packed field

The contents of the rightmost byte are moved into the rightmost byte of the receiving field:

- ☐ without being changed.
- ☐ with the sign and numeric reversed.
- ☐ with the sign stripped from the field.

with the sign and numeric reversed

The Unpack instruction will generate zone bits of a hexadecimal F in:

- ☐ all bytes of the receiving field.
- ☐ all but the rightmost byte of the receiving field.

all but the rightmost byte of the receiving field

85. To calculate the length of the receiving field, double the length of the packed field and subtract one.

Example:

Unpack AMT (4 bytes) into WRK

LABEL	OPERATION	OPERAND
	UNPK	WRK(7),AMT(4)

Write the instructions that will unpack the following:

Unpack AMT 1 (2 bytes) into WRK1

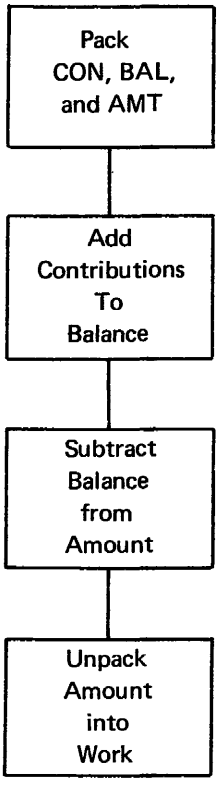
Unpack AMT 2 (3 bytes) into WRK2

Unpack AMT 3 (6 bytes) into WRK3

LABEL	OPERATION	OPERAND

OPERATION	OPERAND
UNPK	WRK1(3),AMT1(2)
UNPK	WRK2(5),AMT2(3)
UNPK	WRK3(11),AMT3(6)

86. Complete the program that will perform the functions indicated in the flowchart.



The data is read into memory from cards as:

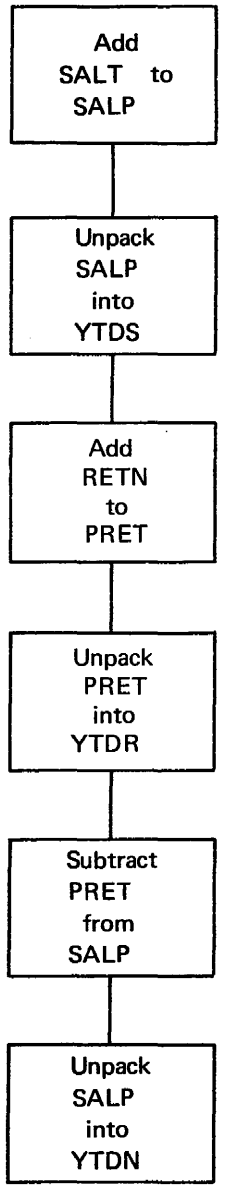
CON 3 bytes (unpacked)
BAL 5 bytes (unpacked)
AMT 5 bytes (unpacked)

The field named WORK is defined as nine bytes long.

LABEL	OPERATION	OPERAND
	PACK	CON(3),,
	PACK	BAL( ),,BAL( )
	PACK	AMT(5),,AMT(5)
	AP	
	SP	AMT(5),,
		WORK(9),,

OPERATION	OPERAND
PACK	CON(3),,CON(3)
PACK	BAL(5),,BAL(5)
PACK	AMT(5),,AMT(5)
AP	BAL(5),,CON(3)
SP	AMT(5),,BAL(5)
UNPK	WORK(9),,AMT(5)

87. Write the code to calculate the year-to-date (YTD) sales, YTD returns, and YTD net sales. The record formats and a segment of the flowchart are shown below:



INPUT TAPE FILE

1-5	Account Number	
6-9	Sales this month	SALT (packed)
10-14	Prior sales	SALP (packed)
15-18	Returned this month	RETN (packed)
19-22	Prior returns	PRET (packed)

OUTPUT TAPE FILE

1-5	Account Number	
6-14	YTD sales	YTDS (unpacked)
15-23	YTD returns	YTDR (unpacked)
24-32	YTD Net Sales	YTDN (unpacked)

LABEL	OPERATION	OPERAND
1	10 16	

OPERATION	OPERAND
10	16
AP	SALP(5),SALT(4)
UNPK	YTDS(9),SALP(5)
AP	PRET(4),RETN(4)
UNPK	YTDR(9),PRET(4)
SP	SALP(5),PRET(4)
UNPK	YTDN(9),SALP(5)

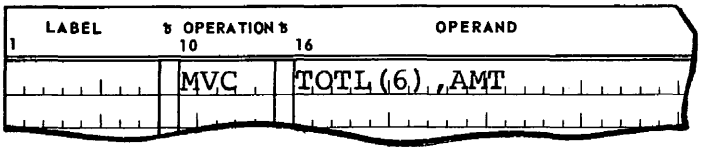
88. **PREVIEW FRAME**

An instruction that moves data from one memory area to another memory area is a familiar component of any computer system. The 9200/9300 System uses a set of Move instructions of which the basic member is the Move Character (MVC) instruction.

The following frames supply you with the basic information required to effectively use the MVC instruction.

The Move Immediate (MVI) instruction, which may be considered a subset of the MVC instruction, is also introduced.

89. When a Move Character instruction is executed, the contents of the second operand are copied into the first operand. The MVC instruction below moves (or copies) six bytes of data from the location named AMT to the location named TOTL.



The data being moved can be in packed or unpacked format and can include letters of the alphabet, punctuation, or mathematical symbols.

Which is the receiving field in the MVC instruction?

- ☐ First-operand field
- ☐ Second-operand field

First-operand field

When this instruction is executed, the data in the first byte of AMT is moved into:

- ☐ the last byte of TOTL.
- ☐ the first byte of TOTL.

the first byte of TOTL

90. Arithmetic instructions require two length factors in their operands. Both length values are stored in a single byte of memory (each using four bits). Therefore, the data areas addressed by a single arithmetic instruction cannot exceed 16 bytes. In the MVC instruction (and others that require only one length), 256 bytes can be addressed and moved with one instruction.

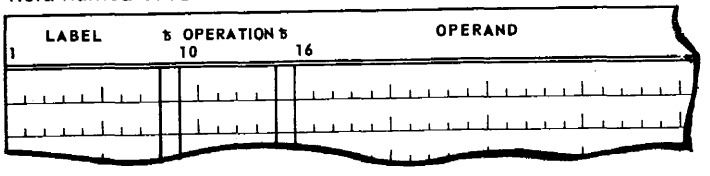
MVC PRNT (132), WORK

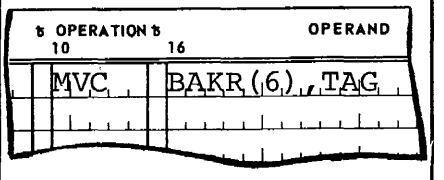
The above instruction moves 132 bytes from the location beginning with _____ to location beginning with _____ .

WORK
PRNT

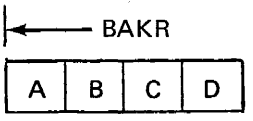
Like other storage to storage instructions, the first operand is the receiving field.

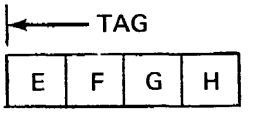
Write an instruction that will move six bytes of data from the field named TAG to a field named BAKR.



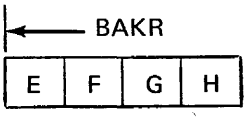


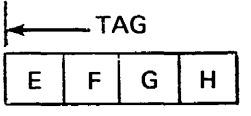
After an input card is read, areas BAKR and TAG contain the following data:





After an MVC instruction is executed areas BAKR and TAG contain the following data:





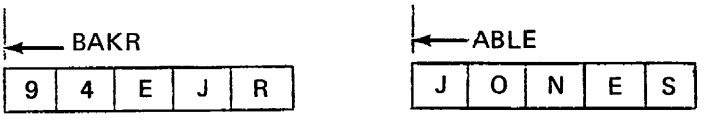
Data was destroyed in the:

- ☐ sending field.
- ☐ receiving field.

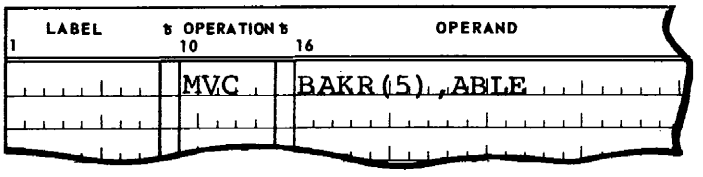
receiving field

91. The move operation in the 9200/9300 does not destroy the sending data. The data is copied into another field.

After an input card is read, the fields BAKR and ABLE contain the following data.



Show the contents of the fields *after* the following MVC instruction is executed.



Both areas will contain JONES

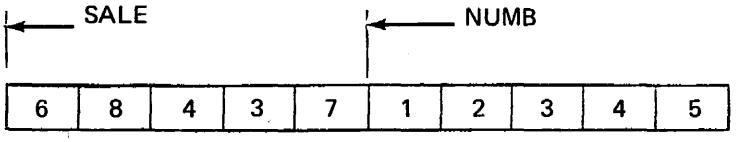
Which field remains unchanged as a result of the MVC instruction? _____

The sending field (ABLE)

92.

```
LABEL    OPERATION    OPERAND
1        10           16
SALE     DS           CL5
NUMB     DS           CL5
WORK     DS           CL10
```

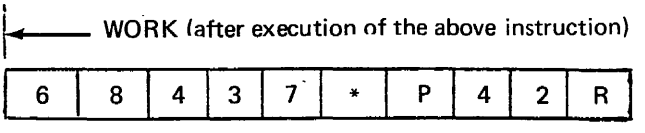
After an input card is read, the fields SALE and NUMB contain the following data:



Write an instruction that will move the data from SALE into the first five bytes of WORK.

```
LABEL    OPERATION    OPERAND
1        10           16
```

```
OPERATION    OPERAND
10           16
MVC          WORK(5),SALE
```



After execution, the first byte of WORK contains a 6. WORK+1 contains an 8, WORK+3,contains a 3, etc. What character is in WORK+5?_____

*(asterisk)

```
LABEL    OPERATION    OPERAND
1        10           16
         MVC          WORK+5(5),NUMB
```

The above instruction moves five bytes of data from NUMB into the last five bytes of WORK. This addressing method is called relative addressing. What is in WORK after execution of the above move operation?

6	8	4	3	7	1	2	3	4	5

93.

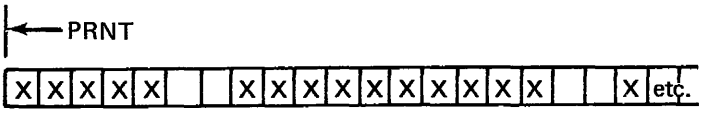
LABEL	OPERATION	OPERAND
EMPN	DS	CL5
NAME	DS	CL1Ø
ADDR	DS	CL1Ø
JOBT	DS	CL3Ø

After an input card is read, the data in the fields listed above must be moved into an output area for printing. Assume the printer area has been defined as:

PRNT DS CL132

For the present, you should also assume that the output area has been cleared of all data. Each byte contains the bit configuration for a blank space.

Write Move instructions that will load PRNT with the data from EMPN, NAME, ADDR, and JOBT. Leave two bytes of blanks between each item moved to form the following:



LABEL	OPERATION	OPERAND
	MVC	
	MVC	
	MVC	
	MVC	

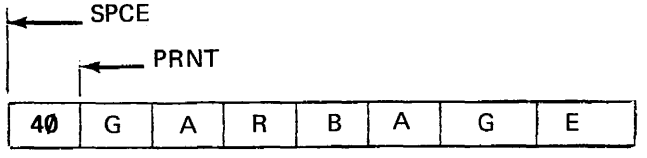
OPERATION	OPERAND
MVC	PRNT(5),EMPN
MVC	PRNT+7(1Ø),NAME
MVC	PRNT+19(1Ø),ADDR
MVC	PRNT+31(3Ø),JOBT

94. Moving a small amount of data into a large area does not clear or change the unused portion of the receiving field in any way. Therefore, the programmer must clear the printer output area before loading each line of data to be printed.

By placing a 1-byte space immediately ahead of the 132-byte printer output area, it is possible to propagate the space character through the print line and thereby clear the print line to spaces.

```
*  LABEL      OPERATION      OPERAND
1            10             16
   SPCE       DC             C'△'
   PRNT       DS             CL132
```

The result of the above coding places a space immediately ahead of the print line output storage area.



Your task is to propagate the space through the entire 132 bytes of PRNT with a single Move instruction.

```
   LABEL      OPERATION      OPERAND
1            10             16
```

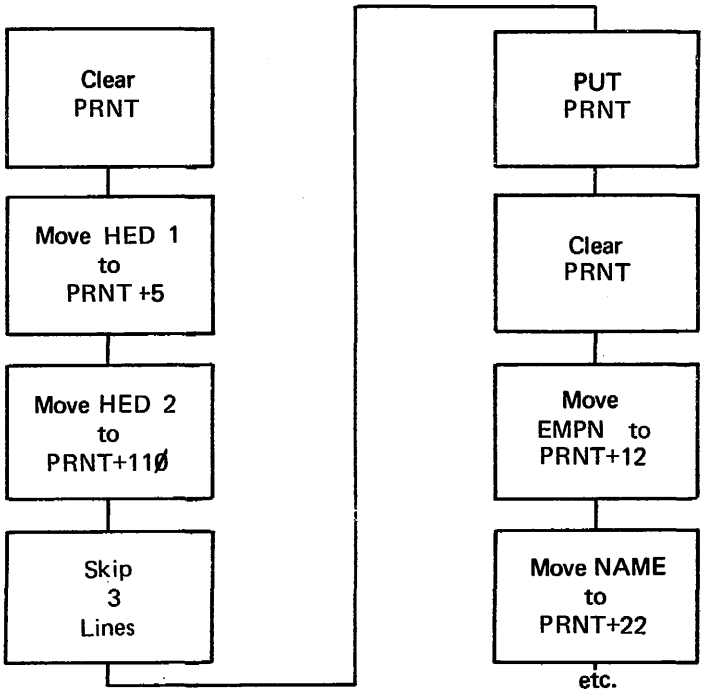
```
   OPERATION      OPERAND
   10             16
   MVC            PRNT(132),SPCE
```

When the above instruction is executed at object time, the first character from SPCE (hex 40) replaces the first character of PRNT. (This places a hex 40 in SPCE+1.) On the next memory cycle, the character from SPCE+1 (now hex 40) replaces the *second* character of PRNT (SPCE+2). The 40 in the second byte is then copied to the third byte, and so on until the space character is propagated through the entire 132-byte PRNT field. The length factor (132) in this example controls the number of bytes to be moved.

* The delta (△) symbol is used to signify a space.

95.

```
LABEL      OPERATION      OPERAND
1          10             16
BLNK       DC             X'40'
PRNT       DS             CL132
EMPN       DS             CL5
NAME       DS             CL20
ADDR       DS             CL20
JOBT       DS             CL30
HED1       DC             C'FINAL REPORT'
HED2       DC             C'JULY 1969'
```



Use the data shown above to code the processing operations indicated by the flowchart.

```
LABEL      OPERATION      OPERAND
1          10             16
           CNTRL          PRNT,SK,3
           PUT            PRNT
```

```
OPERATION      OPERAND
10             16
MVC            PRNT(132),BLNK
MVC            PRNT+5(12),HED1
MVC            PRNT+110(9),HED2

MVC            PRNT(132),BLNK
MVC            PRNT+12(5),EMPN
MVC            PRNT+22(20),NAME
```

96. **FOOTNOTE FRAME**

We have used the relative addressing technique with a Move instruction to load and position the data in a printer output area. This same addressing technique can be used in other instructions, for example:

```
AP   TOTL(4),SUM+3(1)

SP   RSLT+6(3),AMT(2)

ZAP  TAX(9),WORK+4(5)
```

Although a valid technique, relative addressing should be avoided whenever possible. Extensive use will frequently introduce clerical errors into the program. It is better programming practice to assign a symbolic name to each memory area to be addressed. The ORG statement provides you with this capability.

97. The Move Immediate instruction (MVI) permits the programmer to place one character anywhere in memory.

Example:

```
MVI   PRNT,X'4Ø'
```

In the above example the second operand is the hexadecimal configuration of a space (hexadecimal 4Ø). When the instruction is executed at object time, the space character is moved into the:

- ☐ last byte position of PRNT.
- ☐ first byte position of PRNT.

first byte position of PRNT

The second operand is called a self-defining value and must not exceed one byte.

A length is not specified in the MVI instruction because it always operates on:

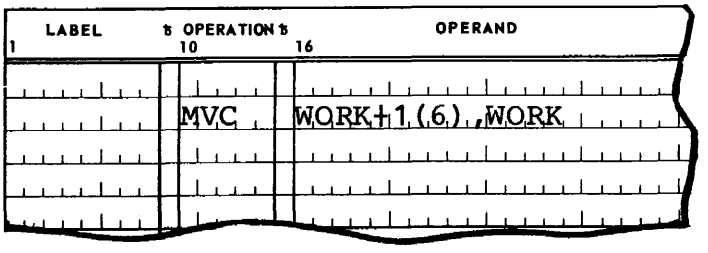
- ☐ three bytes.
- ☐ one byte.

one byte

- ☐ two bytes.

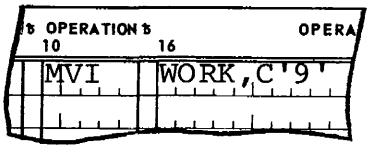
98. The second operand in the MVI instruction can be written as a single character in quotes preceded by the letter C, or as two hexadecimal digits in quotes preceded by the letter X. C and X denote the character and hexadecimal constants studied earlier in the text.

Assume that it is necessary to fill an area named WORK with decimal 9's. Work has been defined as WORK DS CL7. Complete the code below by supplying the MVI instruction.

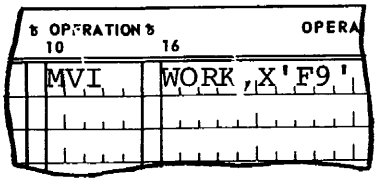


How many characters are moved into WORK by the MVI instruction?

- ☐ Six
- ☐ One



OR

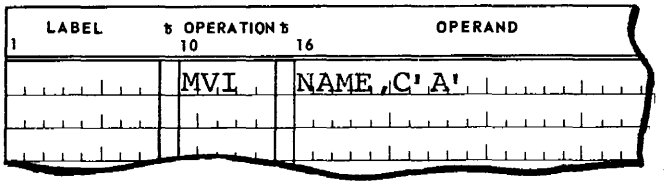


One

99. NAME

X	X	X	X	X	X

Assume that NAME has been defined as a six-byte area. The X's in NAME represent any data that may have been left in the area from a previous program.



Fill in the diagram below to show the contents of NAME after the MVI instruction is executed.

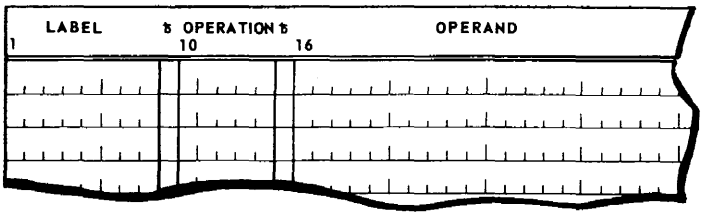
NAME

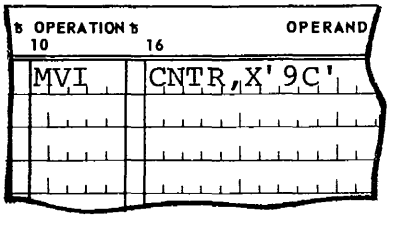
A	X	X	X	X	X

100. Some housekeeping functions, such as setting a one-byte counter to some value, can be accomplished with the MVI instruction.

Because we usually add to or subtract from the counter during processing, the initial counter value must be in packed format with a sign in the rightmost four-bit position. Let the letter C represent a positive sign and the letter D, a negative sign.

Write an instruction that places a positive 9 in a single-byte field named CNTR.





101. Check the following statements as true or false.

T	F		
☐	☐	The first operand is the receiving field.	True
☐	☐	Data to be moved must be stored in packed format.	False
☐	☐	The Move Character instruction copies the contents of one memory area into another area.	True
☐	☐	The data in the sending field is destroyed by the move operation.	False
☐	☐	A maximum of 256 bytes of data can be moved by a single MVC instruction.	True
☐	☐	The minimum of one byte of data can be moved by a single MVC instruction.	True
☐	☐	The number of bytes to be moved is determined by the length factor of the receiving field.	True
☐	☐	Clearing a print line can be accomplished by a single Move instruction.	True
☐	☐	MVC PRT+9(1Ø), NAME is an example of relative addressing.	True
☐	☐	MVI is the operation code for the Move Immediate instruction.	True
☐	☐	The self-defining value in an MVI instruction is written as the first operand.	False
☐	☐	The MVI instruction can be used to insert any character in memory.	True

102. **PREVIEW**

Instructions are usually executed one after the other in the sequence that they are entered into memory at Object Program execution time. However, the program may require a loop or branch back so that the next record may be brought into memory and processed. Frequently, this same routine is executed over and over again until hundreds or thousands of records are processed. When the last record is recognized, the branch back to the beginning of the program must not occur. Instead, control is passed to a wrap-up routine in which files are closed and the program is terminated.

Sophisticated programs are usually organized into a number of different routines, each performing a function of the logic. The programmer uses decision and branching instructions at key points to branch around routines, to branch back to an earlier routine, or to drop through and continue processing the next instruction in line. Often, this decision is based on factors within the data being processed. Therefore, it is necessary to test the data when making decisions that influence the sequence of instruction execution.

The following group of frames teaches you to use instructions that test data and provide branching options that are based on the result of the test.

103. Changing the sequence of instruction execution is usually dependent upon the combined action of *two instructions,* the Compare instruction and the Branch on Condition instruction. A Compare instruction examines two data fields and sets an internal indicator (condition code indicator) that reflects the outcome of the comparison. This same indicator is set when certain decimal arithmetic instructions are executed.

The Branch on Condition instruction tests this indicator and causes the program to branch to another routine or to continue with the next instruction in sequence (depending on the condition setting of the indicator).

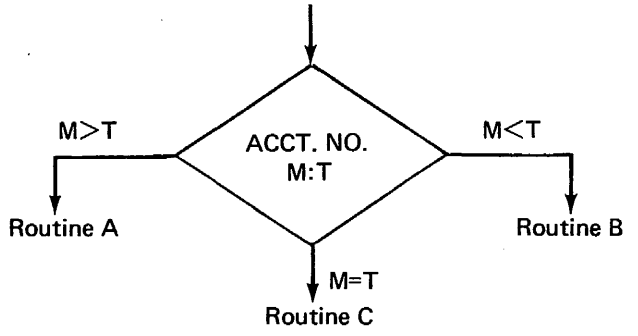
All Compare instructions in the 9200/9300 set perform two functions. They compare two fields of data and:

- ☐ add the two data items.
- ☐ set an internal condition code indicator. — set an internal indicator
- ☐ change the sequence of instruction execution.

The condition code indicator (CC) is set by execution of a Compare instruction or execution of certain _____ instructions. — decimal arithmetic

104. The Compare Logical (CLC) instruction is useful when comparing two account numbers. If we are posting transactions (T) to a master file (M), we must be certain that the transaction is posted to the correct master record.

The following illustration shows the three possible conditions.



Which routine in the program should be followed if the condition code indicates the values to be equal?

- ☐ Routine A
- ☐ Routine B
- ☐ Routine C

Routine C

If the master record account number is less than the transaction account number, which routine will be followed?_____

Routine B

LABEL	OPERATION	OPERAND
1	10	16
	CLC	MAS (4) , TRAN

In the instruction above, we have assumed that the length of each account number is four bytes. After comparing the data, the instruction will:

- ☐ set the CC indicator.
- ☐ cause the program to branch to the correct routine.

set the CC indicator

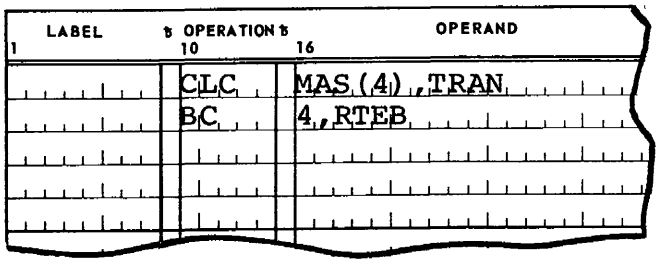
105. The Branch on Condition (BC) instruction tests the condition code settings specified by a test number in operand 1 and will branch to the address in operand 2 if the test condition is met.

Condition code test numbers are assigned as follows:

8	Branch if equal (OP1 = OP2)
4	Branch if less (OP1 < OP2)
2	Branch if greater (OP1 > OP2)
15	Branch unconditionally to OP2 regardless of condition

Example:

Compare the Master Record Account number MAS(4) with the Transaction Account number (TRAN). If the Master Record number is less than the Transaction number branch to routine B.



LABEL	OPERATION	OPERAND
	CLC	MAS(4),TRAN
	BC	4,RTEB

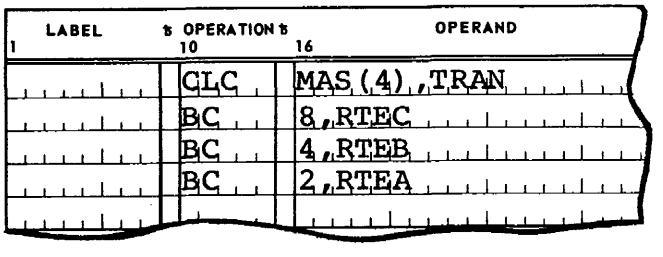
If multiple conditions are tested by a series of BC instructions, the program will branch to the OP2 address corresponding to the condition that is met.

Assume that MAS and TRAN have the following values:

MAS 1082
TRAN 1081

After the following instructions are executed what routine will be executed?_____

RTEA



LABEL	OPERATION	OPERAND
	CLC	MAS(4),TRAN
	BC	8,RTEC
	BC	4,RTEB
	BC	2,RTEA

106. The Compare Logical instruction can compare the relative binary value of two alphanumeric fields. The relative binary value is based on the complete EBCDIC value (all eight bits) of their codes (as shown in the character code chart of panel 6).

Use panel 6 to determine which of the following has the greater binary value.

E or K?	_____ is greater.	K
X or 3?	_____ is greater.	3
7 or %(percent)?	_____ is greater.	7
B or W?	_____ is greater.	W

107. Assume that the binary value of the data in FLD1 is greater. Which code segment will cause a branch to the CALC (calculate) routine?

☐
```
CLC   FLD1,FLD2
BC    2, CALC
```

☐
```
CLC   FLD1,FLD2
BC    4, CALC
```

☐
```
CLC   FLD2,FLD1
BC    8, CALC
```

☐
```
CLC   FLD1,FLD2
BC    8, CALC
```

```
CLC   FLD1,FLD2
BC    2, CALC
```

108. The second operand of a CLC instruction must not include a length factor. The number of bytes that are compared is determined by the length of the first operand.

Based on the data shown below, check the routine that will be executed next.


```
CLC     AMT(2), TOTL
```

- ☐ BC 8, COLD
- ☐ BC 4, HOT
- ☐ BC 2, WARM

HOT

```
CLC     TOTL(2),AMT
```

- ☐ BC 8, COLD
- ☐ BC 4, HOT
- ☐ BC 2, WARM

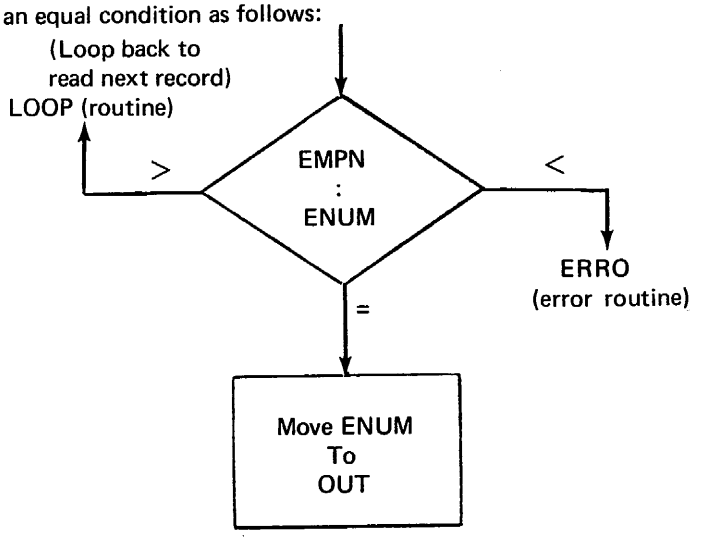
WARM

```
CLC     TOTL(1), AMT
```

- ☐ BC 8, COLD
- ☐ BC 4, HOT
- ☐ BC 2, WARM

COLD

109. In a payroll application, each employee's weekly card contains a five-character employee identification number (EMPN) in columns 1-5. Each employee master record also contains a five-character identification field named ENUM. With a record from each file in memory, the contents of the two fields are tested for an equal condition as follows:



Complete the following coding of the functions in the above flowchart.

LABEL	OPERATION	OPERAND
	CLC	
	BC	
	BC	
	MVC	OUT(5),ENUM

OPERATION	OPERAND
	EMPN(5),ENUM
	2,LOOP
	4,ERRO

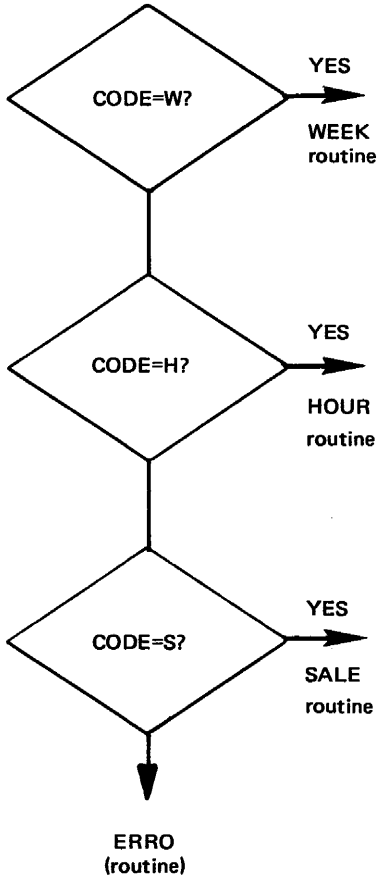
110. Column 6 of the employee card is punched with a pay code (CODE) that may be one of the following:

W = weekly pay scale.

H = hourly pay scale.

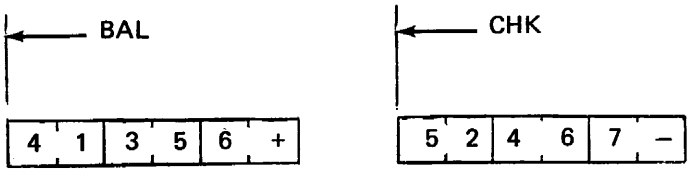
S = pay based on sales.

Obviously, the program must contain a separate procedure to calculate each type of pay. Assume the three routines are named WEEK, HOUR, and SALE. Draw and label a flowchart that illustrates the steps required to test this field of data. If W, H, or S is not present, go to ERRO routine. (Assume that the employee identification number equals the employee master record number.)



111. The Compare Packed Decimal (CP) instruction compares the relative *algebraic* value of two packed decimal fields. The first operand is compared with the second operand and the condition code indicator reflects the result of the comparison.

The Compare Packed Decimal instruction is executed byte-by-byte starting at the right-hand end of each operand. (The rightmost byte of the first operand is compared with the rightmost byte of the second operand.) The rightmost half-byte for both operands contains the sign; they are compared first. If the signs are unlike, the condition code is set to reflect the relative *algebraic* value of the operands.




```
CP    BAL(3), CHK(3)

BC    4, ROTA

BC    2, ROTB
```

After execution of the above CP instruction, control passes to the routine named:

- ☐ ROTA.
- ☐ ROTB.

ROTB

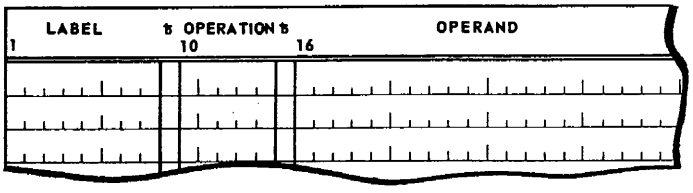
112. Assume three fields of data are in memory as follows.

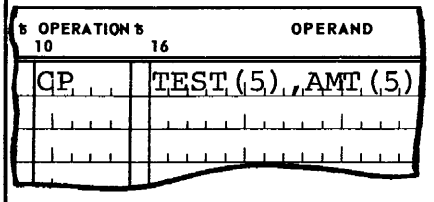
AMT — five bytes of packed decimal data

TEST — five bytes of packed decimal data

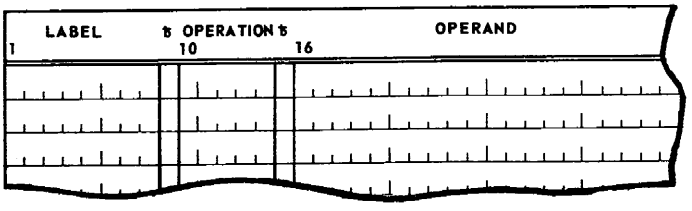
BAL — three bytes of packed decimal data

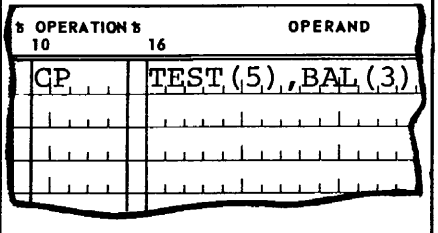
Write a Compare Packed Decimal instruction to compare the contents of TEST with the contents of AMT. (Each operand of a decimal arithmetic instruction requires a length factor.)



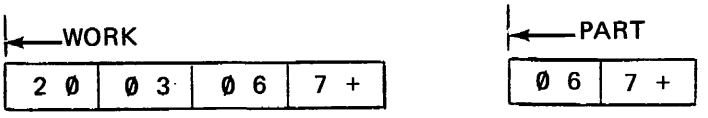


Write an instruction to compare the contents of TEST with the contents of BAL.

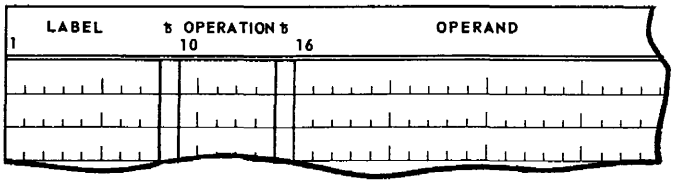


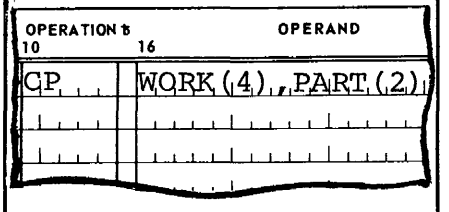


113. The operands of a Compare Packed Decimal instruction can be of unequal lengths. When operand 1 is greater in length, operand 2 is filled with packed zeros in the left-most bytes. If operand 2 is greater, the remaining digits are ignored. A sign is presumed to be in the four right-most bits of the least significant byte of both operands and is considered in the comparison.



Write an instruction to compare WORK with PART.





The instruction compares from right to left; the two bytes that contain (7+) are compared first, then the two bytes that contain (Ø6) are compared. At this point, the fields are found to be equal. What happens next?

- ☐ The instruction terminates.
- ☐ The next byte of WORK is compared with packed zeros.
- ☐ The next byte of WORK is compared with (7+).

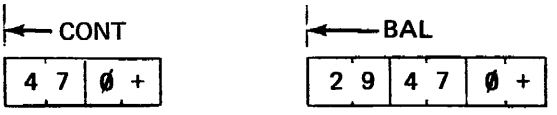
The next byte of WORK is compared with packed zeros.

When does this Compare Packed Decimal instruction terminate?

- ☐ When the leftmost byte of the second operand has been compared with zeros
- ☐ When the leftmost byte of the first operand has been compared with zeros

When the leftmost byte of the first operand has been compared with zeros

114.



Using the above data, which routine receives control after the CP instruction is executed? Check your selection.

CP CONT(2),BAL(3)

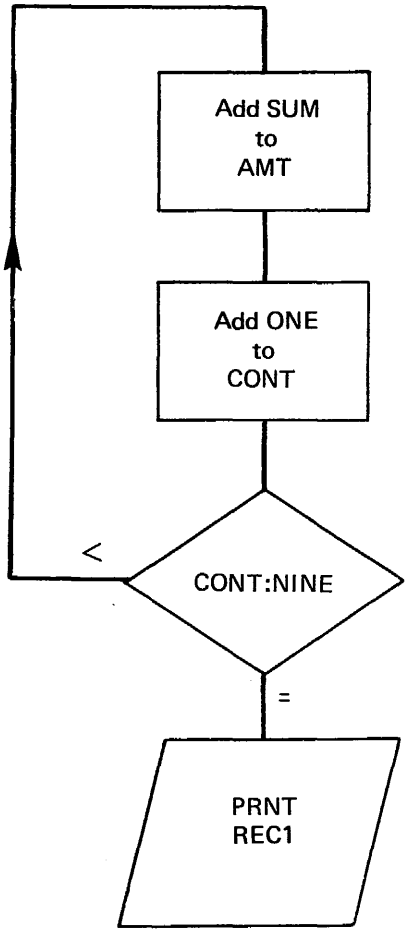
☐ BC 8,PAT1

☐ BC 2,PAT2

☐ BC 4,PAT3

BC 8,PAT1

115. The CP instruction is frequently used to test a field that is used as a counter. The CONT field in the following flowchart controls the number of times the LOOP routine will be repeated. Complete the following code by writing the instruction corresponding to the decision block of the flowchart.



LABEL	OPERATION	OPERAND
CONT	DC	X'ØC'
NINE	DC	X'9C'
ONE	DC	X'1C'
	.	
	.	
	.	
LOOP	AP	AMT(4),SUM(2)
	AP	CONT(1),ONE(1)
	BC	4,LOOP
	PUT	PRNT,REC1

OPERATION	OPERAND
CP	CONT(1),NINE(1)

116. Decision points in a program are frequently based on the outcome of arithmetic operations. Therefore, the Add Packed Decimal (AP), Subtract Packed Decimal (SP), and Zero and Add Packed Decimal (ZAP) instructions also affect the setting of the internal condition code indicator.

Check the instructions that affect the internal condition code indicator.

☐ MVC	☐ BC
☐ AP	☐ ZAP
☐ CLC	☐ PACK
☐ MVI	☐ SP
☐ CP	☐ DS
☐ MP	☐ DP

AP ZAP

CLC

SP

CP

117. The programmer may want to test the setting of the condition code indicator following the execution of an Add Packed Decimal or Subtract Packed Decimal instruction. One or more of the following instructions can be used for this purpose.

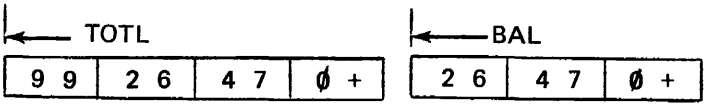
BC	1, SUBA	(Branch on Overflow)
BC	2, SUBB	(Branch on Plus)
BC	4, SUBC	(Branch on Minus)
BC	8, SUBD	(Branch on Zero)

Overflow occurs when there is a 1 carry out of the most significant position, or when the second operand is longer than the first operand. In either case, overflow is an indication that the result field (specified by the first operand) is:

- ☐ larger than the result.
- ☐ smaller than the result.
- ☐ the same size as the result.

smaller than the result

118.



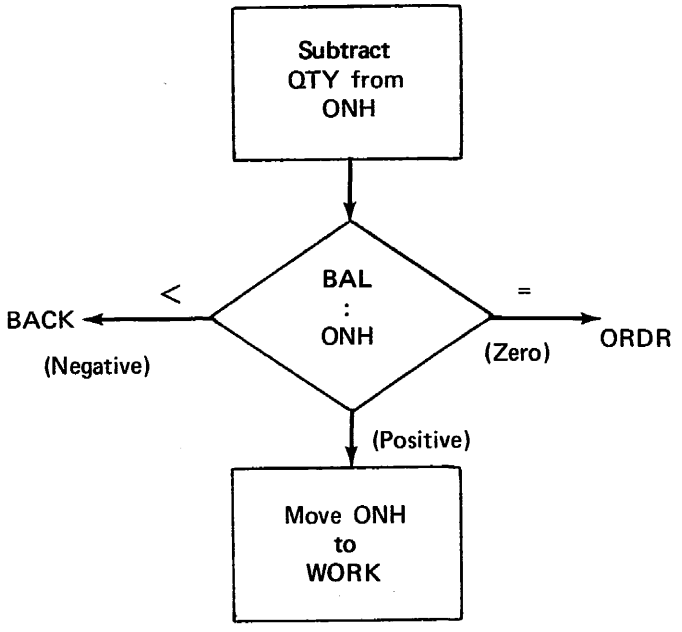
Using the above data, which routine receives control after the following SP instruction is executed? Check your selection.

```
SP    TOTL(4),BAL(3)
```

- ☐ BC 1,DEAD
- ☐ BC 8,MIKE
- ☐ BC 2,SAM
- ☐ BC 4,RICE

BC 2,SAM

119. Write a stock control application routine that will do the following: Subtract the quantity removed from stock (QTY) from the quantity-on-hand (ONH). Assume each data item is packed into three bytes. After each subtraction is executed, test the result field. If zero, branch to a reorder routine (ORDR); if negative, branch to a back-order routine (BACK); if positive, drop through and continue processing.



LABEL	OPERATION	OPERAND
	SP	

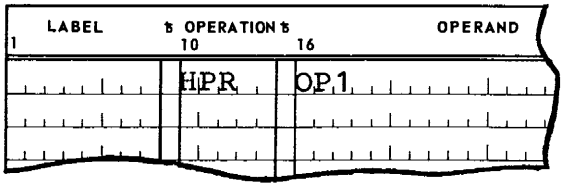
OPERATION	OPERAND
	ONH(3),QTY(3)
BC	8,ORDR
BC	4,BACK
MVC	WORK(3),ONH

120. HALT AND PROCEED (HPR)

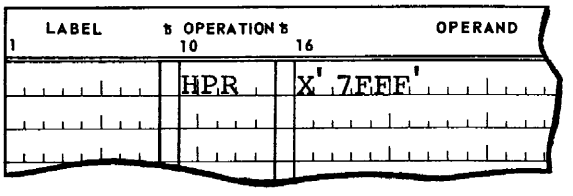
Stops the processor and displays the OP1 address in the HALT/DISPLAY indicators on the Control Console.

OP1 may include 1 to 4 hexadecimal digits.

Format:



Example:



To stop the processor, use the _____ instruction.

HPR

The HPR display indicates a normal halt of the _____ program.

user

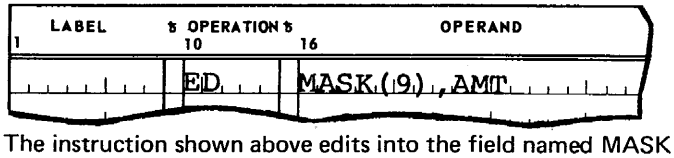
121. **PREVIEW**

When a programmer wants to print the result of an arithmetic operation, he must first unpack it to conform to the printer-graphic representation. At the same time he may want to perform editing operations, such as inserting a dollar sign and a decimal point. The Edit (ED) instruction performs unpacking and editing of packed decimal data with one operation.

The operation of the instruction is highly flexible. With proper planning it is possible to suppress nonsignificant zeros, insert commas and decimal points, insert minus signs or credit symbols, and specify where suppression of leading zeros should stop.

The following group of frames teaches you the use of the Edit instruction.

122. During an edit operation, the source field is edited under control of the edit pattern called a *mask* field.



The instruction shown above edits into the field named MASK data that is stored in the field named _____ .

AMT

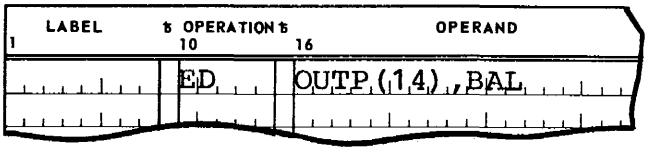
The editing proceeds according to the pattern stored in the field named _____ .

MASK

The edited data appears in the field named _____ .

MASK

123.



Which operand represents each of the following?

_____ is the field containing unedited data. — BAL

_____ is the field containing the *mask* before execution of the edit. — OUTP

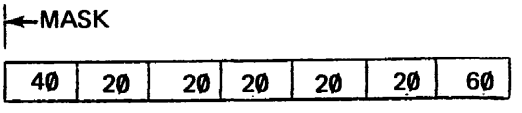
_____ is the field where edited data is located *after* execution. — OUTP

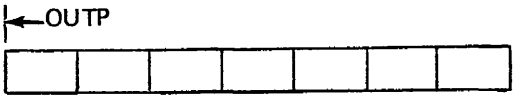
_____ is the field in packed decimal format. — BAL

124. The programmer specifies the mask by writing an X-type constant.

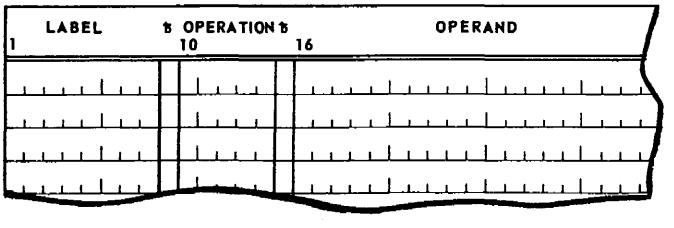
```
MASK    DC    X'4020202020206Ø'
```

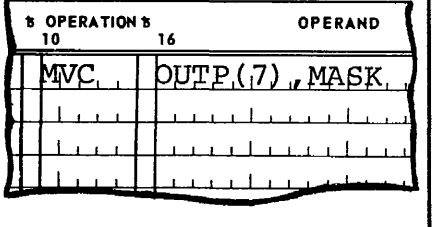
Before the first execution of the ED instruction, the mask must be moved into the field that is to contain the edited data.



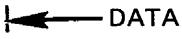


Write a move instruction in the space below that will place the mask in the OUTP field.





125. Write an edit instruction that will unpack the DATA field in the OUTP field.



ØØ	37	8C

LABEL	OPERATION	OPERAND
	ED	

OPERATION	OPERAND
	OUTP(7),DATA

126.

LABEL	OPERATION	OPERAND
	*	
AMT	DS	CL5
PRTL	DS	CL11
MASK	DC	X'4Ø2Ø2Ø2Ø2Ø2Ø2Ø2Ø2Ø2Ø2Ø6Ø'

Write two instructions to edit the data from AMT into PRTL.

LABEL	OPERATION	OPERAND

OPERATION	OPERAND
*	
MVC	PRTL(11),MASK
ED	PRTL(11),AMT

127. As the edit operation proceeds, each of the characters in the mask is examined. The first (leftmost) character in the mask is called a fill character. For many edit operations, the blank or space (hexadecimal 40) is used as a fill character, however any character can be used. (Refer to Panel 9 on page 2-123.)

The ED instruction uses a fill character to supress leading zeros, that is, to eliminate zeros that come before the first significant digit in a numeric field, replacing them with blanks (or some other character).

0000794

What is the first significant digit in the above field?_____ .

7

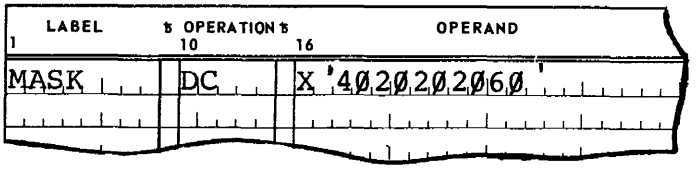
Replacing leading zeros with blanks (b), the field would appear as_____ .

bbbb794 or 794

Replacing leading zeros in 0079400 with asterisks (*) would produce_____ .

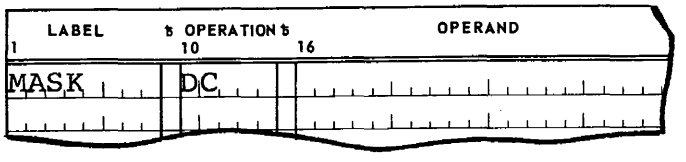
**79400

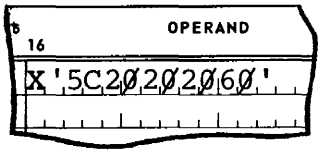
128.



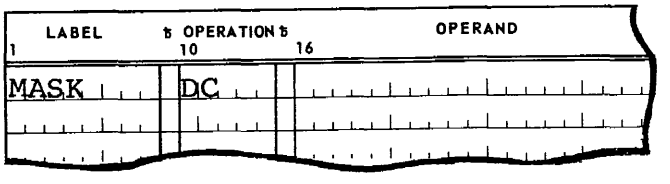
The operand of the DC statement above replaces leading zeros with blanks. The hexadecimal 20's in the mask field are always replaced during an edit operation.

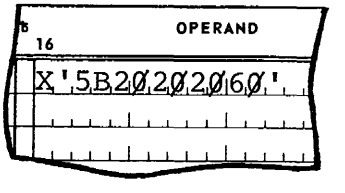
Rewrite the operand shown above so that each leading zero in the data will be replaced by an asterisk (hexadecimal 5C).



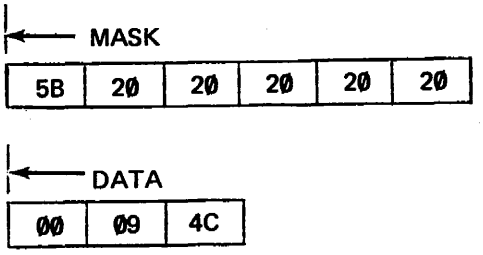


Rewrite the operand to replace the leading zeros with dollar signs (hexadecimal 5B).





129.



ED MASK(6),DATA

The above mask field contains a dollar sign fill character. Which result is produced when the data shown above is edited and printed.

- ☐ $00094
- ☐ $$$$94 — $$$$94
- ☐ $94
- ☐ $94+

130. FOOTNOTE FRAME

The following frames explain additional control characters used in the construction of edit patterns. These characters are:

Hexadecimal	Title
20	Digit select character
21	Significance start character
4B and 6B	Insert character

These characters can appear anywhere in the mask field. (Refer to Panel 9.)

NO RESPONSE REQUIRED.

131. Digit select characters (hex 20) in the mask are *always* replaced during the edit operation. They can be replaced by either the fill character or a digit from the source field.

If the source digit is not to be printed, the digit select character is replaced by_____ .

the fill character

If the source digit is to be printed, the digit select character is replaced by_____.

the source digit

132. Which hexadecimal number is used to represent each of the following (refer to panel 9)?

_____ Digit select character — 20

_____ Blank — 40

_____ Dollar sign — 5B

_____ Significance start character — 21

_____ Comma — 6B

_____ Decimal point — 4B

133. A significance start character is replaced by either the fill character or a significant (nonzero) digit from the source field. The edit operation then assigns significance to the remaining digits including zeros.

Show the edited results of the following fields:

(Use the Δ symbol to indicate a blank space.)

4ϕ	2ϕ	2ϕ	2ϕ	21	2ϕ	2ϕ

1 2 0 0 5 6 _____ — Δ 1 2 0 0 5 6

0 0 1 2 3 4 _____ — Δ Δ Δ 1 2 3 4

0 0 0 0 1 2 _____ — Δ Δ Δ Δ Δ 1 2

0 0 0 0 0 1 _____ — Δ Δ Δ Δ Δ 0 1

0 0 0 0 0 0 _____ — Δ Δ Δ Δ Δ 0 0

134. The significance start character can be used to insert a decimal point in the printed result. Show the edited results of the following fields:

4ϕ	2ϕ	2ϕ	2ϕ	21	4B	2ϕ	2ϕ

1 2 3 4 5 6	1234.56
0 0 1 2 3 4	12.34
0 0 0 0 1 2	.12
0 0 0 0 0 6	.06
0 0 0 0 0 0	.00

Show the printed results when the significance start character is omitted:

4ϕ	2ϕ	2ϕ	2ϕ	2ϕ	4B	2ϕ	2ϕ

1 2 3 4 5 6	1234.56
0 0 1 2 3 4	12.34
0 0 0 0 1 2	12
0 0 0 0 0 6	6
0 0 0 0 0 0	blanks

135. Show the printed results of the following fields after editing:

4ϕ	2ϕ	6B	2ϕ	2ϕ	2ϕ	6B	2ϕ	2ϕ	2ϕ

1 2 3 4 5 6 7	1,234,567
0 1 2 3 4 5 6	123,456
0 0 0 1 0 0 0	1,000
0 0 0 0 8 2 9	829

136. Show the printed results of the following after editing.

40	20	20	6B	20	20	21	4B	20	20

0 9 4 3 0 9 5 _____ 9,430.95

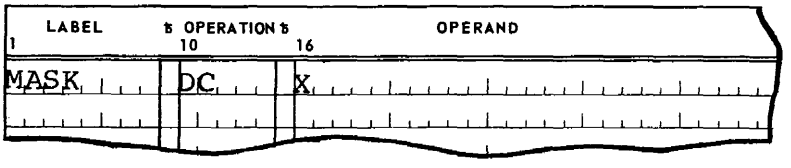
0 0 7 6 2 0 0 _____ 762.00

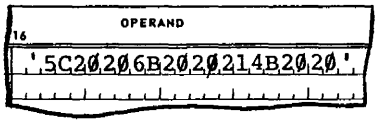
0 0 0 0 1 6 5 _____ 1.65

0 0 0 0 0 0 1 _____ .01

137. Design a mask to edit and print dollar amounts on checks. Use an asterisk (5C) fill character. It is fairly common practice to print dollar amounts with asterisks to the left of the first significant digit in order to protect against fraudulent alteration. This is called asterisk protection.

Assume that the data is a seven-digit field and that the dollar sign is preprinted on the check. The printed result is to contain a decimal point and a comma when applicable.





Show the results using the following data and your mask.

1 2 3 4 5 6 7 _____ *12,345.67

0 0 1 2 3 4 5 _____ ****123.45

0 0 0 0 1 2 3 _____ ******1.23

0 0 0 0 0 1 2 _____ *******.12

0 0 0 0 0 0 1 _____ *******.01

PANEL 1
Define the File Card Reader (DTFCR)

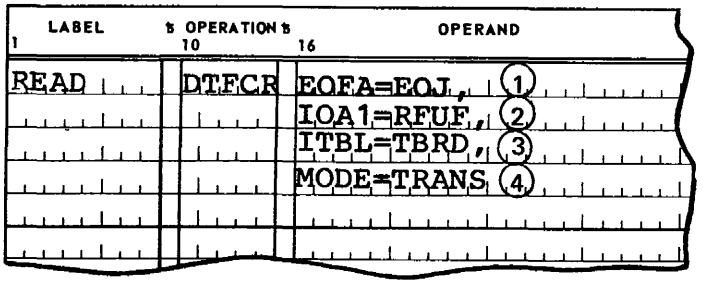
	KEYWORD	SPECIFICATIONS	
(1)	EOFA	Required	Specifies symbolic name of user defined end-of-file routine
(2)	IOA1	Required	Specifies symbolic name of user defined input buffer area
(3)	ITBL	Optional	Required if translation of input table is needed. Specifies symbolic name of user defined input translation table.
(4)	MODE	Required	Specifies that card codes are to be translated by user defined table.

NOTES:

Keywords may be coded in any sequence.

Circled numbers are used for reference only.

EXAMPLE:



PANEL 2
Define the File Printer (DTFPR)

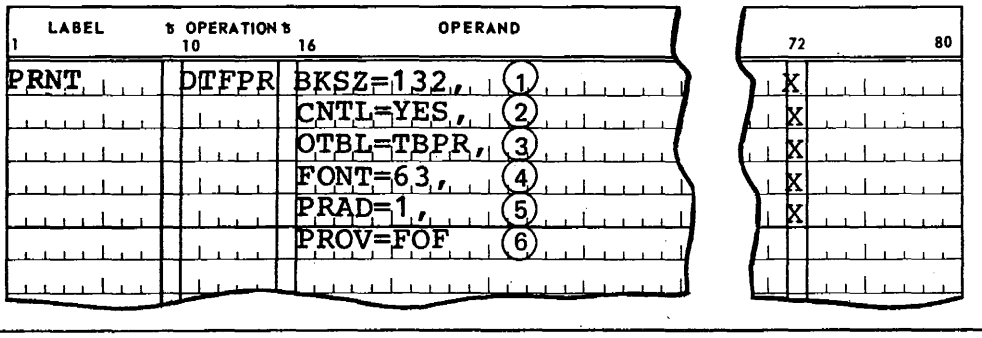
	KEYWORD	SPECIFICATIONS	
(1)	BKSZ	Required	Specifies number of required print positions (1-132)
(2)	CNTL	Optional	Required if CNTRL macro instruction coded by user will direct spacing of skipping.
(3)	OTBL	Optional	Required when 48-character font is used. Specifies symbolic name of user defined output translation table.
(4)	FONT	Required	Specifies either 48 or 63-character print bar set.
(5)	PRAD	Required	PRAD = 1 Specifies 1-line advance after printing PRAD = 2 Specifies 2-line advance after printing
(6)	PROV	Optional	Required for form overflow action by user. PROV = FOF Specifies symbolic name of user defined routine to which control will be transferred. PROV = YES Specifies automatic skip to first line of next page.

NOTE:

Keywords may be coded in any sequence.

Circled numbers are used for reference only.

EXAMPLE:



PANEL 3
Define Storage (DS) Coding Specifications

Label field	The symbolic address (tag) must be unique and should have descriptive meaning. The tag must not exceed four characters and must not have embedded blank spaces. A tag is not required when it is desired to reserve an unused area. When a tag is required, the first character must be alphabetic and must start in column 1. Special characters are not permitted in the Label field
Operation Field	The mnemonic DS must be coded in columns 10 and 11.
Operand Field	The operand must start in column 16. When the number of bytes to be reserved does not exceed 256, the first character is a C followed by the length factor L and the decimal value that specifies the number of bytes to be reserved. No more than 256 bytes can be reserved without a duplication factor.
Duplication Factor	When the number of bytes to be reserved exceeds 256, the operand starts with the duplication factor. Example: TAG DS 2CL256 (Reserves 512 bytes.)

PANEL 4
ORG Instruction Specifications

Label Field	Not used.
Operation Field	The mnemonic ORG starts in column 10.
Operand Field	The Operand field contains a previously defined symbolic address starting in column 16. The Assembler is directed to reset the Location Counter to the value of this address.

LOCTN
CNTR
ADDR

LABEL	OPERATION	OPERAND
	10	16
REC	DS	CL8Ø
	ORG	REC
FLD1	DS	CL4Ø
FLD2	DS	CL4Ø

PANEL 5

UNIVAC
DIVISION OF SPERRY RAND CORPORATION

PRINTER FORMAT CHART

UNIVAC 9000 SERIES

FORM NUMBER _____ APPLICATION _____ DATE _____

FORM PARTS _____ RUN NAME _____ RUN NUMBER _____ PREPARED BY _____

TYPE OF PRINTOUT _____ RECORD NAME _____ RECORD NUMBER _____ APPROVED BY _____

DATE APPROVED _____

LINE NAME	TYPE H,D,T	Space BEFORE	Space AFTER	Skip BEFORE	Skip AFTER	A/N	LINE NO.

CARRIAGE CONTROL TAPE

CHANNELS 1 2 4

DATE XXXXXX

INVENTORY REPORT

PART NUMBER	DESCRIPTION	QUANTITY	UNIT PRICE
XXXXX	XXXXXXXXXXXXXXXXXXXXXXXXX	XXXX	XXXXX

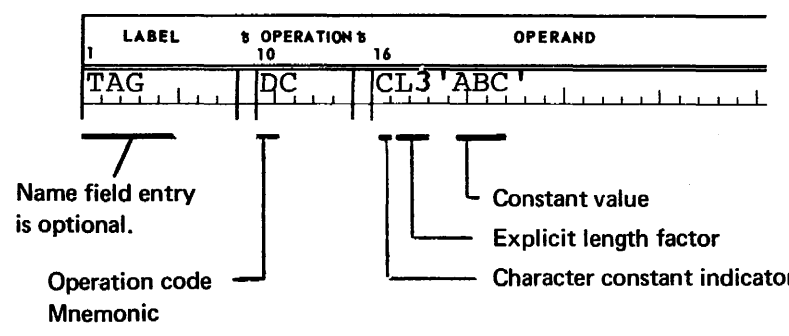
PANEL 6
Character Codes

HEX	EBCDIC		Printer Graphics
C1	1100	0001	A
C2	1100	0010	B
C3	1100	0011	C
C4	1100	0100	D
C5	1100	0101	E
C6	1100	0110	F
C7	1100	0111	G
C8	1100	1000	H
C9	1100	1001	I
D1	1101	0001	J
D2	1101	0010	K
D3	1101	0011	L
D4	1101	0100	M
D5	1101	0101	N
D6	1101	0110	O
D7	1101	0111	P
D8	1101	1000	Q
D9	1101	1001	R
E2	1110	0010	S
E3	1110	0011	T
E4	1110	0100	U
E5	1110	0101	V
E6	1110	0110	W
E7	1110	0111	X
E8	1110	1000	Y
E9	1110	1001	Z
F0	1111	0000	0
F1	1111	0001	1
F2	1111	0010	2
F3	1111	0011	3
F4	1111	0100	4
F5	1111	0101	5
F6	1111	0110	6
F7	1111	0111	7
F8	1111	1000	8
F9	1111	1001	9

HEX	EBCDIC		Printer Graphics
FF	1111	1111	¤ (lozenge)
40	0100	0000	(space)
4A	0100	1010	¢ (cents)
4B	0100	1011	. (period)
4C	0100	1100	< (less than)
4D	0100	1101	((open parenthesis)
4E	0100	1110	+ (plus)
4F	0100	1111	| (vertical)
50	0101	0000	& (ampersand)
5A	0101	1010	! (exclamation)
5B	0101	1011	$ (dollar sign)
5C	0101	1100	* (asterisk)
5D	0101	1101	) (close parenthesis)
5E	0101	1110	; (semicolon)
5F	0101	1111	¬ (logical NOT)
60	0110	0000	- (minus)
61	0110	0001	/ (slash)
6A	0110	1010	∧ (logical AND)
6B	0110	1011	, (comma)
6C	0110	1100	% (percent)
6D	0110	1101	_ (underline)
6E	0110	1110	> (greater than)
6F	0110	1111	? (question mark)
7A	0110	1010	: (colon)
7B	0111	1011	# (number)
7C	0111	1100	@ (at rate of)
7D	0111	1101	' (apostrophe or single quote)
7E	0111	1110	= (equal)
7F	0111	1111	" (quotes)

PANEL 7
Define Character Constant Coding Specifications

Example:



Label Field – A tag is optional in the name field. Coding specifications for the name field are the same as for the Define Storage operation (see panel 3).

Operation Field – The mnemonic DC is coded in columns 10 and 11.

Operand Field – The Operand field designates the actual constant to be stored. The Operand coding starts in column 16.

Constant Value

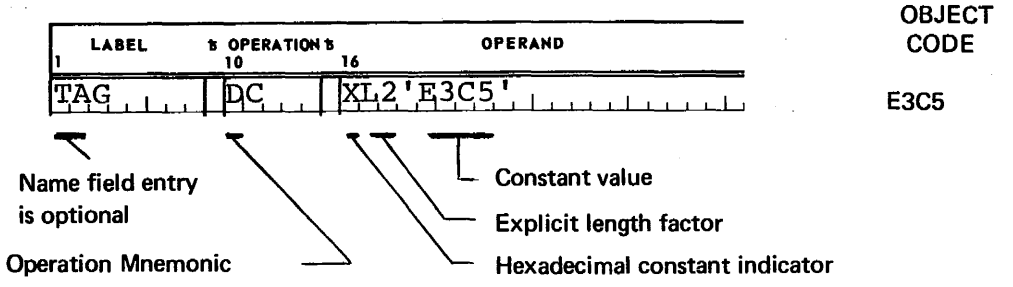
The constant value must be enclosed within single quotes. A single DC statement can specify no more than 16 characters. When the constant exceeds 16 characters, including embedded blanks, one or more additional DC statements must be specified.

Explicit Length Factor

The explicit length factor can be omitted. When it is, the implied length is assumed. (The implied length is the length of the constant value within the enclosing quotes.) When specified, the explicit length overrides the implied length. When the explicit length differs from the implied length, the result field is truncated or padded with blanks (hex 40) on the right side.

PANEL 8
Define Hexadecimal Constant Coding Specifications

Example:



Label Field – A tag is optional in the name field. Coding specifications for the name field are the same as for the Define Storage operation (see panel 3).

Operation Field – The mnemonic DC is coded in columns 10 and 11.

Operand Field – The Operand field designates the actual constant to be stored. The Operand coding starts in column 16.

Constant Value

The constant value must be enclosed within single quotes. A single DC statement can specify no more than 16 bytes. When the constant value exceeds 16 bytes, one or more additional DC statements must be specified.

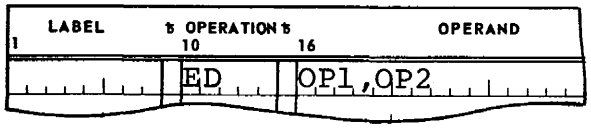
Explicit Length Factor

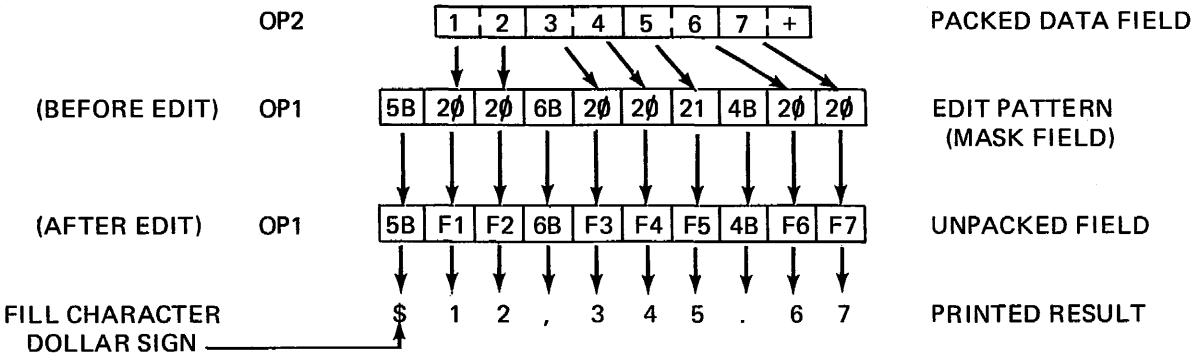
The explicit length factor can be omitted. When it is, the implied length is assumed. (The implied length is the length of the constant value within the enclosing quotes.) When specified, the explicit length overrides the implied length. When the explicit length differs from the implied length, the result field is truncated or padded with hexadecimal zeros (ØØ) on the left side.

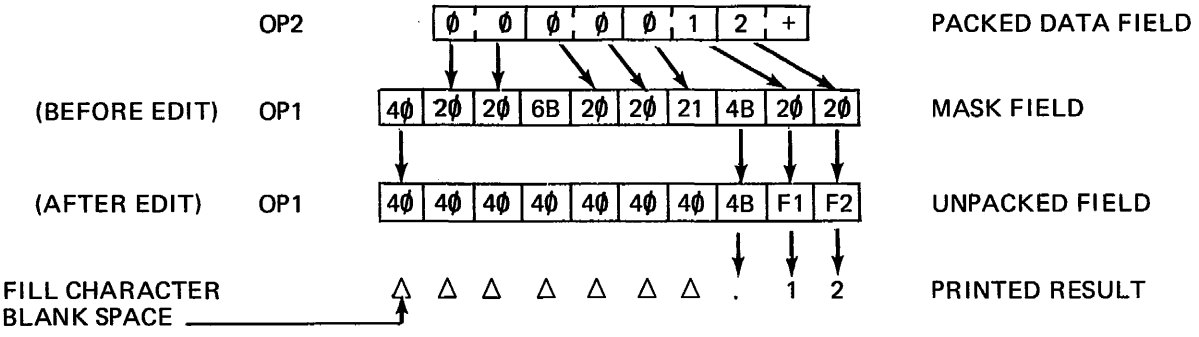
PANEL 9

EDIT INSTRUCTION EXAMPLES

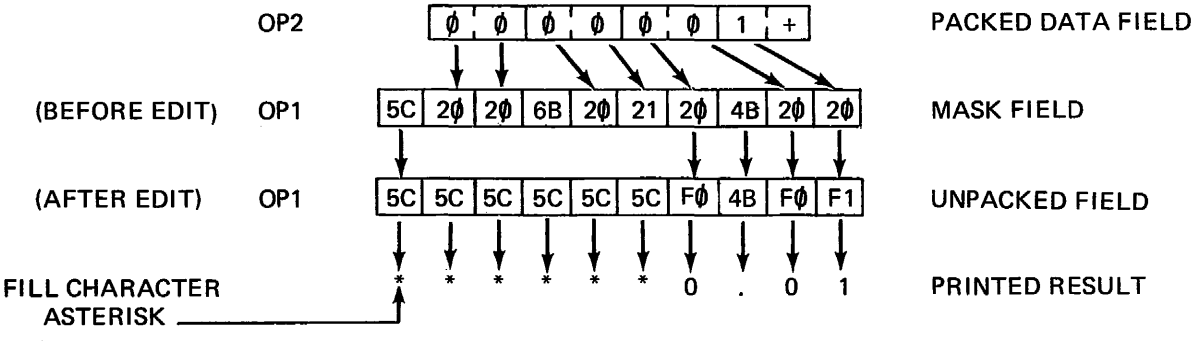
SEE NOTE BELOW







(The symbols Δ or b are used by programmers to indicate blank spaces when checking number of spaces in the result.)



NOTE

Edit control hexadecimal characters:

- Fill character (40, 5B, 5C, etc.)
- Digit Select Character (20)
- Significance Start Character (21)
- Insert characters (6B, 4B)

Check each of the following statements as true or false. Check your answers on page 2-129.

T	F		
☐	☐	1.	An EOF card indicates the beginning of a punched-card file.
☐	☐	2.	An object program is in machine code.
☐	☐	3.	Machine code is used to write a source program.
☐	☐	4.	A systems flowchart is prepared by a programmer.
☐	☐	5.	A program flowchart is prepared by an operator.
☐	☐	6.	AP in a coded statement represents an operation code.
☐	☐	7.	A program is a series of instructions.
☐	☐	8.	The basic function of high-speed memory is processing.
☐	☐	9.	A file name must have nine characters.
☐	☐	10.	A byte contains six bits.
☐	☐	11.	The symbols in a systems flowchart do not represent operands.
☐	☐	12.	The symbols in a program flowchart represent devices.
☐	☐	13.	An Assembler program is produced by the computer manufacturer.
☐	☐	14.	Data files are brought into memory and processed at object time.
☐	☐	15.	The keywords in a DTF statement must be listed in a prescribed sequence.
☐	☐	16.	Card file records are 80 bytes in length.
☐	☐	17.	A DTFCR statement must include the keyword entry IOA1.
☐	☐	18.	A storage area is cleared for use in arithmetic calculation by filling it with spaces.
☐	☐	19.	A storage area is cleared for use in a printing operation by filling it with spaces.
☐	☐	20.	The Edit instruction is a logical instruction.
☐	☐	21.	A DTFPR statement is an Imperative Macro instruction.
☐	☐	22.	Connector symbols can be used on a flowchart to represent a repetitive process.
☐	☐	23.	Data for arithmetic calculations must be in packed decimal format.
☐	☐	24.	A counter can be used to control the number of times a loop will be repeated.
☐	☐	25.	At Assembly time, IOCS routines called for by the programmer become part of the user program.

T	F		
☐	☐	26.	Constants are not stored in memory as hexadecimal values.
☐	☐	27.	One hexadecimal digit is represented by four bits in memory.
☐	☐	28.	A numeric value is used to increment a counter.
☐	☐	29.	Two hexadecimal digits are represented by four binary digits.
☐	☐	30.	The representation of a constant in the Assembly listing is in hexadecimal.
☐	☐	31.	The programmer supplied information about the files in the DTF statements.
☐	☐	32.	Any constant defined as a character constant cannot be defined as a hexadecimal constant.
☐	☐	33.	A program with one input file and one output file requires three DTF statements.
☐	☐	34.	Hexadecimal constants can be used to define printer graphic characters.
☐	☐	35.	Hexadecimal constants are used to define numeric values that will be used in arithmetic operations.
☐	☐	36.	An input area named in a DTF statement does not require a storage area to be reserved by a DS statement.
☐	☐	37.	The response to the keyword IOA1= must be the symbolic name assigned to the I/O area of memory.
☐	☐	38.	When defining a constant, the explicit length is never specified.
☐	☐	39.	The constant value defined in a character constant must be enclosed within a pair of single quotation marks.
☐	☐	40.	A Declarative Macro instruction defines an input or output file.
☐	☐	41.	The last keyword entry in a DTF must be followed by a comma.
☐	☐	42.	An unused field defined by a DS statement must be named.
☐	☐	43.	Univac provides IOCS macro routines.
☐	☐	44.	Card input files are defined by Imperative Macro instructions.
☐	☐	45.	Printline storage can be reserved for 132 bytes.
☐	☐	46.	The Label in a DS statement represents the symbolic address of the second byte of the defined field.
☐	☐	47.	The ORG instruction resets the location counter to permit a storage area to be redefined.
☐	☐	48.	Two areas in memory can be reserved by the same label.
☐	☐	49.	A DS statement can be used to define storage for an input area only.

T	F		
☐	☐	50.	The ORG instruction is an Assembler-directing instruction.
☐	☐	51.	The first character in the Label field of a DC statement cannot be numeric.
☐	☐	52.	The MVC instruction is a Declarative Macro instruction.
☐	☐	53.	The implied length of a constant is indicated within the enclosing quotation marks.
☐	☐	54.	When a numeric value is used in arithmetic operations, it must be in packed format.
☐	☐	55.	The START instruction follows the DTF statements.
☐	☐	56.	Printer graphic symbols are stored in memory as hexadecimal characters.
☐	☐	57.	A DC statement defines a constant and also reserves storage for the value to be stored.
☐	☐	58.	An input or output area named in a DTF statement must be reserved by a DS statement.

Statements 59 through 64 are referenced to the following instruction example. Check them as true or false.

```
AP      WORK(4), SALE(3)
```

☐	☐	59.	SALE is the receiving field address.
☐	☐	60.	The data stored in SALE will not be added to the data stored in WORK.
☐	☐	61.	WORK is the first operand address.
☐	☐	62.	After execution, the result will be stored in WORK.
☐	☐	63.	The data originally stored in WORK will be destroyed when the instruction is executed.
☐	☐	64.	After execution, the data originally stored in SALE will be changed.
☐	☐	65.	A DTF statement is the last instruction coded in a source program.
☐	☐	66.	A Move Character (MVC) instruction operates on packed data only.
☐	☐	67.	An Edit (ED) instruction prepares packed data for printing.
☐	☐	68.	The specified explicit length of the constant XL2'F1F2' is the same as its implied length.
☐	☐	69.	The data in the sending field is destroyed by a move operation.
☐	☐	70.	The Move Immediate (MVI) instruction operates on one character only.
☐	☐	71.	The constant C'1234' will appear in hexadecimal form as F1F2F3F4.
☐	☐	72.	A MVC instruction copies data from one field to another.
☐	☐	73.	Data cannot be moved unless it is stored in packed format.

T F

☐ ☐ 74. A work area cannot be cleared by a single MVC instruction.

☐ ☐ 75. An Add Decimal (AP) instruction stores the sum of two packed fields in the first operand field.

☐ ☐ 76. A printline area can be cleared by a single MVC instruction.

☐ ☐ 77. A Compare Packed (CP) instruction sets a condition code indicator that reflects the result of comparing two packed data fields.

☐ ☐ 78. The Compare Logical (CLC) instruction does not operate on alphanumeric data.

☐ ☐ 79. The condition code indicator is tested by a Branch on Condition instruction after a compare operation is performed.

☐ ☐ 80. An example of relative addressing is:

```
MVC     PRT+9(10),NAME
```

☐ ☐ 81. The Using instruction is an Assembler-directing instruction.

☐ ☐ 82. The Using instruction defines a constant value.

☐ ☐ 83. The DTFPR statement is a Declarative macro instruction.

☐ ☐ 84. The BC instruction follows the CLC instruction.

☐ ☐ 85. The BC instruction is an Assembler-directing instruction.

☐ ☐ 86. A DTF statement cannot specify a peripheral device.

☐ ☐ 87. The filename that identifies a peripheral device is assigned by the programmer in a DTF statement.

☐ ☐ 88. The BAL and USING instructions are not DTF statements.

☐ ☐ 89. Blank spaces are allowed between characters in the Label field of a DS or DC statement.

☐ ☐ 90. The last keyword entry in a DTF statement must be followed by a comma and a continuation character.

☐ ☐ 91. The START instruction is the first instruction coded in a program.

☐ ☐ 92. The DTF statements follow the START instruction.

☐ ☐ 93. The BAL instruction precedes the START instruction.

☐ ☐ 94. A CLOSE instruction is the last instruction in a program.

☐ ☐ 95. The GET instruction is an Imperative Macro instruction.

T F

☐ ☐ 96. A PUT macro instruction precedes a CLOSE macro instruction.

☐ ☐ 97. The Label field of a DS statement must not exceed four characters.

☐ ☐ 98. The keyword BKSZ is not required in a DTFPR statement.

☐ ☐ 99. The Assembler listing prints all EBCDIC values in hexadecimal.

☐ ☐ 100. Internally, the computer represents all printer graphic symbols in machine code.

SELF–TEST ANSWERS

1.	F	26.	T	51.	T	76.	T
2.	T	27.	T	52.	F	77.	T
3.	F	28.	T	53.	T	78.	F
4.	F	29.	F	54.	T	79.	T
5.	F	30.	T	55.	F	80.	T
6.	T	31.	T	56.	F	81.	T
7.	T	32.	F	57.	T	82.	F
8.	F	33.	F	58.	T	83.	T
9.	F	34.	T	59.	F	84.	T
10.	F	35.	T	60.	F	85.	F
11.	T	36.	F	61.	T	86.	F
12.	F	37.	T	62.	T	87.	T
13.	T	38.	F	63.	T	88.	T
14.	T	39.	T	64.	F	89.	F
15.	F	40.	T	65.	F	90.	F
16.	T	41.	F	66.	F	91.	T
17.	T	42.	F	67.	T	92.	T
18.	F	43.	T	68.	T	93.	F
19.	T	44.	F	69.	F	94.	F
20.	T	45.	T	70.	T	95.	T
21.	F	46.	F	71.	T	96.	T
22.	T	47.	T	72.	T	97.	T
23.	T	48.	F	73.	F	98.	F
24.	T	49.	F	74.	F	99.	T
25.	T	50.	T	75.	T	100.	T