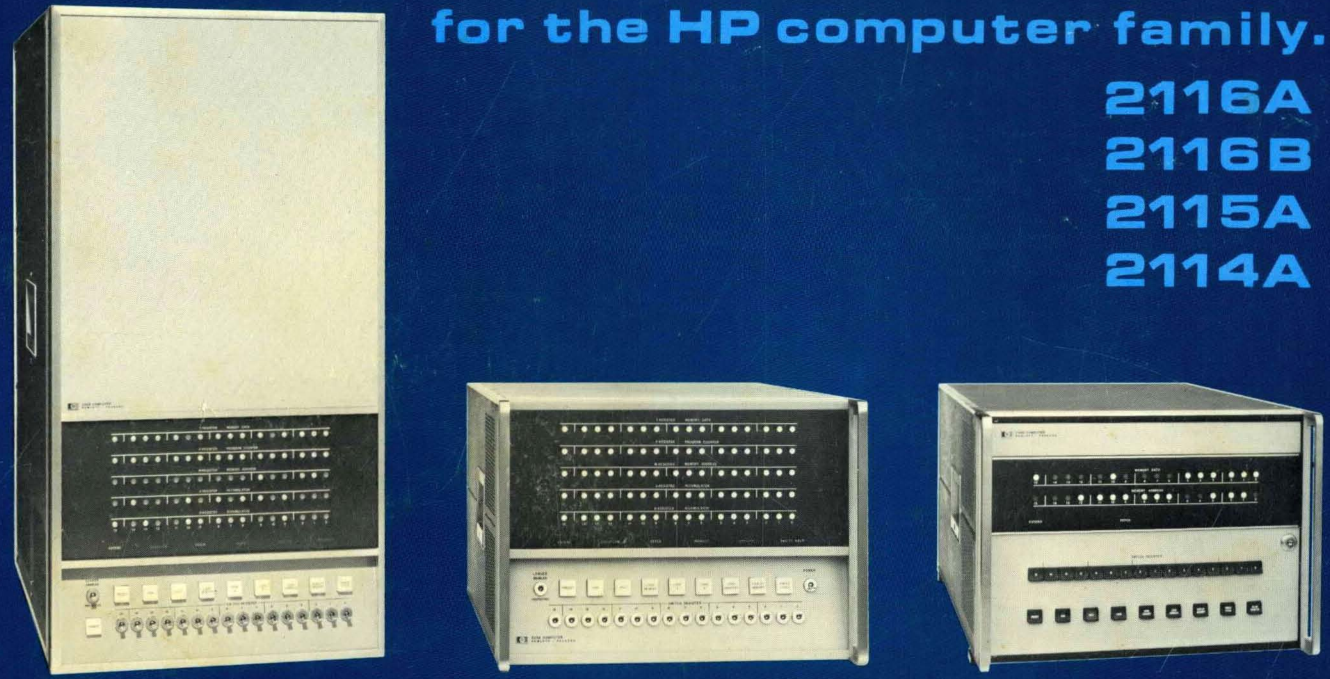
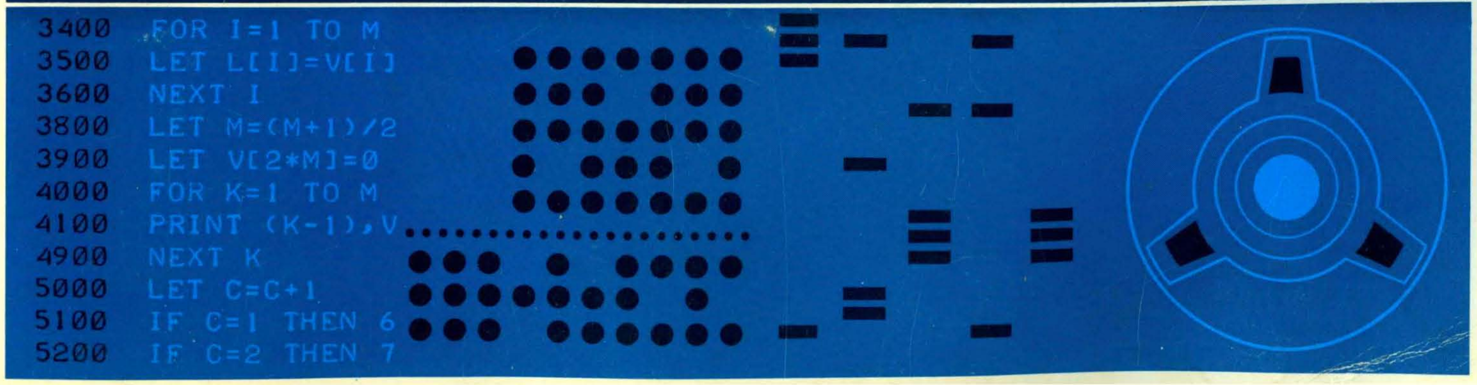
HEWLETT hp PACKARD

PROGRAMMING COURSE

for the HP computer family.

2116A
2116B
2115A
2114A





```
3400  FOR I=1 TO M
3500  LET L[I]=V[I]
3600  NEXT I
3800  LET M=(M+1)/2
3900  LET V[2*M]=0
4000  FOR K=1 TO M
4100  PRINT (K-1),V
4900  NEXT K
5000  LET C=C+1
5100  IF C=1 THEN 6
5200  IF C=2 THEN 7
```

HEWLETT-PACKARD

COMPUTER PROGRAMMING COURSE

STUDENTS MANUAL

(HP STOCK NO. 5950-8312)

-NOTICE-

11000 Wolfe Road, Cupertino, California 95014 Area Code 408 257-7000 TWX 910-338-0221

Printed in U.S.A. APR 69

FOREWORD

Welcome to the Hewlett-Packard Computer Programming Course. We are very pleased to have you attend this training program and we will do our best to make your stay with us interesting and profitable.

About Hewlett-Packard

The Hewlett-Packard Company specializes in the manufacture of instruments and systems to satisfy many of the measurement and computation needs for science and industry. Today, Hewlett-Packard provides over 1500 different products for electronic, chemical and medical instrumentation applications.

Since its founding in Palo Alto almost thirty years ago, Hewlett-Packard has grown from a two-man operation into a world-wide organization of more than 12,000 people, with an annual sales volume exceeding $225 million. The company and its affiliates now have more than a dozen manufacturing plants including facilities in the United States, Western Europe and Japan. Sales and service offices are located in nearly every major city in the free world.

About the Computer Programming Course

The HP Computer Programming Course has been developed to train personnel in the use and operation of the HP computer systems. The course curriculum has evolved to its present level primarily as a result of suggestions by the many thoughtful and interested students who have preceded you. In line with Hewlett-Packard's corporate-wide policy regarding the quality of its training support, much time and effort has been expended to provide you with this planned program for learning about computers and computer programming in general, and specifically about the Hewlett-Packard equipment which you already have or are planning to purchase.

Our experience, after training in excess of 300 students, has shown that our student experience profile breaks down as follows:

1. 60% having no previous experience in computers.
2. 23% having less than one year experience.
3. 17% having more than one year experience.

Based on these figures we have attempted to orient the level of training to the 60% group while still including some degree of challenge to the students with some previous computer programming experience.

Our overall objective is to prepare each of you for the task you face in utilizing the computer system to solve your individual application problems. In order to achieve this objective the combined efforts of both student and instructor will be required.

For those of you with no previous experience the road will not be easy; however, the objective can be reached provided you make every effort possible to communicate with your instructors by your questions during and after class sessions. We believe you will find your individual instructors to be capable and interested in your desire to learn.

For those of you who have had some previous experience in programming a computer, the training program will provide answers to questions you may have and provide the "hands on" experience with the Hewlett-Packard software systems. Since your training program will be an easier one to adjust to we would like to ask your help in training those classmates of yours who may be having difficulty. By your willingness to contribute your ideas and efforts, the attainment of our objective goals to successfully prepare all of you for the tasks you face, upon leaving us, will be assured.

Training Staff
Cupertino Division

LESSON INDEX

Introduction to computers

LESSON I
INTRODUCTION TO COMPUTERS

HEWLETT-PACKARD
COMPUTER PROGRAMMING COURSE

OBJECTIVES:

1. TEACH THE STUDENT HOW TO CREATE SIMPLE FORTRAN AND ASSEMBLY LANGUAGE COMPUTER PROGRAMS.

2. PROVIDE EACH STUDENT WITH "HANDS ON" COMPUTER EXPERIENCE.

3. TEACH THE STUDENT HOW TO USE STANDARD HEWLETT-PACKARD SOFTWARE.

HEWLETT-PACKARD COMPUTER PROGRAMMING COURSE

LESSON PLAN

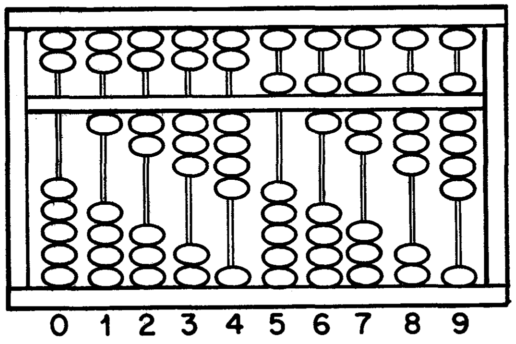
HEWLETT-PACKARD
COMPUTER PROGRAMMING COURSE

TRAINING AIDS:

1. OVERHEAD SLIDES
2. STUDENT TRAINING MANUAL
3. CLASSROOM EXERCISES
4. HOMEWORK ASSIGNMENTS
5. COMPUTER LABORATORY EXERCISES

THE ABACUS

THE HISTORY OF COMPUTER DEVELOPMENT PROBABLY STARTED WITH THE INVENTION OF THE ABACUS. THIS DEVICE WAS CREATED IN CHINA APPROXIMATELY 600 BC.

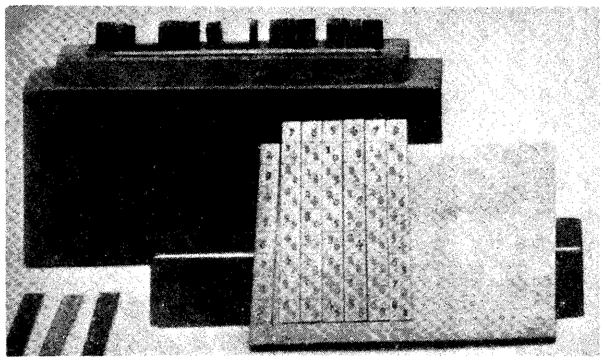


THE ABACUS

IT SHOULD BE NOTED THAT THE ABACUS IS STILL USED EXTENSIVELY IN THE ORIENT.

LOGARITHMS

IN THE EARLY 17th CENTURY JOHN NAPIER INVENTED LOGARITHMS AND ALSO A MULTIPLICATION TABLE THAT WAS REPRODUCED ON PIECES OF BONE AND SUBSEQUENTLY REFERRED TO AS "NAPIERS BONES".

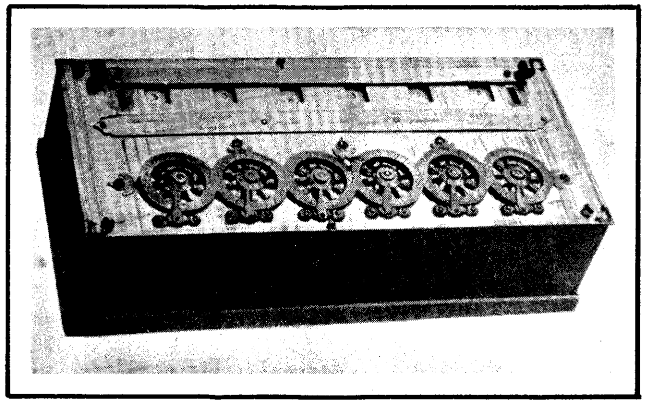


"NAPIERS BONES"

SHORTLY AFTER THE INVENTION OF LOGARITHMS WILLIAM OUGHTRED INSCRIBED LOGARITHMS ON SLIDING PIECES OF WOOD AND THE SLIDE RULE CAME INTO EXISTENCE.

PASCAL'S ADDING MACHINE

IN 1642 BLAISE PASCAL, A FRENCH MATHEMATICIAN BUILT WHAT WAS PROBABLY THE WORLDS FIRST DESK CALCULATOR.

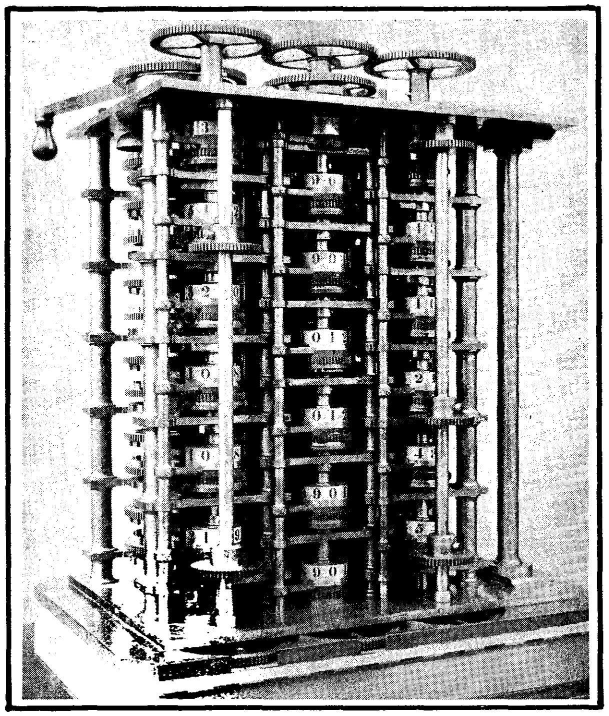


PASCAL'S ADDING MACHINE (1642)

THIS MACHINE WAS DESIGNED TO HELP PASCAL'S FATHER IN KEEPING THE BOOKS OF THE FAMILY STORE.

BABBAGE'S CONTRIBUTION

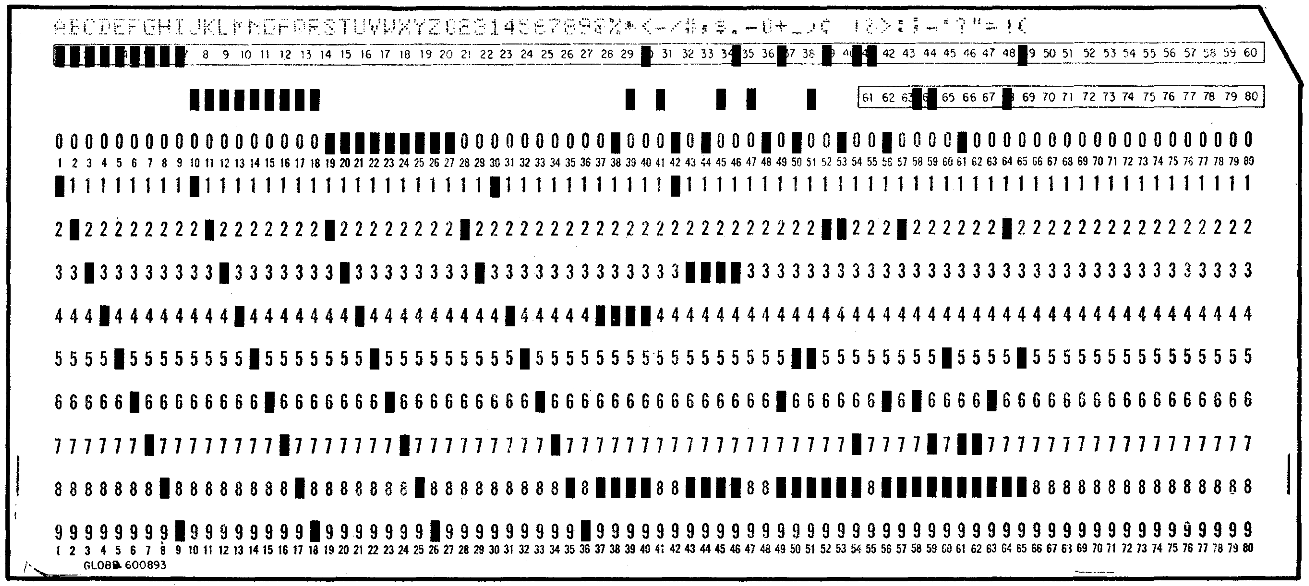
THE NEXT MAJOR MILESTONE WAS CONTRIBUTED BY CHARLES BABBAGE. IN 1822 BABBAGE DEMONSTRATED HIS "DIFFERENCE ENGINE", A MACHINE DESIGNED TO PREPARE TABLES SUCH AS COMPOUND INTEREST, LOGARITHMS AND TRIGONOMETRIC FUNCTIONS, WITHOUT THE HELP OF A HUMAN OPERATOR.



BABBAGE'S DIFFERENCE ENGINE (1822)

THE PUNCHED CARD

In 1890 Herman Hollerith invented the punched card. The original motivation for this invention was a desire to speed up the job of taking the census of the United States.



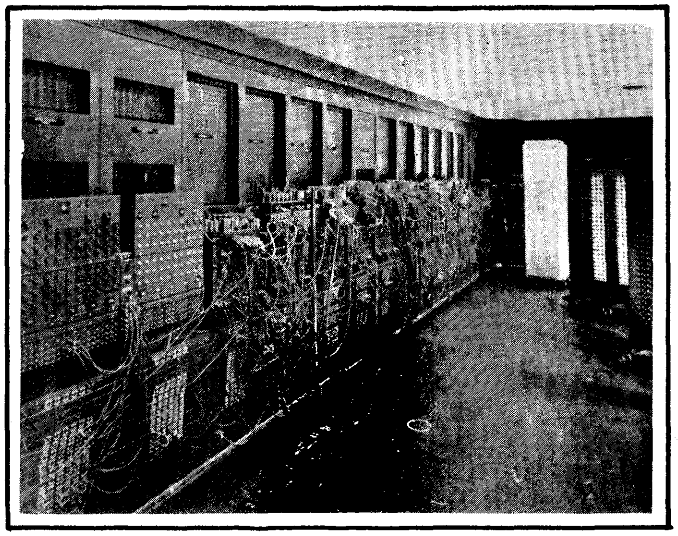
The format used to describe alphanumeric data in modern computers is called "H" or "HOLLERITH" format.

STEPS TO THE MODERN COMPUTER

IN 1939 SAMUEL WILLIAMS OF BELL LABS BUILT THE RELAY COMPUTER. THIS WAS THE FIRST ELECTRICAL DIGITAL COMPUTER AND THE FIRST BINARY MACHINE.

IN 1944 PROFESSOR HOWARD AIKEN DESIGNED THE HARVARD MARK I. THIS WAS THE FIRST OF THE LARGE SCALE GENERAL PURPOSE RELAY COMPUTERS BUILT DURING THIS PERIOD.

IN 1946 J.P. ECKERT AND DR. J.W. MAUCHLY OF THE MOORE SCHOOL OF ENGINEERING DEVELOPED THE ENIAC. THIS WAS THE FIRST ELECTRONIC DIGITAL COMPUTER AND IT CONTAINED 18,000 VACUUM TUBES.

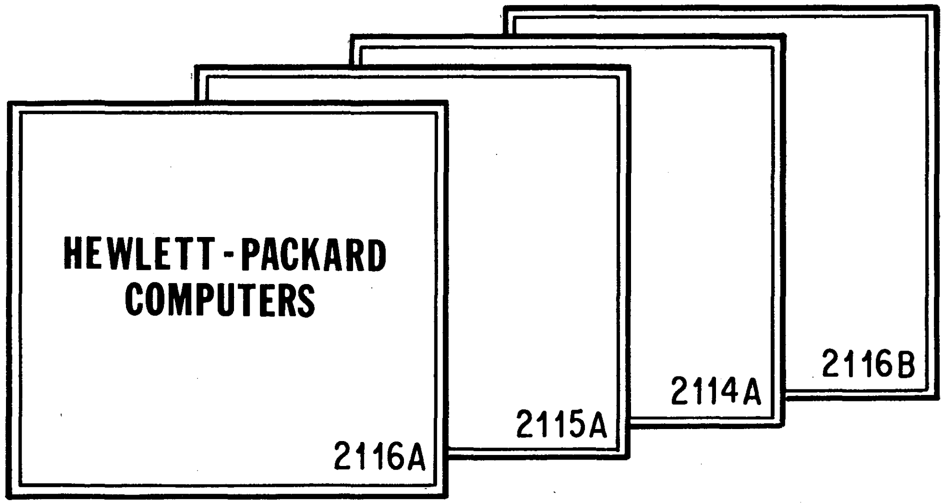


THE ENIAC

THE MODERN COMPUTER

By the 1950's IBM and others were marketing vacuum tube computers that would perform 60 operations per second.

In the late 1950's and early 1960's vacuum tubes gave way to transistors and faster memories were built.



In 1966 HEWLETT-PACKARD entered the computer market and has since contributed a family of low cost high-speed computers using integrated circuits.

COMPUTERS MUST BE PROGRAMMED

ASSUME THE FOLLOWING PROBLEM IS TO BE SOLVED BY A GIRL USING A DESK CALCULATOR –

$$X = \frac{A+B}{C+D}$$

IN MOST CASES THE GIRL WOULD HAVE TO BE PROVIDED WITH A PROCEDURE TO SOLVE THE PROBLEM.

FOR EXAMPLE:

STEP 1. ENTER VALUE FOR C.
STEP 2. ADD THE VALUE OF D.
STEP 3. WRITE DOWN INTERMEDIATE RESULT.
STEP 4. ENTER VALUE FOR A.
STEP 5. ADD THE VALUE OF B.
STEP 6. DIVIDE BY THE VALUE ACHIEVED IN STEP 3.
STEP 7. WRITE DOWN THE FINAL RESULT.

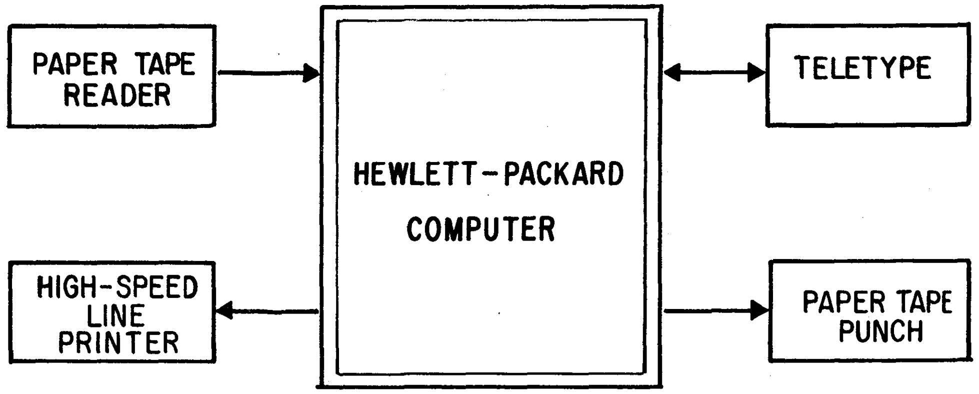
IN A SIMILAR MANNER COMPUTERS ARE "PROGRAMMED" TO SOLVE PROBLEMS.

SPEED AND INTERCOMMUNICATION

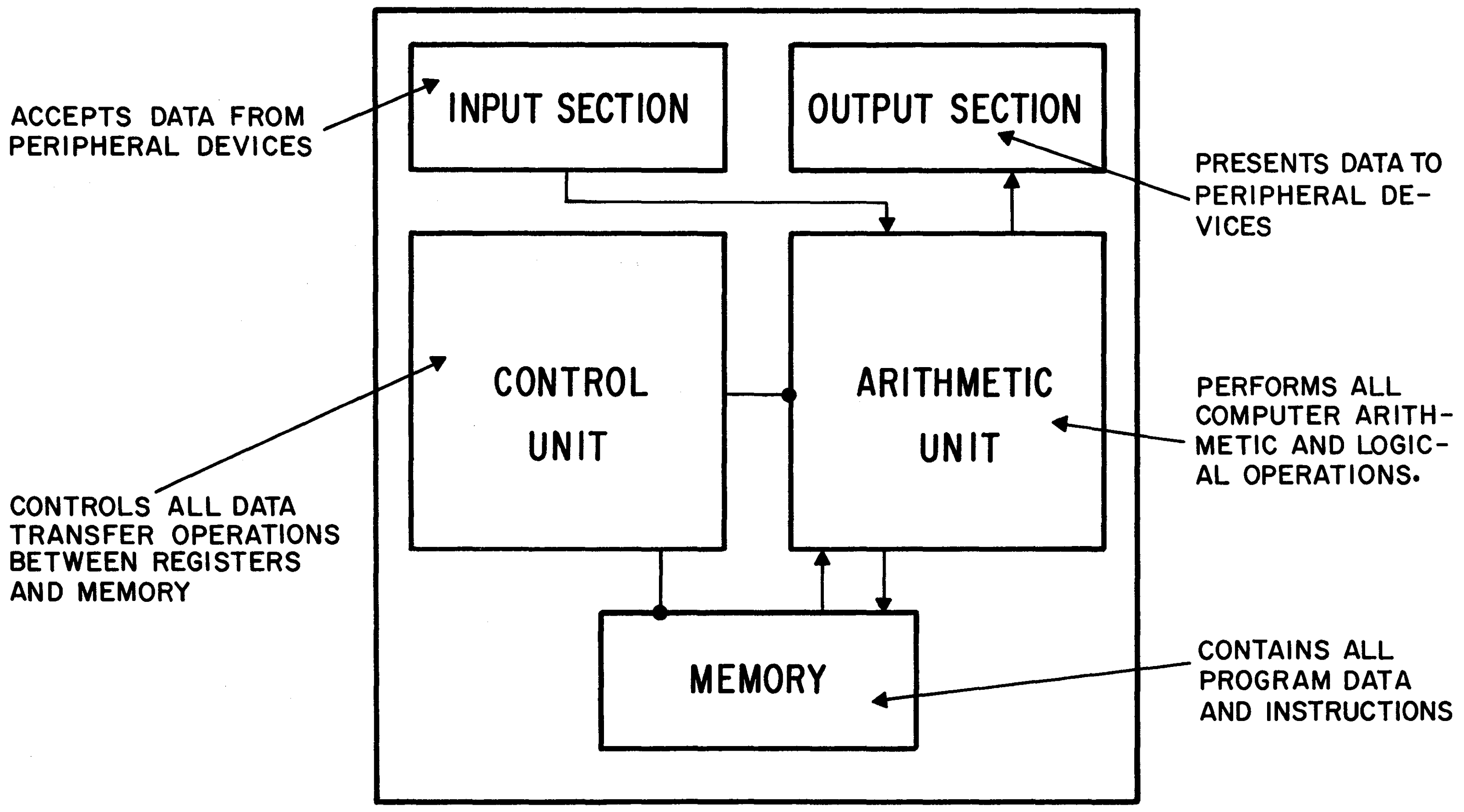
A DESK CALCULATOR IS CAPABLE OF PERFORMING COMPLICATED MATHEMATICAL PROCESSES SUCH AS DIFFERENTIATION AND INTEGRATION, HOWEVER, THE TIME REQUIRED TO SOLVE COMPLEX PROBLEMS USING THIS METHOD BECOMES PROHIBITIVELY LONG.

ADVANTAGES OF A STORED PROGRAM DIGITAL COMPUTER

1. *SPEED* – performs millions of operations in seconds.
2. *INTERCOMMUNICATIONS* – digital data can be received or transmitted by the computer.



COMPUTER BLOCK DIAGRAM


ACCEPTS DATA FROM PERIPHERAL DEVICES
INPUT SECTION
OUTPUT SECTION
PRESENTS DATA TO PERIPHERAL DE-VICES
CONTROL UNIT
ARITHMETIC UNIT
PERFORMS ALL COMPUTER ARITH-METIC AND LOGIC-AL OPERATIONS.
CONTROLS ALL DATA TRANSFER OPERATIONS BETWEEN REGISTERS AND MEMORY
MEMORY
CONTAINS ALL PROGRAM DATA AND INSTRUCTIONS

A BASIC COMPUTER

INTRODUCTION TO NUMBER SYSTEMS

HEWLETT-PACKARD computers operate on numbers in binary form; therefore, it is essential that we:

1. REVIEW THE DECIMAL NUMBER SYSTEM
2. INTRODUCE THE BINARY AND OCTAL NUMBER SYSTEMS
3. INTRODUCE BINARY ARITHMETIC
4. INTRODUCE NUMBER SYSTEM CONVERSION METHODS
5. DISCUSS THE LIMITS OF THE COMPUTER'S ABILITY TO HANDLE LARGE NUMBERS

NUMBER SYSTEMS

0,1,2,3,4,5,6,7,8,9 ARE THE TEN NUMERALS OF THE DECIMAL SYSTEM. DECIMAL VALUES LARGER THAN 9 REQUIRE MORE THAN ONE DIGIT. FOR EXAMPLE, THE DECIMAL NUMBER 109 REALLY STANDS FOR:

$$\underline{(1 \times 10^2) + (0 \times 10^1) + (9 \times 10^0)}$$

(HUNDRED'S) + (TEN'S) + (ONE'S)

$$\underline{(100) + (0) + (9)} = 109_{10}$$

IN GENERAL:

ANY NUMBER = $N \times b^n + N \times b^{n-1} + \cdots + N \times b^2 + N \times b^1 + N \times b^0$

WHERE N = DIGIT
b = BASE
$b^0 = 1$ (BY DEFINITION)

BINARY NUMBERS

Ø and 1 are the TWO numerals of the binary system. Binary values larger than 1 require more than one digit. For example, the BINARY number 1101101 really stands for:

$$(1 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

(SIXTY-FOUR'S) + (THIRTY-TWO'S) + (SIXTEEN'S) + (EIGHT'S) + (FOUR'S) + (TWO'S) + (ONE'S)

$$(64) + (32) + (0) + (8) + (4) + (0) + (1) = 109_{10}$$

THEREFORE:

$$110\,1101_2 = 109_{10}$$

OCTAL NUMBERS

0,1,2,3,4,5,6,7 ARE THE EIGHT NUMERALS OF THE OCTAL SYSTEM. OCTAL VALUES LARGER THAN 7 REQUIRE MORE THAN ONE DIGIT. FOR EXAMPLE, THE OCTAL NUMBER 155 REALLY STANDS FOR:

$$\underline{(1 \times 8^2) + (5 \times 8^1) + (5 \times 8^0)}$$

(SIXTY FOUR'S) + (EIGHT'S) + (ONE'S)

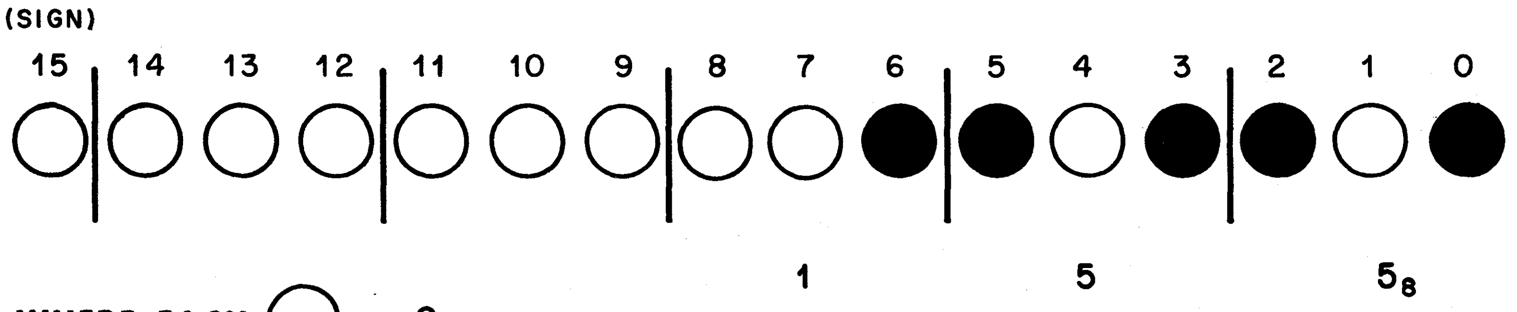
$$\underline{(\ 64\ ) + (\ 40\ ) + (\ 5\ )} = 109_{10}$$

THEREFORE —

$$\underline{1101101_2 = 155_8 = 109_{10}}$$

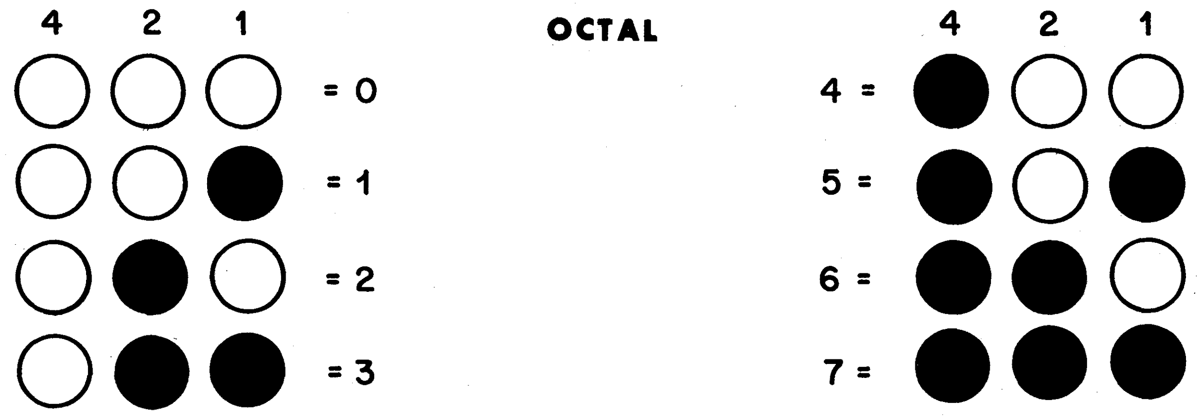
BINARY/OCTAL RELATIONSHIP

**HEWLETT-PACKARD COMPUTERS HAVE 16 BINARY DIGITS. (BIT)
WHEN BINARY DIGITS (BITS) ARE ARRANGED IN GROUPS OF 3, OCTAL VALUES CAN BE READ DIRECTLY.**



WHERE EACH ○ = 0

AND EACH ● = 1

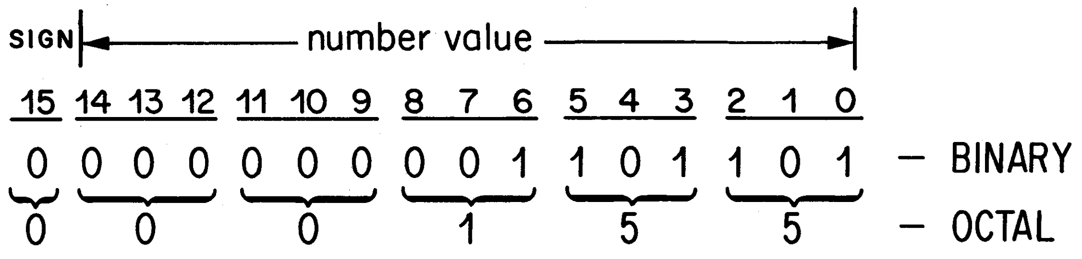


NUMBER SYSTEM CONVERSION METHODS

PROGRAMMERS MUST LEARN THE FOLLOWING NUMBER SYSTEM CONVERSION TECHNIQUES:

CONVERSION	METHOD
BINARY TO OCTAL	BY INSPECTION
OCTAL TO BINARY	BY INSPECTION
OCTAL TO DECIMAL	BY FORMULA
DECIMAL TO OCTAL	BY FORMULA

REMEMBER:
OCTAL IS USED TO REPRESENT BINARY NUMBERS MORE EFFICIENTLY



● TO CONVERT THE OCTAL NUMBER 155 TO DECIMAL, PROCEED IN THE FOLLOWING WAY.

1. **Multiply the most significant octal digit by 8**
2. **Add the next least significant octal digit, then multiply the result by 8**
3. **Continue using step 2 above until the least significant digit is reached**
4. **The least significant digit is added to the total but the result is NOT multiplied by 8.**

EXAMPLE:

CONVERT 155_8 TO DECIMAL

```
  155
 x8
   8
 +5
  13
  x8
 104
  +5
 ----
 109₁₀   DECIMAL RESULT
```

OCTAL TO DECIMAL CONVERSION

● TO CONVERT THE DECIMAL NUMBER 109 TO OCTAL PROCEED IN THE FOLLOWING WAY.

1. **Divide the decimal number by 8 and write down the remainder.** $8\,\dfrac{\angle 109}{13} + 5$ REMAINDER

2. **Divide the quotient of the previous step by 8 and write down the remainder.** $8\,\dfrac{\angle 13}{1} + 5$ REMAINDER

3. **Repeat step 2 until the "new" quotient becomes zero.** $8\,\dfrac{\angle 1}{0} + 1$ REMAINDER

   $\uparrow$ READ OCTAL VALUE

4. **The octal value is read as follows:**

   THE LAST REMAINDER IS THE <u>MOST</u> SIGNIFICANT OCTAL DIGIT.

   THE FIRST REMAINDER IS THE <u>LEAST</u> SIGNIFICANT OCTAL DIGIT.

   $109_{10} = 155_8$

DECIMAL TO OCTAL CONVERSION

BINARY ARITHMETIC

- *In the computer a special logic circuit performs addition using binary arithmetic. Actual computer numbers are 16 "BITS" long, however, for simplicity the following example uses only 6 "BITS."*

EXAMPLE

```
                GENERATED
                  CARRIES
          1  1
X       0 0 1 1 1 0
Y       0 0 1 1 0 1
       -------------
sum     0 1  1 0 1 1
```

RULES OF BINARY ADDITION

CARRY (IN)	X	Y
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

SUM	CARRY (OUT)
0	0
1	0
1	0
0	1
1	0
0	1
0	1
1	1

TWO'S COMPLEMENT NUMBERS

HEWLETT-PACKARD COMPUTERS USE THE TWO'S COMPLEMENT ARITHMETIC TECHNIQUE. THE PROCESS OF "TWO'S COMPLEMENTATION" CHANGES A POSITIVE INTEGER VALUE TO NEGATIVE AND VICE-VERSA.

NOTE: IF SIGN = 0, NORMAL FORM (POSITIVE)
IF SIGN = 1, TWO'S COMPLEMENT FORM (NEGATIVE)

FOR EXAMPLE:

SIGN	VALUE					
0	1	1	0	1	1	A NORMAL NUMBER (POSITIVE)
1	0	0	1	0	0	THE ONE'S COMPLEMENT $\left\{\begin{array}{l}\text{ALL 1's BECOME 0's}\\\text{ALL 0's BECOME 1's}\end{array}\right.$
0	0	0	0	0	1	ADD ONE
1	0	0	1	0	1	THE TWO'S COMPLEMENT (NEGATIVE)

COMPLEMENTATION TECHNIQUES

The decimal number 109_{10} when converted to octal appears as 155_8. The example shows the two's complement operation performed on this value.

EXAMPLE:

SIGN	BINARY		OCTAL	
0	000 000 001 101 101	(POSITIVE)	0	00155
1	111 111 110 010 010	(ONE'S COMPLEMENT)	1	77622
0	000 000 000 000 001	(ADD ONE)	0	00001
1	111 111 110 010 011	(NEGATIVE TWO'S COMPLEMENT)	1	77623

NOTE THE MOST SIGNIFICANT OCTAL DIGIT REPRESENTS A SINGLE BIT. TO COMPLEMENT WITH OCTAL NUMBERS REMEMBER —

1 — COMPLEMENT THE SIGN DIGIT. (1 or Ø)

2 — TAKE THE EIGHTS COMPLEMENT ON THE REMAINING DIGITS.

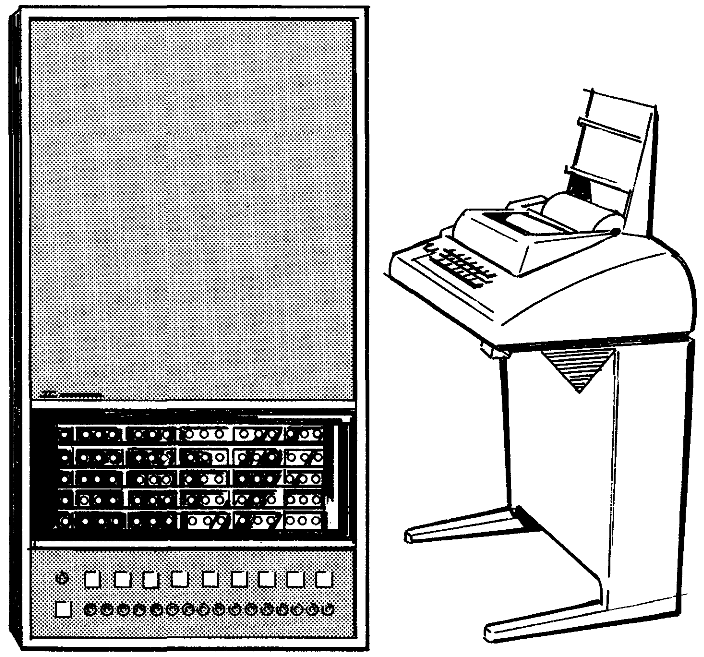
NEGATIVE NUMBER CONVERSIONS

TO CONVERT A NEGATIVE DECIMAL NUMBER TO 16 BIT MACHINE FORM.

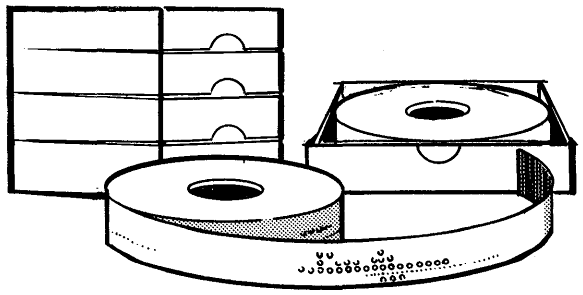
1. ASSUME THE DECIMAL VALUE IS POSITIVE
2. CONVERT TO OCTAL FORM
3. TAKE THE TWO'S COMPLEMENT. (OR EIGHT'S COMPLEMENT)

TO CONVERT TWO'S COMPLEMENT NUMBERS TO DECIMAL FORM.

1. TAKE THE TWO'S COMPLEMENT.
2. CONVERT TO DECIMAL
3. AFFIX A MINUS SIGN TO THE DECIMAL RESULT

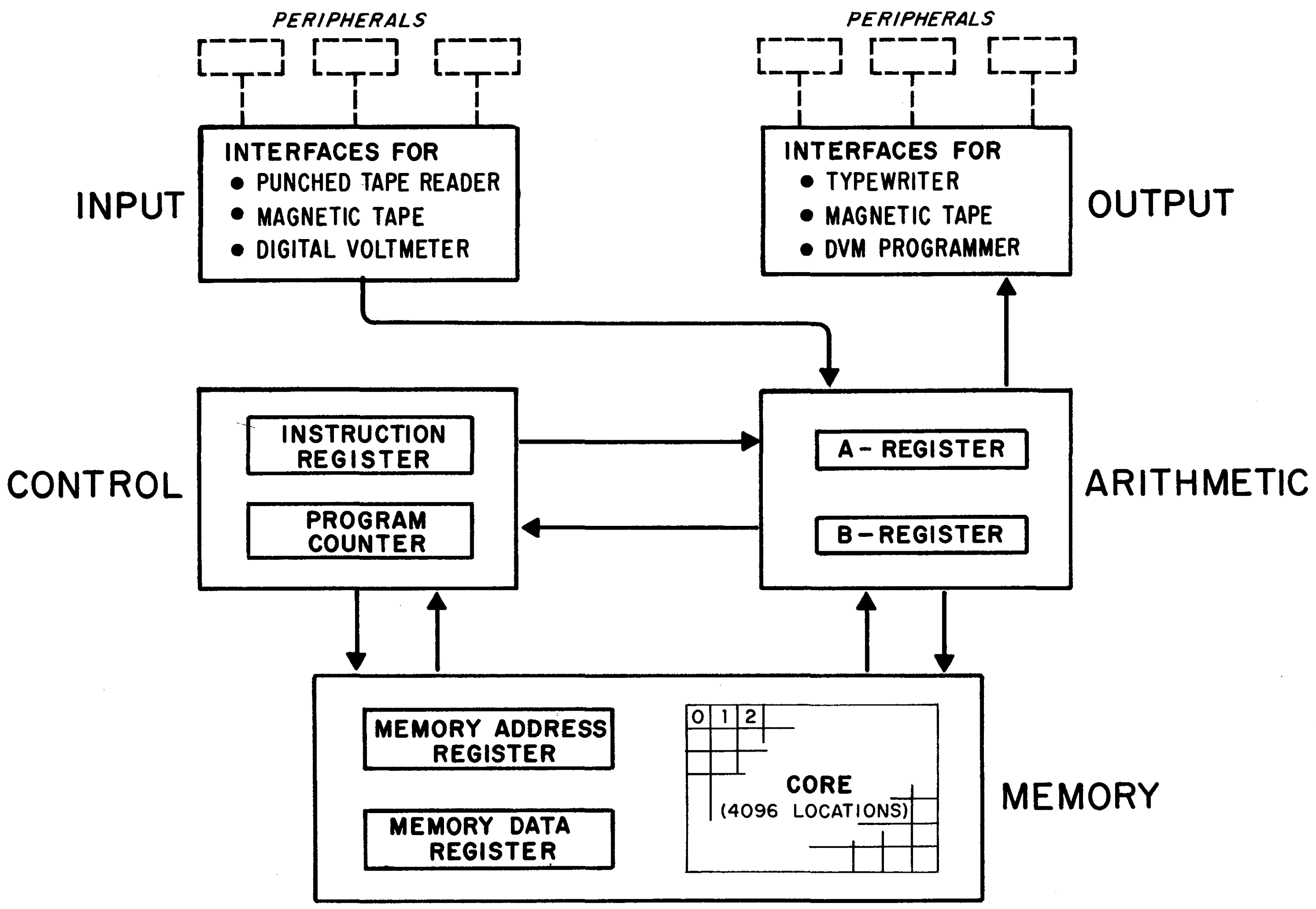


HARDWARE



SOFTWARE

A COMPUTER SYSTEM IS COMPOSED OF HARDWARE AND SOFTWARE



BASIC ELEMENTS OF COMPUTER HARDWARE

COMPUTER WORD FORMAT

DATA FORMAT (INTEGER)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIGN	INTEGER														

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

EXAMPLE: $+12357_8$

INSTRUCTION FORMAT (MEMORY REFERENCE INSTRUCTION)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	INSTRUCTION				Z/C	MEMORY WORD ADDRESS									

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

EXAMPLE: "LOAD REGISTER "A" WITH THE CONTENTS OF LOCATION 200_8"

FULL ADDRESS FORMAT

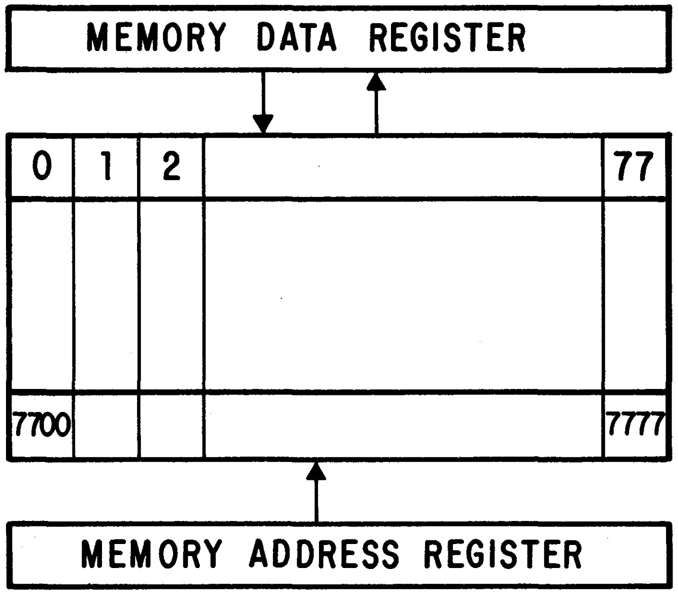
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	PAGE ADDRESS					MEMORY WORD ADDRESS									

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

EXAMPLE: MEMORY ADDRESS 17700_8

MEMORY

ADDRESSING & DATA TRANSFER CONCEPTS

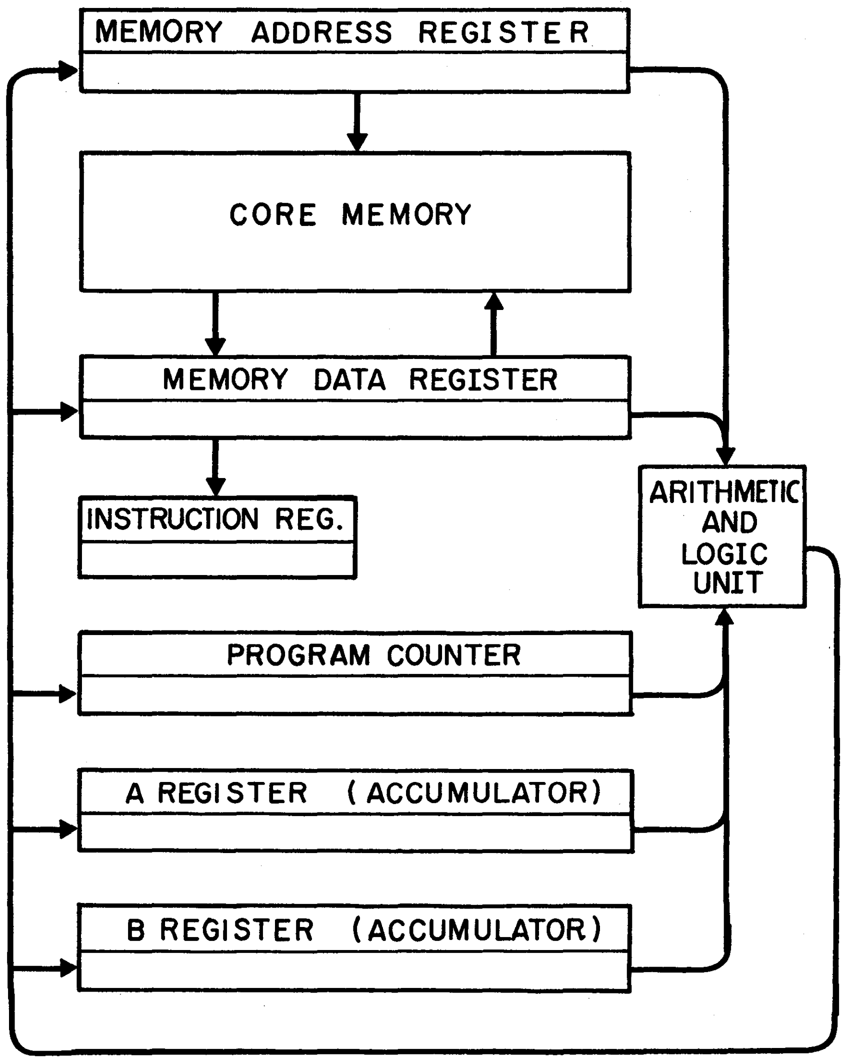


1.) ASSUME MEMORY ADDRESS 7700_8 CONTAINS 1001110000111100
$\longleftarrow$ 16 BITS $\longrightarrow$

2.) ALL DATA TO AND FROM THE MEMORY PASSES THROUGH THE MEMORY DATA REGISTER — READING WORD 7700_8 PUTS 1001110000111100 IN THE MEMORY DATA REGISTER.

3) MEMORY ADDRESS REGISTER — HOLDS THE NUMBER OF THE WORD ADDRESSED IN MEMORY — TO READ THE CONTENTS OF MEMORY ADDRESS 7700_8 THE NUMBER 7700_8 IS PLACED IN THE MEMORY ADDRESS REGISTER.

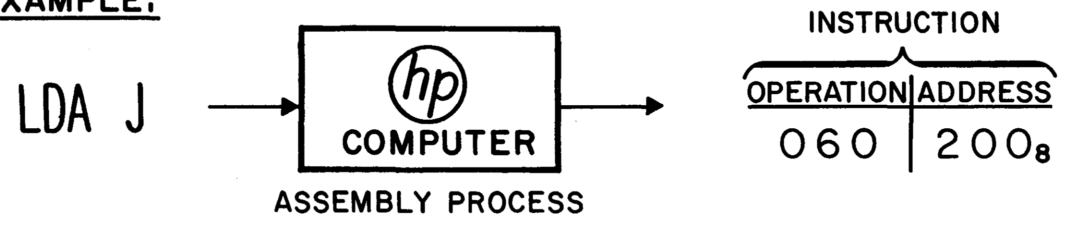
COMPUTER REGISTERS



COMPUTER INSTRUCTIONS

COMPUTER INSTRUCTIONS TAKE TWO BASIC FORMS. ONE FORM IS HUMAN ORIENTED, WHILE THE OTHER IS MACHINE ORIENTED. THE COMPUTER TRANSLATES FROM "MAN" TO "MACHINE" LANGUAGE.

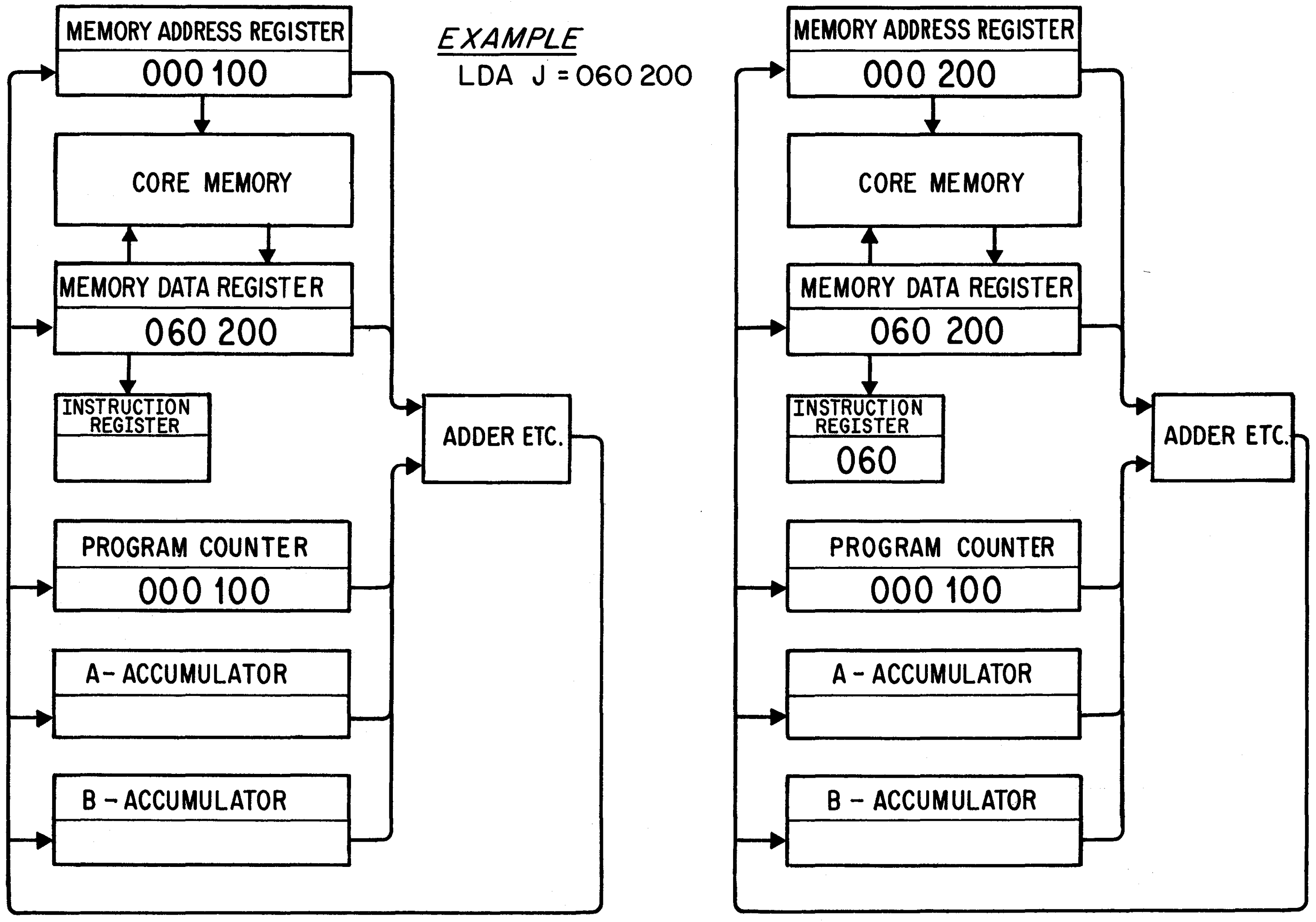
FOR EXAMPLE:



THE ABOVE INSTRUCTION MEANS; "LOAD REGISTER A WITH THE CONTENTS OF MEMORY LOCATION J". THE ASSEMBLY PROCESS CONVERTS "LDA J" TO THE MACHINE INSTRUCTION 060200_8.

NOTE: IN THIS EXAMPLE "J" IS ARBITRARILY REPRESENTING MEMORY LOCATION 200_8.

INSTRUCTION EXECUTION SEQUENCE



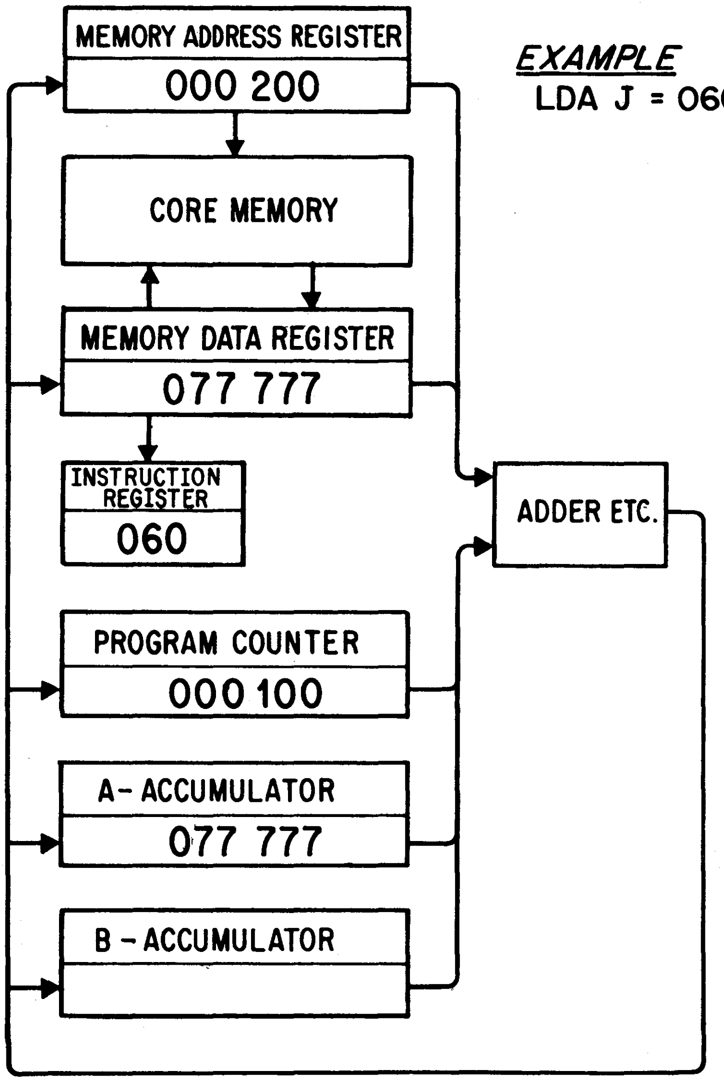
a. Instruction is Read from Memory

b. End of Fetch Phase

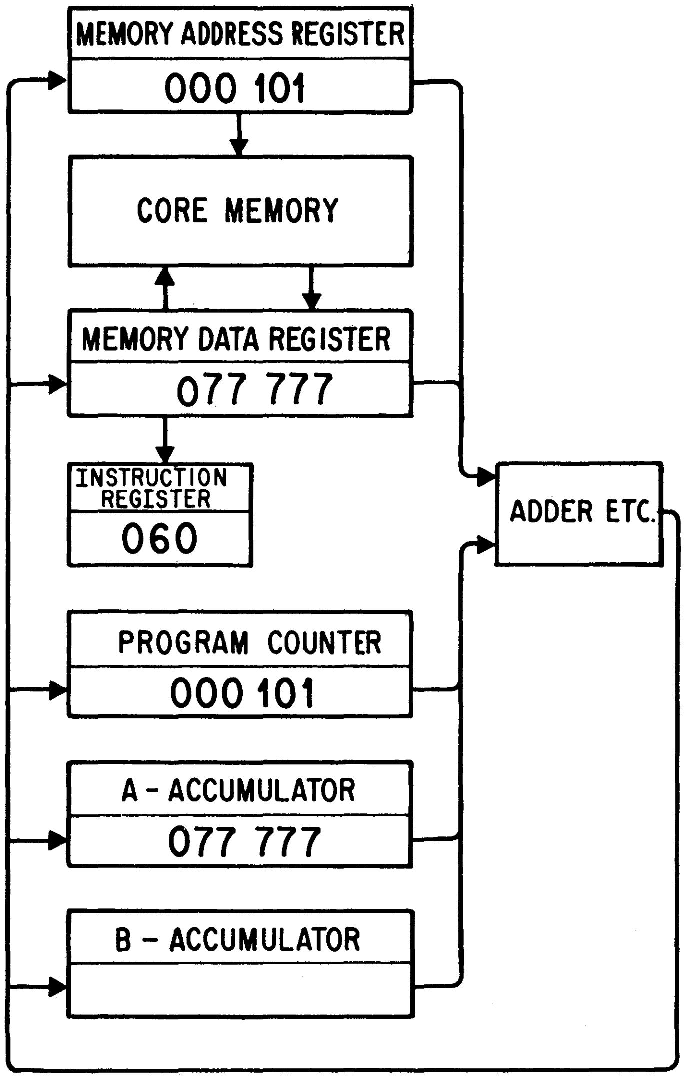
INSTRUCTION IN LOCATION 100 - FETCH PHASE

INSTRUCTION EXECUTION cont'd

EXAMPLE
LDA J = 060200



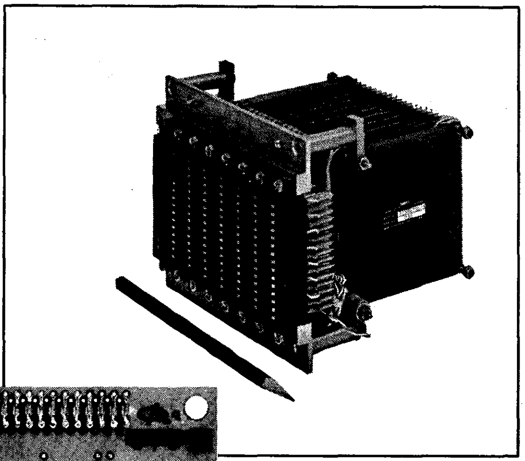
c. Instruction is Executed



d. End of Execute Phase

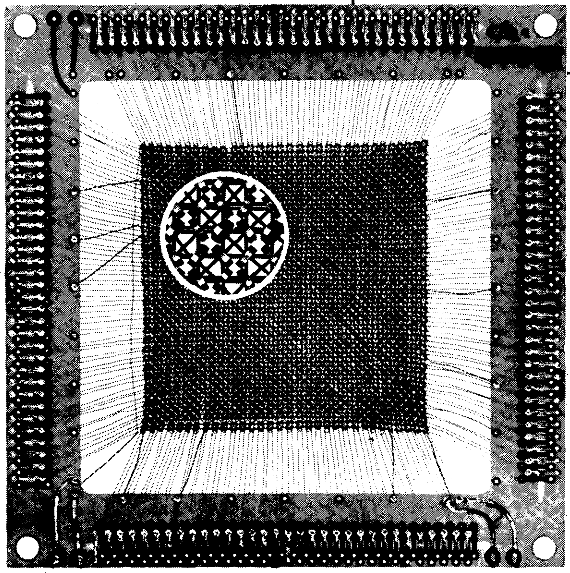
INSTRUCTION IN LOCATION 100 -EXECUTE PHASE

CORE MEMORY



4096 – Word
Core Module

17 CORE PLANES PER MODULE. EACH CORE PLANE SUPPLIES ONE BIT OF THE COMPUTER WORD. (16 DATA BITS + PARITY BIT).



Memory Plane

4096 CORES PER MEMORY PLANE. ONLY ONE CORE ON EACH PLANE IS INTERROGATED WHEN A MEMORY LOCATION IS ADDRESSED.

TYPES OF COMPUTER INSTRUCTIONS

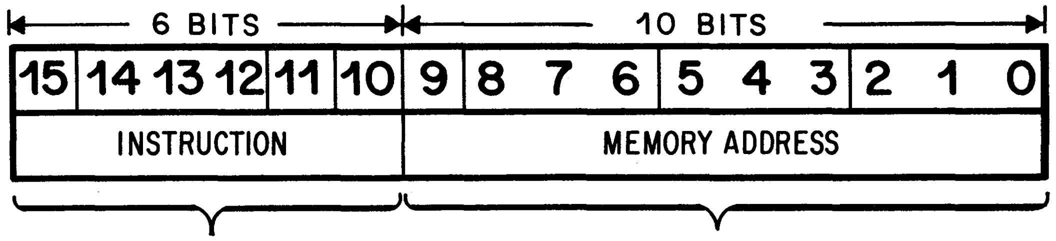
THERE ARE THREE TYPES OF COMPUTER INSTRUCTIONS —

- ✓ *Memory Reference*
- ✓ *Register Reference*
- ✓ *Input/output*

MEMORY REFERENCE INSTRUCTION

— USED FOR —

READING DATA FROM MEMORY
STORING DATA IN MEMORY
ARITHMETIC OPERATIONS
LOGIC OPERATIONS
ALTERATION OF PROGRAM COUNTER
CONTROLLING PROGRAM LOOPS



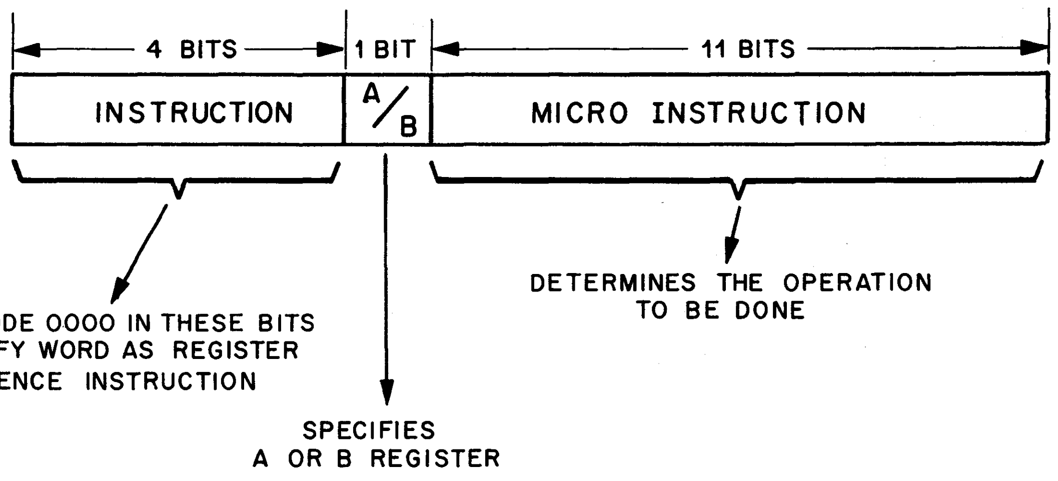
SELECTS 1 OF 14 INSTRUCTIONS
AND DETERMINES ADDRESSING MODE

SPECIFIES THE MEMORY WORD ADDRESS

REGISTER REFERENCE INSTRUCTIONS

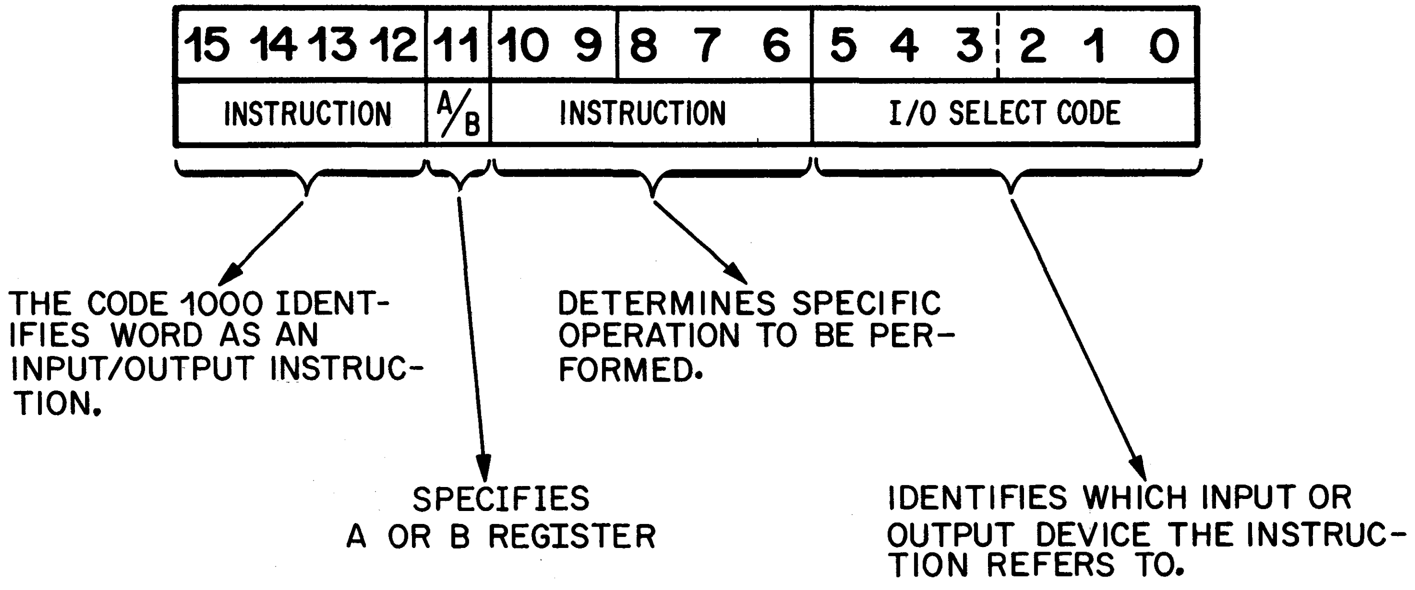
- MOVE DATA WITHIN AND BETWEEN ACCUMULATORS
- CLEAR OR COMPLEMENT ACCUMULATORS
- TEST BITS IN ACCUMULATORS

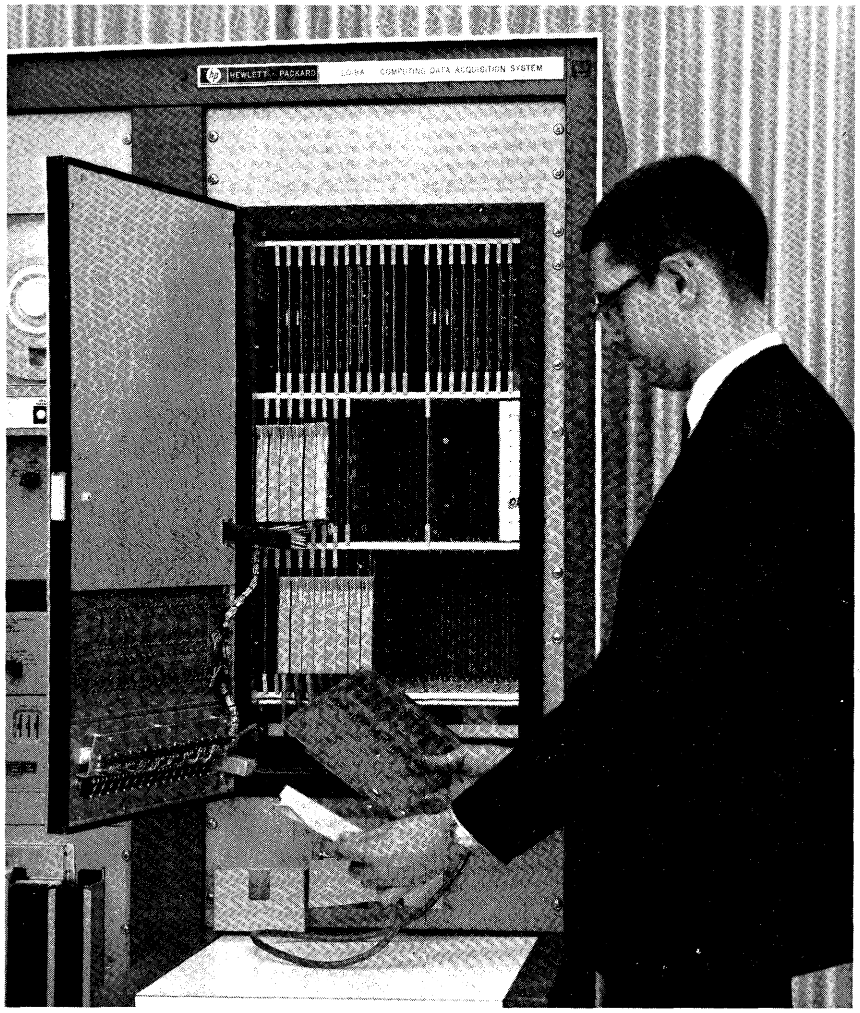
INSTRUCTION FORMAT


4 BITS
1 BIT
11 BITS
INSTRUCTION
A/B
MICRO INSTRUCTION
THE CODE 0000 IN THESE BITS IDENTIFY WORD AS REGISTER REFERENCE INSTRUCTION
SPECIFIES A OR B REGISTER
DETERMINES THE OPERATION TO BE DONE

INPUT/OUTPUT INSTRUCTIONS

READ DATA FROM DEVICES
OUTPUT DATA TO DEVICES
CHECK STATUS OF DEVICES


15 14 13 12
11
10 9
8 7 6
5 4 3
2 1 0
INSTRUCTION
A/B
INSTRUCTION
I/O SELECT CODE
THE CODE 1000 IDENTIFIES WORD AS AN INPUT/OUTPUT INSTRUCTION.
SPECIFIES A OR B REGISTER
DETERMINES SPECIFIC OPERATION TO BE PERFORMED.
IDENTIFIES WHICH INPUT OR OUTPUT DEVICE THE INSTRUCTION REFERS TO.



HP-INTERFACE CARD

I/O Interface cards are simple to install or rearrange

PRIORITY INTERRUPT SYSTEM

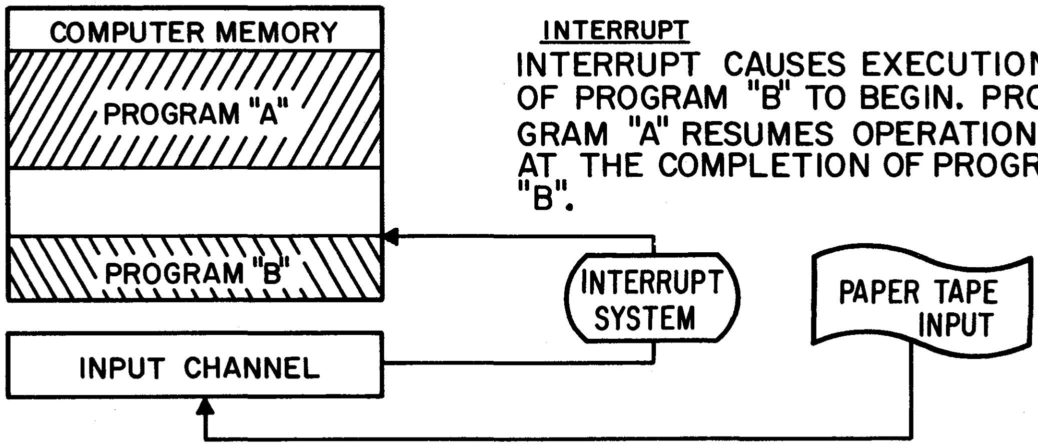
THE INTERRUPT SYSTEM ALLOWS THE COMPUTER TO PERFORM USEFUL WORK WHILE A PERIPHERAL DEVICE IS COMPLETING A CYCLE. EACH INTERRUPTING DEVICE IS UNIQUELY IDENTIFIED AND ASSIGNED A PRIORITY TO PREVENT SIMULTANEOUS INTERRUPT REQUESTS FROM MORE THAN ONE DEVICE.

FOR EXAMPLE:

ASSUME PROGRAM "A" IS A LARGE INTEGRATION ROUTINE, AND PROGRAM "B" IS A ROUTINE THAT HANDLES PAPER TAPE INPUT DATA.

NORMAL OPERATION

PROGRAM "A" HAS CONTROL AND IS IN EXECUTION.



INTERRUPT

INTERRUPT CAUSES EXECUTION OF PROGRAM "B" TO BEGIN. PROGRAM "A" RESUMES OPERATION AT THE COMPLETION OF PROGRAM "B".

INPUT/OUTPUT DATA TRANSFERS

Data transfers that do not use the interrupt system are made under program control. The controlling program must cause the computer to "WAIT" for the slower peripheral device. The steps in a non-interrupt data input program are:

1 - TURN THE INTERRUPT SYSTEM OFF.
2 - START THE DEVICE AND TURN THE READY FLAG OFF.
3 - WAIT FOR THE DEVICE READY FLAG TO COME ON.(WAIT LOOP)
4 - WHEN THE FLAG COMES ON, TRANSFER DATA TO COMPUTER.
5 - HAS ALL THE DATA BEEN TRANSFERRED?

NO, GO TO STEP 2
YES, GO TO STEP 6

6 - HALT THE COMPUTER

NOTE: THE COMPUTER WILL SPEND THE MOST TIME ON STEP 3

COMPUTER SOFTWARE

SOFTWARE IS THE GENERAL TERM GIVEN TO ALL PROGRAMS AND ROUTINES THAT EXTEND THE CAPABILITY OF THE COMPUTER. SOFTWARE CAN BE DIVIDED LOOSELY INTO FOUR CLASSES:

1. <u>TRANSLATORS</u> — PROGRAMS WHICH TRANSLATE HUMAN-ORIENTED LANGUAGES INTO MACHINE LANGUAGES.
2. <u>CONTROL SYSTEMS</u> — PROGRAMS WHICH TAKE CARE OF ALL FUNCTIONS ESSENTIAL TO OPERATION OF THE COMPUTER SYSTEM.
3. <u>UTILITY ROUTINES</u> — PROGRAM EDITORS, PROGRAM DEBUGGING ROUTINES, HARDWARE DIAGNOSTICS.

THE ABOVE SOFTWARE IS NORMALLY SUPPLIED BY THE COMPUTER MANUFACTURER.

4. <u>APPLICATIONS PROGRAMS</u> — THESE ENABLE THE COMPUTER TO BE EFFECTIVE IN A SPECIFIC APPLICATION.

APPLICATIONS PROGRAMS ARE NORMALLY CREATED BY THE USER.

HEWLETT-PACKARD SOFTWARE

TRANSLATION PROGRAMS

FORTRAN, ALGOL and "BASIC" COMPILERS
HP SYMBOLIC ASSEMBLER

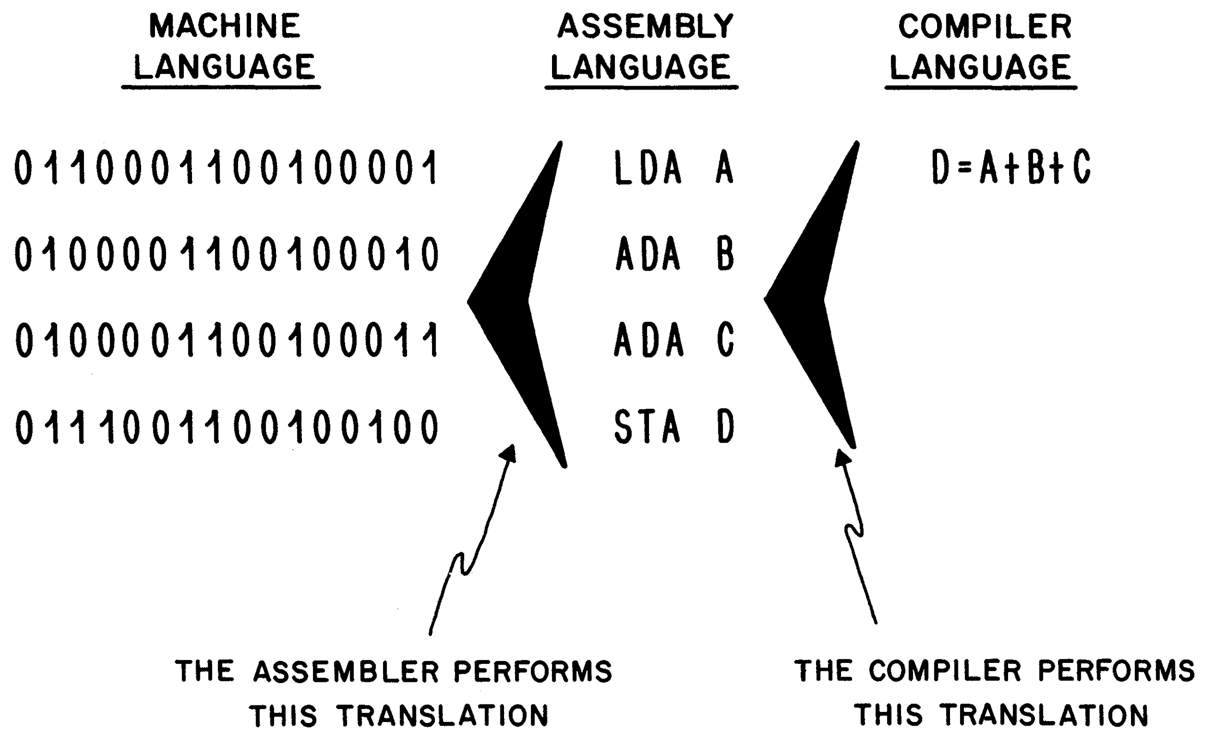
CONTROL SYSTEM

BASIC CONTROL SYSTEM

UTILITY ROUTINES

SYMBOLIC EDITOR
LIBRARY ROUTINES
DEBUGGING ROUTINE
PREPARE CONTROL SYSTEM
HARDWARE DIAGNOSTICS
PREPARE TAPE SYSTEM
SYSTEM INPUT OUTPUT DUMP

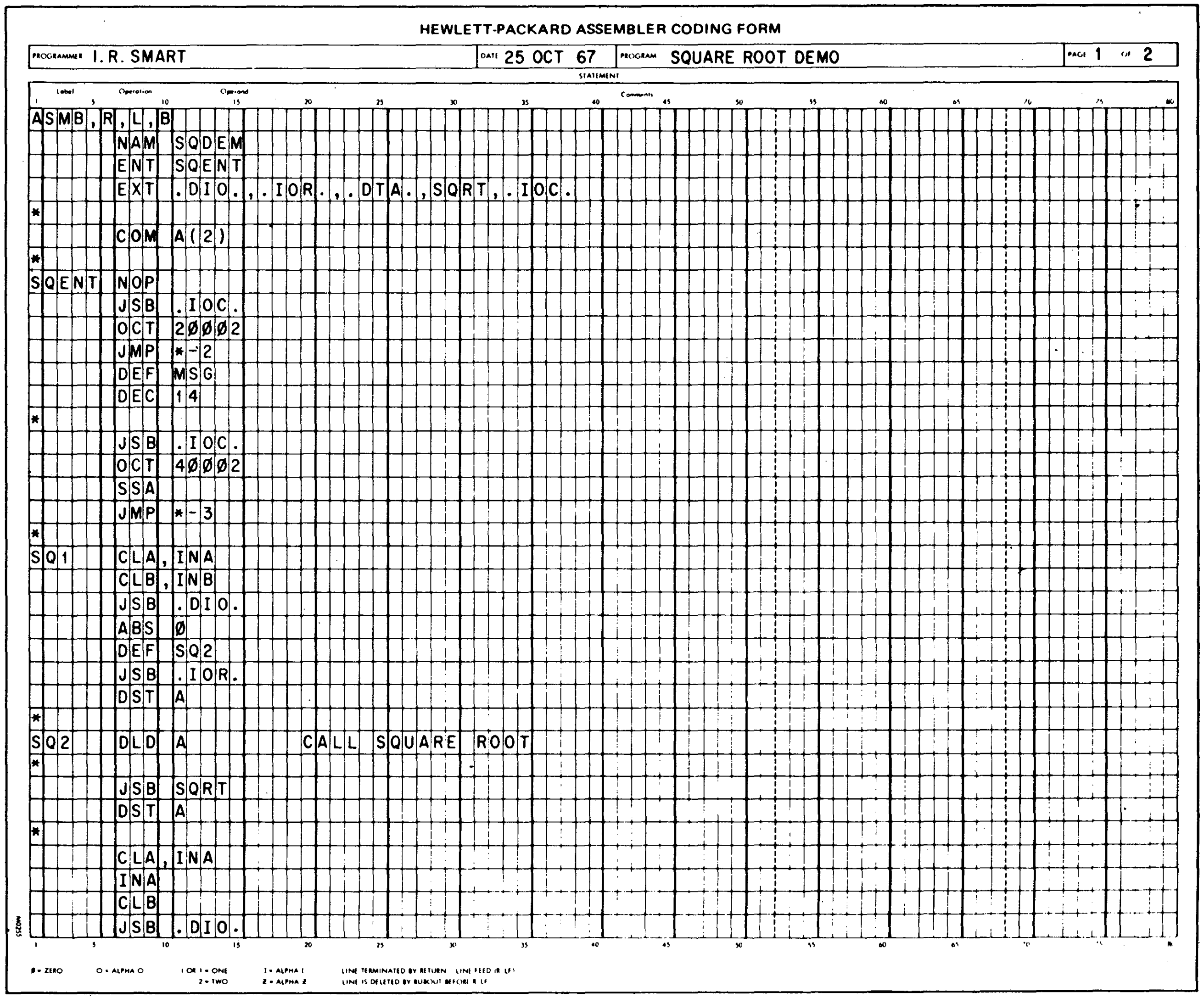
PROGRAMMING LANGUAGES


MACHINE LANGUAGE
ASSEMBLY LANGUAGE
COMPILER LANGUAGE

```
0110001100100001      LDA  A      D=A+B+C
0100001100100010      ADA  B
0100001100100011      ADA  C
0111001100100100      STA  D
```

THE ASSEMBLER PERFORMS THIS TRANSLATION
THE COMPILER PERFORMS THIS TRANSLATION

SOURCE PROGRAM IN ASSEMBLY LANGUAGE



HEWLETT-PACKARD ASSEMBLER CODING FORM

PROGRAMMER I. R. SMART | DATE 25 OCT 67 | PROGRAM SQUARE ROOT DEMO | PAGE 1 OF 2

STATEMENT

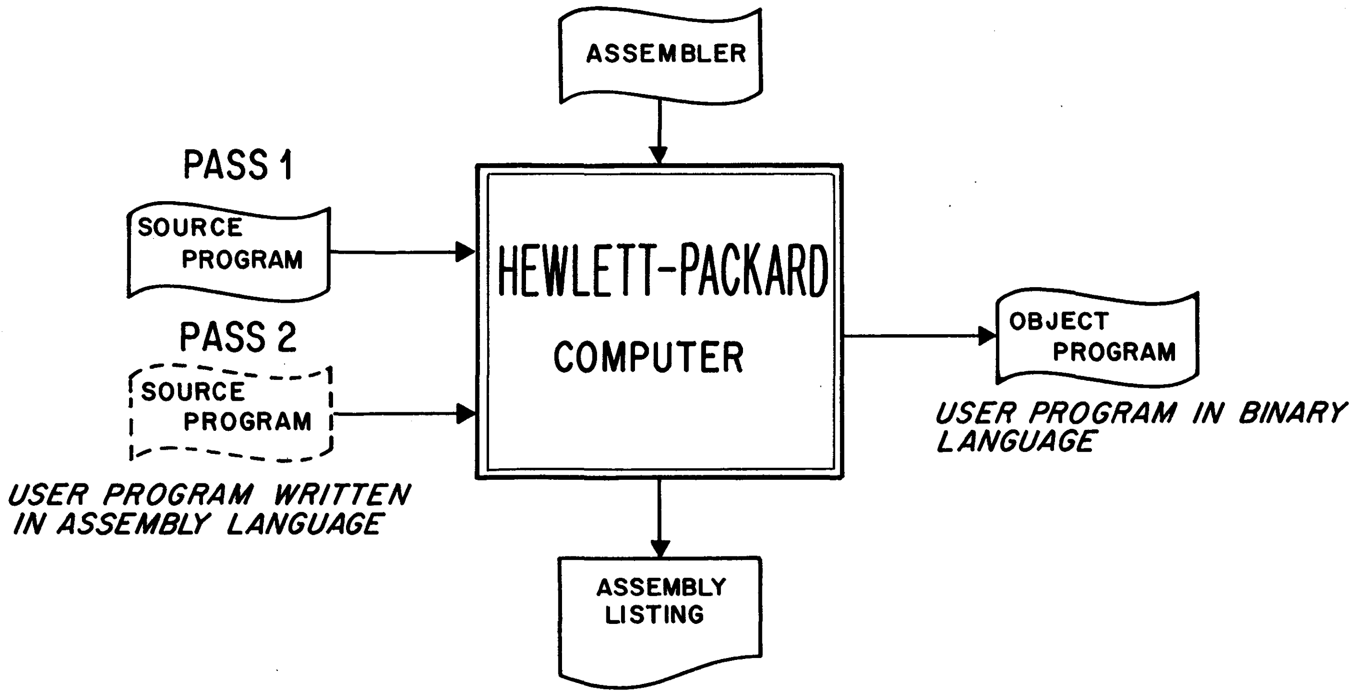
Label | Operation | Operand | Comments

```
ASMB,R,L,B
      NAM SQDEM
      ENT SQENT
      EXT .DIO.,.IOR.,.DTA.,SQRT,.IOC.
*
      COM A(2)
*
SQENT NOP
      JSB .IOC.
      OCT 2ØØØ2
      JMP *-2
      DEF MSG
      DEC 14
*
      JSB .IOC.
      OCT 4ØØØ2
      SSA
      JMP *-3
*
SQ1   CLA,INA
      CLB,INB
      JSB .DIO.
      ABS Ø
      DEF SQ2
      JSB .IOR.
      DST A
*
SQ2   DLD A        CALL SQUARE ROOT
*
      JSB SQRT
      DST A
*
      CLA,INA
      INA
      CLB
      JSB .DIO.
```

Ø = ZERO   O = ALPHA O   1 OR I = ONE   I = ALPHA I   LINE TERMINATED BY RETURN LINE FEED (R LF)
2 = TWO   Z = ALPHA Z   LINE IS DELETED BY RUBOUT BEFORE R LF

M0255

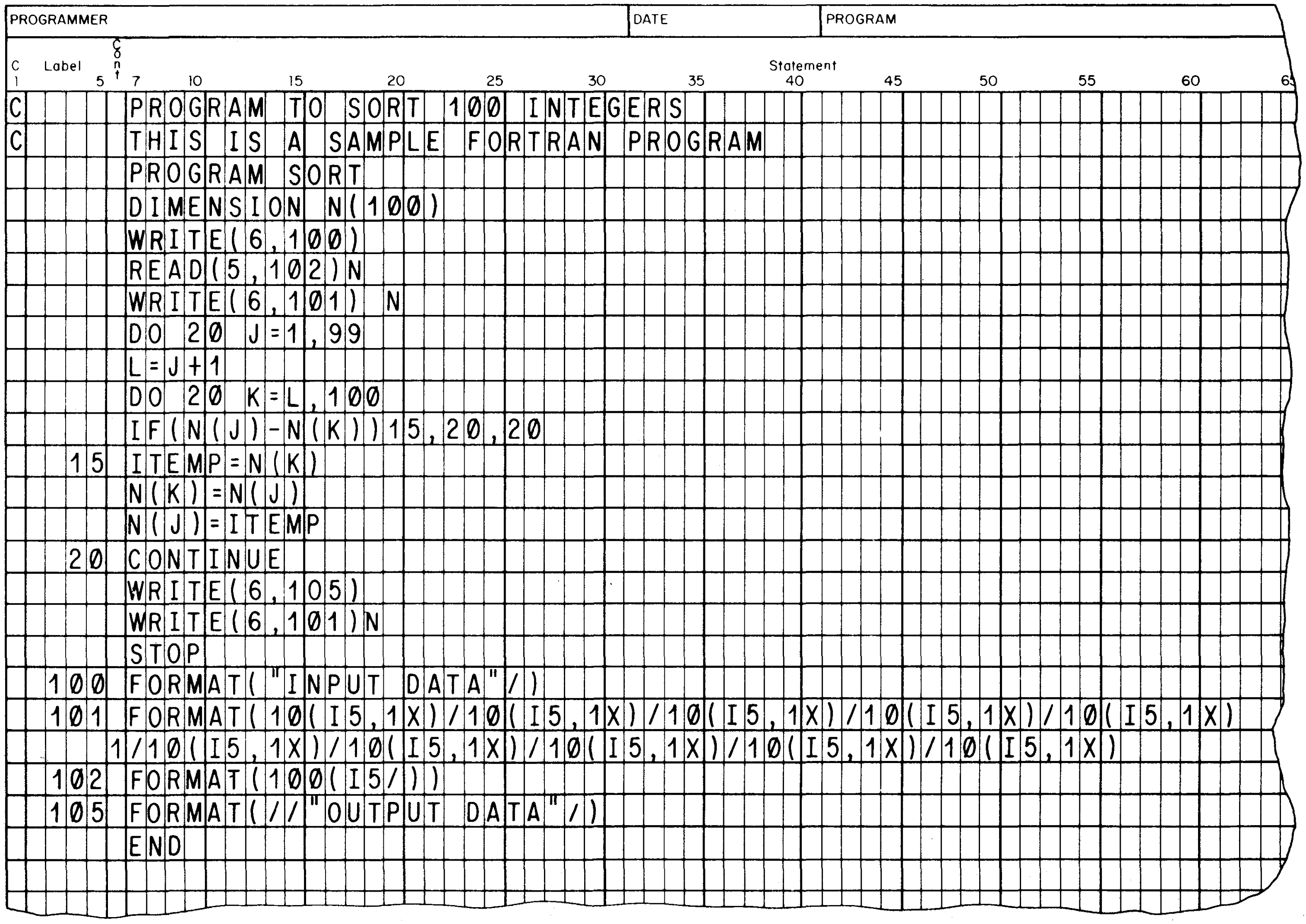
ASSEMBLY PROCESS



1. ASSEMBLER PROGRAM IS LOADED INTO THE COMPUTER.
2. SOURCE PROGRAM IS PROCESSED BY THE ASSEMBLER, PRODUCING THE OBJECT PROGRAM TAPE AND THE ASSEMBLY LISTING IN A TWO PASS OPERATION.

ASSEMBLY LISTING

```
 A      B      C                         D
0001                 ASMB,R,L,B
0002  00000                 NAM SQDEM
0003                        ENT SQFNT
0004                        EXT .DIO.,.IOR.,.DTA.,SQRT,.IOC.
0005*
0006                        COM A(2)
0007*
0008  00000 000000   SQENT  NOP
0009  00001 016005X         JSB .IOC.
0010  00002 020002          OCT 20002
0011  00003 026001R         JMP *-2
0012  00004 000045R         DEF MSG
0013  00005 000016          DEC 14
0014*
0015  00006 016005X         JSB .IOC.
0016  00007 040002          OCT 40002
0017  00010 002020          SSA
0018  00011 026006R         JMP *-3
0019*
0020  00012 002404   SQ1    CLA,INA
0021  00013 006404          CLB,INB
0022  00014 016001X         JSB .DIO.
0023  00015 000000          ABS 0
0024  00016 000022R         DEF SQ2
0025  00017 016002X         JSB .IOR.
0026  00020 016006X         DST A
      00021 000000C
0027*
0028  00022 016007X  SQ2    DLD A           CALL SQUARE ROOT
      00023 000000C
0029  00024 016004X         JSB SQRT
0030  00025 016006X         DST A
      00026 000000C
0031  00027 002404          CLA,INA
0032  00030 002004          INA
0033  00031 006400          CLB
0034  00032 016001X         JSB .DIO.
```



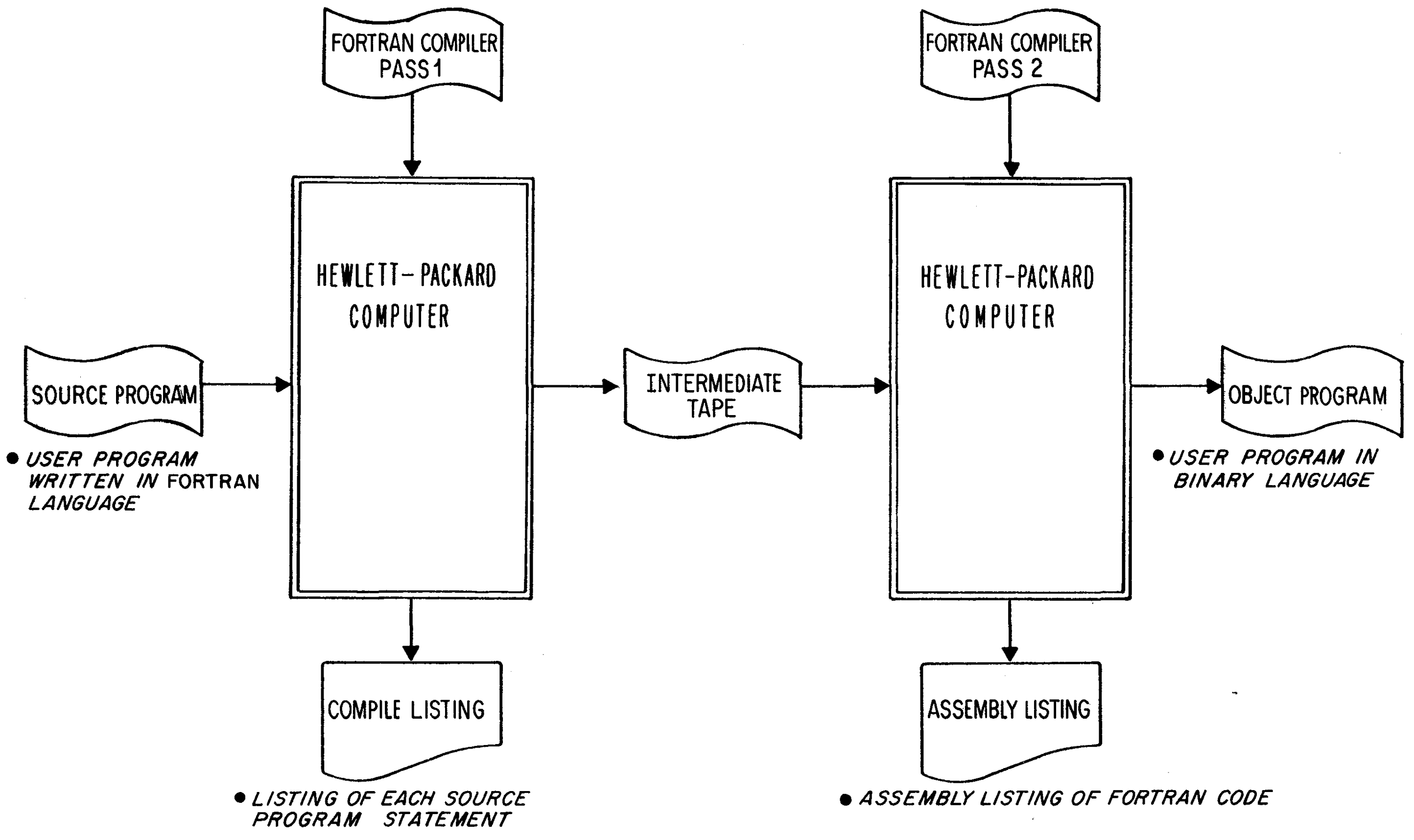
PROGRAMMER | DATE | PROGRAM

C Label Cont Statement

```
C     PROGRAM TO SORT 1ØØ INTEGERS
C     THIS IS A SAMPLE FORTRAN PROGRAM
      PROGRAM SORT
      DIMENSION N(1ØØ)
      WRITE(6,1ØØ)
      READ(5,1Ø2)N
      WRITE(6,1Ø1) N
      DO 2Ø J=1,99
      L=J+1
      DO 2Ø K=L,1ØØ
      IF(N(J)-N(K))15,2Ø,2Ø
   15 ITEMP=N(K)
      N(K)=N(J)
      N(J)=ITEMP
   2Ø CONTINUE
      WRITE(6,105)
      WRITE(6,1Ø1)N
      STOP
  1ØØ FORMAT("INPUT DATA"/)
  1Ø1 FORMAT(1Ø(I5,1X)/1Ø(I5,1X)/1Ø(I5,1X)/1Ø(I5,1X)/1Ø(I5,1X)
     1/1Ø(I5,1X)/1Ø(I5,1X)/1Ø(I5,1X)/1Ø(I5,1X)/1Ø(I5,1X))
  1Ø2 FORMAT(1ØØ(I5/))
  1Ø5 FORMAT(//"OUTPUT DATA"/)
      END
```

SOURCE PROGRAM IN FORTRAN

FORTRAN COMPILATION PROCESS



1 – FORTRAN COMPILER PASS 1 IS LOADED INTO THE COMPUTER

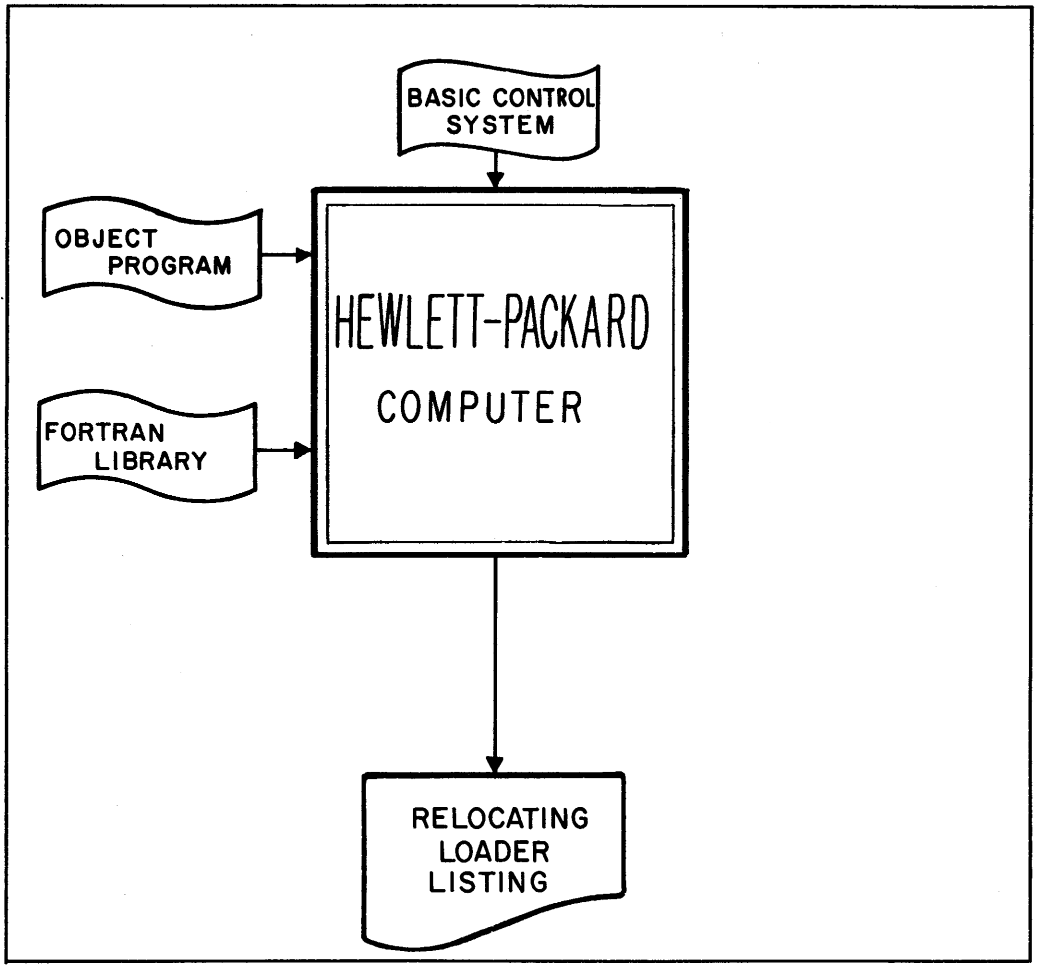
2 – SOURCE PROGRAM TAPE IS PROCESSED, BY THE COMPILER, PRODUCING THE INTERMEDIATE TAPE AND THE COMPILE LISTING.

3 – FORTRAN COMPILER PASS 2 IS LOADED INTO THE COMPUTER

4 – INTERMEDIATE TAPE IS PROCESSED, PRODUCING THE OBJECT PROGRAM TAPE & THE ASSEMBLY LISTING

USING THE BASIC CONTROL SYSTEM

THE B.C.S. IS USED TO LOAD OBJECT PROGRAMS PRODUCED BY THE FORTRAN COMPILER AND THE SYMBOLIC ASSEMBLER.



1. LOAD THE B.C.S. TAPE INTO THE COMPUTER.

2. PROCESS (LOAD) THE OBJECT PROGRAM TAPE.

3. PROCESS (LOAD) THE REQUIRED LIBRARY ROUTINES.

NOTE: THE BASIC CONTROL SYSTEM ALSO CONTAINS SUBROUTINES THAT ARE USED TO CONTROL THE INPUT/OUTPUT EQUIPMENT.

UTILITY PROGRAMS

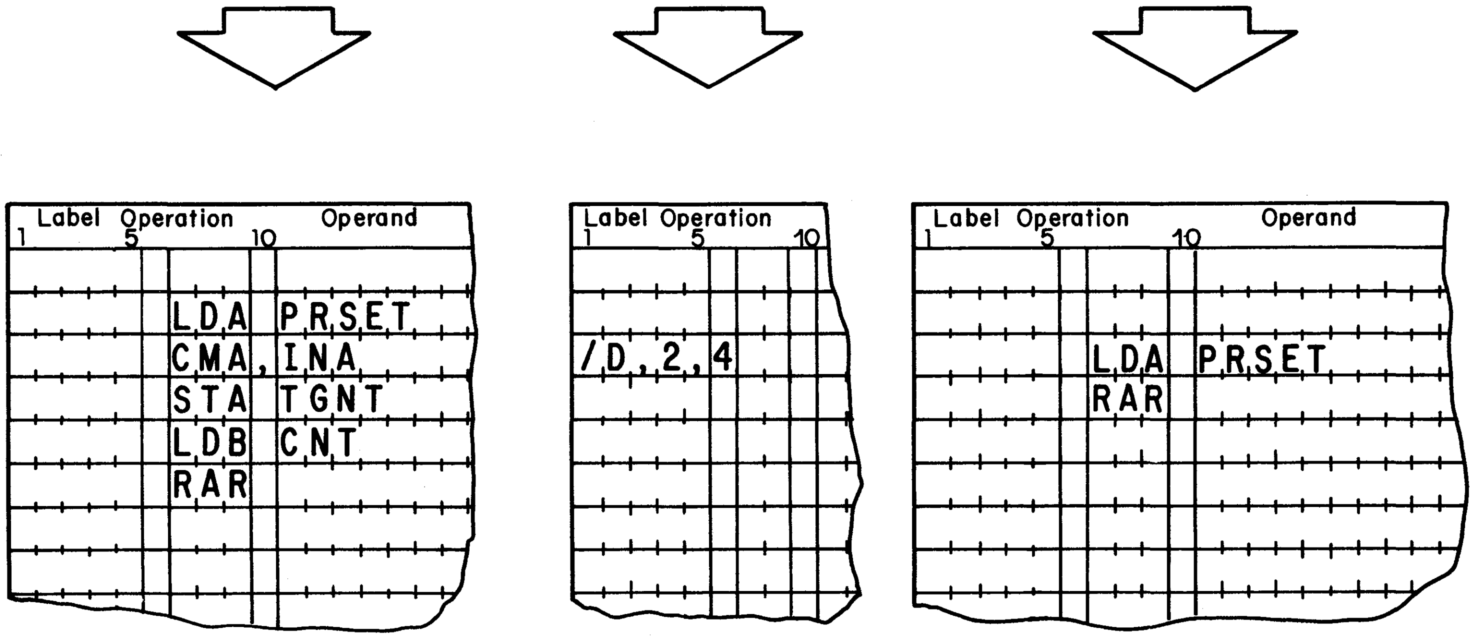
NAME		FUNCTION
FORTRAN-LIBRARY	–	Used primarily with compiler object programs. Standard mathematical subroutines for evaluating SIN, COSINE, SQUARE ROOT and other functions are found in the library.
PREPARE CONTROL SYSTEM	–	Used to create a BASIC CONTROL SYSTEM tailored to a specific hardware configuration.
HARDWARE DIAGNOSTICS	–	Used primarily in hardware maintenance to check the operation of the computer or peripheral equipment.
SYSTEM INPUT OUTPUT DUMP	–	Used to provide input-output flexibility for all HEWLETT-PACKARD standard software systems.
PREPARE TAPE SYSTEM	–	Used to create a magnetic tape operating system.
SYMBOLIC EDITOR	_	Used to make insertions, deletions, or replacements in source language program tapes.

EXAMPLE OF PROGRAM EDITING

ORIGINAL SOURCE PROGRAM

EDIT FILE, SHOWING CODING NEEDED TO DELETE STATEMENTS 2 THROUGH 4 FROM SOURCE PROGRAM

NEW SYMBOLIC FILE (SOURCE PROGRAM) PRODUCED BY EDITOR

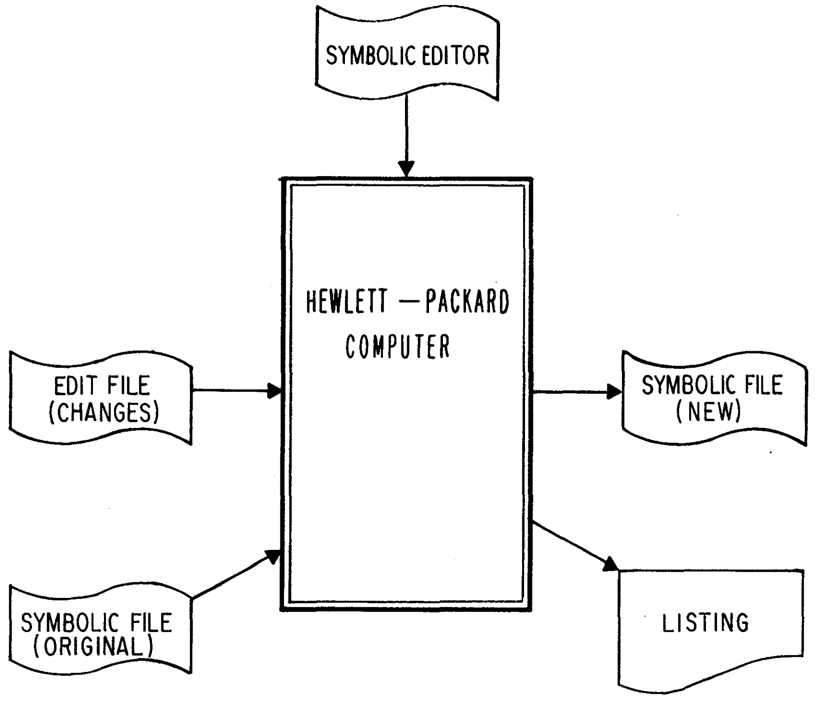


```
Label Operation    Operand
1       5       10
        LDA     PRSET
        CMA,INA
        STA     TGNT
        LDB     CNT
        RAR
```

```
Label Operation
1       5       10
/D,2,4
```

```
Label Operation    Operand
1       5       10
        LDA     PRSET
        RAR
```

EDITING PROCESS



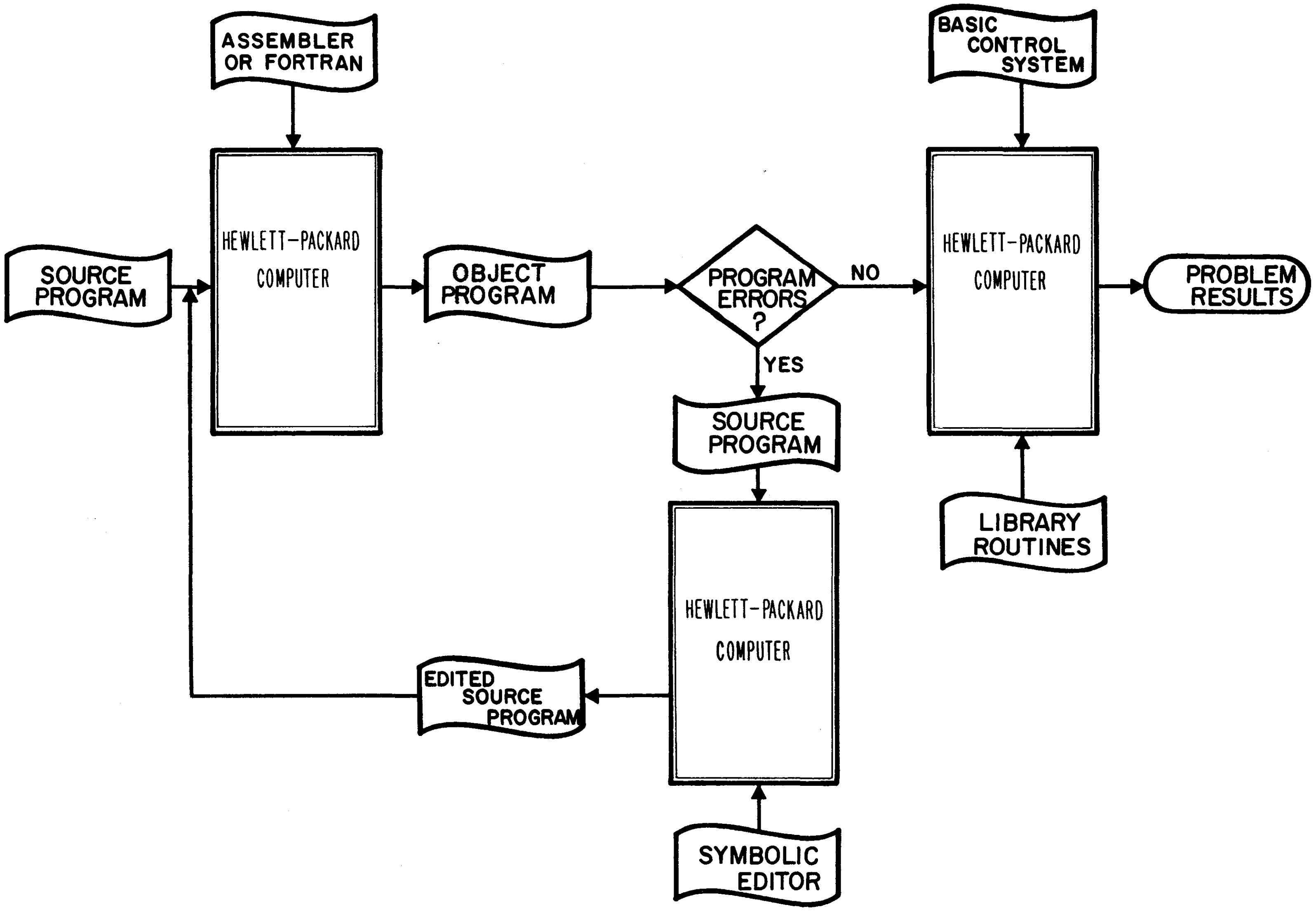
1 - SYMBOLIC EDITOR PROGRAM IS LOADED INTO THE COMPUTER

2 - THE EDIT FILE IS LOADED INTO THE COMPUTER

3 - THE ORIGINAL SYMBOLIC FILE IS PROCESSED, PRODUCING A NEW SYMBOLIC FILE

NOTE: LISTING A SYMBOLIC FILE REQUIRES A SECOND PASS

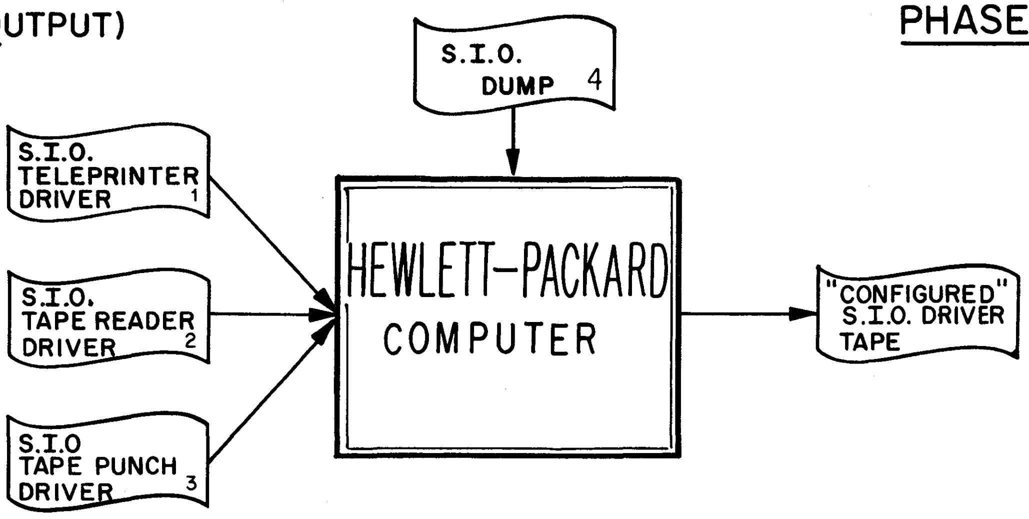
FORTRAN/ASSEMBLER PROGRAMMING ENVIRONMENT



S.I.O. CONFIGURATION PROCESS

(SYSTEM INPUT OUTPUT)

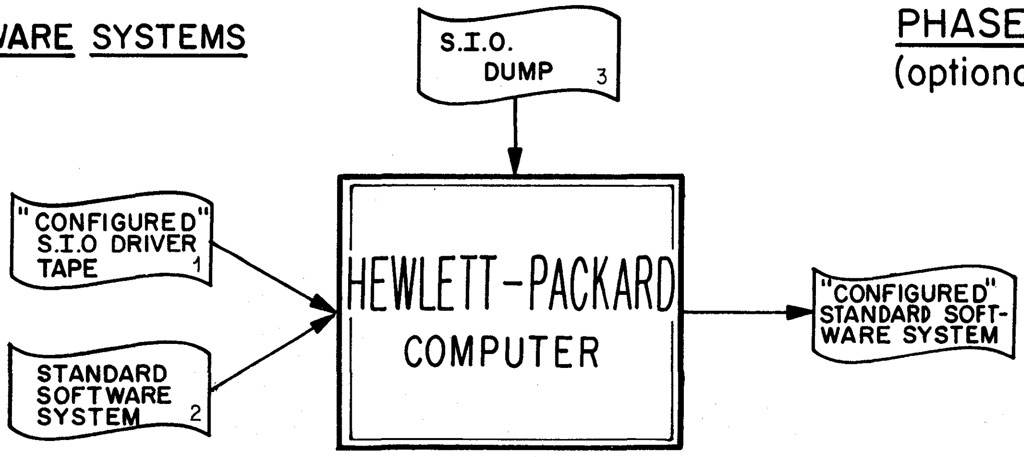
PHASE 1


S.I.O. DUMP 4
S.I.O. TELEPRINTER DRIVER 1
S.I.O. TAPE READER DRIVER 2
S.I.O TAPE PUNCH DRIVER 3
HEWLETT-PACKARD COMPUTER
"CONFIGURED" S.I.O. DRIVER TAPE

STANDARD SOFTWARE SYSTEMS

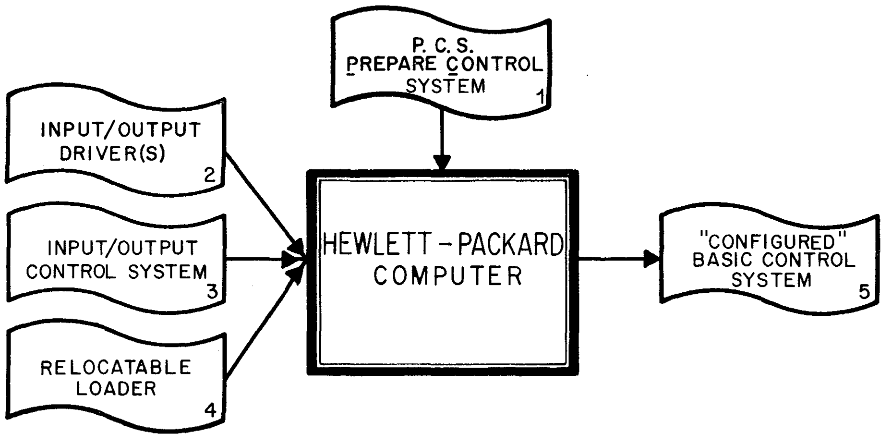
FORTRAN
ALGOL
ASSEMBLER
SYMBOLIC EDITOR

PHASE 2
(optional)


S.I.O. DUMP 3
"CONFIGURED" S.I.O DRIVER TAPE 1
STANDARD SOFTWARE SYSTEM 2
HEWLETT-PACKARD COMPUTER
"CONFIGURED" STANDARD SOFTWARE SYSTEM

B.C. S. CONFIGURATION PROCESS

(BASIC CONTROL SYSTEM)

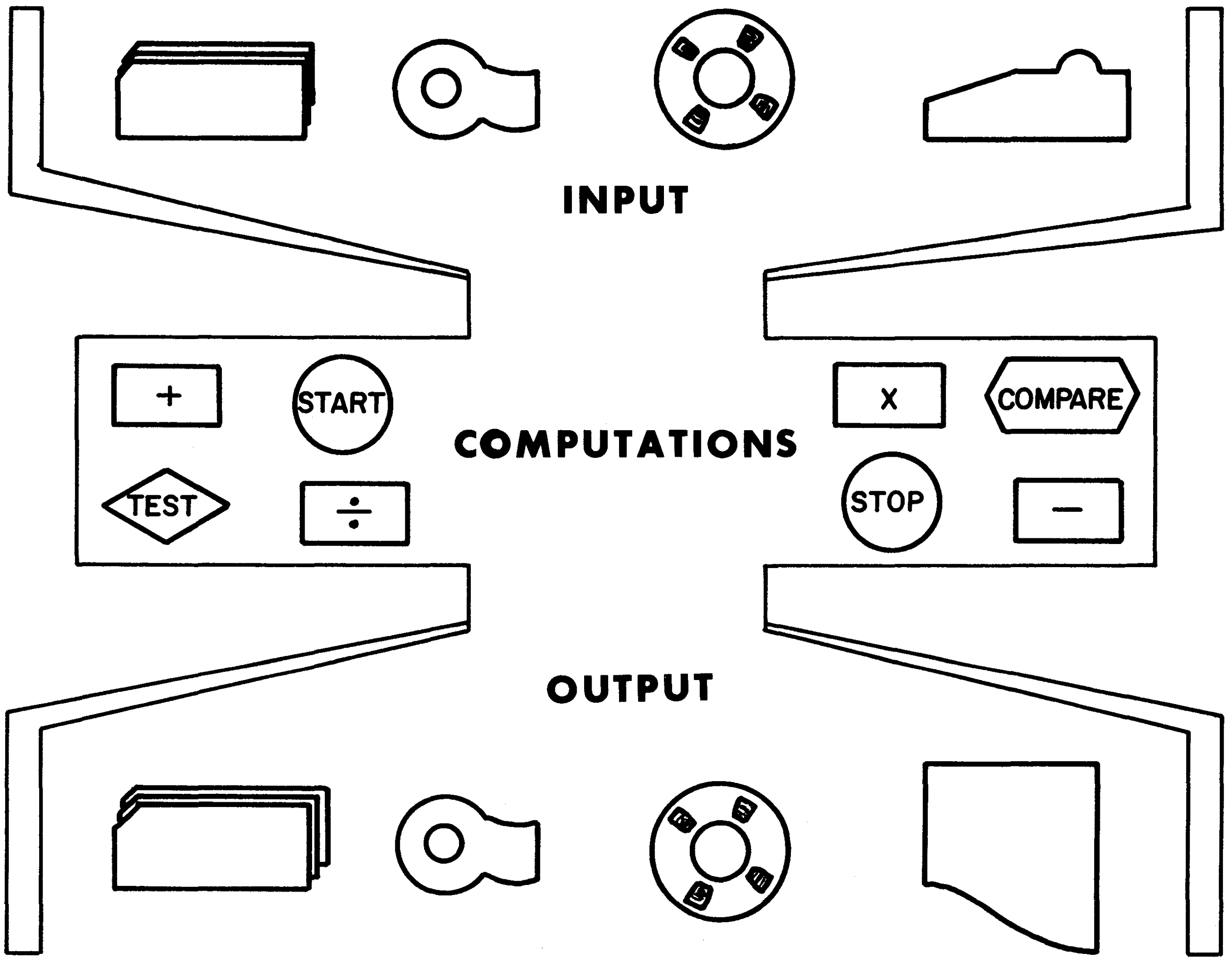

P. C. S.
PREPARE CONTROL SYSTEM
1
INPUT/OUTPUT DRIVER(S)
2
INPUT/OUTPUT CONTROL SYSTEM
3
RELOCATABLE LOADER
4
HEWLETT - PACKARD COMPUTER
"CONFIGURED" BASIC CONTROL SYSTEM
5

A BASIC CONTROL SYSTEM IS "TAILORED" TO THE HARDWARE CONFIGURATION OF THE SYSTEM.

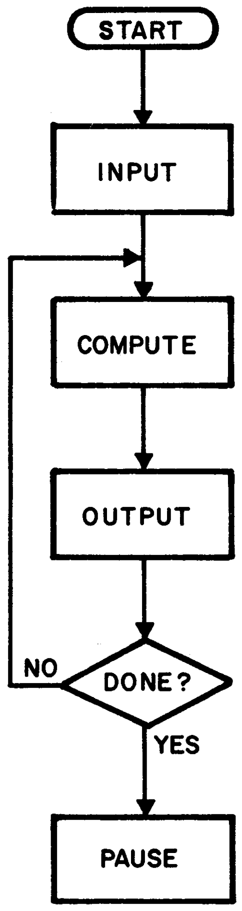
1. THE P.C.S CONTROL PROGRAM IS LOADED
2. THE INPUT/OUTPUT DRIVER (S) MODULE IS PROCESSED.
3. THE INPUT/OUTPUT CONTROL SYSTEM MODULE IS PROCESSED
4. THE RELOCATABLE LOADER MODULE IS PROCESSED
5. THE "CONFIGURED" B.C.S. TAPE IS PRODUCED.

THE COMPONENTS OF A COMPUTER PROGRAM

MOST COMPUTER PROGRAMS CONSIST OF THREE PARTS



EXAMPLE - **DEFINING THE PROBLEM**

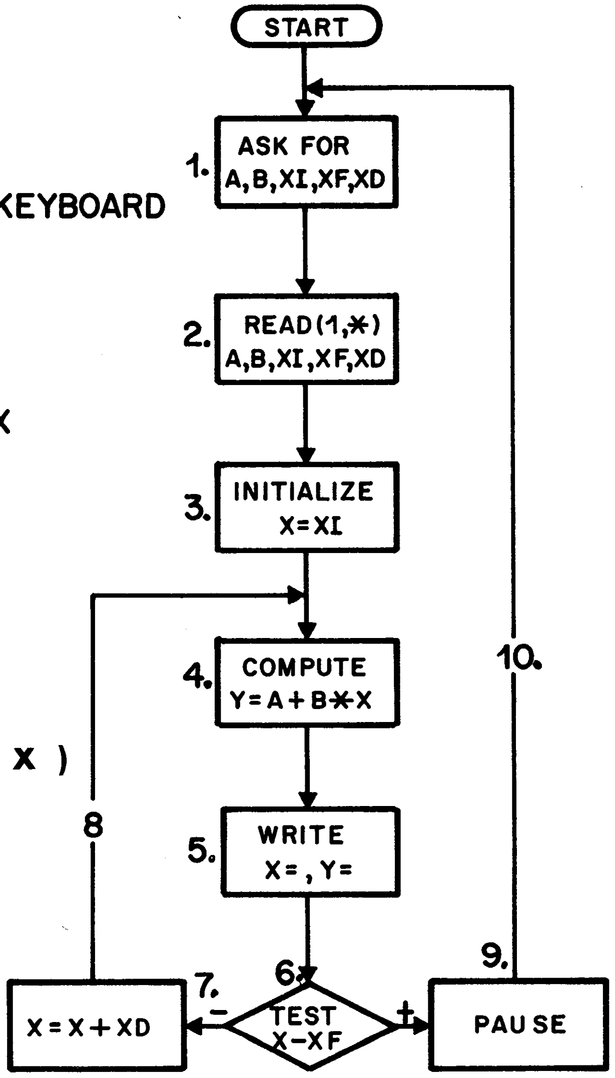


PROBLEM : SOLVE Y=A+B*X
WHERE: A&B ARE ENTERED ON THE KEYBOARD

X TAKES ON THE VALUES:
XI - INITIAL VALUE OF X
XF - FINAL VALUE OF X
XD - INCREMENTAL VALUE FOR X

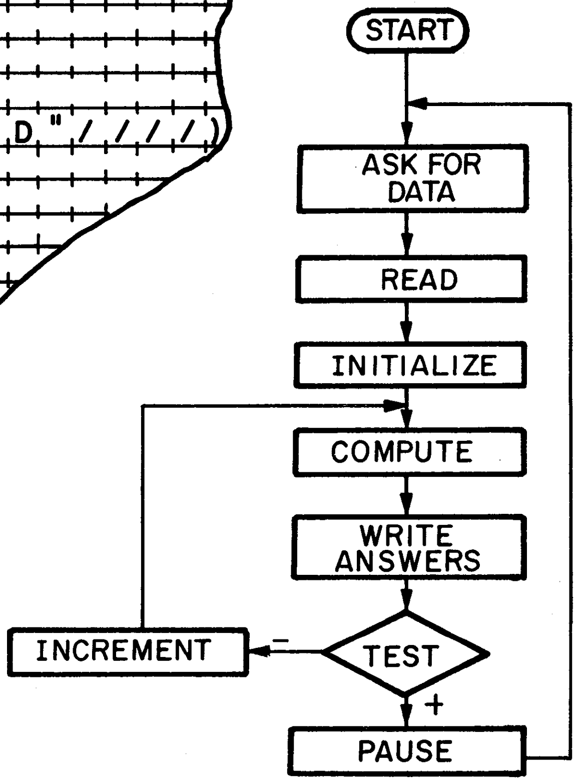
SOLUTION STEPS

1. ASK FOR A, B, XI, XF, XD
2. READ A, B, XI, XF, XD
3. INITIALIZE (X = XI)
4. CALCULATE (Y = A + B * X)
5. WRITE X AND Y
6. IF X<XF, GO TO STEP 7.
   IF X≥XF, GO TO STEP 9.
7. ADD XD TO X
8. GO TO STEP 4
9. PAUSE
10. WHEN RUN IS PUSHED, GO TO STEP 1.

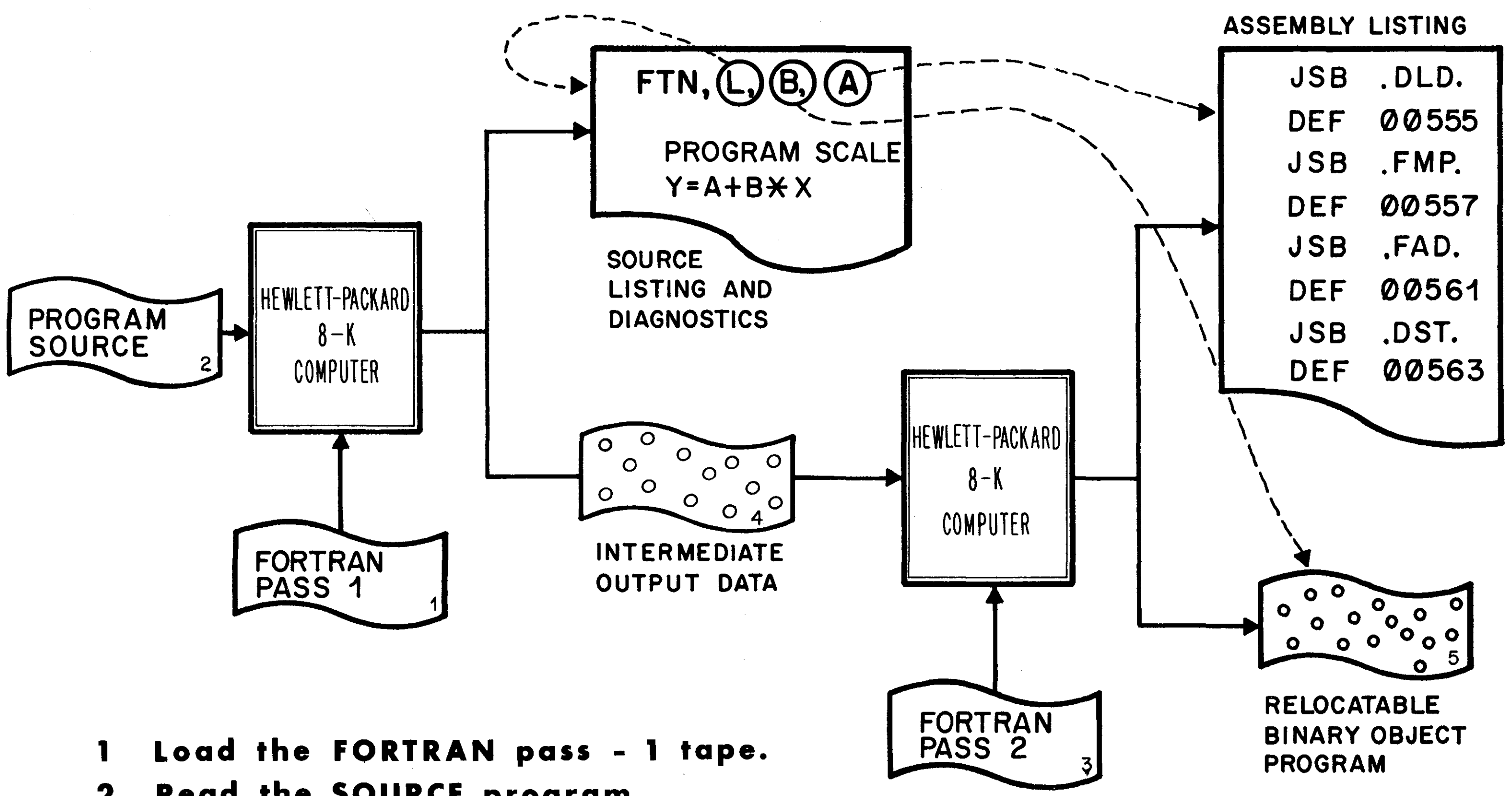


EXAMPLE- CODING THE PROGRAM

```
LABEL  C  STATEMENT
1    5 6  7
FTN,L ,   B,A
          PROGRAM SCALE

    1     WRITE(2,20)
   20     FORMAT("ENTER A,B,XI,XF,XD"////)

          READ(1,*)A,B,XI,XF,XD

          X=XI

    4     Y=A+B*X

          WRITE(2,30)X,Y
   30     FORMAT("(X,Y) = "2F12.4)

          IF (X-XF)7,9

    7     X=X+XD

          GO TO 4

    9     PAUSE

          GO TO 1

          END
          END$
```

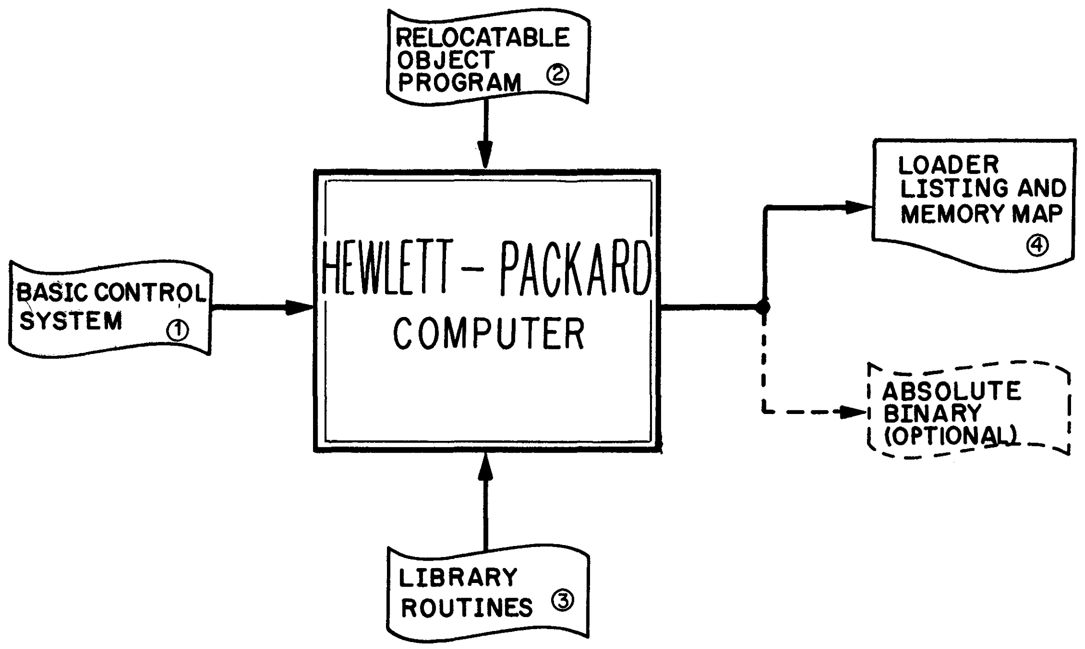


COMPILING THE PROGRAM



1 **Load the FORTRAN pass - 1 tape.**
2 **Read the SOURCE program.**
3 **Load the FORTRAN pass - 2 tape.**
4 **Read the INTERMEDIATE tape.**
5 **At this point compilation is complete.**

LOADING THE OBJECT PROGRAM


RELOCATABLE OBJECT PROGRAM ②
BASIC CONTROL SYSTEM ①
HEWLETT-PACKARD COMPUTER
LOADER LISTING AND MEMORY MAP ④
ABSOLUTE BINARY (OPTIONAL)
LIBRARY ROUTINES ③

1 - LOAD THE BASIC CONTROL SYSTEM TAPE.

2 - PROCESS THE OBJECT PROGRAM TAPE.

3 - PROCESS THE LIBRARY TAPE.

4 - THE LOADER LISTING ENDS WITH THE MESSAGE "*RUN".

5 - DEPRESSING THE "RUN" PUSHBUTTON ALLOWS PROGRAM EXECUTION TO BEGIN.

EXAMPLE – THE PROGRAM IN EXECUTION

```
ENTER A,B,XI, XF, XD                 ←——————— THE REQUEST FOR VALUES

1,2,1,10,1  CR LF              ←———————————— OPERATOR'S RESPONSE
(X,Y) =         1.0000      3.0000   ⎫
(X,Y) =         2.0000      5.0000   ⎪
(X,Y) =         3.0000      7.0000   ⎪
(X,Y) =         4.0000      9.0000   ⎪
(X,Y) =         5.0000     11.0000   ⎬      THE COMPUTED ANSWERS
(X,Y) =         6.0000     13.0000   ⎪
(X,Y) =         7.0000     15.0000   ⎪
(X,Y) =         8.0000     17.0000   ⎪
(X,Y) =         9.0000     19.0000   ⎪
(X,Y) =        10.0000     21.0000   ⎭
PAUSE                          ←———————————— THE PROGRAM PAUSES
ENTER A,B,XI, XF, XD                 ←——————— THE RUN BUTTON IS PUSHED

2,4,1,10,1  CR LF
(X,Y) =         1.0000      6.0000
(X,Y) =         2.0000     10.0000
(X,Y) =         3.0000     14.0000
(X,Y) =         4.0000     18.0000
(X,Y) =         5.0000     22.0000
(X,Y) =         6.0000     26.0000
(X,Y) =         7.0000     30.0000
(X,Y) =         8.0000     34.0000
(X,Y) =         9.0000     38.0000
(X,Y) =        10.0000     42.0000
PAUSE
```

TELETYPE KEYBOARD FUNCTIONS

CR = Carriage Return Key

LF = Line Feed Key

Introduction to HP FORTRAN

II

LESSON II
INTRODUCTION TO HP FORTRAN

LESSON II OBJECTIVES

THE PRIMARY OBJECTIVE OF LESSON II IS:

EXPLAIN <u>JUST</u> <u>ENOUGH</u> OF FORTRANS DO'S AND DONT'S TO ENABLE THE STUDENT TO:

1 - WRITE A SIMPLE FORTRAN PROGRAM

2 - KEYPUNCH THE PROGRAM

3 - COMPILE THE PROGRAM

4 - LOAD AND EXECUTE THE PROGRAM

THE METHOD OF "LEARNING BY DOING" WILL BE USED WHENEVER POSSIBLE IN THIS COURSE.

AN INTRODUCTION TO $\longrightarrow$

FORMULA TRANSLATION

FORMULATED STATEMENT

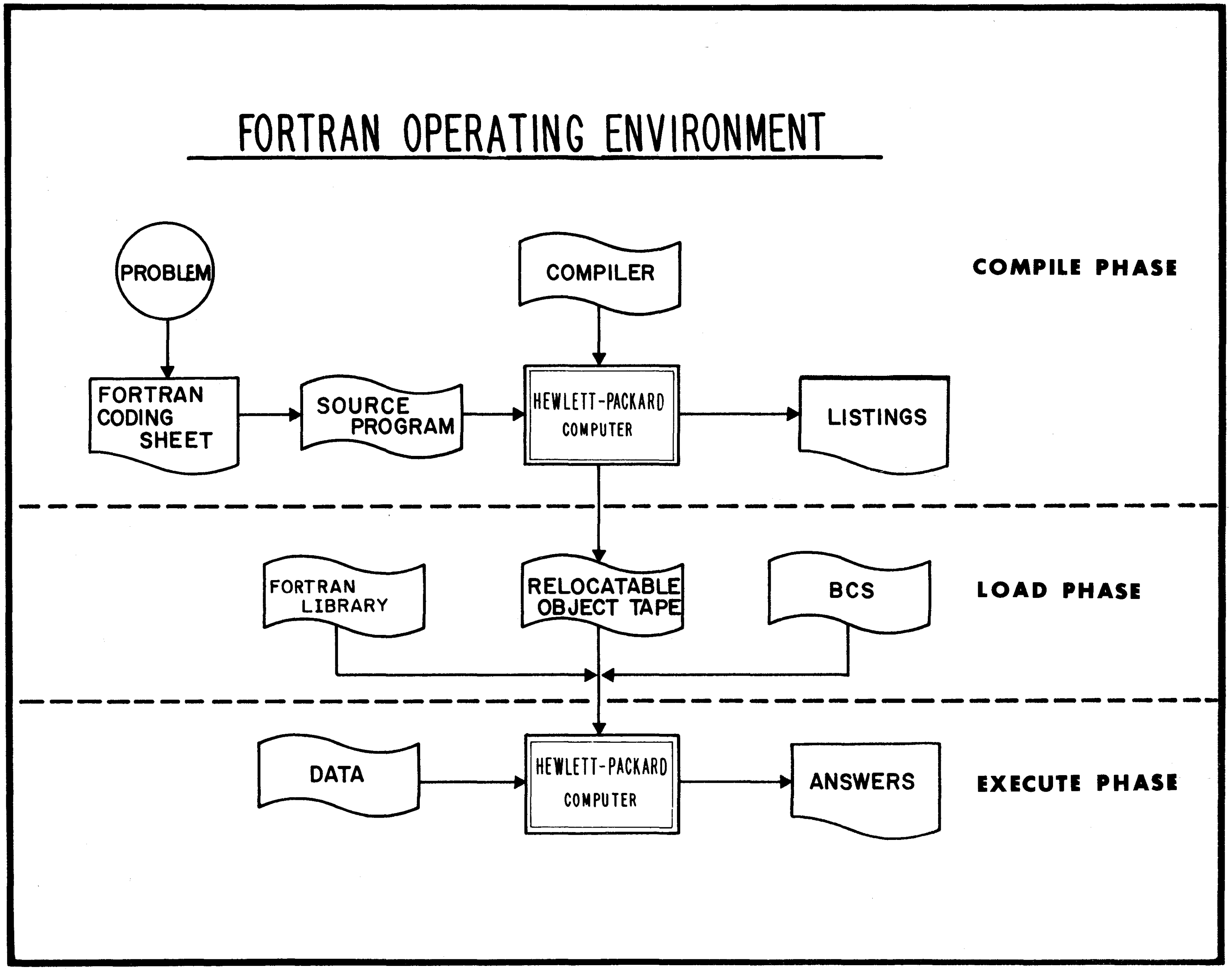
X=B*B-4.*A*C

YOU WILL:

1.) GET ACQUAINTED WITH FORTRAN

2.) WRITE FORTRAN PROGRAMS

3.) OPERATE THE COMPUTER


FORTRAN OPERATING ENVIRONMENT
PROBLEM
COMPILER
COMPILE PHASE
FORTRAN CODING SHEET
SOURCE PROGRAM
HEWLETT-PACKARD COMPUTER
LISTINGS
FORTRAN LIBRARY
RELOCATABLE OBJECT TAPE
BCS
LOAD PHASE
DATA
HEWLETT-PACKARD COMPUTER
ANSWERS
EXECUTE PHASE

A	THROUGH Z
Ø	THROUGH 9
	SPACE
=	EQUALS
+	PLUS
-	MINUS
*	ASTERISK
/	SLASH
( )	PARENTHESES
,	COMMA
$	DOLLAR SIGN
.	DECIMAL POINT
"	QUOTATION MARK

THE FORTRAN CHARACTER SET

2116A FORTRAN CODING FORM

PROGRAMMER	DATE	PROGRAM	PAGE OF

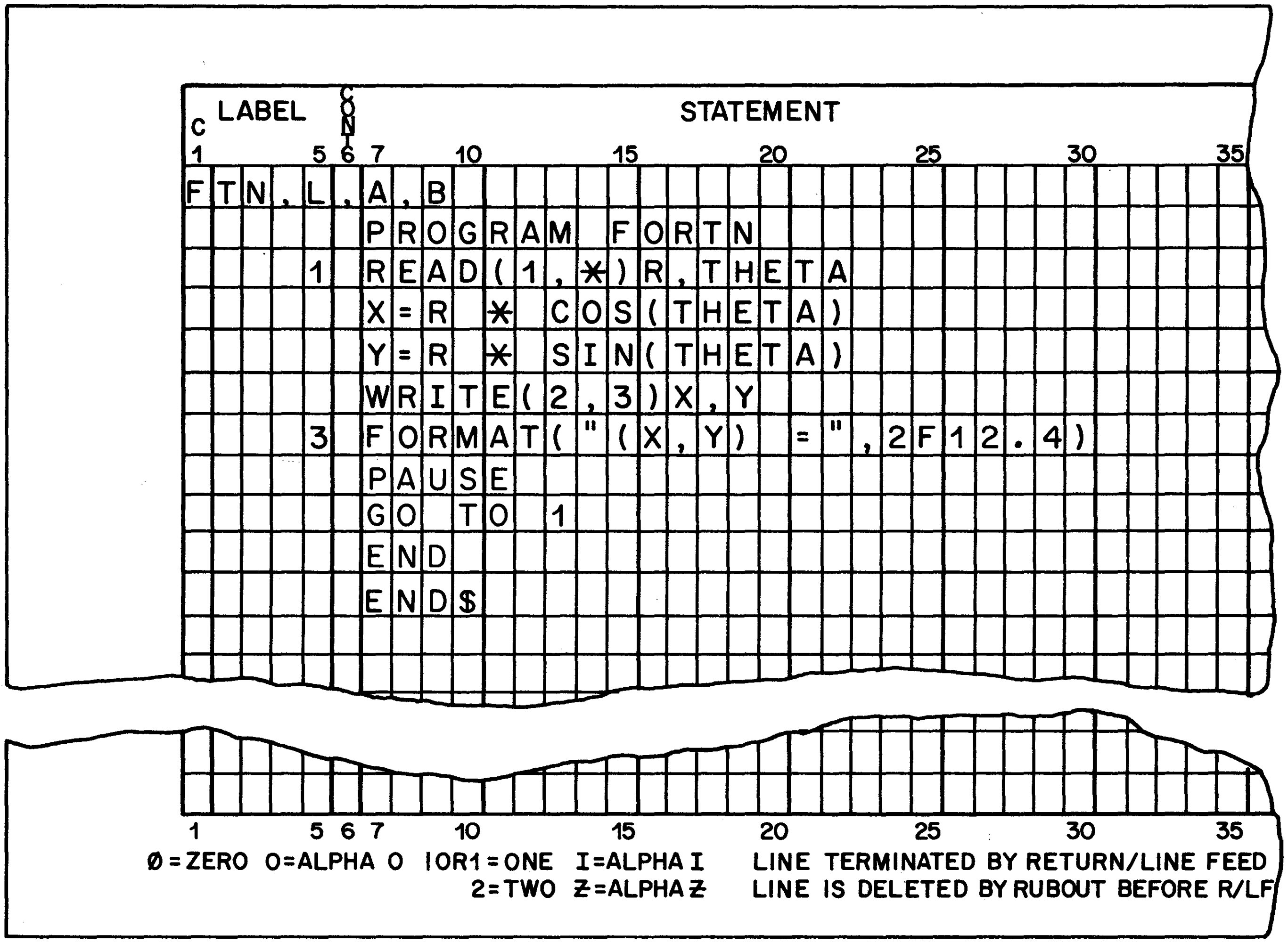
C Label CONT STATEMENT

1 5 6 7 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80

1 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80

Ø = ZERO O = ALPHA O I OR 1 = ONE 2 = TWO I = ALPHA I Z = ALPHA Z LINE TERMINATED BY RETURN / LINE FEED (R/LF) LINE IS DELETED BY RUBOUT BEFORE R/LF

M0251



C
LABEL
CONT
STATEMENT
1 5 6 7 10 15 20 25 30 35

```
FTN,L,A,B
      PROGRAM FORTN
    1 READ(1,*)R,THETA
      X=R * COS(THETA)
      Y=R * SIN(THETA)
      WRITE(2,3)X,Y
    3 FORMAT("(X,Y) =",2F12.4)
      PAUSE
      GO TO 1
      END
      END$
```

Ø=ZERO O=ALPHA O IOR1=ONE I=ALPHA I 2=TWO Ƶ=ALPHA Z

LINE TERMINATED BY RETURN/LINE FEED
LINE IS DELETED BY RUBOUT BEFORE R/LF

USING THE FORTRAN CODING FORM

FORTRAN STATEMENT LABEL RULES

1. LABELS ARE USED FOR REFERENCE BETWEEN PROGRAM STATEMENTS.

2. THE LABEL MAY CONSIST OF 1-4 NUMERIC DIGITS PLACED IN ANY OF THE FIRST FIVE POSITIONS OF A STATEMENT LINE. NO DUPLICATE LABELS ARE PERMITTED.

3. THE NUMBER IS UNSIGNED AND IN THE RANGE 1 TO 9999. THE LABELS DO NOT HAVE TO BE IN NUMERICAL SEQUENCE.

4. IMBEDDED SPACES AND LEADING ZEROS ARE IGNORED.

5. IF NO LABEL IS USED THE FIRST FIVE POSITIONS OF THE STATEMENT LINE MUST BE BLANK.

EXAMPLES

1	2	3	4	5
				1
			1	
9	9	9	9	
9		9	9	9
Ø	Ø	5	1	2
A	B	C	D	
2	3	.	5	
3	4	5	6	7

VALID LABELS

INVALID LABELS

COMMENTS AND CONTINUATION STATEMENTS

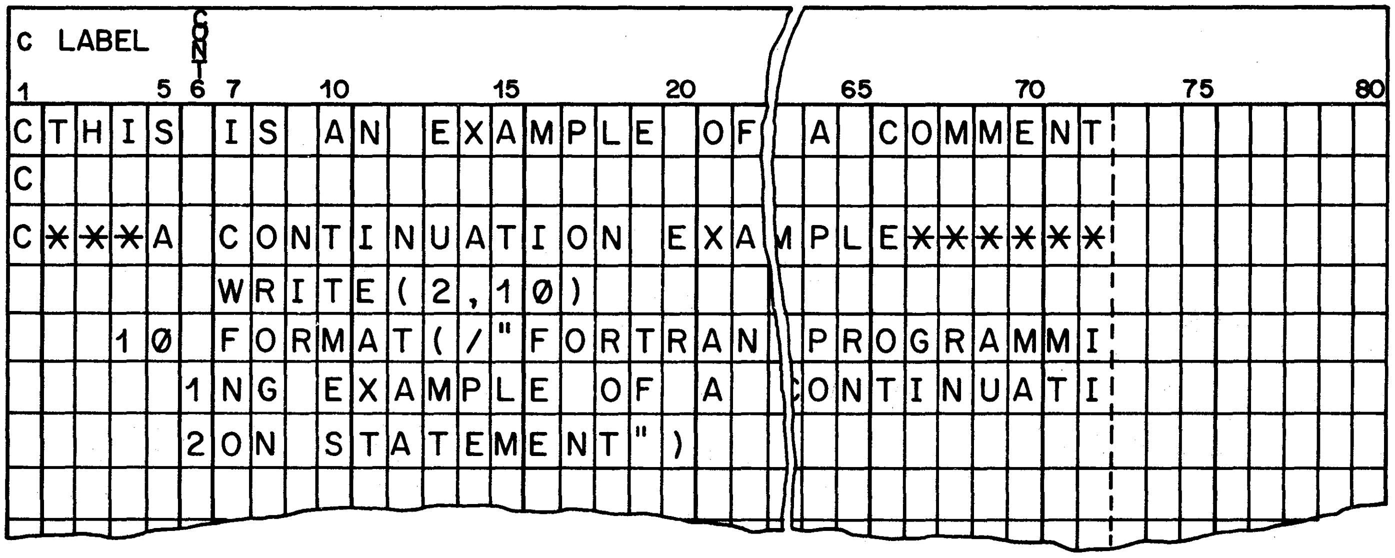
COMMENTS ARE IDENTIFIED BY THE CHARACTER "C" IN COLUMN 1. POSITIONS 2-72 MAY CONTAIN THE COMMENT.

CONTINUATION STATEMENTS ARE IDENTIFIED BY ANY CHARACTER OTHER THAN "SPACE OR ZERO" IN COLUMN 6 AND DO NOT CONTAIN A "C" IN COLUMN 1.

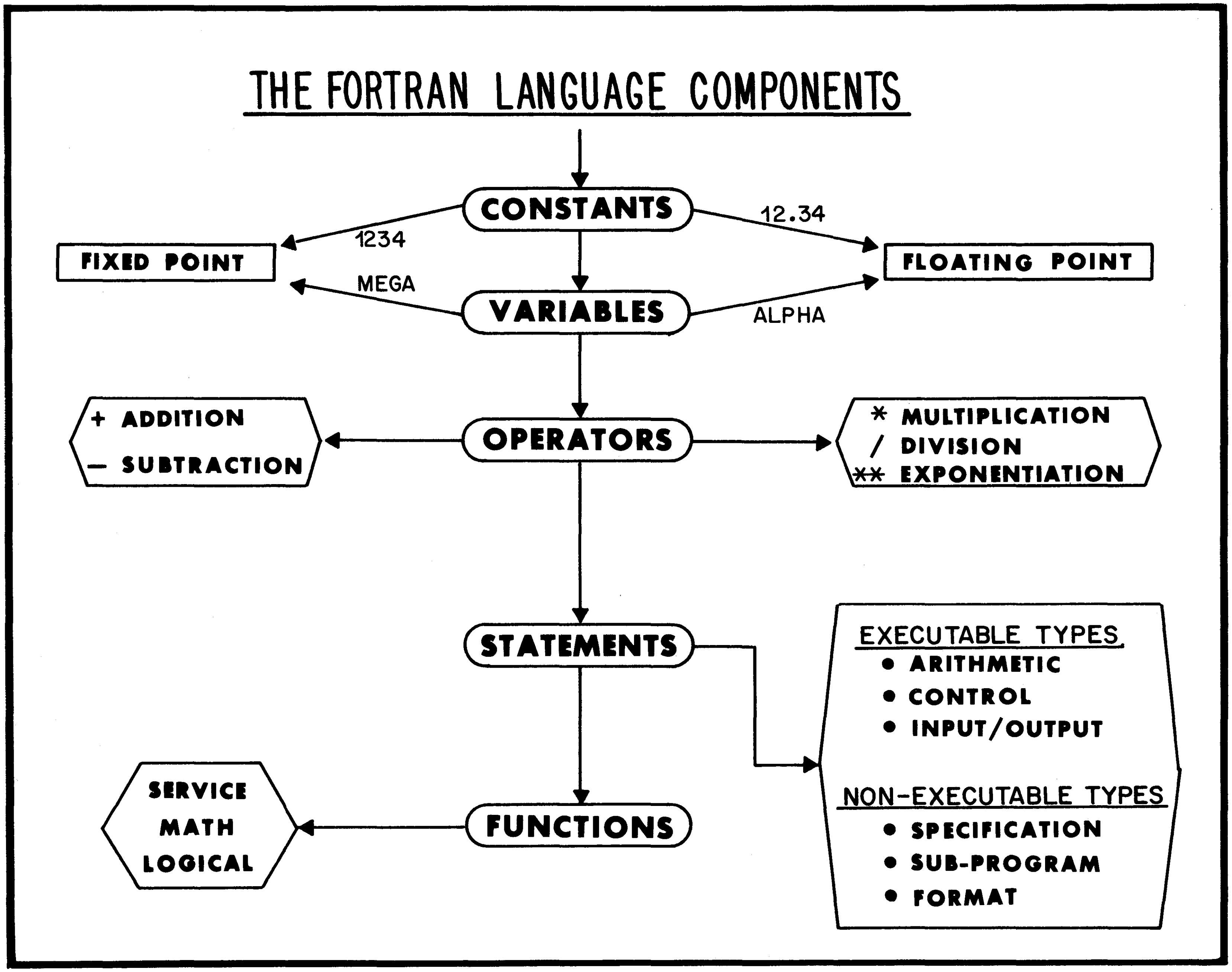
UP TO FIVE CONTINUATION LINES PER STATEMENT ARE PERMITTED.

COLUMNS 7-72 MAY BE USED FOR THE CONTINUATION STATEMENT.

EXAMPLES:



```
CTHIS IS AN EXAMPLE OF A COMMENT
C
C***A CONTINUATION EXAMPLE******
      WRITE(2,1Ø)
   1Ø FORMAT(/"FORTRAN PROGRAMMI
     1NG EXAMPLE OF A CONTINUATI
     2ON STATEMENT")
```


THE FORTRAN LANGUAGE COMPONENTS
CONSTANTS
1234
12.34
FIXED POINT
FLOATING POINT
MEGA
ALPHA
VARIABLES
OPERATORS
+ ADDITION
– SUBTRACTION
* MULTIPLICATION
/ DIVISION
** EXPONENTIATION
STATEMENTS
EXECUTABLE TYPES
• ARITHMETIC
• CONTROL
• INPUT/OUTPUT
NON-EXECUTABLE TYPES
• SPECIFICATION
• SUB-PROGRAM
• FORMAT
FUNCTIONS
SERVICE
MATH
LOGICAL

STATEMENTS

1.) EXECUTABLE TYPES

A - ARITHMETIC

B - CONTROL

1. GO TO
2. IF
3. DO
4. CALL
5. PAUSE
6. CONTINUE
7. RETURN
8. END

C - INPUT/OUTPUT

1. READ
   FREE- FIELD
   FORMATTED
2. WRITE
3. *REWIND*
4. *BACKSPACE*
5. *END FILE*

2.) NON-EXECUTABLE TYPES

A - SPECIFICATION

1. DIMENSION
2. *COMMON*
3. *EQUIVALENCE*

B - SUB-PROGRAM

1. FUNCTION
2. PROGRAM
3. SUBROUTINE

C - FORMATS

1. QUOTE
2. I
3. F
4. SLASH
5. H
6. X
7. E
8. A

FUNCTIONS

GROUP	TYPE	SYMBOL
SERVICE	TRANSFER SIGN	SIGN
	FLOAT	FLOAT
	FIX	IFIX
MATH	ABSOLUTE VALUE	ABS
	EXPONENTIAL	EXP
	NATURAL LOGARITHM	ALOG
	TRIGONOMETRIC SINE	SIN
	TRIGONOMETRIC COSINE	COS
	TRIGONOMETRIC TANGENT	TAN
	HYPERBOLIC TANGENT	TANH
	SQUARE ROOT	SQRT
	ARCTANGENT	ATAN
LOGICAL	BOOLEAN AND	IAND
	BOOLEAN OR	IOR
	BOOLEAN NOT	NOT

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS
FUNCTIONS

= ANY QUANTITY REPRESENTED BY A NUMERIC VALUE

FIXED POINT CONSTANTS

REPRESENT INTEGER NUMBERS. THERE IS NO DECIMAL POINT. THE NUMBER MUST BE IN THE RANGE -32768 TO +32767. THE VALUE IS REPRESENTED BY ONE COMPUTER WORD.

EXAMPLES— 7, -5, +132, 697, 1234, 32715

FLOATING POINT CONSTANTS

REPRESENT REAL NUMBERS. THERE MUST BE A DECIMAL POINT. THE NUMBER MUST BE IN THE RANGE -10^{38} TO $+10^{38}$. THE NUMBER YIELDS A PRECISION OF SEVEN DECIMAL DIGITS AND IS REPRESENTED BY TWO COMPUTER WORDS:

THE FRACTION PLUS SIGN UTILIZES 24 BITS.
THE EXPONENT PLUS SIGN UTILIZES 8 BITS.

EXAMPLES— 7., 7.Ø, -7., 52.3, 4.12, .17, 75.

NOTE: IN FORTRAN .... 3 IS NOT THE SAME AS 3.

THEREFORE	3.+5 3 +5.	ARE NOT PERMITTED	BUT	3.+5. 5 +3	ARE PERMITTED

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS
FUNCTIONS

= ANY QUANTITY REPRESENTED BY AN ALPHANUMERIC SYMBOL

<u>FIXED POINT VARIABLES</u> **THEY REPRESENT <u>INTEGER</u> NUMBERS. THERE IS NO DECIMAL POINT. THEY RANGE FROM -32768 TO +32767**

<u>*EXAMPLES:*</u> I, J, K, L, M, N, IKE, JOHN, KEN,

<u>FLOATING POINT VARIABLES</u> **THEY REPRESENT <u>REAL</u> NUMBERS.**
THEY ARE IN FLOATING POINT REPRESENTATION.
THEY RANGE FROM -10^{38} TO $+10^{38}$.

<u>*EXAMPLES:*</u> A, B,·· H, O, P,··· Z, ALPHA, BETA, SIGMA,

<u>NOTE:</u> A VARIABLE NAME IS COMPOSED OF 5 <u>OR</u> LESS ALPHANUMERIC CHARACTERS.

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS
FUNCTIONS

= SPECIAL CHARACTERS USED TO REPRESENT ARITHMETIC OPERATIONS + - * / **

WHEN OPERATORS ARE USED IN A STATEMENT TO FORM AN EXPRESSION, THE COMPILER EVALUATES THE EXPRESSION FROM LEFT TO RIGHT AND PERFORMS THE ARITHMETIC OPERATIONS IN THE FOLLOWING SEQUENCE:

CLASS 1 -	**	EXPONENTIATION
CLASS 2 -	*	MULTIPLICATION
	/	DIVISION
CLASS 3 -	+	ADDITION
	—	SUBTRACTION

EXCEPTION: **OPERATIONS MAY BE GROUPED BY THE USE OF PARENTHESES. WHEN USED; EXPRESSIONS WITHIN PARENTHESES ARE EVALUATED FIRST, THEN **, THEN * AND /, THEN + AND – .**

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS = EXECUTABLE TYPES
FUNCTIONS

ARITHMETIC

GENERAL FORM: $\alpha = \beta$

MUST BE A VARIABLE — α

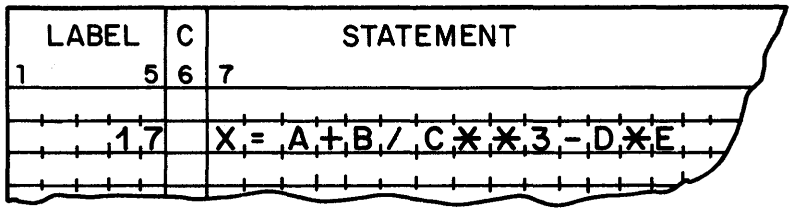
STANDS FOR "REPLACE" — $=$

MUST BE AN EXPRESSION — β

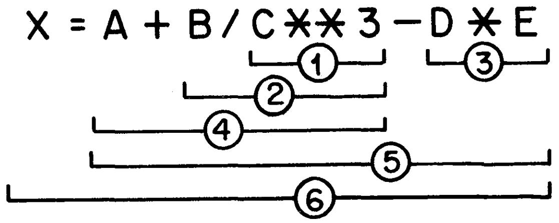
DEFINITION

AN EXPRESSION IS A COMBINATION OF CONSTANTS AND/OR VARIABLES SEPARATED BY OPERATORS.

EXAMPLE



WOULD BE SOLVED IN THIS ORDER



FORTRAN ARITHMETIC

FORTRAN ARITHMETIC IS VERY SIMILAR TO CONVENTIONAL MATHEMATICAL NOTATION. ONE IMPORTANT DIFFERENCE CONCERNS THE LEFT SIDE OF THE EQUAL SIGN. IN FORTRAN, THE TERM ON THE LEFT SIDE OF THE EQUAL SIGN MUST BE SINGLE VALUED.

FOR EXAMPLE: **ONE MIGHT SOLVE FOR** X/5:

$$X/5 = C^2 + Y^2$$

IN FORTRAN, THIS IS NOT PERMITTED, THE ABOVE MUST BE WRITTEN:

```
X = 5. *(C**2+Y**2)
```

— ADDITIONAL EXAMPLES —

CONVENTIONAL NOTATION	FORTRAN NOTATION
$X=3Y$	`X=3.*Y`
$N=6(K-2)$	`N=6*(K-2)`
$X+4=2Y$	`Y=(X+4.)/2.`
$\frac{Y}{2}=\frac{Z}{3}+\frac{X}{3}$	`Y=2.*((X+Z)/3.)`
$A^{(X+Y)}=Z$	`Z=A**(X+Y)`

ALGEBRAIC OPERATIONS IN FORTRAN

ALGEBRA	OPERATION	FORTRAN STATEMENT
$X = Y + Z$	Addition	X = Y + Z
$X = Y - Z$	Subtraction	X = Y - Z
$X = Y \cdot Z$	Multiplication	X = Y * Z
$X = Y/Z$	Division	X = Y/Z
$X = Y^Z$	Raise to a power	X = Y ** Z
$X = \sqrt{Y}$	Square Root	X = SQRT(Y)
$X = e^Y$	Natural Anti-Log	X = EXP(Y)
$X = \mathrm{SIN}(Y)$	Sine	X = SIN(Y)
$X = \mathrm{COS}(Y)$	Cosine	X = COS(Y)
$X = \mathrm{TAN}^{-1}(Y)$	Arc Tangent	X = ATAN(Y)
$X = \mathrm{TANH}(Y)$	Hyperbolic Tangent	X = TANH(Y)
$X = \mathrm{LN}(Y)$	Natural Log	X = ALOG(Y)

A WORD OF CAUTION! ! !

THE USE OF REAL AND INTEGER VALUES WITHIN AN EXPRESSION MUST NOT BE MIXED.

INTEGER VALUES SHOULD BE USED IN INTEGER EXPRESSIONS
AND
REAL VALUES SHOULD BE USED IN REAL EXPRESSIONS

VALID EXAMPLES

```
I  =  5
X  =  5.Ø
J  =  I+3
Y  =  X+3.Ø
K  =  I+J*K-3
Z  =  X+Y*Z-3.Ø
```

INVALID EXAMPLES

```
I  =  5+X*J      -    X IS A REAL VARIABLE
X  =  5.+I*Y     -    I IS AN INTEGER VARIABLE
J  =  K/2.5      -    2.5 IS A REAL CONSTANT
Y  =  Z*A/5      -    5 IS AN INTEGER CONSTANT
```

LEGAL INTERMIXING OF INTEGER AND REAL VALUES

EXPONENTIATION:

A REAL NUMBER MAY BE RAISED TO AN INTEGER POWER:

```
X = B**I
```

AN INTEGER NUMBER MAY NOT BE RAISED TO A REAL POWER:

```
J=I ** R
```

ACROSS THE EQUAL SIGN

AN INTEGER MAY BE SET EQUAL TO A REAL EXPRESSION:

```
I = X
```

A REAL NUMBER MAY BE SET EQUAL TO AN INTEGER:

```
X= I
```

EXPONENTIAL AND CONVERSION LIBRARY ROUTINES

.RTOI	REAL NUMBER TO INTEGER POWER
.RTOR	REAL NUMBER TO REAL POWER
.ITOI	INTEGER TO INTEGER POWER
FLOAT	INTEGER TO FLOATING POINT CONVERSION
IFIX	FLOATING POINT TO INTEGER CONVERSION

EVALUATION PROBLEM

GIVEN:

THE FORTRAN STATEMENT: X=A*B**C/D

PROBLEM:

WHICH OF THE FOLLOWING IS A CORRECT INTERPRETATION OF THE STATEMENT GIVEN?

a.) $X=[A*B]^{(C/D)}$) c.) $X=\frac{[A*B]^{C}}{D}$

b.) $X=A*[B^{(C/D)}]$ d.) $X=\frac{A*(B)^{C}}{D}$

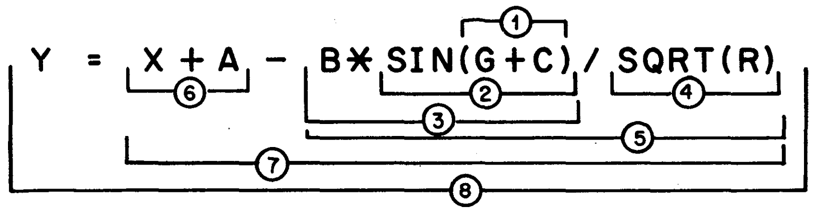
RULES:

- **EXPRESSIONS IN PARENTHESES ARE EVALUATED FIRST**
- **THEN **, THEN * AND /, THEN + AND —.**
- **STATEMENT SCANNING IS FROM LEFT TO RIGHT**

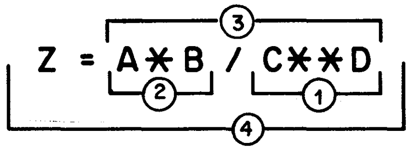
SUMMARY

EVALUATION OF STATEMENTS

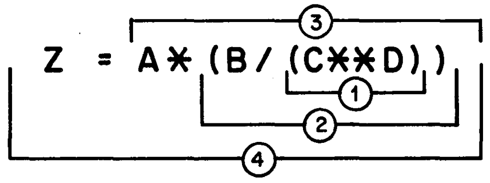
1



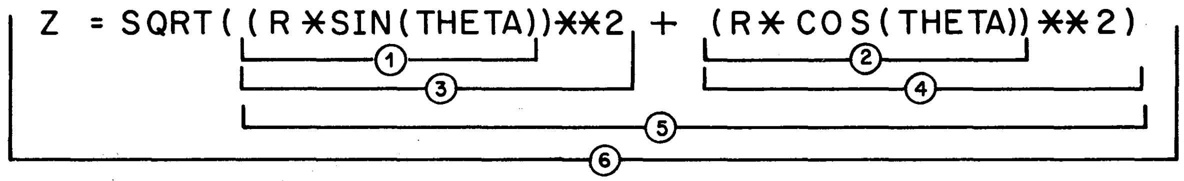
2



3



4



VARIABLE NAMES

- INteger names start with I,J,K,L,M,N
- REAL names start with A through H and O through Z
- Names have FIVE or LESS alphanumeric characters, the first being a letter

FORTRAN LIBRARY FUNCTIONS INCLUDE:

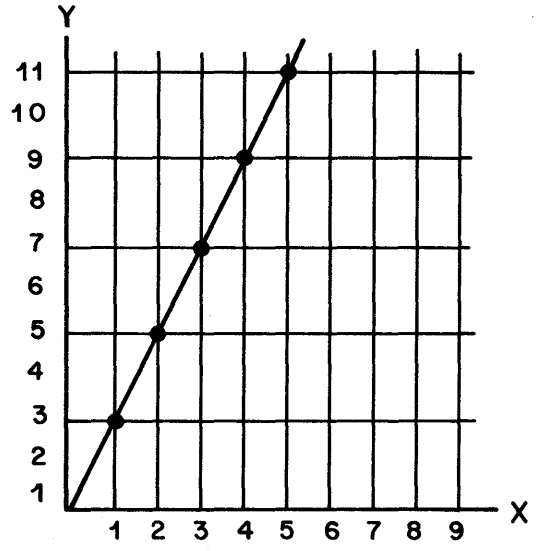
Y = SQRT (X)	Y = COS (X)
Y = EXP (X)	Y = TAN (X)
Y = ALOG (X)	Y= ATAN (X)
Y = SIN (X)	Y = TANH (X)

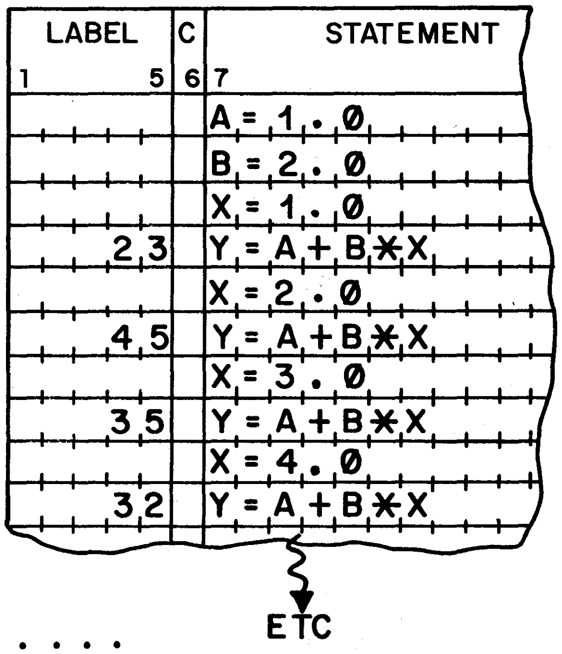
SAMPLE PROBLEM - IN-LINE CODING

PROBLEM

SOLVE Y=A+B·X
Where A=1, B=2 AND X=1,2,3,4,5....

SOLUTION

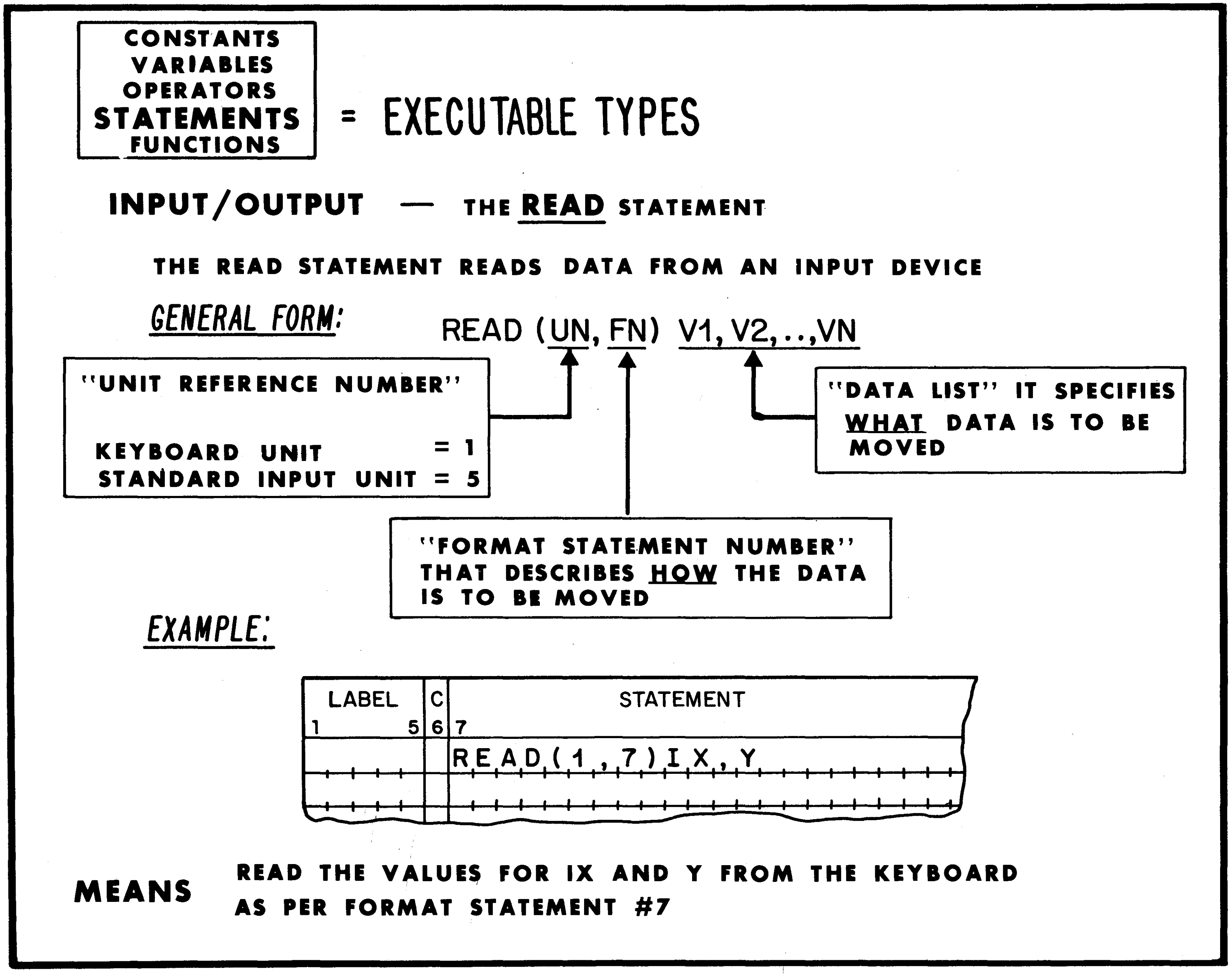




LABEL 1 5	C 6	STATEMENT 7
		A = 1.Ø
		B = 2.Ø
		X = 1.Ø
23		Y = A + B * X
		X = 2.Ø
45		Y = A + B * X
		X = 3.Ø
35		Y = A + B * X
		X = 4.Ø
32		Y = A + B * X

ETC

NOTE:
Y TAKES THE VALUES 3,5,7,9,11,13, ...


CONSTANTS
VARIABLES
OPERATORS
STATEMENTS
FUNCTIONS
= EXECUTABLE TYPES
INPUT/OUTPUT — THE READ STATEMENT
THE READ STATEMENT READS DATA FROM AN INPUT DEVICE
GENERAL FORM:
READ (UN, FN) V1, V2,..,VN
"UNIT REFERENCE NUMBER"
KEYBOARD UNIT = 1
STANDARD INPUT UNIT = 5
"DATA LIST" IT SPECIFIES WHAT DATA IS TO BE MOVED
"FORMAT STATEMENT NUMBER" THAT DESCRIBES HOW THE DATA IS TO BE MOVED
EXAMPLE:
LABEL
1 5
C
6
7
STATEMENT
READ(1,7)IX,Y
MEANS
READ THE VALUES FOR IX AND Y FROM THE KEYBOARD AS PER FORMAT STATEMENT #7

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS = **EXECUTABLE TYPES**
FUNCTIONS

INPUT/OUTPUT — THE **WRITE** STATEMENT

THE WRITE STATEMENT WRITES INFORMATION ON AN OUTPUT DEVICE

GENERAL FORM: WRITE (UN, FN) V1, V2, . . , VN

"UNIT REFERENCE NUMBER"

TELEPRINTER OUTPUT	=	2
PUNCH OUTPUT	=	4
LIST OUTPUT	=	6

"DATA LIST" IT SPECIFIES WHAT DATA IS TO BE MOVED

"FORMAT STATEMENT NUMBER" THAT DESCRIBES HOW THE DATA IS TO BE MOVED

EXAMPLE:

LABEL 1 5	C 6	STATEMENT 7
		WRITE(2,17)I2,PHI

MEANS

WRITE THE VALUES OF I2 AND PHI ON THE TELETYPE AS PER FORMAT STATEMENT #17

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS
FUNCTIONS

= NON-EXECUTABLE TYPES

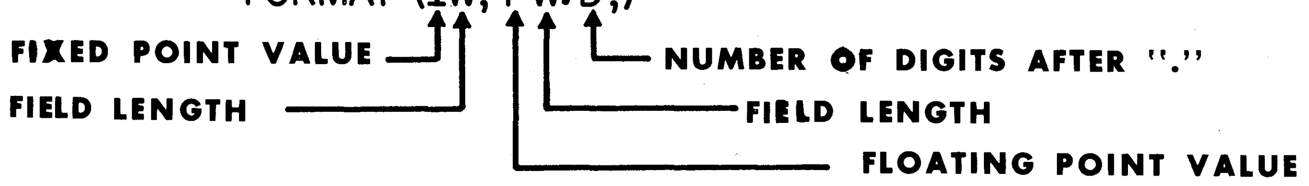
FORMATS – THE GENERAL SPECIFICATIONS

A FORTRAN FORMAT STATEMENT TELLS

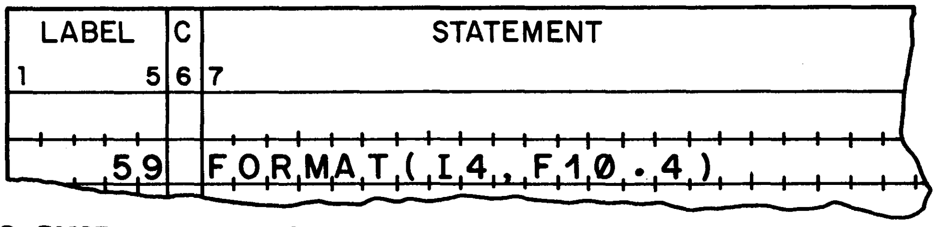
1- The length of the fields - I.E. the number of characters allocated to each variable in the data list
2- The mode of the values - I.E. fixed or floating point
3- Position of the decimal point- only for floating point

GENERAL FORM:

FORMAT (IW, FW.D,)



EXAMPLE:



LABEL 1–5	C 6	STATEMENT 7
59		FORMAT(I4,F10.4)

MEANS

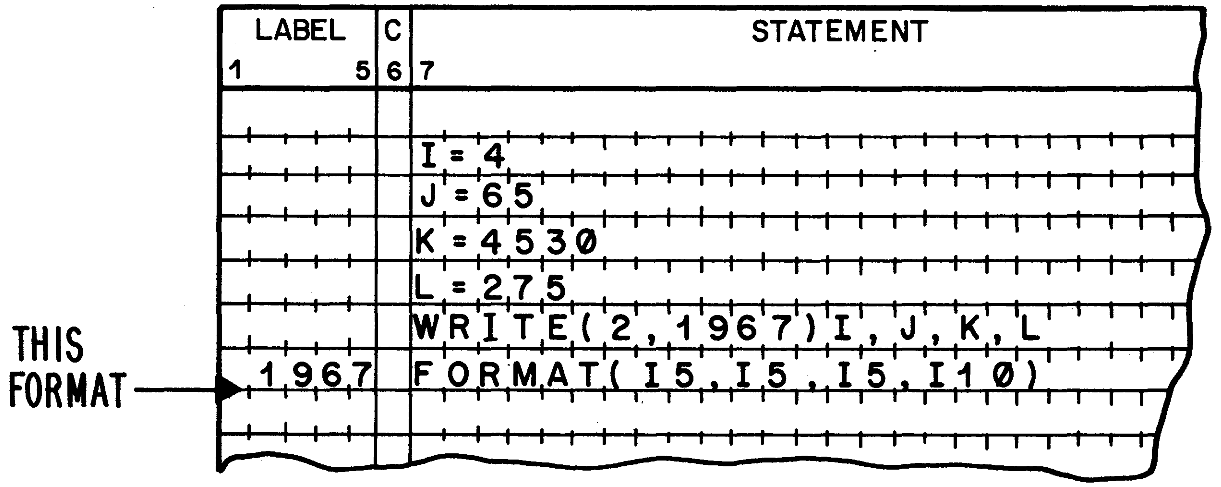
THE FIRST VALUE IS FIXED POINT (INTEGER) 4 CHARACTERS LONG
THE SECOND VALUE IS FLOATING POINT 10 DIGITS LONG WITH 4 DIGITS AFTER "."

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS
FUNCTIONS

= NON-EXECUTBLE TYPES

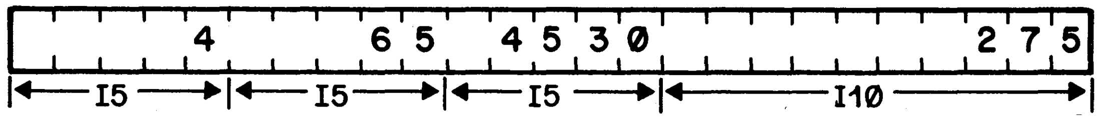
FORMATS – THE I FORMAT

THE I FORMAT IS USED TO DEFINE THE SPECIFICATIONS OF **INTEGER** VALUES.



```
      I=4
      J=65
      K=4530
      L=275
      WRITE(2,1967)I,J,K,L
 1967 FORMAT(I5,I5,I5,I10)
```

DEFINES ⟶ INTEGER DATA SPECIFICATIONS



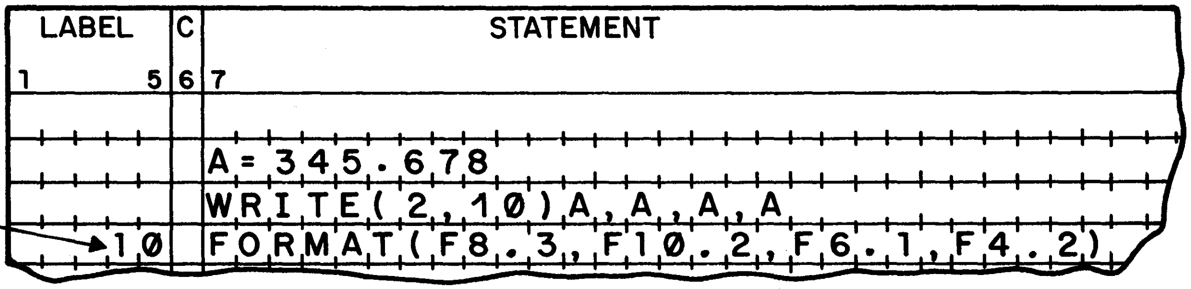
CONSTANTS
VARIABLES
OPERATORS
STATEMENTS
FUNCTIONS

= NON-EXECUTABLE TYPES

FORMATS — THE F FORMAT

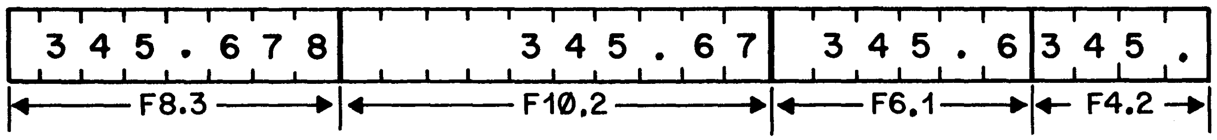
THE F FORMAT IS USED TO DEFINE THE SPECIFICATIONS OF FLOATING POINT VALUES.

THIS FORMAT



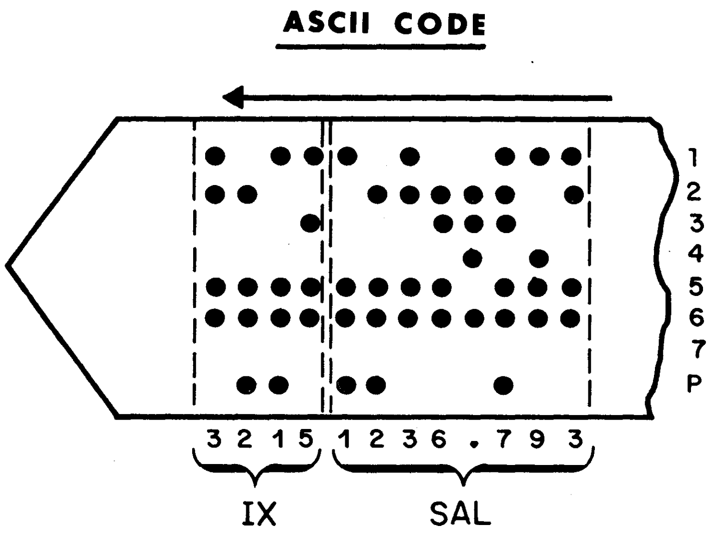
```
LABEL  C  STATEMENT
1    5 6 7

           A=345.678
           WRITE(2,1Ø)A,A,A,A
   1Ø      FORMAT(F8.3,F1Ø.2,F6.1,F4.2)
```

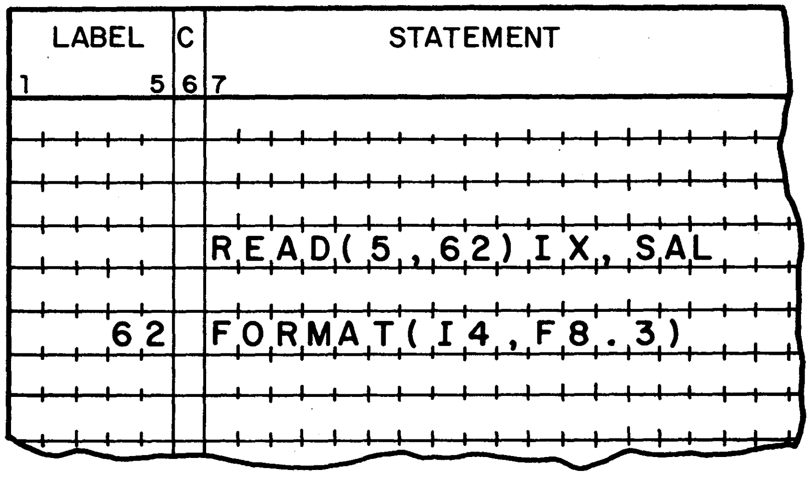
DEFINES **FLOATING POINT DATA SPECIFICATIONS**



EXAMPLE: **USING THE I AND F FORMAT WITH A READ STATEMENT**

READ FROM THE PHOTOREADER IX AND SAL
IX IS 4 CHARACTERS LONG
SAL IS 8 CHARACTERS LONG, 3 OF WHICH ARE AFTER THE DECIMAL POINT



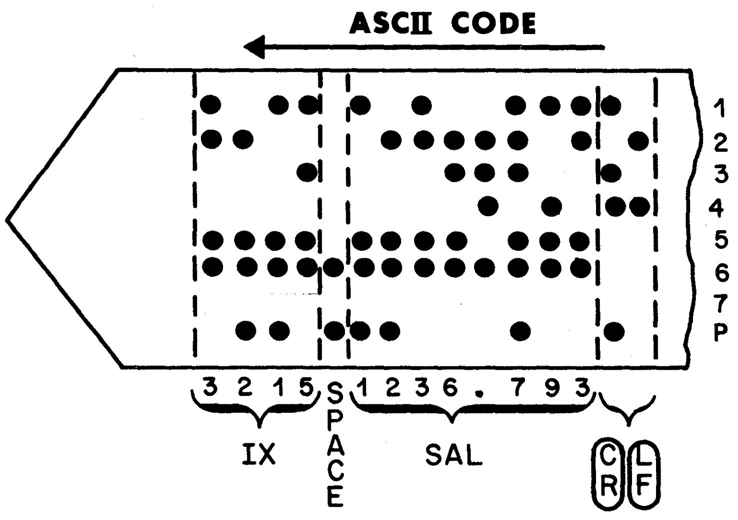


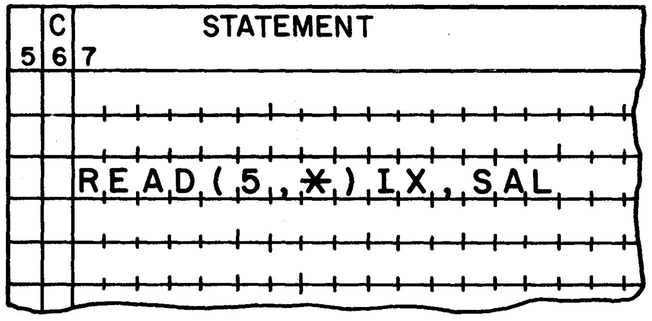
```
      READ(5,62)IX,SAL
   62 FORMAT(I4,F8.3)
```

FREE FIELD INPUT

FREE FIELD INPUT PERMITS THE READING OF NUMERIC DATA WITHOUT A DEFINITIVE FORMAT STATEMENT. DATA ITEMS ARE SEPARATED BY SPACES OR A COMMA. THE ASTERISK (✱) CHARACTER IS USED IN PLACE OF THE FORMAT NUMBER IN THE READ STATEMENT TO DEFINE THE FREE FIELD MODE OF INPUT.

EXAMPLE: USING FREE FIELD INPUT, READ VALUES FOR IX AND SAL

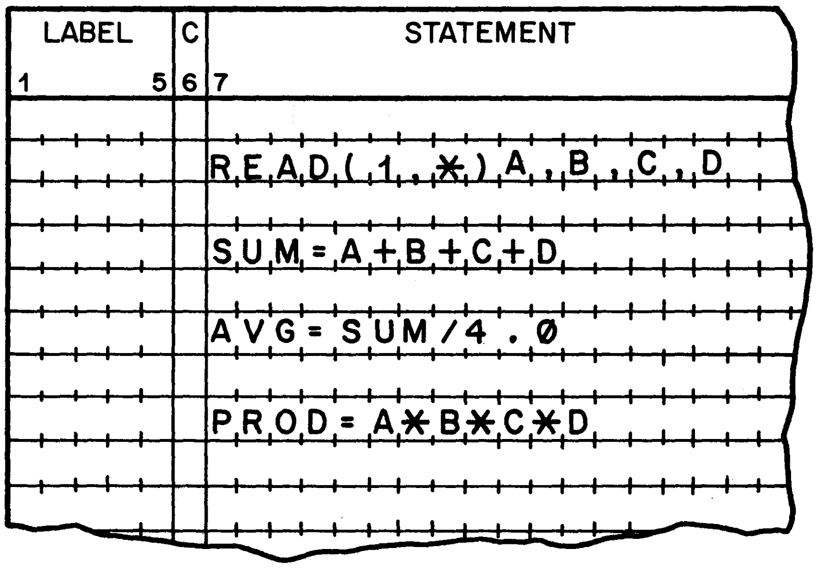




SAMPLE PROBLEM - FREE-FIELD DATA INPUT

PROBLEM: READ FOUR NUMBERS FROM THE KEYBOARD AND CALCULATE THE SUM, AVERAGE AND PRODUCT.

SOLUTION



```
LABEL    C  STATEMENT
1      5 6  7
            READ(1,*)A,B,C,D
            SUM=A+B+C+D
            AVG=SUM/4.Ø
            PROD=A*B*C*D
```

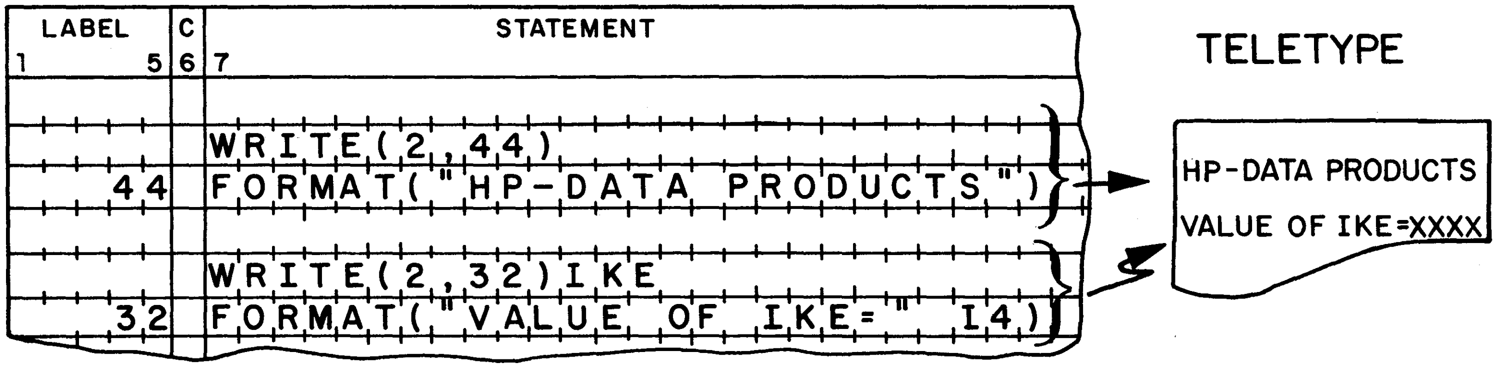
CONSTANTS
VARIABLES
OPERATORS
STATEMENTS – NON-EXECUTABLE TYPES
FUNCTIONS

FORMATS – THE QUOTE FORMAT

THE QUOTE FORMAT IS USEFUL FOR WRITING MESSAGES AND HEADINGS ON AN OUTPUT DEVICE. TO ACCOMPLISH THIS YOU ENCLOSE THE MESSAGE IN QUOTATION MARKS.

EXAMPLE:

THE FOLLOWING STATEMENTS WILL GENERATE CODING TO PRODUCE MESSAGES ON THE TELETYPE:



```
      WRITE(2,44)
   44 FORMAT("HP-DATA PRODUCTS")

      WRITE(2,32)IKE
   32 FORMAT("VALUE OF IKE=" I4)
```

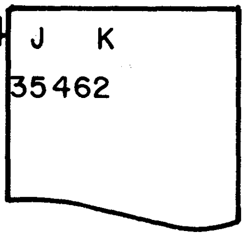
```
HP-DATA PRODUCTS
VALUE OF IKE=XXXX
```

SAMPLE PROBLEM–DATA OUTPUT

EXAMPLE: USING THE I, F AND QUOTE FORMATS WITH A WRITE STATEMENT.

WRITE ON THE TELETYPE THE HEADING "J K". THEN WRITE THE VALUES OF J AND K WHERE J IS 3 CHARACTERS LONG AND K IS 2 CHARACTERS LONG.

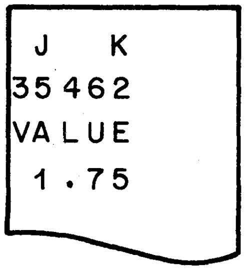
SAY J=354
K=62



```
 J  K
35462
```

TTY OUTPUT

NOW, WRITE THE HEADING "VALUE" AND THE VALUE OF V, WHERE V IS XX.XX (SAY 1.75)



```
 J  K
35462
VALUE
 1.75
```

TTY OUTPUT

```
LABEL  C  STATEMENT
1   5  6  7
          WRITE(2,3)

          WRITE(2,7)J,K

    7     FORMAT(I3,I2)

    3     FORMAT(" J  K")



          WRITE(2,6)

          WRITE(2,37)V

    6     FORMAT("VALUE")

   37     FORMAT(F5.2)
```

The HP symbolic editor program III

LESSON III
THE HP SYMBOLIC EDITOR PROGRAM

LESSON III OBJECTIVES

TO INSTRUCT THE STUDENT IN THE USE OF THE HEWLETT - PACKARD SYMBOLIC EDITOR PROGRAM. MASTERING THE USE OF THE SYMBOLIC EDITOR WILL ALLOW THE STUDENT TO CORRECT ERRORS IN SOURCE LANGUAGE PROGRAMS BY REPLACING, DELETING OR INSERTING THE APPROPRIATE STATEMENT (S).

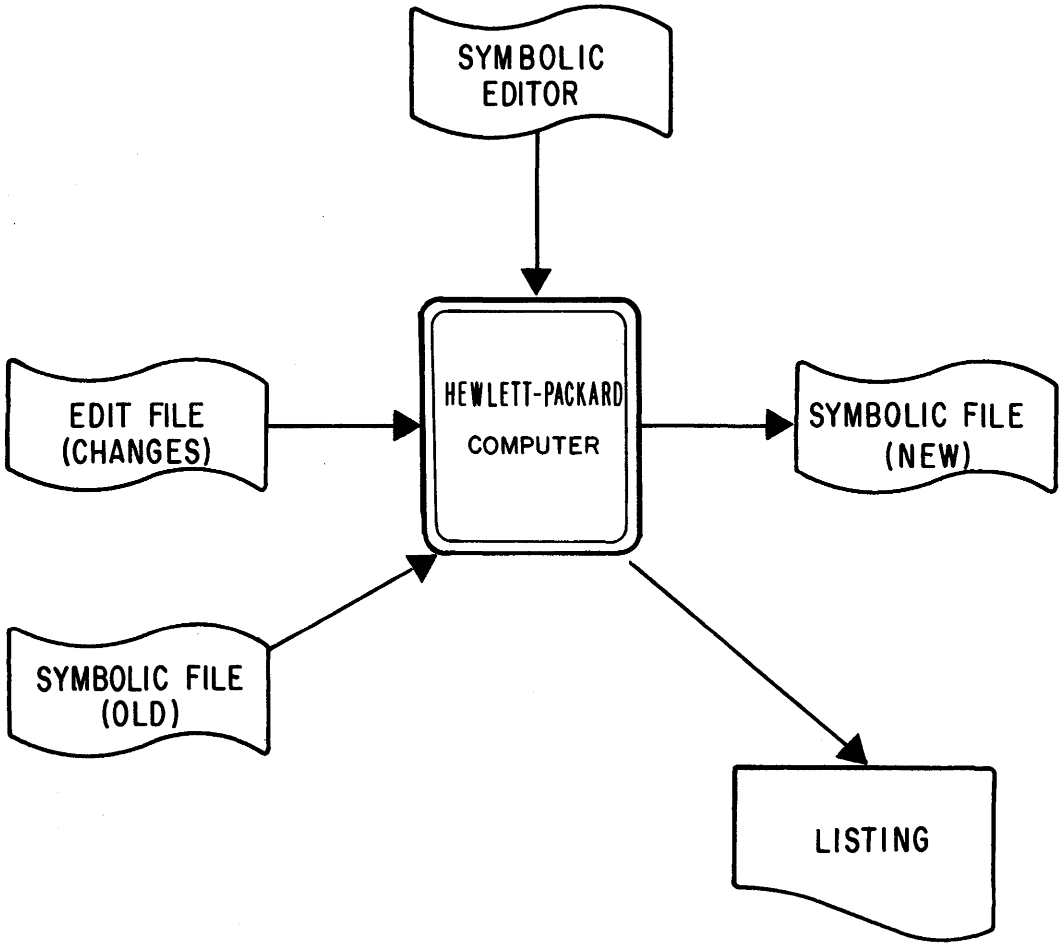
THE SYMBOLIC EDITOR SYSTEM

DEFINITION -----A software program for maintenance functions associated with development of computer programs.

PURPOSE------Provides a method for editing and updating symbolic source language programs or files.

MAJOR PROGRAM CAPABILITIES

1) Provides for the insertion, deletion and replacement of:
   - Entire source statements
   - Characters within a source statement
2) Provides a listing of a source program
3) Produces an updated source tape



EDIT PROCESS

EDITOR RULES AND FORMATS

▶ ALL CONTROL STATEMENTS BEGIN WITH THE CHARACTER SLASH (/).

▶ ALL STATEMENTS ARE TERMINATED BY A CARRIAGE RETURN, LINE FEED CODE. (CR, LF)

▶ THE EDIT FILE IS TERMINATED BY A (/E) CONTROL STATEMENT.

▶ ALL STATEMENTS ARE REFERRED TO BY THEIR SEQUENCE IN THE SOURCE FILE.

STATEMENT EDITING

/e, r_1 [, r_2] where:

e = An editing code: I, D or R

r's = Sequence numbers in the range 1 through 9999. r_2, if specified, must be greater than r_1.

CHARACTER EDITING

/ee, r, c_1 [, c_2] where:

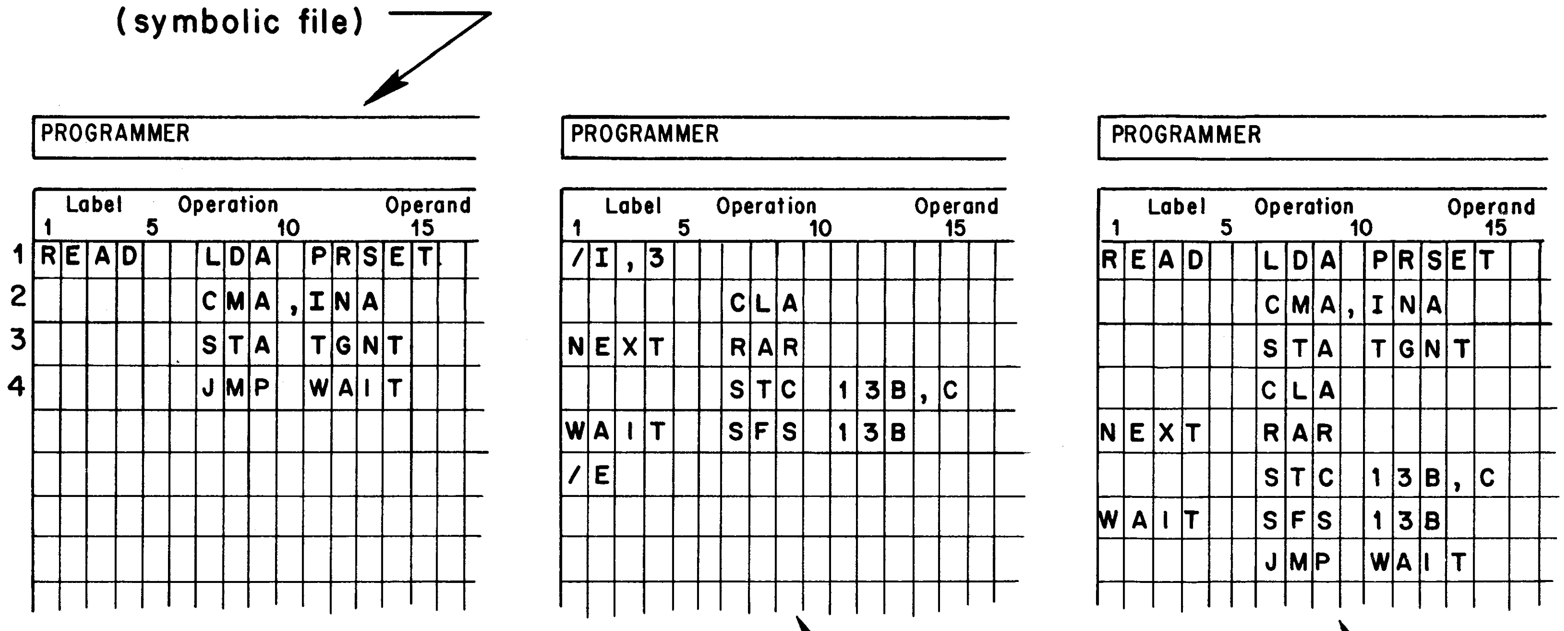
ee = An editing code: CI, CD, or CR

r = Sequence number in the range 1 through 9999.

c's = Character positions within the record that are to be edited. Blank positions must be included in the character count. An edited statement MAY NOT exceed 72 characters.

STATEMENT INSERTIONS

EXAMPLE: If we wanted to insert four statements into the original source program following statement number 3



```
1 READ  LDA PRESET
2       CMA,INA
3       STA TGNT
4       JMP WAIT
```

```
/I,3
      CLA
NEXT  RAR
      STC 13B,C
WAIT  SFS 13B
/E
```

```
READ  LDA PRESET
      CMA,INA
      STA TGNT
      CLA
NEXT  RAR
      STC 13B,C
WAIT  SFS 13B
      JMP WAIT
```

This coding would be necessary to create an updated file (edit file)

NOTE: REFERENCES TO LINE NUMBERS MUST BE SEQUENTIAL AND UNIQUE.

Such that the editor can produce a new symbolic file (new source program)

STATEMENT DELETIONS

EXAMPLE: If we wanted to delete statement numbers 2, 4, 5 and 6 from the original source program

(symbolic file)

```
PROGRAMMER

  Label    Operation      Operand
  1   5       10            15
1 READ  LDA PRSET
2       CMA,INA
3       STA TGNT
4       CLA
5 NEXT  RAR
6       STC 13B,C
7 WAIT  SFS 13B
8       JMP WAIT
```

```
PROGRAMMER

Label    Operation      Operand
1   5       10            15
/D,2
/D,4,6
/E
```

```
PROGRAMMER

Label    Operation      Operand
1   5       10            15
READ  LDA PRSET
      STA TGNT
WAIT  SFS 13B
      JMP WAIT
```

This coding would be necessary to create an updated file (edit file)

Such that the editor can produce a new symbolic file (new source program)

STATEMENT REPLACEMENTS

EXAMPLE: If we wanted to replace statement numbers 1 through 3 and 6 and 7 from the original source program

(symbolic file)

PROGRAMMER

	Label	Operation	Operand
1	READ	LDA	PRSET
2		CMA,INA	
3		STA	TGNT
4		CLA	
5	NEXT	RAR	
6		STC	13B,C
7	WAIT	SFS	13B
8		JMP	WAIT

PROGRAMMER

Label	Operation	Operand
/R,1,3		
READ	LDB	PRSET
/R,6,7		
	STC	15B,C
WAIT	SFS	15B
/E		

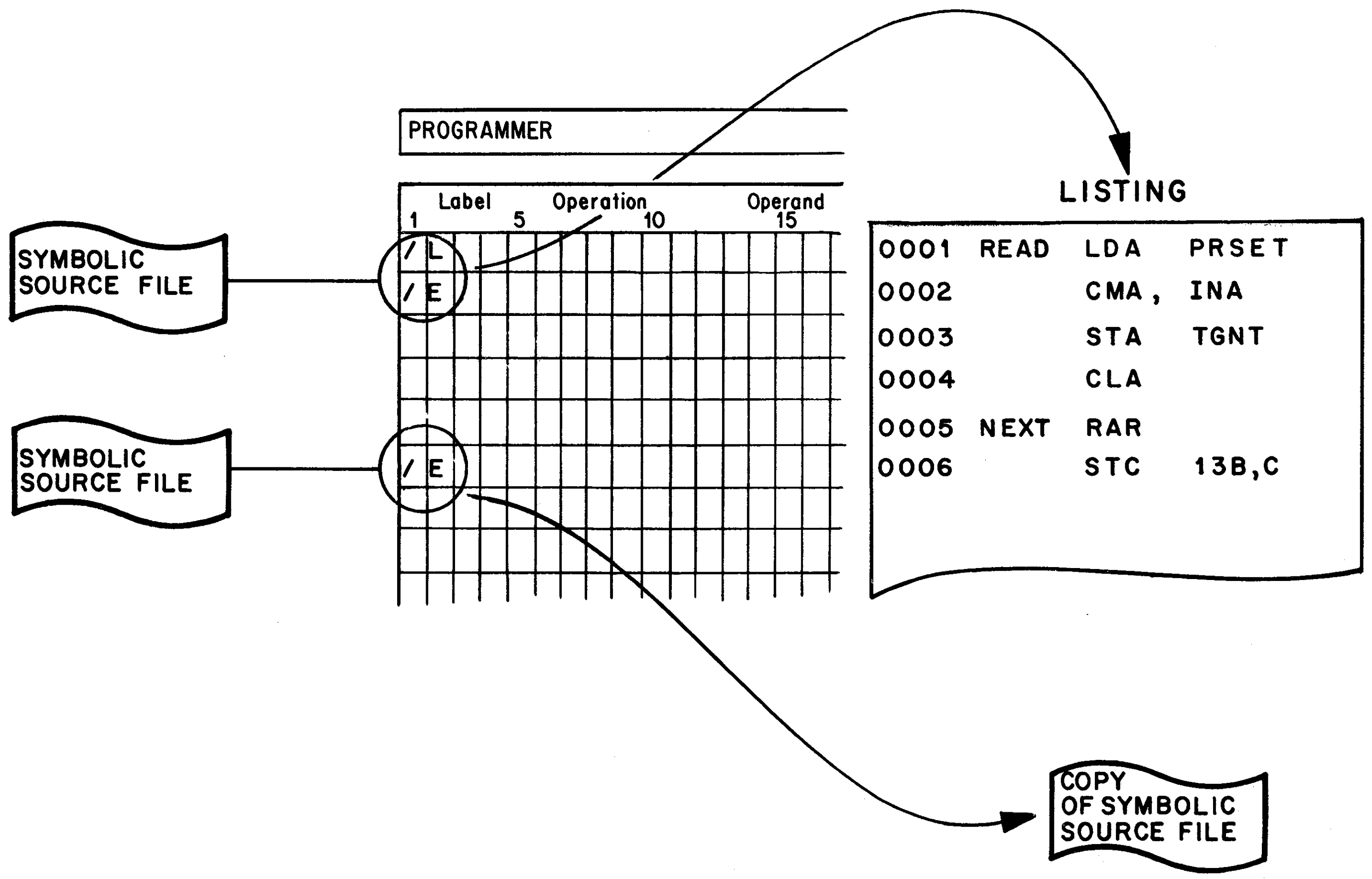
PROGRAMMER

Label	Operation	Operand
READ	LDB	PRSET
	CLA	
NEXT	RAR	
	STC	15B,C
WAIT	SFS	15B
	JMP	WAIT

This coding would be necessary to create an updated file (edit file)

Such that the editor can produce a new symbolic file (new source program)

SYMBOLIC FILE LIST/COPY FUNCTIONS



```
0001  READ  LDA   PRSET
0002        CMA,  INA
0003        STA   TGNT
0004        CLA
0005  NEXT  RAR
0006        STC   13B,C
```

FORTRAN control statements IV

LESSON IV
FORTRAN CONTROL STATEMENTS

LESSON IV OBJECTIVES

TO INTRODUCE SOME ADDITIONAL CAPABILITIES OF FORTRAN.

1 - TRANSFER OF PROGRAM CONTROL FROM ONE FORTRAN STATEMENT TO ANOTHER.

2 - MAKING LOGICAL DECISIONS BASED ON THE RESULTS OF AN EVALUATED EXPRESSION.

3 - EXECUTING A GROUP OF FORTRAN STATEMENTS A SPECIFIED NUMBER OF TIMES.

4 - CREATING AND OPERATING ON ARRAYS OF DATA USING VARIABLES WITH SUBSCRIPTS.

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS
FUNCTIONS

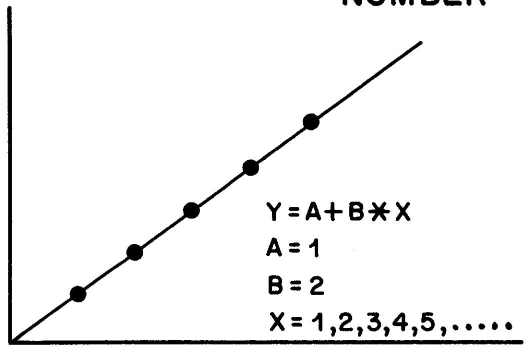
– EXECUTABLE TYPES

CONTROL — THE GO TO STATEMENT

GENERAL FORM

GO TO N

STATEMENT NUMBER



LABEL 1 – 5	C 6	STATEMENT 7
		A=1.Ø
		B=2.Ø
		X=1.Ø
45		Y=A+B✱X
		X=X+1.Ø
		GO TO 45

NOTE: THE "GO TO" STATEMENT IN THIS EXAMPLE ALLOWS THE PROGRAMMER TO REPEAT THE CALCULATION INDEFINITELY. Y TAKES THE VALUES: 3,5,7,9,11,13,15. . . . .

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS
FUNCTIONS

- EXECUTABLE TYPES

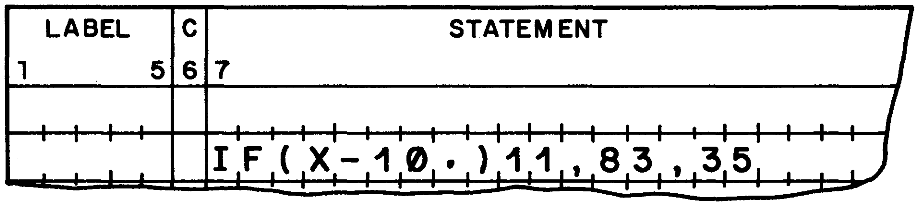
CONTROL — THE IF STATEMENT

GENERAL FORM

$$\mathrm{IF}\ \underbrace{(E)}_{\text{AN EXPRESSION}}\ \underbrace{N_1,N_2,N_3,}$$

STATEMENT TO "GO TO" DEPENDING UPON THE EVALUATION OF E
IF $E<\emptyset$ ∴ GO TO N_1
IF $E=\emptyset$ ∴ GO TO N_2
IF $E>\emptyset$ ∴ GO TO N_3

EXAMPLE



```
IF(X-1Ø.)11,83,35
```

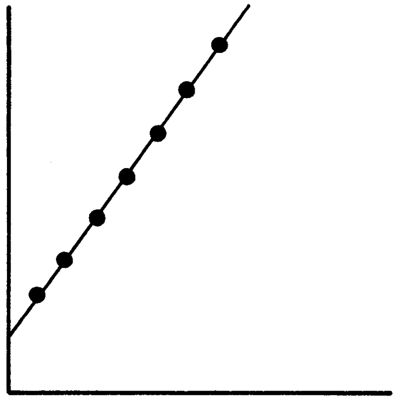
IT SAYS:

IF	X - 1Ø. < Ø	THEN GO TO STATEMENT	11
IF	X - 1Ø. = Ø	THEN GO TO STATEMENT	83
IF	X - 1Ø. > Ø	THEN GO TO STATEMENT	35

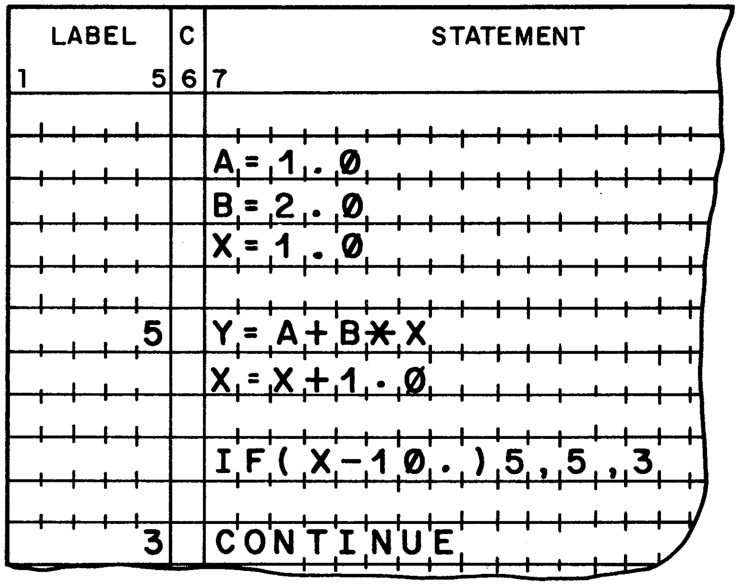
SAMPLE PROBLEM - USING THE "IF" STATEMENT

PROBLEM

SOLVE: Y = A + B ∗ X
WHERE: A = 1, B = 2
AND X = 1, 2,3,4,5,6,
7,8,9 & 1Ø



SOLUTION



LABEL 1–5	C 6	STATEMENT 7
		A = 1 . Ø
		B = 2 . Ø
		X = 1 . Ø
5		Y = A + B ∗ X
		X = X + 1 . Ø
		I F (X − 1 Ø .) 5 , 5 , 3
3		C O N T I N U E

NOTE: Y TAKES THE VALUES 3,5,7,9,11,13,15,17,19 & 21 ONLY!!

THE TWO BRANCH IF STATEMENT

GENERAL FORM:

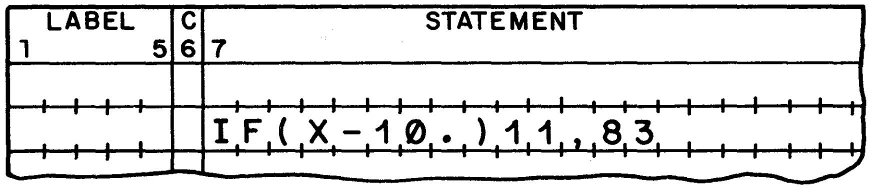
IF (E) N_1,N_2,

AN EXPRESSION

STATEMENT TO "GO TO" DEPENDING UPON THE EVALUATION OF E

IF E $< \emptyset$	**GO TO N_1**
IF E $\geq \emptyset$	**GO TO N_2**

EXAMPLE:



```
      IF(X-1Ø.)11,83
```

IT SAYS:

IF X - 1Ø. $<$ Ø THEN GO TO STATEMENT 11
IF X - 1Ø. $\geq$ Ø THEN GO TO STATEMENT 83

NOTE:
IF (E) N_1, N_2
HAS THE SAME EFFECT AS
IF (E) N_1, N_2, N_2

SUMMARY

SAMPLE PROBLEM

**WRITE A SET OF STATEMENTS TO PERFORM THE FOLLOWING:
READ TWO VALUES FROM THE KEYBOARD AND CALCULATE THE SUM.
IF THE SUM IS POSITIVE; TYPE OUT THE SUM.
IF THE SUM IS NEGATIVE; TYPE OUT THE WORD "REJECT".
PROGRAM A LOOP FOR CONTINUOUS PROBLEM SOLUTIONS.**

SAMPLE SOLUTION

LABEL 1 5	C 6	STATEMENT 7
5		READ(1,*)A,B
		SUM=A+B
		IF(SUM)1Ø,2Ø
2Ø		WRITE(2,1ØØ)SUM
1ØØ		FORMAT("SUM="F1Ø.4)
		GO TO 5
1Ø		WRITE(2,2ØØ)
2ØØ		FORMAT("REJECT")
		GO TO 5

RULES: **READ/WRITE statements: indicate the UNIT NO., FORMAT NO. and the DATA LIST elements. Using an (*) for the format no. of a READ statement indicates the FREE-FIELD INPUT mode.**

IF statements: TRANSFER CONTROL to one of TWO or THREE branches. The TWO branch IF represents NEGATIVE and NON-NEGATIVE. The THREE branch IF represents NEGATIVE, ZERO and POSITIVE NON-ZERO.

These two statements are equivalent:

IF (N) 1 Ø, 2 Ø   IF (N) 1 Ø, 2 Ø, 2 Ø

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS – EXECUTABLE TYPES
FUNCTIONS

CONTROL – THE PAUSE STATEMENT

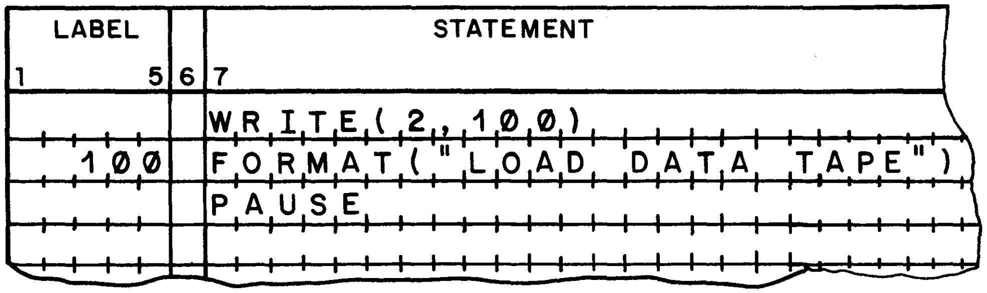
GENERAL FORM **PAUSE**

WHEN THIS STATEMENT IS EXECUTED THE COMPUTER WILL WRITE "PAUSE" ON THE TELEPRINTER AND THEN HALT. WHEN THE RUN BUTTON IS PUSHED THE COMPUTER WILL RESUME EXECUTION AT THE NEXT FORTRAN STATEMENT.

EXAMPLE

CONSIDER A PROGRAM WHICH MUST PROCESS DATA READ FROM A TAPE PLACED IN THE PHOTOREADER. AT SOME TIME THE FORTRAN PROGRAM MAY:

1. **WRITE A MESSAGE ON THE TELEPRINTER REQUESTING THE DATA TAPE.**
2. **PAUSE THE OPERATOR WOULD NOW PUT THE DATA TAPE IN THE PHOTOREADER, AND PRESS RUN.**



LABEL 1 5	6	STATEMENT 7
		WRITE(2,100)
100		FORMAT("LOAD DATA TAPE")
		PAUSE

SUBSCRIPT NOTATION

ANALYSIS:

IN ALGEBRA, SUBSCRIPTS ARE WRITTEN:

$a_1 X + a_2 X^2 + a_3 X^3 \ldots .$

IN FORTRAN, THE EQUIVALENT EXPRESSION WOULD BE:

```
A(1)*X+A(2)*X**2+A(3)*X**3 . . . .
```

GENERAL FORM:

$$\alpha \quad (I) \quad = \quad \beta$$

FIXED OR FLOATING POINT VARIABLE NAME → α

SUBSCRIPT → (I)

ANY EXPRESSION → β

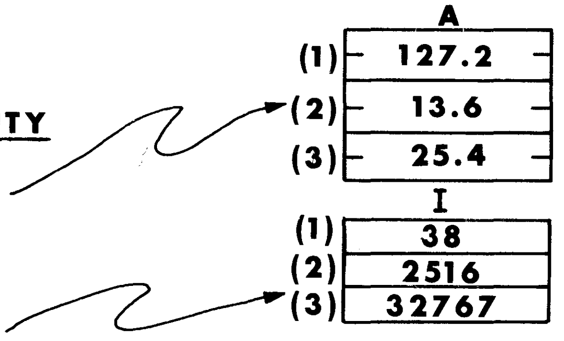
SUBSCRIPT

INTEGER CONSTANT, VARIABLE, OR CERTAIN EXPRESSIONS

- **A SET OF SUBSCRIPTED VARIABLE QUANTITIES IS CALLED AN ARRAY**
- **THE INDIVIDUAL QUANTITIES OF THE ARRAY ARE CALLED ELEMENTS**

EXAMPLE:

ARRAY NAME	ELEMENT NAME	QUANTITY
	A (1)	127.2
A	A (2)	13.6
	A (3)	25.4
	I (1)	38
I	I (2)	2516
	I (3)	32767



CONSTANTS
VARIABLES
OPERATORS
STATEMENTS – NON - EXECUTABLE
FUNCTIONS

SPECIFICATION — THE DIMENSION STATEMENT

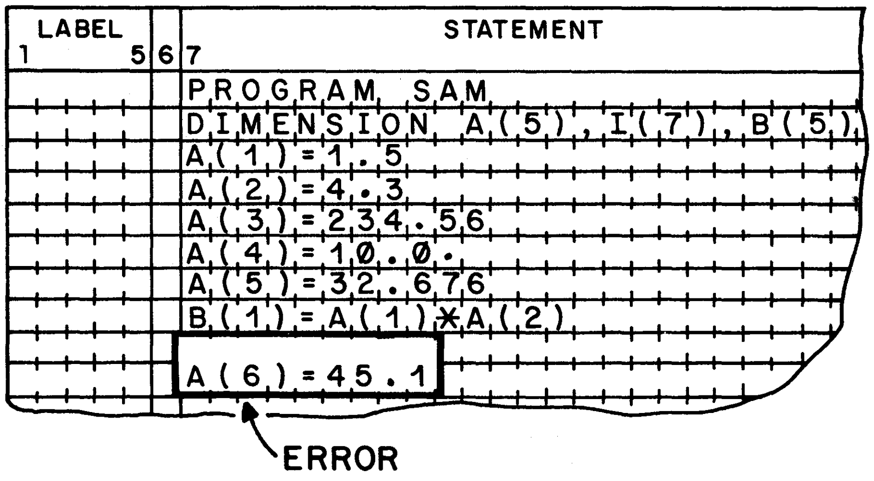
GENERAL FORM:

```
DIMENSION A(5), I(7), B(5)
```

VARIABLE NAME ↑ ↑ NUMBER OF ELEMENTS

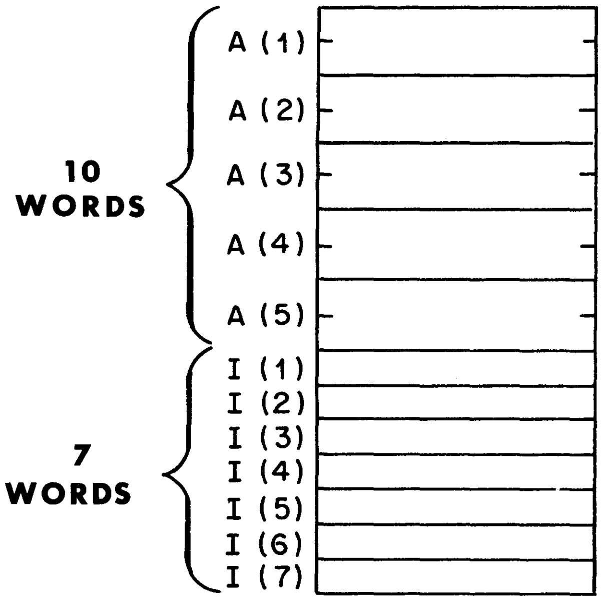
A DIMENSION STATEMENT MUST APPEAR BEFORE THE FIRST EXECUTABLE STATEMENT OF THE PROGRAM.

EXAMPLE:

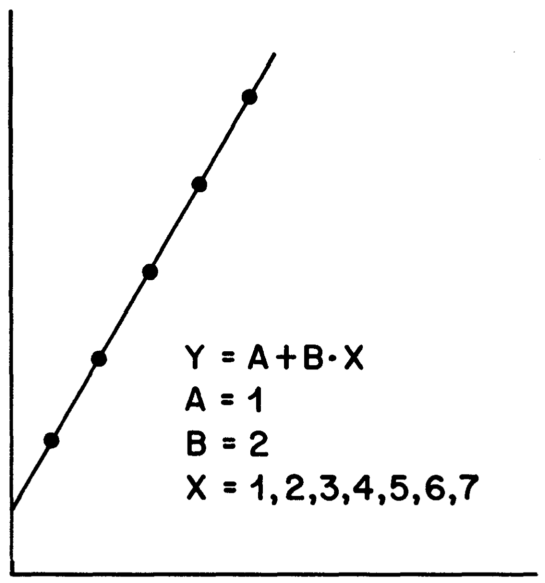


```
LABEL      STATEMENT
1    5 6 7
           PROGRAM SAM
           DIMENSION A(5), I(7), B(5)
           A(1)=1.5
           A(2)=4.3
           A(3)=234.56
           A(4)=10.0.
           A(5)=32.676
           B(1)=A(1)*A(2)
           A(6)=45.1
```

ERROR



SUBSCRIPTED VARIABLES IN FORTRAN ARITHMETIC STATEMENTS



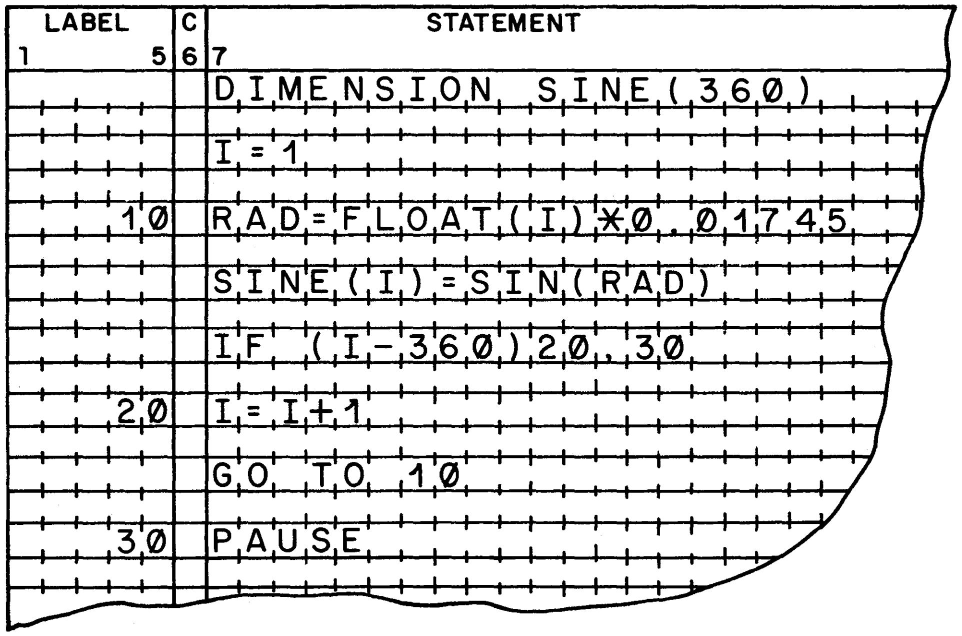
Y(1)	= 3
Y(2)	= 5
Y(3)	= 7
Y(4)	= 9
Y(5)	= 11
Y(6)	= 13
Y(7)	= 15

```
LABEL  C  STATEMENT
1    5 6 7
         PROGRAM DEMO4
         DIMENSION Y(7)
         A=1.Ø
         B=2.Ø
         X=1.Ø
         I=1
   1Ø    Y(I)=A+B*X
         IF(I-7)2Ø,3Ø
   2Ø    I=I+1
         X=X+1.Ø
         GO TO 1Ø
   3Ø    PAUSE
```

A SAMPLE PROBLEM — COMPUTE THE SINE FOR 36Ø ANGLES

PROBLEM: FILL AN ARRAY WHICH HAS 36Ø ELEMENTS WITH THE TRIGONOMETRIC SINE OF THE NUMBER OF DEGREES CORRESPONDING TO THAT ELEMENT.

SOLUTION:

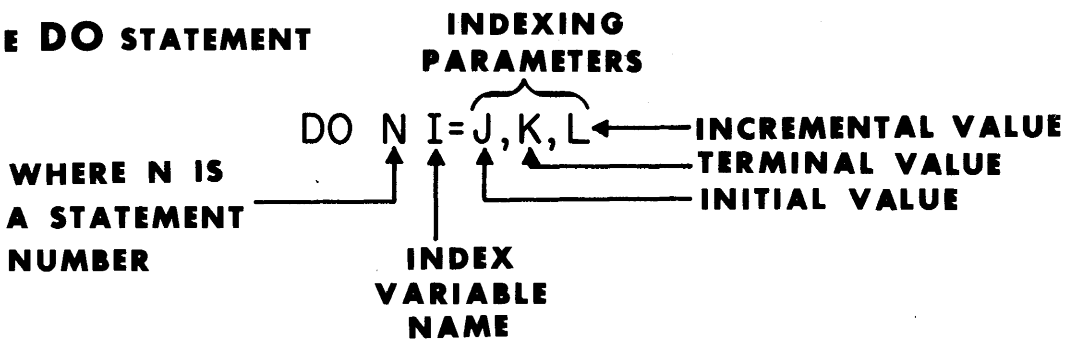


```
LABEL  C  STATEMENT
1    5 6  7
          DIMENSION SINE(36Ø)
          I=1
   1Ø     RAD=FLOAT(I)*Ø.Ø1745
          SINE(I)=SIN(RAD)
          IF (I-36Ø)2Ø,3Ø
   2Ø     I=I+1
          GO TO 1Ø
   3Ø     PAUSE
```

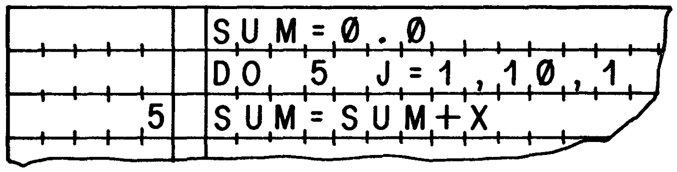
CONSTANTS
VARIABLES
OPERATORS
STATEMENTS - EXECUTABLE
FUNCTIONS

CONTROL — THE DO STATEMENT

GENERAL FORM



EXAMPLE:



```
      SUM=Ø.Ø
      DO 5 J=1,1Ø,1
    5 SUM=SUM+X
```

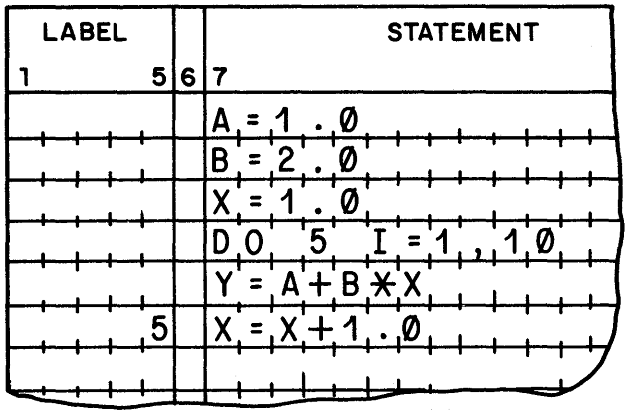
- SET J = 1, (INITIAL VALUE). DO ALL STATEMENTS DOWN TO AND INCLUDING STATEMENT 5 (THE RANGE OF THE "DO").
- ADD 1 (INCREMENTAL VALUE) TO J.
- IF J ≤ 10, (TERMINAL VALUE) RE-EXECUTE THE STATEMENTS IN THE RANGE.
- IF J > 10, THE "DO" IS SATISFIED AND CONTROL PASSES TO THE NEXT STATEMENT IN SEQUENCE.

USING THE DO PROBLEM #1

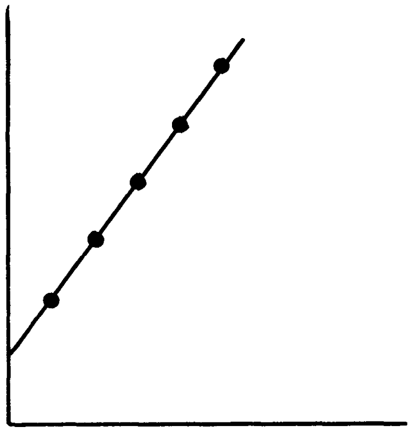
PROBLEM: SOLVE : $Y = A + B \cdot X$

Where $A = 1$ $B = 2$
And $X = 1,2,3,4,5,6,7,8,9,10$

SOLUTION:



LABEL 1–5	6	STATEMENT 7
		A=1.Ø
		B=2.Ø
		X=1.Ø
		DO 5 I=1,1Ø
		Y=A+B*X
5		X=X+1.Ø



NOTE: IF THE INCREMENTAL VALUE OF THE "DO" IS 1, IT NEED NOT BE SPECIFIED.

USING THE DO — PROBLEM #2

PROBLEM:

ASSUME THAT ARRAY X CONTAINS 1ØØ VALUES. FIND THE SUM.

SOLUTION:

LABEL 1 5	C 6	STATEMENT 7
		DIMENSION X(1ØØ)
		SUM=Ø.Ø
		DO 1Ø I=1,1ØØ
1Ø		SUM=SUM+X(I)
		PAUSE

RULES:

- **THE LABEL WHICH TERMINATES THE DO LOOP IS ON A STATEMENT WHICH IS PART OF THE LOOP**
- **THE LABEL WHICH TERMINATES THE DO LOOP MAY NOT BE ON A "GO TO" , "IF","RETURN", "STOP", "PAUSE", OR "DO" STATEMENT.**

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS – EXECUTABLE
FUNCTIONS

CONTROL—THE CONTINUE STATEMENT

GENERAL FORM: *CONTINUE*

THE CONTINUE STATEMENT IS A DUMMY FORTRAN STATEMENT WHICH IS USED TO PROVIDE A LEGAL TERMINATION FOR A "DO" LOOP WHEN THE LAST STATEMENT WOULD OTHERWISE BE A :
GO TO, IF, RETURN, PAUSE, STOP OR ANOTHER "DO" STATEMENT.

EXAMPLE:

INVALID "DO" LOOP

```
      SUM=Ø.Ø
      DO 1Ø I=1,1ØØ
      SUM=SUM+DELTA
1Ø    IF(SUM-5ØØ.)1Ø,2Ø,2Ø
2Ø    J=J+1
```

VALID "DO" LOOP

```
      SUM=Ø.Ø
      DO 1Ø I=1,1ØØ
      SUM=SUM+DELTA
      IF(SUM-5ØØ.)1Ø,2Ø,2Ø
1Ø    CONTINUE
2Ø    J=J+1
```

USING THE DO PROBLEM #3

PROBLEM: WRITE a FORTRAN program which reads 1ØØ numbers from the photoreader and calculates the sum, average, and RMS value.

GIVEN: The RMS value = the square root of the sum of the squares of the difference between the mean value and the individual data points:

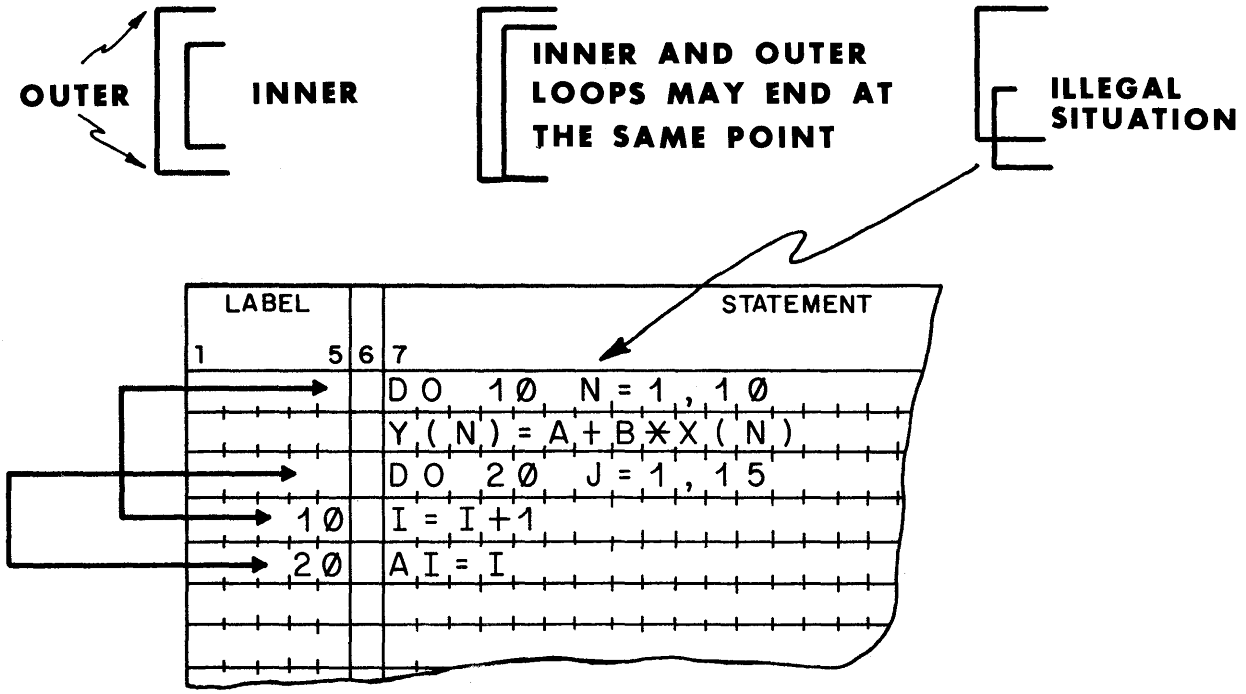
$$RMS = \sqrt{\sum_{i=1}^{100} \frac{(X_i-\overline{X})^2}{100}}$$

SOLUTION:

```
      PROGRAM DEMO6
      DIMENSION X(1ØØ)
    5 WRITE(2,1)
    1 FORMAT("INSERT DATA TAPE IN P.R., PUSH RUN")
      PAUSE
      SUM=Ø.Ø
      DO 2 I=1,1ØØ
         READ(5,*)X(I)
         SUM=SUM+X(I)
    2 CONTINUE
      AVG=SUM/1ØØ.
      SUMSQ=Ø.Ø
      DO 3 I=1,1ØØ
    3    SUMSQ=SUMSQ+(X(I)-AVG)**2
      RMS=SQRT(SUMSQ/1ØØ.)
      WRITE(2,4)SUM,AVG,RMS
    4 FORMAT("SUM="F1Ø.3/"AVG="F1Ø.3/"RMS="F1Ø.3///)
      PAUSE
      GO TO 5
      END
```

NESTED DO LOOPS

RULE: It is permissible for one "DO" loop to contain another "DO" loop within its range; however, care must be given to avoid illegal transfers. Generally, it is required that all statements in the range of the inner "DO" also be within the range of the outer "DO".



```
      DO 1Ø  N=1,1Ø
      Y(N)=A+B*X(N)
      DO 2Ø  J=1,15
   1Ø I=I+1
   2Ø AI=I
```

FORTRAN programming techniques

V

LESSON V
FORTRAN PROGRAMMING TECHNIQUES

LESSON V OBJECTIVES

TO INTRODUCE SOME ADDITIONAL CAPABILITIES OF FORTRAN.

THESE INCLUDE:

1 - SOME ADDITIONAL FORMAT SPECIFICATIONS FOR MORE INPUT/OUTPUT FLEXIBILITY.

2 - MORE "FREE FIELD" INPUT CAPABILITIES.

3 - TWO DIMENSIONAL ARRAYS

4 - ARRAY INPUT/OUTPUT TECHNIQUES

5 - SUBROUTINES AND FUNCTION SUBPROGRAMS.

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS - NON EXECUTABLE
FUNCTIONS

FORMATS — THE E FORMAT

GENERAL FORM

E w.d

- E — SPECIFIES E FORMAT
- w — FIELD WIDTH INCLUDING SIGNS, DECIMAL POINT, AND EXPONENT. **NOTE:** PROPER OUTPUT WILL **ALWAYS** RESULT IF $w \geq d+7$
- d — NUMBER OF DIGITS TO THE RIGHT OF THE DECIMAL POINT

OUTPUT EXAMPLES

NUMBER	FORMAT	PRINTED AS	REMARKS
+10.4365	E1Ø.3	^^.1Ø4E+Ø2	
−12.34	E12.3	^^^−.123E+Ø2	
−1Ø.4365	E7.5	$$$$$$$	$w<d+7$

INPUT EXAMPLES

NUMBER	FORMAT	CONVERTED VALUE	REMARKS
+1.2345E2	E9.3	123.45	Decimal point overides format
1234	E4.2	12.34	Format inserts decimal point

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS — NON-EXECUTABLE
FUNCTIONS

FORMATS — THE FORMAT REPEAT FACTOR

A FORMAT SPECIFICATION CAN BE USED "n" TIMES BY USING THE REPEAT FACTOR IN FRONT OF THE SPECIFICATION, AND PROVIDING THE PROPER PARENTHESES TO EFFECT THE DESIRED REPEAT COUNT.

EXAMPLE 1

```
  1ØØ  FORMAT(F1Ø.2,I5,F1Ø.2,I5,F1Ø.2,I5)
  2ØØ  FORMAT(3(F1Ø.2,I5))
```

THE TWO FORMATS SHOWN ABOVE ARE EQUIVALENT.

EXAMPLE 2

WRITE ON THE TELEPRINTER:
THE TIME IS XX:XX:XX.

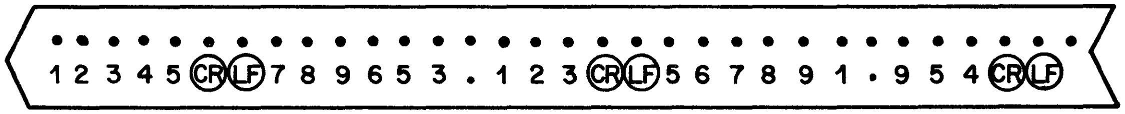
```
       WRITE(2,1Ø)IHH,IMM,ISS
   1Ø  FORMAT("THE TIME IS",I2,2(":",I2))
```

CONTENTS
VARIABLES
OPERATORS
STATEMENTS – NON-EXECUTABLE
FUNCTIONS

FORMATS THE SLASH (/) IN A FORMAT

THE **SLASH (/)** CHARACTER IS USED TO INDICATE THE END OF ONE RECORD AND THE BEGINNING OF ANOTHER. THE CLOSING PARENTHESIS OF THE FORMAT STATEMENT INDICATES THE END OF THE INPUT/OUTPUT OPERATION.

EXAMPLE


1 2 3 4 5 CR LF 7 8 9 6 5 3 . 1 2 3 CR LF 5 6 7 8 9 1 . 9 5 4 CR LF

THESE STATEMENTS

PRODUCE THIS RESULT

```
LABEL  C  STATEMENT
1    5 6 7
         READ(5,102)I,A,B
  102    FORMAT(I5/F10.3/F10.3)

         WRITE(2,1024)I
 1024    FORMAT("THE VALUE OF I IS"
       1  I6/"INPUT TEST OVER")
```

```
THE VALUE OF I IS 12345
INPUT TEST OVER
```

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS – NON EXECUTABLE
FUNCTIONS

FORMATS – THE X FORMAT

THE X FORMAT IS USED TO INSERT SPACES IN OUTPUT DATA AND CAN BE USED TO IGNORE ALPHA CHARACTERS AND PUNCTUATION MARKS ON INPUT.

GENERAL FORM:

REPEAT COUNT X IDENTIFIER

```
  100 FORMAT(F10.2,5X,I2,3X,I5/)
```

EXAMPLE

AN INPUT DATA TAPE HAS THE FOLLOWING FORM:

WEIGHT ∧∧ 1Ø ∧∧ PRICE ∧∧ $1.98 ∧∧ TOTAL ∧∧ $19.8Ø

THE FORTRAN STATEMENTS SHOWN WILL CAUSE THE PUNCTUATION MARKS TO BE IGNORED.

LABEL 1 5	6	STATEMENT 7
		READ(5,1Ø4)I,A,B
1Ø4		FORMAT(8X,I2,1ØX,F4.2,1ØX,F5.2)

RESULT: I CONTAINS 1Ø, A CONTAINS 1.98, B CONTAINS 19.8Ø

CONSTANTS
VARIABLES
OPERATORS
STATEMENTS — NON-EXECUTABLE
FUNCTIONS

FORMATS - THE A FORMAT

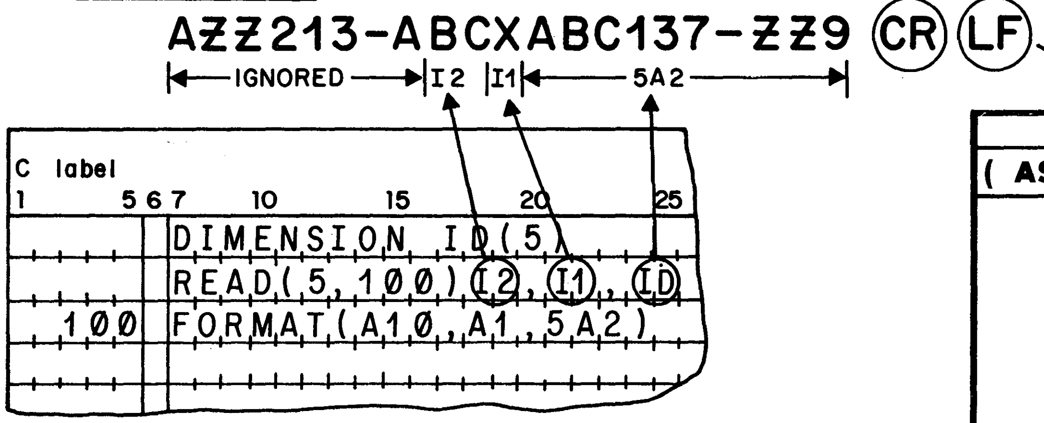
THE A FORMAT CAUSES ALPHANUMERIC DATA ON AN EXTERNAL MEDIUM TO BE TRANSLATED TO OR FROM ASCII FORM IN MEMORY.

GENERAL FORM: **r A w**

REPEAT COUNT — r; IDENTIFIER — A; FIELD — w

FOR EXAMPLE: ASSUME AN INPUT DATA TAPE CONTAINS:

AƵƵ213-ABCXABC137-ƵƵ9 (CR) (LF)

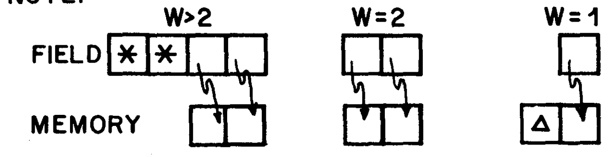


```
      DIMENSION ID(5)
      READ(5,1ØØ)I2,I1,ID
  1ØØ FORMAT(A1Ø,A1,5A2)
```

RESULTS
(ASCII DATA IN MEMORY)

I2	B	C
I1	Ø	X
ID	A	B
	C	1
	3	7
	-	Ƶ
	Ƶ	9

NOTE:



* IGNORED ON INPUT
SPACES ON OUTPUT

△ IGNORED ON OUTPUT
ZERO ON INPUT

ADDITIONAL FREE-FIELD INPUT CAPABILITY

FREE-FIELD INPUT permits the reading of ASCII numeric data items without a definitive format statement. Special symbols included with the data items direct the formatting.

SPECIAL SYMBOLS:

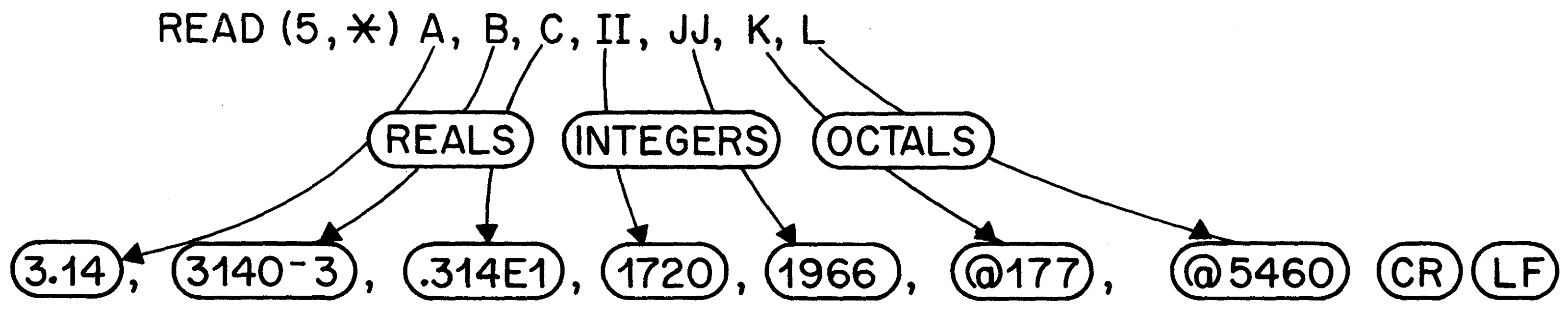
(SPACE) (,)	=	DATA ITEM DELIMITERS
(/)	=	RECORD TERMINATOR
(+)(−)	=	SIGN OF ITEM
(.)(E)(+)(−)	=	FLOATING POINT NUMBER
(@)	=	OCTAL INTEGER
("...")	=	COMMENTS

RULES:

- **FREE-FIELD is indicated when an ASTERISK is used instead of a format statement number in the READ statement.**
- **A DATA ITEM is any continuous string of numeric and special symbols occurring between two commas, a comma and a space or two spaces. The data value corresponds to a list element.**
- **Two consecutive commas indicate that no data item is supplied for the corresponding list element. The current value of the list element in memory is unchanged.**
- **An initial comma indicates the first list element is to be skipped.**
- **A CR LF terminates each input data line record.**

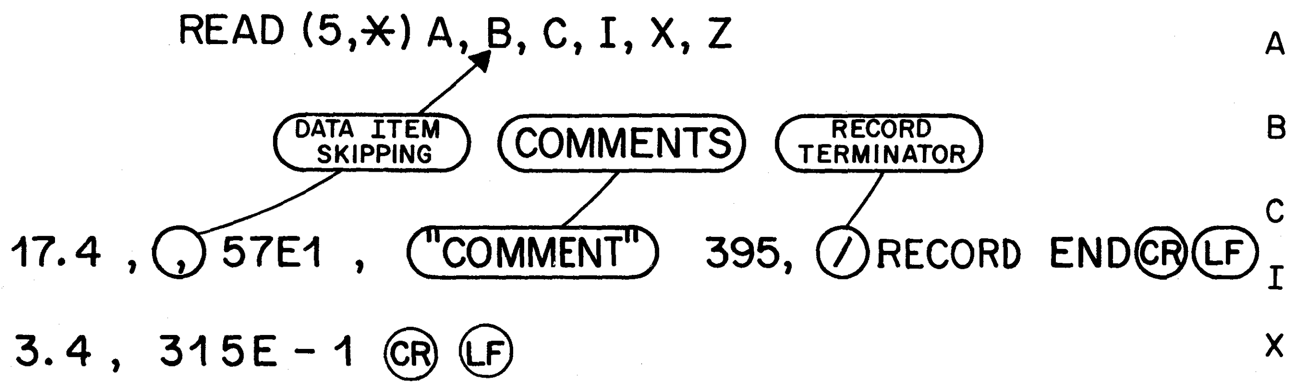
EXAMPLES : FREE-FIELD INPUT

#1 - DATA TYPES



```
READ (5,*) A, B, C, II, JJ, K, L

3.14, 3140-3, .314E1, 1720, 1966, @177,  @5460 CR LF
```

#2 - SPECIAL SYMBOL USAGE



```
READ (5,*) A, B, C, I, X, Z

17.4 , , 57E1 , "COMMENT"  395, /RECORD END CR LF
3.4 , 315E-1 CR LF
```

	MEMORY RESULTS
A	17.4
B	UNCHANGED
C	570.
I	395
X	3.4
Z	31.5

MORE INFORMATION ON SUBSCRIPTS

- IN HP FORTRAN A VARIABLE MAY HAVE ONE OR <u>TWO</u> SUBSCRIPTS.

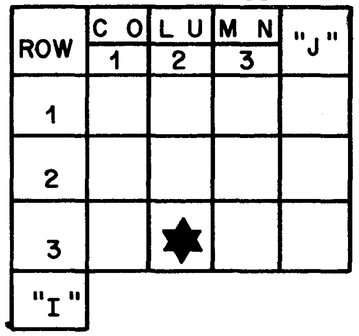
  FOR EXAMPLE: Y (I) = A + B -C OR X (I,J) = A + B -C

- THE SECOND SUBSCRIPT IMPLIES THAT THE ARRAY HAS TWO DIMENSIONS.
- A TWO DIMENSIONAL ARRAY CAN BE VISUALIZED USING <u>ROWS</u> AND <u>COLUMNS.</u>
- THE LEFT SUBSCRIPT YIELDS THE <u>ROW</u> NUMBER.
- THE RIGHT SUBSCRIPT YIELDS THE <u>COLUMN</u> NUMBER.
- THE SIZE OF A TWO DIMENSIONAL ARRAY IS EQUAL TO THE PRODUCT OF THE SUBSCRIPTS

<u>EXAMPLE</u>

```
LABEL     C  STATEMENT
1      5  6  7
             DIMENSION X(3,3)
             I = 3
             J = 2
             X(I,J)=SQRT(ALPHA+GAMMA)
```

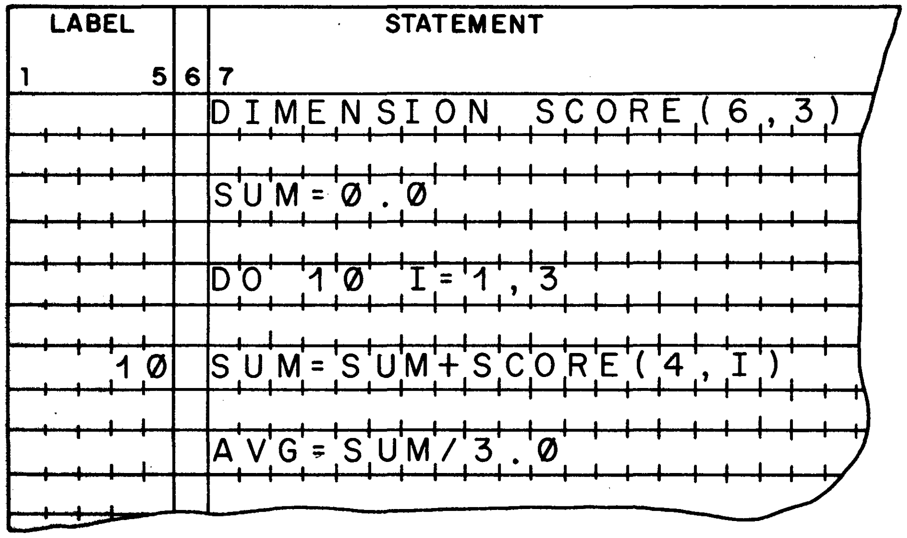
2 DIMENSIONAL
ARRAY "X"



SAMPLE PROBLEM: USING A TWO DIMENSIONAL ARRAY

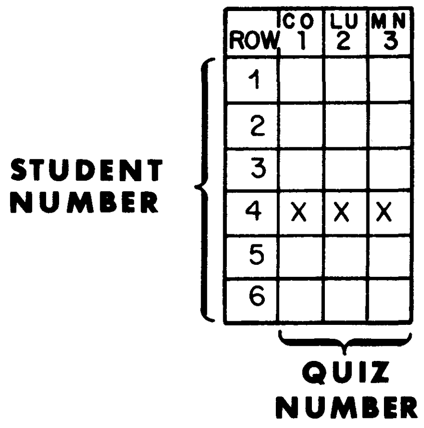
PROBLEM: Assume the existence of a two dimensional array SCORE (6,3) which is filled out to contain the scores of 6 students for each of 3 quizzes. Write a series of statements which finds the average score of student 4.

SOLUTION:



```
LABEL    STATEMENT
1   5 6 7
        DIMENSION SCORE(6,3)

        SUM=Ø.Ø

        DO 1Ø I=1,3

   1Ø   SUM=SUM+SCORE(4,I)

        AVG=SUM/3.Ø
```

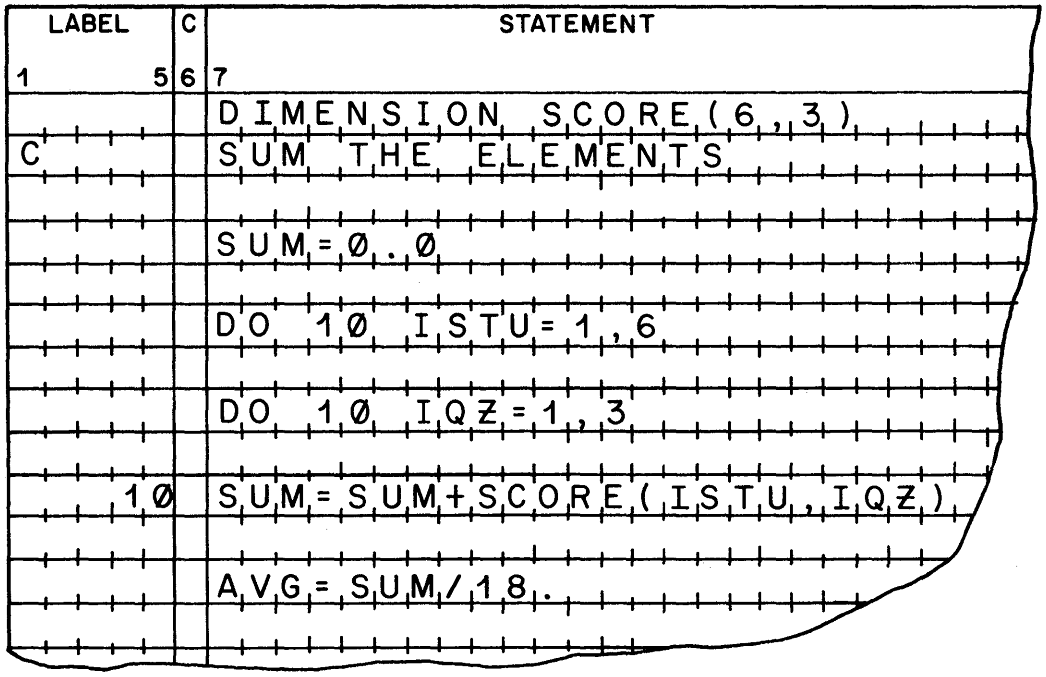
ARRAY SCORE



ROW	CO 1	LU 2	MN 3
1			
2			
3			
4	X	X	X
5			
6			

SAMPLE PROBLEM – USING "NESTED DO LOOPS"

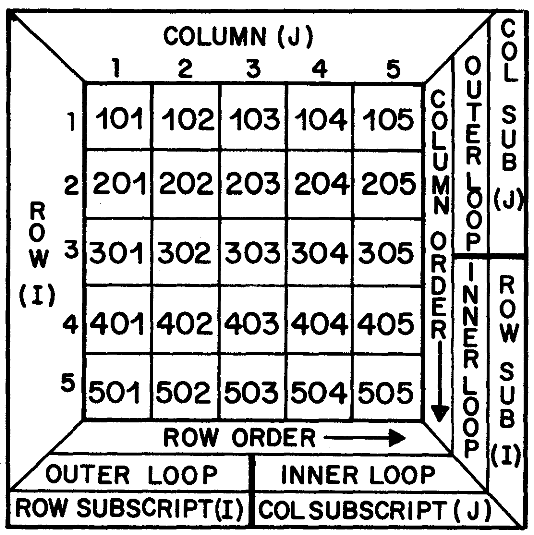
Assume the existence of a two dimensional array SCORE (6, 3) which is filled out with the scores of 6 students for each of three quizzes. Find the composite average.



```
LABEL |C| STATEMENT
1    5|6|7
      DIMENSION SCORE(6,3)
C     SUM THE ELEMENTS

      SUM=Ø.Ø

      DO 1Ø ISTU=1,6

      DO 1Ø IQZ=1,3

   1Ø SUM=SUM+SCORE(ISTU,IQZ)

      AVG=SUM/18.
```

2 DIMENSIONAL ARRAY CONVENTIONS

ARRAY IA



	1	2	3	4	5
1	101	102	103	104	105
2	201	202	203	204	205
3	301	302	303	304	305
4	401	402	403	404	405
5	501	502	503	504	505

RULE:

TO FILL A TWO DIMENSIONAL ARRAY IN COLUMN ORDER THE ROW SUBSCRIPT VARIES MOST RAPIDLY.

RULE:

TO FILL A TWO DIMENSIONAL ARRAY IN ROW ORDER THE COLUMN SUBSCRIPT VARIES MOST RAPIDLY.

PROBLEM:

FILL ARRAY "IA" IN COLUMN ORDER.

SOLUTION:

```
      PROGRAM ARRAY
      DIMENSION IA(5,5)
       DO 1ØØ J=1,5
       DO 1ØØ I=1,5
1ØØ    IA(I,J)=1ØØ*I+J
```

ARRAY IA IN MEMORY
101
201
301
401
501
102
202
302
402
502
103
203
303
403
503
104
204
304
404
504
105
205
305
405
505

SUMMARY:

TRANSMIT ARRAY	OUTER LOOP CONTROLS	INNER LOOP CONTROLS
BY COLUMN	COLUMN SUBSCRIPT (J)	ROW SUBSCRIPT (I)
BY ROW	ROW SUBSCRIPT (I)	COLUMN SUBSCRIPT (J)

INPUT/OUTPUT OF ENTIRE ARRAYS IN NATURAL ORDER

- **When a Dimensioned variable name is used in an input-output list without subscripts, the entire array is transmitted.**

  Example: **write the contents of array IA in natural order.**

PROGRAM

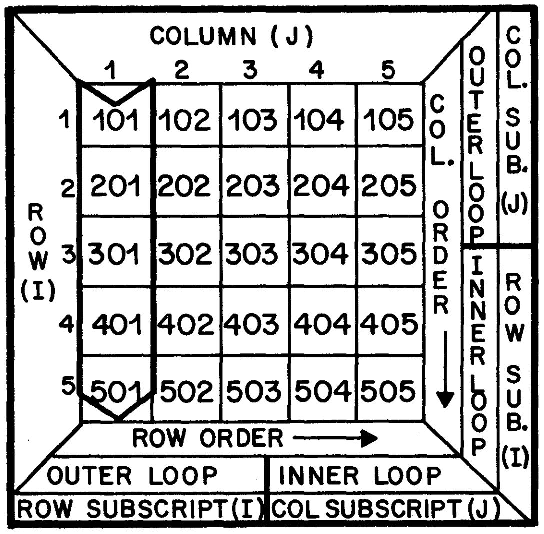
```
      PROGRAM DEMO1
      DIMENSION IA(5,5)
      WRITE(2,100)IA
100   FORMAT(I6)
```

OUTPUT DATA

```
101
201
301
401
501
102
202
302
402
502
103
203
303
403
503
104
204
304
404
504
105
205
305
405
505
```

- **The array is output in column order from sequential memory locations-THIS IS NATURAL ORDER**

- **The single format specification is used repeatedly until all elements of the array are transmitted**



	1	2	3	4	5
1	101	102	103	104	105
2	201	202	203	204	205
3	301	302	303	304	305
4	401	402	403	404	405
5	501	502	503	504	505

ARRAY IA IN MEMORY
101
201
301
401
501
102
202
302
402
502
103
203
303
403
503
104
204
304
404
504
105
205
305
405
505

INPUT/OUTPUT OF ENTIRE ARRAYS IN ROW ORDER

- **If natural order (column) is not to be used, subscripting information must be provided.**

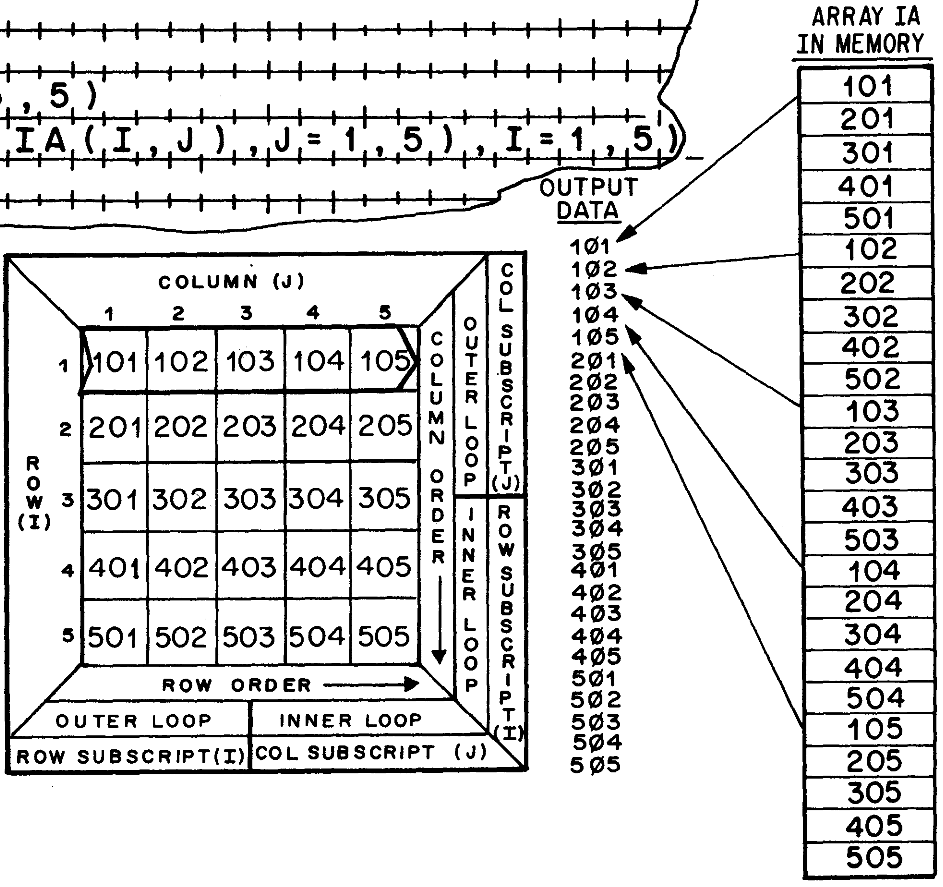
  Example: write the contents of array IA in row order.

PROGRAM:

```
      PROGRAM DEMO2
      DIMENSION IA(5,5)
      WRITE(2,1Ø1)((IA(I,J),J=1,5),I=1,5)
1Ø1   FORMAT(I6)
```

- **The array is output in row order from non-sequential memory locations.**

- **The single format specification is used repeatedly until all elements of the array are transmitted.**



```
OUTPUT
 DATA
1Ø1
1Ø2
1Ø3
1Ø4
1Ø5
2Ø1
2Ø2
2Ø3
2Ø4
2Ø5
3Ø1
3Ø2
3Ø3
3Ø4
3Ø5
4Ø1
4Ø2
4Ø3
4Ø4
4Ø5
5Ø1
5Ø2
5Ø3
5Ø4
5Ø5
```

SAMPLE PROBLEM : FINDING THE LARGEST ELEMENT IN AN ARRAY

CASE 1 A ONE DIMENSIONAL ARRAY X(232) FIND THE LARGEST VALUE IN ARRAY X

SOLUTION: ASSUME THAT THE FIRST IS THE LARGEST ELEMENT. COMPARE THE SECOND NUMBER WITH THE FIRST. IF THE SECOND IS LARGER THAN THE FIRST, EXCHANGE VALUES. IF THE FIRST WAS LARGER, COMPARE THE THIRD VALUE WITH THE FIRST, ETC.

```
      BIG=X(1)
      DO 1Ø I=2,232
      IF(BIG-X(I))5,1Ø
    5 BIG=X(I)
   1Ø CONTINUE
```

CASE 2 A TWO DIMENSIONAL ARRAY Y (25,34) FIND THE LARGEST VALUE IN ARRAY Y

SOLUTION: SAME TECHNIQUE AS ABOVE

```
      BIG=Y(1,1)
      DO 1Ø I=1,25
      DO 1Ø J=1,34
      IF(BIG-Y(I,J))5,1Ø
    5 BIG=Y(I,J)
   1Ø CONTINUE
```

SUBSCRIPT EXPRESSIONS

ALLOWABLE FORMS:

A SUBSCRIPTED EXPRESSION IS RESTRICTED TO THE FOLLOWING ALLOWABLE FORMS.

n
i
m*i
i+n
i−n
m*i+n
m*i−n

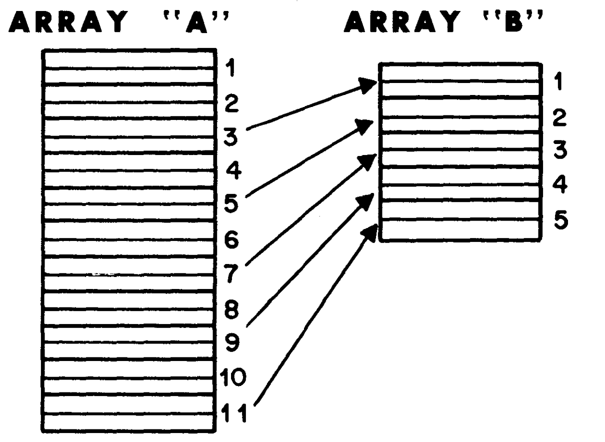
THE VALUE OF THE SUBSCRIPT EXPRESSION, EVEN WITHOUT THE ADDED OR SUBTRACTED CONSTANTS, MUST NEVER BE LESS THAN 1, OR GREATER THAN THE VALUE SPECIFIED WITHIN THE DIMENSION STATEMENT.

where m and n → **REPRESENT INTEGER CONSTANTS**
and i → **REPRESENTS AN INTEGER VARIABLE**

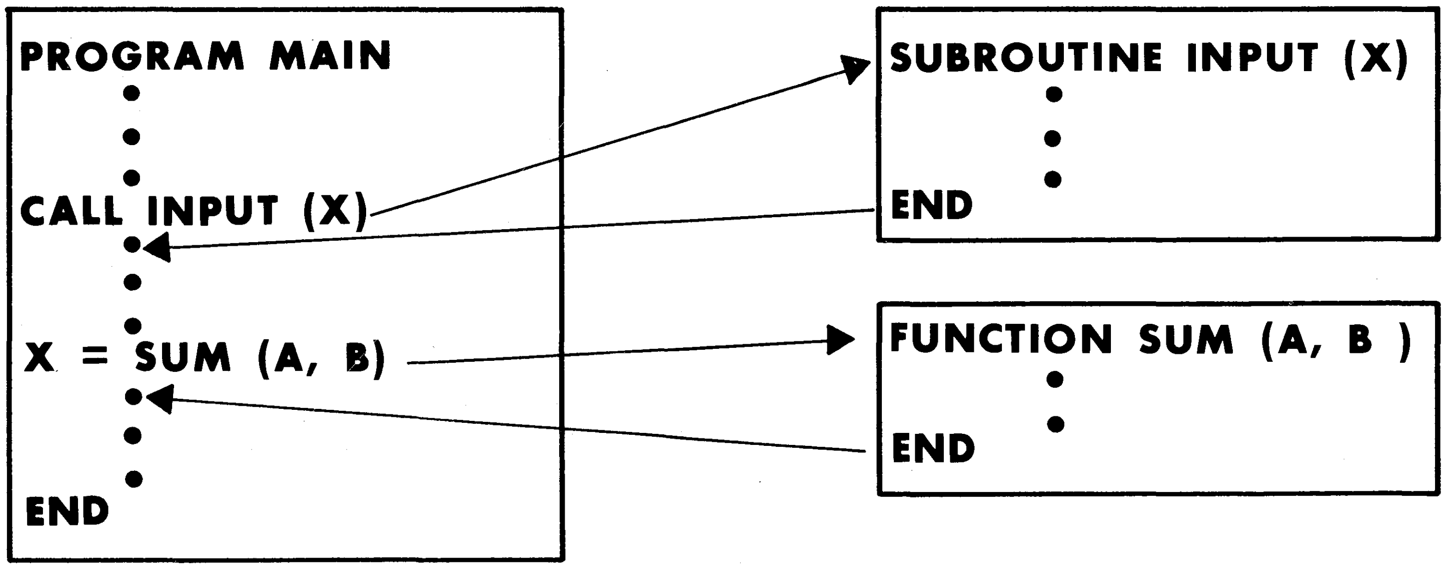
EXAMPLE: **ASSUME THAT WE WANT TO TAKE THE 3, 5, 7, 9, 11, ... elements out of array A and put them into the 1, 2, 3, 4, 5 elements of array B.**

SOLUTION:

LABEL 1 – 5	C 6	STATEMENT 7
		I = 1
20		B(I) = A(2*I+1)
		I = I+1
		IF(I-5)20,20,10
10		PAUSE



INTRODUCTION TO SUBROUTINE AND FUNCTION SUBPROGRAMS



- **SUBROUTINES** and **FUNCTIONS** are subprograms that are **EXTERNAL** to the main program and perform a specific operation.

- **SUBROUTINES** may or may not require parameter (argument) data.

- **FUNCTIONS** must have at least one parameter and produce a single value that is returned in the function name.

SUBROUTINE SUBPROGRAMS

SUBROUTINE DECLARATION

GENERAL FORM: **SUBROUTINE B λλλλ (a_1, a_2, a_3, . . . , a_n)**

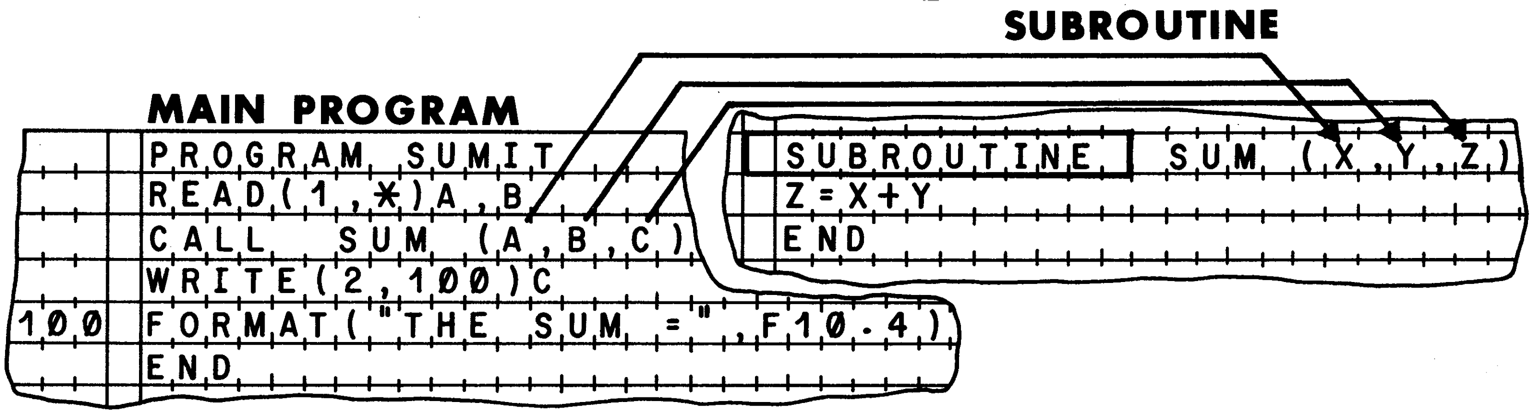
SUBROUTINE NAME **DUMMY PARAMETERS (OPTIONAL)**

SUBROUTINE CALL

GENERAL FORM: **CALL B λλλλ (A_1, A_2, A_3, . . . , A_n)**

ACTUAL PARAMETERS (OPTIONAL)

EXAMPLE:



```
      PROGRAM SUMIT
      READ(1,*)A,B
      CALL SUM (A,B,C)
      WRITE(2,100)C
100   FORMAT("THE SUM =",F10.4)
      END
```

```
      SUBROUTINE SUM (X,Y,Z)
      Z=X+Y
      END
```

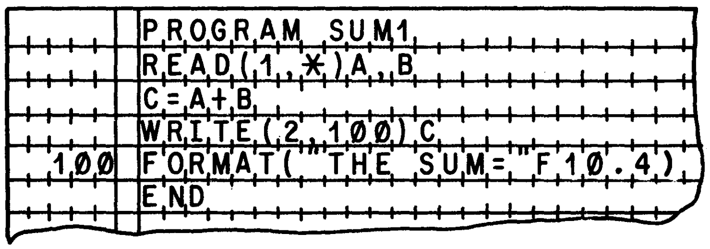
THE "CALL SUM" TRANSFERS CONTROL FROM THE MAIN PROGRAM TO THE SUBROUTINE.
THE ADDRESSES OF THE ACTUAL ARGUMENTS IN THE MAIN PROGRAM REPLACE THE DUMMY ARGUMENTS OF THE SUBROUTINE.
THE SUBROUTINE OBTAINS ARGUMENT VALUES (A, B) FROM THE MAIN PROGRAM, COMPUTES A + B AND STORES (C) IN THE MAIN PROGRAM.

SUBROUTINE EXAMPLES

THE PROGRAMS SHOWN WILL PRODUCE IDENTICAL RESULTS

EXAMPLE 1

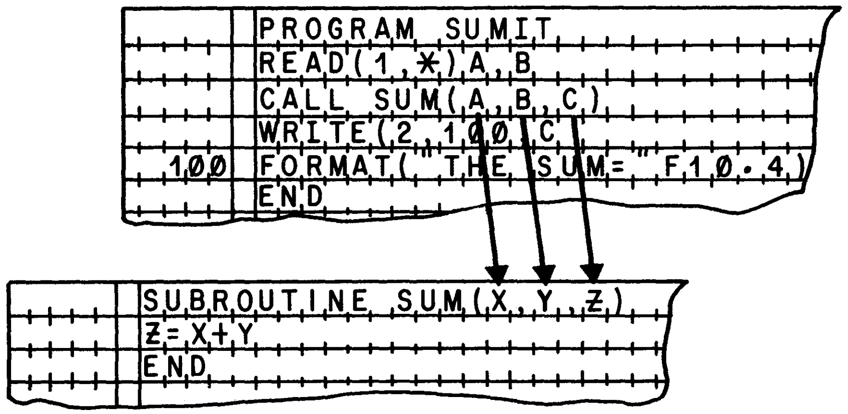
IN LINE



```
      PROGRAM SUM1
      READ(1,*)A,B
      C=A+B
      WRITE(2,1ØØ)C
  1ØØ FORMAT("THE SUM="F1Ø.4)
      END
```

EXAMPLE 2

EXTERNAL



```
      PROGRAM SUMIT
      READ(1,*)A,B
      CALL SUM(A,B,C)
      WRITE(2,1ØØ)C
  1ØØ FORMAT("THE SUM="F1Ø.4)
      END
```

```
      SUBROUTINE SUM(X,Y,Z)
      Z=X+Y
      END
```

ACTUAL ARGUMENTS (A,B,C) "REPLACE" DUMMY ARGUMENTS (X,Y,Z) WHICH ARE USED TO WRITE THE SUBROUTINE

A SAMPLE SUBROUTINE PROBLEM

PROBLEM: Write a program which reads two numbers from the Teletype and considers them R and THETA. It calls SUBROUTINE CONVT (R, THETA, X,Y) which returns X and Y as rectangular components of the polar vector R, THETA. Assume that THETA is in radians. The program should write the value of X and Y on the Teleprinter. Write the subroutine.

SOLUTION:

```
      PROGRAM DEMOS
    1 READ(1,*)R,THETA
      CALL CONVT(R,THETA,X,Y)
      WRITE(2,2)X,Y
    2 FORMAT("(X,Y) =",2F1Ø.3)
      END

      SUBROUTINE CONVT(RAD,ANGLE,XCORD,YCORD)
      XCORD=RAD*COS(ANGLE)
      YCORD=RAD*SIN(ANGLE)
      END
```

- **R, THETA ARE INPUT PARAMETERS**
- **X,Y ARE OUTPUT PARAMETERS**

FUNCTION SUBPROGRAMS

FUNCTION DECLARATION

GENERAL FORM: **FUNCTION B ΛΛΛΛ** $(a_1, a_2, a_3, \ldots, a_n) = e$

NAME OF THE FUNCTION **DUMMY PARAMETERS** **ARITHMETIC EXPRESSION**

FUNCTION CALL

GENERAL FORM: **Y = B ΛΛΛΛ** $(A_1, A_2, A_3, \ldots, A_n)$

(ACTUAL PARAMETERS)

MAIN PROGRAM

```
      PROGRAM ADDIT
      READ(1,*)A,B
      C=ADDF(A,B)
      WRITE(2,100)C
100   FORMAT("THE SUM =",F9.2)
      END
```

FUNCTION SUBPROGRAM

```
      FUNCTION ADDF(A,B)
      ADDF=A+B
      END
```

THE FUNCTION DECLARATION ESTABLISHES (A, B) TO BE DUMMY PARAMETERS.

THE PROGRAM CALLS THE FUNCTION AND PROVIDES ACTUAL PARAMETERS (A, B) THE (A, B) IN THE PROGRAM AND THE (A, B) IN THE FUNCTION ARE NOT THE SAME. THE MODE OF THE DUMMY PARAMETERS (REAL AND/OR INTEGER) USED IN THE FUNCTION DECLARATION MUST CORRESPOND TO THE MODE OF THE ACTUAL PARAMETERS USED IN THE FUNCTION CALL.

INTEGER AND REAL FUNCTIONS

- **INTEGER FUNCTIONS START WITH THE LETTERS I,J,K,L,M,N**
- **REAL FUNCTIONS START WITH THE LETTERS A-H AND O-Z**

EXAMPLE 1 **A FUNCTION TO FIND THE LARGEST OF TWO INTEGERS.**

```
      FUNCTION IBIG(J,K)
      IF(J-K)1Ø,2Ø
   1Ø IBIG=K
      RETURN
   2Ø IBIG=J
      END
```

CAUSES A RETURN TO THE CALLING PROGRAM.

EXAMPLE 2 **A FUNCTION THAT COMPARES AN INTEGER AGAINST A HIGH AND LOW LIMIT, AND:**

SETS $I = +1$ **IF DATA** $\geq$ **HIGH LIMIT**
$I = -1$ **IF DATA** $\leq$ **LOW LIMIT**
$I = Ø$ **IF DATA IS WITHIN LIMITS**

```
      FUNCTION ICMPR(XHI,XLO,DATA)
      I=1
      IF(DATA-XHI)1Ø,1ØØ,1ØØ
   1Ø I=Ø
      IF(DATA-XLO)2Ø,2Ø,1ØØ
   2Ø I=-1
  1ØØ ICMPR=I
      END
```

EXAMPLE 3 **A "REAL" FUNCTION TO COMPUTE THE HYPOTENUSE OF A RIGHT TRIANGLE.**

```
      FUNCTION HYPOT(X,Y)
      HYPOT=SQRT(X*X+Y*Y)
      END
```

A SAMPLE FUNCTION PROBLEM

THE PROGRAM SHOWN BELOW WILL EVALUATE A QUADRATIC EQUATION AND PRINT ONE ROOT IF THE DISCRIMINANT IS POSITIVE. THE FUNCTION DISC (A,B,C,I) EVALUATES SQRT (ABS(B*B—4.*A*C)) AND RETURNS:

I = 1 FOR A POSITIVE DISCRIMINANT
I = -1 FOR A NEGATIVE DISCRIMINANT

THE PROGRAM

```
      PROGRAM QUAD
    1 READ(5,*)A,B,C
      ROOT1=(-B+DISC(A,B,C,I))/(2.*A)
      IF(I)3,2
    2 WRITE(2,1ØØ)ROOT1
  1ØØ FORMAT("ROOT ONE =",E13.7)
    3 PAUSE
      GO TO 1
      END
```

THE FUNCTION

```
      FUNCTION DISC(A,B,C,I)
      X=B*B-4.*A*C
      I=1
      IF(X)1Ø,2Ø
   1Ø I=-1
      X=-X
   2Ø DISC=SQRT(X)
      END
```

NOTE: FUNCTIONS CAN RETURN MORE THAN ONE VALUE.

THE RESULTS OF FUNCTION "DISC" ARE RETURNED TO THE MAIN PROGRAM USING THE A & B REGISTERS WHILE "I" IS STORED IN THE MAIN PROGRAM USING THE PARAMETER LIST.

Introduction to HP computer hardware VI

LESSON VI
INTRODUCTION TO HP COMPUTER HARDWARE

LESSON VI OBJECTIVES

THIS LESSON IS A DISCUSSION OF THE HARDWARE CAPABILITIES OF HEWLETT PACKARD COMPUTERS, AND IS A PREREQUISITE TO ANY DISCUSSION OF THE HEWLETT PACKARD ASSEMBLER PRO– GRAM. LESSON VI WILL PROVIDE A KNOWLEDGE OF THE HARDWARE CAPABILITIES OF THE COMPUTER THAT IS ESSENTIAL TO THE ASSEMBLY LANGUAGE PROGRAMMER.

COMPUTER WORDS

COMPUTER WORDS ARE USED TO REPRESENT:

- <u>DATA WORDS</u> – *STORE DATA USED IN COMPUTATION SUCH AS: 5, 10, +32767, –32767.*
- <u>INSTRUCTION WORDS</u>-*ARE ORDERS THAT TELL THE MACHINE WHAT TO DO –SUCH AS ADD, SHIFT OR STORE DATA.*
- *<u>ADDRESS WORDS</u>* - *ARE USED TO SPECIFY A <u>15</u> BIT MEMORY ADDRESS VALUE IN THE RANGE 0–32767_{10}.*

FIVE BASIC WORD FORMATS -

ARE USED IN HP COMPUTERS TO REPRESENT INSTRUCTIONS, ADDRESSES AND DATA.

1. Memory Reference Instruction

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	OP				Z/C	WORD ADDRESS									

2. Register Reference Instruction

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP						MICRO OP									

3. Input-Output Instruction

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OP						SUB OP				SELECT CODE					

4. Full Address

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	PAGE ADDRESS					WORD ADDRESS									

5. Data (single-precision integer)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SIGN	INTEGER														

COMPUTING OPERATIONS

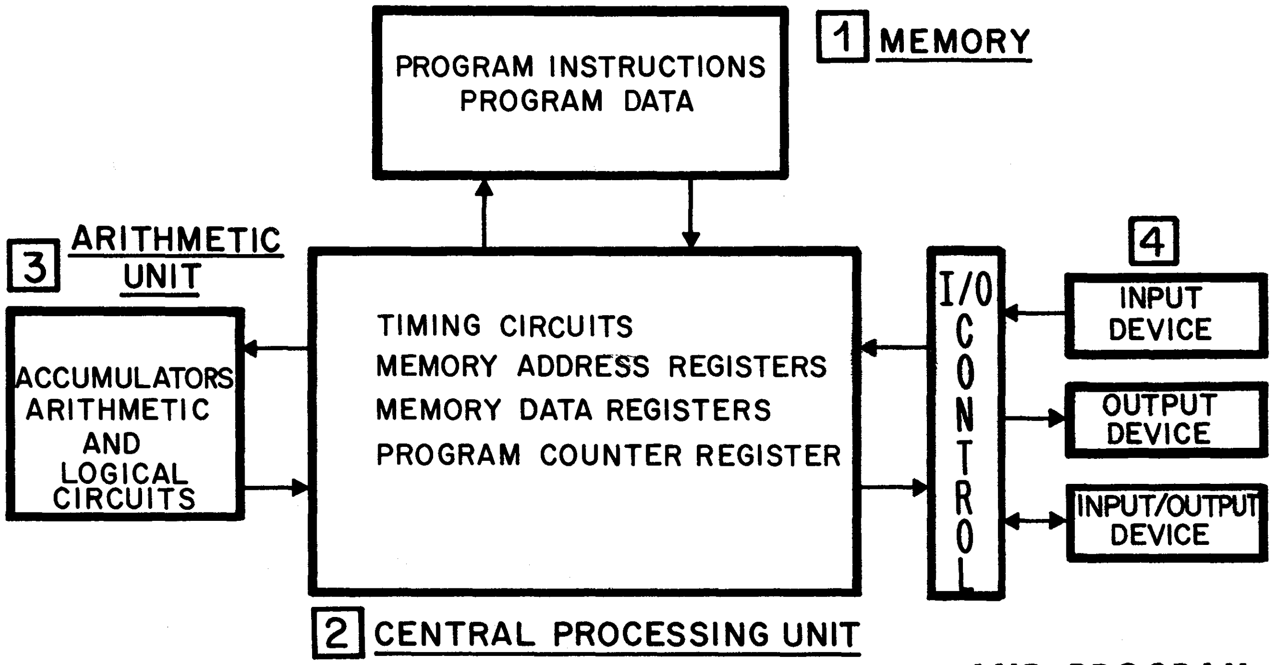
ASSUME THE FOLLOWING INITIAL CONDITIONS:

1. A PROBLEM CONSISTING OF n COMPUTER INSTRUCTIONS IS STORED IN SEQUENTIAL MEMORY LOCATIONS.
2. THE MEMORY ADDRESS OF THE FIRST INSTRUCTION IS PLACED IN THE COMPUTER "PROGRAM COUNTER" REGISTER.

PROGRAM EXECUTION FOLLOWS THIS SIMPLE PATTERN:

1. READ (FETCH) AN INSTRUCTION FROM MEMORY.
2. DECODE THE INSTRUCTION. IF THE INSTRUCTION IS NOT A MEMORY REFERENCE TYPE, GO TO 4.
3. READ THE OPERAND FROM MEMORY.
4. EXECUTE THE INSTRUCTION.
5. INCREMENT THE "PROGRAM COUNTER" REGISTER.
6. GO TO STEP 1.

A BASIC COMPUTER



1 **MEMORY CONTAINS ALL PROGRAM INSTRUCTIONS AND PROGRAM CONSTANTS (NUMBERS)**

2 **THE C.P.U. CONTROLS AND DIRECTS ALL COMPUTER OPERATIONS.**

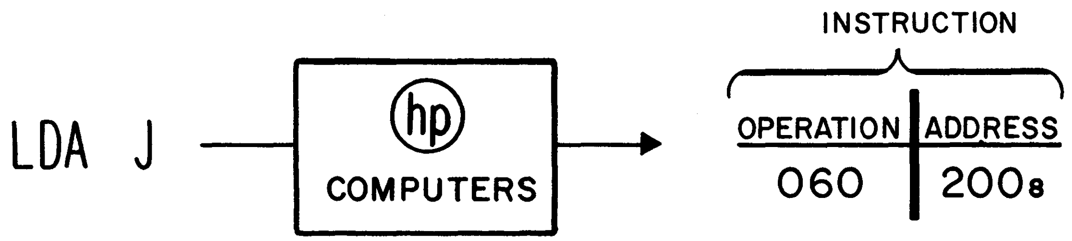
3 **THE ARITHMETIC UNIT PERFORMS ALL LOGICAL AND ARITHMETIC OPERATIONS, AS DIRECTED BY THE C.P.U.**

4 **THE I/O DEVICES, DIRECTED BY THE C.P.U., TRANSFER DATA OR INSTRUCTIONS TO OR FROM MEMORY.**

COMPUTER INSTRUCTIONS

ON THE NEXT FEW SLIDES THE COMPUTER'S INSTRUCTION EXECUTION SEQUENCE WILL BE DISCUSSED. A BRIEF REVIEW OF COMPUTER INSTRUCTIONS IS SHOWN.

FOR EXAMPLE

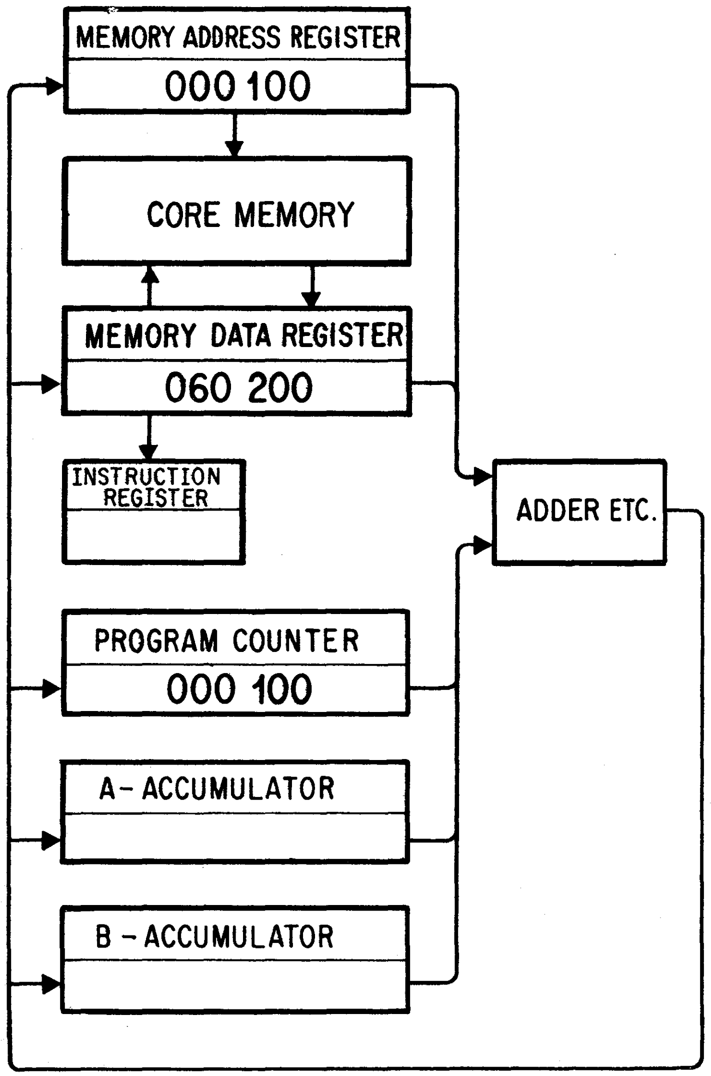


TRANSLATION:
LOAD REGISTER "A" WITH THE CONTENTS OF MEMORY LOCATION "J". REMEMBER, THE COMPUTER ONLY UNDERSTANDS 060200. LDA J IS STRICTLY FOR OUR BENEFIT.

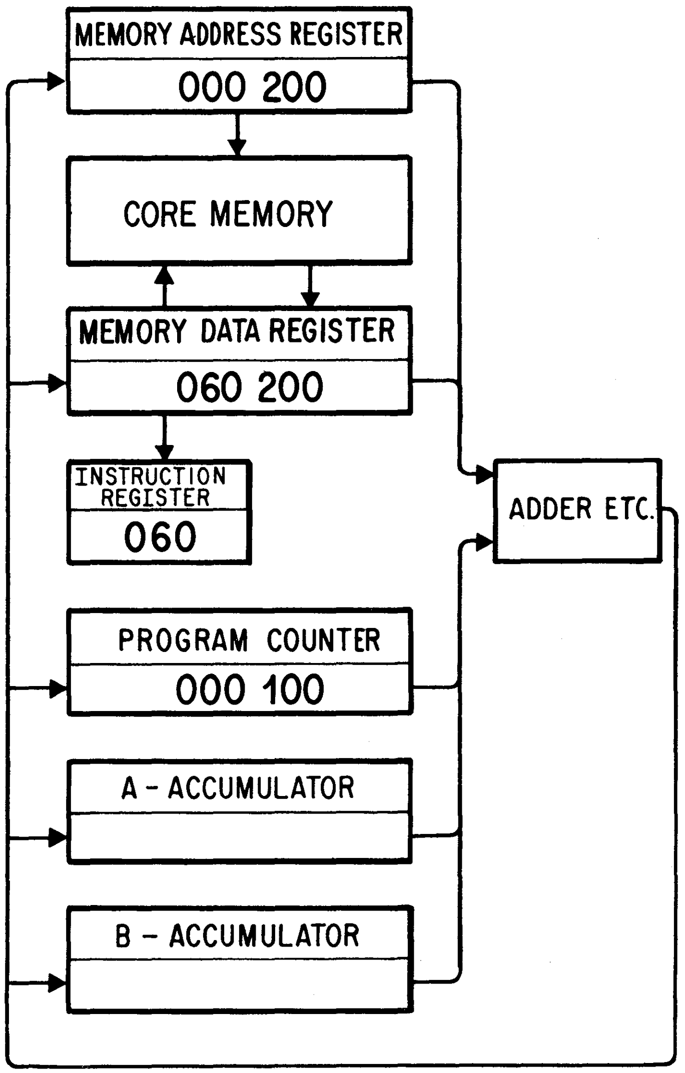
NOTE: IN THIS EXAMPLE "J" IS ARBITRARILY REPRESENTING MEMORY LOCATION 200_8.

EXAMPLE
(LDA J = 060200)

INSTRUCTION EXECUTION



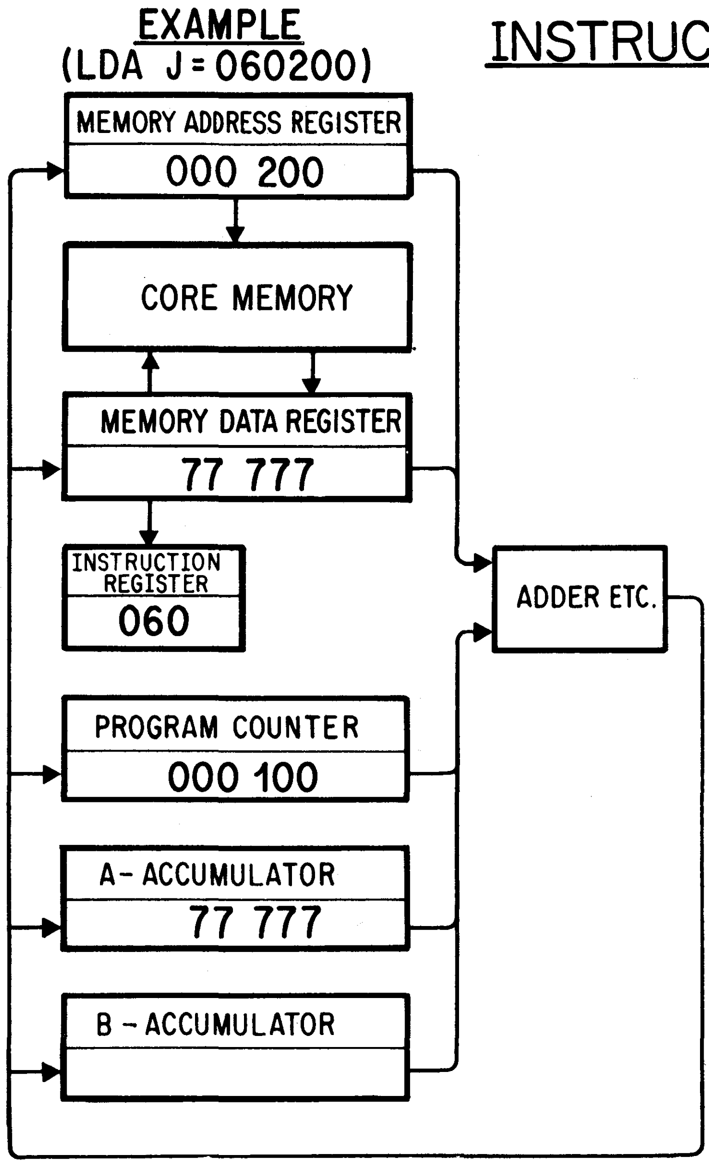
a. Instruction is Read from Memory



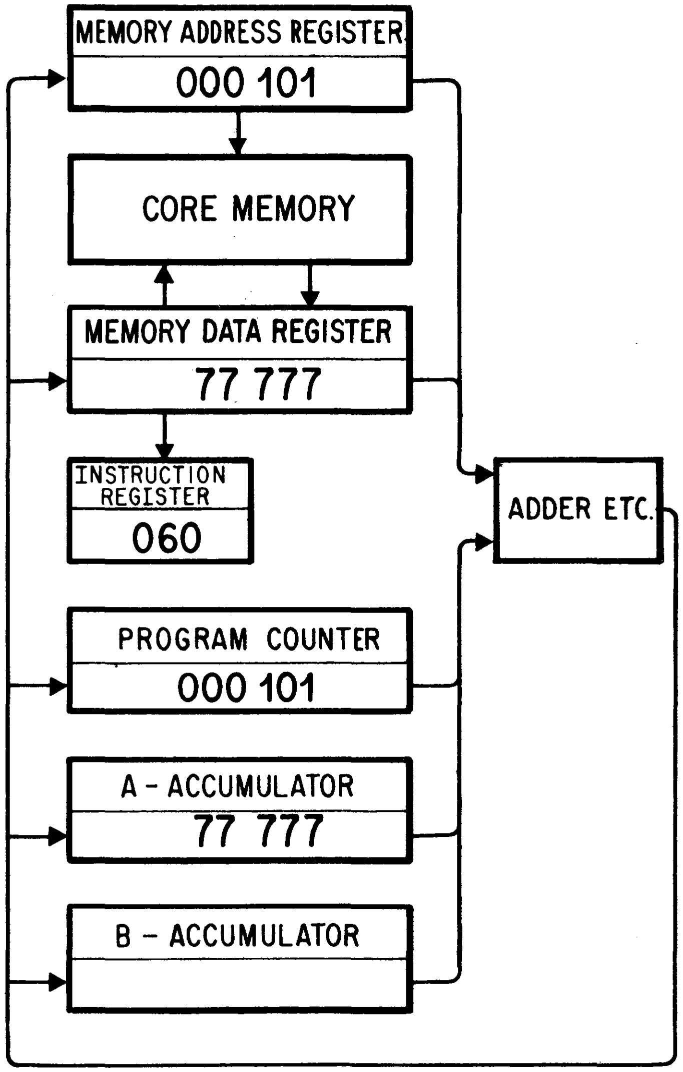
b. End of Fetch Phase

INSTRUCTION IN LOCATION 100 - FETCH PHASE

INSTRUCTION EXECUTION cont'd

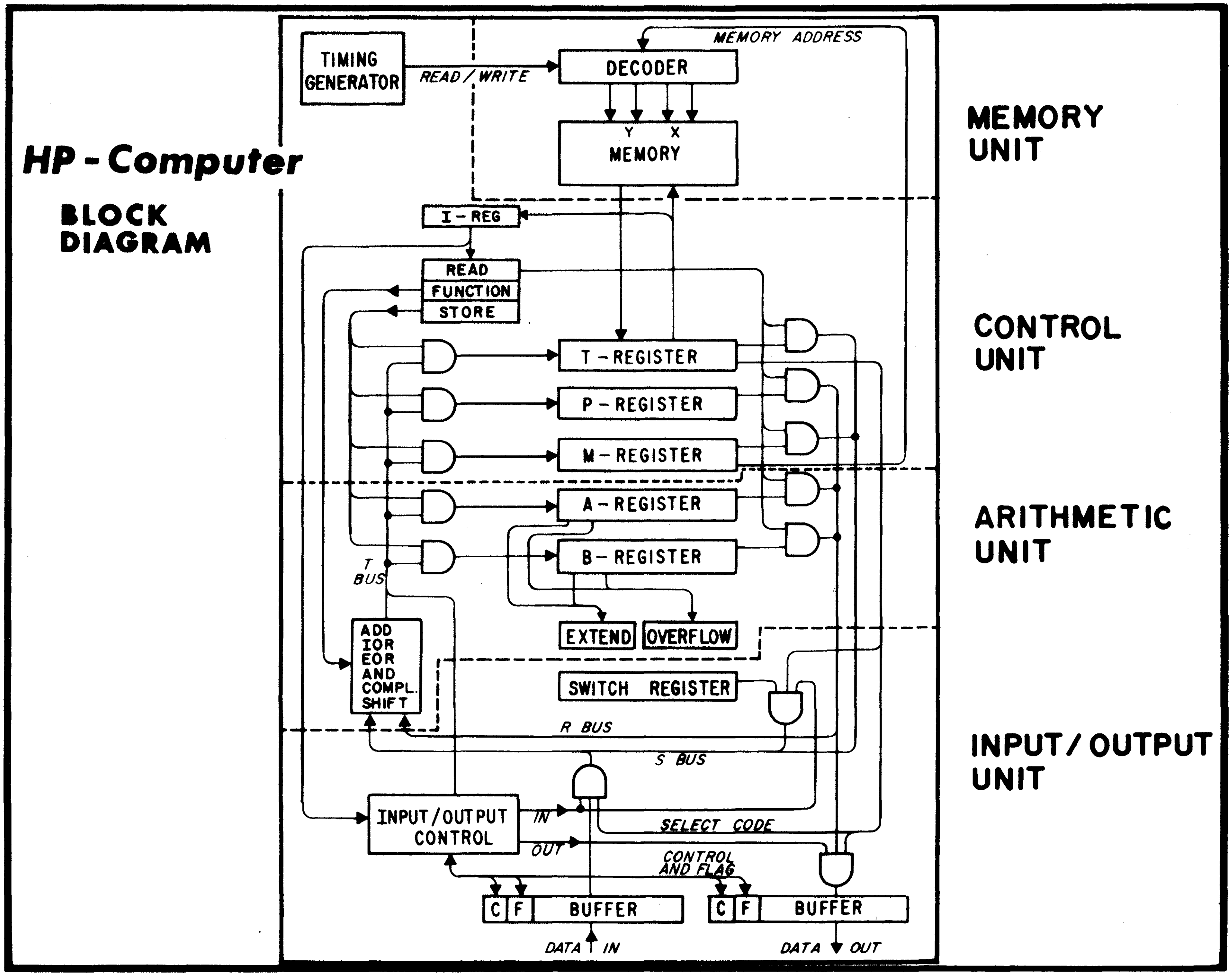


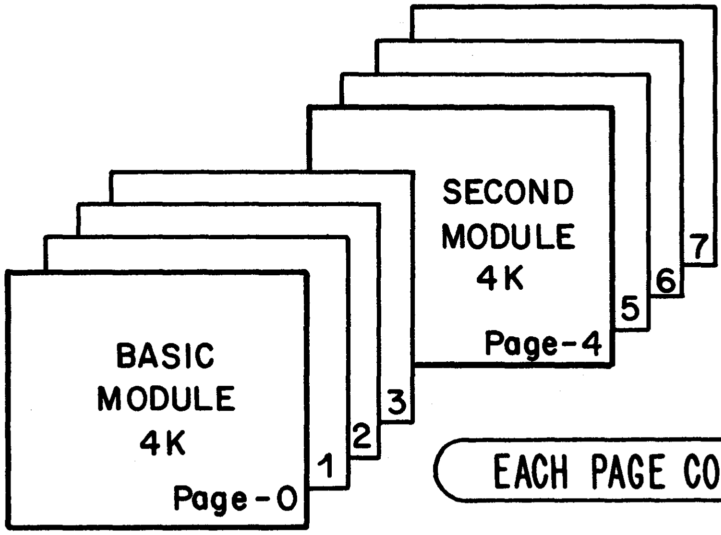
c. Instruction Executed



d. End of Execute Phase

INSTRUCTION IN LOCATION 100-EXECUTE PHASE


HP - Computer
BLOCK DIAGRAM
TIMING GENERATOR
READ / WRITE
MEMORY ADDRESS
DECODER
Y X
MEMORY
I - REG
READ
FUNCTION
STORE
T - REGISTER
P - REGISTER
M - REGISTER
A - REGISTER
B - REGISTER
T BUS
EXTEND
OVERFLOW
ADD
IOR
EOR
AND
COMPL.
SHIFT
SWITCH REGISTER
R BUS
S BUS
INPUT / OUTPUT CONTROL
IN
OUT
SELECT CODE
CONTROL AND FLAG
C
F
BUFFER
C
F
BUFFER
DATA IN
DATA OUT
MEMORY UNIT
CONTROL UNIT
ARITHMETIC UNIT
INPUT / OUTPUT UNIT



EACH PAGE CONTAINS 1024_{10} LOCATIONS

MODULE	PAGE	OCTAL	
		FROM	TO
BASIC	0	0	01777
"	1	02000	03777
"	2	04000	05777
"	3	06000	07777
SECOND	4	10000	11777
"	5	12000	13777
"	6	14000	15777
"	7	16000	17777

MEMORY ADDRESSING (8K)

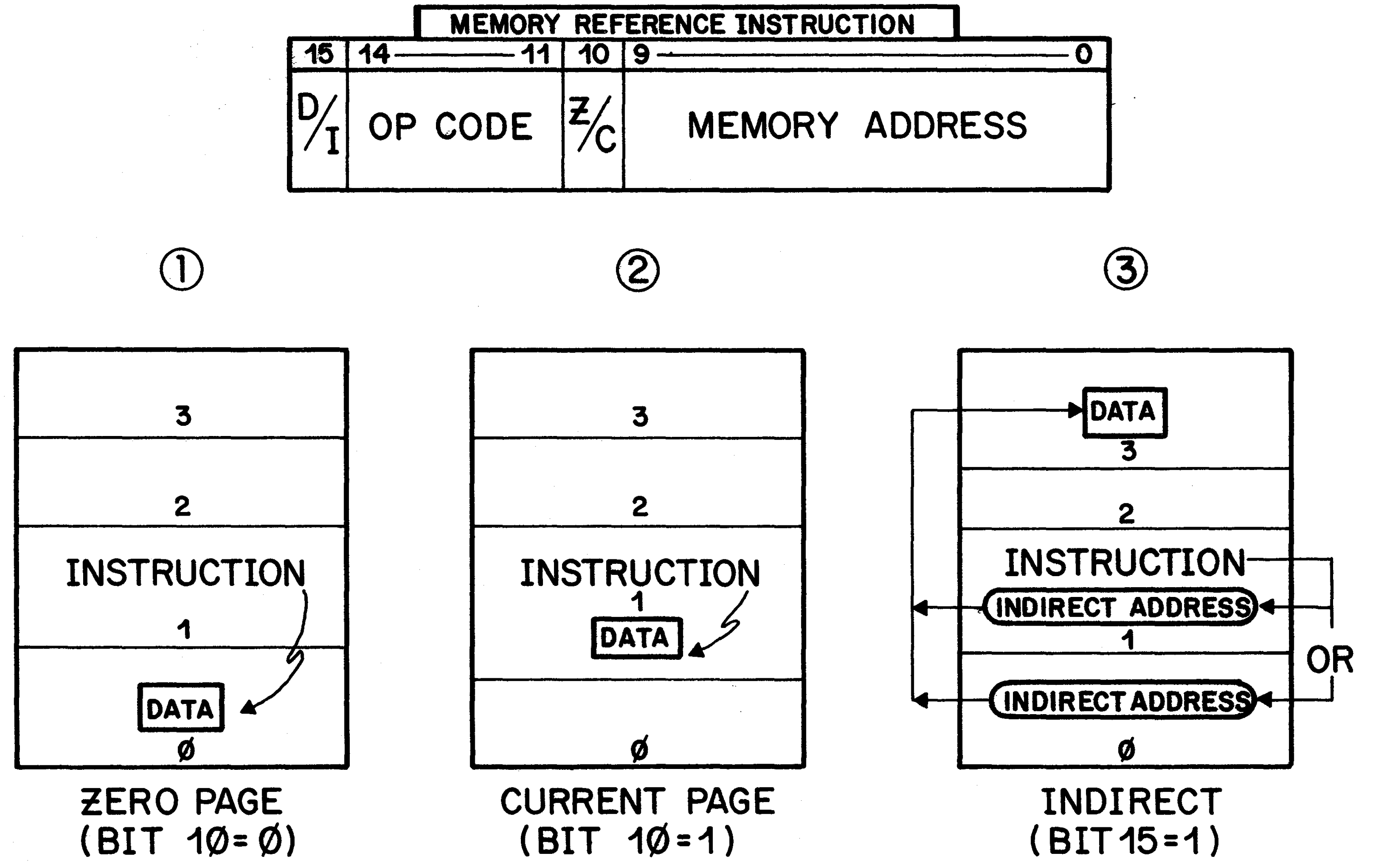
MEMORY

6000 – 7777
4000 – 5777
2000 – 3777
0000 – 1777

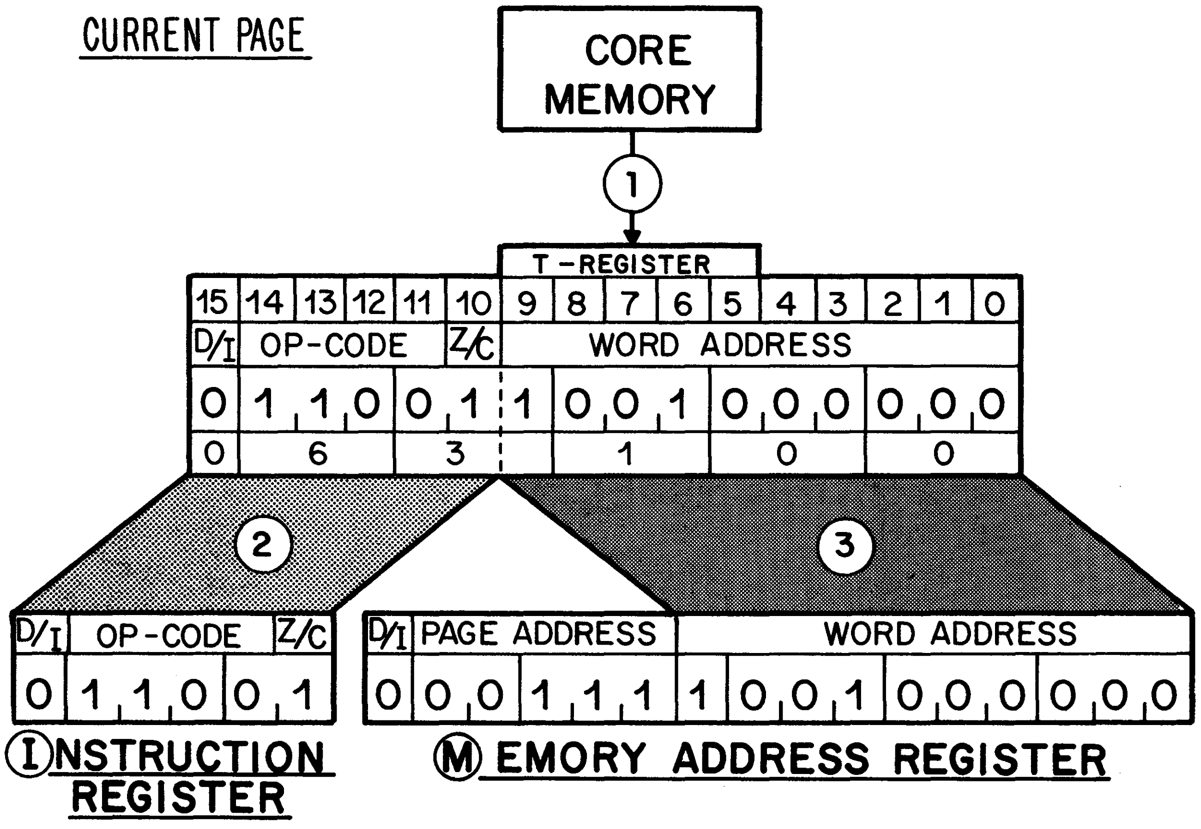
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/I	PAGE ADDRESS					WORD ADDRESS									

3	0000 – 1777
2	0000 – 1777
1	0000 – 1777
0	0000 – 1777

MEMORY ADDRESS REGISTER



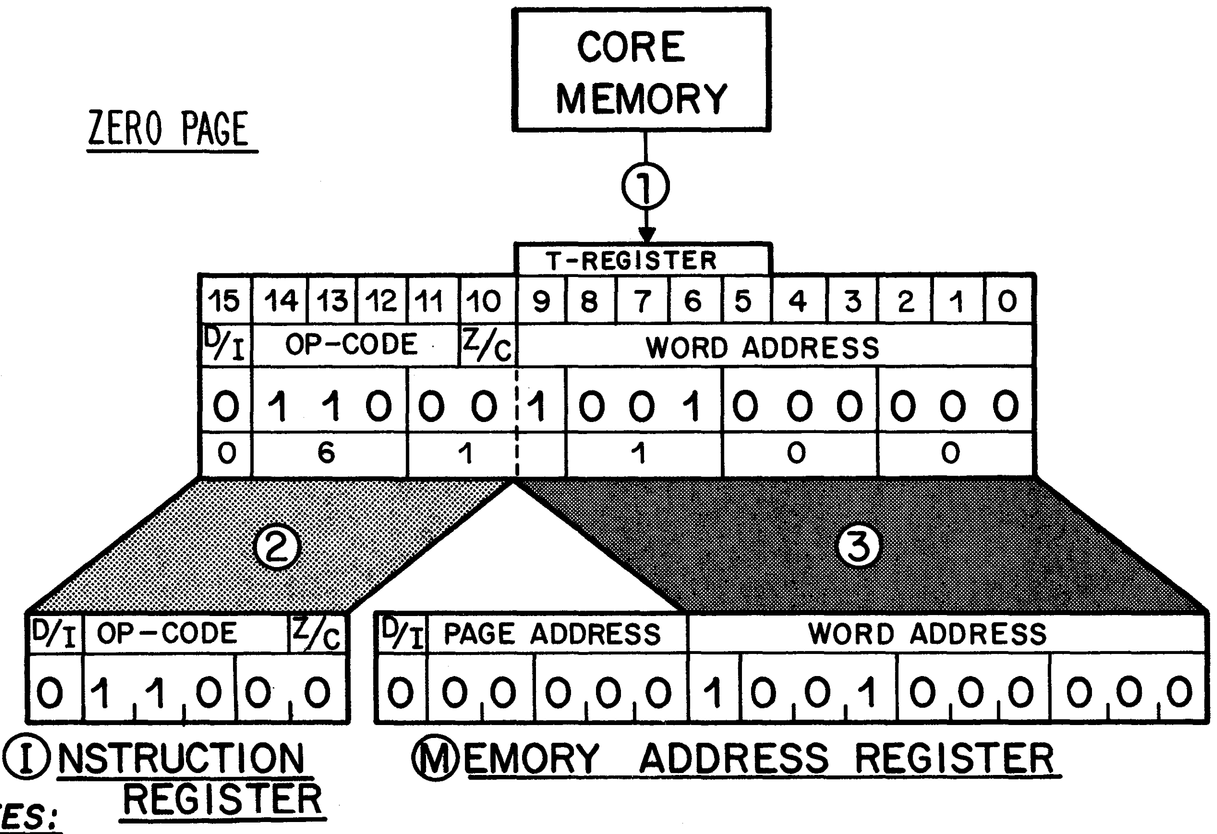
MEMORY ADDRESSING MODES



NOTES:

1. INSTRUCTION TRANSFERRED FROM MEMORY TO THE T-REGISTER.
2. BITS 10-15 TRANSFERRED FROM T-REG. TO THE I-REG.
3. BITS 0-9 FROM T-REG. ARE MERGED WITH BITS 10-15 OF THE M-REG.

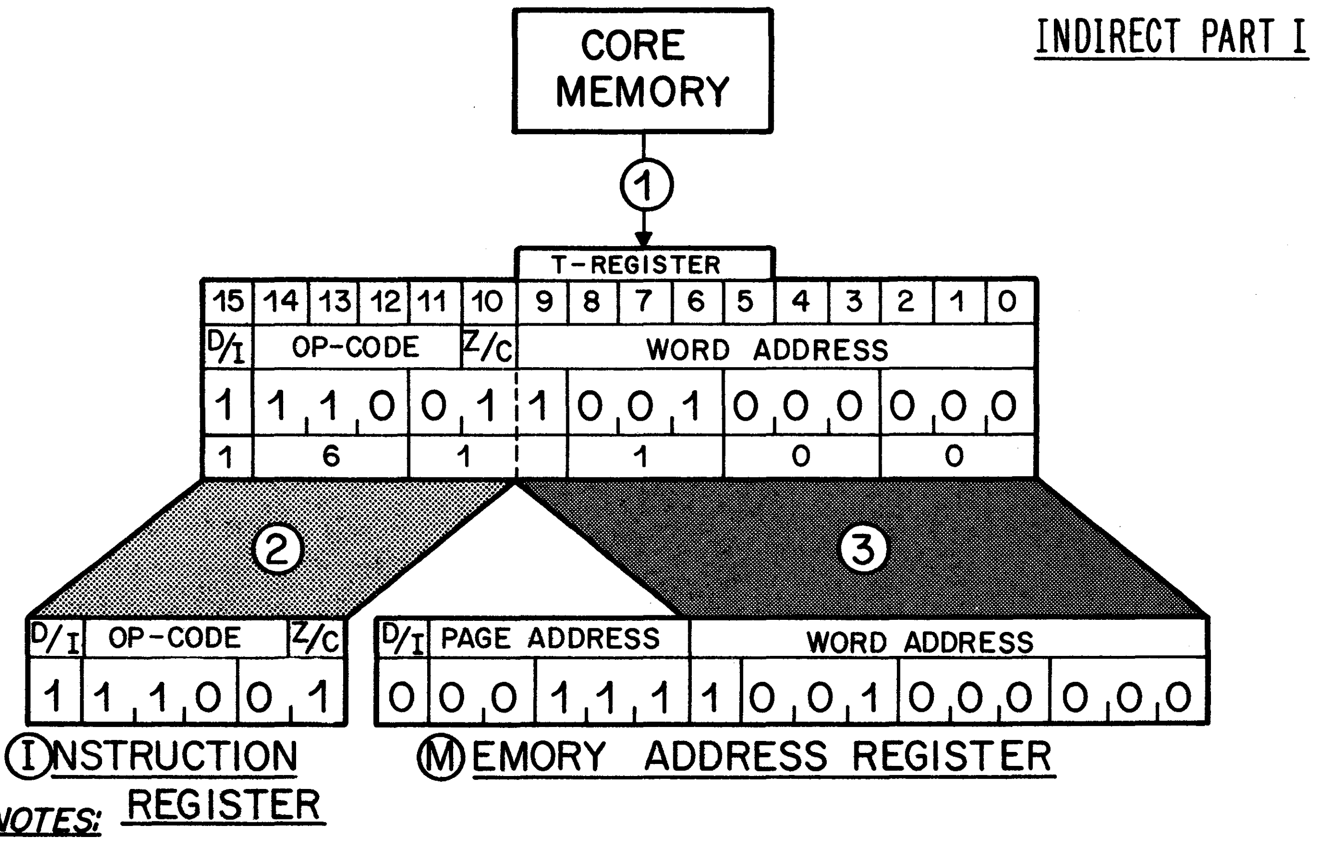
MEMORY REFERENCE INSTRUCTION DECODING



NOTES:

1. INSTRUCTION TRANSFERRED FROM MEMORY TO THE T-REGISTER.
2. BITS 10-15 TRANSFERRED FROM <u>T-REG.</u> TO THE <u>I-REG.</u>
3. BITS 0-9 TRANSFERRED FROM T-REG TO THE M-REG <u>AND</u> BITS 10-15 OF <u>M-REG</u> ARE <u>CLEARED TO ZERO</u>.

MEMORY REFERENCE INSTRUCTION DECODING



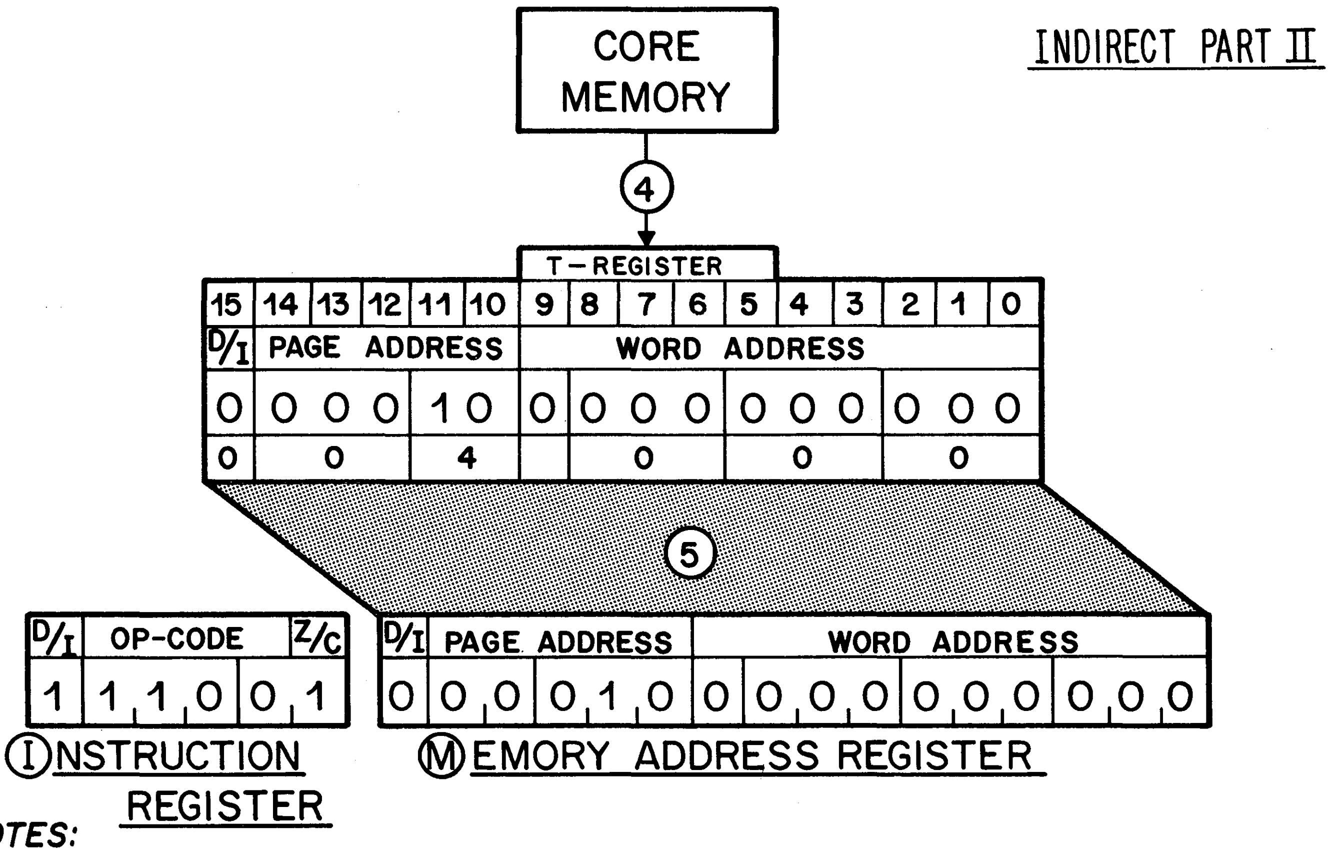
NOTES:

1. INSTRUCTION TRANSFERRED FROM MEMORY TO THE T-REGISTER.

2. BITS 10-15 TRANSFERRED FROM T-REG. TO THE I-REG.

3. BITS 0-9 FROM T-REG. ARE MERGED WITH BITS 10-15 OF THE M-REG.
BIT 15 OF I-REG. = 1 CAUSES ANOTHER CYCLE TO BEGIN.

MEMORY REFERENCE INSTRUCTION DECODING



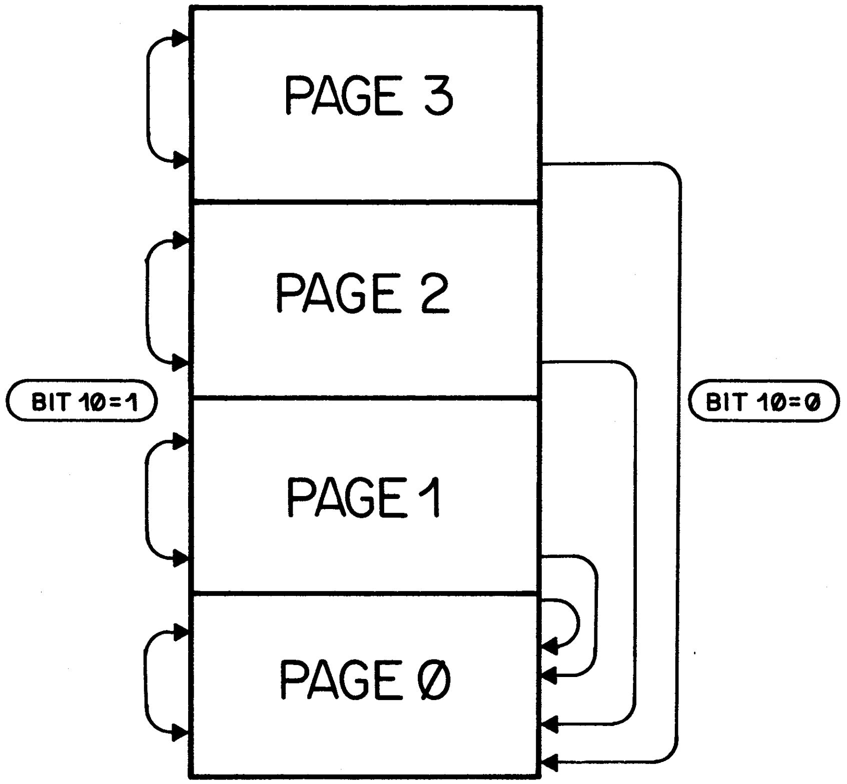
NOTES:

(4) THE 15 BIT ADDRESS IS TRANSFERRED FROM MEMORY TO THE "T"-REGISTER

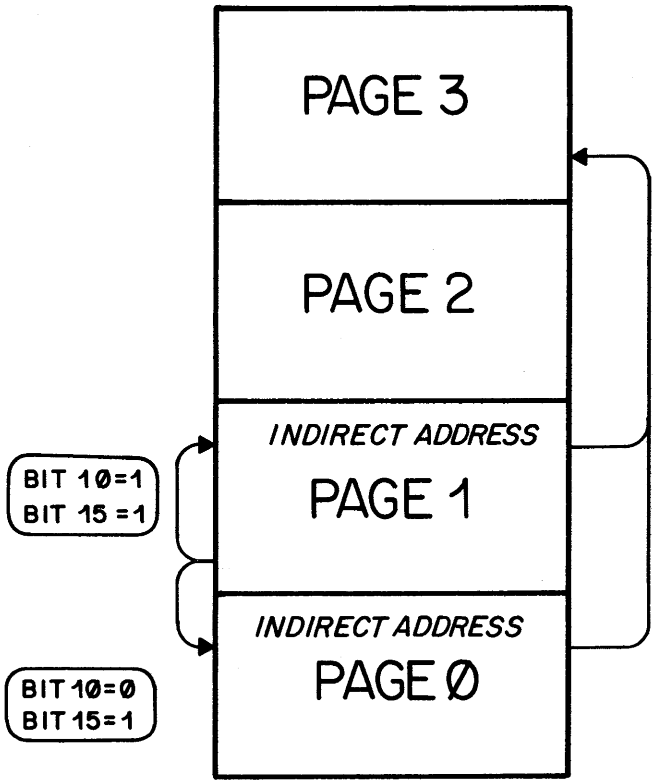
(5) BITS 0-15 TRANSFERRED FROM T-REGISTER TO THE M-REGISTER.

I-REG IS NOT CHANGED

MEMORY REFERENCE INSTRUCTION DECODING



MEMORY ADDRESSING
(DIRECT)



MEMORY ADDRESSING
(INDIRECT)

MEMORY ADDRESSING REVIEW

A UNIQUE FEATURE OF H-P COMPUTERS IS THE ABILITY TO ADDRESS THE "A" OR "B" REGISTERS DIRECTLY. THE METHOD USED TO PROVIDE THIS FEATURE WAS TO MAKE REGISTER "A" SYNONYMOUS WITH MEMORY ADDRESS Ø AND REGISTER "B" SYNONYMOUS WITH MEMORY ADDRESS 1.

THEREFORE

MEMORY ADDRESS Ø IS THE "A" REGISTER.
MEMORY ADDRESS 1 IS THE "B" REGISTER.

EXAMPLE

LOAD THE "A" REGISTER WITH THE CONTENTS OF THE "B" REGISTER.

MNEMONIC	MACHINE CODE
LDA 1	060001

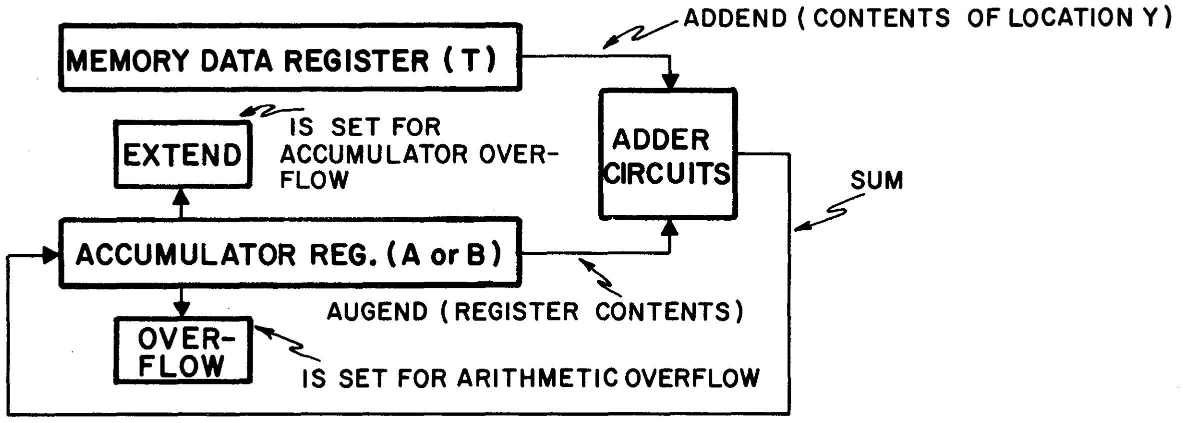
ADDRESSABLE REGISTERS

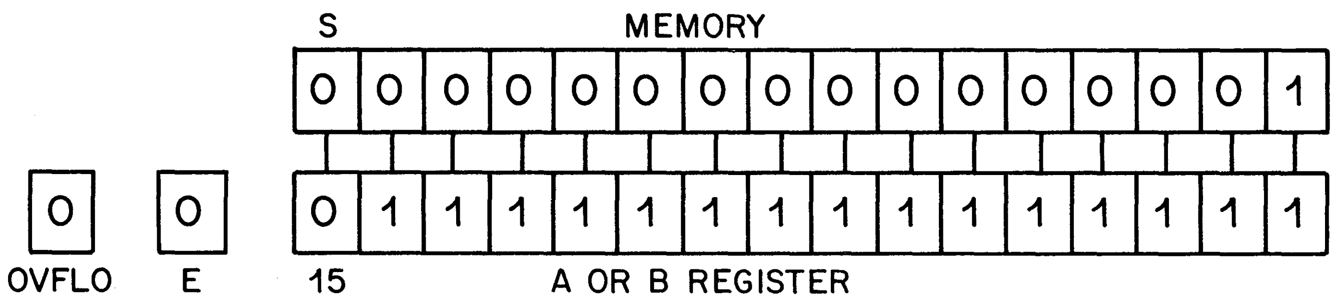
/● COMPUTER ARITHMETIC OPERATIONS

THE BASIC ARITHMETIC OPERATION OF THE COMPUTER IS THE INTEGER ADD. IT IS IMPORTANT THAT THE PROGRAMMER UNDERSTAND HOW THE MACHINE PERFORMS THIS BASIC OPERATION.

FOR EXAMPLE, ADA Y MEANS:

TO THE CONTENTS OF REGISTER "A" ADD THE CONTENTS OF MEMORY LOCATION Y. THE SUM REPLACES THE PREVIOUS CONTENTS OF REGISTER "A".

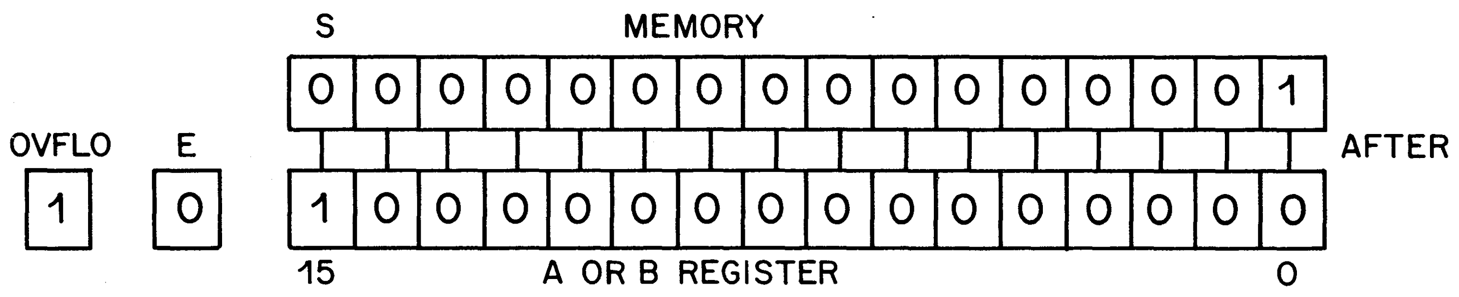

ADDEND (CONTENTS OF LOCATION Y)
MEMORY DATA REGISTER (T)
IS SET FOR ACCUMULATOR OVERFLOW
EXTEND
ADDER CIRCUITS
SUM
ACCUMULATOR REG. (A or B)
AUGEND (REGISTER CONTENTS)
OVER-FLOW
IS SET FOR ARITHMETIC OVERFLOW



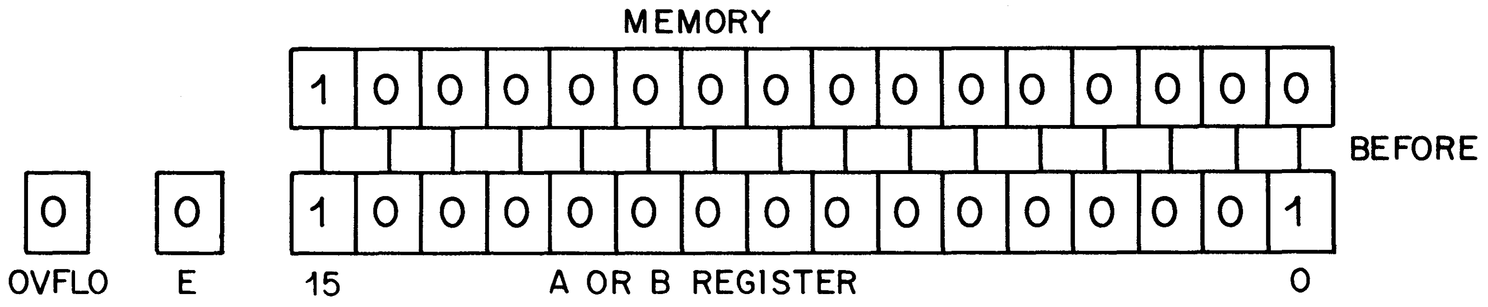
INSTRUCTION

[ADD]

POSITIVE OVERFLOW CASE. THE CARRY IN TO BIT 15 CHANGES THE SIGN. THE ADDITION OF TWO POSITIVE NUMBERS CANNOT CORRECTLY PRODUCE A NEGATIVE RESULT. THE OVERFLOW LAMP IS ON.



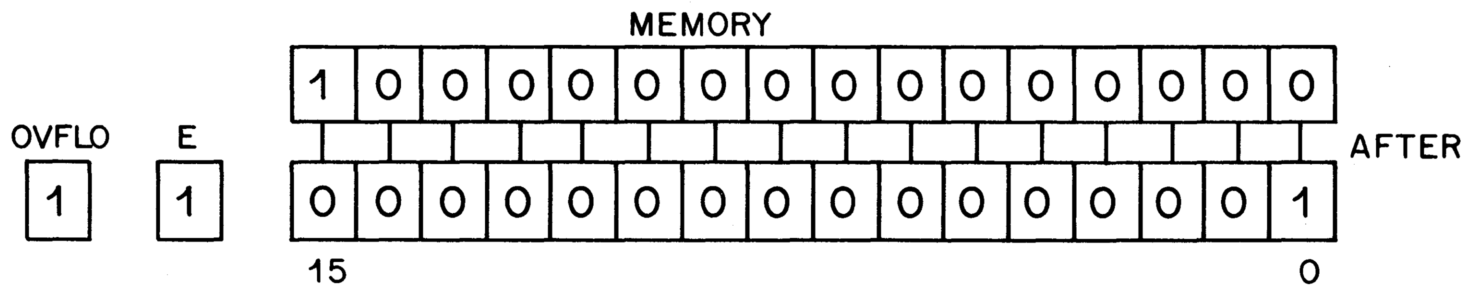
POSITIVE OVERFLOW



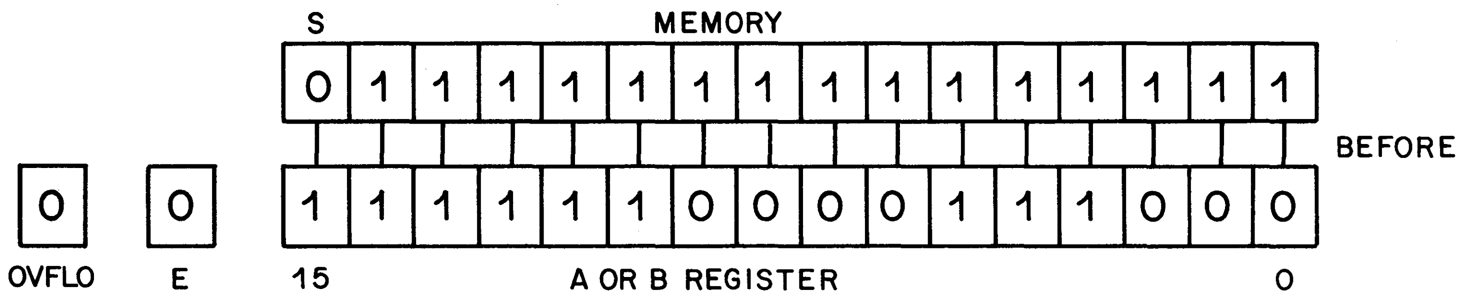
INSTRUCTION

[ADD]

NEGATIVE OVERFLOW. NO CARRY IN TO BIT 15 CHANGES THE SIGN. THE ADDITION OF TWO NEGATIVE NUMBERS CANNOT CORRECTLY PRODUCE A POSITIVE RESULT. THE OVERFLOW LAMP IS ON AND THE CARRY FROM BIT 15 WILL SET E TO 1.



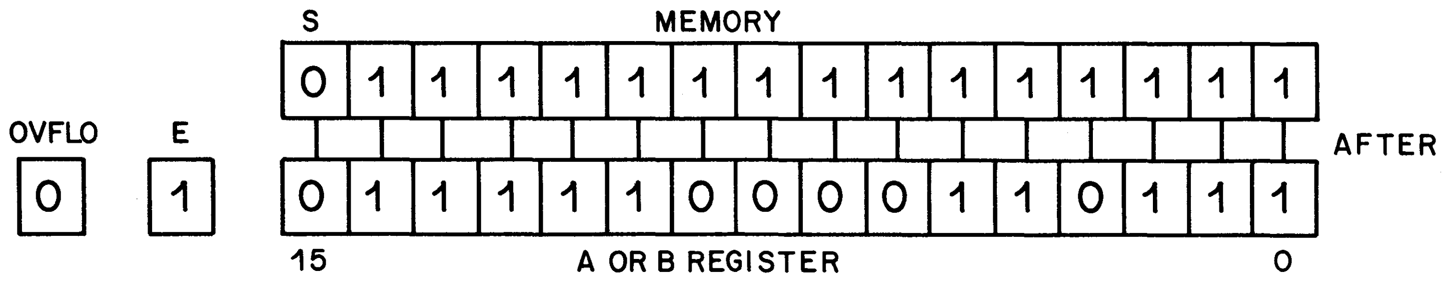
NEGATIVE OVERFLOW



INSTRUCTION

[ADD]

A POSITIVE NUMBER ADDED TO A NEGATIVE NUMBER (OR THE CONVERSE) WILL NEVER SET THE OVERFLOW CONDITION. IT IS POSSIBLE HOWEVER TO SET "E" TO 1 WITHOUT THE OVERFLOW CONDITION.



ADDITION OF POSITIVE AND NEGATIVE NUMBERS

MEMORY	A/B REGISTER	RESULT	"OVFLO"	"E" REGISTER
+	+	+	NO	Ø
+	+	−	YES	Ø
+	−	±	NO	1 OR Ø
−	+	±	NO	1 OR Ø
−	−	−	NO	1
−	−	+	YES	1

* OVFLO, E REGISTERS CAN BE SET BY ADD OR INCREMENT INSTRUCTIONS.

TABLE OF CONDITIONS

(STATUS OF "OVF" & "E" REGISTERS)

6-22

Introduction to the HP Assembler program VII

LESSON VII
INTRODUCTION TO THE HP ASSEMBLER PROGRAM

LESSON VII OBJECTIVES

THIS LESSON PROVIDES A BRIEF OVER VIEW OF ASSEMBLY LANGUAGE PROGRAMMING AND INTRODUCES THE INDIVIDUAL ASSEMBLER INSTRUCTIONS. THE ASSEMBLER INSTRUCTIONS ARE LIKE TOOLS IN A TOOLBOX. LEARNING HOW TO USE EACH "TOOL" IN THE TOOLBOX IS THE JOB OF THE ASSEMBLY LANGUAGE PROGRAMMER.

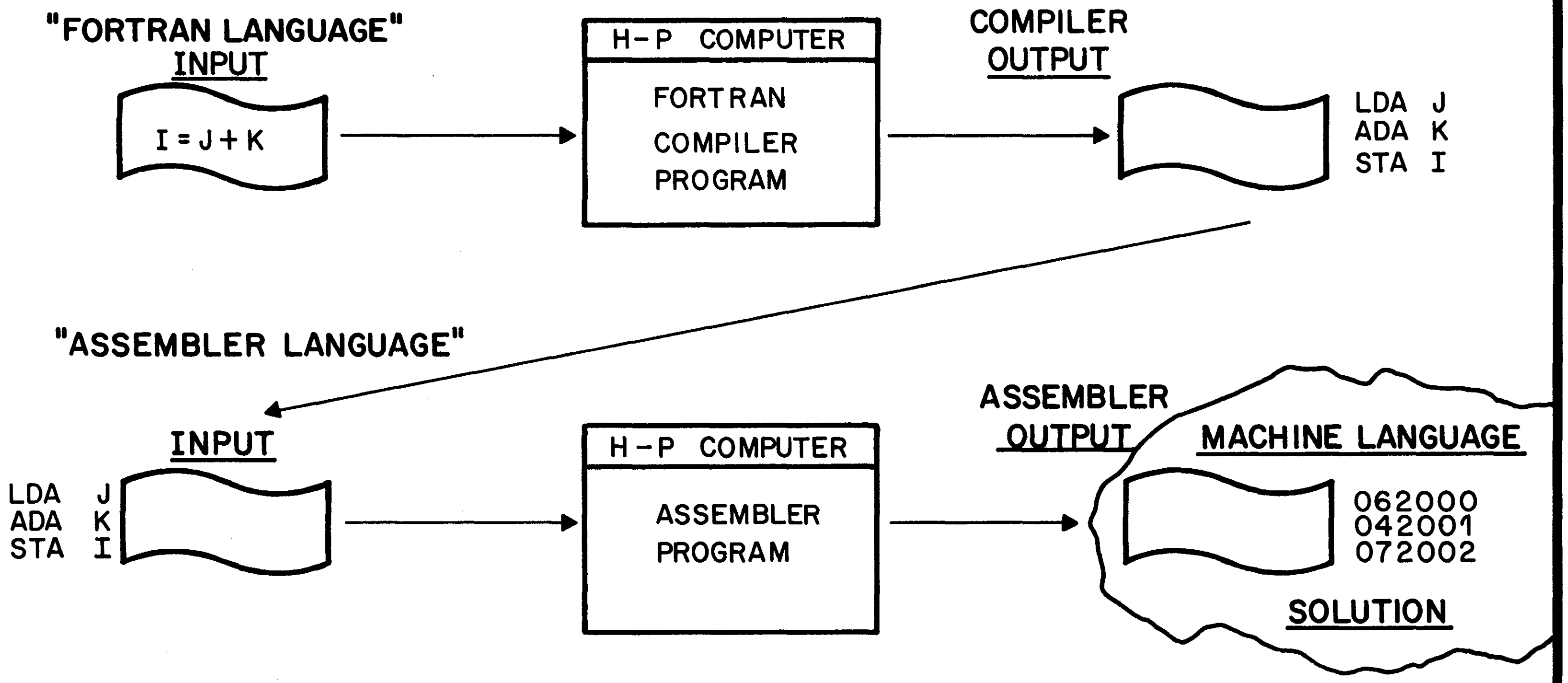
THE PRIMARY OBJECTIVE OF LESSON VII IS TO PROVIDE ENOUGH ASSEMBLER "TOOLS" TO ENABLE THE STUDENT TO WRITE A SIMPLE ASSEMBLY LANGUAGE PROGRAM.

INTRODUCTION TO HP ASSEMBLY LANGUAGE

THE FORTRAN, STATEMENT, X=B*B-4.*A*C PRODUCED THE FOLLOWING ASSEMBLY LANGUAGE CODE:

SYMBOLIC CODE	EXPLANATION
DLD B	LOAD B (INTO ACCUMULATORS)
FMP B	MULTIPLY B (B^2)
DST X	STORE RESULT TO X
DLD RM4	LOAD MINUS 4. (-4.)
FMP A	MULTIPLY A (-4.A)
FMP C	MULTIPLY C (-4.AC)
FAD X	ADD X (B^2-4.AC)
DST X	STORE RESULT TO X

COMPILERS AND ASSEMBLERS



NOTES: OUTPUT OF THE COMPILER BECOMES THE ASSEMBLER INPUT
OUTPUT OF THE ASSEMBLER IS IN MACHINE LANGUAGE

THE HP FORTRAN COMPILER (PASS 2 SECTION) CONTAINS A VERSION OF THE HP ASSEMBLER PROGRAM.

THE ASSEMBLY LANGUAGE VS FORTRAN

WHY TEACH ASSEMBLY LANGUAGE WHEN FORTRAN IS AVAILABLE?

ANSWER

1. ASSEMBLY LANGUAGE IS USED FOR ALL HP SOFTWARE DEVELOPMENT.
2. CERTAIN PROGRAMMING APPLICATION PROBLEMS ARE DIFFICULT OR IMPOSSIBLE TO SOLVE USING FORTRAN.
3. USER DEVELOPED ASSEMBLER LANGUAGE SUB-ROUTINES CALLED BY FORTRAN MAIN PROGRAMS, ENHANCE THE TOTAL CAPABILITY OF THE COM-PUTING SYSTEM.

THE HEWLETT-PACKARD SYMBOLIC ASSEMBLER PROVIDES "MEMORY PAGE FREE" PROGRAMMING AND IS A VERY EFFICIENT METHOD OF CREATING MACHINE LANGUAGE COMPUTER PROGRAMS.

THE TEN STEPS FROM PROBLEM TO PROGRAM

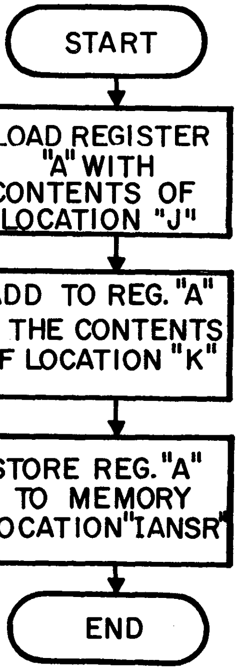
STEP 1 -	DEFINE THE PROBLEM.
STEP 2 -	PREPARE A FLOWCHART SOLUTION.
STEP 3 -	WRITE AN ASSEMBLY LANGUAGE PROGRAM.
STEP 4 -	KEYPUNCH THE SOURCE LANGUAGE TAPE USING A TELEPRINTER.
STEP 5 -	LOAD THE ASSEMBLER PROGRAM INTO THE HP COMPUTER.
STEP 6 -	ASSEMBLE THE SOURCE PROGRAM.
STEP 7 -	LOAD THE BASIC CONTROL SYSTEM INTO THE HP COMPUTER.
STEP 8 -	LOAD THE ASSEMBLER PRODUCED BINARY OBJECT TAPE.
STEP 9 -	(OPTIONAL) LOAD LIBRARY ROUTINES.
STEP 10	EXECUTE THE OBJECT PROGRAM.

PROBLEM DEFINITION

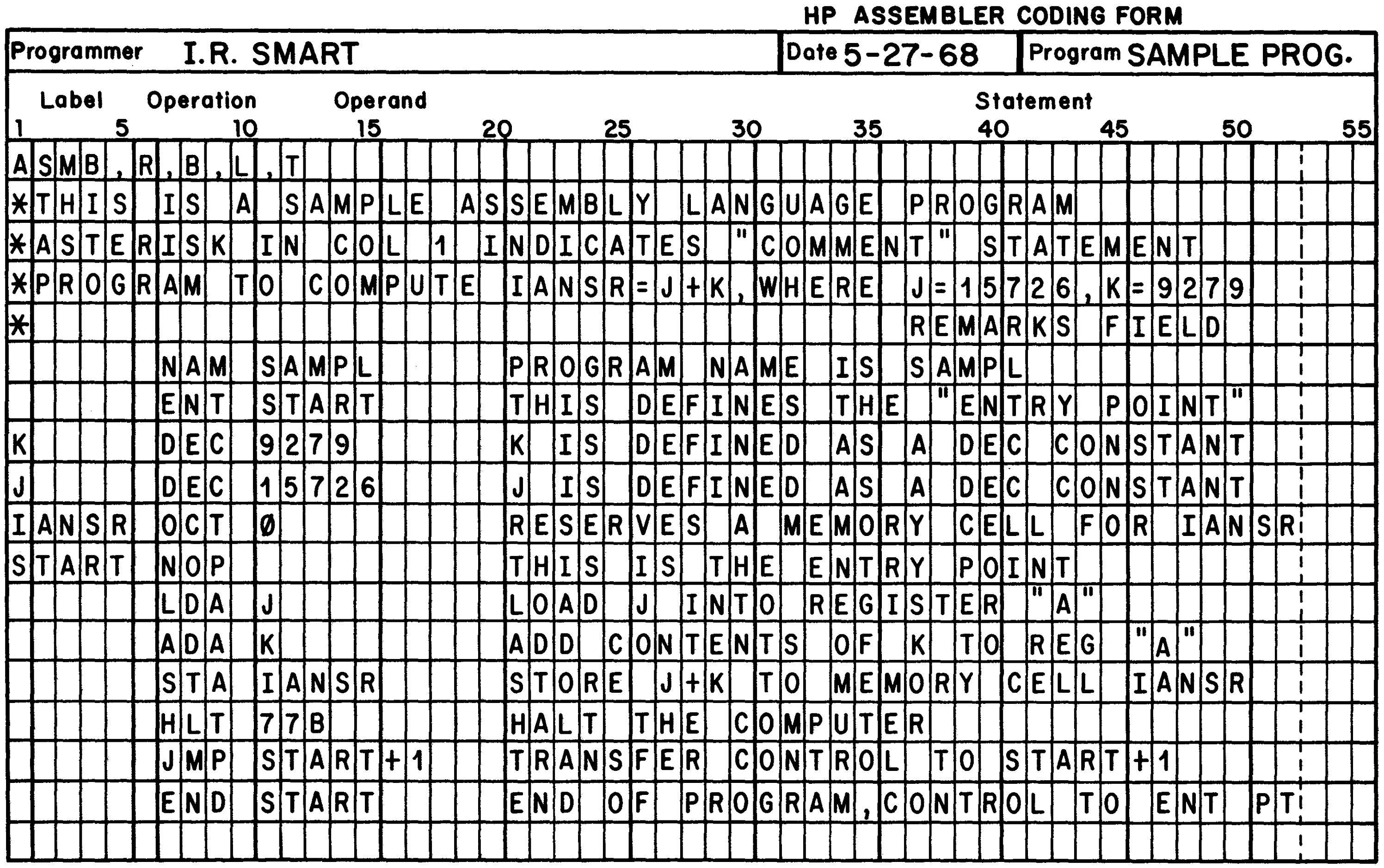
1. USING ASSEMBLY LANGUAGE TECHNIQUES WRITE A PROGRAM TO COMPUTE IANSR= J + K

   Where $J = 15726_{10}$
   $K = 9279_{10}$

2. FLOWCHART SOLUTION


START
LOAD REGISTER "A" WITH CONTENTS OF LOCATION "J"
ADD TO REG. "A" THE CONTENTS OF LOCATION "K"
STORE REG. "A" TO MEMORY LOCATION "IANSR"
END

ASSEMBLER CODING FORM

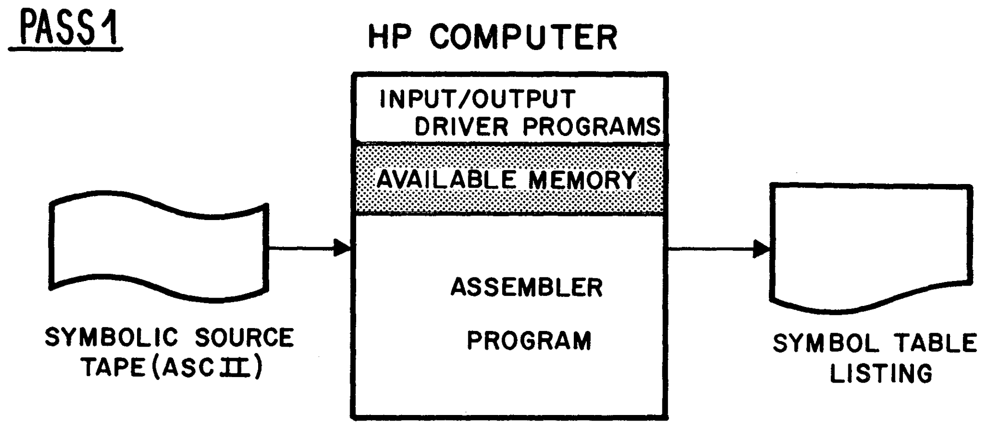


HP ASSEMBLER CODING FORM

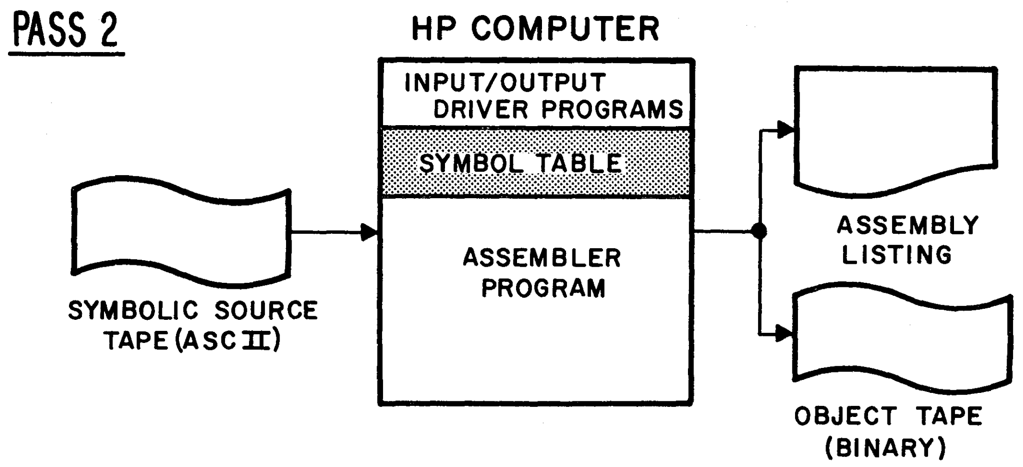
Programmer I.R. SMART | Date 5-27-68 | Program SAMPLE PROG.

```
 Label  Operation  Operand                           Statement
1   5    10   15   20   25   30   35   40   45   50   55
ASMB,R,B,L,T
*THIS IS A SAMPLE ASSEMBLY LANGUAGE PROGRAM
*ASTERISK IN COL 1 INDICATES "COMMENT" STATEMENT
*PROGRAM TO COMPUTE IANSR=J+K, WHERE J=15726,K=9279
*                                   REMARKS FIELD
      NAM SAMPL     PROGRAM NAME IS SAMPL
      ENT START     THIS DEFINES THE "ENTRY POINT"
K     DEC 9279      K IS DEFINED AS A DEC CONSTANT
J     DEC 15726     J IS DEFINED AS A DEC CONSTANT
IANSR OCT Ø         RESERVES A MEMORY CELL FOR IANSR
START NOP           THIS IS THE ENTRY POINT
      LDA J         LOAD J INTO REGISTER "A"
      ADA K         ADD CONTENTS OF K TO REG "A"
      STA IANSR     STORE J+K TO MEMORY CELL IANSR
      HLT 77B       HALT THE COMPUTER
      JMP START+1   TRANSFER CONTROL TO START+1
      END START     END OF PROGRAM,CONTROL TO ENT PT
```

ASSEMBLER PROGRAM OPERATIONS



PASS 1 OPERATIONS CREATE A "SYMBOL TABLE" IN THE AVAILABLE MEMORY AREA.



PASS 2 OPERATIONS RELATE THE SOURCE DATA TO THE SYMBOL TABLE, AND PRODUCE THE BINARY OBJECT TAPE AND THE ASSEMBLY LISTING.

NOTE: ASSEMBLER PRODUCED OUTPUT IS OPTIONAL.

ASSEMBLER PROCESSING

PASS 1

PROGRAM LOCATION COUNTER =	PLC_8	LABEL	OP CODE	OPERAND
		ASMB,R,B,L,T		
NAM SETS P.L.C. TO Ø	Ø		NAM	SAMPL
ASSEMBLER SYMBOL TABLE	Ø		ENT	START
"K" IS ASSIGNED THE VALUE Ø	Ø	K	DEC	9279
"J" " " 1	1	J	DEC	15726
"IANSR" " " 2	2	IANSR	OCT	Ø
"START" " " 3	3	START	NOP	
	4		LDA	J
	5		ADA	K
	6		STA	IANSR
	7		HLT	77B
	10		JMP	START+1
			END	START

NOTE: ONLY STATEMENTS WITH LABELS CREATE SYMBOL TABLE ENTRIES. THE SYMBOL VALUE IS ASSIGNED BY THE PROGRAM LOCATION COUNTER.

ASSEMBLER PROCESSING

PASS 2

LOCATION$_8$	CONTENTS$_8$	LABEL	OP CODE	OPERAND
ØØØØØ			NAM	SAMPL
			ENT	START
ØØØØØ	Ø22Ø77	K	DEC	9279
ØØØØ1	Ø36556	J	DEC	15726
ØØØØ2	ØØØØØØ	IANSR	OCT	Ø
ØØØØ3	ØØØØØØ	START	NOP	
ØØØØ4	Ø62ØØ1R		LDA	J
ØØØØ5	Ø42ØØØR		ADA	K
ØØØØ6	Ø72ØØ2R		STA	IANSR
ØØØØ7	1Ø2Ø77		HLT	77B
ØØØ1Ø	Ø26ØØ4R		JMP	START+1
			END	START

NOTE: MEMORY REFERENCE INSTRUCTIONS SEARCH THE SYMBOL TABLE TO FIND THE PROPER OPERAND VALUE.
MNEMONIC CODES ARE CONVERTED TO THEIR BINARY EQUIVALENT.

ASSEMBLY LISTING

```
 PAGE ØØØ1


ØØØ1                  ASMB,R,B,L,T
K      R ØØØØØØ
J      R ØØØØØ1
IANSR  R ØØØØØ2
START  R ØØØØØ3
** NO ERRORS*




 PAGE ØØØ2


ØØØ1                  ASMB,R,B,L,T
ØØØ2*THIS IS A SAMPLE ASSEMBLY LANGUAGE PROGRAM
ØØØ3*ASTERISK IN COL 1 INDICATES "COMMENT" STATEMENT
ØØØ4*PROGRAM TO COMPUTE IANSR=J+K,WHERE J =15726,K=9279
ØØØ5*                                        REMARKS FIELD
ØØØ6  ØØØØØ                  NAM SAMPL       PROGRAM NAME IS SAMPL
ØØØ7                         ENT START       THIS DEFINES THE "ENTRY POINT"
ØØØ8  ØØØØØ Ø22Ø77  K        DEC 9279        K IS DEFINED AS A DEC CONSTANT
ØØØ9  ØØØØ1 Ø36556  J        DEC 15726       J IS DEFINED AS A DEC CONSTANT
ØØ1Ø  ØØØØ2 ØØØØØØ  IANSR    OCT Ø           RESERVES A MEMORY CELL FOR IANSR
ØØ11  ØØØØ3 ØØØØØØ  START    NOP             THIS IS THE ENTRY POINT
ØØ12  ØØØØ4 Ø62ØØ1R          LDA J           LOAD J INTO REGISTER "A"
ØØ13  ØØØØ5 Ø42ØØØR          ADA K           ADD CONTENTS OF K TO REG. "A"
ØØ14  ØØØØ6 Ø72ØØ2R          STA IANSR       STORE J+K TO MEMORY CELL IANSR
ØØ15  ØØØØ7 1Ø2Ø77           HLT 77B         HALT THE COMPUTER
ØØ16  ØØØ1Ø Ø26ØØ4R          JMP START+1     TRANSFER CONTROL TO START+1
ØØ17                         END START       END OF PROGRAM, CONTROL TO ENT PT
** NO ERRORS*
```

A	THROUGH	Ƶ
Ø	THROUGH	9
.	PERIOD	
*	ASTERISK	
+	PLUS	
−	MINUS	
,	COMMA	
( )	PARENTHESES	
	SPACE	

ALL CHARACTERS ARE ASCII CODE

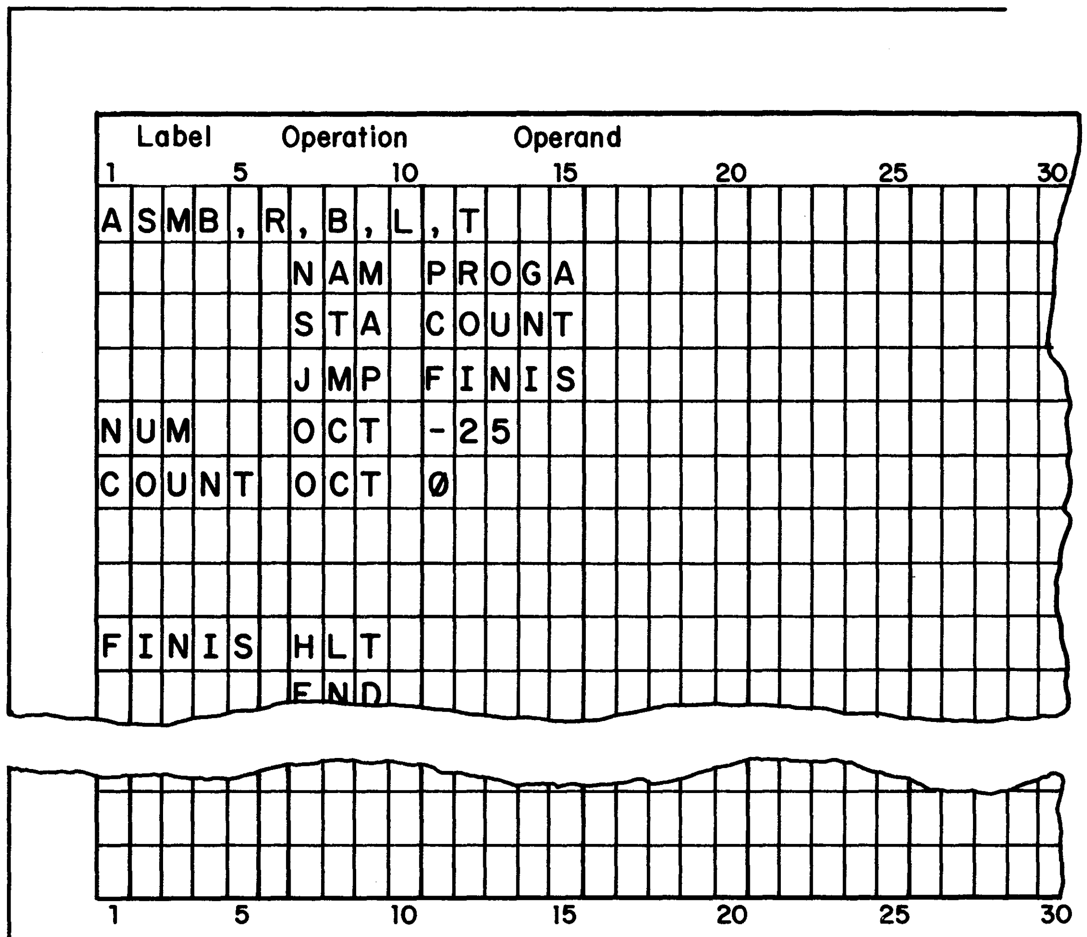
THE ASSEMBLER CHARACTER SET

```
 LABEL       OP CODE     OPERAND     REMARKS
ASMB,R,B,L,T
             NAM         TEST 1
BEGIN        NOP
             LDA         COUNT
             JMP         GO
NUM          OCT         -12
COUNT        OCT         Ø
GO           PROGRAM     CONTINUATION
             END         BEGIN
```

THE CONTROL STATEMENT MUST BEGIN IN COLUMN 1, AND IT MUST BE THE FIRST PHYSICAL STATEMENT OF A SOURCE PROGRAM.

ASMB	IDENTIFIES ASSEMBLY INPUT
A/R	ABSOLUTE OR RELOCATABLE PROGRAM.
B	BINARY OBJECT TAPE REQUESTED
L	ASSEMBLY LISTING REQUESTED
T	LISTING OF SYMBOL TABLE REQUESTED
END	MUST BE THE LAST PHYSICAL STATEMENT OF A PROGRAM

THE CONTROL STATEMENTS



```
ASMB,R,B,L,T
      NAM PROGA
      STA COUNT
      JMP FINIS
NUM   OCT -25
COUNT OCT Ø


FINIS HLT
      END
```

LABEL FIELD

ONE TO FIVE CHARACTERS, THE FIRST OF WHICH MUST BE ALPHABETIC OR THE PERIOD.

OPERATION FIELD

ALWAYS A THREE LETTER MNEMONIC CODE. COMMA IN COL. 1Ø EXTENDS OPERATION FIELD FOR COMBINABLE INSTRUCTIONS.

OPERAND FIELD

ALPHANUMERIC CHARACTERS, MAY EXTEND TO COL 52.

Ø=ZERO O=ALPHA O I OR 1= ONE I=ALPHA I LINE TERMINATED BY RETURN/LINE FEED (R/LF)
2= TWO Ƶ=ALPHA Ƶ LINE IS DELETED BY RUBOUT BEFORE R/LF

USING THE ASSEMBLER CODING SHEET

LABEL | OP CODE | OPERAND

	1	2	3	4	5		
VALID LABELS	A						
	·	A	B	C	D		
	·	1	2	3	4		
	A	·	1	2	3		
	·						
INVALID LABELS	1	·	A	B			FIRST CHARACTER NUMERIC
	A	B	C	1	2	3	6 CHARS.,TRUNCATED TO ABC 12
	A	B	*	C			ASTERISK ILLEGAL
		A	B	C			NO LABEL, FIRST BLANK TERMINATES LABEL FIELD.

EXAMPLES OF LABELS

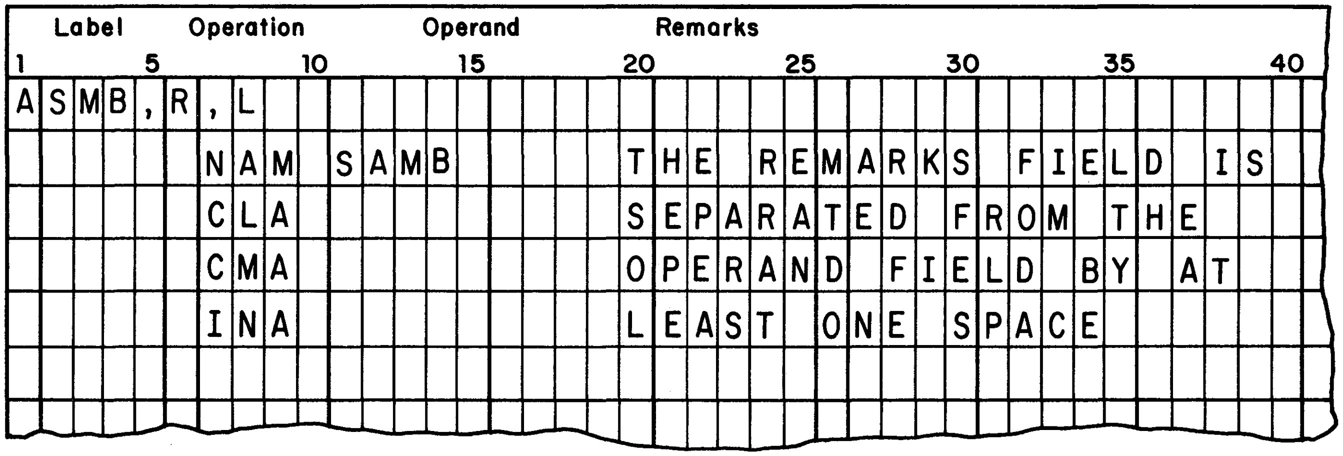
SPECIAL USE OF THE ASTERISK IN THE LABEL FIELD

- Asterisk in column 1 identifies a comment statement.
- Positions 2 –80 are available for comments.
- Comments appear in the assembly listing exactly as they appear in the source program.
- Comments are not processed by the assembler and use no storage.

  NOTE: POSITIONS 1 – 68 ONLY WILL BE PRINTED ON THE 2752A TELEPRINTER.

EXAMPLE:

```
                LABEL        OP CODE    OPERAND
COLUMN          1|2|3|4|5
_____________________________________________________
                *|T|H|I|S| IS AN EXAMPLE OF WRITING A COMMENT
                *|S|T|A|T|EMENT
```



Label
Operation
Operand
Remarks
1
5
10
15
20
25
30
35
40

```
ASMB,R,L
      NAM SAMB     THE REMARKS FIELD IS
      CLA          SEPARATED FROM THE
      CMA          OPERAND FIELD BY AT
      INA          LEAST ONE SPACE
```

THE REMARKS FIELD EXTENDS FROM THE OPERAND FIELD TO THE 80th CHARACTER. THE ENTIRE STATEMENT LENGTH SHOULD NOT EXCEED 52 CHARACTERS. REMARKS SHOULD BE OMITTED IF THE FOLLOWING STATEMENTS ARE USED WITHOUT OPERANDS: NAM, END, HLT, SOC, SOS.

REMARKS FIELD

PROGRAM LOCATION COUNTER	LABEL	OP CODE	OPERAND	REMARKS
2001		LDA	COUNT	
2002		STA	COUNT-1	
2003		JMP	*+3	
2004		OCT	Ø	
2005	COUNT	DEC	-25	
2006		HLT		

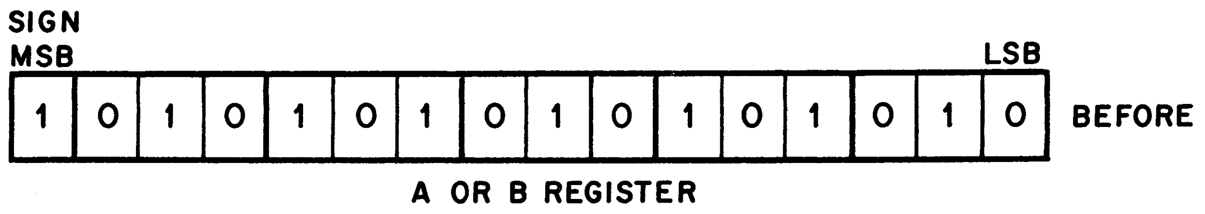
IN THIS EXAMPLE THE * HAS A VALUE OF 2003 THEREFORE * + 3 = 2006. * EQUALS THE VALUE OF THE P.L.C. WHEN IT IS ENCOUNTERED IN THE ASSEMBLY.

RELATIVE ADDRESSING

A SYMBOL USED IN THE OPERAND FIELD MUST BE DEFINED ELSEWHERE IN THE PROGRAM IN ONE OF THE FOLLOWING WAYS:

	LABEL	OP CODE	OPERAND
		LDA	*ALPHA*
		⋮	
AS A LABEL IN A MACHINE OPERATION	*ALPHA*	NOP	——
OR, AS A LABEL OF A PSEUDO	*ALPHA*	DEC	100
OR, IN THE <u>OPERAND</u> FIELD OF A COM OR EXT		COM EXT	*ALPHA* (10) *ALPHA*
OR, AS A LABEL OF AN ARITHMETIC PSEUDO.	*ALPHA*	MPY	——

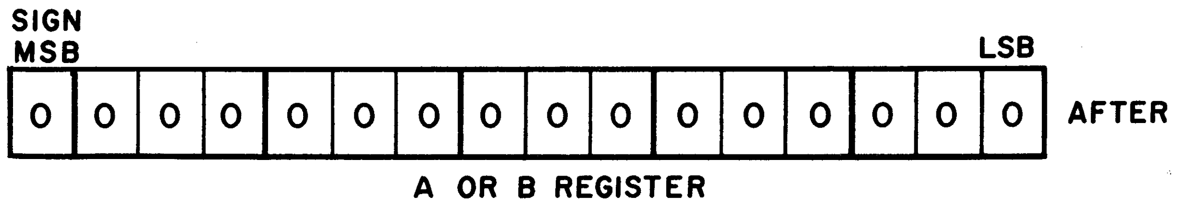
SYMBOL DEFINITION



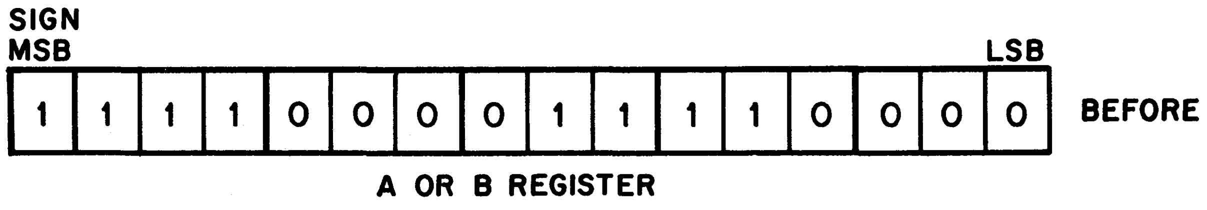
INSTRUCTION

[CLA/CLB]

CLEAR THE INDICATED REGISTER. ALL 16 BITS ARE SET TO Ø. OVFLO, 'E' ARE NOT AFFECTED.



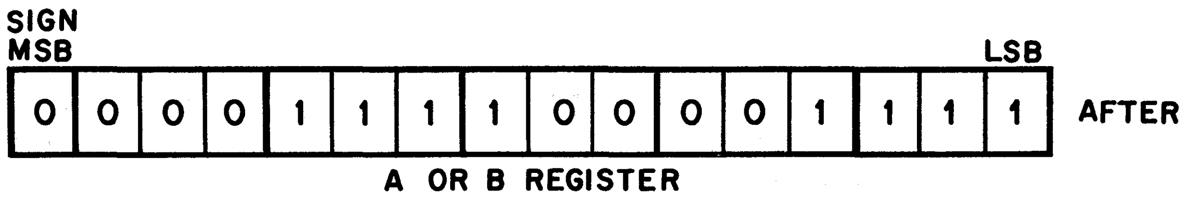
CLEAR ACCUMULATOR



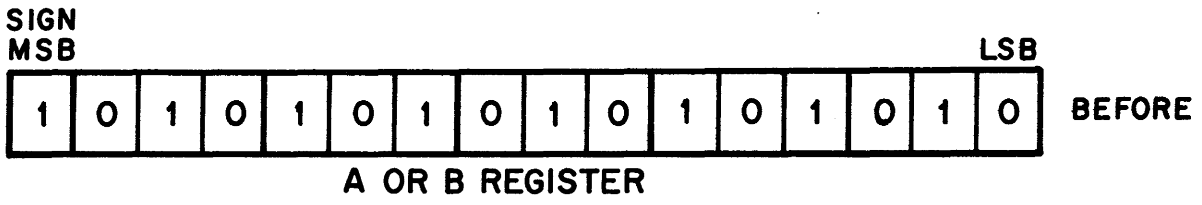
INSTRUCTION

[CMA/CMB]

COMPLEMENT THE CONTENTS OF THE INDICATED REGISTER. THIS IS A 1's COMPLEMENT. ALL Ø's BECOME 1's. ALL 1's BECOME Ø's. OVFLO, 'E' ARE NOT AFFECTED.



COMPLEMENT ACCUMULATOR

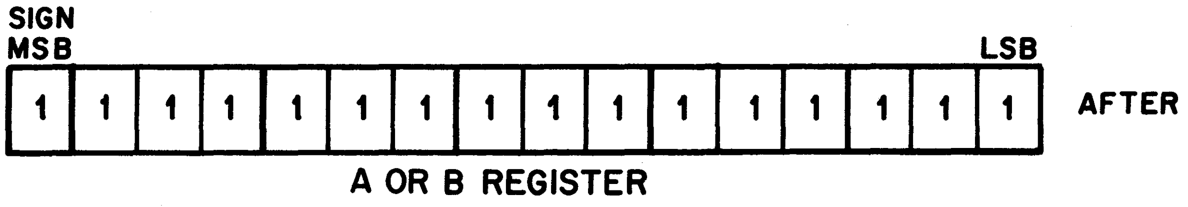


INSTRUCTION

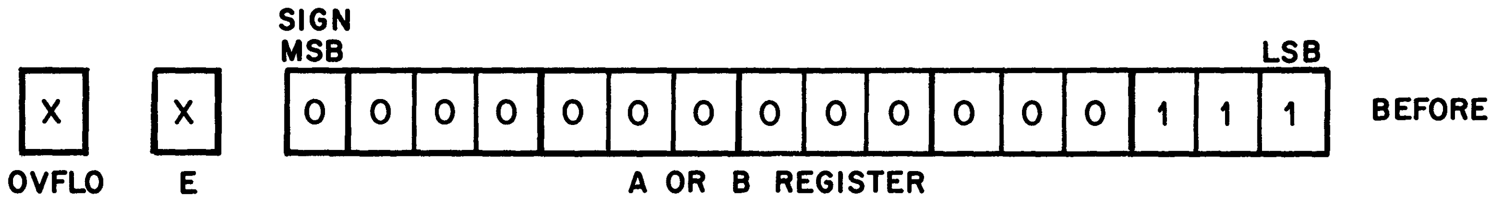
[CCA/CCB]

CLEAR AND THEN COMPLEMENT THE INDICATED REGISTER.

ALL 16 BITS ARE SET TO 1. OVFLO, 'E' ARE NOT AFFECTED.



CLEAR, COMPLEMENT THE ACCUMULATOR

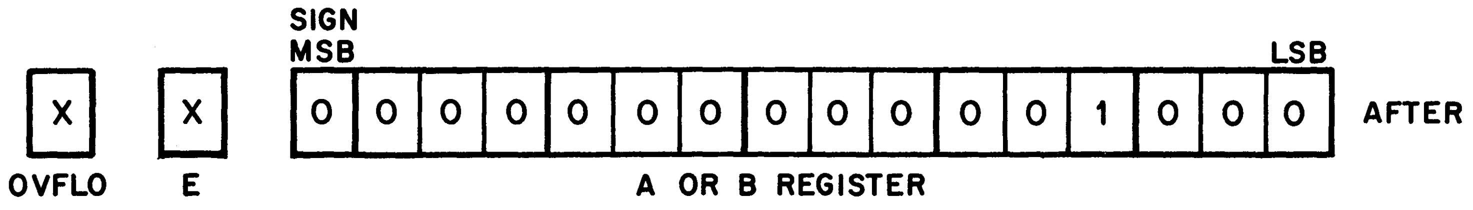


INSTRUCTION

[INA/INB]

X = 1 OR Ø

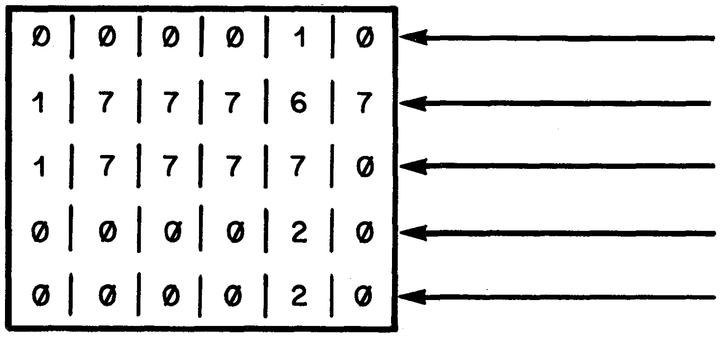
INCREMENT THE CONTENTS OF THE INDICATED REGISTER BY 1. OVERFLOW CAN BE SET AS A RESULT OF THIS OPERATION. IF A CARRY IS GENERATED FROM BIT 15, THE E REGISTER WILL BE SET TO 1 ALSO.



INCREMENT THE ACCUMULATOR

PROBLEM: compute Z = X − Y

REGISTER "A"



```
LABEL   OP CODE   OPERAND   REMARKS

          •
          •
        LDA       Y         LOAD Y
        CMA                 1'S COMPLEMENT
        INA                 2'S COMPLEMENT
        ADA       X         SUBTRACT
        STA       Ƶ         Ƶ=X−Y
        JMP       STOP      JUMP AROUND
                                  CONSTANTS
Y       OCT       1Ø
X       OCT       3Ø
Ƶ       OCT       Ø(2Ø)
STOP    CLA
          •
          •
          •
```

SUBTRACT EXAMPLE

INSTRUCTION

[CLE]

E $\boxed{1}$ OR $\boxed{0}$ **BEFORE**

CLEAR THE CONTENTS OF E, E IS RESET TO Ø, THE CONTENTS OF THE OTHER REGISTERS ARE NOT ALTERED BY ANY 'E' REGISTER INSTRUCTIONS.

E $\boxed{0}$ $\boxed{0}$ **AFTER**

INSTRUCTION

[CME]

E $\boxed{0}$ OR $\boxed{1}$ **BEFORE**

COMPLEMENT E,Ø BECOMES 1,1 BECOMES 0

E $\boxed{1}$ $\boxed{0}$ **AFTER**

INSTRUCTION

[CCE]

E $\boxed{0}$ OR $\boxed{1}$ **BEFORE**

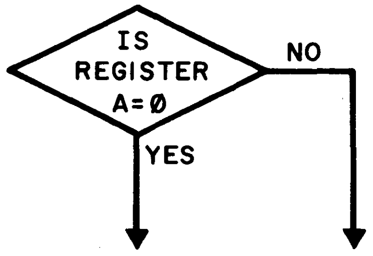
CLEAR AND THEN COMPLEMENT E

E $\boxed{1}$ $\boxed{1}$ **AFTER**

'E' REGISTER INSTRUCTIONS

THE ABILITY TO MAKE LIMITED DECISIONS BASED ON PRE-DEFINED CONDITIONS IS VERY IMPORTANT IN COMPUTER PROGRAMS.

FOR EXAMPLE



IN ORDER TO IMPLEMENT THE DECISION SYMBOL ONE OR MORE MACHINE INSTRUCTIONS ARE REQUIRED.

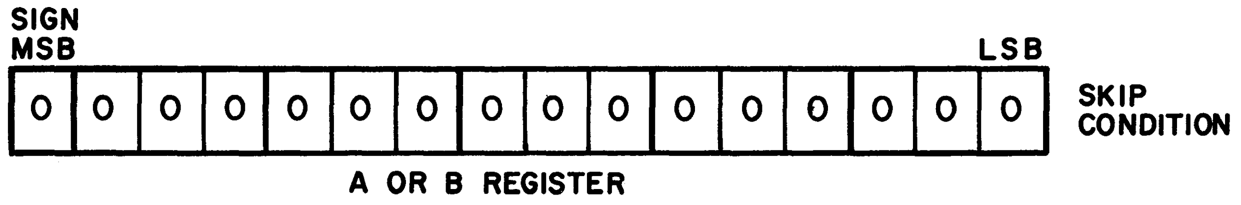
THE INSTRUCTION CODE TO IMPLEMENT THIS DECISION WOULD BE:

SZA (SKIP IF REG."A" IS ZERO)

```
LABEL  OPCODE  OPERAND  REMARKS
         :
        SZA             IS REG A=Ø?
        JMP    N ZERO   NO
        JMP      ZERO   YES
```

NOTE: ALL HP COMPUTER "SKIP-TYPE INSTRUCTIONS WORK IN THIS MANNER.

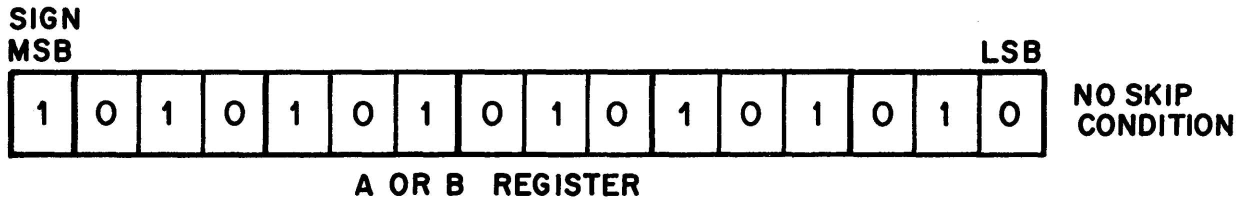
DECISION MAKING INSTRUCTIONS



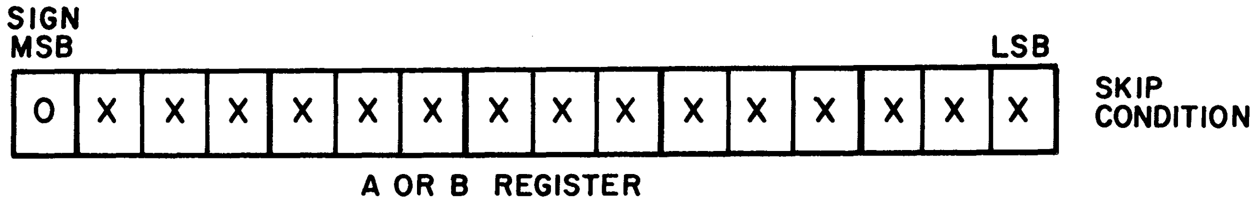
INSTRUCTION

[SZA/SZB]

THIS INSTRUCTION TESTS THE CONTENTS OF THE INDICATED REGISTER. IF THE TEST CONDITION IS PRESENT (16 Ø'S) THE NEXT SEQUENTIAL INSTRUCTION IS SKIPPED. ANY CONDITION OF THE REGISTER OTHER THAN 16 Ø'S CAUSES THE NEXT SEQUENTIAL INSTRUCTION TO BE EXECUTED. THE CONTENTS OF THE A, B, E, OR OVFLO REGISTERS ARE NOT AFFECTED BY THIS INSTRUCTION.



SKIP ON ZERO

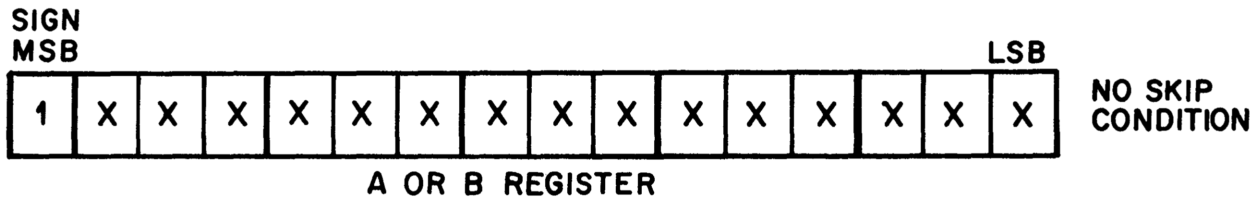


INSTRUCTION

[SSA/SSB]

X = 1 OR Ø

THIS INSTRUCTION TESTS THE CONTENTS OF BIT POSITION 15. IF BIT 15=Ø (POSITIVE) THE NEXT SEQUENTIAL INSTRUCTION IS SKIPPED. IF BIT POSITION 15=1 (NEGATIVE) THE NEXT SEQUENTIAL INSTRUCTION IS EXECUTED. THE CONTENTS OF A,B,E, OR OVFLO ARE NOT AFFECTED BY THIS INSTRUCTION.



SKIP SIGN POSITIVE

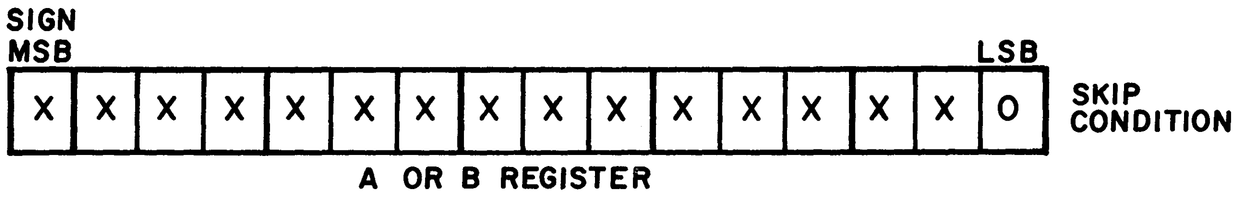
EXAMPLE

PROBLEM:

READ A 16 BIT VALUE FROM THE CONSOLE SWITCH REGISTER. IF THE VALUE IS POSITIVE TAKE THE 2's COMPLEMENT. IF THE VALUE IS NEGATIVE, CONTINUE THE PROGRAM.

SOLUTION:

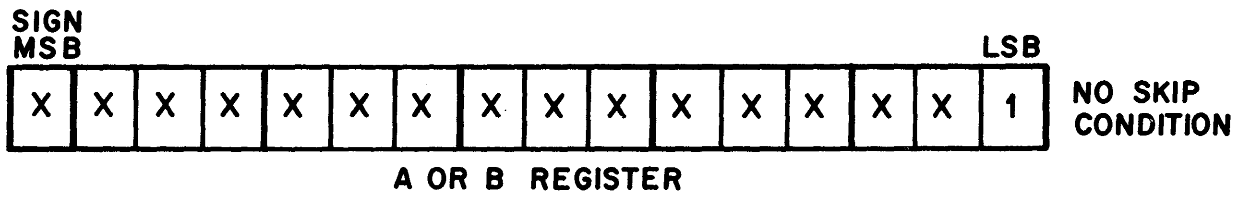
```
  Label     Operation        Operand          Remarks
1        5 6 7       10          15        20        25        30
      LIA 1    READ SWITCH REGISTER
      SSA      IS VALUE POSITIVE?
      JMP CONT NO
      CMA      YES,TAKE COMPLEMENT
      INA      ADD ONE
CONT  ——       CONTINUE PROGRAM
```



INSTRUCTION

[SLA/SLB]

THIS INSTRUCTION TESTS THE CONTENTS OF BIT POSITION Ø. IF THIS BIT IS Ø THE NEXT SEQUENTIAL INSTRUCTION IS SKIPPED. IF BIT POSITION Ø CONTAINS A 1 THE NEXT SEQUENTIAL INSTRUCTION IS EXECUTED. THE CONTENTS OF A, B, E, OR OVFLO ARE NOT AFFECTED BY THIS INSTRUCTION.



SKIP ON L.S.B. ZERO

INSTRUCTION

[SEZ] **SKIP THE NEXT SEQUENTIAL INSTRUCTION IF THE 'E' REGISTER IS Ø**

$\boxed{Ø}$ E **SKIP CONDITION**

$\boxed{1}$ E **NO SKIP CONDITION**

SEZ INSTRUCTION

INSTRUCTION

[RSS]

REVERSE THE SKIP 'SENSE' FOR ALL SKIP INSTRUCTIONS. AN RSS USED WITH A SKIP INSTRUCTION COMPLEMENTS THE SKIP CONDITION.

EXAMPLES:

RSS	=	**UNCONDITIONAL SKIP**
SEZ , RSS	=	**SKIP IF E ≠ Ø**
SZA , RSS	=	**SKIP IF A ≠ Ø**
SLB , RSS	=	**SKIP IF LSB OF B ≠ Ø**
SSA , RSS	=	**SKIP IF MSB OF A ≠ Ø**
SSA , SLA , RSS	=	**SKIP IF MSB AND LSB OF A = 1**

RSS INSTRUCTION

OP CODE

$$\left\{\begin{matrix}\text{CLA}\\ \text{CMA}\\ \text{CCA}\end{matrix}\right\}, [\text{SEZ}], \left\{\begin{matrix}\text{CLE}\\ \text{CME}\\ \text{CCE}\end{matrix}\right\}, [\text{SSA}], [\text{SLA}], [\text{INA}], [\text{SZA}], [\text{RSS}]$$

$$\left\{\begin{matrix}\text{CLB}\\ \text{CMB}\\ \text{CCB}\end{matrix}\right\}, [\text{SEZ}], \left\{\begin{matrix}\text{CLE}\\ \text{CME}\\ \text{CCE}\end{matrix}\right\}, [\text{SSB}], [\text{SLB}], [\text{INB}], [\text{SZB}], [\text{RSS}]$$

1. INSTRUCTIONS ARE COMBINED FROM LEFT TO RIGHT IN THE ORDER SHOWN
2. IF TWO OR MORE SKIP CONDITIONS ARE INCLUDED, A SKIP OCCURS IF EITHER OR BOTH CONDITIONS ARE PRESENT: EXCEPTION, SSA/B, SLA/B, RSS; BOTH CONDITIONS MUST BE MET.

COMBINING GUIDE ALTER - SKIP INSTRUCTIONS

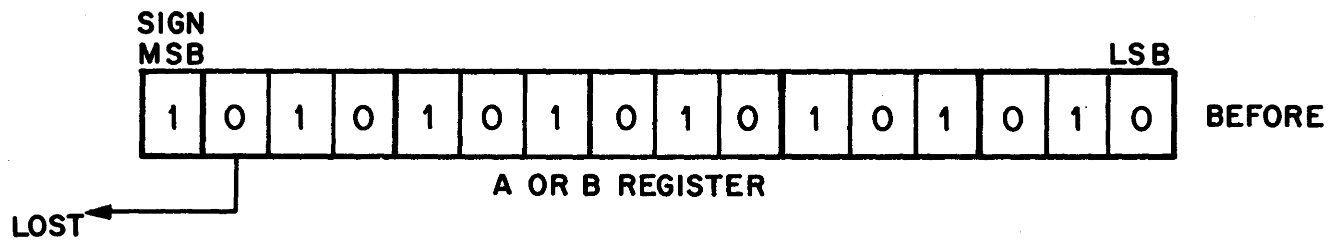
EXAMPLE

PROBLEM:

TEST A VALUE IN REGISTER "B".
IF ODD AND NEGATIVE JMP XYZ
IF EVEN OR POSITIVE JMP ABC

SOLUTION:

```
   Label      Operation        Operand        Remarks
1        5           10           15           20           25           30
      SSB,SLB,RSS
      JMP ABC
      JMP XYZ
```

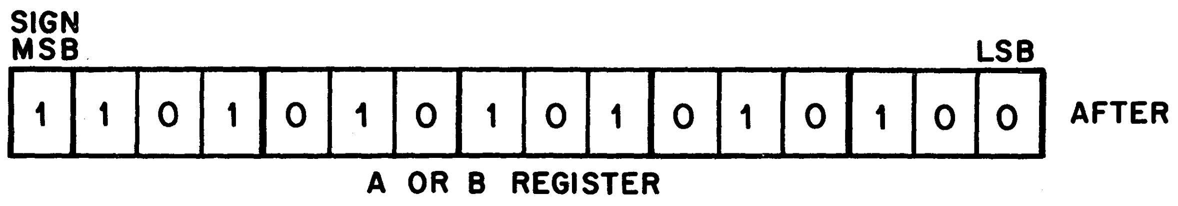


INSTRUCTION

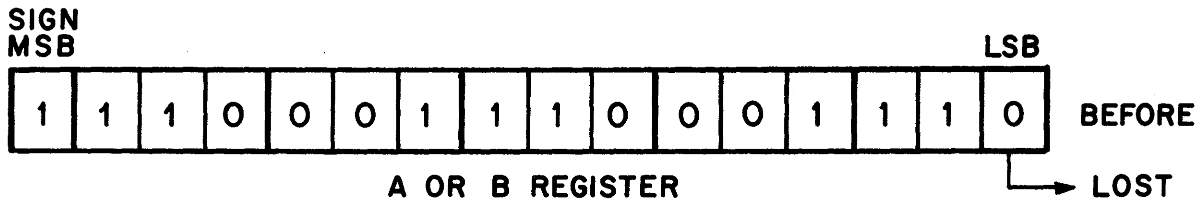
[ALS/BLS]

←

SHIFT THE INDICATED REGISTER LEFT 1 BIT ARITHMETICALLY, THAT IS, BITS Ø THRU 14 SHIFT LEFT. SIGN, BIT 15, IS NOT AFFECTED. BITS SHIFTED OUT OF BIT POSITION 14 ARE LOST. OVFLO, 'E' REGISTER ARE NOT AFFECTED.



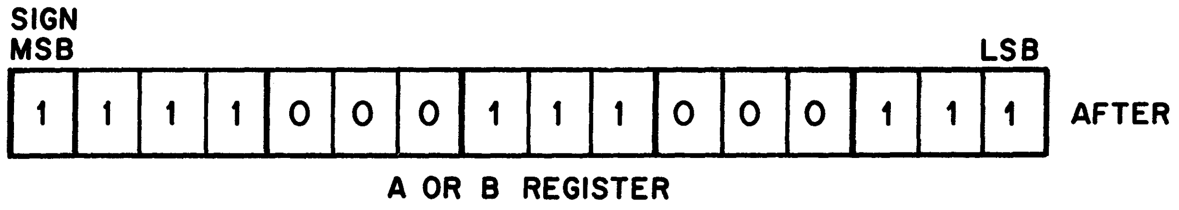
ACCUMULATOR LEFT SHIFT



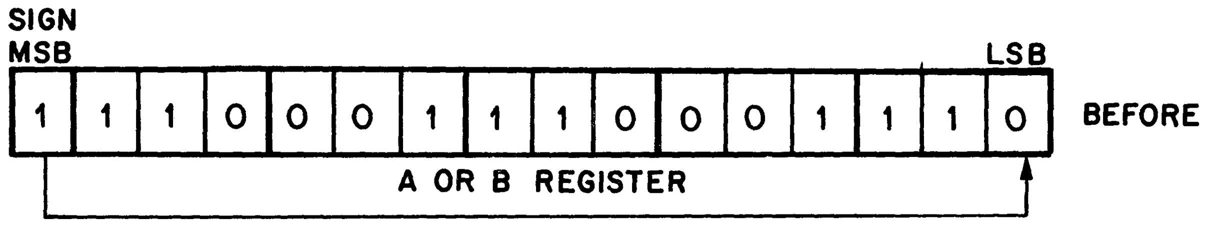
INSTRUCTION

[ARS/BRS] →

SHIFT THE INDICATED REGISTER RIGHT 1 BIT ARITHMETICALLY, THAT IS, BITS 14 THRU Ø SHIFT RIGHT. SIGN, BIT 15, IS NOT AFFECTED. BITS SHIFTED OUT OF BIT Ø ARE LOST. A COPY OF BIT 15 (SIGN) IS SHIFTED INTO BIT 14. OVFLO, 'E' REGISTER ARE NOT AFFECTED.

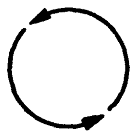


ACCUMULATOR RIGHT SHIFT

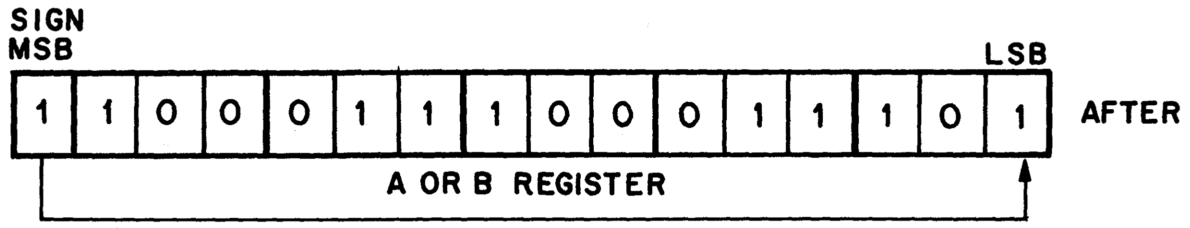


INSTRUCTION

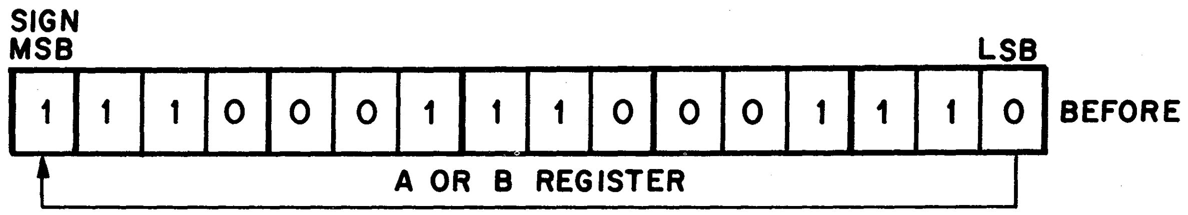
[RAL/RBL]



ROTATE THE INDICATED REGISTER LEFT 1 BIT. BIT 15 IS ROTATED AROUND TO BIT POSITION Ø. NO BITS ARE LOST. OVFLO, 'E' NOT AFFECTED.

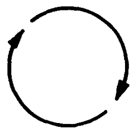


ROTATE ACCUMULATOR LEFT



INSTRUCTION

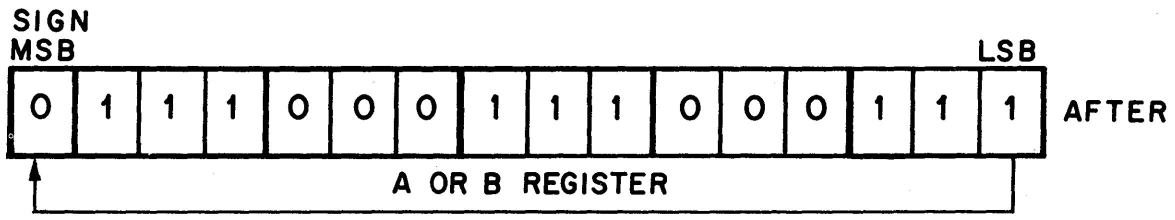
[RAR/RBR]



ROTATE THE INDICATED REGISTER RIGHT 1 BIT.

BIT Ø IS ROTATED AROUND TO BIT POSITION 15.

NO BITS ARE LOST. OVFLO, 'E' NOT AFFECTED.



ROTATE ACCUMULATOR RIGHT

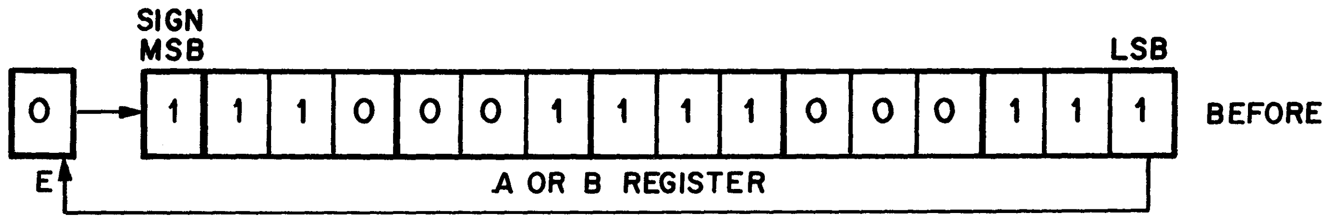
EXAMPLE

PROBLEM:

TEST BIT 15 OF THE "A" REGISTER.
IF BIT 15 = 1 , JMP TO LOCATION BUSY
IF BIT 15 = 0 , TEST BIT 14
IF BIT 14 = 1 , JMP TO LOCATION ERROR
IF BIT 14 = 0, (PROGRAM CONTINUATION)

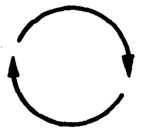
SOLUTION:

```
  Label     Operation      Operand     Remarks
1   5        10    15         20   25    30
      SSA          BIT 15=1
      JMP BUSY     YES
      RAL          NO
      SSA          BIT 14=1
      JMP ERROR    YES
      PROGRAM CONTINUATION
```

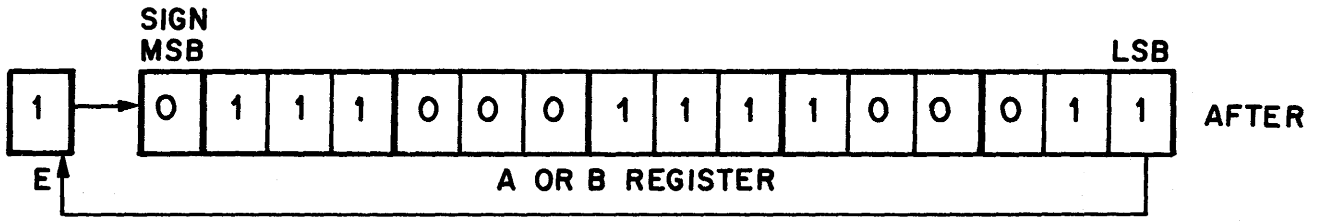


INSTRUCTION

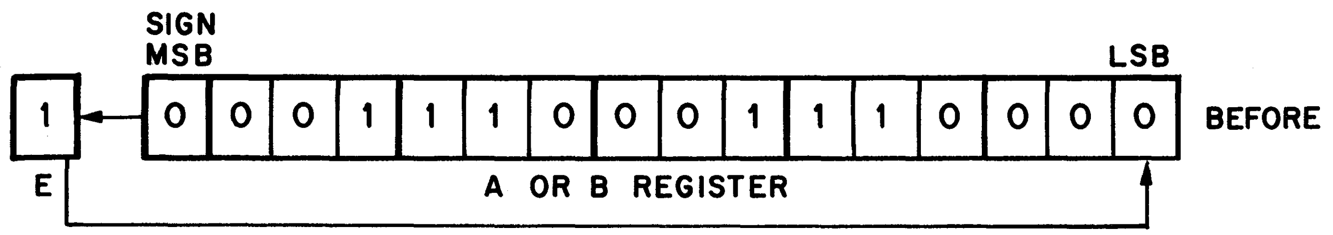
[ERA/ERB]



ROTATE THE INDICATED REGISTER RIGHT, 1 BIT, WITH THE EXTEND REGISTER ('E'). BIT Ø IS ROTATED INTO 'E' AND CONTENTS OF 'E' ARE ROTATED INTO BIT POSITION 15. NO BITS ARE LOST. OVFLO IS NOT AFFECTED.

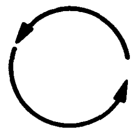


'E' RIGHT WITH ACCUMULATOR

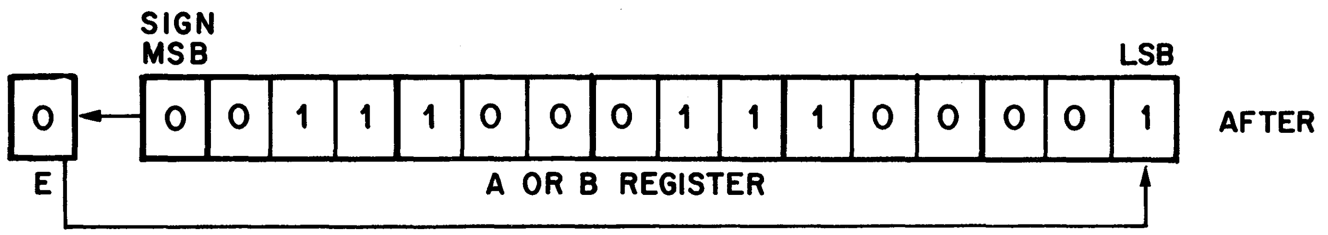


INSTRUCTION

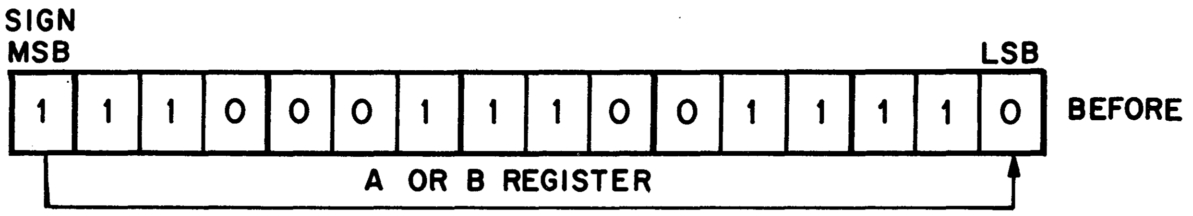
[ELA/ELB]



ROTATE THE INDICATED REGISTER LEFT, 1 BIT, WITH THE EXTEND REGISTER ('E'). BIT 15 IS ROTATED INTO 'E' AND CONTENTS OF 'E' ARE ROTATED AROUND TO BIT POSITION Ø. NO BITS ARE LOST. OVFLO IS NOT AFFECTED.

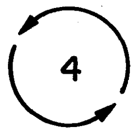


'E' LEFT WITH ACCUMULATOR

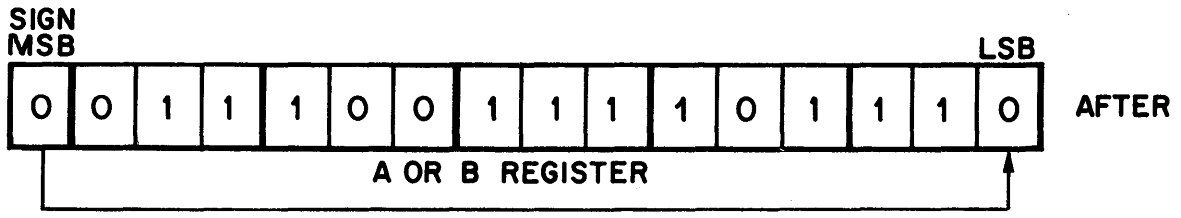


INSTRUCTION

[ALF/BLF]



ROTATE THE INDICATED REGISTER LEFT 4 PLACES. NO BITS ARE LOST. BIT 15, 14, 13, 12 ARE ROTATED AROUND TO BIT POSITIONS 3, 2, 1, 0 RESPECTIVELY. OVFLO, 'E' ARE NOT AFFECTED.



ACCUMULATOR LEFT ROTATE FOUR

$$\left\{\begin{matrix}\text{ALS}\\ \text{ARS}\\ \text{RAL}\\ \text{RAR}\\ \text{ALR}\\ \text{ALF}\\ \text{ERA}\\ \text{ELA}\end{matrix}\right\}, \left[\text{CLE}\right], \left[\text{SLA}\right], \left\{\begin{matrix}\text{ALS}\\ \text{ARS}\\ \text{RAL}\\ \text{RAR}\\ \text{ALR}\\ \text{ALF}\\ \text{ERA}\\ \text{ELA}\end{matrix}\right\}$$

$$\left\{\begin{matrix}\text{BLS}\\ \text{BRS}\\ \text{RBL}\\ \text{RBR}\\ \text{BLR}\\ \text{BLF}\\ \text{ERB}\\ \text{ELB}\end{matrix}\right\}, \left[\text{CLE}\right], \left[\text{SLB}\right], \left\{\begin{matrix}\text{BLS}\\ \text{BRS}\\ \text{RBL}\\ \text{RBR}\\ \text{BLR}\\ \text{BLF}\\ \text{ERB}\\ \text{ELB}\end{matrix}\right\}$$

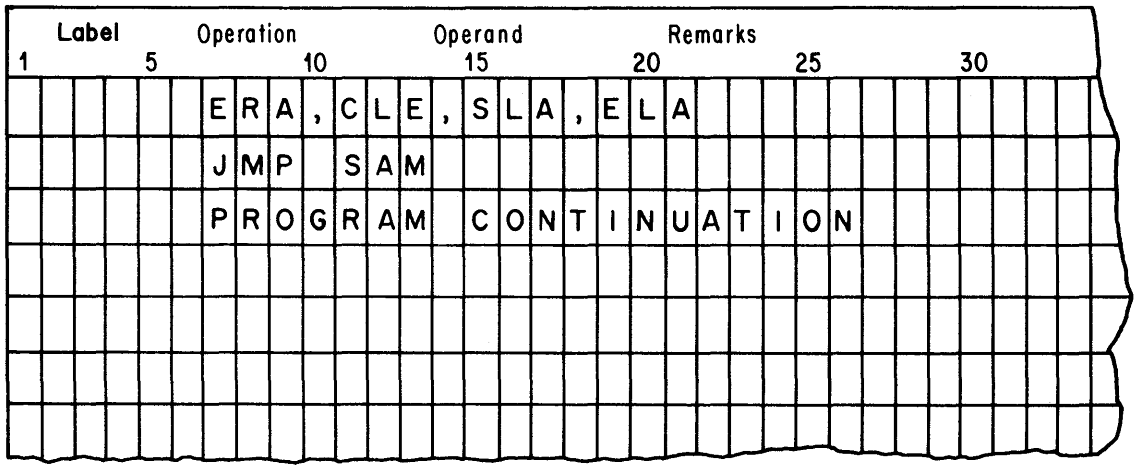
COMBINING GUIDE
SHIFT-ROTATE INSTRUCTIONS

EXAMPLE

PROBLEM:

CLEAR BIT Ø. TEST BIT 1 AND JUMP TO LOCATION "SAM" IF BIT 1 IS A ONE. RESTORE ALL BITS TO THEIR ORIGINAL POSITIONS IN THE REGISTER.

SOLUTION:

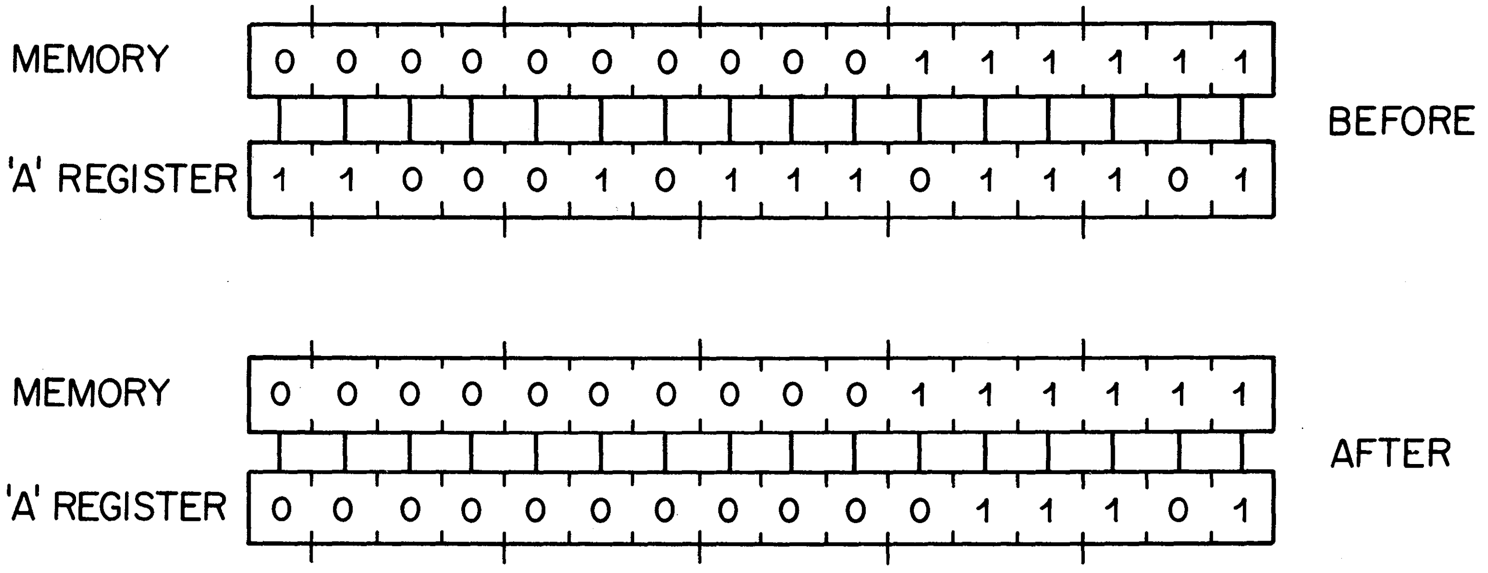


```
 Label     Operation        Operand        Remarks
1      5          10        15        20        25        30
      ERA,CLE,SLA,ELA
      JMP SAM
      PROGRAM CONTINUATION
```

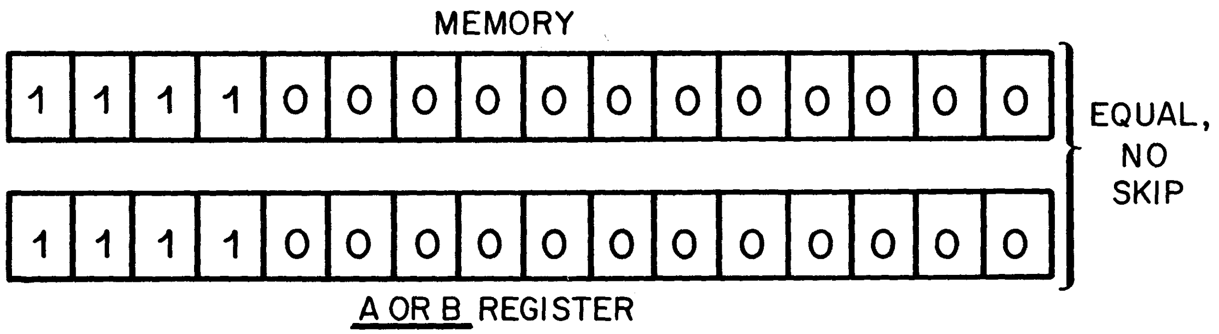
"A" REGISTER	MEMORY LOCATION	AND	IOR	XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

<u>LOGICAL TRUTH TABLE</u>

LABEL	OP CODE	OPERAND
	AND	MASK
	•	
	•	
MASK	OCT	77



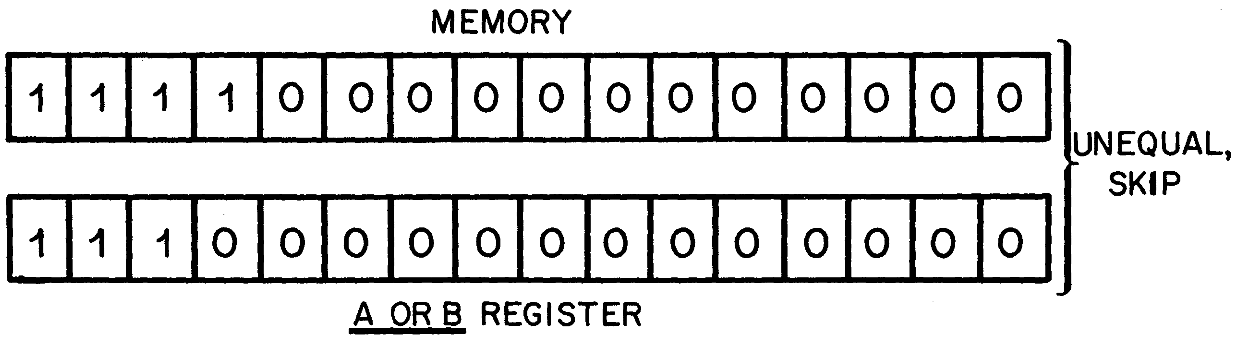
THE AND INSTRUCTION



INSTRUCTION

[CPA/B Y]

COMPARE THE CONTENTS OF THE SPECIFIED REGISTER AGAINST THE CONTENTS OF MEMORY LOCATION Y. IF ALL 16 BITS COMPARE (EQUAL) THE NEXT SEQUENTIAL INSTRUCTION IS EXECUTED. IF THE COMPARE FAILS, (UNEQUAL) THE NEXT SEQUENTIAL INSTRUCTION IS SKIPPED.



THE COMPARE INSTRUCTION

A COMPARE INSTRUCTION EXAMPLE

THE CONTENTS OF REGISTER "A" ARE UNKNOWN. DEVISE A PROGRAM SEGMENT THAT WILL TEST THE STATUS OF BITS 3 THROUGH 6. IF THIS FIELD CONTAINS THE OCTAL VALUE 12, TRANSFER TO A LABEL CALLED TRUE. IF THIS FIELD CONTAINS ANY OTHER VALUE THE PROGRAM SHOULD CONTINUE.

REGISTER "A" CONTENTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	?	?	?	?	X	X	X

```
1   5 6 7  10   15   20   25   30   35   40   45   50
      AND M17Ø      ISOLATE BITS 3 THROUGH6
      CPA M12Ø      DOES"A"COMPARE TO TEST VALUE?
      JMP TRUE      YES,JUMP TO TRUE ROUTINE
      JMP FALSE     NO,CONTINUE PROGRAM
M17Ø  OCT 17Ø       OCTAL MASK
M12Ø  OCT 12Ø       OCTAL TEST VALUE
FALSE ———
```

Assembler pseudo instructions VIII

LESSON VIII
ASSEMBLER PSEUDO INSTRUCTIONS

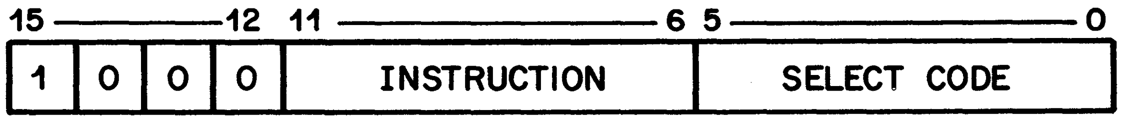
LESSON VIII OBJECTIVES

THE OBJECTIVES OF LESSON VIII ARE:

1 - INTRODUCE THE STUDENT TO BASIC I/O OPERATIONS.

2 - PROVIDE THE STUDENT WITH MORE ASSEMBLY LANGUAGE PROGRAMMING "TOOLS' IN THE FORM OF ASSEMBLY DIRECTING PSEUDO INSTRUCTIONS.

3 - INTRODUCE THE TECHNIQUES OF LOOPING AND INDIRECT ADDRESSING.



I/O INSTRUCTION FORMAT

INPUT/OUTPUT DEVICE	–	A PHYSICAL DEVICE CAPABLE OF TRANSMITTING AND/OR RECEIVING COMPUTER DATA.
I/O INTERFACE CARD	–	A COMPUTER ELECTRONICS CARD THAT PROVIDES THE PHYSICAL AND ELECTRICAL CONNECTION BETWEEN THE DEVICE AND THE COMPUTER.
I/O CHANNEL	–	THE RECEPTACLE IN THE I/O CARD CAGE THAT HOLDS THE I/O INTERFACE CARD.
SELECT CODE	–	IDENTIFIES A PARTICULAR I/O CHANNEL.
INTERRUPT LOCATION	–	A MEMORY LOCATION IN THE RANGE $4\text{-}77_8$, EACH SELECT CODE IDENTIFIES AN INTERRUPT LOCATION.
INTERRUPT	–	A PHASE OF COMPUTER OPERATION.

INTRODUCTION TO INPUT/OUTPUT

INPUT/ OUTPUT STRUCTURE

THE STRUCTURE OF THE *HEWLETT-PACKARD* COMPUTER PROVIDES 2 DISTINCT METHODS OF INPUT-OUTPUT DATA TRANSFER OPERATIONS.

1 - NON-INTERRUPT METHOD

The user commands the I/O device to cycle and then programs a loop that "waits" for the device cycle to complete.

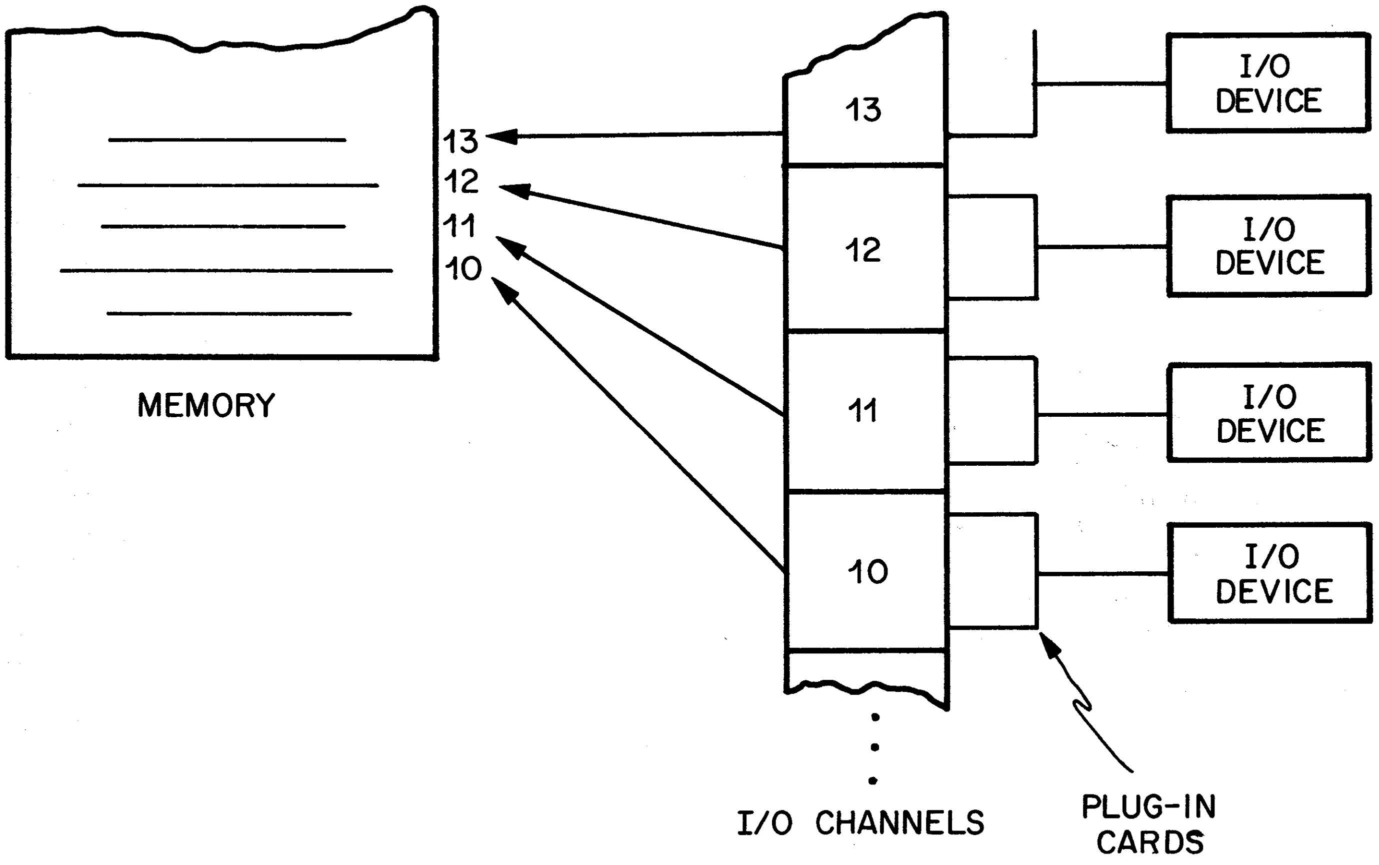
ADVANTAGE - easy to use
DISADVANTAGE - inefficient

2 - INTERRUPT METHOD

The user commands the I/O device to cycle and continues execution of the "main" program. The completion of the device cycle will interrupt the main program and transfer control to a subroutine that will handle the actual data transfer.

ADVANTAGE - efficient

DISADVANTAGE - requires more programming effort.

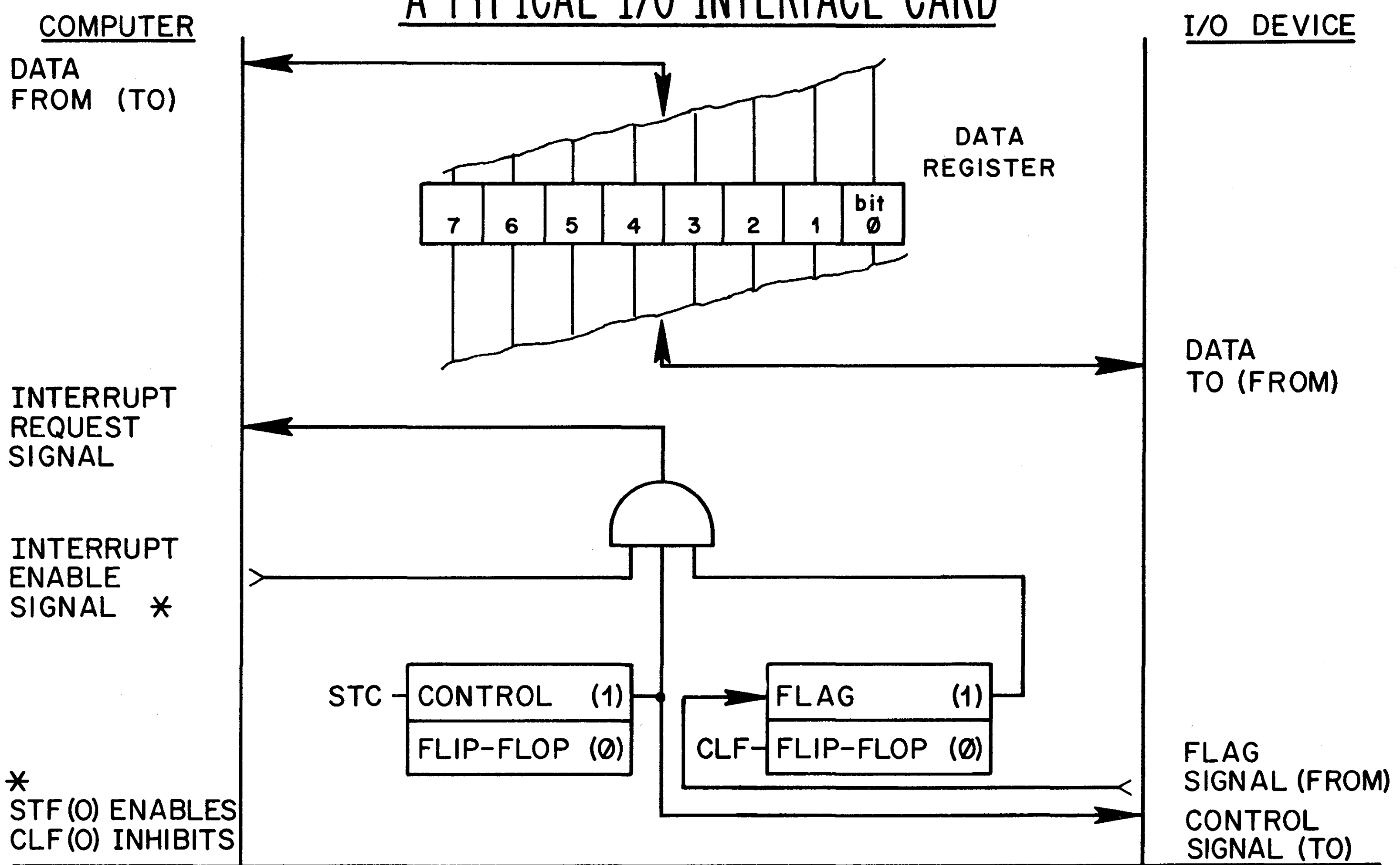


SELECT CODES AND INTERRUPT ADDRESSES

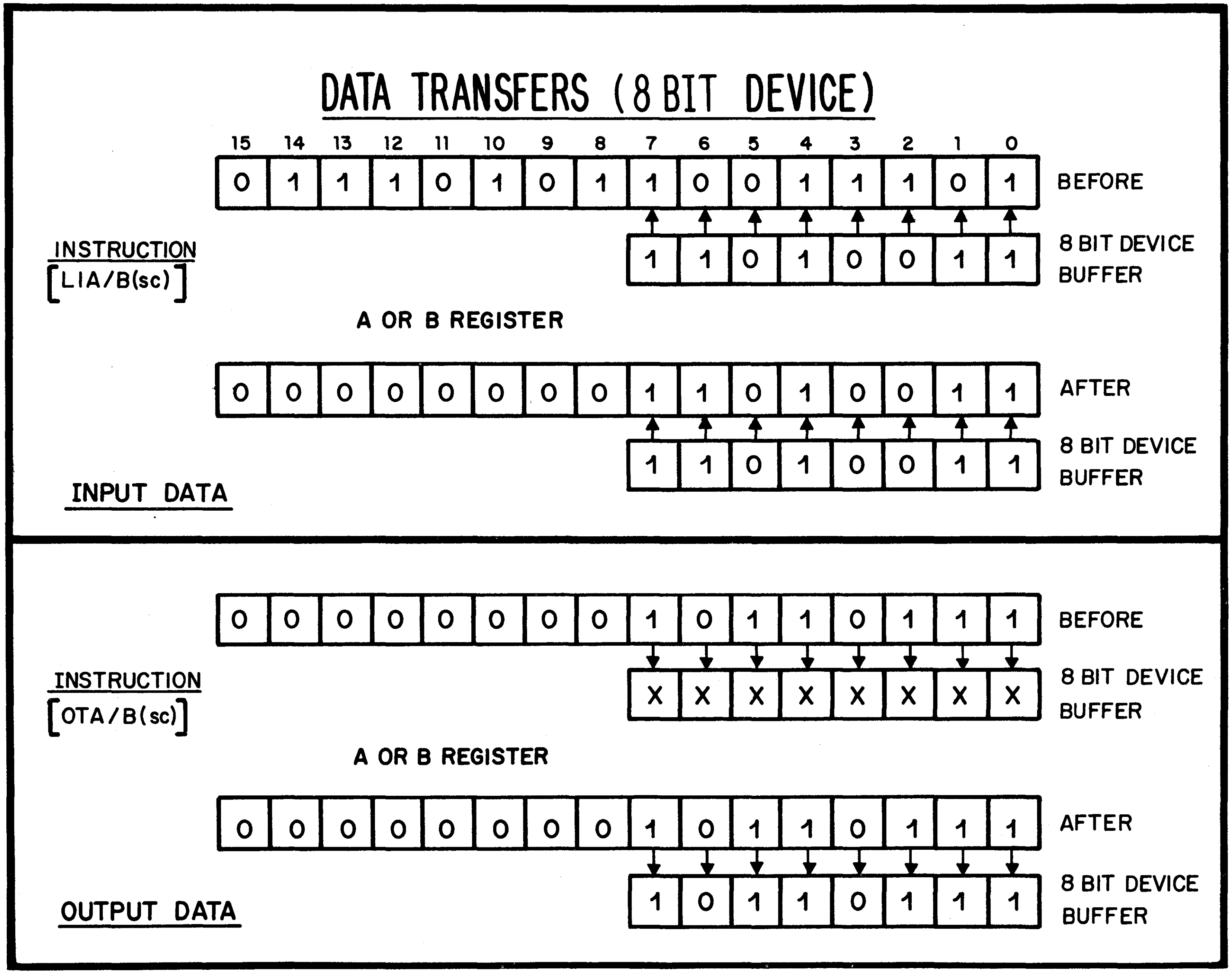
SELECT CODE	INTERRUPT LOCATION	FUNCTIONAL ASSIGNMENTS
0	NONE	ENABLE/DISABLE I/O AND INT. SYST.
1	NONE	SWITCH REGISTER
2	NONE	DMA CH 1
3	NONE	DMA CH 2
4	4 –	POWER FAIL
5	5 –	MEMORY PROTECT
6	6 –	DMA CH1
7	7 –	DMA CH2
10	10 –	I/O DEVICE HIGHEST PRIORITY
.	.	. .
.	.	. .
.	.	. .
.	.	. .
77	77	I/O DEVICE LOWEST PRIORITY

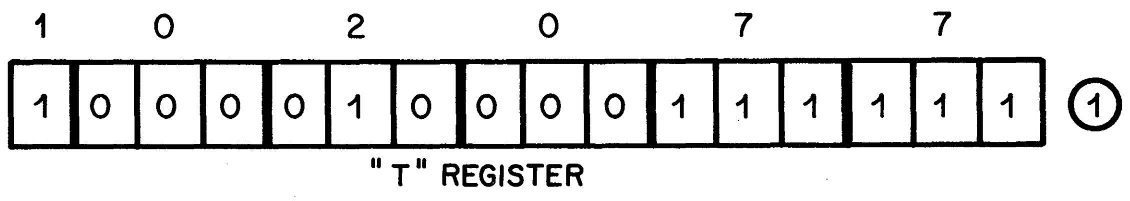
SELECT CODE ASSIGNMENTS

A TYPICAL I/O INTERFACE CARD



mnemonic	instruction	function
STC (sc)	SET CONTROL FLIP-FLOP	START DEVICE
CLC (sc)	CLEAR CONTROL " "	CLEAR DEVICE
STF (sc)	SET FLAG " "	SET FLAG, "READY"
CLF (sc)	CLEAR FLAG " "	CLEAR FLAG, "BUSY"


DATA TRANSFERS (8 BIT DEVICE)
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 1 1 1 0 1 0 1 1 0 0 1 1 1 0 1
BEFORE
INSTRUCTION
[LIA/B(sc)]
1 1 0 1 0 0 1 1
8 BIT DEVICE BUFFER
A OR B REGISTER
0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1
AFTER
1 1 0 1 0 0 1 1
8 BIT DEVICE BUFFER
INPUT DATA
0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 1
BEFORE
INSTRUCTION
[OTA/B(sc)]
X X X X X X X X
8 BIT DEVICE BUFFER
A OR B REGISTER
0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 1
AFTER
1 0 1 1 0 1 1 1
8 BIT DEVICE BUFFER
OUTPUT DATA



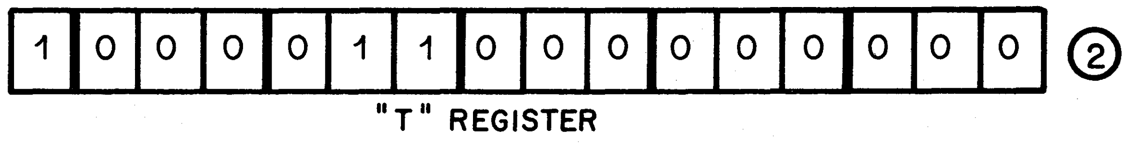
INSTRUCTION

[HLT (sc), C]

THIS INSTRUCTION WILL HALT THE COMPUTER. THE INSTRUCTION WILL BE DISPLAYED IN THE "T" REGISTER. THE (SC) OPTION ALLOWS THE SELECTION OF I/O ADDRESSES 0-77. THE, C OPTION ALLOWS THE FLAG BIT OF THE SELECTED DEVICE TO BE CLEARED.

1. SHOWS HALT INSTRUCTION DISPLAY; SC=77_8, NO (C) OPTION.

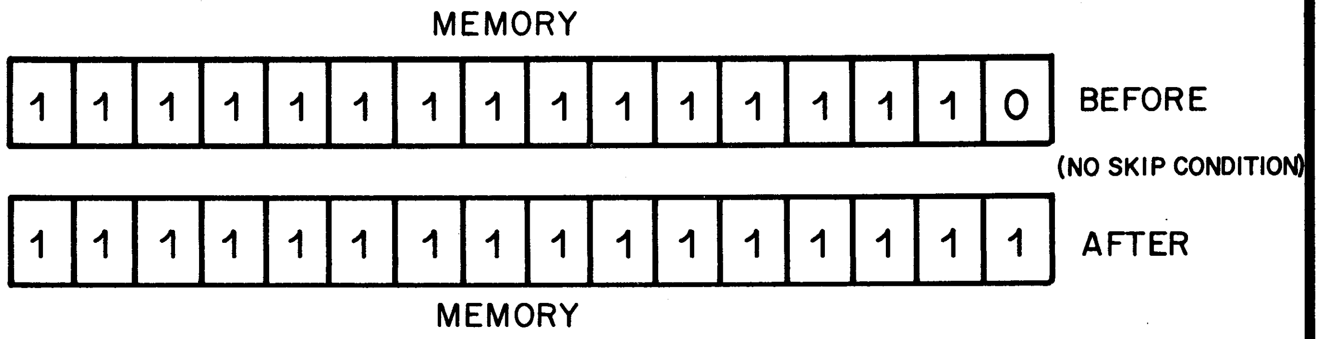
2. SHOWS HALT INSTRUCTION DISPLAY; WITH (C) OPTION TO CLEAR FLAG ON DEVICE ØØ (TURN OFF INT. SYST.)



THE HALT INSTRUCTION

```
LABEL     OP CODE     OPERAND     REMARKS
          CLF         Ø           INHIBIT THE INTERRUPT SYSTEM
          STC         1ØB, C      START READER
WAIT1     SFS         1ØB         FLAG = 1?
          JMP         WAIT1       NO
          LIA         1ØB         YES, LOAD DATA IN "A"
          ALF, ALF                POSITION IN HIGH "A"
          STC         1ØB, C      START READER
WAIT2     SFS         1ØB         FLAG = 1?
          JMP         WAIT2       NO
          MIA         1ØB         YES, MAKE 16 BIT WORD
```

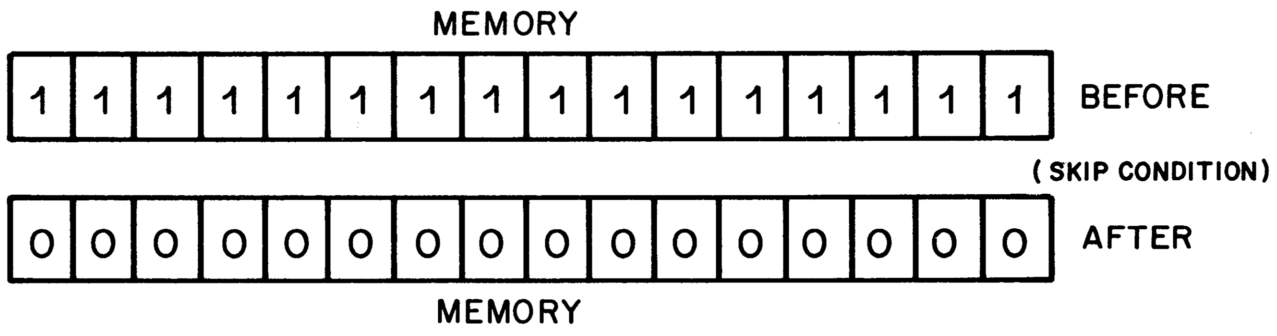
I/O DATA TRANSFER EXAMPLE



INSTRUCTION

[ISZ Y]

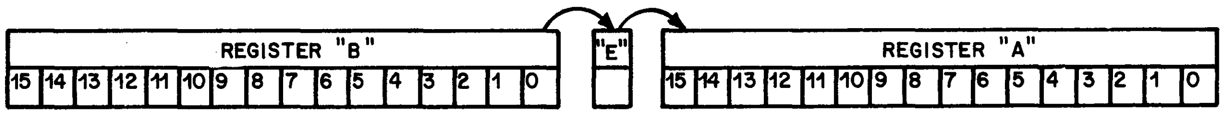
INCREMENT THE CONTENTS OF MEMORY LOCATION Y AND TEST FOR ZERO. IF Y IS EQUAL TO ZERO THE NEXT SEQUENTIAL INSTRUCTION IS SKIPPED. IF Y IS NOT EQUAL TO ZERO, THE NEXT SEQUENTIAL INSTRUCTION IS EXECUTED.

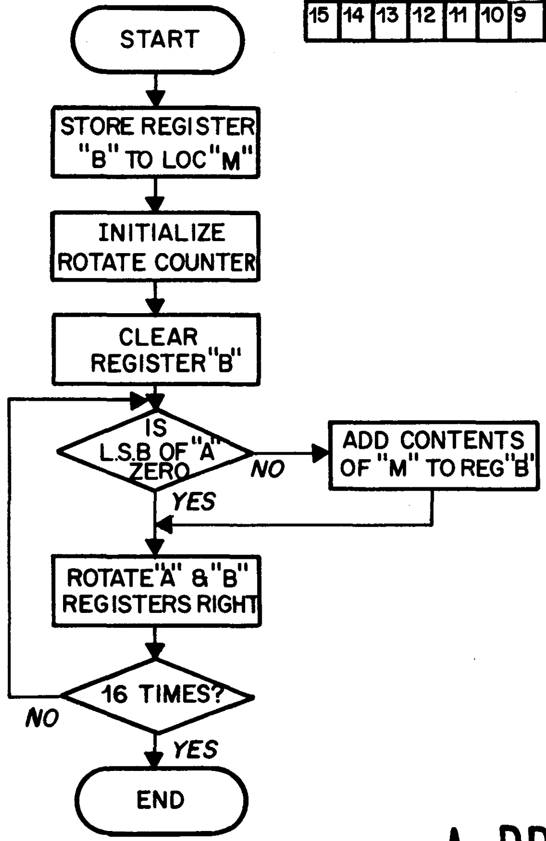


THE INCREMENT-SKIP ZERO INSTRUCTION

MULTIPLY THE CONTENTS OF REGISTER "B" BY THE CONTENTS OF REGISTER "A".

NOTE: Register "A" can be linked to Register "B" through Register "E".




START
STORE REGISTER "B" TO LOC "M"
INITIALIZE ROTATE COUNTER
CLEAR REGISTER "B"
IS L.S.B OF "A" ZERO
NO
ADD CONTENTS OF "M" TO REG "B"
YES
ROTATE "A" & "B" REGISTERS RIGHT
16 TIMES?
NO
YES
END

LABEL	OP CODE	OPERAND	REMARKS
START	STB	M	SAVE B
	LDB	M20	SET COUNTER TO MINUS 20_8
	STB	CNTR	
	CLB		CLEAR B
LOOP	CLE		
	SLA		
	ADB	M	
	ERB		
	ERA		
	ISZ	CNTR	ADD 1 TO CNTR, ZERO?
	JMP	LOOP	NO, STAY IN LOOP
	HLT		YES, HALT COMPUTER
	JMP	START	
M	OCT	Ø	
M20	OCT	177760	MINUS 20_8
CNTR	OCT	Ø	WORKING COUNTER

A PROGRAM LOOP EXAMPLE

THE B S S PSEUDO INSTRUCTION

<u>B</u>LOCK <u>S</u>TARTING <u>S</u>YMBOL

THIS PSEUDO WILL CAUSE THE ASSEMBLER TO ALLOCATE A BLOCK OF MEMORY LOCATIONS TO A PROGRAM. THE CONTENTS OF THE MEMORY BLOCK <u>CAN NOT</u> BE DETERMINED WHEN THE OBJECT PROGRAM IS LOADED FOR EXECUTION AND MUST BE TAKEN INTO CONSIDERATION BY THE PROGRAMMER.

<u>FOR EXAMPLE</u>:

```
LOCATION   CONTENTS   LABEL   OP CODE   OPERAND   REMARKS

00000                         NAM       BLOCK
                              ENT       START
00000      000000     BUFFR   BSS       512       SET ASIDE 512 MEMORY
01000      000000     START   NOP                 LOCATIONS
01001      002400             CLA
01002      006400             CLB
                               •
                               •
                               •
```

INDIRECT ADDRESSING

THE ASSEMBLER PROGRAM WILL SET THE INDIRECT ADDRESSING BIT (15) FOR ALL MEMORY REFERENCE OPERANDS TAGGED WITH THE ",I" DESIGNATOR.

FOR EXAMPLE:

LOCATION	CONTENTS	LABEL	OPCODE	OPERAND	REMARKS
			•		
2ØØØ	ØØ2Ø21	ADRES	OCT	2Ø21	OCTAL CONSTANT
•			•		
•			•		
•			•		
2Ø1Ø	Ø62ØØØ		LDA	ADRES	PICK UP OCTAL CONSTANT
2Ø11	162ØØØ		LDB	ADRES,I	PICK UP DECIMAL CONSTANT
•			•		
•			•		
•			•		
2Ø21	Ø77777		DEC	32767	DECIMAL CONSTANT
			•		
			•		
			END		

NOTE: AFTER EXECUTION OF CODING
REGISTER "A" = 002021
REGISTER "B" = 077777

THE DEF PSEUDO INSTRUCTION

THE DEF PSEUDO DEFINES THE MEMORY ADDRESS OF A PROPERLY DEFINED SYMBOL. THE ASSEMBLER GENERATES A 15 BIT MEMORY ADDRESS IN THE OBJECT PROGRAM WHEREVER THE DEF APPEARS.

FOR EXAMPLE:

```
 LOCATION   CONTENTS   LABEL   OPCODE   OPERAND   REMARKS

 00000                          NAM      SAMPL
 00000      000114R    ADRES    DEF      TABLE     DEF ADDRESS OF TABLE
                                ENT      START
 00001      000000     START    NOP
 00002      066000R             LDB      ADRES     GET ADDRESS OF TABLE
 00003      160001              LDA      1,I       LOAD "A" THRU "B"
   •          •                  •        •        (GET FIRST TABLE VALUE)
   •          •                  •        •
   •          •                  •        •
   •          •                  •        •
 00114      000000     TABLE    BSS      100
                                END      START
```

PSEUDO INSTRUCTIONS ENT AND EXT

ENTRY POINT AND EXTERNAL PSEUDO INSTRUCTIONS PROVIDE OBJECT PROGRAM LINKAGE CAPABILITY.

FOR EXAMPLE:

PROGRAM "A" IS TO BE EXECUTED FIRST WITH CONTROL THEN PASSING TO PROGRAM "B".

SEGMENT 1

```
LABEL     OPCODE    OPERAND

          NAM       PROGA
          ENT       START
          EXT       SAM
START     NOP
          •
          •
          •
          JMP       SAM
          END       START
```

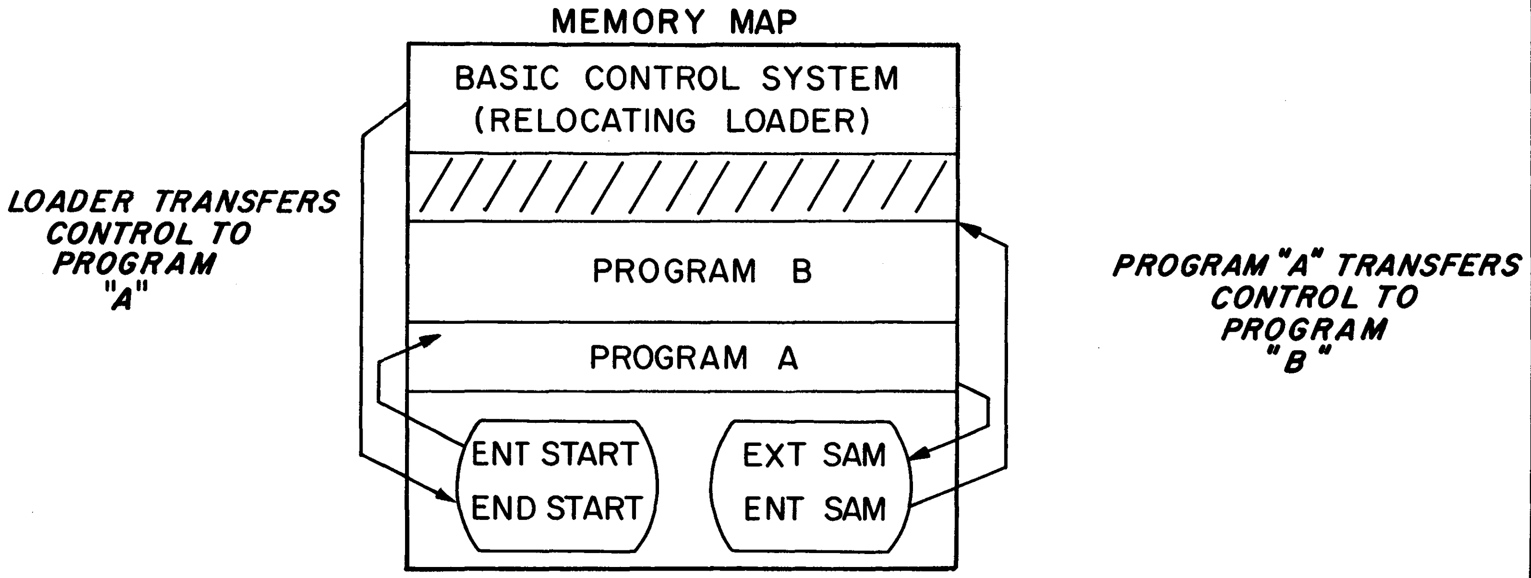
SEGMENT 2

```
LABEL     OPCODE    OPERAND

          NAM       PROGB
          ENT       SAM
SAM       CLA
          •
          •
          •
          END
```

OBJECT PROGRAM LINKAGE

OBJECT PROGRAM LINKAGE IS ACCOMPLISHED BY THE RELOCATING LOADER. THE LOADER CREATES LINKAGES ON THE BASE PAGE BY MATCHING "ENT" POINTS WITH "EXT" SYMBOLS. THE LOADER TRANSFERS TO THE LAST PROGRAM LOADED THAT HAS A VALID "END" RECORD.

FOR EXAMPLE:


MEMORY MAP
BASIC CONTROL SYSTEM (RELOCATING LOADER)
PROGRAM B
PROGRAM A
ENT START
END START
EXT SAM
ENT SAM
LOADER TRANSFERS CONTROL TO PROGRAM "A"
PROGRAM "A" TRANSFERS CONTROL TO PROGRAM "B"

THE JUMP SUBROUTINE INSTRUCTION (JSB)

THE JUMP SUBROUTINE INSTRUCTION (JSB) PROVIDES A METHOD TO EXECUTE A "SUBROUTINE" AND RETURN TO THE PROPER POINT IN THE "MAIN PROGRAM". TO PERFORM THIS FUNCTION 3 DISTINCT OPERATIONS ARE REQUIRED.

① - PRESERVE THE RETURN ADDRESS.

② - TRANSFER CONTROL TO THE SUBROUTINE.

③ - RETURN TO THE "MAIN PROGRAM".

EXAMPLE:

MAIN PROGRAM

LOCATION	LABEL	OP CODE	OPERAND
100		LDA	I
101		JSB	CMP
102		ADA	J
103		HLT	
104	I	OCT	1
105	J	OCT	7

SUBROUTINE

LOCATION	LABEL	OP CODE	OPERAND
200	CMP	102 ~~NOP~~	
201		CMA	
202		INA	
203		JMP	CMP,I

A JSB EXAMPLE

A SUBROUTINE TO CLEAR THE "A" AND "B" REGISTERS IS SHOWN AS AN EXAMPLE. THE SUBROUTINE IS "ENTERED" FROM 3 DIFFERENT POINTS IN THE "MAIN PROGRAM."

MAIN PROGRAM

LOCATION	LABEL	OP CODE	OPERAND
2000		SSA	
2001		JSB	CLEAR
2002		INA	
•		•	
•		•	
•		•	
2077		JSB	CLEAR
2100		ADA	J
2101		ADA	K
•		•	
•		•	
•		•	
2500		JSB	CLEAR
2501		HLT	

SUBROUTINE

LOCATION	LABEL	OP CODE	OPERAND
3000	CLEAR	NOP	
3001		CLA	
3002		CLB	
3003		JMP	CLEAR, I

THE EQU PSEUDO INSTRUCTION

THE EQU PSEUDO INSTRUCTION EQUATES THE OPERAND SYMBOL TO THE LABEL FIELD.

FOR EXAMPLE:

```
LOCATION   CONTENTS   LABEL   OPCODE   OPERAND    REMARKS

00000                         NAM      CHAR
00017                 READR   EQU      17B        READR "EQUALS" 17B
                              ENT      START
00000      000000     START   NOP
00001      103717             STC      READR,C
00002      102317             SFS      READR
00003      026002R            JMP      *-1
00004      102517             LIA      READR
00005      126000R            JMP      START,I
```

NOTE:

THE VALUE OF THE LABEL "READR" DEPENDS ENTIRELY ON THE OPERAND OF THE EQU PSEUDO INSTRUCTION. ALL OPERAND REFERENCES TO THE "READR" SYMBOL ARE ASSIGNED THE VALUE 17_8.

THE COM PSEUDO INSTRUCTION

- THE COM PSEUDO RESERVES A BLOCK OF STORAGE LOCATIONS THAT MAY BE USED IN COMMON BY SEVERAL SUBPROGRAMS.

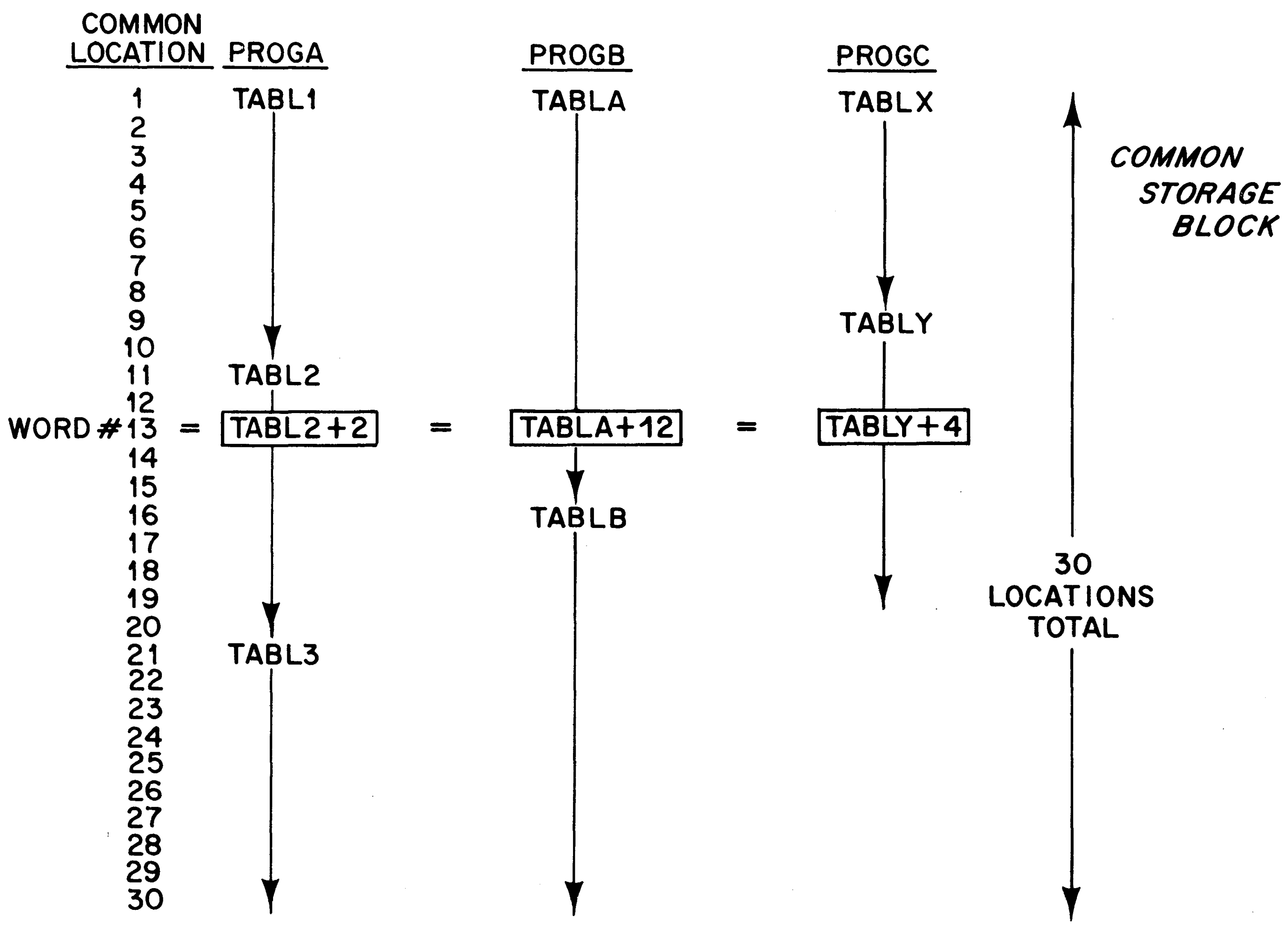
  GENERAL FORM:

COM	$\text{name}_1\ (\text{size}_1),\ \text{name}_2\ (\text{size}_2), \ldots,\ \text{name}_n\ (\text{size}_n)$	REMARKS

- EACH NAME IDENTIFIES A SEGMENT OF THE BLOCK FOR THE SUBPROGRAM IN WHICH THE COM STATEMENT APPEARS.
- STORAGE LOCATIONS ARE ASSIGNED CONTIGUOUSLY
- THE LENGTH OF THE BLOCK IS EQUAL TO THE SUM OF THE LENGTHS OF ALL SEGMENTS NAMED IN ALL COM STATEMENTS IN THE SUBPROGRAM.
- TO REFER TO THE COMMON BLOCK OTHER SUBPROGRAMS MUST ALSO INCLUDE A COM STATEMENT.
- AT LOAD TIME; THE SUBPROGRAM WITH THE GREATEST COMMON DECLARATION MUST BE LOADED FIRST.

```
          LABEL       OP CODE        OPERAND

                      NAM            PROGA
S         START       NOP
E                     LDA            TABL1
G                      :
M                      :
E                     COM            TABL1 (1Ø) ,TABL2 (1Ø), TABL3 (1Ø)
N                      :
T                      :
"A"                   END            START
--------------------------------------------------------------------
S
E                     NAM            PROGB
G                     COM            TABLA (15),  TABLB (15)
M                      :
E                      :
N                      :
T                     END
"B"
--------------------------------------------------------------------
S                     NAM            PROGC
E                     STA            TABLX
G                      :
M                      :
E                      :
N                     COM            TABLX (8), TABLY (11)
T                      :
"C"                    :
                      END
```

USING THE COM PSEUDO



COMMON STORAGE

ASCII CHARACTER CODES

TAPE CHANNELS $\underbrace{87}$ $\underbrace{654}$ * $\underbrace{321}$

OCTAL CODE

	0	1	2	3	4	5	6	7
00	NULL	SOM	EOA	EOM	EOT	WRU	RU	BELL
01	FE	H.TAB	LF	V. TAB	FORM	CR	SO	SI
02	DC	X-ON	TAPE ON	X-OFF	TAPE OFF	ERROR	SYNC	LEM
03	SØ	S1	S2	S3	S4	S5	S6	S7
04	SPACE	!	"	#	$	%	&	'
05	(	)	*	+	,	–	.	/
06	Ø	1	2	3	4	5	6	7
07	8	9	:	;	<	=	>	?
10	@	A	B	C	D	E	F	G
11	H	I	J	K	L	M	N	O
12	P	Q	R	S	T	U	V	W
13	X	Y	Z	[	\	]	↑	←
14								
15								
16								
17					ACK	ALT MODE	ESC	RO

Example: Character 'S'

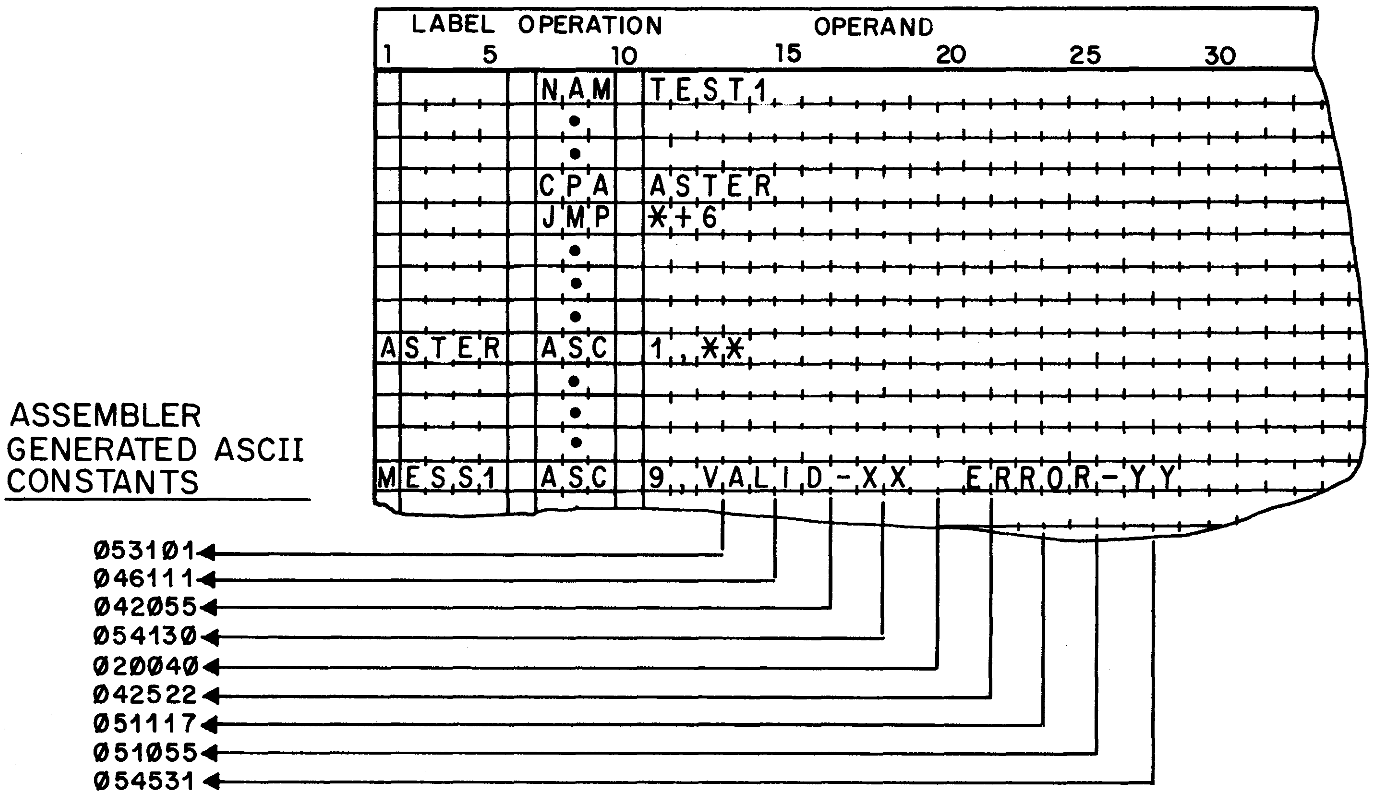
Octal	1	2	3
Binary	01	010	011
Tape Channels	87	654 *	321

(*Feed hole)

THE ASC PSEUDO INSTRUCTION

THE ASC PSEUDO IS USED TO DEFINE ALPHANUMERIC CONSTANTS.

FOR EXAMPLE:



```
        NAM   TEST1
         •
         •
        CPA   ASTER
        JMP   *+6
         •
         •
         •
ASTER   ASC   1,**
         •
         •
         •
MESS1   ASC   9,VALID-XX  ERROR-YY
```

```
Ø531Ø1
Ø46111
Ø42Ø55
Ø5413Ø
Ø2ØØ4Ø
Ø42522
Ø51117
Ø51Ø55
Ø54531
```

Assembler programming techniques IX

LESSON IX
ASSEMBLER PROGRAMMING TECHNIQUES

LESSON IX OBJECTIVES

THE OBJECTIVES OF LESSON IX ARE TO TEACH THE STUDENT SOME BASIC PROGRAMMING TECHNIQUES.

THESE INCLUDE:

- **ADDRESS MODIFICATION**
- **SUBROUTINES**
- **ARITHMETIC PSEUDO INSTRUCTIONS**
- **INPUT/OUTPUT TECHNIQUES (FORTRAN I/O)**

ADDRESS MODIFICATION

ADDRESS MODIFICATION IS AN IMPORTANT PROGRAMMING TECHNIQUE.

FOR EXAMPLE: A PROGRAM TO SUM THE CONTENTS OF 1Ø SEQUENTIAL MEMORY LOCATIONS.

```
 LABEL   OPCODE OPERAND              REMARKS
1    5         10    15       20       25      30
        NAM    MOD1
B       EQU    1
        ENT    START          ENTRY POINT
START   NOP
        LDA    CNT            INITIALIZE
        STA    CNTR           COUNTER
        CLA                   CLEAR A
        LDB    PNTR           LOAD ADDRESS OF TABLE
LOOP    ADA    B,I            ADD INDIRECTLY THRU "B"
        INB                   ADD 1 TO ADDRESS
        ISZ    CNTR           IS COUNTER ZERO?
        JMP    LOOP           NO, CONTINUE
        HLT    77B            YES, HALT COMPUTER
        JMP    START+1        PROGRAM RESTART
CNT     DEC    -1Ø            COUNT VALUE
CNTR    BSS    1              WORKING COUNTER
PNTR    DEF    TABLE          ADDRESS OF TABLE
TABLE   DEC    1Ø,5,15,23,752,8272,33,84,9,28
        END    START
```

SUBROUTINES

SUBROUTINES ARE WRITTEN TO DO A SPECIFIC JOB. MOST SUBROUTINES REQUIRE DATA (PARAMETERS) FROM THE MAIN PROGRAM.

FOR EXAMPLE:

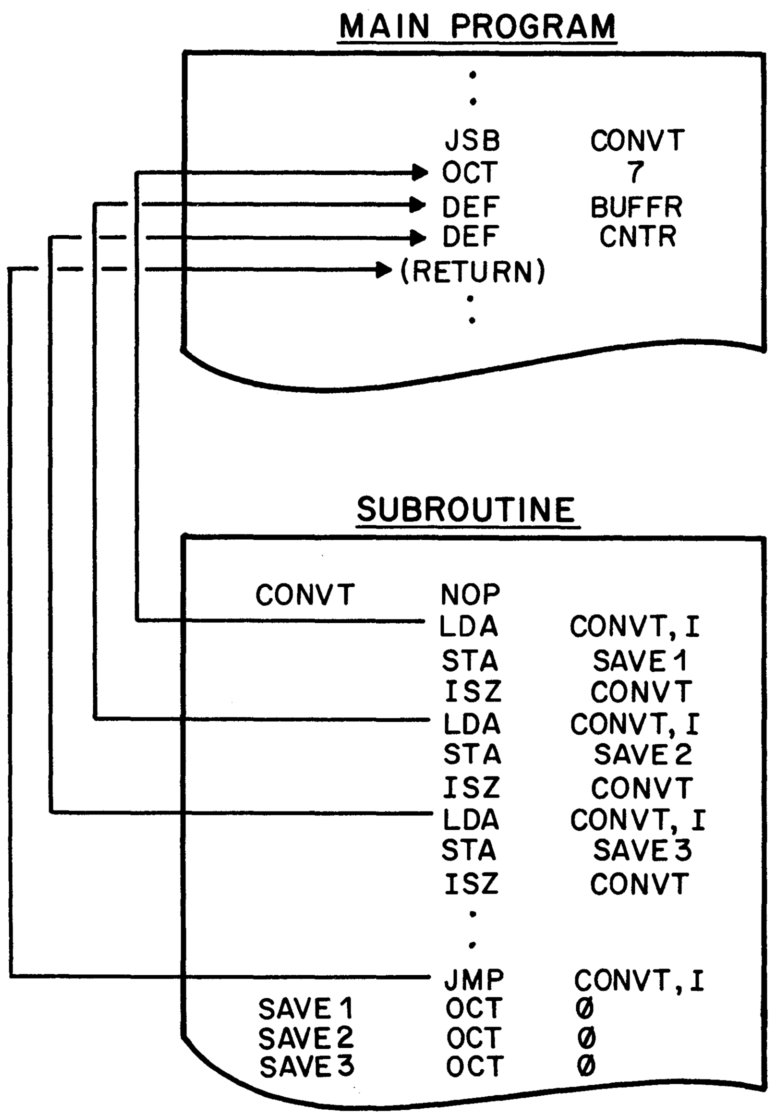
A SUBROUTINE TO COMPUTE THE ABSOLUTE VALUE OF THE CONTENTS OF REGISTER "A."

LABEL	OP CODE	OPERAND	REMARKS
IABS	NOP		
	SSA, RSS		A < Ø
	JMP	IABS,I	NO, A = ANSWER
	CMA, INA		YES, COMPLEMENT VALUE
	SSA		DID A = 100000
	CMA		YES, SET A=077777
	JMP	IABS,I	NO, A=ANSWER

	LABEL	OP CODE	OPERAND	REMARKS
ONE PARAMETER CALLING SEQUENCE		LDA	NUMBR	CONVERT (NUMBR)
		JSB	IABS	TO ITS ABSOLUTE VALUE
		(RETURN)		
TWO PARAMETER CALLING SEQUENCE		LDA	ADDRS	BUFFER ADDRESS IN "A" REGISTER
		LDB	COUNT	COUNT IN "B" REGISTER
		JSB	READ	TRANSFER TO READ ROUTINE
		(RETURN)		
THREE OR MORE PARAMETER CALLING SEQUENCE		JSB	CONVT	TRANSFER TO CONVERT ROUTINE
		OCT	7	CODE WORD
		DEF	BUFFR	BUFFER ADDRESS
		DEF	CNTR	COUNTER
		(RETURN)		

CALLING SEQUENCE

(TECHNIQUES FOR TRANSFER OF PARAMETER DATA TO SUBROUTINES)



MAIN PROGRAM

```
           .
           .
           JSB      CONVT
           OCT      7
           DEF      BUFFR
           DEF      CNTR
          (RETURN)
           .
           .
```

SUBROUTINE

```
CONVT      NOP
           LDA      CONVT,I
           STA      SAVE1
           ISZ      CONVT
           LDA      CONVT,I
           STA      SAVE2
           ISZ      CONVT
           LDA      CONVT,I
           STA      SAVE3
           ISZ      CONVT
           .
           .
           JMP      CONVT,I
SAVE1      OCT      Ø
SAVE2      OCT      Ø
SAVE3      OCT      Ø
```

DIRECT TRANSFER OF PARAMETERS

FORTRAN COMPATIBLE ASSEMBLER SUBROUTINES

PART 1 - THE FORTRAN CALL

FORTRAN MAIN PROGRAMS MAY COMMUNICATE WITH ASSEMBLY LANGUAGE SUBROUTINES. THIS FEATURE IS POSSIBLE ONLY IF THE SUBROUTINE IS COMPATIBLE WITH THE STANDARD FORTRAN CALLING SEQUENCE.

FOR EXAMPLE:

```
   FORTRAN              GENERATED ASSEMBLY LANGUAGE CODING

                                        REMARKS

                   JSB  SAM   TRANSFER TO "SAM"
                   DEF  *+4   DEFINE RETURN ADDRESS
CALL SAM(J,K,L)    DEF  J     DEFINE ADDRESS OF J
                   DEF  K     DEFINE ADDRESS OF K
                   DEF  L     DEFINE ADDRESS OF L
                   (RETURN)
```

THE ACTUAL TRANSFER OF DATA ITEMS IS THE RESPONSIBILITY OF THE SUBROUTINE

FORTRAN COMPATIBLE ASSEMBLER SUBROUTINES

PART 2 - THE ASSEMBLY LANGUAGE SUBROUTINE

A FORTRAN LIBRARY ROUTINE CALLED .ENTR MAY BE USED TO TRANSFER THE ADDRESSES OF THE PARAMETERS FROM THE MAIN PROGRAM TO THE SUBROUTINE.

FOR EXAMPLE:

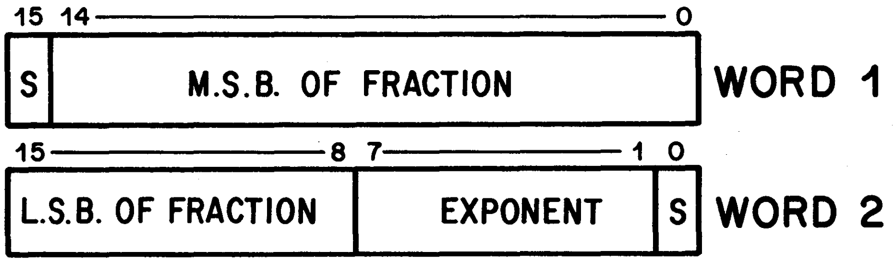
```
LABEL    OPCODE  OPERAND          REMARKS

         NAM     SAM
         ENT     SAM
         EXT     .ENTR
J        BSS     1         LOCATION FOR ADDRESS OF PARAMETER  J
K        BSS     1            "      "     "     "     "      K
L        BSS     1            "      "     "     "     "      L
SAM      NOP               ENTRY POINT
         JSB     .ENTR     TRANSFER TO .ENTR
         DEF     J         DEFINE ADDRESS OF FIRST PARAMETER
         LDA     J,I
         ADA     K,I
         STA     L,I
         JMP     SAM,I     RETURN TO MAIN PROGRAM
```

NOTE: MAIN PROGRAM AND SUBROUTINE PARAMETERS MUST AGREE AS TO TYPE:

REAL OR INTEGER

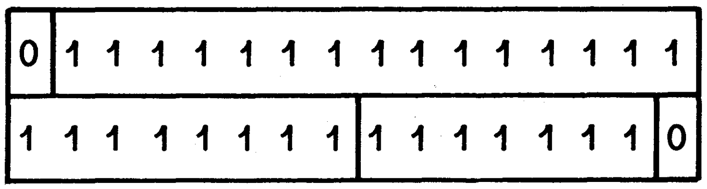
FLOATING POINT NUMBERS

FLOATING POINT NUMBERS USE TWO MEMORY LOCATIONS IN THE FOLLOWING FORM:



FLOATING POINT NUMBERS HAVE A RANGE IN MAGNITUDE OF APPROXIMATELY -10^{38} TO 10^{38}

FOR EXAMPLE: THE GREATEST POSITIVE FLOATING POINT NUMBER WOULD BE:



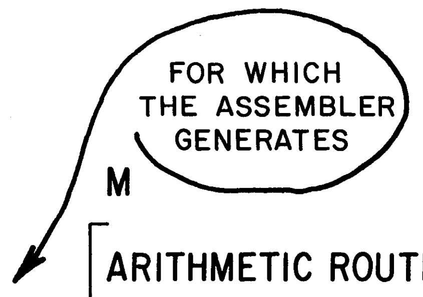
THE FRACTIONAL VALUE (MANTISSA) OF THE FLOATING POINT NUMBER IS ALWAYS IN THE RANGE $0 \leq n < 1$. IN THE EXAMPLE ABOVE THE VALUE OF THE MANTISSA IS APPROXIMATELY 1. THE EXPONENT VALUE IS 2^7-1 (127_{10}), THE VALUE OF THE NUMBER IS 1×2^{127}; BY RAISING 2 TO THE 127th POWER (USING A LOG TABLE) IT BECOMES APPROXIMATELY 1.7×10^{38}

ARITHMETIC PSEUDO INSTRUCTIONS

AN ADDITIONAL SET OF PSEUDO INSTRUCTIONS ARE INCLUDED IN THE ASSEMBLER TO ALLOW THE PROGRAMMER TO CONVENIENTLY USE THE ARITHMETIC SUBROUTINES DEVELOPED FOR THE COMPUTER.

ALL OF THE ARITHMETIC PSEUDO INSTRUCTIONS HAVE THE FOLLOWING FORM:

```
LABEL     OPCODE      OPERAND

            •

            •
         [ARITH ]
         [PSEUDO]      M

           JSB        [ARITHMETIC ROUTINE]

           DEF         M
```



REFERENCES TO ARITHMETIC PSEUDO INSTRUCTIONS DO NOT HAVE TO BE DECLARED AS EXTERNAL SYMBOLS.

ARITHMETIC PSEUDO INSTRUCTIONS

PSEUDO	FUNCTION	OPERATION
MPY	FIXED POINT MULTIPLICATION	(A) X (m) $\rightarrow$ (B± MSB and ALSB)
DIV	FIXED POINT DIVISION	(B± MSB and ALSB)/(m) $\rightarrow$ A, remainder $\rightarrow$ B
FAD	FLOATING POINT ADDITION	(AB) X (m, m+1) $\rightarrow$ AB
FSB	FLOATING POINT SUBTRACTION	(AB)/(m, m+1) $\rightarrow$ AB
FMP	FLOATING POINT MULTIPLICATION	(m, m+1) + (AB) $\rightarrow$ AB
FDV	FLOATING POINT DIVISION	(AB−(m, m+1) $\rightarrow$ AB
DLD	DOUBLE LOAD	(m) and (m+1) $\rightarrow$ A and B
DST	DOUBLE STORE	(A) and (B) $\rightarrow$ m and m+1

LEGEND:

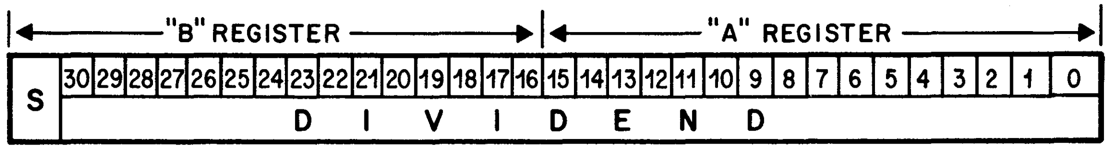
A= Reg. A	m = Operand
B= Reg. B	MSB/LSB = Most/Least Significant Bits
±= Sign	

MPY - FIXED POINT MULTIPLY EXAMPLE

	REGISTER "B"						REGISTER "A"				
	15, 14	13 12,	11 10 9,	8 7 6,	5 4 3,	2 1 0 | 15,	14 13 12,	11 10 9,	8 7 6,	5 4 3,	2 1 0
	S | 30	29—27	26—24	23—21	20—18	17—15	14—12	11—9	8—6	5—3	2—0
	11	10	9	8	7	6	5	4	3	2	1
EXAMPLE 1 MULTIPLY 7x 3											
LDA SEVEN	X	X	X	X	X	0	0	0	0	0	7
MPY THREE	0	0	0	0	0	0	0	0	0	2	5
EXAMPLE 2 MULTIPLY (-1) x 1											
LDA NGONE	X	X	X	X	X	1	7	7	7	7	7
MPY ONE	3	7	7	7	7	7	7	7	7	7	7

FORMAT OF PRODUCT

DIV - FIXED POINT DIVIDE EXAMPLE



EXAMPLE

DIVIDE 15 BY 6

CLB
LDA FIFTH
DIV SIX

11	10	9	8	7	6	5	4	3	2	1
0	0	0	0	0	0	X	X	X	X	X
0	0	0	0	0	0	0	0	0	1	7

RESULT▶

"A" REGISTER

S	QUOTIENT
0	0 0 0 0 2

"B" REGISTER

S	REMAINDER
0	0 0 0 0 3

FLOATING POINT ARITHMETIC

EXAMPLE: CONSIDER THE FOLLOWING ASSEMBLY LANGUAGE STATEMENTS

```
LABEL     OP CODE     OPERAND          REMARKS

  X         DEC       3000.
  Y         DEC       25.
  A         DEC       265.
  B         DEC       100.             (THE DECIMAL POINTS WILL CAUSE
  C         BSS       2                THE ASSEMBLER TO CONVERT THE
  Z         BSS       2                NUMBERS TO 32 BIT FLOATING
                                       POINT REPRESENTATION)

 *          TO CALCULATE  X *Y
            DLD       X
            FMP       Y
            DST       Z                Z = X*Y

 *          TO CALCULATE A/B
            DLD       A
            FDV       B
            DST       C                C = A/B
```

MATH LIBRARY FUNCTIONS

TO PERFORM THE FOLLOWING OPERATIONS:
(FLOATING POINT QUANTITIES) $\longrightarrow$ THE FOLLOWING INSTRUCTIONS CAN BE USED:

Y = ABS (X)	ABSOLUTE VALUE	EXT (NAME)
Y = ATAN (X)	ARCTANGENT	LDA X
Y = ALOG (X)	NATURAL LOG	LDB X + 1
Y = COS (X)	COSINE	JSB (NAME)
Y = EXP (X)	$Y = e^X$	STA Y
Y = SIN (X)	SINE	STB Y + 1
Y = SQRT (X)	SQUARE ROOT	
Y = TAN (X)	TANGENT	
Y = TANH (X)	HYPERBOLIC TANGENT	

NOTES

- LIBRARY FUNCTIONS MUST BE DEFINED AS EXTERNAL (EXT) SYMBOLS.
- THEY MAY ONLY BE REFERENCED BY RELOCATABLE PROGRAMS.

EXAMPLE: FIND THE SQUARE ROOT OF QUANTITY "X" AND STORE TO LOCATION "Y"

METHOD 1

```
EXT SQRT
LDA X
LDB X+1
JSB SQRT
STA Y
STB Y+1
```

OR

METHOD 2

```
EXT SQRT
DLD X
JSB SQRT
DST Y
```

USING A LIBRARY FUNCTION

PROBLEM STATEMENT

FIND THE HYPOTENUSE OF A RIGHT TRIANGLE

WHERE: HYPOT = SQRT (X^2+Y^2)

PROBLEM SOLUTION

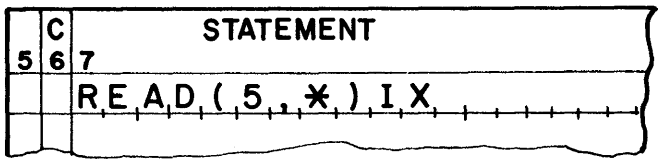
```
LABEL      OP CODE      OPERAND      REMARKS

           NAM          HYPOT
           EXT          SQRT
X          BSS          2
Y          BSS          2
HYPOT      BSS          2
TEMP       BSS          2
            •
            •
START      NOP
           DLD          X
           FMP          X            X² = X*X
           DST          TEMP
           DLD          Y
           FMP          Y            Y² = Y*Y
           FAD          TEMP         X² + Y²
           JSB          SQRT
           DST          HYPOT        HYPOT = √(X² + Y²)
            •
            •
           END          START
```

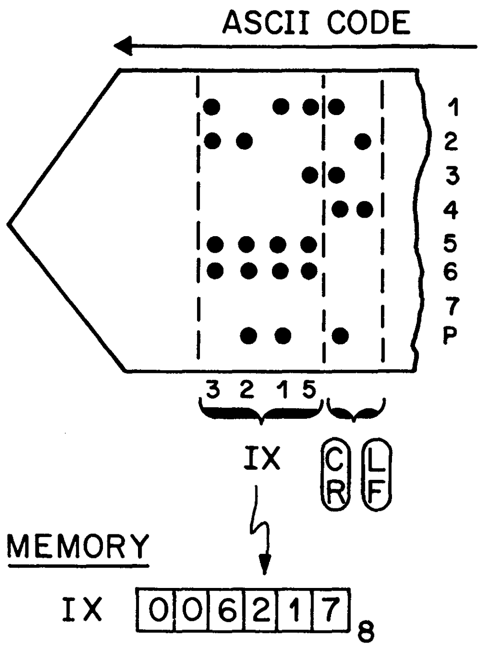
DATA CONVERSION

EXTERNAL(INPUT)		INTERNAL(COMPUTER)		EXTERNAL(OUTPUT)
ASCII	⟶	BINARY	⟶	ASCII

THE DATA CONVERSION PROCESS IS NOT APPARENT TO THE FORTRAN PROGRAMMER.

FOR EXAMPLE:





AS A RESULT OF THE READ STATEMENT MEMORY LOCATION "IX" WOULD CONTAIN 006217_8

QUESTIONS!

1. HOW WAS THE CONVERSION PERFORMED?
2. HOW CAN A PROGRAMMER PERFORM THE EQUIVALENT OPERATION USING ASSEMBLY LANGUAGE?

THE FORMATTER

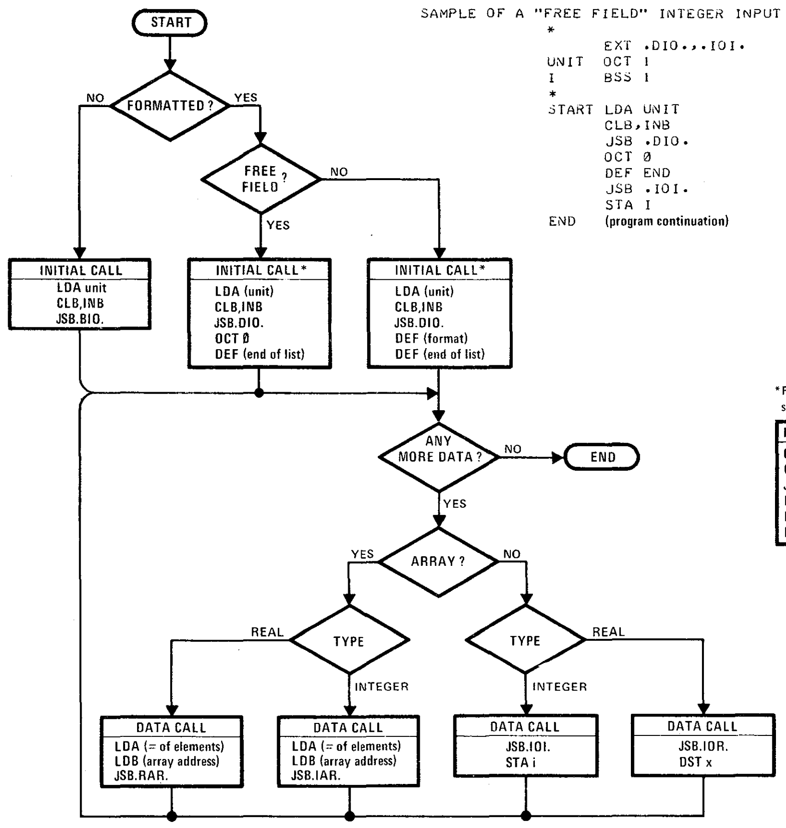
THE FORMATTER IS A LIBRARY ROUTINE DESIGNED PRIMARILY TO PROVIDE DATA CONVERSION AND INPUT-OUTPUT CAPABILITY FOR FORTRAN PROGRAMS. ASSEMBLY LANGUAGE PROGRAMS MAY USE THE FORMATTER BY CODING THE CORRECT CALLING SEQUENCE AND PROVIDING THE PROPER PARAMETERS. THE FORMATTER HAS 7 ENTRY POINTS.

THE FORMATTER CONTAINS 7 SUB-PROGRAMS, EACH DESIGNED TO PERFORM A SPECIFIC PART OF THE TOTAL INPUT-OUTPUT OPERATION. FOR EXAMPLE:

FORMATTER
.DIO. (DECIMAL INPUT/OUTPUT)
.BIO. (BINARY INPUT/OUTPUT)
.IOR. (INPUT/OUTPUT REAL)
.IOI. (INPUT/OUTPUT INTEGER)
.RAR. (REAL ARRAY)
.IAR. (INTEGER ARRAY)
.DTA. (TERMINATOR)

FORMATTER

INPUT CALLING SEQUENCE SELECTOR OUTPUT



```
SAMPLE OF A "FREE FIELD" INTEGER INPUT CALL
*
      EXT .DIO.,.IOI.
UNIT  OCT 1
I     BSS 1
*
START LDA UNIT
      CLB,INB
      JSB .DIO.
      OCT Ø
      DEF END
      JSB .IOI.
      STA I
END   (program continuation)
```

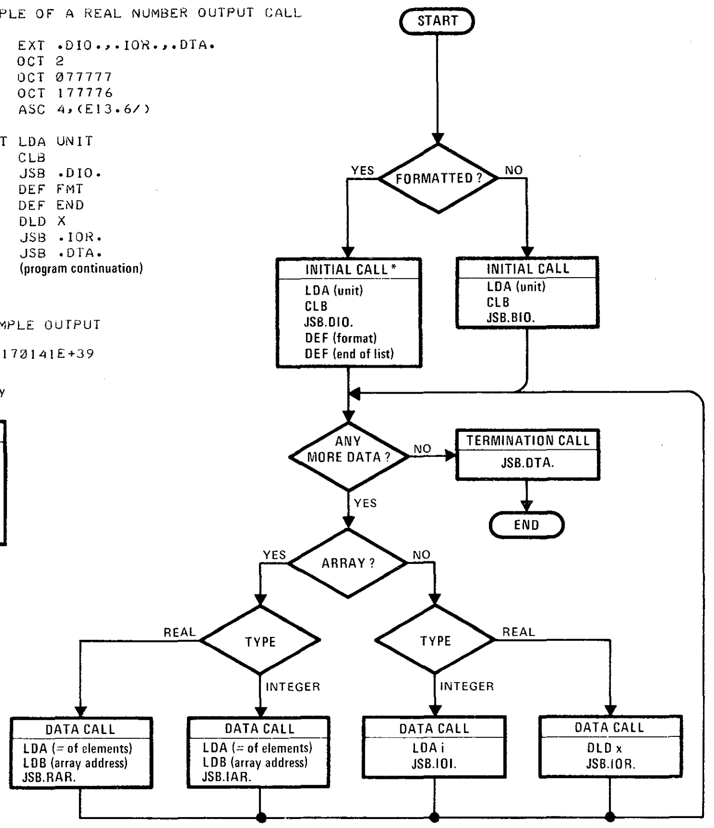
```
*SAMPLE OF A REAL NUMBER OUTPUT CALL
*
      EXT .DIO.,.IOR.,.DTA.
UNIT  OCT 2
X     OCT Ø77777
      OCT 177776
FMT   ASC 4,(E13.6/)
*
START LDA UNIT
      CLB
      JSB .DIO.
      DEF FMT
      DEF END
      DLD X
      JSB .IOR.
      JSB .DTA.
END   (program continuation)
```

```
SAMPLE OUTPUT

.17Ø141E+39
```

*For internal conversion only substitute the following –

INITIAL CALL
CLA (unit = Ø)
CLB,INB (in) or CLB (out)
JSB.DIO.
DEF (buffer)
DEF (format)
DEF (end of list)

in - ASCII to binary
out - binary to ASCII



USING THE FORMATTER

```
                    EXT .IOC.,.DIO.,.IOI.,.RAR.,.IAR.,.BIO.,.DTA.
*DATA STORAGE AND CONSTANTS
*
              BUFFR DEC 1,2,3,4,5,6,7,8,9,10
              N     DEC 10
              UNIT2 OCT 2
              UNIT4 OCT 4
              IARRY DEF BUFFR
              FMT2  ASC 16,("PRINT ELEMENTS 3,5,AND 7"/3I5)

*CALLING SEQUENCE TO PUNCH AN INTEGER ARRAY IN BINARY FORM
*
                    LDA UNIT4     LOAD"A"WITH UNIT #
                    CLB           0 TO "B" FOR OUTPUT
                    JSB .BIO.     INITIAL CALL (BINARY)
                    LDA N         # OF ELEMENTS
                    LDB IARRY     ARRAY ADDRESS
                    JSB .IAR.     DATA CALL
                    JSB .DTA.     NO MORE ITEMS ON DATA LIST
                         :
*CALLING SEQUENCE TO PRINT THE 3RD, 5TH, AND 7TH ELEMENTS OF BUFFR
*
                    LDA UNIT2     LOAD "A" WITH UNIT#
                    CLB           0 TO "B" FOR OUTPUT
                    JSB .DIO.     INITIAL CALL (FORMATTED)
                    DEF FMT2      ADDRESS OF ASCII FORMAT STRING
                    DEF EOL3      END OF LIST ADDRESS
                    LDA BUFFR+2   GET 3RD ELEMENT
                    JSB .IOI.     DATA CALL
                    LDA BUFFR+4   GET 5TH ELEMENT
                    JSB .IOI.     DATA CALL
                    LDA BUFFR+6   GET 7TH ELEMENT
                    JSB .IOI.     DATA CALL
                    JSB .DTA.     NO MORE ITEMS ON DATA LIST
              EOL3  (Program Continuation)
```

HP Basic Control System, I.O.C. section

X

LESSON X
HP BASIC CONTROL SYSTEM, IOC SECTION

LESSON X OBJECTIVES

THE OBJECTIVES OF LESSON X ARE:

1 - TO INTRODUCE THE STUDENT TO THE HEWLETT-PACKARD BASIC CONTROL SYSTEM.

2 - TO INSTRUCT THE STUDENT IN THE USE OF THE INPUT/OUTPUT CONTROL (IOC) AND I/O EQUIPMENT DRIVER SUBROUTINES.

INTRODUCTION TO THE BASIC CONTROL SYSTEM

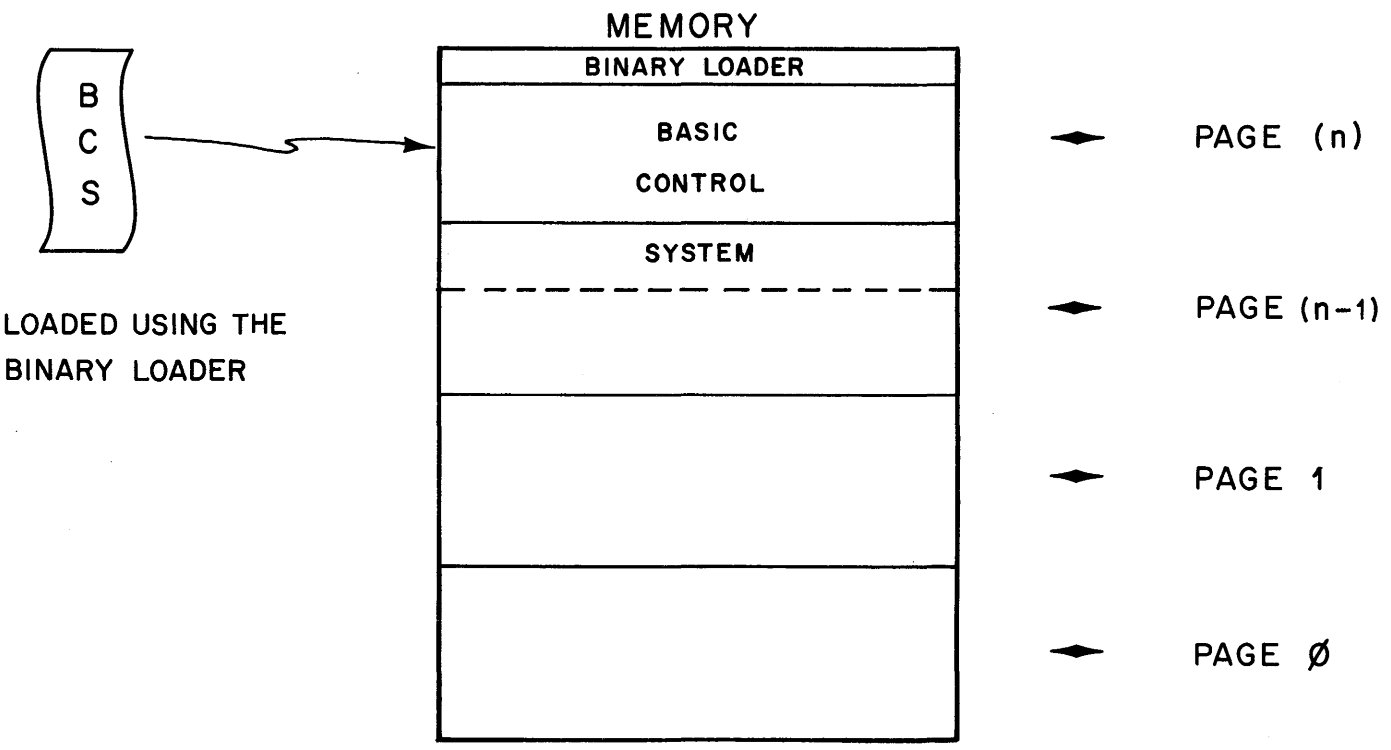
THE BASIC CONTROL SYSTEM PROVIDES 2 MAIN FUNCTIONS TO THE COMPUTER SYSTEM.

1. Provides a flexible, systematic structure to handle input/output requests from system and user programs.

2. Provides the loading and linking capability required for relocatable object programs produced by FORTRAN and the ASSEMBLER.

BASIC CONTROL SYSTEM

THE BASIC CONTROL SYSTEM IS AN ABSOLUTE PROGRAM ON PAPER TAPE



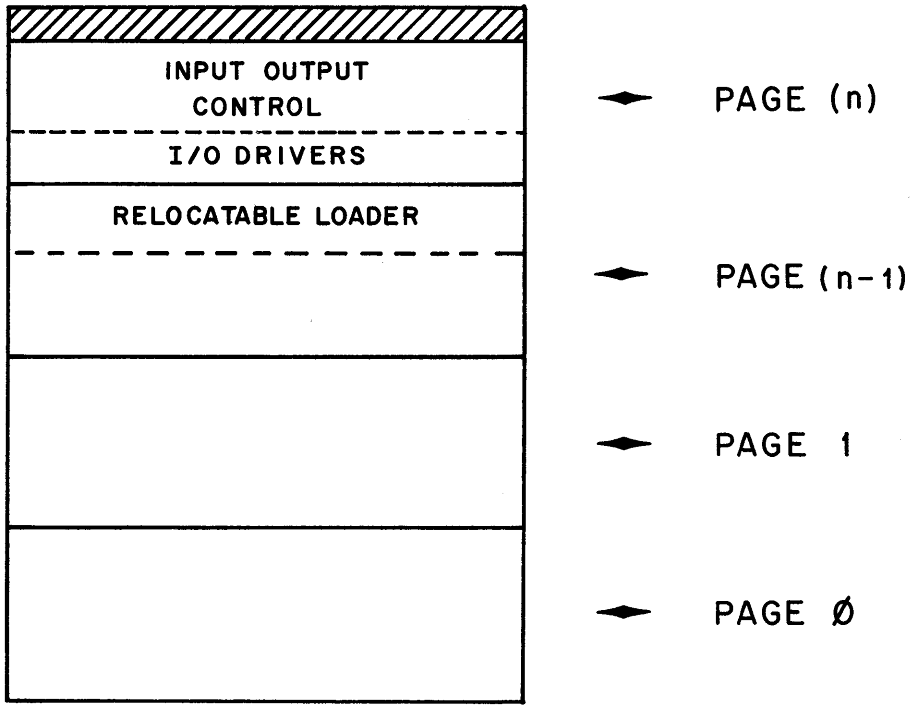
THE BASIC CONTROL SYSTEM ALWAYS RESIDES IN UPPER MEMORY

THE BASIC CONTROL SYSTEM IS MADE UP OF THREE INTEGRAL PARTS ——————

1 *INPUT OUTPUT CONTROL*

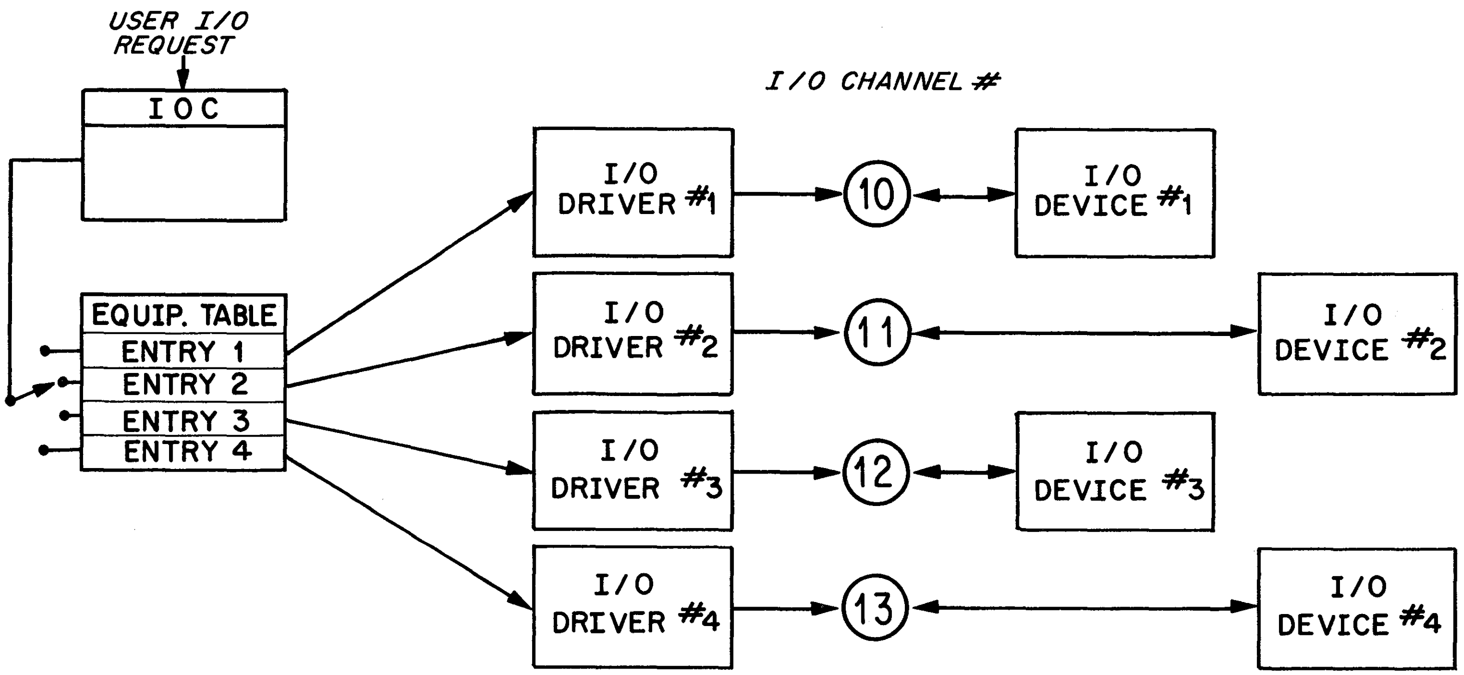
2 *I/O DRIVERS*

3 *RELOCATABLE LOADER*



BCS MODULES

SIMPLIFIED IOC BLOCK DIAGRAM



① The user requests an I/O operation using a <u>logical</u> unit number.

② IOC finds the logical unit entry in the equipment table.

③ The equipment table entry contains the address of the driver.

UNIT REFERENCE NUMBERS

IN ORDER TO ALLOW NEW INPUT/OUTPUT HARDWARE TO BE INTEGRATED INTO THE COMPUTER SYSTEM, USER REQUESTS TO IOC NEVER REFER TO THE PHYSICAL I/O DEVICE CHANNEL NUMBER. EACH I/O DEVICE IS ASSIGNED A LOGICAL NUMBER CALLED THE UNIT REFERENCE NUMBER.

UNIT REFERENCE NUMBERS REFER TO ONE OF TWO TABLES:

1 - STANDARD UNIT TABLE
2 - EQUIPMENT TABLE

EQUIPMENT TABLE

UNIT REFERENCE NUMBER$_8$	I/O CHANNEL(S)$_8$	DEVICE
7	14	TELEPRINTER
10	15	H. S. TAPE READER
11	16	H. S. TAPE PUNCH
•	•	•
•	•	•
•	•	•

EACH I/O DEVICE MAKES ONE ENTRY IN THE EQUIPMENT TABLE, HOWEVER THE FIRST ENTRY IS ASSIGNED UNIT REFERENCE NUMBER 7.

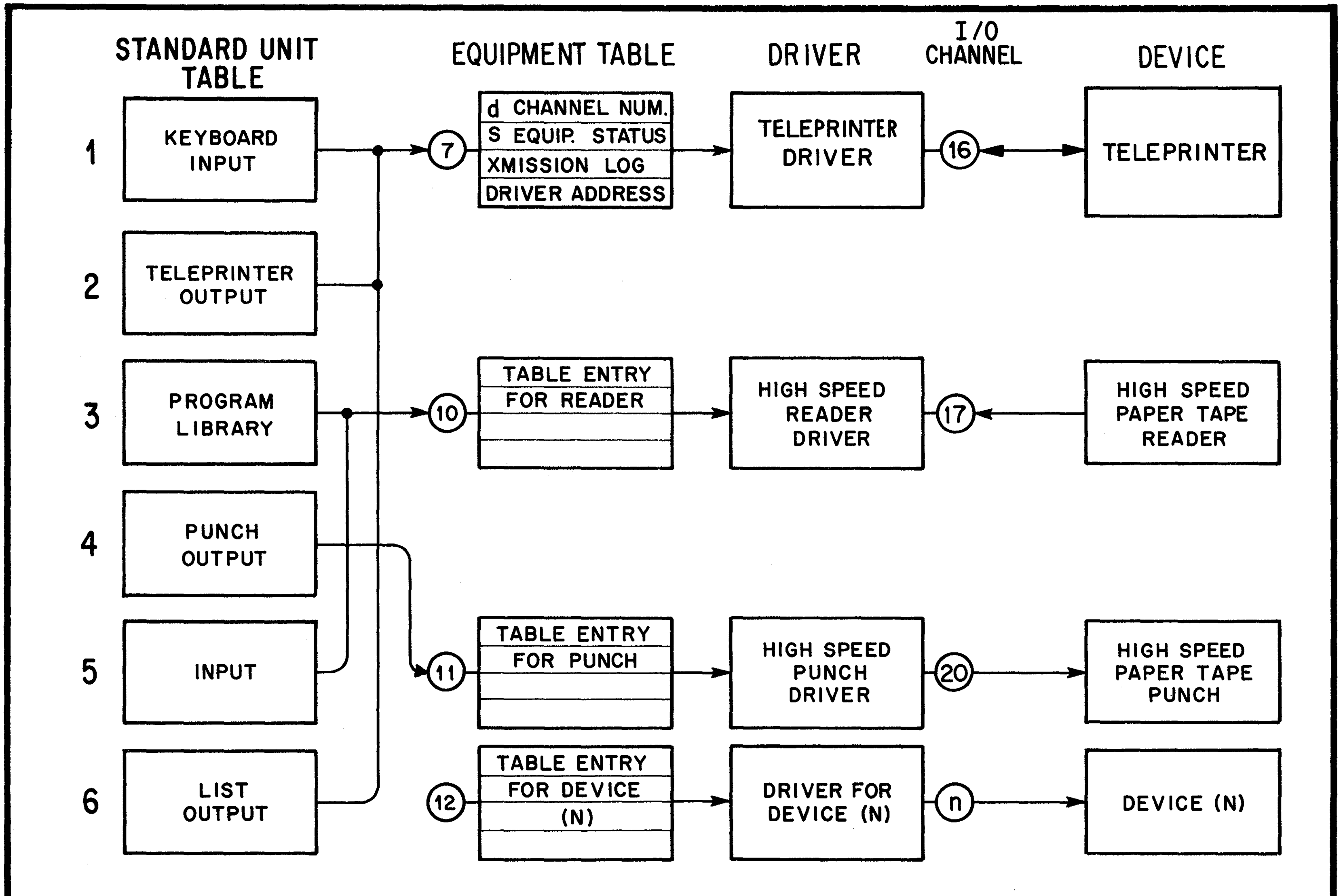
EXAMPLE OF AN INITIAL EQUIPMENT TABLE

EQUIPMENT TABLE

UNIT REFERENCE NUMBER$_8$	I/O CHANNEL (S)$_8$	DEVICE
7	16	TELEPRINTER
10	17	H.S. TAPE READER
11	20	H.S. TAPE PUNCH
12	14/15	MAGNETIC TAPE
•	•	•
•	•	•

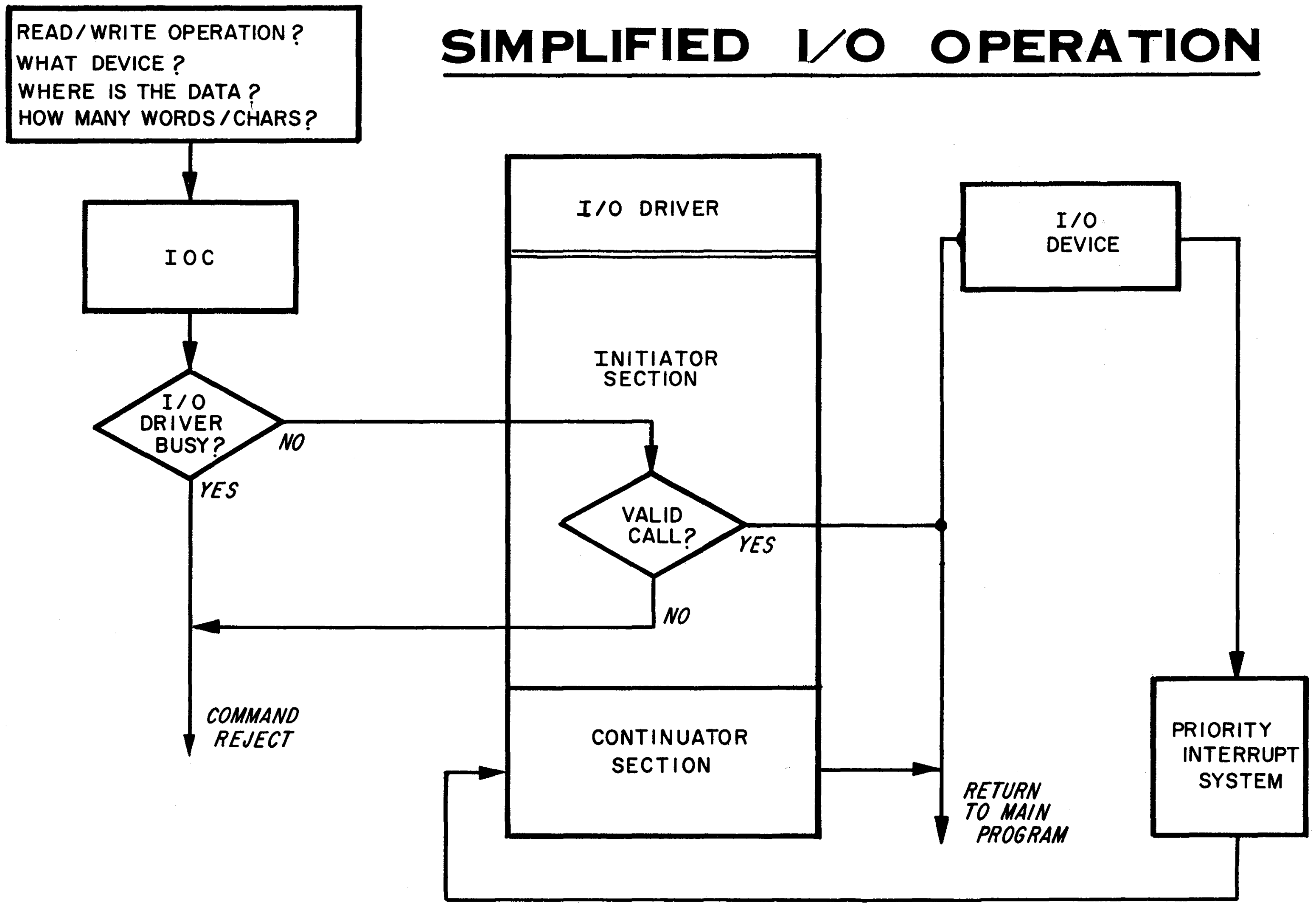
ALTHOUGH NEW CHANNEL ASSIGNMENTS FOR THE DEVICES ARE SHOWN. REFERENCES TO UNIT 7 WILL STILL BE ROUTED TO THE TELEPRINTER. THE EQUIPMENT TABLE PROVIDES THE LOGICAL/PHYSICAL FLEXIBILITY REQUIRED TO CHANGE OR UPGRADE A COMPUTER INSTALLATION WITHOUT CHANGE TO EXISTING PROGRAMS.

EXAMPLE OF UPGRADING THE SYSTEM



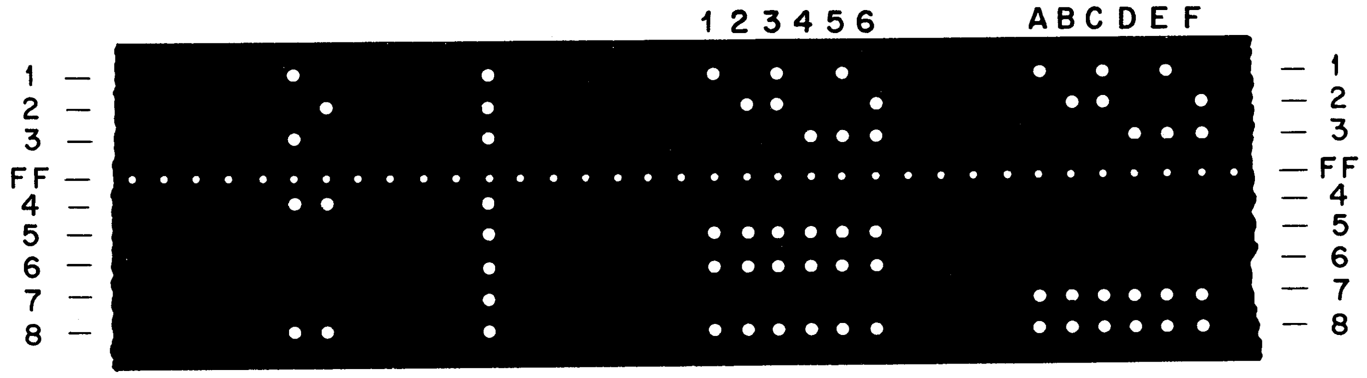
LOGICAL TO PHYSICAL TRANSLATION PROCESS

SIMPLIFIED I/O OPERATION



```
                                    EXT    .IOC.

                                    JSB    .IOC.

DRIVER OR DEVICE BUSY          ⎫    OCT   (<FUNCTION><SUBFUNCTION><UNIT REF.>)
ILLEGAL CALL OR DMA            ⎬
CHANNEL NOT AVAILABLE....      ⎪
IOC WILL RETURN HERE           ⎭    JMP   (REJECT ADDRESS)

                                    DEF   (BUFFER ADDRESS)

                                    DEC   (BUFFER LENGTH OR COUNT)

IF THE REQUEST IS ACCEPTED
IOC WILL RETURN TO THE              PROGRAM CONTINUATION
LOCATION FOLLOWING THE
COUNT PARAMETER
```

INPUT OUTPUT REQUESTS



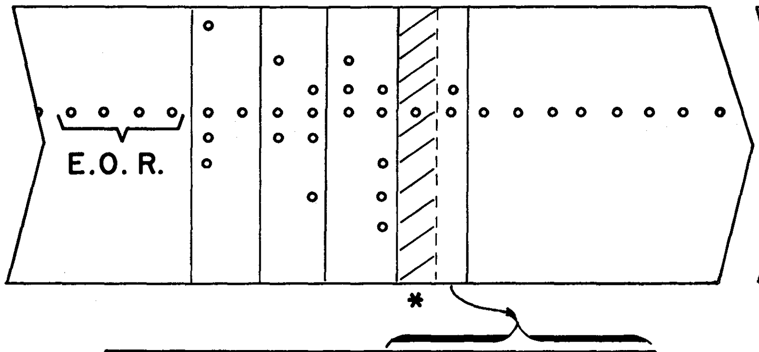
1) LEVELS 1 - 7 CONSTITUTE THE ASCII CODE

2) THE DRIVERS WITHIN BCS REMOVE THE 8th LEVEL ON INPUT AND PUNCH THE 8th LEVEL ON OUTPUT WHEN READING/WRITING

3) CARRIAGE RETURN, LINE FEED IS THE ASCII END OF RECORD MARK

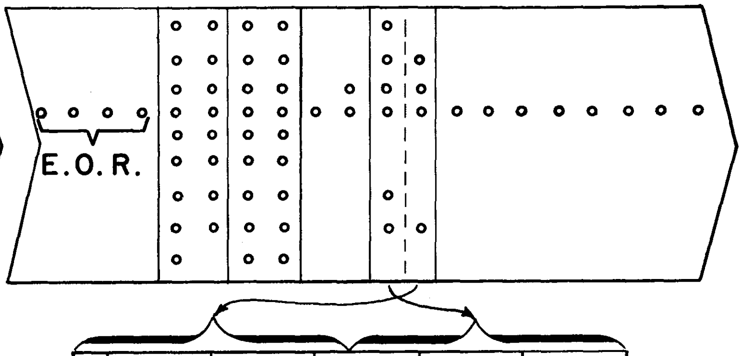
4) A 'RUB OUT' CHARACTER FOLLOWED BY C.R., L.F. WILL CAUSE THE DRIVER TO IGNORE THE PRECEDING RECORD

(TELETYPE) 8 LEVEL TAPE



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

8 BIT WORD COUNT FORMAT



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	0	1	1	0	0	1	1	1

16 BIT BINARY WORD FORMAT

BITS

Ø	8
1	9
2	10
3	11
4	12
5	13
6	14
7	15

A VARIABLE LENGTH BINARY RECORD (SHOWN WITH 4 WORDS)

A FIXED LENGTH BINARY RECORD (SHOWN WITH 4 WORDS)

1. **Feed frames prior to the first character are ignored by the driver. Therefore the 1st frame of a binary record must be non-zero.**

2. **Four feed frames separate binary records.**

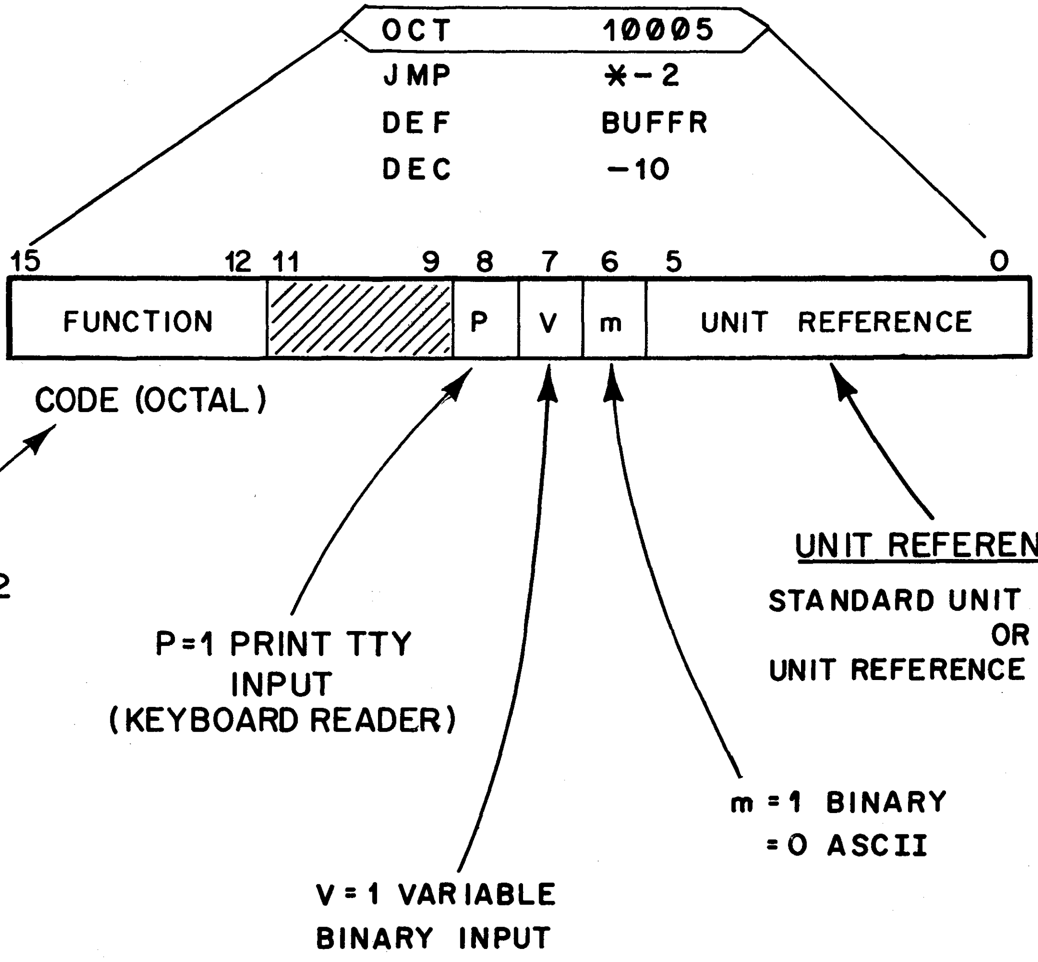
3. **To output a record in variable binary form, the number of words in the data block must be in the range $1 \leq n \leq 255_{10}$. The second frame of the word count * is ignored on input.**

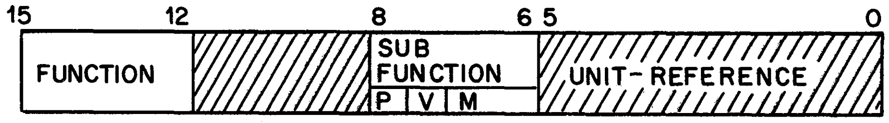
4. **Variable binary input uses the word count, or the specified buffer size to terminate transmission, whichever value is smaller.**

8 LEVEL PAPER TAPE (BINARY)

IOC CALL (PARAMETER 1)

```
LABEL    OPCODE    OPERAND
         EXT       .IOC.
         JSB       .IOC.
         OCT       10005
         JMP       *-2
         DEF       BUFFR
         DEC       -10
```


15
12
11
9
8
7
6
5
0
FUNCTION
P
V
m
UNIT REFERENCE
CODE (OCTAL)
READ 01
WRITE 02
P=1 PRINT TTY INPUT (KEYBOARD READER)
V = 1 VARIABLE BINARY INPUT
m = 1 BINARY
= 0 ASCII
UNIT REFERENCE VALUES
STANDARD UNIT NUMBERS 1-6
OR
UNIT REFERENCE NUMBERS 7-n



OPERATION				
READ ASCII RECORD	0	1	0	0
READ ASCII RECORD AND PRINT	0	1	0	4
READ BINARY RECORD	0	1	0	1
READ VARIABLE LENGTH BINARY RECORD	0	1	0	3
WRITE ASCII RECORD	0	2	0	0
WRITE BINARY RECORD	0	2	0	1

ALLOWABLE COMBINATIONS OF READ/WRITE FUNCTIONS

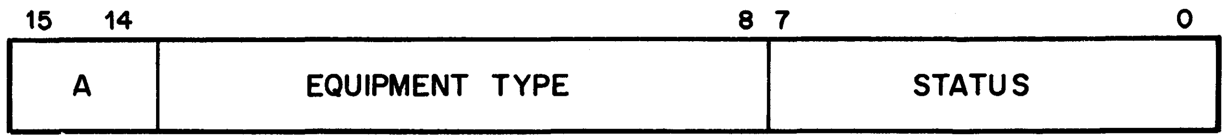
COMMAND REJECT

AN I/O REQUEST THAT IS REJECTED WILL CAUSE IOC TO RETURN CONTROL HERE

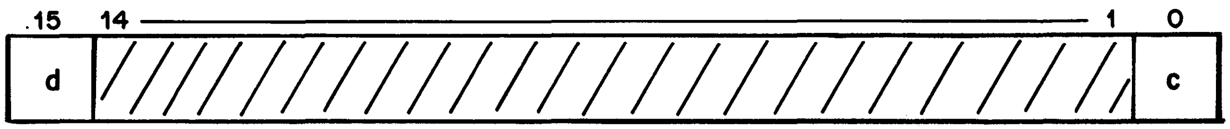
THE CAUSE OF THE REJECT WILL BE RETURNED IN THE B REGISTER.

```
LABEL  OPCODE  OPERAND
       JSB     .IOC.
       OCT     XXXXX
       JMP     *-2
       DEF     ADDR
       DEC     COUNT
```

A-REGISTER



B-REGISTER



d = 1 DEVICE OR DRIVER BUSY
c = 1 DMA CHANNEL NOT AVAILABLE
d = c = Ø ILLEGAL FUNCTION OR SUBFUNCTION

```
LABEL        OP CODE      OPERAND      REMARKS

             EXT          .IOC.
              |             |
              |             |
              |             |
             JSB          .IOC.
             OCT          10005        READ ASCII, STD INPUT
             JMP          *-2
             DEF          TABL         ADDRESS OF BUFFER
             DEC          10           NUMBER OF WORDS
              |
              |
              |
              |
              |
TABL         BSS          10
              |
              |
              |
             END
```

1 - THE ADDRESS OF THE FIRST WORD OF THE BUFFER IS DEFINED USING THE DEF PSEUDO.

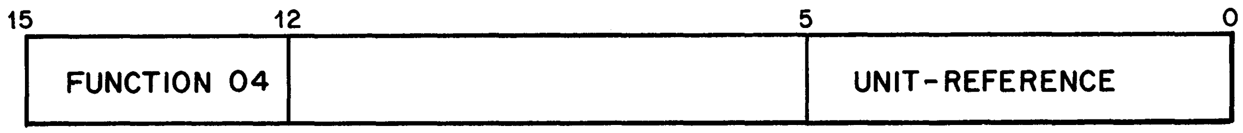
2 - THE NUMBER OF WORDS/CHARACTERS TO BE TRANSFERRED IS DEFINED BY:
WORDS = A POSITIVE VALUE
CHARACTERS = A NEGATIVE VALUE

EXAMPLE CALL TO IOC

```
LABEL       OP CODE       OPERAND       REMARKS

              .
              .
              .
            EXT           .IOC.         .IOC. EXTERNAL
LINE        BSS           36            36 WORD PROGRAM BUFFER
            COM           BKB(100)      100 WORD COMMON BUFFER
              .
              .
              .
READ1       JSB           .IOC.
            OCT           10005         READ ASCII
            JMP           *-2
            DEF           LINE          PROGRAM BUFFER
            DEC           -72           72 ASCII CHARS
              .
              .
              .
WRITE 1     JSB           .IOC.         WRITE BINARY
            OCT           20111
            JMP           *-2
            DEF           BKB           COMMON BUFFER
            DEC           100           100 BINARY WORDS
              .
              .
              .
```

EXAMPLE OF IOC CALLING SEQUENCES

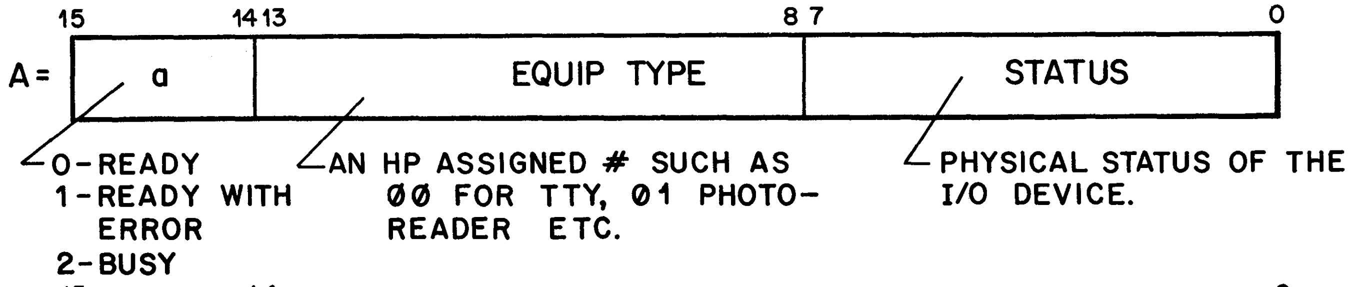
STATUS REQUEST (DEVICE)

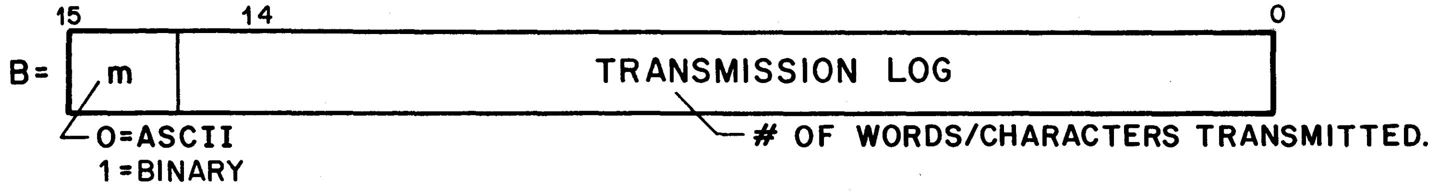


(CALLING SEQUENCE)

```
JSB     .IOC.
OCT     <FUNCTION> <UNIT-REF>
```

IOC **WILL RETURN TO THE MAIN PROGRAM WITH WORD 2 OF THE EQT IN THE A REGISTER, AND WORD 3 OF THE EQT IN THE 'B' REGISTER.**


15
14 13
8 7
0
A=
a
EQUIP TYPE
STATUS
0-READY
1-READY WITH ERROR
2-BUSY
AN HP ASSIGNED # SUCH AS ØØ FOR TTY, Ø1 PHOTO-READER ETC.
PHYSICAL STATUS OF THE I/O DEVICE.

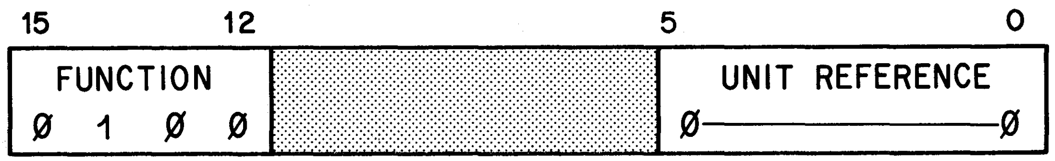


```
LABEL      OP CODE      OPERAND     REMARKS

             COMMAND INITIATION

READ       JSB          .IOC.
           OCT          1ØØ15       READ ASCII
           JMP          REJECT
           DEF          INBUF       BUFFER ADDRESS
           DEC          -2Ø         2Ø CHARACTERS
           JMP          STAT
            ⋮
REJECT     SSB                      IS DRIVER BUSY?
           JMP          READ        YES, RE-INITIATE COMMAND
           JSB          ABORT       NO, GO TO ERROR ROUTINE
INBUF      BSS          1Ø
```

```
               STATUS CHECKING

STAT       JSB          .IOC.
           OCT          4ØØ15       REQUEST STATUS
           SSA                      IS DRIVER BUSY?
           JMP          STAT        YES, LOOP UNTIL FREE
           RAL                      NO, ROTATE & TEST BIT 14
           SSA, RSS                 ANY ERROR ?
           JMP          PROCS       NO, CONTINUE PROCESSING.
           ALF, ALF                 YES, POSITION BIT 5
           RAL                      FOR STATUS TEST
           SSA                      EOT CONDITION?
           JMP          ENDPR       YES, GO TO EOT ROUTINE
           JSB          ABORT       NO, GO TO ERROR ROUTINE
```

CODING EXAMPLE TO CHECK STATUS CONDITIONS

STATUS REQUEST (SYSTEM)



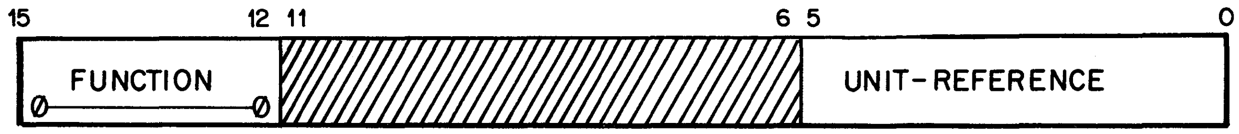
CALLING SEQUENCE

```
JSB .IOC.
OCT 4ØØØØ
(RETURN)
```

IOC WILL RETURN TO THE MAIN PROGRAM WITH REGISTER "A":

POSITIVE - No system devices are busy
NEGATIVE - A system device is busy

THE SECOND WORD CONSISTS OF THE FOLLOWING:



THE ONLY OTHER PARAMETER REQUIRED IS THE UNIT-REFERENCE NUMBER.

THE CLEAR REQUEST TERMINATES A PREVIOUSLY ISSUED INPUT OR OUTPUT OPERATION BEFORE ALL DATA IS TRANSMITTED. IT HAS THE FOLLOWING FORM:

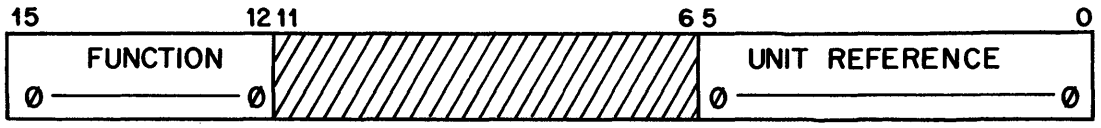
```
(CALLING SEQUENCE)
  JSB     .IOC.
  OCT      <FUNCTION> <UNIT-REFERENCE>
```

CLEAR REQUEST (DEVICE)

```
LABEL     OP CODE     OPERAND     REMARKS

            •
            •
            •
READM       JSB       .IOC.       OPEN TTY INPUT CHANNEL
            OCT       10401       READ FROM KEYBOARD
            JMP       *-2         BUSY, TRY AGAIN
            DEF       MSG         DATA BUFFER
            DEC       -72         72 CHARACTERS MAX
            JSB       TIMER       RTN TO TIME OPERATOR RESPONSE
            •
            •
            •
CLRRD       JSB       .IOC.       CLEAR REQUEST
            OCT       1           ON UNIT 1
            •
            •
            •
```

EXAMPLE OF A CLEAR REQUEST

The second word consists of the following:



CALLING SEQUENCE

```
JSB .IOC.
OCT Ø
(RETURN)
```

THE SYSTEM CLEAR REQUEST CAUSES IOC TO CLEAR ALL DEVICES AND DRIVERS DEFINED BY THE EQUIPMENT TABLE. THIS MAKES ALL DEVICES AVAILABLE FOR INITIATING AN OPERATION.

CLEAR REQUEST (SYSTEM)

INTERNAL CONVERSION EXAMPLE

```
*CALLING SEQUENCE TO INTERNALLY CONVERT A REAL ARRAY (BINARY)
*TO ASCII AND OUTPUT WITH A CALL DIRECTLY TO .IOC.
*
      EXT .IOC.,.DIO.,.RAR.,.DTA.
      ENT START

START NOP
      CLA             UNIT =0 MEANS INTERNAL CONVERSION
      CLB             0 TO "B" FOR OUTPUT
      JSB .DIO.       INITIAL CALL (FORMATTED)
      DEF ABUFR       ADDRESS OF BUFFER FOR ASCII DATA
      DEF FMT4        ADDRESS OF ASCII FORMAT STRING
      DEF EOL5        END OF LIST ADDRESS
      LDA N           #OF ELEMENTS
      LDB RARRY       ARRAY ADDRESS
      JSB .RAR.       DATA CALL
      JSB .DTA.       NO MORE ITEMS ON DATA LIST
EOL5  LDA CNT         INITIALIZE COUNTER
      STA CNTR
      LDA ADDRS       PICK UP BUFFER ADDRESS
      STA POINT       INITIALIZE ADDRESS
LOOP  JSB .IOC.       CALL .IOC.
      OCT 20002       OUTPUT ON UNIT #2
      JMP *-2         REJECT
POINT OCT 0           BUFFER ADDRESS
      DEC -6          6 ASCII CHARACTERS
      JSB .IOC.       STATUS CHECK
      OCT 40002       UNIT #2
      SSA             BUSY?
      JMP *-3         YES, CHECK STATUS AGAIN
      LDA POINT       NO, MODIFY BUFFER
      ADA P6          ADDRESS TO OUTPUT
      STA POINT       NEXT ELEMENT
      ISZ CNTR        IS CNTR 0?
      JMP LOOP        NO, OUTPUT NEXT ELEMENT
      (Program Continuation)

*DATA STORAGE AND CONSTANTS

BUFFX DEC 1.,2.,....,9.,10.
N     DEC 10
CNT   DEC -10
CNTR  OCT 0
P6    OCT 3
FMT4  ASC 3,(F6.2)
ABUFR BSS 30
ADDRS DEF ABUFR
RARRY DEF BUFFX
```

HP relocating loader, configuration routines XI

LESSON XI
HP RELOCATING LOADER, CONFIGURATION ROUTINES

LESSON XI OBJECTIVES

THE PRIMARY OBJECTIVES OF LESSON XI ARE:

1. TO DISCUSS THE OPERATION OF THE HP RELOCATING LOADER IN MORE DETAIL AND DESCRIBE ADDITIONAL LOADER FEATURES.

2. TO TEACH THE STUDENT HOW TO USE THE "CONFIGURATION" ROUTINES—

   PREPARE CONTROL SYSTEM.
   SYSTEM INPUT/OUTPUT DUMP.

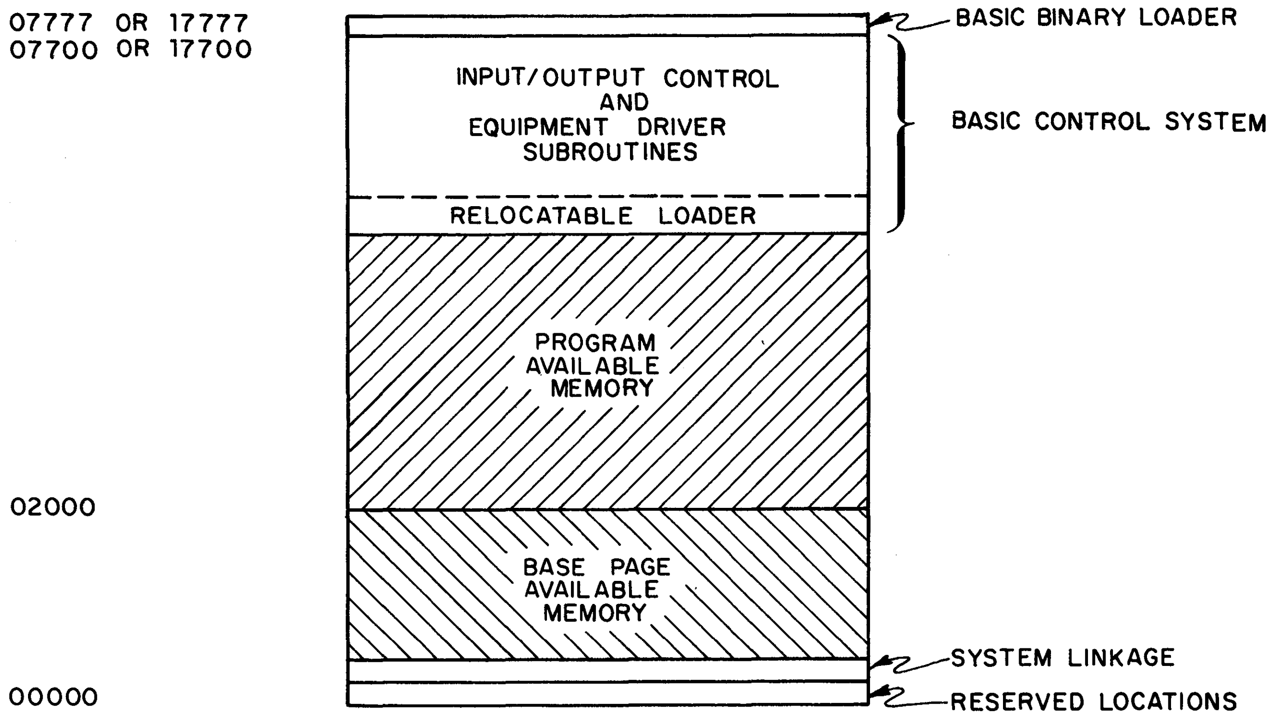
THE RELOCATABLE LOADER:

- LOADS RELOCATABLE OBJECT PROGRAMS.
- ESTABLISHES COMMON STORAGE BOUNDARIES.
- PROVIDES LINKAGES WHEN THE OBJECT PROGRAM IS LOADED ACROSS PAGE BOUNDARIES.
- WILL PUNCH AN ABSOLUTE BINARY TAPE OF THE OBJECT PROGRAM. (OPTION)
- PROVIDES A MEMORY LISTING OF PROGRAM BOUNDARIES AND THE ABSOLUTE ADDRESS OF ALL 'ENT' POINTS DECLARED IN THE SOURCE PROGRAM.
- PROVIDES A 'LOAD AND GO' FEATURE OR THE OPTION OF MANUAL ENTRY OF THE STARTING ADDRESS.

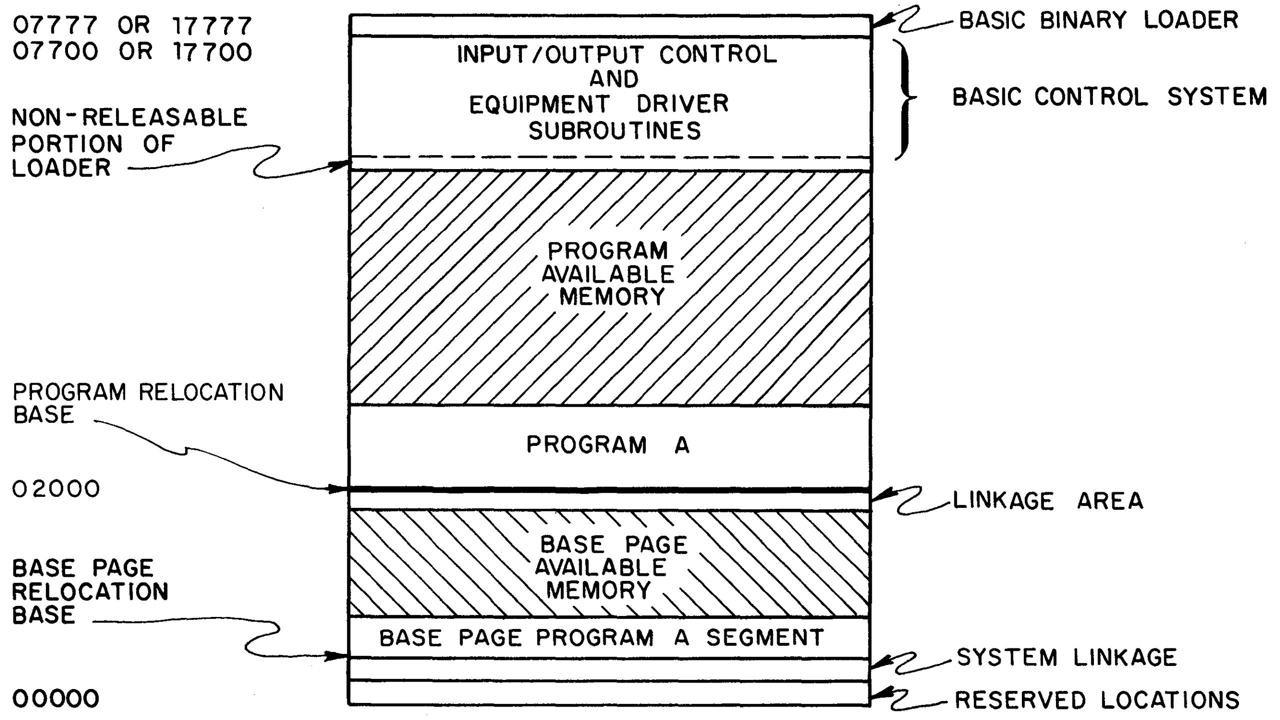
	LABEL	OP CODE	OPERAND
		⋮	
	1776	DEF	B
PAGE 0	1777	DEF	A
PAGE 1	ABSOLUTE START OF PROGRAM		
		⋮	
		ISZ	~~A~~ 1777,I
		LDA	~~A~~ 1777,I
PAGE 1		ADA	~~B~~ 1776,I
PAGE 2		ADA	C
		JMP	X
	A	BSS	1
	B	BSS	1
	C	BSS	1
	X	LDA	A
		HLT	

IN THE EXAMPLE, SYMBOLIC TERMS ARE USED FOR SIMPLICITY AND TO DESCRIBE THE 'EFFECT' OF THE LOADERS ACTION.

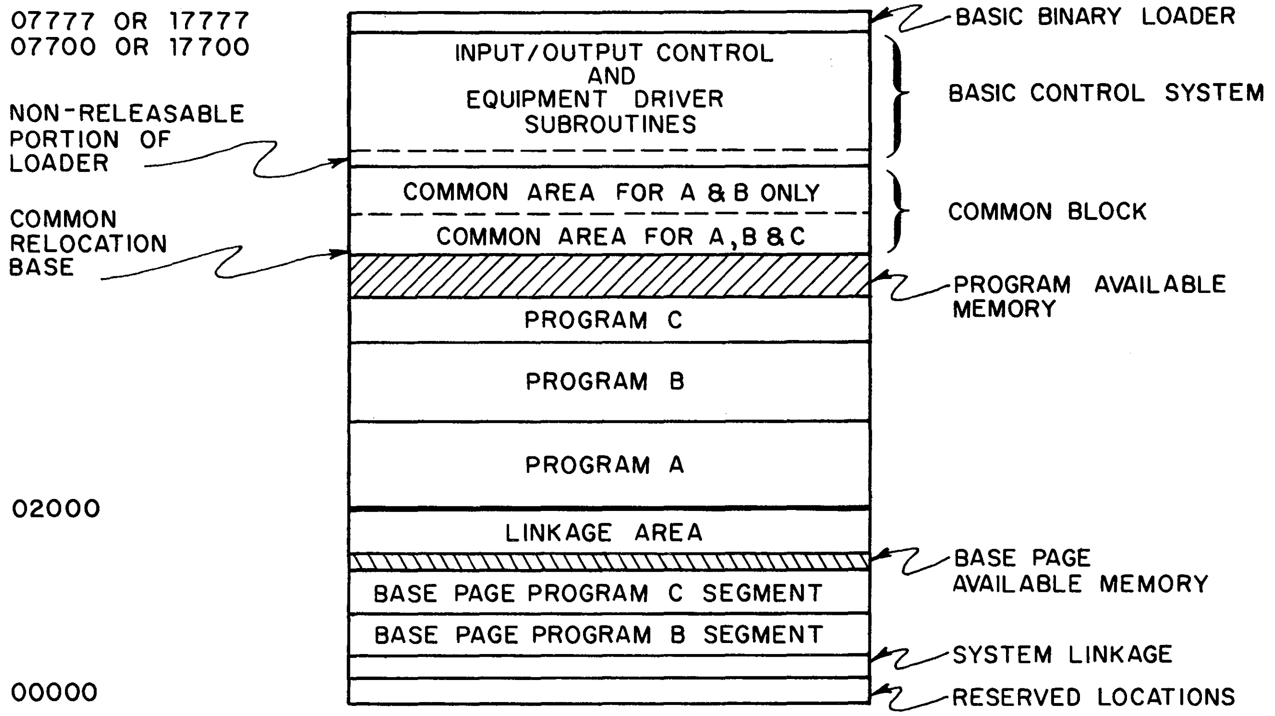
LOADER PROVIDED LINKAGES


07777 OR 17777
07700 OR 17700
02000
00000
BASIC BINARY LOADER
INPUT/OUTPUT CONTROL AND EQUIPMENT DRIVER SUBROUTINES
BASIC CONTROL SYSTEM
RELOCATABLE LOADER
PROGRAM AVAILABLE MEMORY
BASE PAGE AVAILABLE MEMORY
SYSTEM LINKAGE
RESERVED LOCATIONS

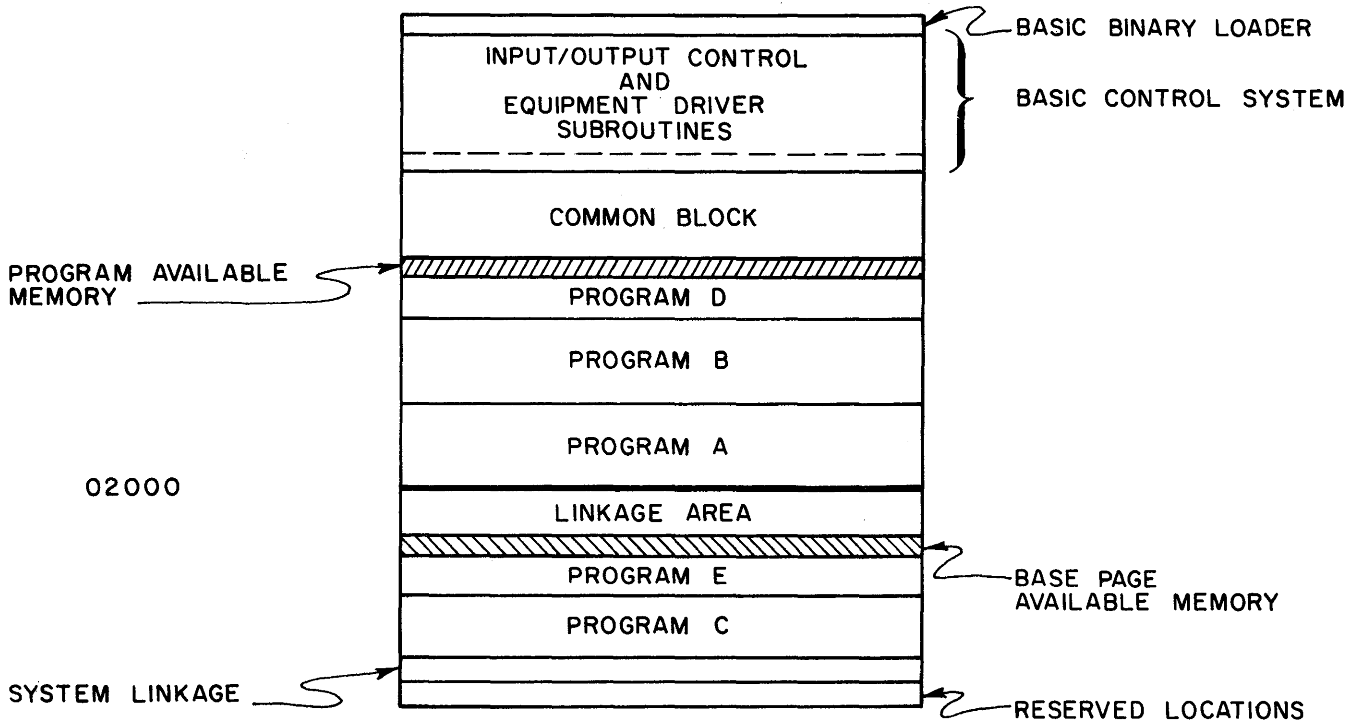
MEMORY MAP 1


07777 OR 17777
07700 OR 17700
NON-RELEASABLE PORTION OF LOADER
PROGRAM RELOCATION BASE
02000
BASE PAGE RELOCATION BASE
00000
INPUT/OUTPUT CONTROL AND EQUIPMENT DRIVER SUBROUTINES
PROGRAM AVAILABLE MEMORY
PROGRAM A
BASE PAGE AVAILABLE MEMORY
BASE PAGE PROGRAM A SEGMENT
BASIC BINARY LOADER
BASIC CONTROL SYSTEM
LINKAGE AREA
SYSTEM LINKAGE
RESERVED LOCATIONS

MEMORY MAP 2


07777 OR 17777
07700 OR 17700
NON-RELEASABLE PORTION OF LOADER
COMMON RELOCATION BASE
02000
00000
INPUT/OUTPUT CONTROL AND EQUIPMENT DRIVER SUBROUTINES
COMMON AREA FOR A & B ONLY
COMMON AREA FOR A, B & C
PROGRAM C
PROGRAM B
PROGRAM A
LINKAGE AREA
BASE PAGE PROGRAM C SEGMENT
BASE PAGE PROGRAM B SEGMENT
BASIC BINARY LOADER
BASIC CONTROL SYSTEM
COMMON BLOCK
PROGRAM AVAILABLE MEMORY
BASE PAGE AVAILABLE MEMORY
SYSTEM LINKAGE
RESERVED LOCATIONS

MEMORY MAP 3


BASIC BINARY LOADER
INPUT/OUTPUT CONTROL AND EQUIPMENT DRIVER SUBROUTINES
BASIC CONTROL SYSTEM
COMMON BLOCK
PROGRAM AVAILABLE MEMORY
PROGRAM D
PROGRAM B
PROGRAM A
02000
LINKAGE AREA
BASE PAGE AVAILABLE MEMORY
PROGRAM E
PROGRAM C
SYSTEM LINKAGE
RESERVED LOCATIONS

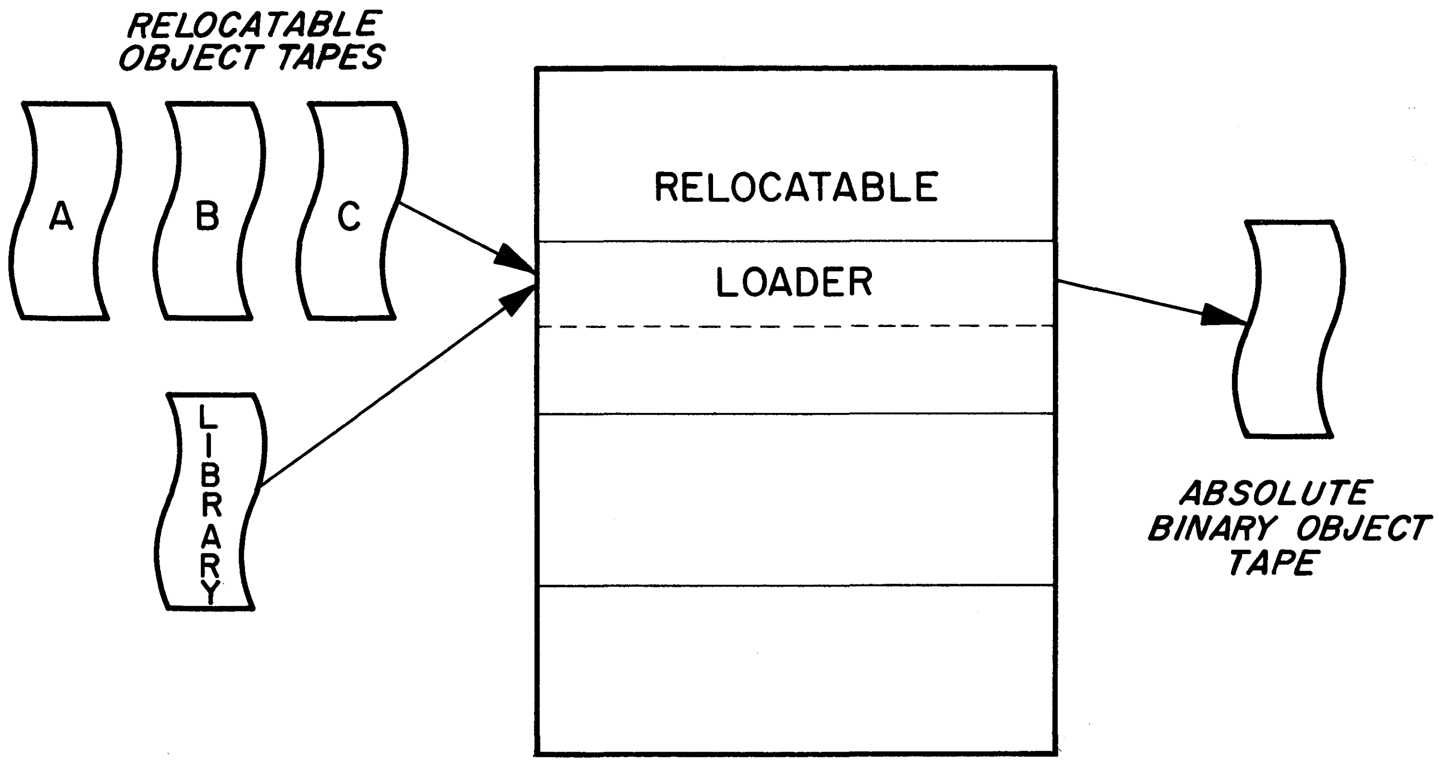
MEMORY MAP 4

```
LABEL         OP CODE       OPERAND            REMARKS

ASMB, R, B, L, T
              NAM           PROGA
              EXT           .IOC.
              ENT           BEGIN
START         NOP
BEGIN         MPY           SAM                PROGA
               .                               SEGMENT
              STA           MIKE
              COM           MIKE(512)
               .
               .
              END           START


ASMB, R, B, L, T
              NAM           PROGB
              EXT           FLOAT, BEGIN
               .
               .
              JSB           FLOAT
               .
              LDA           TABL               PROGB
              COM           TABL (255)         SEGMENT
               .
               .
              JMP           BEGIN
               .
               .
              END
```

LOADING EXAMPLE

LOADER LISTING

LOADER MESSAGES		COMMENTS
PROGA		
02000	03002	PROGRAM "A" IS LOADED
LOAD		MORE PROGRAMS?
PROGB		
03003	04006	YES, LOAD PROGRAM "B"
LOAD		LIST UNDEFINED EXT SYMBOLS; SW.Ø UP&PRESS RUN
MPY		
FLOAT		
LOAD		MORE PROGRAMS? YES, LOAD FROM LIBRARY
MPY		
04007	04117	LIBRARY ROUTINE
FLOAT		
04120	04124	LIBRARY ROUTINE
.PACK		
04125	04231	FLOAT CALLS .PACK
*LST		PRINT LOADER SYMBOL TABLE? SW. 15=Ø , YES-SW. 15=1, NO.
.IOC.	17515	ENTRY POINTS WITH ABSOLUTE ADDRESSES
.MEM.	16113	
BEGIN	02001	
MPY	04007	
FLOAT	04120	
.PACK	04125	
*COM		
15112	16111	COMMON STORAGE BOUNDS
*LINKS		
01773	01777	LOADER PROVIDED LINKAGES
*RUN		READY TO EXECUTE



BINARY OUTPUT OPTION SWITCH 14=1

THE LOADER PRODUCES THE ABSOLUTE BINARY OBJECT TAPE DURING A 'PSEUDO LOAD' OPERATION. THE ABSOLUTE BINARY TAPE CAN BE LOADED BY THE <u>BINARY</u> LOADER.

ABSOLUTE BINARY OUTPUT

MESSAGE	EXPLANATION	ACTION
*L01	CHECKSUM ERROR	REREAD THE RECORD
*L02	ILLEGAL RECORD	RIGHT TAPE? REREAD THE RECORD
*L03	MEMORY OVERFLOW	REVISE PROGRAM
*L04	LINKAGE AREA OVERFLOW	REVISE LOADING ORDER, OR REVISE PROGRAM
*L05	LOADER SYMBOL TABLE OVERFLOW	REVISE PROGRAM
*L06	COMMON BLOCK ERROR (Current common declaration exceeds initial common declaration)	LOAD PROGRAM CONTAINING THE LARGEST COMMON BLOCK FIRST
*L07	DUPLICATE ENTRY POINTS	REVISE PROGRAM
*L08	NO TRANSFER ADDRESS	LOAD STARTING ADDRESS IN 'A' REGISTER, PUSH RUN.
*L09	RECORD OUT OF SEQUENCE	REASSEMBLE PROGRAM OR RELOAD BCS AND TRY AGAIN

LOADER DIAGNOSTICS

	'T' REGISTER	EXPLANATION	ACTION
HALT 66	(102066)	TAPE SUPPLY ON 2753A PUNCH IS LOW	REPLENISH TAPE SUPPLY , PUSH RUN
HALT 55	(102055)	A LINE IS ABOUT TO BE PRINTED ON THE BINARY OUTPUT DEVICE	TURN PUNCH OFF, PUSH RUN
HALT 56	(102056)	A LINE HAS BEEN PRINTED WHILE THE PUNCH UNIT WAS OFF	TURN PUNCH ON, PUSH RUN

HALT INDEX, BINARY OUTPUT OPTION

PREPARE CONTROL SYSTEM (P.C.S.)

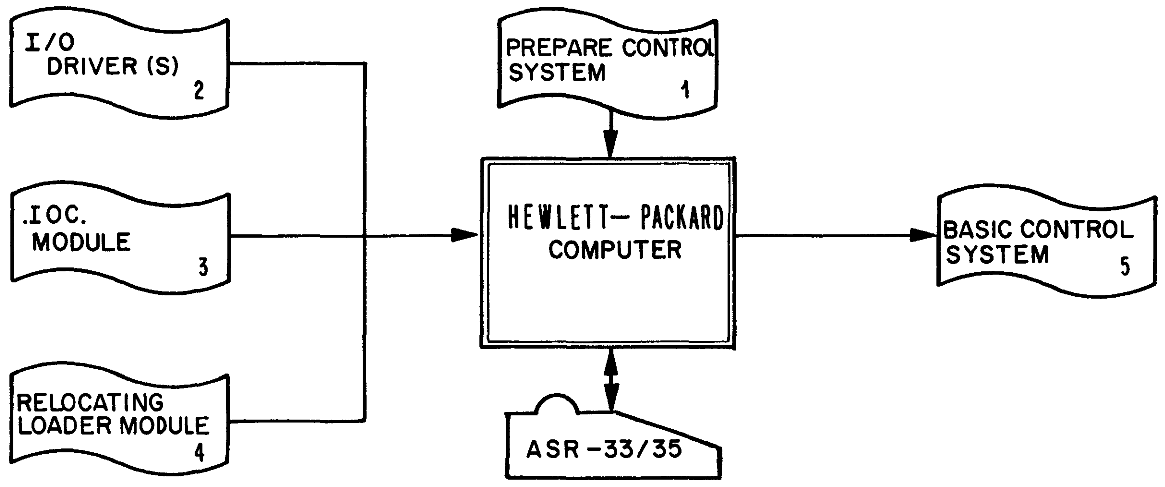
WHAT IS IT?

A COMPUTER PROGRAM WHICH PROCESSES RELOCATABLE MODULES OF THE BASIC CONTROL SYSTEM AND PRODUCES AN ABSOLUTE VERSION OF B.C.S. TAILORED TO THE SPECIFIC HARDWARE CONFIGURATION.

WHAT DOES IT DO?

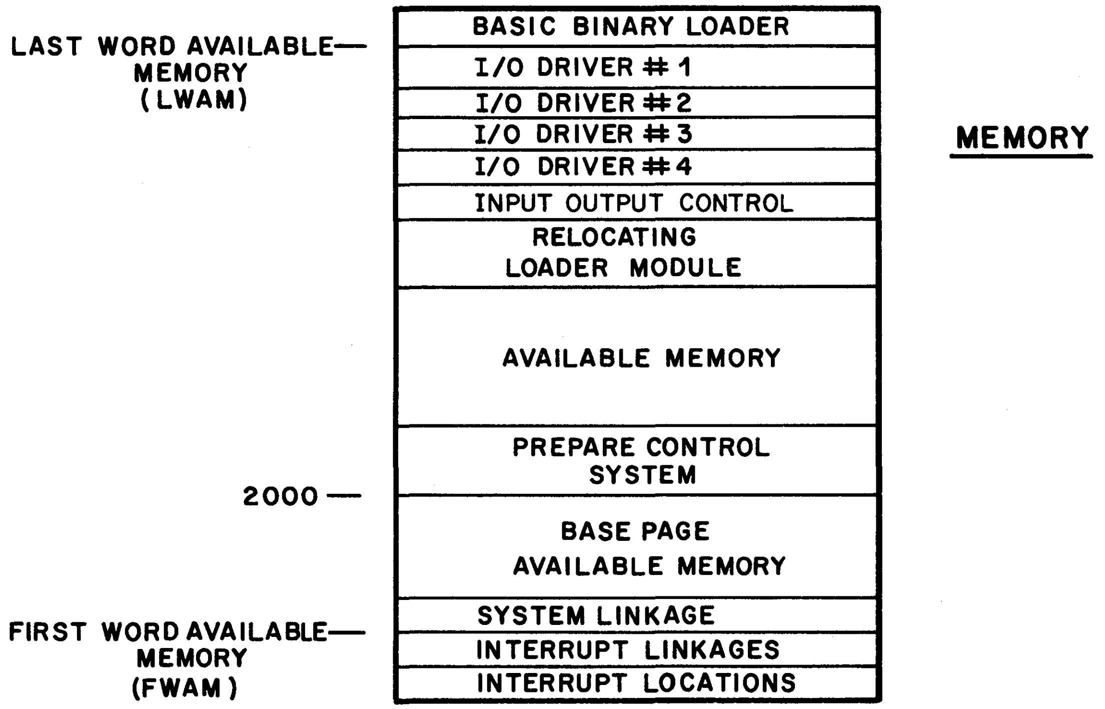
IT CREATES AN OPERATING SYSTEM CONSISTING OF THE INPUT/OUTPUT SUBROUTINE (I.O.C.), THE RELOCATABLE LOADER (LDR) AND THE REQUIRED PERIPHERAL EQUIPMENT INPUT/OUTPUT DRIVER SUBROUTINES.

PROCESSING ENVIRONMENT



P.C.S. OVER VIEW

P.C.S PROVIDES THE CAPABILITY OF CREATING A COMPLETE BASIC CONTROL SYSTEM IN THE COMPUTERS MEMORY.



WHEN ALL INDIVIDUAL ELEMENTS ARE PRESENT IN MEMORY. P.C.S. WILL PUNCH AN ABSOLUTE BINARY VERSION OF THE COMPLETE BASIC CONTROL SYSTEM.

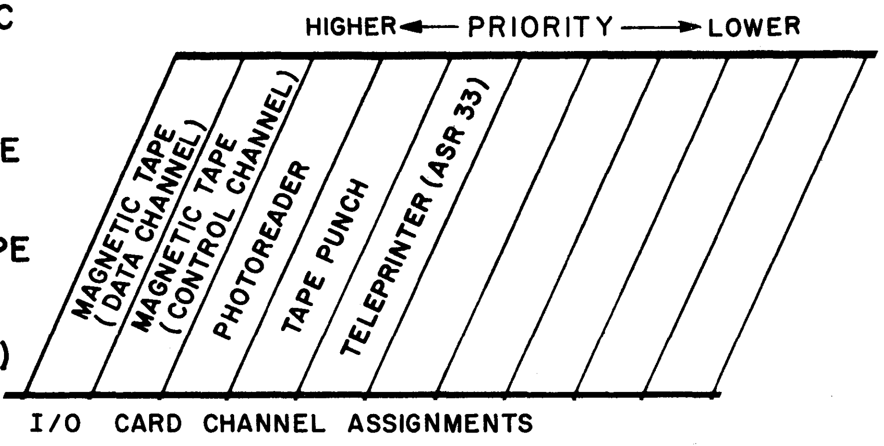
PLANNING THE SYSTEM

THE FIRST CONSIDERATION TO BE MADE IS THE PHYSICAL PLACEMENT OF THE I/O INTERFACE CARDS. CHANNEL #10 HAS THE HIGHEST PRIORITY, #11 NEXT HIGHEST, ETC. GENERALLY, THE DEVICE THAT GENERATES THE GREATEST NUMBER OF INTERRUPTS PER UNIT OF TIME IS ASSIGNED THE HIGHEST PRIORITY.

FOR EXAMPLE:

ASSUME A COMPUTER SYSTEM IS MADE UP OF THE FOLLOWING UNITS:

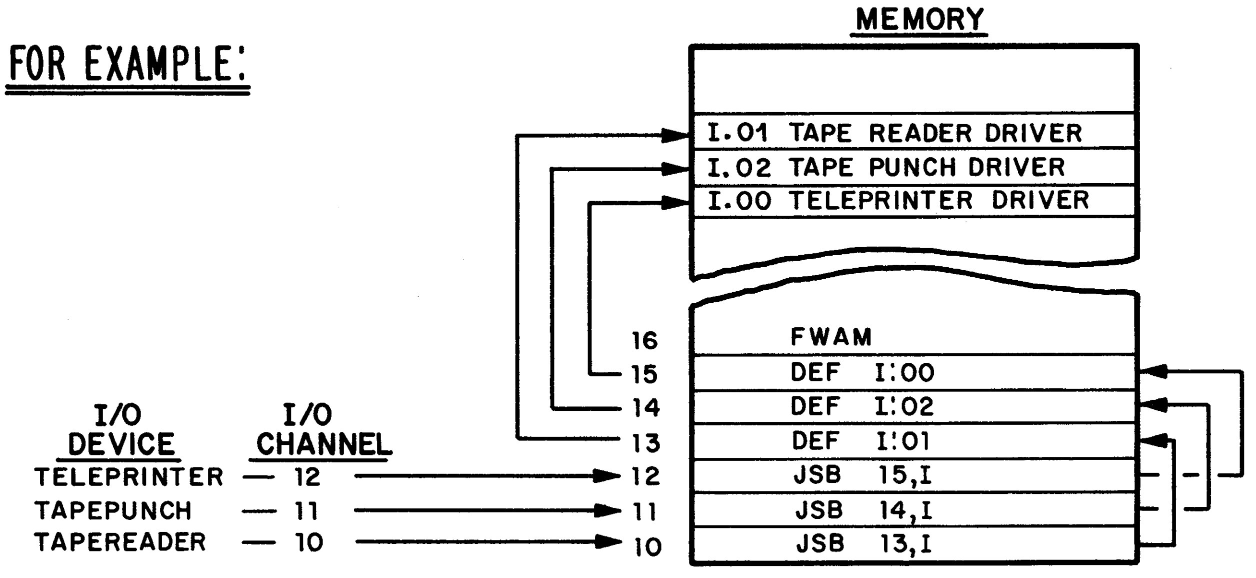
1. - READ/WRITE MAGNETIC TAPE (REQUIRES TWO INTERFACE BOARDS)
2. - HIGH-SPEED PAPER TAPE READER
3. - HIGH-SPEED PAPER TAPE PUNCH
4. - TELEPRINTER (ASR-33)


HIGHER ← PRIORITY → LOWER
MAGNETIC TAPE (DATA CHANNEL)
MAGNETIC TAPE (CONTROL CHANNEL)
PHOTOREADER
TAPE PUNCH
TELEPRINTER (ASR 33)
I/O CARD CHANNEL ASSIGNMENTS

INTERRUPT LINKAGE

WHEN AN I/O DEVICE CAUSES AN INTERRUPT IT FORCES THE COMPUTER TO EXECUTE THE CONTENTS OF THE INTERRUPT LOCATION. SINCE ALL INTERRUPT LOCATIONS ARE ON THE BASE PAGE AND THE I/O DRIVERS ARE IN HIGH MEMORY THE TRANSFER TO THE DRIVER MUST USE INDIRECT ADDRESSING.

FOR EXAMPLE:



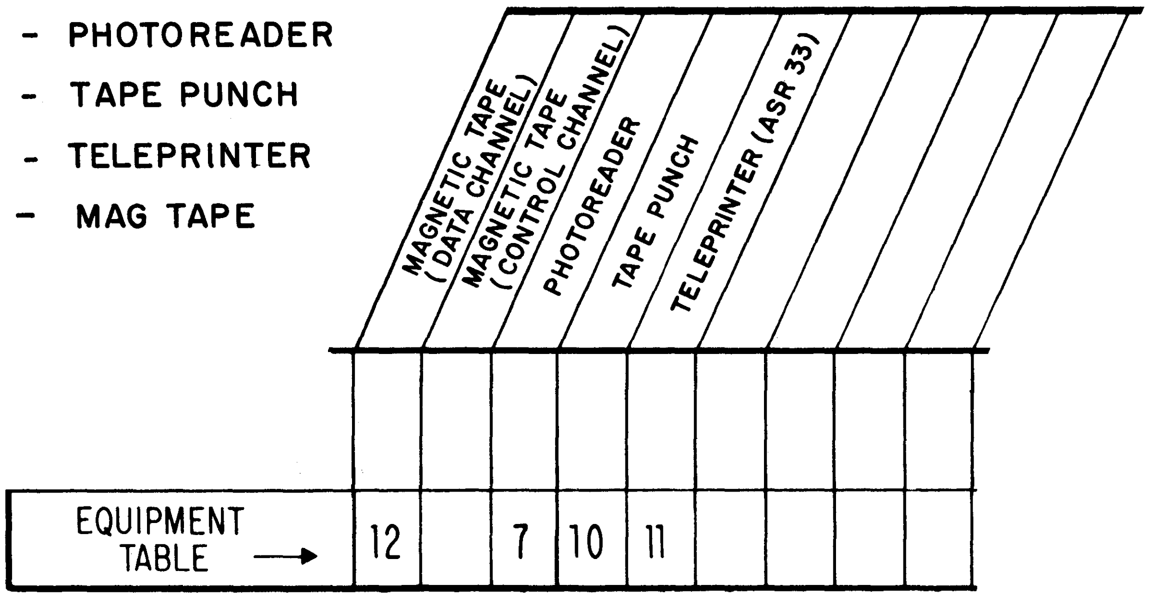
EQUIPMENT TABLE NUMBERS

EQUIPMENT TABLE NUMBERS BEGIN WITH 7. EACH DEVICE IS ASSIGNED A SEQUENTIAL OCTAL NUMBER. WITHIN THIS FRAMEWORK THE INITIAL NUMBER ASSIGNMENTS ARE ARBITRARY.

FOR EXAMPLE:

EQT

1st ENTRY	-	PHOTOREADER
2nd ENTRY	-	TAPE PUNCH
3rd ENTRY	-	TELEPRINTER
4th ENTRY	-	MAG TAPE



INTERRUPTS, LINKAGE, DRIVER I.D.

<u>INTERRUPT LOCATION</u> — P.C.S. WILL CAUSE A COMPUTER INSTRUCTION TO BE STORED HERE. (USUALLY A JSB, I)

<u>LINKAGE LOCATION</u> — P.C.S. WILL CAUSE THE ADDRESS OF THE <u>CONTINUATOR</u> SECTION OF THE I/O DRIVER TO BE STORED HERE.

<u>DRIVER IDENTIFICATION</u> — THE SYMBOLIC NAME OF THE I/O DRIVER INITIATOR SECTION ENTRY POINT.

<u>INTERRUPT IDENTIFICATION</u> — THE SYMBOLIC NAME OF THE I/O DRIVER CONTINUATOR SECTION ENTRY POINT.

<u>FWAM</u> — THE <u>F</u>IRST <u>W</u>ORD OF <u>A</u>VAILABLE <u>M</u>EMORY.

NOTE: DRIVER AND INTERRUPT ID CODES ASSIGNED BY H-P. THE SYMBOLS USED MUST BE UNIQUE.

	MAG TAPE (DATA CHANNEL)	MAG TAPE (CONTROL CHANNEL)	PHOTOREADER	TAPE PUNCH	TELEPRINTER (ASR 33)
DRIVER IDENTIFICATION	D.21	—	D.Ø1	D.Ø2	D.ØØ
INTERRUPT LOCATION	10	11	12	13	14
LINKAGE LOCATION	15	16	17	20	21
INTERRUPT IDENT.	I.21	C.21	I.Ø1	I.Ø2	I.ØØ
EQUIPMENT TABLE →	12		7	10	11

(22) FWAM

STANDARD UNIT NUMBERS

The standard unit numbers are simply pointers to the appropriate equipment table entries.

To assign standard units place a checkmark at the intersection of the standard unit table number (x-axis), and the correct equipment table number (y-axis)

	MAG TAPE (DATA CHANNEL)	MAG TAPE (CONTROL CHANNEL)	PHOTOREADER	TAPE PUNCH	TELEPRINTER (ASR 33)
DRIVER IDENTIFICATION	D.21	—	D.Ø1	D.Ø2	D.ØØ
INTERRUPT LOCATION	10	11	12	13	14
LINKAGE LOCATION	15	16	17	20	21
INTERRUPT IDENT.	I.21	C.21	I.Ø1	I.Ø2	I.ØØ
EQUIPMENT TABLE →	12		7	10	11

STANDARD UNIT TABLE					
1. KEYBOARD INPUT					✓
2. TELEPRINTER OUTPUT					✓
3. PROGRAM LIBRARY			✓		
4. PUNCH OUTPUT				✓	
5 INPUT			✓		
6. LIST OUTPUT					✓

P.C.S. OPERATIONS

THE NEXT FEW CHARTS WILL DESCRIBE A SIMPLE B.C.S. CONFIGURATION. THE SYSTEM WILL CONSIST OF A COMPUTER SYSTEM WITH 8K OF MEMORY AND THE FOLLOWING PERIPHERALS:

1. READ/WRITE MAGNETIC TAPE	— I/O CHANNELS	10,11
2. PHOTOELECTRIC PUNCHED PAPER TAPE READER	— I/O CHANNEL	12
3. HIGH SPEED PAPER TAPE PUNCH	— I/O CHANNEL	13
4. TELEPRINTER (ASR 33)	— I/O CHANNEL	14

THE ACTUAL CONFIGURATION PROCESS MAY BE DESCRIBED IN FIVE PHASES.

PHASE 1– INITIALIZATION

PHASE 2– LOADING THE I/O EQUIPMENT DRIVER

PHASE 3– LOADING THE IOC MODULE
- a. CREATING THE EQUIPMENT TABLE
- b. CREATING THE STANDARD UNIT TABLE

PHASE 4– LOADING THE RELOCATING LOADER MODULE
- a. ESTABLISH THE INTERRUPT LINKAGES

PHASE 5– PUNCH THE ABSOLUTE OUTPUT TAPE

INITIALIZATION PHASE

THE P.C.S. PROGRAM INITIALIZATION PHASE

COMMUNICATIONS	REMARKS
HS INP?	Is H.S. input unit available ?
17	Channel number of photo-reader
HS PUN?	Is H.S punch available ?
2Ø	Channel number of tape punch
FWA MEM?	Request first word address of available memory
22	First word following required interrupt locations
LWA MEM?	Request last word address of available memory
17677	Word preceding basic loader (8K memory)
*LOAD	Request to load first BCS module

THESE ENTRIES REFER TO THE "CONFIGURING" SYSTEM.

LOADING THE I/O EQUIPMENT DRIVERS

COMMUNICATIONS		REMARKS
D.21	→	MAGNETIC TAPE DRIVER PROCESSED*
1622Ø 17677	→	MEMORY BOUNDS OF THE DRIVER
* LOAD	→	REQUEST TO LOAD NEXT MODULE
D.Ø1 15661 16217		PHOTO-READER DRIVER PROCESSED
* LOAD		
D.Ø2 15351 1566Ø		TAPE PUNCH DRIVER PROCESSED
* LOAD		
D.ØØ 14615 1535Ø		TELEPRINTER DRIVER PROCESSED
* LOAD		

* *WHEN PRESENT, THIS DRIVER SHOULD BE LOADED FIRST DUE TO ITS LARGE SIZE*

LOADING THE IOC MODULE

COMMUNICATIONS		REMARKS
IOC 14376 14614	→	IOC MODULE PROCESSED
* TABLE ENTRY	→	EQUIPMENT TABLE
EQT?		UNIT #
12,D.Ø1	→	7
13,D.Ø2	→	10
14,D.ØØ	→	11
1Ø,D.21	→	12
/E		
SQT?	→	STANDARD UNIT TABLE
-KYBD? 11 -TTY? 11 -LIB? 7 -PUNCH? 1Ø -INPUT? 7 -LIST? 11		
DMA? Ø	→	DIRECT MEMORY ACCESS OPTION. Ø INDICATES DMA NOT AVAILABLE.
* LOAD		

	MAGNETIC TAPE (DATA CHANNEL)	MAGNETIC TAPE (CONTROL CHANNEL)	PHOTOREADER	TAPE PUNCH	TELEPRINTER (ASR 33)
DRIVER IDENTIFICATION	D.21	—	D.Ø1	D.Ø2	D.ØØ
INTERRUPT LOCATION	10	11	12	13	14
LINKAGE LOCATION	15	16	17	20	21
INTERRUPT IDENT.	I.21	C.21	I.Ø1	I.Ø2	I.ØØ
EQUIPMENT TABLE →	12		7	10	11

STANDARD UNIT TABLE					
1. KEYBOARD INPUT					✓
2. TELEPRINTER OUTPUT					✓
3. PROGRAM LIBRARY			✓		
4. PUNCH OUTPUT				✓	
5 INPUT			✓		
6. LIST OUTPUT					✓

THE RELOCATING LOADER MODULE

DRIVER IDENTIFICATION	D.21	–	D.Ø1	D.Ø2	D.ØØ
INTERRUPT LOCATION	10	11	12	13	14
LINKAGE LOCATION	15	16	17	20	21
INTERRUPT IDENT.	I.21	C.21	I.Ø1	I.Ø2	I.ØØ

COMMUNICATIONS

```
LOADR
 12115 14346


 INTERRUPT LINKAGE?


10,15,I.21
11,16,C.21
12,17,I.01  (ERROR)
*UN NAME    ( 0≠Ø )
12,17,I.Ø1
13,2Ø,I.Ø2
14,21,I.ØØ
/E                  ——→ LIST TERMINATOR
```

MEANING	ENTRY POINT LIST	
ADDRESS OF SYSTEM TABLE	.SQT.	14347
ADDRESS OF EQUIP. TABLE	.EQT.	14355
I/O DRIVER – INITIATOR AND CONTINUATOR ENTRY POINTS	D.21	1622Ø
	I.21	17216
	C.21	1713Ø
	D.Ø1	15661
	I.Ø1	15776
	D.Ø2	15351
	I.Ø2	15465
MAINTAINS COMPATABILITY BETWEEN BUFFERED AND UNBUFFERED VERSIONS OF I.O.C.	.BUFR	14544
	D.ØØ	14615
	I,ØØ	14771
ADDRESS OF .IOC. ENTRY POINT	.IØC.	14376
DMA STATUS WORD CH #1	DMAC1	14613
DMA STATUS WORD CH #2	DMAC2	14614
ADDRESS OF I/O ERROR HALT	IOERR	14572
SYSTEM TABLE LINK WORD	XSQT	14611
EQUIPMENT TABLE LINK WORD	XEQT	14612
RELOCATING LOADER ENTRY POINT	.LDR.	136Ø1
ADDRESS OF MEMORY TABLE*	.MEM.	14342
LOADER SYMBOL TABLE ADDRESS	LST	12141

*SYSTEM LINK
ØØØ22 00153

* MEMORY TABLE

FWABP	______
LWABP	______
FWAM	______
LWAM	______

*BCS ABSOLUTE OUTPUT

ADDITIONAL P.C.S./B.C.S. CAPABILITIES

SETTING CONSTANTS INTO INTERRUPT LOCATIONS

```
1Ø, 15, I.ØØ
11, 16, I.Ø4
12, 17, I.Ø2                                          LOC.  CONTENTS₈
13, 1Ø6713    These entries will cause P.C.S. to do this   13    1Ø6713
14, Ø                                                 14    ØØØØØØ
```

SPECIFYING INTERRUPT AND/OR SYSTEM ROUTINES AS EXTERNAL

11, 16, I.Ø4	Original input entry.
* UN NAME	P.C.S. diagnostic message.
!	Response to establish the name as external.
	— OR —
11, 16, I.Ø3	Response to indicate a corrected input entry.

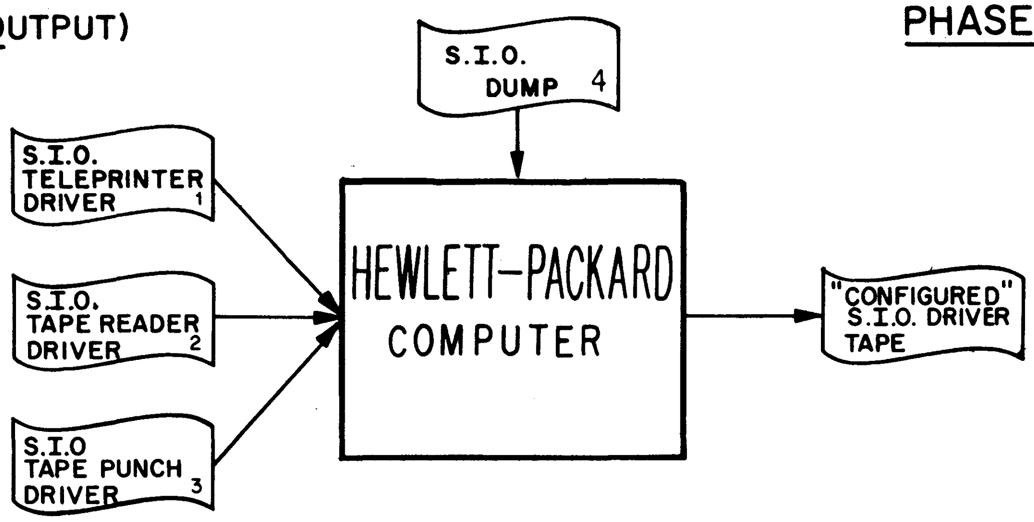
SPECIFYING I/O DRIVERS AS EXTERNAL

I/O DRIVER ? D.XX	P.C.S. diagnostic message indicating the referenced driver has not been loaded.
!	Response to define the driver as external
* UNDEFINED SYMBOL: XXXX	P.C.S. diagnostic caused by specifying a driver as external. The computer will halt. Push run to continue. Each undefined symbol is given the dummy address 77777.

SYSTEM INPUT-OUTPUT DUMP ROUTINE

(SYSTEM INPUT OUTPUT)

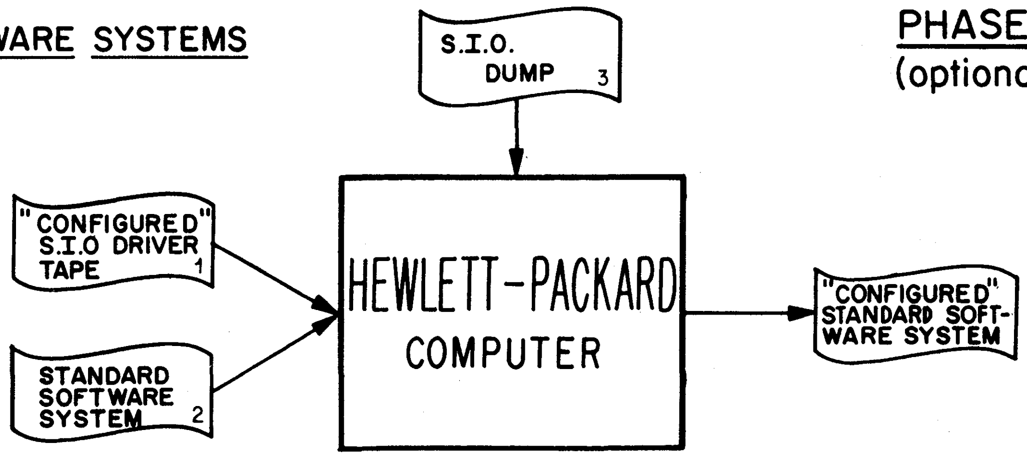
PHASE 1


S.I.O. DUMP 4
S.I.O. TELEPRINTER DRIVER 1
S.I.O. TAPE READER DRIVER 2
S.I.O TAPE PUNCH DRIVER 3
HEWLETT-PACKARD COMPUTER
"CONFIGURED" S.I.O. DRIVER TAPE

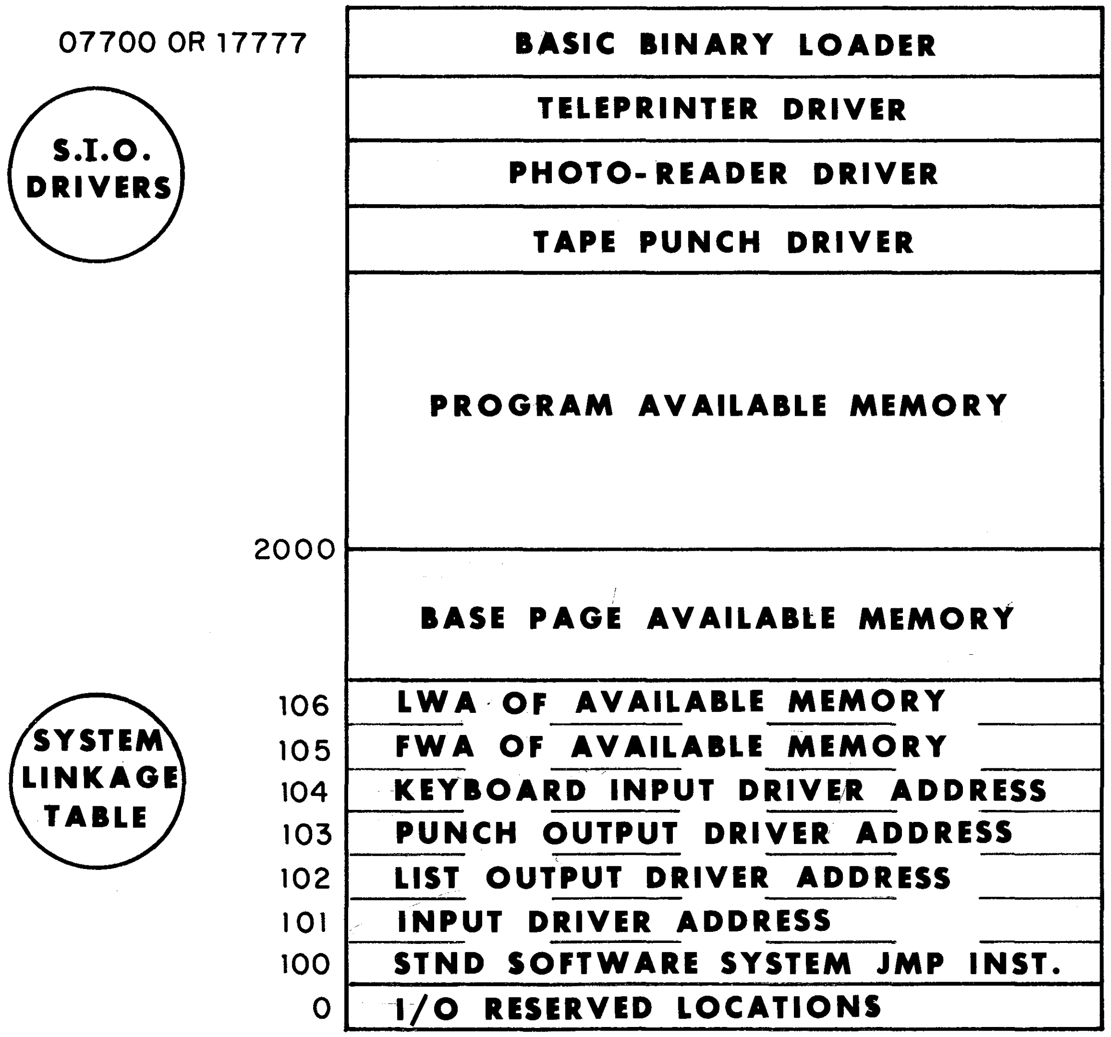
STANDARD SOFTWARE SYSTEMS

PHASE 2
(optional)

FORTRAN
ALGOL
ASSEMBLER
SYMBOLIC EDITOR


S.I.O. DUMP 3
"CONFIGURED" S.I.O DRIVER TAPE 1
STANDARD SOFTWARE SYSTEM 2
HEWLETT-PACKARD COMPUTER
"CONFIGURED" STANDARD SOFTWARE SYSTEM

S.I.O. MEMORY MAP


07700 OR 17777
BASIC BINARY LOADER
TELEPRINTER DRIVER
PHOTO-READER DRIVER
TAPE PUNCH DRIVER
S.I.O. DRIVERS
PROGRAM AVAILABLE MEMORY
2000
BASE PAGE AVAILABLE MEMORY
106 LWA OF AVAILABLE MEMORY
105 FWA OF AVAILABLE MEMORY
104 KEYBOARD INPUT DRIVER ADDRESS
103 PUNCH OUTPUT DRIVER ADDRESS
102 LIST OUTPUT DRIVER ADDRESS
101 INPUT DRIVER ADDRESS
100 STND SOFTWARE SYSTEM JMP INST.
0 I/O RESERVED LOCATIONS
SYSTEM LINKAGE TABLE

CONFIGURING A PROGRAM SYSTEM

THE SYSTEMS TO BE CONFIGURED

- ASSEMBLER SYSTEM
- SYMBOLIC EDITOR SYSTEM
- FORTRAN COMPILER SYSTEM-PASS 1 TAPE ONLY
- ALGOL COMPILER

THE S.I.O. DRIVERS (ONLY PROVIDED WHEN I/O DEVICE ORDERED)

- TELEPRINTER
- TAPE READER
- TAPE PUNCH

THE PROCEDURE (BASIC BINARY LOADER USED FOR ALL MODULE LOADING)

1. LOAD A DRIVER. (THE TELEPRINTER MUST BE LOADED FIRST)(PHOTOREADER SECOND)(PUNCH LAST)
2. PLACE THE ADDRESS 2 INTO THE P-REGISTER; SET SWITCHES 5-0 OF THF SWITCH REGISTER TO THE CHANNEL NUMBER ASSOCIATED WITH THAT DEVICE AND PRESS RUN.
3. REPEAT ABOVE STEPS FOR EACH DRIVER TO BE INCLUDED.
4. LOAD THE PERTINENT PROGRAMMING SYSTEM.
5. LOAD THE S.I.O. DUMP ROUTINE.
6. PLACE THE ADDRESS 2 INTO THE P-REGISTER & SET SWITCH 15 OF THE SWITCH REGISTER TO OBTAIN THE FOLLOWING OPTIONS:

   0 = OUTPUT TO CONTAIN ONLY S.I.O. DRIVERS AND SYSTEM LINKAGE TABLE.
   1 = PROGRAM SYSTEM IS TO BE INCLUDED ON OUTPUT.
7. PRESS RUN TO COMMENCE PUNCH-OUT.
8. MULTIPLE COPIES MAY BE OBTAINED BY REPEATING FROM SWITCH 15 SETTING OF STEP 6.

Introduction to HP BASIC XII

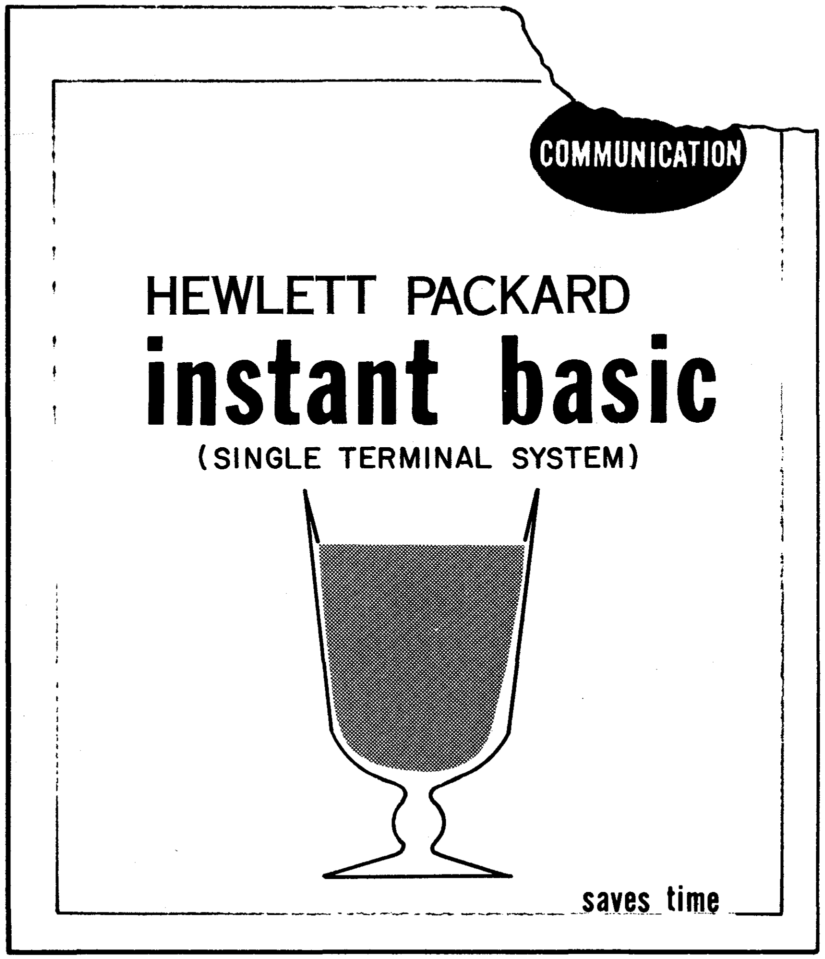
LESSON XII
INTRODUCTION TO HP BASIC

LESSON XII OBJECTIVES

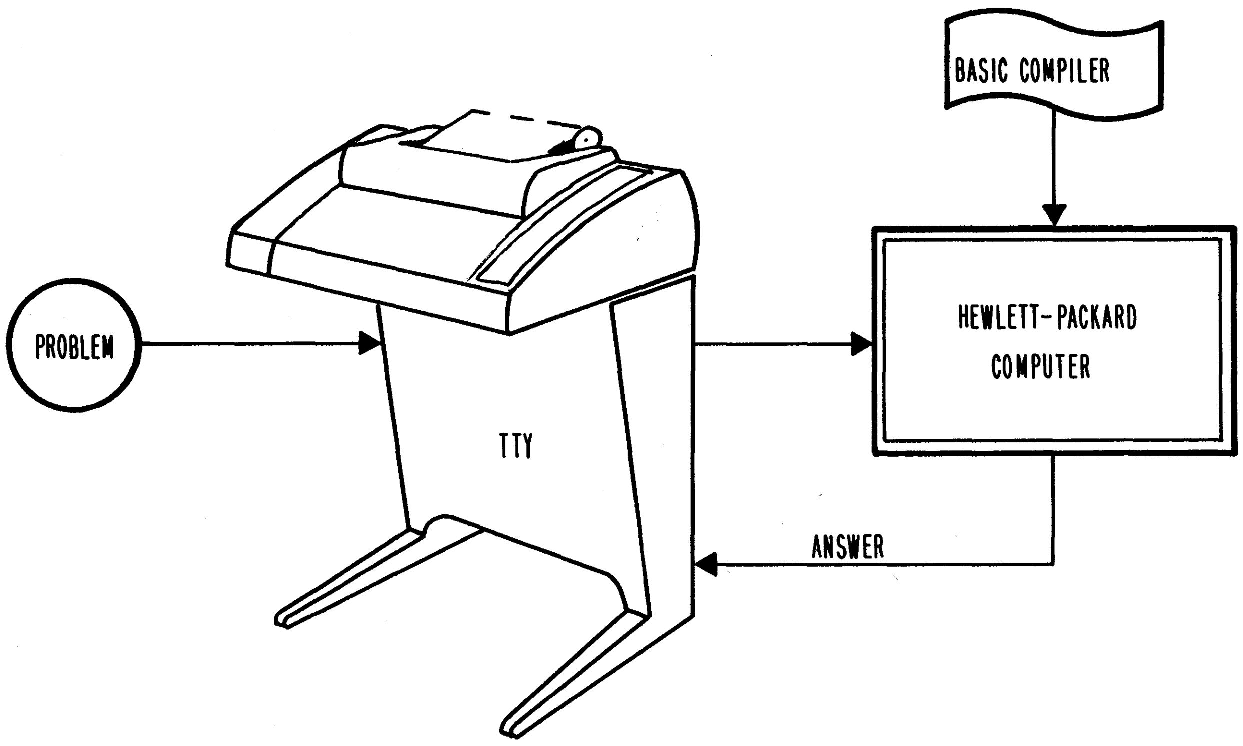
THE PRINCIPLE OBJECTIVES OF LESSON XII ARE:

1. **TO INTRODUCE THE STUDENT TO THE ELEMENTS OF THE HEWLETT- PACKARD SINGLE TERMINAL BASIC COMPILER.**
2. **TO PRESENT THE LANGUAGE AND OPERATING CAPABILITIES IN SUFFICIENT DETAIL TO PERMIT THE STUDENT TO CREATE SOLUTIONS TO SIMPLE PROBLEMS WITH RELATIVELY LITTLE INSTRUCTION TIME REQUIRED.**
3. **TO ILLUSTRATE THE EASE AND FLEXIBILITY OF USING THE SYSTEM, BY PROVIDING SAMPLE PROBLEM SOLUTIONS, FOR ANALYSIS, AND SUGGESTIONS FOR PROGRAMMING.**

NOTE: **THE "BASIC" LANGUAGE WAS DEVELOPED BY DARTMOUTH COLLEGE, UNDER THE DIRECTION OF PROFESSORS JOHN G. KEMENY AND THOMAS E. KURTZ. THE HEWLETT-PACKARD BASIC COMPILER IS AN ADAPTATION OF THAT DEVELOPMENT.**


COMMUNICATION
HEWLETT PACKARD
instant basic
(SINGLE TERMINAL SYSTEM)
saves time

BASIC OPERATING ENVIRONMENT



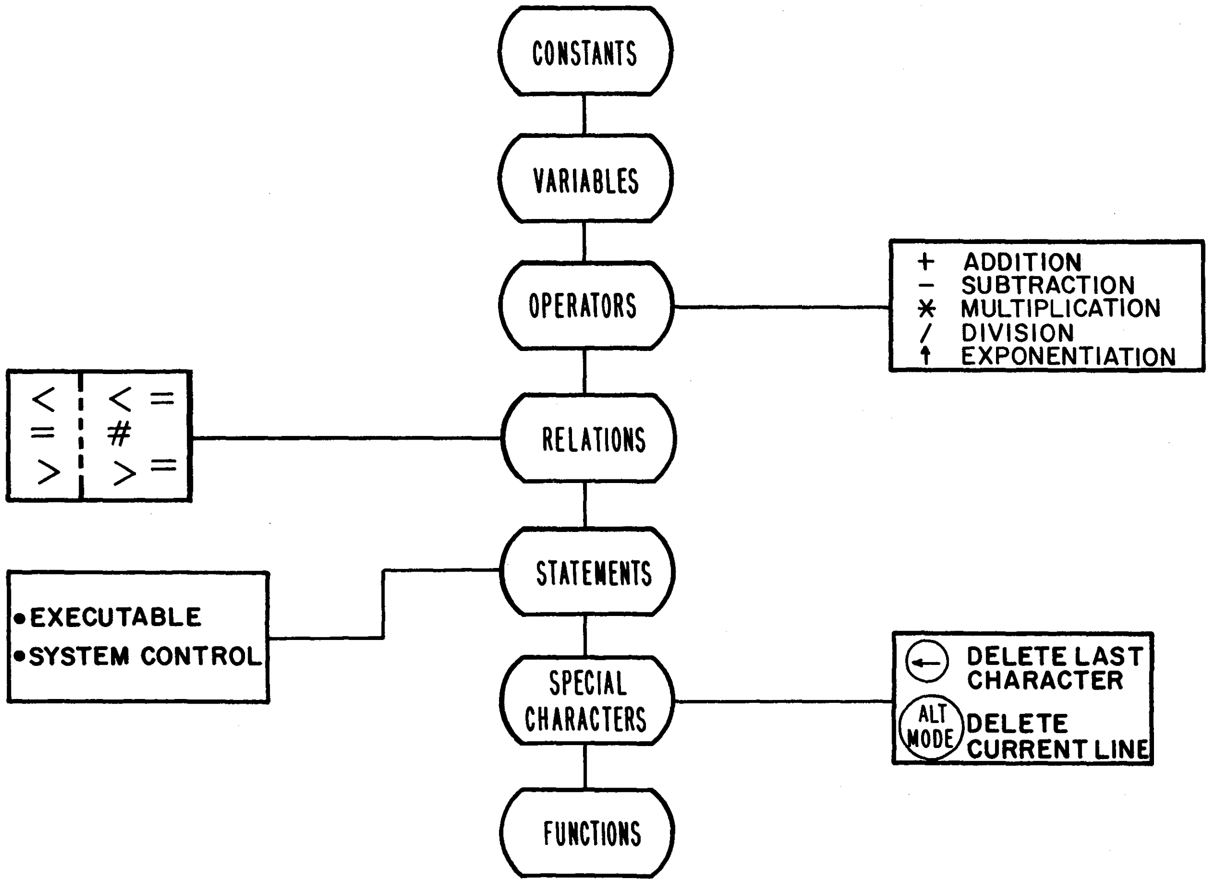
USING THE HP BASIC LANGUAGE

```
READY

10  FOR N = 1 TO 7

20  PRINT N, SQR (N)

31  NEXT N

43  PRINT "DONE"

50  END
```

NOTE

1. CONVERSATIONAL MODE
2. EACH STATEMENT MUST HAVE A STATEMENT NUMBER WHICH IDENTIFIES ITS SEQUENCE WITHIN THE PROGRAM.
3. FREE FORM – SELF TEACHING
4. ALL STATEMENTS ARE TERMINATED BY $\boxed{\text{CR}}$
5. THE HIGHEST NUMBERED STATEMENT MUST BE AN END STATEMENT.

THE BASIC LANGUAGE COMPONENTS



STATEMENTS

EXECUTABLE

ARITHMETIC

- LET

CONTROL

- GO TO
- IF
- FOR
- NEXT
- END

INPUT/OUTPUT

- READ
- DATA
- PRINT
- INPUT

SYSTEM CONTROL

- LIST
- RUN
- SCRATCH
- STOP

FUNCTIONS

SIN (X)	SINE X
COS (X)	COSINE X
TAN (X)	TANGENT X
ATN (X)	ARCTANGENT X
EXP (X)	e^X
LOG (X)	Ln x
ABS (X)	Absolute value of x
SQR (X)	$\sqrt{X}$
INT (X)	INTeger part of x
SGN (X)	Sign of x

CONSTANTS

ALL NUMBERS ARE REPRESENTED IN THE COMPUTER IN FLOATING-POINT FORMAT. THE RANGE IS -10^{38} TO 10^{38}.

EXAMPLES: 3 , 5 , 7 , – 65
1.5, 14.7E-2, .45E7, 1000

VARIABLES

GENERAL FORM: $\alpha \lambda$

WHERE α MUST BE A LETTER (A-Z)
λ MUST BE A NUMERIC (0-9)

EXAMPLES: B2, K2, K6, R7, Z, F

LINE NUMBERS

$1 \leq$ Line # ≤ 9999

STATEMENTS → EXECUTABLE

ARITHMETIC — the LET statement

GENERAL FORM: line # LET variable = formula

EXAMPLE:

```
152 LET X  =  12.Ø
3Ø1 LET A1 =   4 + 3 * X
451 LET Z  =  (TAN(X) ↑ A1)/88.98
```

INPUT-OUTPUT — the PRINT statement

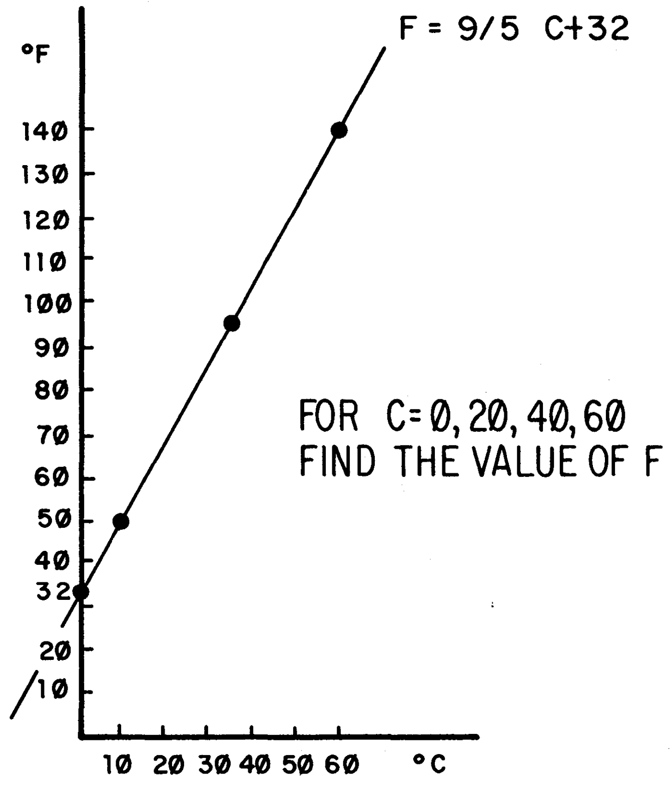
GENERAL FORM: line # PRINT variable / formula / message

EXAMPLE:

```
657 PRINT A1;X
737 PRINT TAN(X);(4*5) ↑ 2; A1
8Ø8 PRINT "START PROCESS", A1*COS(X), 367
```

EXAMPLE

CENTIGRADE TO FAHRENHEIT CONVERSION



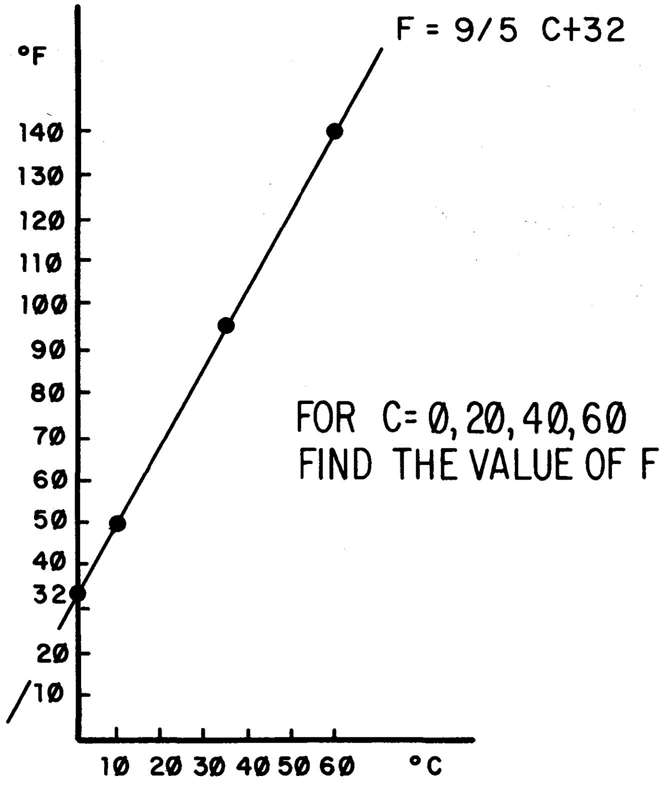
```
10    LET C=0
20    LET F = 9/5*C+32
30    PRINT "FOR C = ";C;"THEN F = ";F
35    LET C=20
40    LET F= (9/5)*C+32
45    PRINT "FOR C = ";C; "THEN F= "; F
 .           .
 .           .
 .           .
150   LET C = 60
155   LET F= 9/5*C +32
160   PRINT "FOR C = "; C; "THEN F = ";
170   PRINT 9/5*C+32
             .
             .
             .
            ETC
```

THE GO TO STATEMENT

GENERAL FORM: line # GO TO line #

EXAMPLE: 101 GO TO 35

EXAMPLE:



```
10   LET C = 0
20   LET F = 9/5*C + 32
30   PRINT "FOR C ="; C; "THEN F ="; F
40   LET C = C + 20
50   GO TO 20
99   END
```

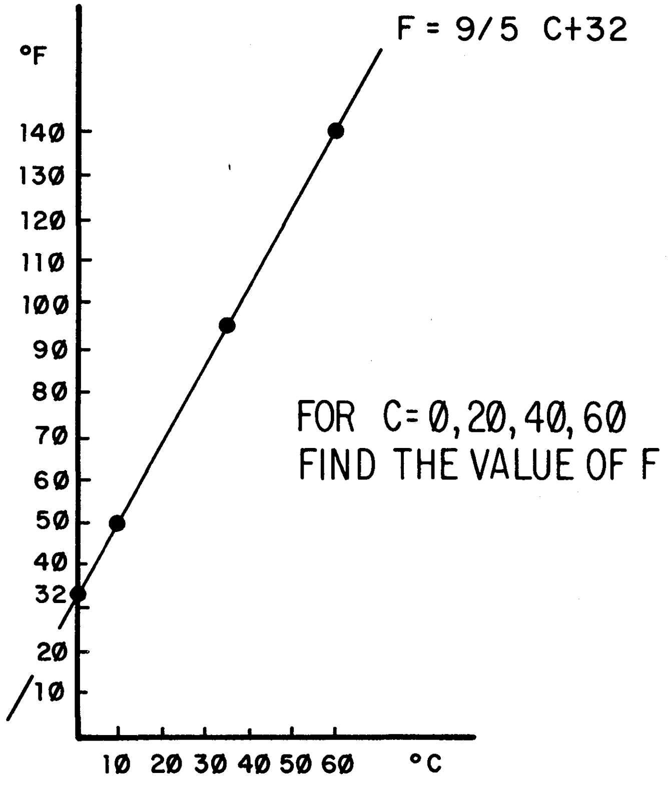
THE IF STATEMENT

GENERAL FORM: line # IF {formula} {relation} {formula} THEN {line #}

EXAMPLE:

EXAMPLES:

```
10 IF (A+B)*C < T↑2 THEN 461
20 IF K > 4.7 THEN 34
30 IF X <= SQR(6932) THEN 90
```



```
10  LET  C = 0
20  LET  F = 9/5*C + 32
30  PRINT "FOR C ="; C; "THEN F ="; F
40  LET C = C+20
50  IF  C <= 60 THEN 20
60  PRINT "THAT IS IT"
99  END
```

INPUT OUTPUT STATEMENTS

THE READ STATEMENT
THE DATA STATEMENT

GENERAL FORM:

$$\text{line \# READ } \left\{\text{variable, variable, variable....}\right\}$$
$$\text{line \# DATA } \left\{\text{value, value, value ....}\right\}$$

EXAMPLE:

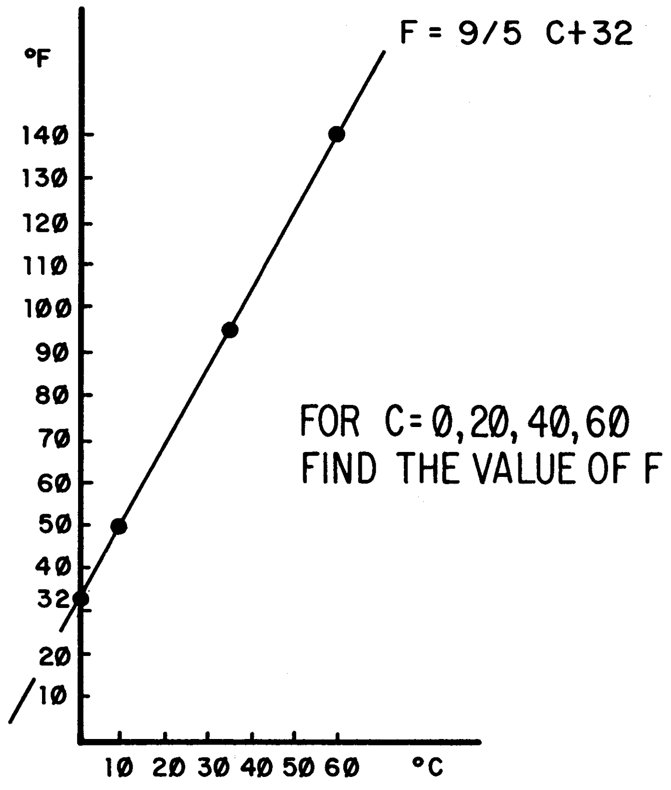
$$\left.\begin{array}{l}\text{READ A1, A2, A3, A4, A5} \\ \text{DATA 2, 13, 4, 8, 1.7E2}\end{array}\right\}$$

A1 = 2 A3 = 4
A2 = 13 A4 = 8
A5 = 1.7E2

EXAMPLE:

$$\left.\begin{array}{l}\text{READ A1, A2, A3, A4, A5} \\ \text{DATA 2, 13, 4} \\ \text{DATA 8, 1.7E2}\end{array}\right\}$$

EXAMPLE



```
10   READ C1, C4, CØ
2Ø   DATA Ø, 6Ø, 2Ø
3Ø   LET C = C1
4Ø   LET F = C*9/5 + 32
5Ø   PRINT "FOR C ="; C; "THEN F ="; F
6Ø   LET C = C + CØ
7Ø   IF C < = C4 THEN 4Ø
99   END
```

THE INPUT STATEMENT

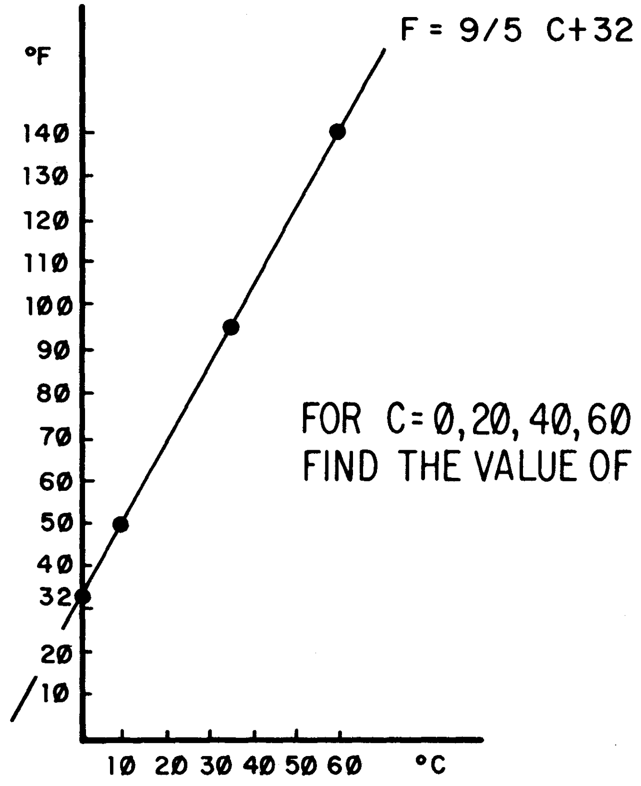
GENERAL FORM:

line # INPUT $\left\{ \text{Variable} \right\}, \left\{ \text{variable} \right\}, \cdots \left\{ \text{variable} \right\}$

EXAMPLES:

```
35   INPUT  A1, A2, A3

56   INPUT  B1 , B, C, X, A(3)
```

EXAMPLE



FOR C=0,20,40,60
FIND THE VALUE OF F

```
20   INPUT C1,C4,C0
30   LET C = C1
40   LET F = C*9/5 + 32
50   PRINT "FOR C ="; C; "THEN F ="; F
60   LET C = C + C0
70   IF C < = C4 THEN 40
99   END
```

THE FOR AND NEXT STATEMENTS

GENERAL FORM:

line # FOR {Variable} = {formula} TO {formula} STEP {formula}

line # NEXT {Variable}

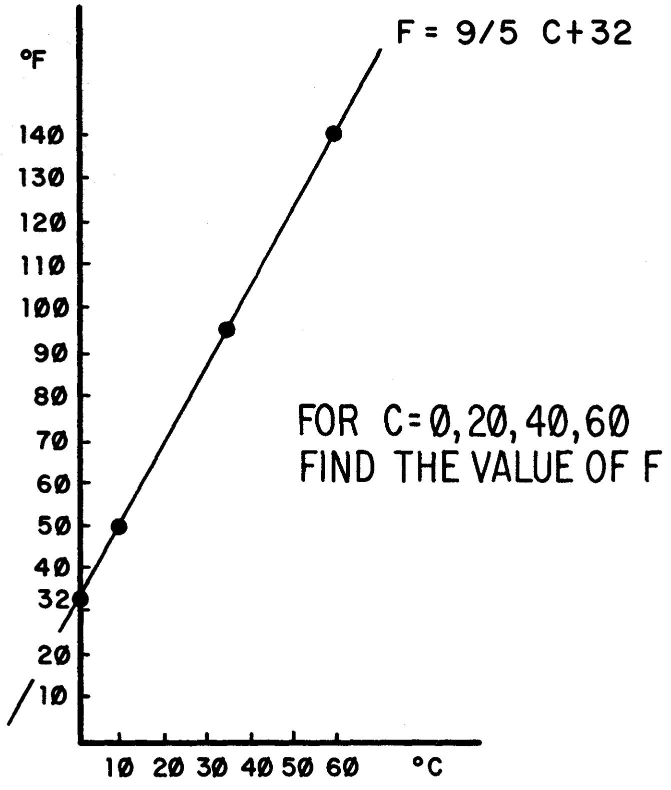
EXAMPLE:

```
753   FOR A3 = B+7  TO B+21  STEP 2

856   NEXT  A3
```

EXAMPLE:

```
333   FOR A7 = 3 TO 10 STEP 1

36Ø   LET A9 = A7 ↑ 2

37Ø   PRINT A7, A9

4ØØ   NEXT A7
```

EXAMPLE



```
1Ø   READ C1,  C4, CØ
2Ø   DATA Ø, 6Ø, 2Ø
3Ø   FOR C = C1 TO C4 STEP CØ
45   PRINT "FOR C ="; C; "THEN F =";
47   PRINT C*9/5 + 32
5Ø   NEXT C
6Ø   PRINT "FINISHED"
99   END
```

USING SUBROUTINES

THE GO SUB STATEMENT

GENERAL FORM: GO SUB line #

THE RETURN STATEMENT

GENERAL FORM: RETURN

EXAMPLE:

```
 10   PRINT "READ FIRST SET OF VALUES"
 20   GO SUB 666
 30   PRINT "READ SECOND SET OF VALUES"
 40   GO SUB 666
 50   PRINT "READ THIRD SET OF VALUES"
 55   GO SUB 666
 60   GO TO 777
666   INPUT A,B,C
676   IF A<=0 THEN 777
686   IF B<=0 THEN 777
696   IF C<=0 THEN 777
700   RETURN
777   END
```

SYSTEM CONTROL STATEMENTS

LIST

▶ GENERAL FORM: LIST {line #}

EXAMPLE: LIST 30 — LIST FROM STATEMENT 30 UNTIL THE END STATEMENT

EXAMPLE: LIST — LIST THE ENTIRE PROGRAM

SCRATCH

▶ GENERAL FORM: SCRATCH
IT DELETES THE CURRENT PROGRAM IN MEMORY

RUN

▶ GENERAL FORM: RUN
IT STARTS EXECUTION OF THE PROGRAM

STOP

▶ GENERAL FORM: STOP
IT STOPS EXECUTION OF THE PROGRAM

HELPFUL HINTS

FREQUENTLY TYPING ERRORS TAKE PLACE AND CORRECTIONS ARE NEEDED, THEREFORE:

1. ALT MODE KEY DELETES CURRENT LINE

EXAMPLE

```
37 LET A-B+C\ ← TYPED BY BASIC TO INDICATE
                DELETION
```

2. ← DELETES THE PREVIOUS CHARACTER

EXAMPLE

```
42 FER ← ← OR X=3 TO 7 STEP Ø.1

     (42 FOR X=3 TO 7 STEP Ø.1)
```

3. TO DELETE A LINE, TYPE THE LINE # WITH (CR)

EXAMPLE

```
151 (CR)
```

DATA FORMATTING

(SPECIAL USE OF THE COMMA AND SEMI-COLON)

WHEN USING PRINT COMMANDS THE TELETYPE IS DIVIDED INTO 5 ZONES STARTING AT POSITIONS Ø, 15, 3Ø, 45, AND 6Ø.

COMMA-CONTROLS PRINT ZONES FROM POSITIONS Ø, 15, 3Ø, 45 AND 6Ø.

SEMI-COLON-INHIBITS ZONE SPACING.

EXAMPLES

1.

```
35 FOR X=4 TO 1Ø STEP 2
6Ø PRINT X, X + 1
7Ø NEXT X
```

2.

```
35 FOR X=4 TO 1Ø STEP 2
6Ø PRINT X, X + 1,
7Ø NEXT X
```

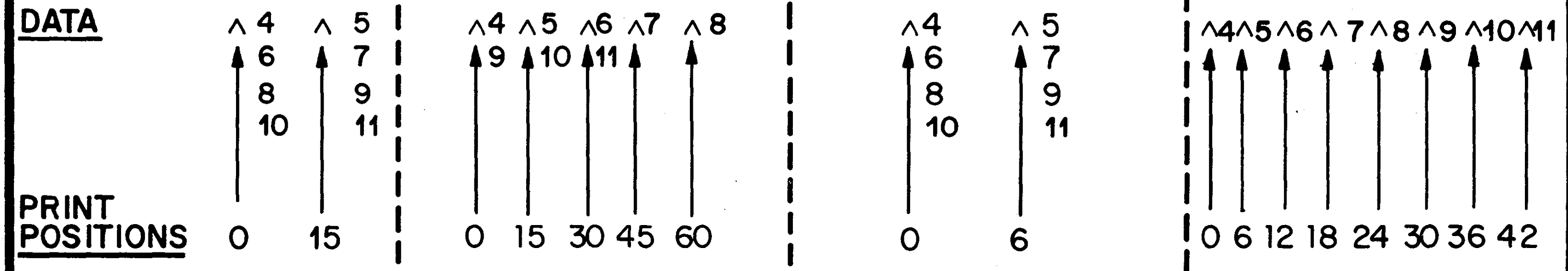
3.

```
35 FOR X=4 TO 1Ø STEP 2
4Ø PRINT X; X + 1
7Ø NEXT X
```

4.

```
35 FOR X=4 TO 1Ø STEP 2
4Ø PRINT X; X + 1;
7Ø NEXT X
```

OUTPUT RESULTS



NOTE: ∧= space

THE TAB FEATURE

GENERAL FORM: TAB (POS #)

EXAMPLE: PRINT THE MESSAGE "THREE" BEGINNING IN POSITION 36 AND THE VALUE FOR 10+3 ↑ IN POSITION 48

```
17 PRINT TAB (36), "THREE", TAB(48), 1Ø+3↑2
```

```
     THREE            ∧19
     ↑                ↑
   Position         Position
     36               48
```

ERROR DIAGNOSTICS

FORMAT: ERROR XX IN LINE NN

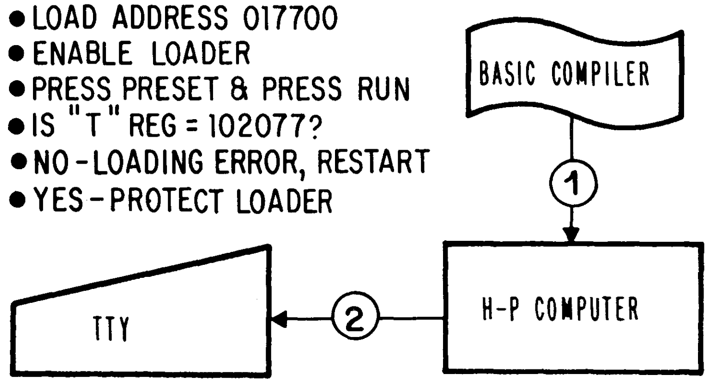
9	MISSING OR INCORRECT FUNCTI
10	MISSING PARAMETER IN DEF STATEMENT.
11	MISSING ASSIGNMENT OPERATOR.
12	MISSING THEN.
13	MISSING OR INCORRECT FOR-VARIABLE.
14	MISSING TO.
15	INCORRECT STEP IN FOR STATEMENT.
16	CALLED ROUTINE DOES NOT EXIST.
17	WRONG NUMBER OF PARAMETERS IN CALL STATEMENT.
18	MISSING OR INCORRECT CONSTANT IN DATA STATEMENT.
19	MISSING OR INCORRECT VARIABLE IN READ STATEMENT.
20	NO CLOSING QUOTE FOR PRINT STRING.
21	MISSING PRINT DELIMITER OR BAD PRINT QUANTITY.
22	ILLEGAL WORD FOLLOWS MAT.
23	MISSING DELIMITER.
24	IMPROPER MATRIX FUNTION

SINGLE TERMINAL BASIC

LOADING INSTRUCTIONS

FOR 1

- LOAD ADDRESS 017700
- ENABLE LOADER
- PRESS PRESET & PRESS RUN
- IS "T" REG = 102077?
- NO-LOADING ERROR, RESTART
- YES-PROTECT LOADER



FOR 2

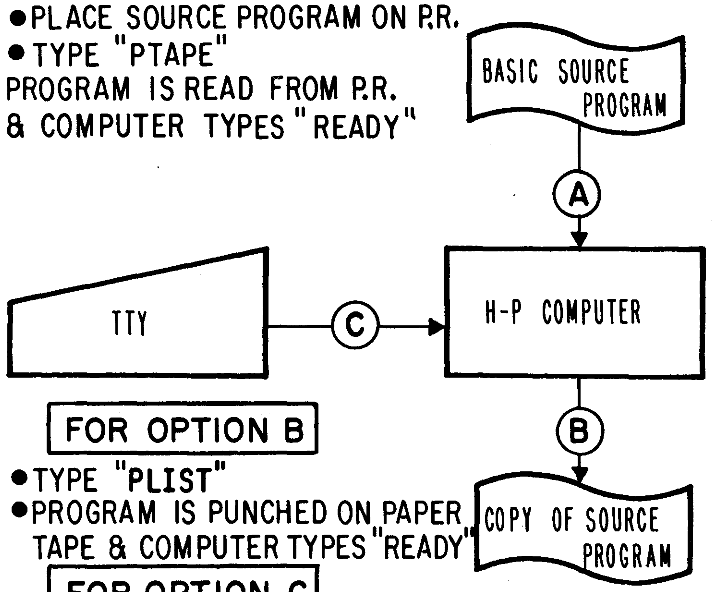
- LOAD ADDRESS 000100
- PRESS PRESET/ PRESS RUN
- "READY" ON TTY

EXECUTION OPTIONS

FOR OPTION A

- PLACE SOURCE PROGRAM ON P.R.
- TYPE "PTAPE"

PROGRAM IS READ FROM P.R. & COMPUTER TYPES "READY"



FOR OPTION B

- TYPE "PLIST"
- PROGRAM IS PUNCHED ON PAPER TAPE & COMPUTER TYPES "READY"

FOR OPTION C

- PLACE SOURCE PROGRAM ON TTY RDR
- TYPE "TAPE"
- PROGRAM IS READ FROM TTY & COMPUTER TYPES "READY"

World-Wide Hewlett-Packard Sales & Service

Call your HP Computer Specialist at any of these convenient locations:

UNITED STATES

ALABAMA
Huntsville
Tel: (205) 881-4591

ARIZONA
Scottsdale
Tel: (602) 945-7601
Tucson
Tel: (602) 298-2313

CALIFORNIA
North Hollywood
Tel: (213) 877-1282
Palo Alto
Tel: (415) 327-6500
Sacramento
Tel: (916) 482-1463
San Diego
Tel: (714) 223-8103

COLORADO
Englewood
Tel: (303) 771-3455

CONNECTICUT
East Hartford
Tel: (203) 289-9394
Norwalk
Tel: (203) 853-1251

DELAWARE
Wilmington
Tel: (302) 655-6161

FLORIDA
Miami Shores
Tel: (305) 754-4565
Orlando
Tel: (305) 841-3970
St. Petersburg
Tel: (813) 391-0211

GEORGIA
Atlanta
Tel: (404) 436-6181

ILLINOIS
Skokie
Tel: (312) 677-0400

INDIANA
Indianapolis
Tel: (317) 546-4891

LOUISIANA
Kenner
Tel: (504) 721-6201

MARYLAND
Baltimore
Tel: (301) 944-5400
Rockville
Tel: (301) 948-6370

MASSACHUSETTS
Lexington
Tel: (617) 861-8960

MICHIGAN
Southfield
Tel: (313) 353-9100

MINNESOTA
St. Paul
Tel: (612) 645-9461

MISSOURI
Kansas City
Tel: (816) 333-2445
St. Louis
Tel: (314) 962-5000

NEW JERSEY
Paramus
Tel: (201) 265-5000
Cherry Hill
Tel: (609) 667-4000

NEW MEXICO
Albuquerque
Tel: (505) 255-5586
Las Cruces
Tel: (505) 526-2485

NEW YORK
Albany
Tel: (518) 869-8462
Endicott
Tel: (607) 754-0050
Poughkeepsie
Tel: (914) 454-7330
Rochester
Tel: (716) 473-9500
Roslyn, Long Island
Tel: (516) 869-8400
Syracuse
Tel: (315) 454-2486

NORTH CAROLINA
High Point
Tel: (919) 882-6873

OHIO
Cleveland
Tel: (216) 884-9209
Dayton
Tel: (513) 298-0351

OKLAHOMA
Oklahoma City
Tel: (405) 848-2801

OREGON
Portland
Tel: (503) 292-9171

PENNSYLVANIA
Monroeville
Tel: (412) 271-0724
West Conshohocken
Tel: (215) 248-1600

TEXAS
Richardson
Tel: (214) 231-6101
Houston
Tel: (713) 667-2407
San Antonio
Tel: (512) 434-4171

UTAH
Salt Lake City
Tel: (801) 486-8166

VIRGINIA
Richmond
Tel: (703) 282-5451

WASHINGTON
Bellevue
Tel: (206) 454-3971

CANADA

ALBERTA
Edmonton
Tel: (403) 482-5561

BRITISH COLUMBIA
Vancouver
Tel: (604) 731-5301

MANITOBA
St. James
Tel: (204) 786-7581

NOVA SCOTIA
Halifax
Tel: (902) 455-0511

ONTARIO
Ottawa
Tel: (613) 722-4223
Toronto
Tel: (416) 249-9196

QUEBEC
Pointe Claire
Tel: (514) 6[illegible]32

CENTRAL AND SOUTH AMERICA

ARGENTINA
Buenos Aires

BRAZIL
Sao Paulo
Rio de Janeiro

MEXICO
Mexico City

VENEZUELA
Caracas

EUROPE

BELGIUM
Brussels

DENMARK
Naerum

FINLAND
Helsinki

FRANCE
Orsay
Lyon

GERMANY
Berlin W
Böblingen
Düsseldorf
Frankfurt
Hamburg
München

ITALY
Milan
Rome

NETHERLANDS
Amsterdam

NORWAY
Haslum

SWEDEN
Mölndal
Solna

SWITZERLAND
Zurich
Meyrin-Geneva

UNITED KINGDOM
Slough, Bucks

AFRICA, ASIA, AUSTRALIA

AUSTRALIA
Melbourne
Sydney
Adelaide
Perth, W. A.

JAPAN
Osaka
Nagoya
Tokyo

NEW ZEALAND
Wellington

SOUTH AFRICA
Cape Town
Johannesburg

FOR EUROPEAN AREAS NOT LISTED, CONTACT:
Hewlett-Packard S.A., Rue du Bois-du-Lan 7,
1217 Meyrin-Geneva, Tel: (022) 41 54 00

FOR OTHER AREAS NOT LISTED, CONTACT:
Hewlett-Packard Export Marketing, 3200 Hillview Ave.,
Palo Alto, California 94304, U.S.A.; Telex: 034-8461,
Cable: HEWPACK Palo Alto.

HEWLETT hp PACKARD

DIGITAL COMPUTERS

22920

This advertisement appeared in the April, 1969, issue of *Datamation* magazine.

HEWLETT hp PACKARD

11000 WOLFE ROAD, CUPERTINO, CALIFORNIA 95014 TELEPHONE 408 257-7000 TWX 910-338-0221
EUROPE: 1217 MEYRIN-GENEVA, SWITZERLAND • CABLE: "HEWPACKSA" TEL. (002) 41.54.00

Printed in U.S.A. 9/68

```
2900 NEXT I
3000 LET V[1]=L[1]*X
3100 LET V[2]=(L[3]-L[1])*X/2
3200 LET V[M-1]=(L[M]-L[M-2])*X/2
3300 LET V[M]=(L[M]-L[M-1])*X
3400 FOR I=1 TO M
3500 LET L[I]=V[I]
3600 NEXT I
3800 LET M=(M+1)/2
3900 LET V[2*M]=0
4000 FOR K=1 TO M
4100 PRINT (K-1),V[K],(K+M-1),V[K+M]
4900 NEXT K
5000 LET C=C+1
5100 IF C=1 THEN 6000
5200 IF C=2 THEN 7000
5900 GOTO 9999
6000 PRINT "ANGLE","ACCELERATION","ANGLE"
6100 GOTO 1990
7000 PRINT "ANGLE","PULSE","ANGLE","PULSE
7100 GOTO 1990
9000 DATA 0,2.68221E-07,8.52943E-06,4.329
9001 DATA 3.32433E-04,6.86967E-04,1.26852
9002 DATA 5.21470E-03,7.59392E-03,1.06916
9003 DATA 2.55621E-02,3.28269E-02,4.14800
9004 DATA 7.72076E-02,9.28516E-02,.110598
9005 DATA .205206,.235357,.268313,.30416,
9006 DATA .479037,.534128,.592369,.65368,
9007 DATA .927508,1.00244,1.07965,1.15895
9008 DATA 1.49308,1.57968,1.66713,1.75515
9009 DATA 2.10792,2.19509,2.28136,2.36651
9010 DATA 2.69149,2.71878,2.78304,2.84613
9011 DATA 3.08507,3.14104,3.19535,3.24793
9012 DATA 3.43948,3.48236,3.5231,3.56167,
9013 DATA 3.69309,3.72001,3.74448,3.76646
9014 DATA 3.829,3.83818,3.84474,3.84868,3
```

```
9000 DATA 0,2.68221E-07,8.52943E-06,4.329
9001 DATA 3.32433E-04,6.86967E-04,1.26852
9002 DATA 5.21470E-03,7.59392E-03,1.06916
9003 DATA 2.55621E-02,3.28269E-02,4.14800
9004 DATA 7.72076E-02,9.28516E-02,.110598
9005 DATA .205206,.235357,.268313,.30416,
9006 DATA .479037,.534128,.592369,.65368,
9007 DATA .927508,1.00244,1.07965,1.15895
9008 DATA 1.49308,1.57968,1.66713,1.75515
9009 DATA 2.10792,2.19509,2.28136,2.36651
9010 DATA 2.69149,2.71878,2.78304,2.84613
9011 DATA 3.08507,3.14104,3.19535,3.24793
9012 DATA 3.43948,3.48236,3.5231,3.56167,
9013 DATA 3.69309,3.72001,3.74448,3.766
```