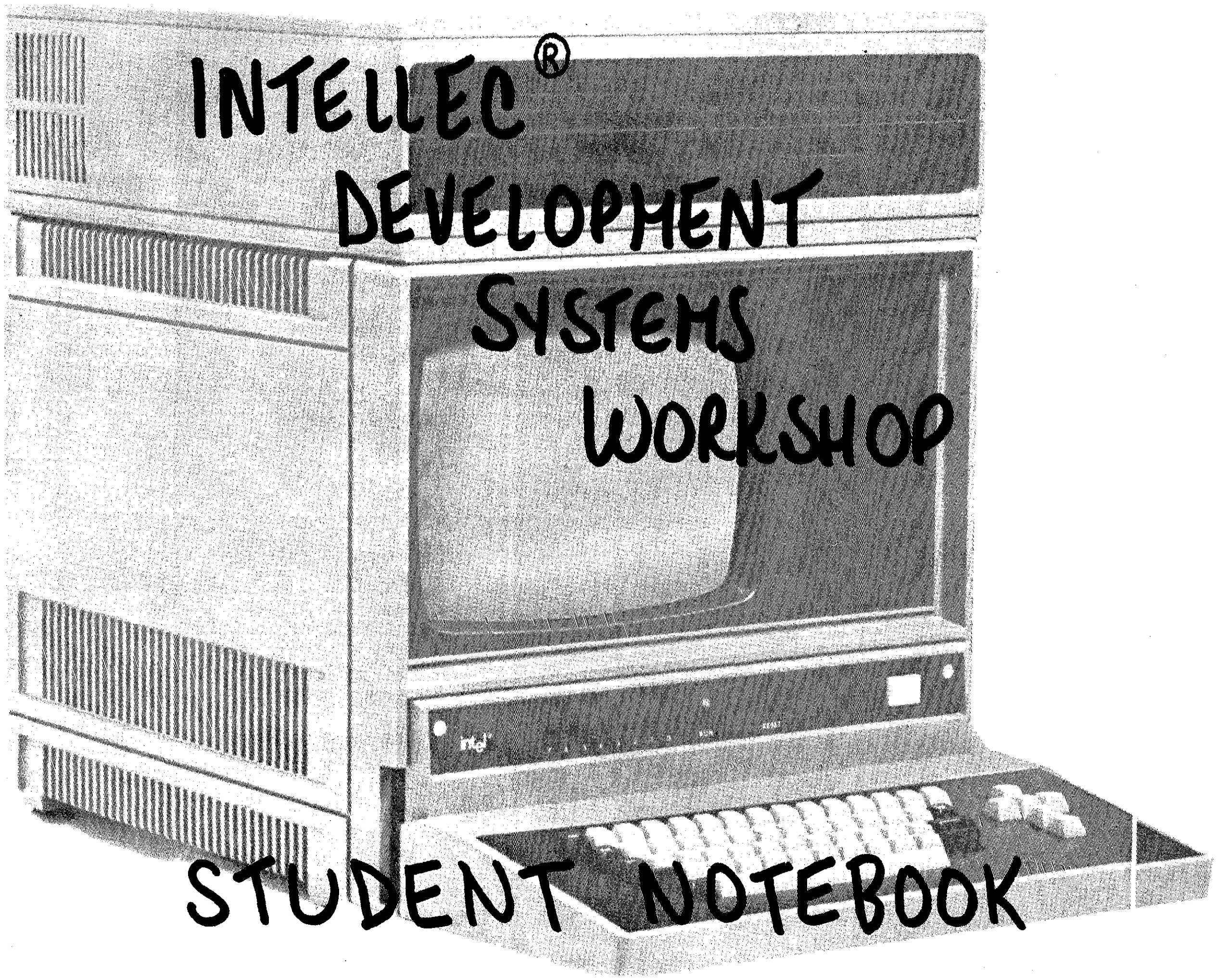
INTELLEC® DEVELOPMENT SYSTEMS WORKSHOP



STUDENT NOTEBOOK

The information in this document is subject to change without notice.

Intel Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Intel Corporation assumes no responsibility for any errors that may appear in this document. Intel Corporation makes no commitment to update nor to keep current the information contained in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Intel Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Intel.

No part of this document may be copied or reproduced in any form or by any means without the prior written consent of Intel Corporation.

The following are trademarks of Intel Corporation and may be used only to describe Intel products:

ICE–30
ICE–48
ICE–80
ICE–85
INSITE
INTEL
INTELLEC
LIBRARY MANAGER
MCS
MEGACHASSIS
MICROMAP
MULTIBUS
PROMPT
UPI

Intel Corporation, 3065 Bowers Avenue, Santa Clara, California 95051

PREFACE

THIS STUDENT STUDY GUIDE IS INTENDED FOR PARTICIPANTS IN THE INTELLEC DEVELOPMENT SYSTEMS OPERATIONS WORKSHOP.

TABLE OF CONTENTS

CLASS SCHEDULE

DAY 1

CHAPTER 1: SYSTEM OVERVIEW AND SETUP

CHAPTER 2: INTRODUCTION TO ISIS

DAY 2

CHAPTER 3: INTRODUCTION TO FORTRAN-80

CHAPTER 4: ICE-85

DAY 3

CHAPTER 5: MAINTENANCE

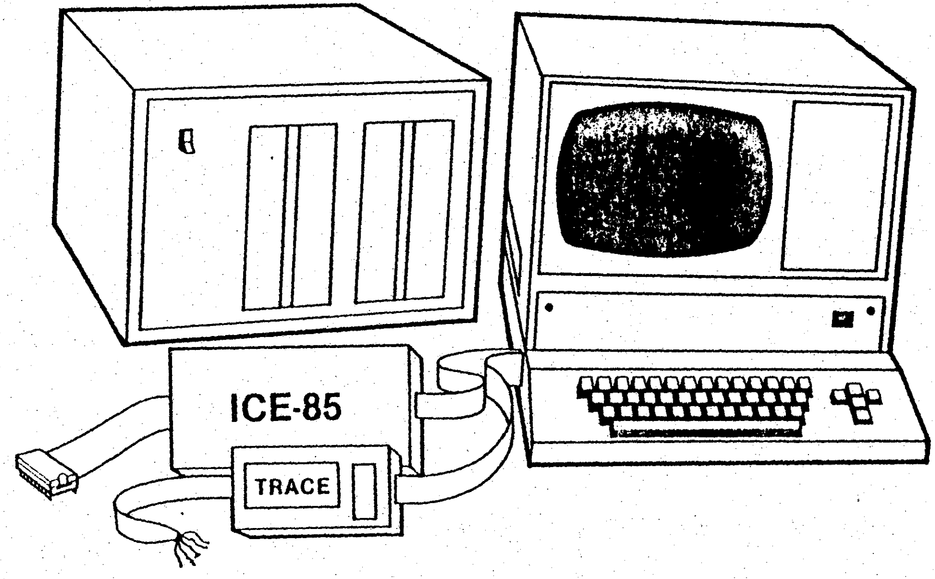
CHAPTER 6: PROGRAMMING AIDS

CHAPTER 7: PROM PROGRAMMING

CHAPTER 1

An Overview of the INTELLEC DEVELOPMENT SYSTEM

INTELLEC DEVELOPMENT SYSTEM



WHY HAVE AN INTELLEC DEVELOPMENT SYSTEM

- SOFTWARE DEVELOPMENT FOR AN 8080, 8048, 8086, etc., BASED PRODUCT

- HARDWARE DEVELOPMENT FOR ALL OF THE SAME

How does the

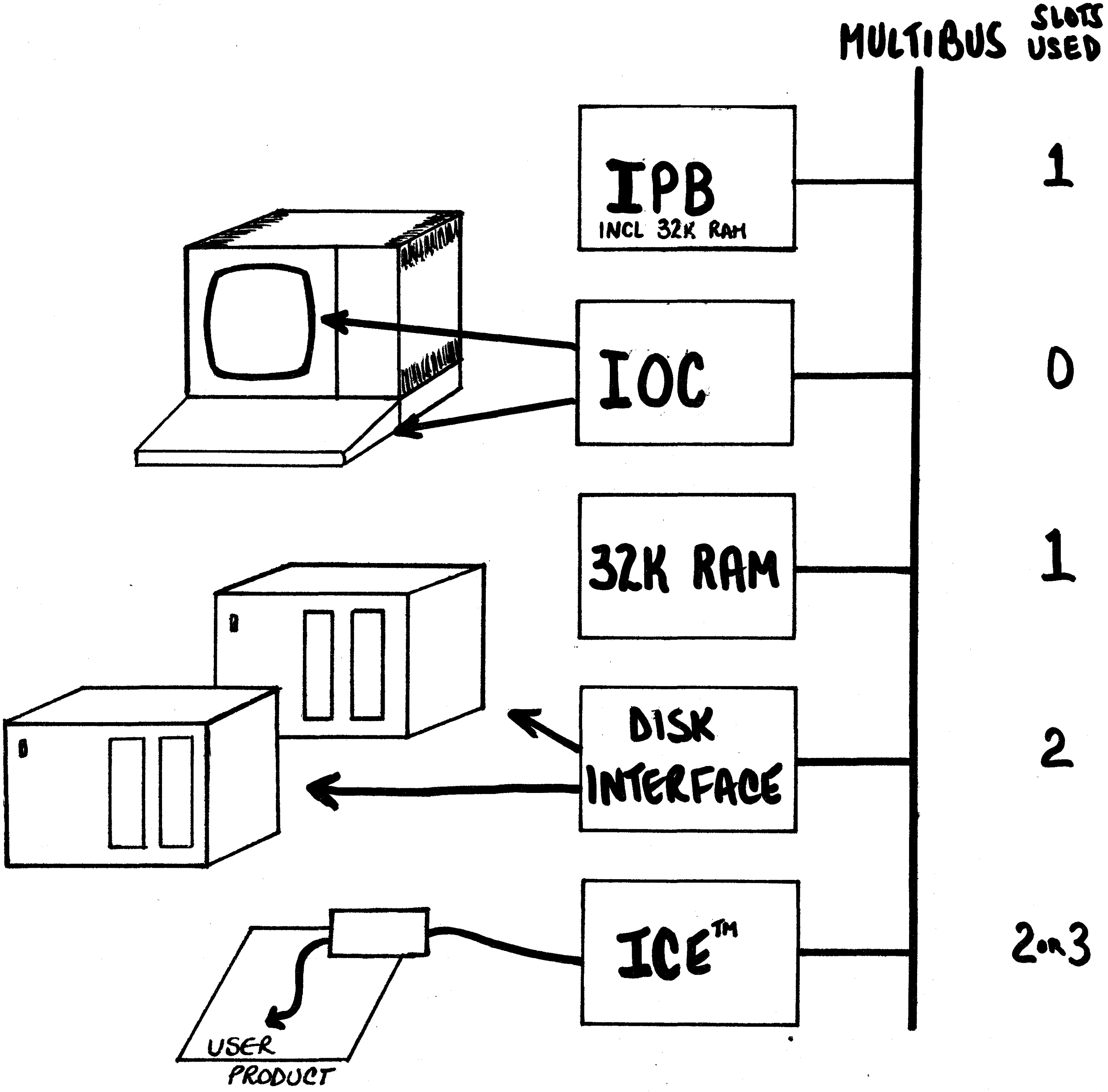
INTELLEC

DEVELOPMENT SYSTEM

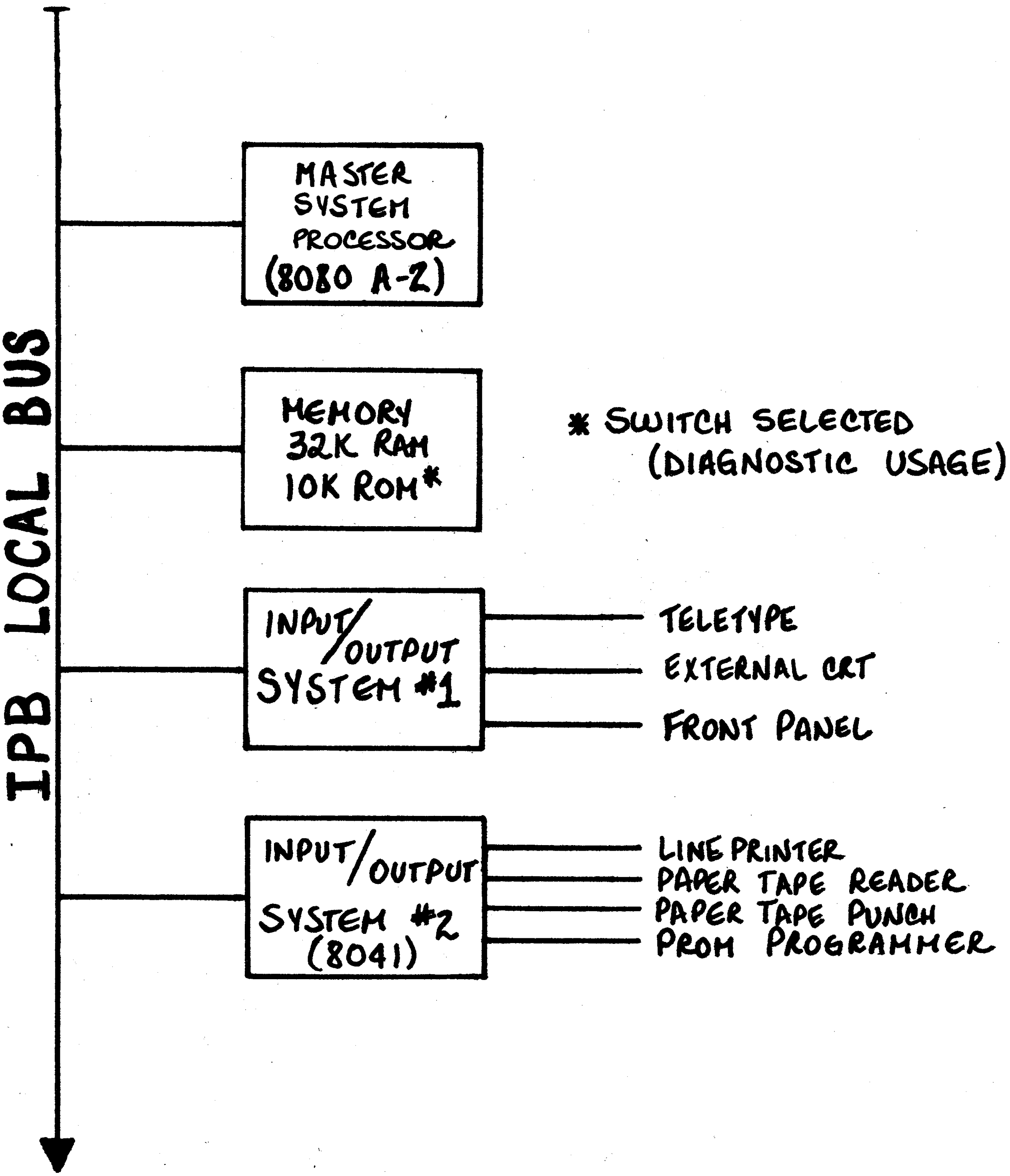
fit into

the development process?

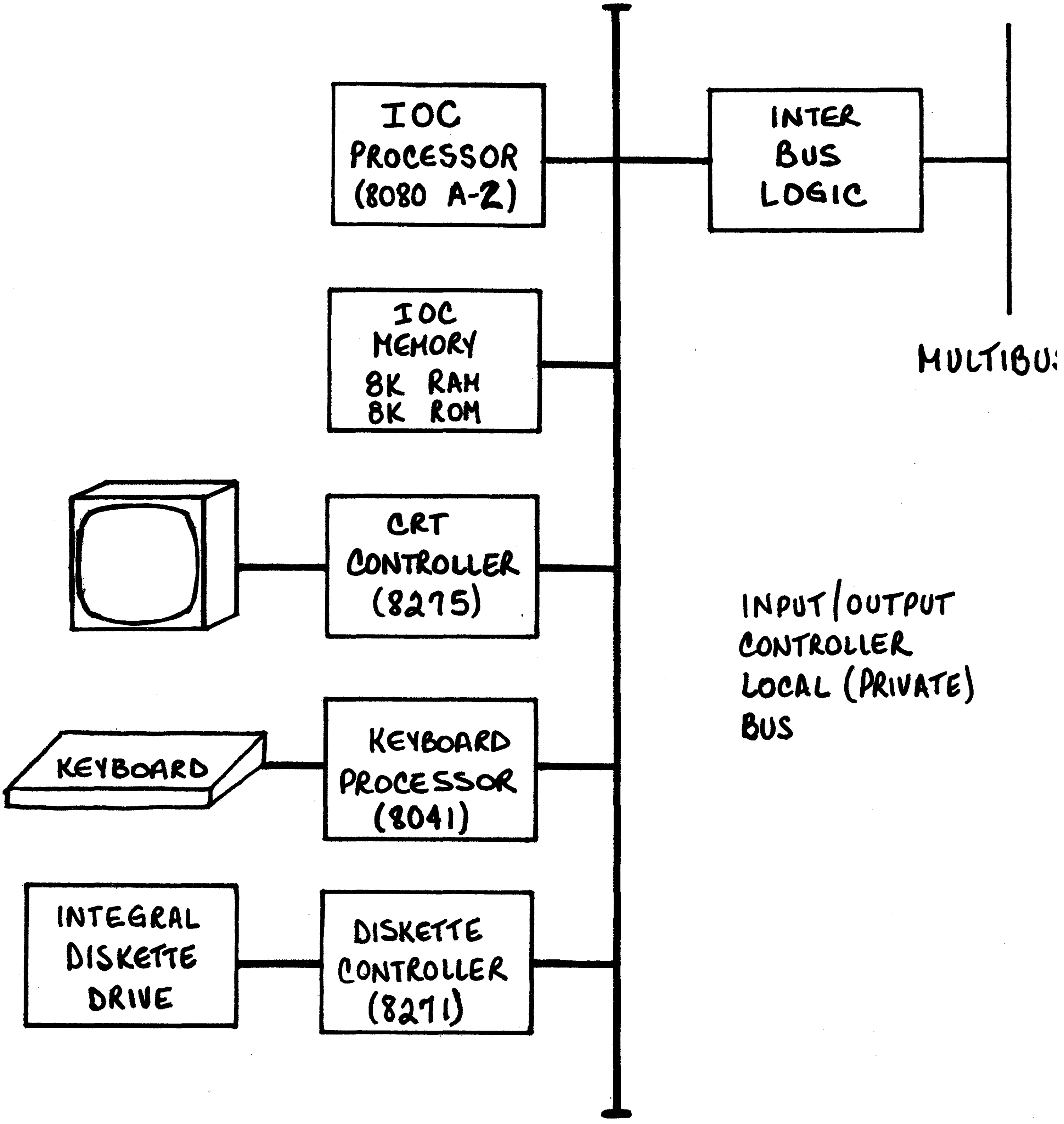
BLOCK DIAGRAM OF SYSTEM HARDWARE



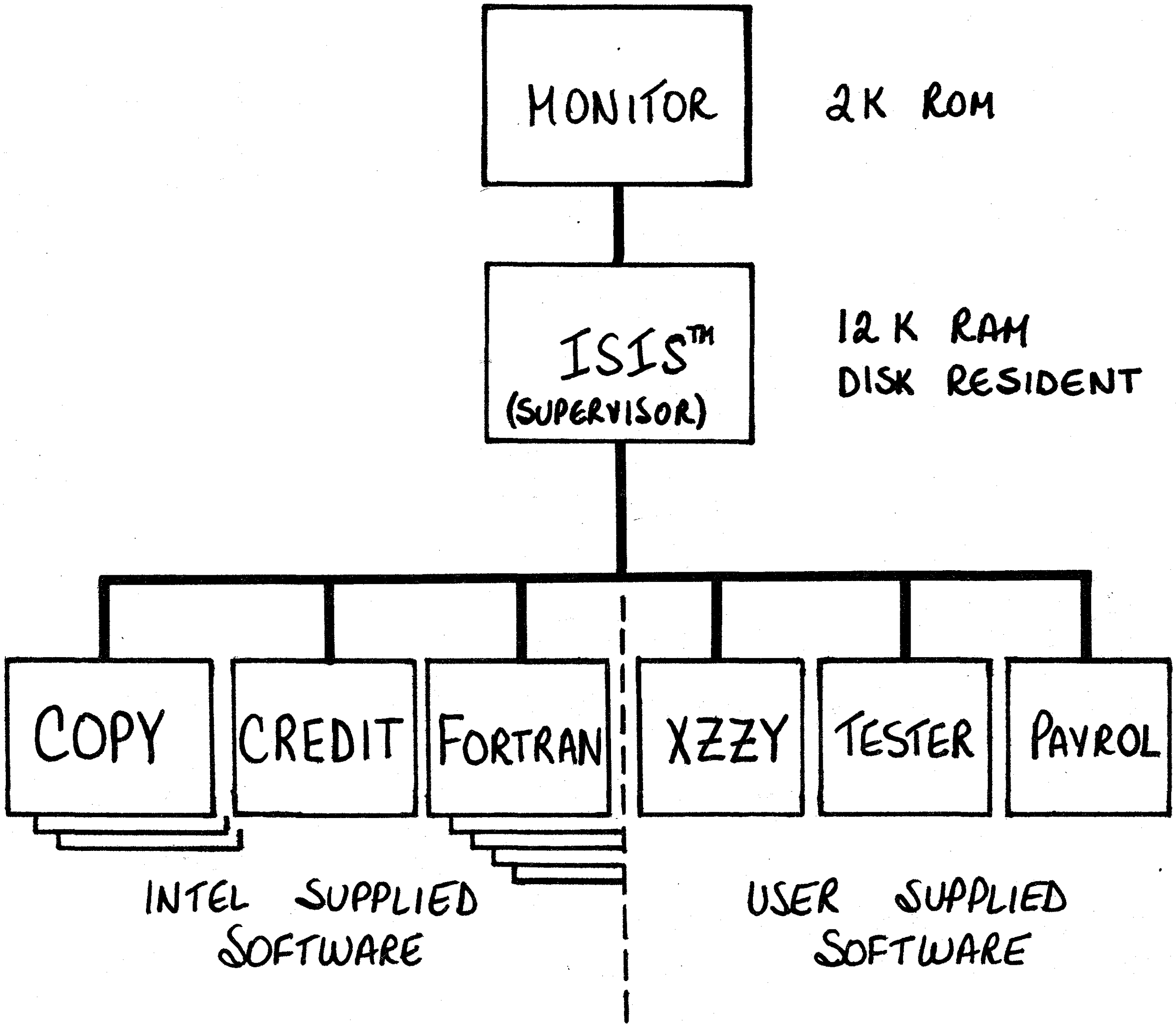
BLOCK DIAGRAM
OF
INTEGRATED PROCESSOR BOARD



BLOCK DIAGRAM OF INPUT/OUTPUT CONTROLLER



SOFTWARE OVERVIEW



① MONITOR ALWAYS PRESENT

② PROGRAMS CAN CALL FOR SERVICES OF LOWER LEVEL PROGRAMS; ie: TESTER CAN USE FACILITIES OF ISIS OR MONITOR

SOFTWARE FUNCTIONS

MONITOR— LOWEST LEVEL
LIMITED DEBUGGING
MEMORY AND REGISTER DISPLAY & CHANGE
LOADS ISIS WHEN SYSTEM DISK IS PRESENT

ISIS— DISK PROGRAM LOADING SERVICES (READ, WRITE, ETC.)

OTHER PROGRAMS—

- INTEL SUPPLIED (A SAMPLE)

  COPY - COPY A BLOCK OF DATA
  CREDIT - TEXT INSERT & EDIT
  FORTRAN - TRANSLATION OF SOURCE TO OBJECT

- USER WRITTEN (?)

SYSTEM SETUP

1. UNPACK AND CHECK FOR SHIPPING DAMAGE (SAVE ALL PACKING AND INFO UNTIL THE SYSTEM IS FULLY CHECKED OUT!!) REMEMBER TO FILL OUT REGISTRATION CARDS!!

2. AFTER READING INSTALLATION MANUAL, SET UP AND PLUG IN PROCESSOR BOX

3. POWER UP PROCESSOR BOX

4. CHECK OUT MONITOR FUNCTIONS (READ AND WRITE ALL AVAILABLE MEMORY (Z$))

5. PLUG IN DISK BOX

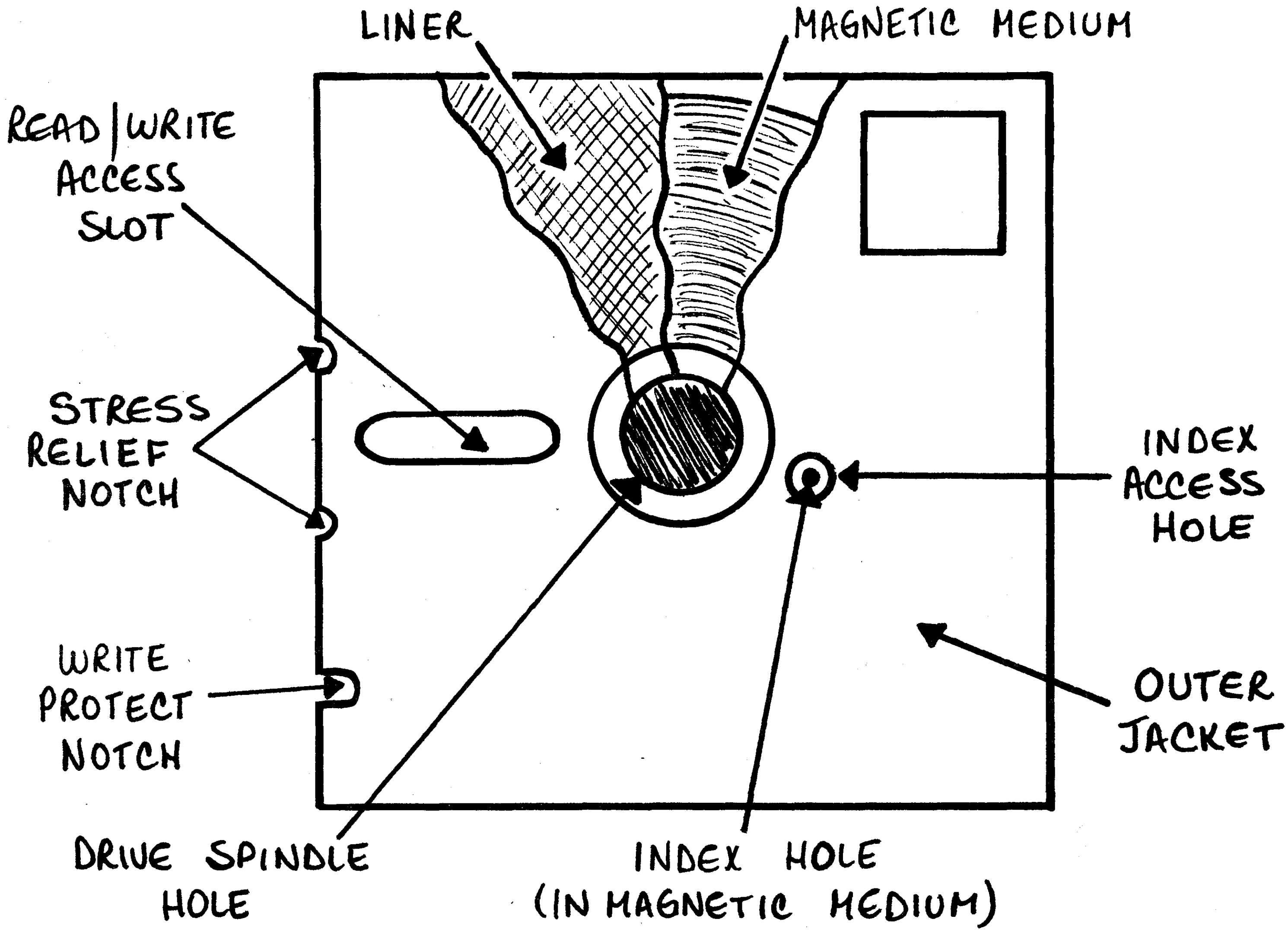
6. POWER UP DISK BOX AND INSERT CONFIDENCE TEST DISKETTE

7. RESET SYSTEM TO BRING TEST ONLINE (FOLLOW DIRECTIONS OF TEST)

8. REMOVE TEST DISKETTE AND INSERT ISIS DISKETTE

9. RESET SYSTEM AND ENJOY

THE DISKETTE



PARAMETER	SINGLE DENSITY	DOUBLE DENSITY
#TRACKS	77	77
#SECTORS PER TRACK	26	52
# BYTES PER SECTOR	128	128
TOTAL # SECTORS	2002	4004
TOTAL # BYTES	256,256	512,512

DISKETTE CARE

DO's ☺

1. KEEP IT IN THE JACKET WHEN NOT IN USE.

2. FILE IT IN A SHIELDED ENVIRONMENT WHEN NOT IN USE (A METAL DESK DRAWER WITH OTHER DISKETTES, BUT NO HARMFUL THINGS).

3. PERIODICALLY CHECK THE DATA ON VALUABLE, FREQUENTLY USED DISKETTES.

4. USE DISKETTES AT ROOM TEMPERATURE ONLY.

DISKETTE CARE

DON'TS ☹

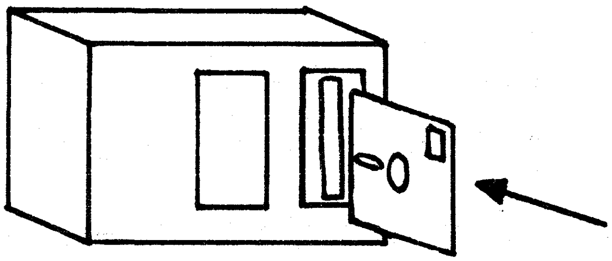
1. NEVER BEND OR FOLD A DISKETTE.
2. DO NOT USE RUBBER BANDS OR PAPER CLIPS ON DISKETTES.
3. DO NOT TOUCH RECORDING SURFACE.
4. DO NOT SMOKE, EAT, OR DRINK WHILE HANDLING DISKETTES.
5. DO NOT EXPOSE DISKETTE TO ANY EXCESSIVE HEAT; JACKET WILL WARP.
6. DO NOT EXPOSE DISKETTE TO ANY MAGNETIC FIELD (THIS IS TOUGH SINCE MAGNETIC FIELDS ARE INVISIBLE!).
7. DO NOT WRITE ON THE JACKET!
8. DO NOT TREAT A DISKETTE LIKE A 45-RPM RECORD (FINGER THRU THE SPINDLE HOLE).

SYSTEM LOAD

1. TURN ON SYSTEM BY PRESSING POWER SWITCH.

2. TURN ON OUTBOARD DISKETTE DRIVE (IF ANY) WITH ROCKER SWITCH.

3. INSERT SYSTEM DISKETTE INTO DRIVE Ø; LABEL TO THE LEFT, READ|WRITE SLOT HORIZONTAL AS SHOWN. CLOSE DRIVE DOOR.



4. PRESS RESET.

SYSTEM HAS FINISHED LOADING WHEN

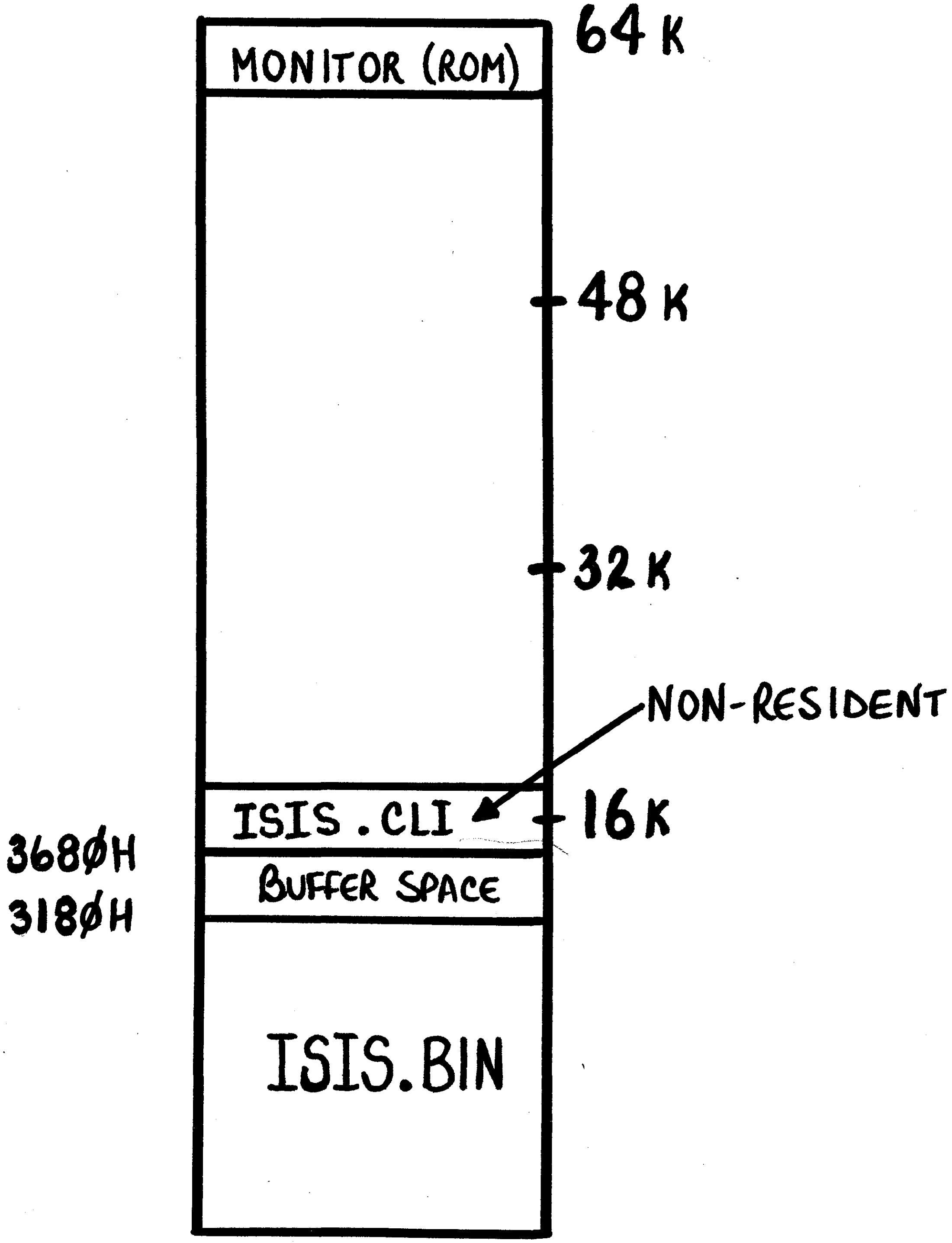
ISIS V~~3.4~~

_

4.0

APPEARS ON THE SCREEN.

SYSTEM MEMORY
WITH
ISIS LOADED



TYPING ON A DEVELOPMENT SYSTEM

SPECIAL CHARACTERS EASE TYPING TASK:

RUBOUT – (RO) DELETES LAST CHARACTER AND ECHOS IT ON THE SCREEN

I.E.

TYPED: JACQUE(RO)(RO)(RO)

SCREEN: JACQUE E U Q

AT THIS POINT MEMORY CONTAINS JAC

↑X – (CONTROL–X) DELETES ENTIRE LINE

I.E.

TYPED: JACK AND JILL WENT (↑X)

SCREEN: JACK AND JILL WENT #

≡ ← BLINKING CURSOR

↑R – (CONTROL–R) REVIEW CURRENT CONTENTS OF LINE BEING TYPED (CORRECTIONS ALREADY MADE)

I.E.

(RO) (RO)(RO)

TYPED: JACK AND JILF F L NE E N WEN(↑R

SCREEN: JACK AND JILFFL NE EN WENT

JACK AND JILL WENT≡ BLINKING CURSOR

OUTPUT CONTROL

TWO MORE SPECIAL CHARACTERS TO CONTROL SYSTEM <u>OUTPUT</u> TO SCREEN

↑S (CONTROL-S) STOP OUTPUT

↑Q (CONTROL-Q) RESUME OUTPUT

HOW TO RUN A PROGRAM

WHEN SYSTEM IS READY (I.E. THE PROMPTING "–" IS SHOWING), TYPE THE NAME OF THE FILE THE PROGRAM IS STORED IN.

Example:

–DIR (CARRIAGE RETURN)

WILL LOAD AND RUN THE DIRECTORY LISTING PROGRAM.

–:F1: JACK

WILL LOAD AND RUN THE PROGRAM STORED UNDER THE NAME JACK (ON THE DISKETTE IN DRIVE 1).

PROGRAMS CAN BE LOADED AND RUN FROM ANY DISKETTE.

A FILE

DEFINITION:

A FILE IS A COLLECTION OF DATA (BYTES).
IT RESIDES ON PAPER TAPE, DISKETTE, etc.
IT CAN BE COPIED FROM ONE PLACE TO ANOTHER.
WHAT THE DATA MEANS DEPENDS ON YOU.
IT MAY BE DATA FROM A SERIES OF TESTS, A SOURCE PROGRAM (MAN READABLE), AN OBJECT PROGRAM (MACHINE READABLE); EVEN A MEMO TO THE BOSS.

FILE NAMES

IN THE INTELLEC SYSTEM, FILES HAVE NAMES. ALL FILES HAVE THE NAME OF THE DEVICE ON WHICH THEY ARE STORED OR WHERE THEY ARE GOING AS PART OF THEIR NAME. IN SOME CASES, THIS IS ALL THAT IS NECESSARY. FOR EXAMPLE;

:LP:	LINE PRINTER (A DESTINATION ONLY)
:CI:	CONSOLE (KEYBOARD) INPUT (A SOURCE ONLY)
:CO:	CONSOLE (SCREEN) OUTPUT (A DESTINATION ONLY)

(CONT.)

FILE NAMES

THE DISKETTE FILE NAME HAS 3 PARTS:

- DEVICE (DRIVE) – :FØ:, :F1:, ..., :F5:
- ROOT – 1 to 6 CHARACTERS (A-Z, Ø-9)
  A, JACK, Ø123, A3BZ, etc.
- EXTENSION – 1 to 3 CHARACTERS (A-Z, Ø-9)

OK ☺	ILLEGAL ☹
:F1: JACK.PLM	:F6: JACK.PLM
:F4: J32. FOR	:F4: J32.FOR
EDITOR	EDI#TO
SAMMY. ASM	:F4: PRESENTR

NOTES: ① IF NO DRIVE # IS PRESENT, SYSTEM ASSUMES :FØ:

② EXTENSION IS OPTIONAL (BUT RECOMMENDED)

(CONT.)

FILE NAMES

SOMETIMES WE WANT TO WORK WITH GROUPS OF FILES WHICH HAVE SIMILAR NAMES. FOR INSTANCE:

```
:F1: JACK.FOR
:F1: JACK.LST
:F1: JACK.OBJ
:F1: JACK.LNK
```

ARE ALL RELATED FILES WITH THE COMMON ROOT JACK. WE CAN TREAT THEM AS A GROUP WITH

```
:F1: JACK.*
```

*.LST REFERS TO

```
JACK.LST
JILL.LST
PROJ1.LST
```

ISIS SYSTEM DISKETTE FILES

ISIS.DIR - The directory of the diskette

ISIS.MAP - The map of occupied space on this diskette

ISIS.TØ - The ISIS bootstrap program

ISIS.LAB - The label of the diskette

ISIS.CLI - The ISIS command line interpreter

ISIS.BIN - ISIS herself

COPY - An Intel supplied program for copying files

DELETE - An Intel supplied program to delete files

DIR - An Intel supplied program to print the directory of a diskette

CREDIT - The text editor. (Supplied by Intel of course!)

IDISK - An Intel supplied program to initialize diskettes

FORT80 - The Intel FORTRAN-77 translator

SAMPLE USER DISKETTE FILES

ISIS.DIR - SAME AS SYSTEM DISKETTE
ISIS.MAP - " " " "
ISIS.TØ - " " " "
ISIS.LAB - " " " "

DATA.ZZZ - TEST DATA FOR A PROCESS CONTROL PROGRAM

TEST.PLM - SOURCE CODE FOR PROCESS CONTROL

TEST - EXECUTABLE CODE FOR PROCESS CONTROL

PAYROL.FOR - FORTRAN SOURCE CODE FOR PAYROLL PROGRAM

102379.LET - LETTER I WROTE ON 10/23/79

NOTE: THERE IS NO ISIS.CLI OR ISIS.BIN ON A USER DISKETTE!

INTEL SUPPLIED PROGRAMS

- BATCH PROCESSING

  SUBMIT

- DISKETTE INITIALIZATION

  IDISK

- CROSS REFERENCE LISTING GENERATION (INTER-PROGRAM)

  ASXREF

INTEL SUPPLIED PROGRAMS

-DIR-

WHAT- THE DIR PROGRAM ALLOWS US TO READ THE DIRECTORY OF A DISKETTE TO FIND FILES.

DIR [drive#] [I] [To filename]

EXAMPLES:

DIR	OBTAIN A LISTING OF "VISIBLE" FILES ON DRIVE Ø LISTING ON CRT
DIR 1	OBTAIN A LISTING OF "VISIBLE FILES" ON CRT FOR DRIVE 1
DIR 2 I	OBTAIN A LISTING OF ALL FILES ON DRIVE 2 ON CRT
DIR 1 TO :LP:	OBTAIN A LISTING OF "VISIBLE" FILES ON DRIVE 1 AND PRINT IT ON THE LINE PRINTER

INTEL SUPPLIED PROGRAMS

-ATTRIB-

WHAT- THE ATTRIB PROGRAM WILL CHANGE THE ATTRIBUTES (WRITE PROTECT, INVISIBILITY, etc.) OF A FILE.

```
ATTRIB filename [OPTIONS]
```

OPTIONS ARE

W	WRITE PROTECT
S	SYSTEM PROGRAM
I	INVISIBLE
F	FORMAT

W1 = SET WRITE PROTECT

WØ = TURN OFF WRITE PROTECT

EXAMPLES:

```
ATTRIB :F1:POEM.DAT W1 S1
```

INTEL SUPPLIED PROGRAMS

-IDISK-

WHAT - THE IDISK PROGRAM INITIALIZES A DISKETTE SO THAT DATA CAN BE STORED ON IT BY OTHER PROGRAMS.

WHY - THE BLANK DISKETTE DOES NOT HAVE THE TRACK AND SECTOR INFORMATION NEEDED BY THE OPERATING SYSTEM. (STREETS & SEWERS)

```
IDISK diskette.label [S]
```

EXAMPLES:

```
IDISK :F1:MAR12.79   S   (a system diskette)
IDISK :F1:GAMES.BAS      (a user diskette)
```

INTEL SUPPLIED PROGRAMS

-COPY-

WHAT- THE COPY PROGRAM MAKES A COPY OF A FILE ON A SECOND FILE

COPY filename TO filename [OPTIONS]

EXAMPLES:

```
COPY        JACK TO JILL
COPY        :F3: SAM.* TO :F1:
COPY        :F2: DATA TO :LP:
COPY        :F1: MEMO TO :CO:
COPY        FINAL.* TO FINAL.* P
```

INTEL SUPPLIED PROGRAMS

-DELETE-

WHAT- THE DELETE PROGRAM WILL DELETE A FILE FROM A DISKETTE BY REMOVING ITS NAME FROM THE DIRECTORY. (NOTE: THE DATA IS NOT ACTUALLY REMOVED, BUT THE SPACE IT OCCUPIES IS MARKED AS VACANT SO IT CAN BE REUSED)

DELETE filename [OPTIONS]

EXAMPLES:

```
DELETE     TEOTL. ASM
DELETE     SAM1. *
DELETE     *.*  Q
```

INTEL SUPPLIED PROGRAMS

- RENAME -

WHAT - THE RENAME PROGRAM WILL RENAME A NON-WRITE PROTECTED FILE ON A DISKETTE.

RENAME filename1 TO filename2

EXAMPLES:

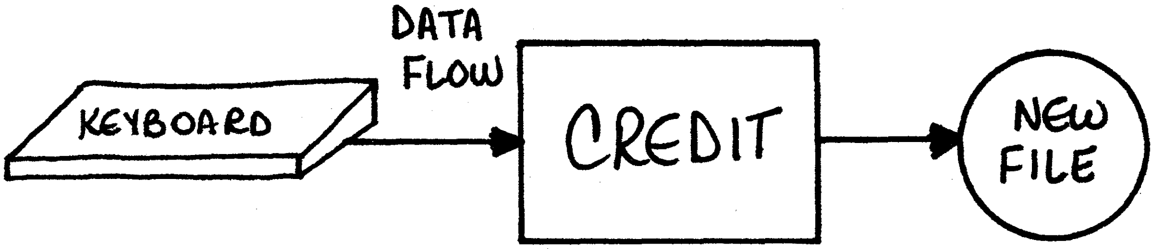
```
RENAME    TOM TO JERRY
RENAME    :F1: PROB1.LST  TO  :F1: GRAD.LST
RENAME    :F2: JACK.* TO :F2: JILL.*
```

NOTE: BOTH FILES MUST BE ON THE SAME DISKETTE

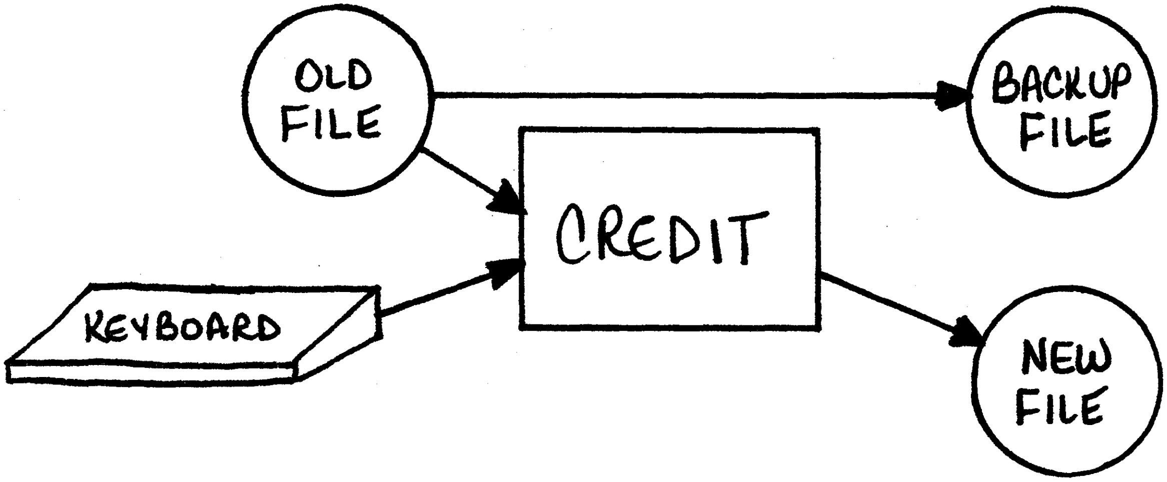
INTEL SUPPLIED PROGRAMS

-CREDIT-

PURPOSE: CREDIT IS A PROGRAM THAT ALLOWS THE USER TO EASILY ENTER ANY TEXTUAL DATA INTO A FILE ON THE SYSTEM. IT ALSO ALLOWS THE USER TO EASILY MODIFY ANY TEXTUAL DATA ALREADY IN A FILE ON THE SYSTEM. IT IS, IN SHORT, A TEXT EDITOR.



INITIAL CREATION



EDITING AN OLD FILE

CREDIT

INVOCATION

CREDIT filename1[TO filename2]

WHERE filename1 IS THE NEW FILE TO BE CREATED OR AN EXISTING FILE TO BE UPDATED. filename2 IS THE NAME OF THE NEW FILE IF YOU ARE UPDATING AN OLD FILE.

EXAMPLES

CREDIT	JACK.ASM		(NEW FILE)
CREDIT	JACK.ASM	TO	JACK1.ASM (EDIT OLD FILE AND STORE NEW COPY IN JACK1.ASM)
CREDIT	JACK1.ASM		(EDIT OLD FILE. STORE ORIGINAL IN JACK1. BAK. STORE NEW COPY IN JACK1. ASM)

CREDIT

SCREEN MODE COMMANDS

KEY	EFFECT
CURSOR CONTROLS ↑ ← → ↓	POSITION CURSOR 1 SPACE OR LINE. USE REPEAT KEY FOR MOVEMENT OVER LONG DISTANCES
HOME	CHANGE TO COMMAND MODE
LETTER, NUMBER, OR SPECIAL CHARACTER	REPLACES CHARACTER AT CURRENT CURSOR POSITION

NOTE: THE CURSOR CONTROLS AND HOME KEY WORK <u>ONLY</u> WITH CREDIT!

ESCAPE (ESC) WILL CANCEL ANY COMMAND.

CREDIT

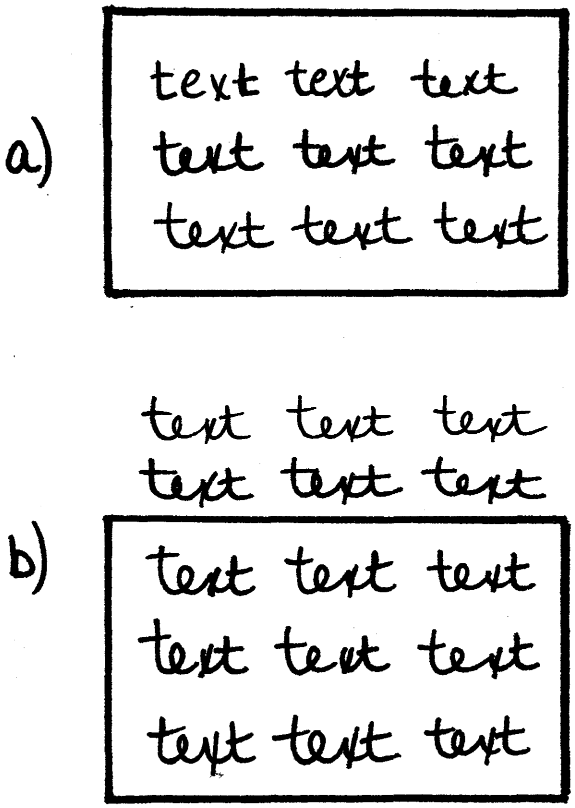
SCREEN MODE COMMANDS

KEY	EFFECT
↑C	INSERT A CHARACTER AT CURRENT CURSOR POSITION. EXAMPLE: ↑CA WOULD INSERT AN A.
↑D	DELETE THE CHARACTER AT THE CURRENT POSITION.
↑A (STRING OF CHARACTERS) ↑A	INSERT A GROUP OF CHARACTERS. (STRING MAY BE ANY LENGTH) IN THIS MODE RUBOUT AND ↑X ARE FUNCTIONAL.
↑Z ↑Z	DELETE A STRING OF CHARACTERS (POSITION CURSOR AT BEGINNING, TYPE ↑Z; POSITION CURSOR AT END, TYPE ↑Z).

CREDIT

Screen Mode Commands

The Screen Moves
As Text is Entered



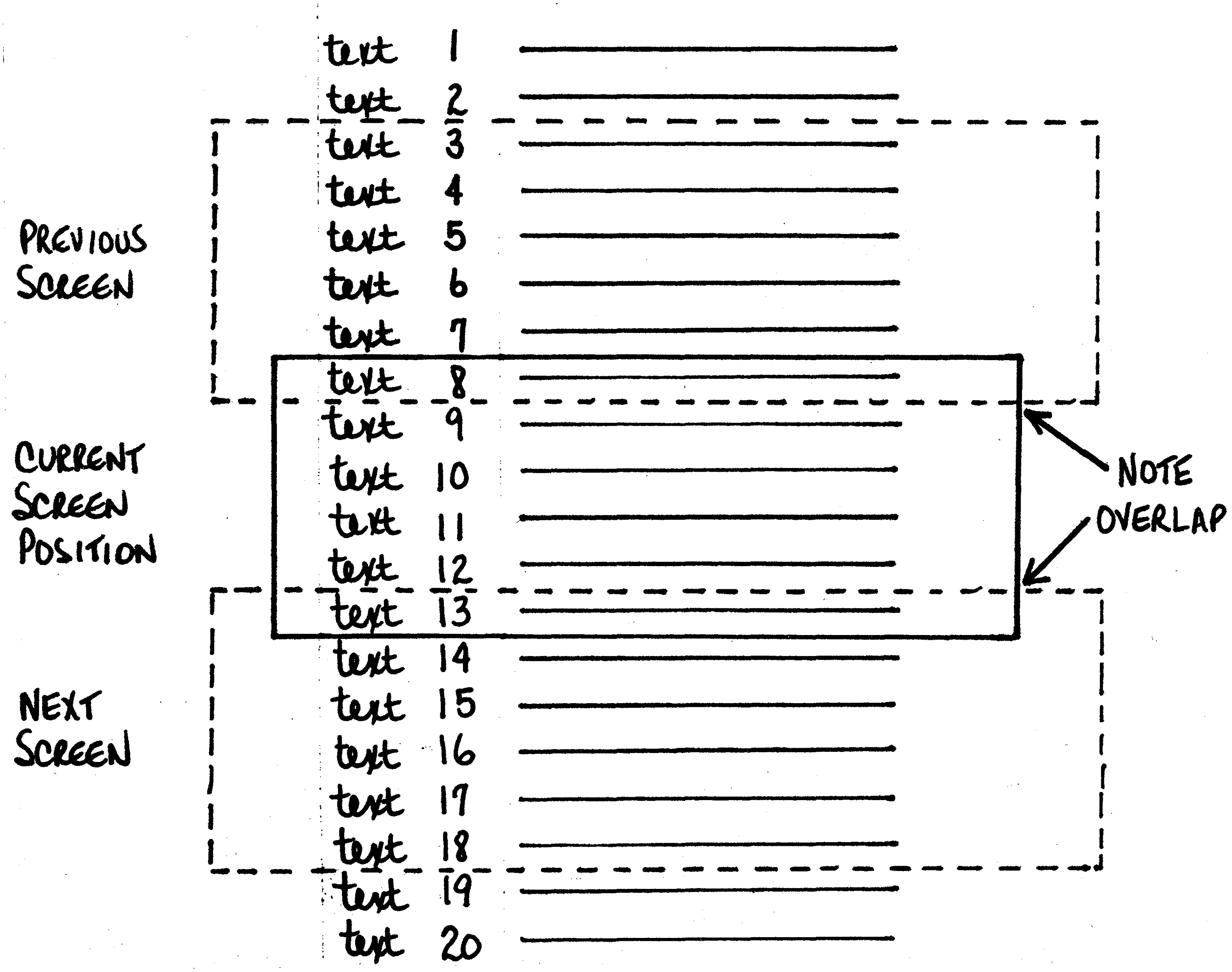
CREDIT

Screen Mode Commands

The Screen Can be Positioned with

↑N (Next Screen)

↑P (Previous Screen)

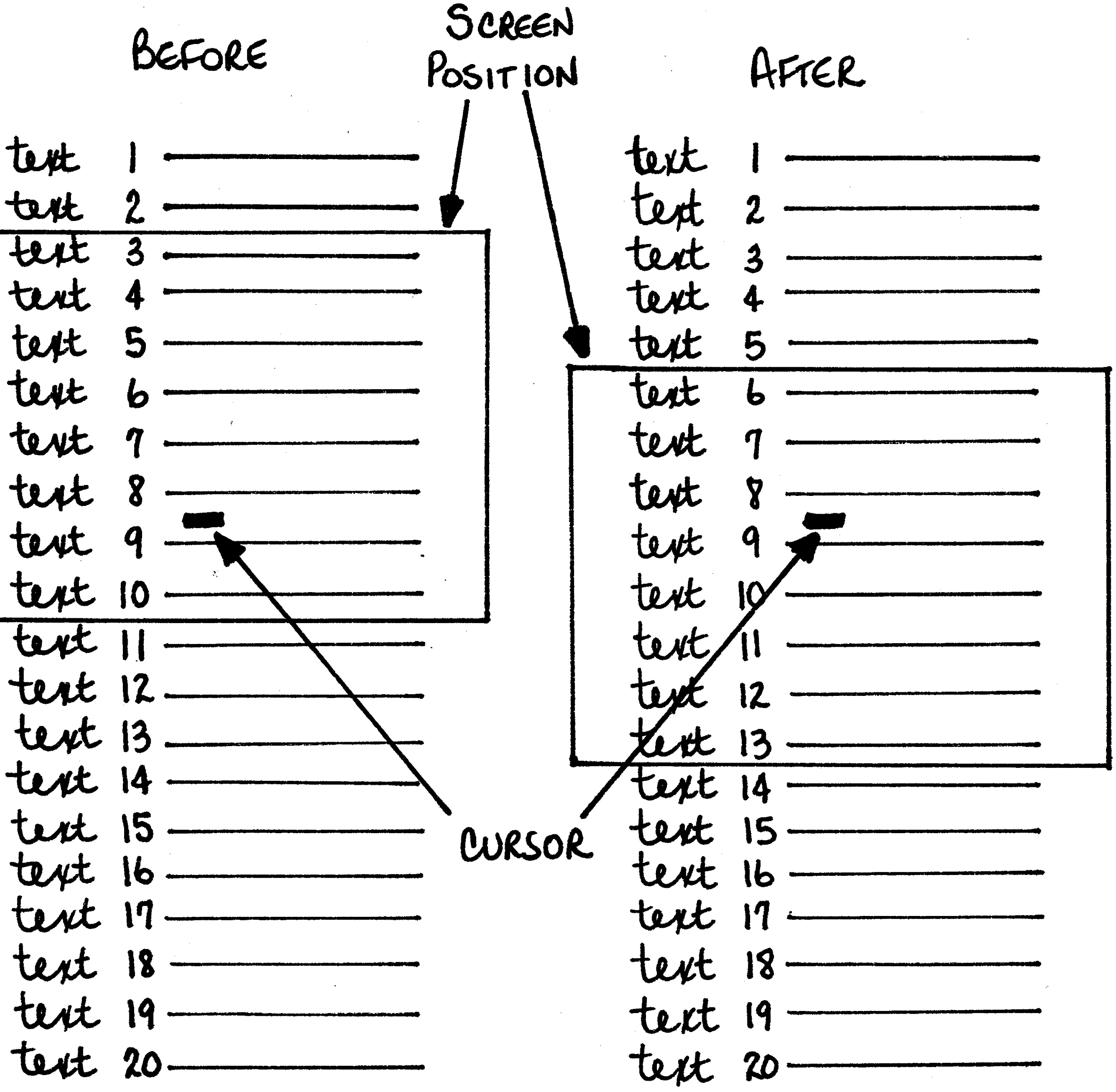


CREDIT

SCREEN MODE COMMANDS

THE SCREEN CAN BE POSITIONED WITH

↑V (SET POSITION OF SCREEN ACCORDING TO CURSOR)



CREDIT

COMMAND MODE COMMANDS

TO GET TO COMMAND MODE: **HOME**

TO GET BACK TO SCREEN MODE: ↑V

HELP COMMAND

H	CREDIT WILL PRINT A MENU OF COMMANDS

CURSOR POSITIONING

L	MOVE BY THE LINE
L = L1	MOVE FORWARD 1 LINE
L3	MOVE FORWARD 3 LINES
L-17	MOVE BACKWARD 17 LINES
JTT	JUMP TO TOP OF FILE
JTE	JUMP TO END OF FILE

CREDIT

COMMAND MODE COMMANDS

TEXT ALTERING
(CAN USE RUBOUT AND ↑X)

I	INSERT STRING
I/string/	WHERE / IS THE STRING DELIMITER (THE FIRST CHARACTER AFTER **I** IS TAKEN AS THE DELIMITER, SO ANY CHARACTER MAY BE USED!).
I/JACK SPRAT COULD EAT NO FAT/	WOULD INSERT THE LINES AT THE CURRENT POSITION.

CREDIT

COMMAND MODE COMMANDS

S SUBSTITUTE STRING

SQ SUBSTITUTE STRING AFTER QUERY

S/STRINGOLD/STRINGNEW/

WHERE STRINGOLD IS THE STRING TO BE REPLACED, AND STRINGNEW IS THE DATA IT WILL BE REPLACED WITH

S/JACK SPRAT/JILL SPRAT/

SQ/NO FAT/MORE FAT/

WILL REPLACE 'NO FAT' AFTER THE LINE CONTAINING IT IS PRINTED ON THE SCREEN, AND THE USER ANSWERS THE QUERY 'YES'

CREDIT

COMMAND MODE COMMANDS

DL	DELETES CURRENT LINE
DL(-n/n)	WHERE 'n' IS THE NUMBER OF LINES TO BE DELETED WRT CURSOR
DL-5	DELETES 5 PREVIOUS LINES
DL30	DELETES NEXT 30 LINES
F	FIND STRING
F/string/	WILL FIND THE FIRST OCCURANCE OF A STRING (FORWARD OR BACKWARD) FROM PRESENT CURSOR POSITION
F/BILLY/TE	SEARCH TO END OF FILE TO FIND 'BILLY'
F/SALLY/TB	SEARCH BACKWARDS TO BEGINNING OF FILE TO FIND 'SALLY'
F/JACK/	SAME AS F/JACK/TE

CREDIT

COMMAND MODE COMMANDS

EX - EXIT EDITOR

EX EXIT EDITOR AND CREATE BACKUP FILE (EITHER .BAK OR NAMED FILE)

EQ EXIT EDITOR AND QUIT. NO BACKUP FILE IS CREATED

CHAPTER 3

An Introduction to FORTRAN-80

WHY A HIGH LEVEL LANGUAGE?

TO COMMUNICATE WITH THE 8080

DIFFICULTY TO PEOPLE	LANGUAGE	DIFFICULTY TO 8080
NO PROBLEM	ENGLISH, SPANISH, GREEK, etc.	(ARE YOU KIDDING?)
UNDERSTANDABLE	FORTRAN, BASIC, PL/M, etc.	(A TRANSLATABLE LANGUAGE)
MORE MYSTERIOUS	ASSEMBLER CODE	(A TRANSLATABLE LANGUAGE)
DIGITAL HIEROGLYPHICS	BINARY MACHINE CODE	(A NATURAL FOR THE 8080)

PEOPLE CAN UNDERSTAND ALL LEVELS, BUT ANYTHING BELOW HIS NATURAL TONGUE IS DIFFICULT. HIGH LEVEL LANGUAGES ARE ONE OF THE SMALLEST STEPS AWAY FROM THE HUMAN SPEECH THAT IS TRANSLATABLE TO THE 8080.

HIGH LEVEL LANGUAGES
(A SAMPLER)

BASIC - INVENTED AT DARTMOUTH; QUICK PROBLEM SOLVING.

FORTRAN - FOR ENGINEERING PROBLEM SOLVING.

PL/M - BLOCK STRUCTURED LANGUAGE. GOOD FOR EXECUTIVE LEVEL PROGRAMS AND PROCESS CONTROL.

COBOL - COMMON BUSINESS ORIENTED LANGUAGE; THE KING OF THE PAYROLL BOYS.

PASCAL - A NEW BLOCK STRUCTURED LANGUAGE.

STEPS TO COMPUTER PROBLEM SOLVING

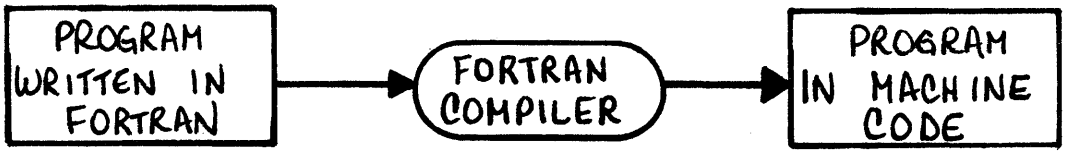
1. DESCRIBE THE PROBLEM COMPLETELY AND CAREFULLY IN YOUR NATIVE TONGUE.

2. DESCRIBE THE SOLUTION COMPLETELY AND CAREFULLY IN YOUR NATIVE TONGUE.

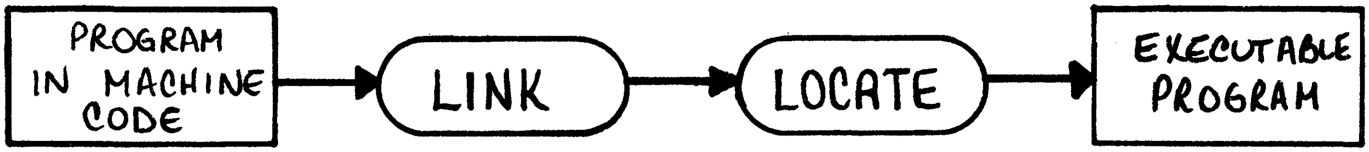
3. TRANSLATE SOLUTION TO FORTRAN OR ANY OTHER SELECTED LANGUAGE.

STEPS TO COMPUTER PROBLEM SOLVING

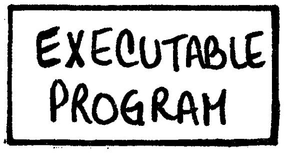
4. TRANSLATE FORTRAN TO MACHINE CODE USING A PROGRAM CALLED THE FORTRAN COMPILER.



5. FINISH PROCESSING MACHINE CODE WITH LINK AND LOCATE PROGRAMS.



6. RUN THE FINISHED PROGRAM.



FORTRAN

- THE LANGUAGE

FORMULA **TRAN**SLATION

ORIGINALLY DEVELOPED TO AID SCIENTISTS
TO PROGRAM EARLY COMPUTERS

IT HAS SINCE BECOME ONE OF THE
"STANDARD" COMPUTER
LANGUAGES THROUGHOUT THE WORLD

A GENERAL OUTLINE

Practically All Computer Languages Have Two Types of Statements:

Executable

and

Non-Executable (or)
Data Description

Executable: Statements which, when translated, direct the computer to ACT.

Ex:
```
A = B + C
IF (X.EQ.Y), THEN
Z = 26
ELSE
Z = 19
```

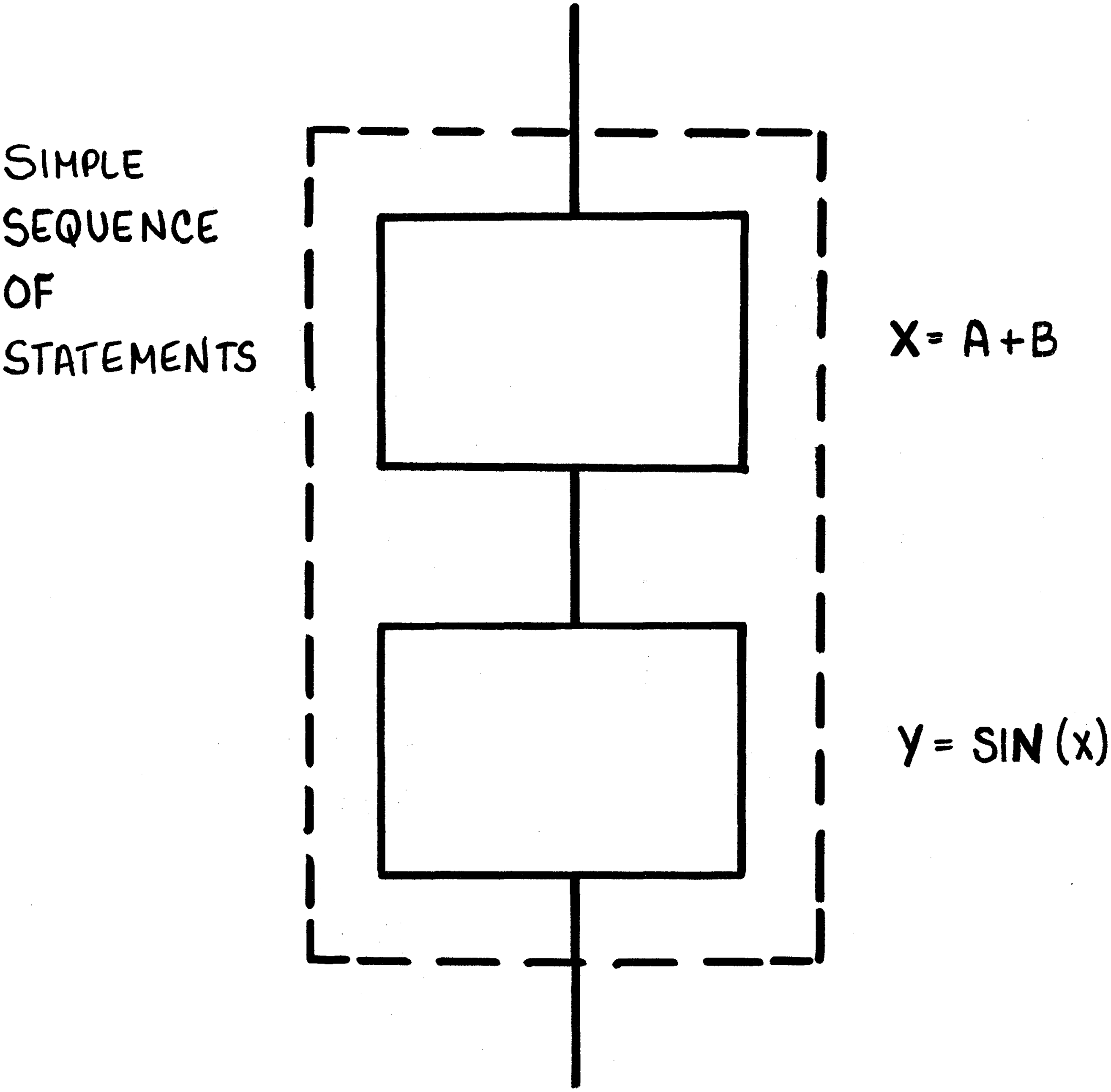
Data Description: Statements which describe the data being processed.

Ex:
```
DIMENSION ANALOG (2Ø)
REAL SAM3, BINT
```

A SIMPLE PROGRAM

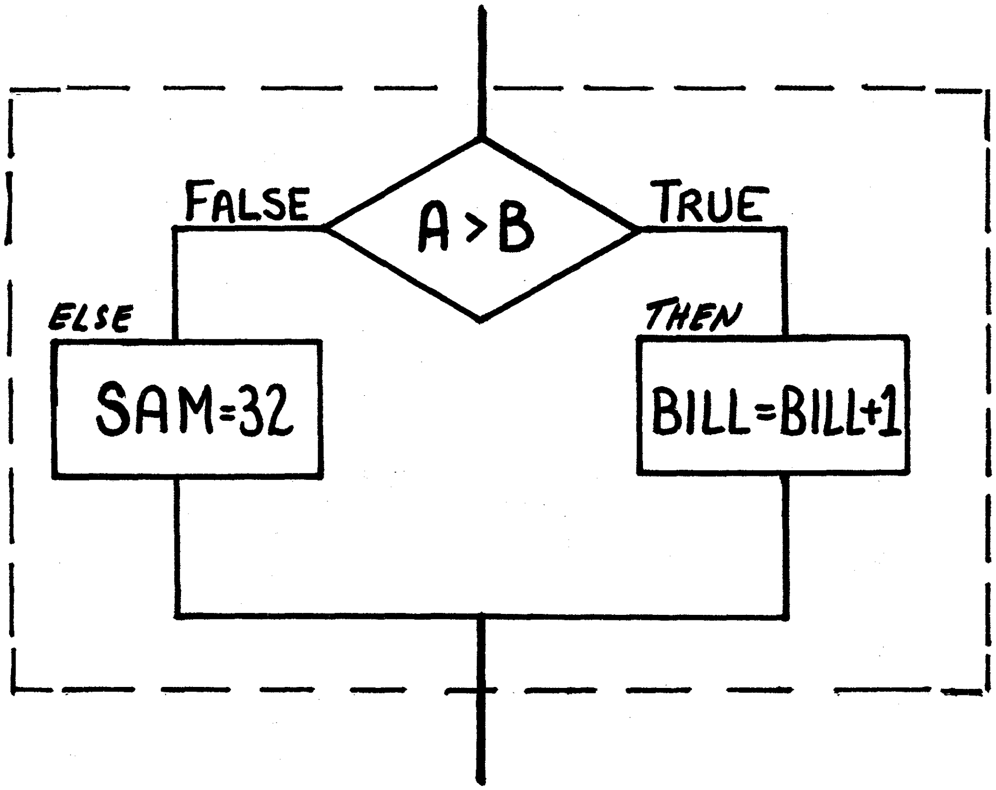
```
     COL 1          COL 8
       ↓            ↓
 ①                 PROGRAM ONE

       C FIRST PROGRAM I EVER WROTE

 ②                 INTEGER I,J,K

 ③     10          READ(5,*) I,J

 ④                 K= I+J

 ⑤                 WRITE(6,*) K

 ⑥                 STOP

 ⑦                 END
```

EXECUTABLE STATEMENTS
THREE BASIC CONSTRUCTS:



EXECUTABLE STATEMENTS

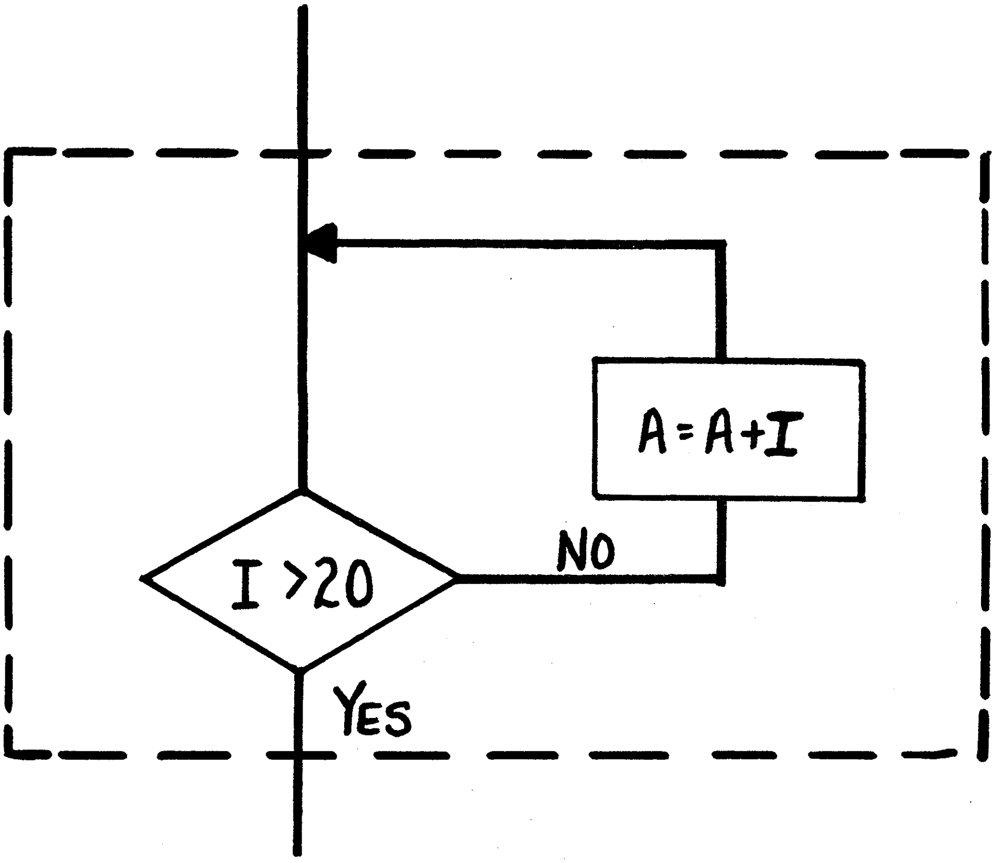
IF - THEN - ELSE



```
IF (A .GT. B) THEN
BILL = BILL+1
ELSE
SAM = 32
ENDIF
```

EXECUTABLE STATEMENTS

DO LOOP



```
      DO 10 I=1,20
      A=A+I
10    CONTINUE
```

ANOTHER SIMPLE PROGRAM

```
                  PROGRAM TWO
     C PROGRAM    TO USE IF-THEN-ELSE
                  INTEGER X,Y,Q
     10           READ (5,*) X,Y
                  Q = X + Y
①                 IF (Q.LT.122), THEN
                  WRITE (6,2Ø)
②    20           FORMAT ('NUMBER IS LESS THAN 122')
                  ELSE
                  WRITE (6,21)
     21           FORMAT ('NUMBER IS GREATER THAN 121')
                  ENDIF
                  GO TO 10
③                 END
```

YET ANOTHER PROGRAM!

```
                     PROGRAM THREE
          C PROGRAM WHICH USES THE DO LOOP
                     INTEGER X,Y,I
①                    DIMENSION X(5)
          10         READ (5,*) X(1),X(2),X(3),X(4),X(5)
                     Y=Ø
②                    DO 20 I=1,5
          20         Y=Y+X(I)
                     WRITE (6,*) Y
                     GO TO 10
                     END
```

SUMMARY

THE ASSIGNMENT STATEMENT:

```
X = A+B
```

MEANS

ADD A TO B AND PLACE THE RESULT IN X.

```
X = X+1
```

MEANS

TAKE THE CURRENT VALUE OF X, ADD 1 TO IT, AND PLACE RESULT BACK IN X.

MORE COMPLICATED ASSIGNMENT STATEMENTS:

```
X(I) = (B * 75.Ø1)**2
```

OR $(75.\emptyset 1\ B)^2$

```
JHAWK = (SQRT((B**2)-4.0*A*C))/2.0*A
```

SUMMARY

IF - THEN - ELSE   ENDIF

```
        IF (A.GT.B), THEN
 ①      A = B + C - 35.0
 ②      BSAT = 25.0 * FOURVA
        ELSE
 ③      B = 26.05 - 4 * A
 ④      CSAT = SIN (X)
 ⑤      JBK = B ** 3 - (2 * B) ** 2
        ENDIF
```

IF A IS GREATER THAN B, EXECUTE STATEMENTS ① AND ②.

IF A IS LESS THAN OR EQUAL TO B, EXECUTE STATEMENTS ③, ④, AND ⑤.

NOTE: THE CONDITIONAL STATEMENT CAN BE CONSIDERABLY MORE COMPLICATED THAN JUST A > B.

SUMMARY

THE DO LOOP

```
      DO 20 JVAL = 1, 152,2
20    ARRAY (JVAL) = SIN (SQRT(JVAL))
```

THIS DO LOOP SETS THE VALUE OF ELEMENT 1,3,5, etc.(UP TO 151) OF ARRAY TO THE SINE OF THE SQUARE ROOT OF THE VALUE OF ITS INDEX. (WHEW!)

1 - BEGINNING VALUE OF JVAL
2 - INCREMENT USED (1,3,5,7, etc.)
152 - THE LAST VALUE; IN THIS CASE, THE LOOP WILL END 147, 149, 151. THE NEXT NUMBER WOULD BE 153, BUT IT IS BIGGER THAN 152, SO IT IS NOT DONE.

EXTENSIONS

TO ACCOMODATE THE ADDED INPUT/OUTPUT CAPABILITIES OF ITS PROCESSORS, INTEL HAS ADDED TWO EXTENSIONS TO FORTRAN-80: INPUT AND OUTPUT FUNCTIONS.

CALL INPUT (PORTNUMBER, VAR) CAUSES PORTNUMBER TO BE READ AND THE DATA PLACED IN VAR (8 BITS ONLY)

CALL OUTPUT (PORTNUMBER, VAR) CAUSES 8 BITS OF DATA FROM VAR TO BE OUTPUT TO PORTNUMBER

NOTE: PORTNUMBER MUST BE A <u>CONSTANT</u>

EXAMPLES:

```
        CALL INPUT(3, JDUM)
        CALL OUTPUT(23, PVAL)
```

AN EXAMPLE
OF
INPUT AND OUTPUT

```
              PROGRAM FIVE
C  THIS PROGRAM TURNS THE SYSTEM
C  INTO AN EXPENSIVE SWITCH!
              INTEGER TEMPDT
10            CALL INPUT (Ø,TEMPDT)
              CALL OUTPUT (Ø,TEMPDT)
              GO TO 10
              END
```

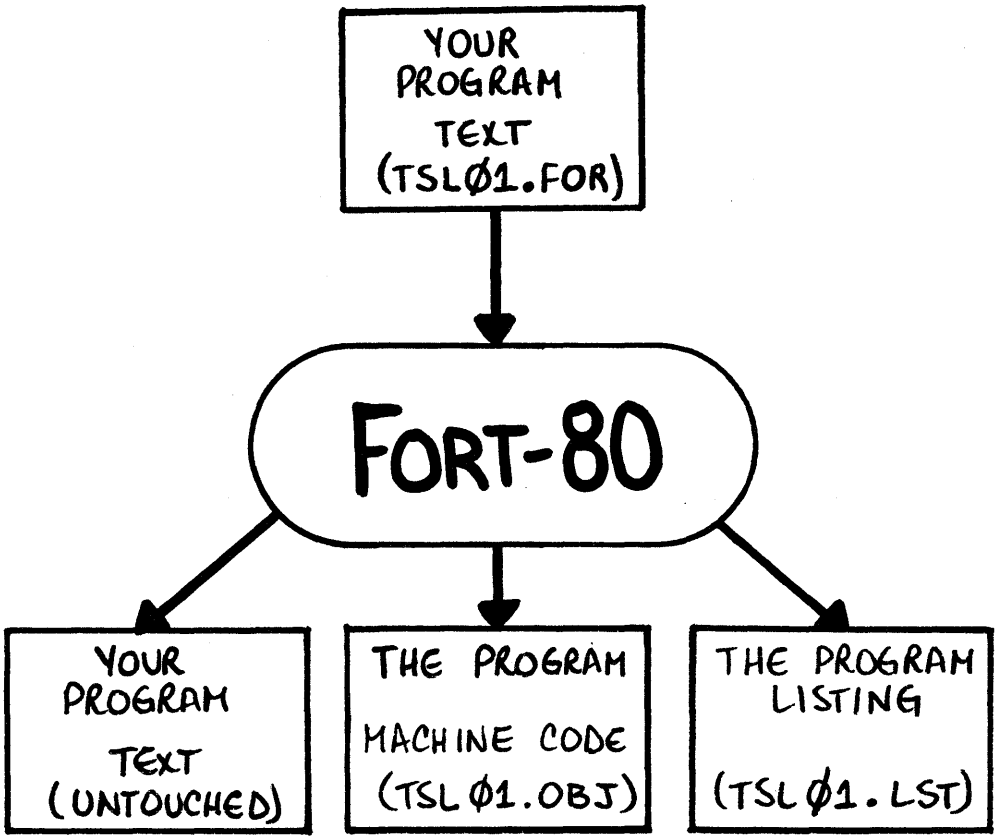
THE TRANSLATION STEP

ONCE YOU HAVE CREATED A PROGRAM (SUCH AS 1 THRU 5) USING THE TEXT EDITOR, YOU ARE NOW READY TO TRANSLATE IT TO MACHINE CODE. YOU COULD DO IT BY HAND OR LET THE MACHINE DO IT FOR YOU. TO DO THIS YOU:

```
FORT80 filename DEBUG
```

NOTE: DEBUG SPECIFIES THAT A SPECIAL SYMBOL TABLE SHOULD BE CREATED.

THE TRANSLATION STEP

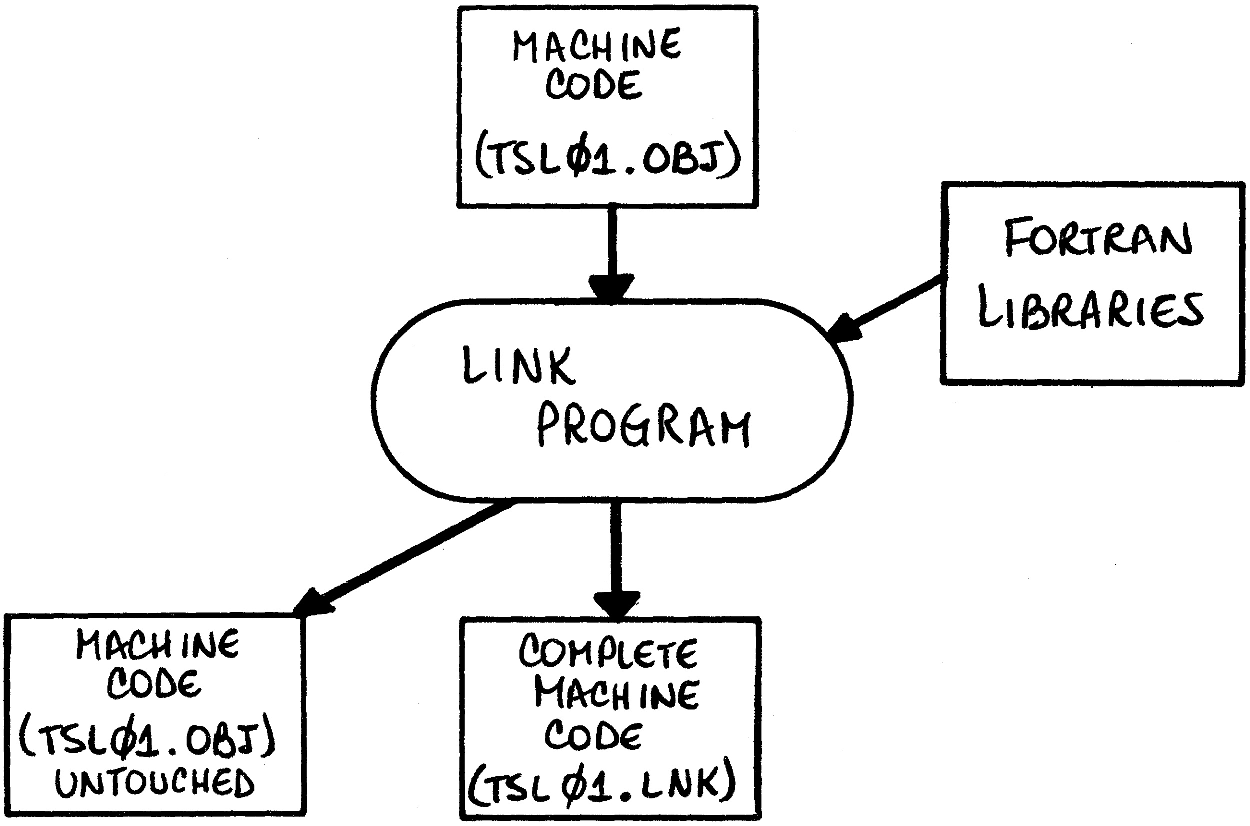

YOUR PROGRAM TEXT (TSLØ1.FOR)
FORT-80
YOUR PROGRAM TEXT (UNTOUCHED)
THE PROGRAM MACHINE CODE (TSLØ1.OBJ)
THE PROGRAM LISTING (TSLØ1.LST)

NOTE: FORT80 USES YOUR ROOT AND SUPPLIES .OBJ AND .LST FOR THE EXTENSIONS ON THE MACHINE CODE AND PROGRAM LISTING FILES IT CREATES.

THE LINK STEP

THE .OBJ FILE CREATED BY THE TRANSLATION STEP IS NOT COMPLETE. IT LACKS THE CODE OF THE FORTRAN ROUTINES (SUCH AS SINE, COSINE, etc.) THAT WERE INVOKED BY YOUR PROGRAM. TO CREATE A COMPLETE PROGRAM, WE MUST LINK THE .OBJ FILE WITH THE FORTRAN LIBRARIES.

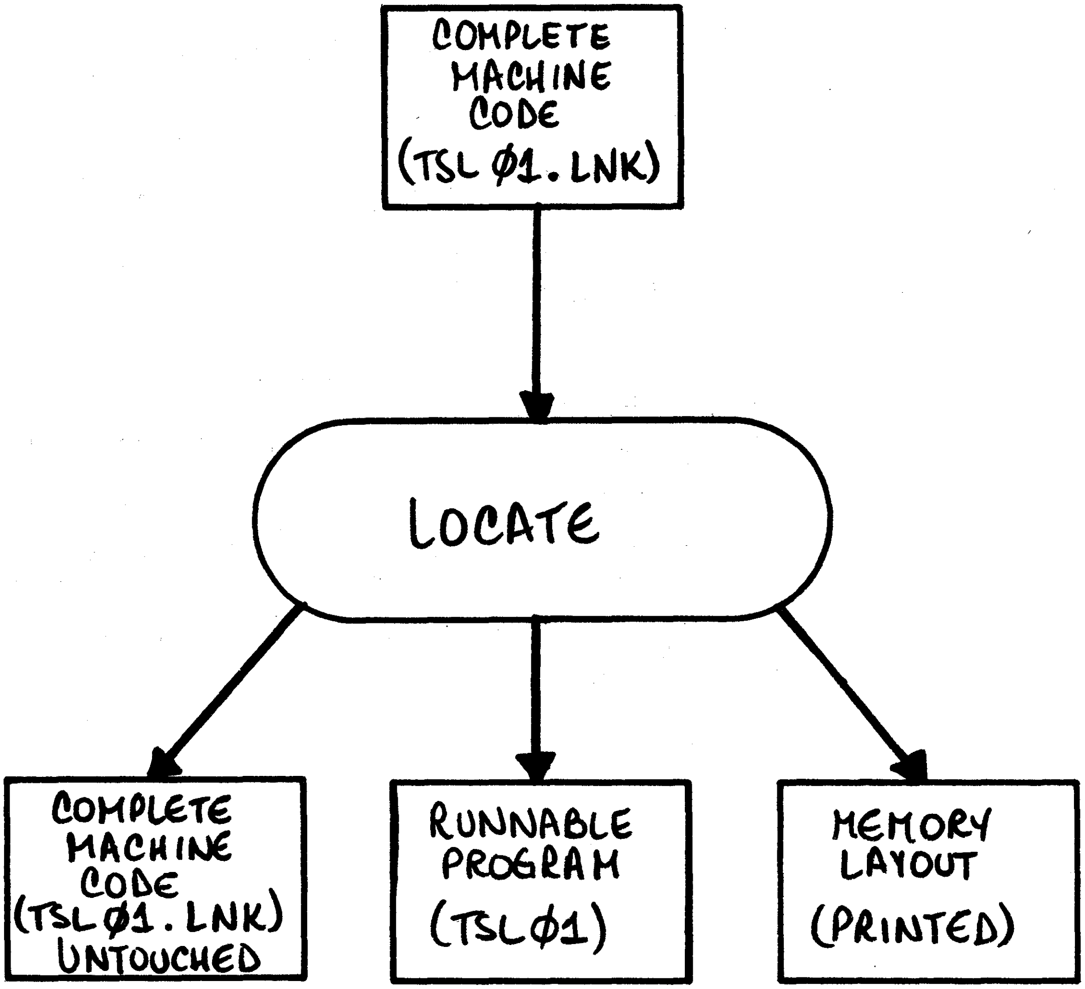
– SUBMIT FLINK(TSLØ1.OBJ, TSLØ1.LNK)


MACHINE CODE (TSLØ1.OBJ)
FORTRAN LIBRARIES
LINK PROGRAM
MACHINE CODE (TSLØ1.OBJ) UNTOUCHED
COMPLETE MACHINE CODE (TSLØ1.LNK)

THE LOCATE STEP

WHILE THE PROGRAM IS NOW COMPLETE, IT IS NOT ASSIGNED TO ANY PARTICULAR MEMORY LOCATION. THE FINAL STEP OF PROCESSING IS TO <u>LOCATE</u> THE PROGRAM.

```
LOCATE TSLØ1.LNK  MAP LINES SYMBOLS PRINT(:LP:)
```


COMPLETE MACHINE CODE (TSL Ø1.LNK)
LOCATE
COMPLETE MACHINE CODE (TSLØ1.LNK) UNTOUCHED
RUNNABLE PROGRAM (TSLØ1)
MEMORY LAYOUT (PRINTED)

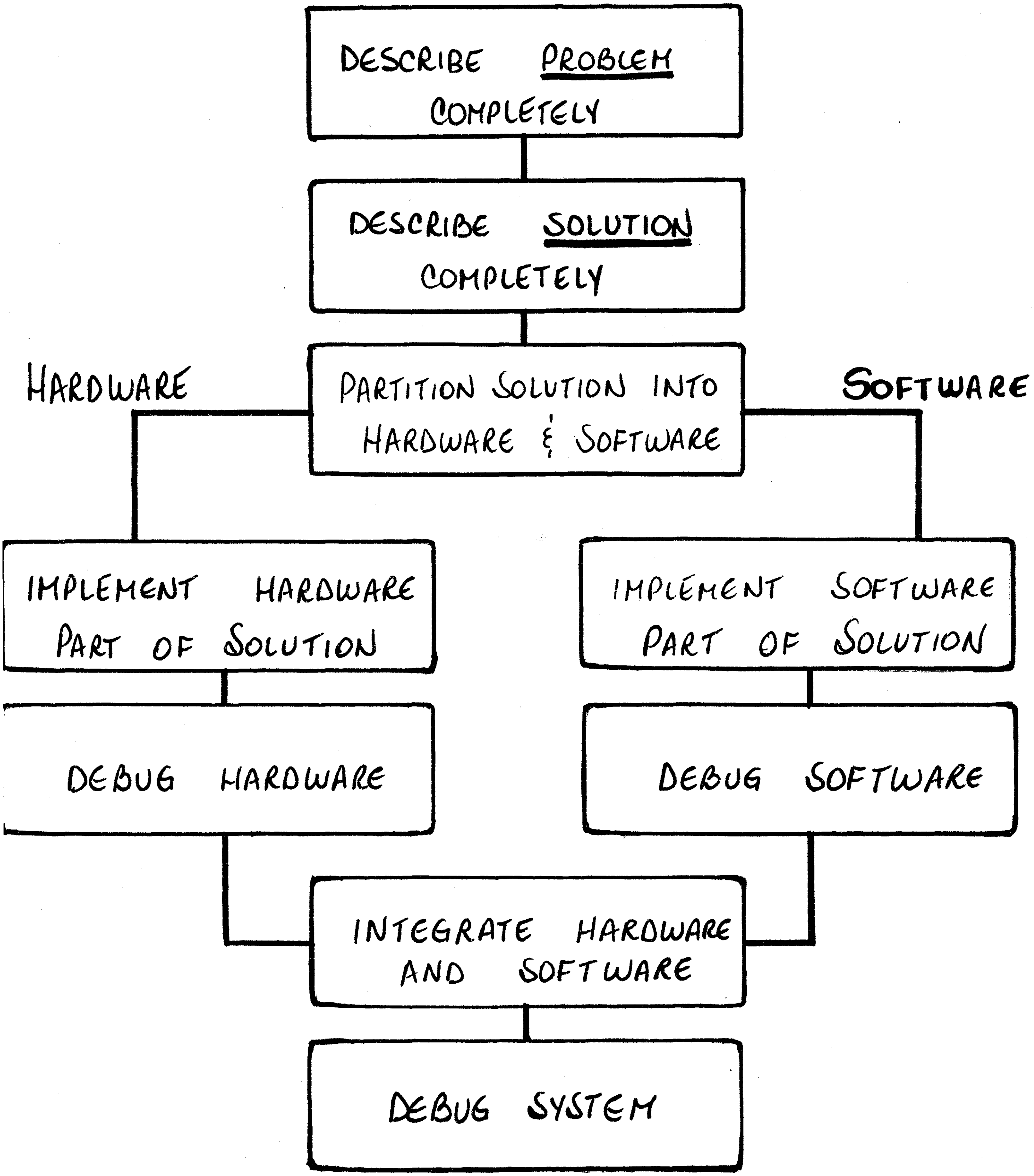
RUNNING YOUR PROGRAM

TO RUN YOUR PROGRAM, YOU NEED ONLY REFER TO THE FILE THAT IT IS STORED IN.

IN OUR RUNNING EXAMPLE THIS IS:

```
TSLØ1
```

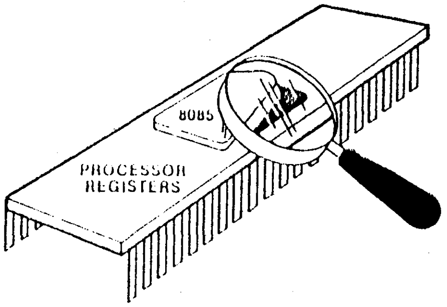
The Development Steps
(A review)



Describe Problem Completely
Describe Solution Completely
Hardware
Partition Solution into Hardware & Software
Software
Implement Hardware Part of Solution
Implement Software Part of Solution
Debug Hardware
Debug Software
Integrate Hardware and Software
Debug System

CHAPTER 4

ICE-85



ICE
IN-CIRCUIT EMULATOR

WHAT CAN ICE DO FOR ME?

1. HARDWARE DEBUG

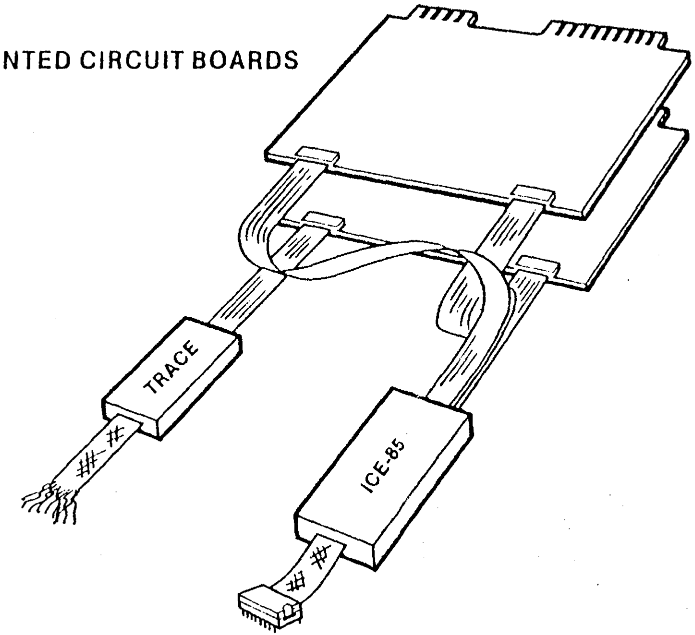
2. SOFTWARE DEBUG

3. SYSTEM DEBUG

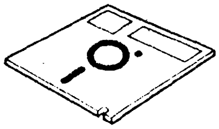
4. FINAL SYSTEM TEST FOR PRODUCTION

ICE

THIS IS AN ICE.85 UNIT

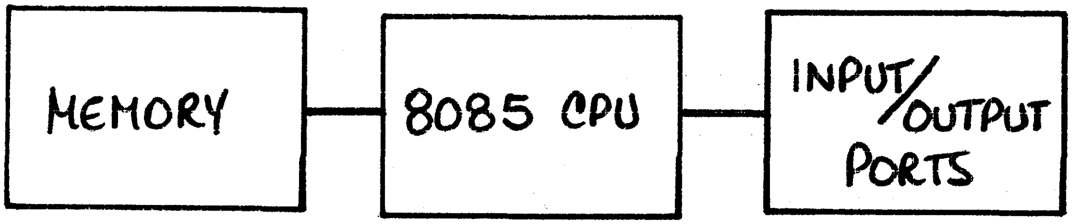


- TWO PRINTED CIRCUIT BOARDS
- TRACE CABLE
- UMBILICAL CABLE

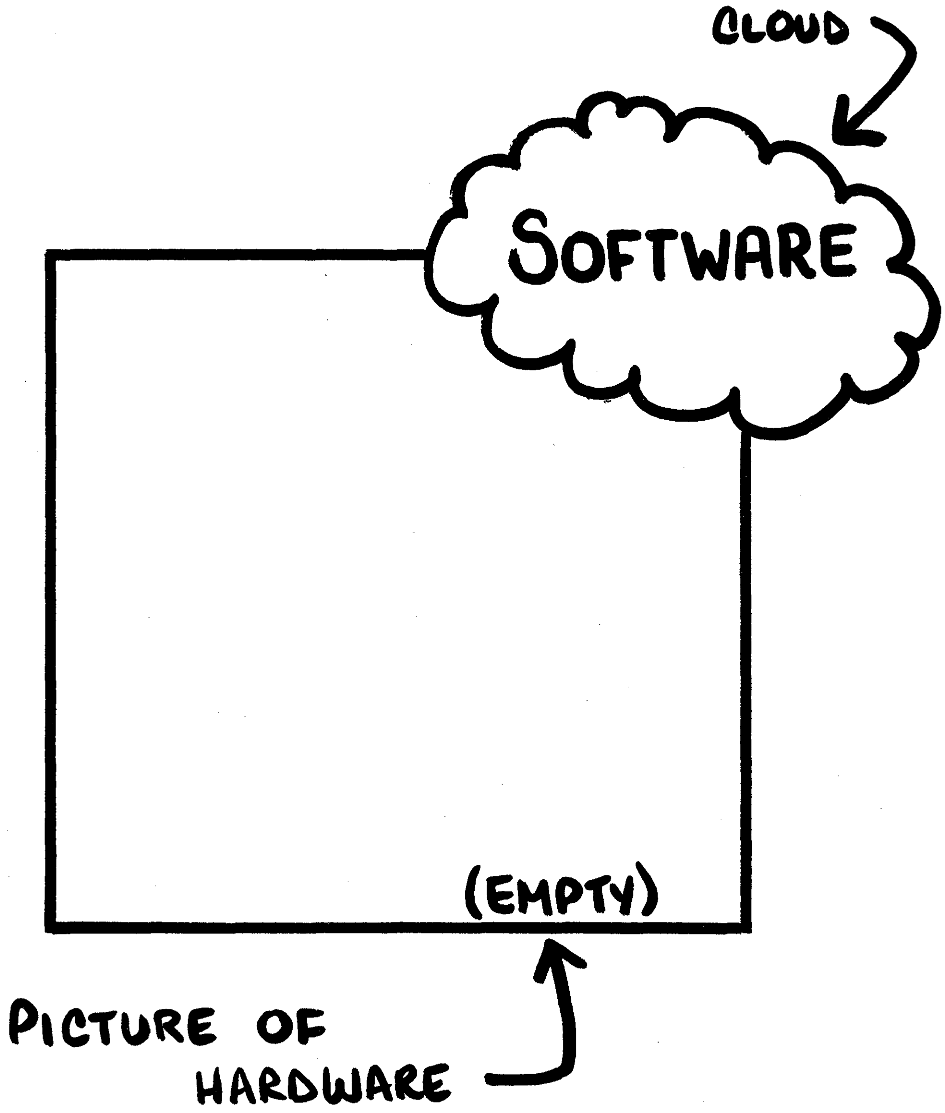


- SOFTWARE DRIVER

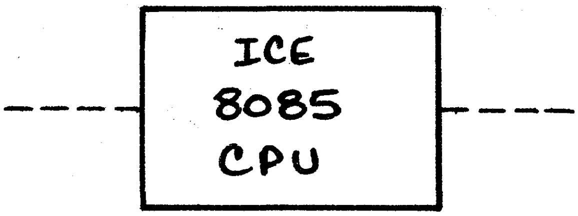
A TYPICAL SYSTEM



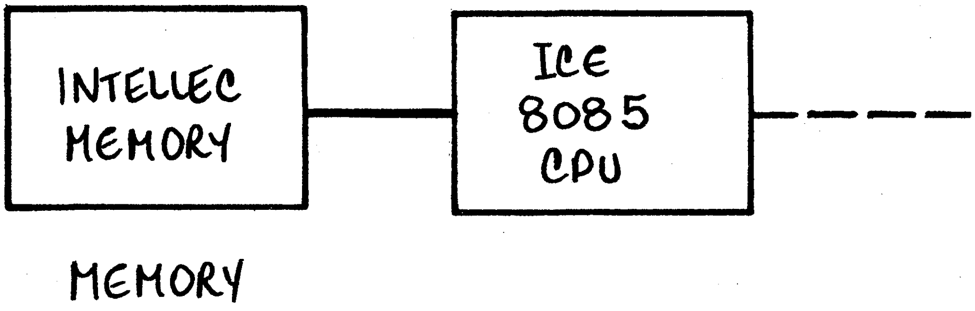
IN THE BEGINNING OF HARDWARE DEVELOPMENT, OUR SYSTEM LOOKS LIKE THIS:

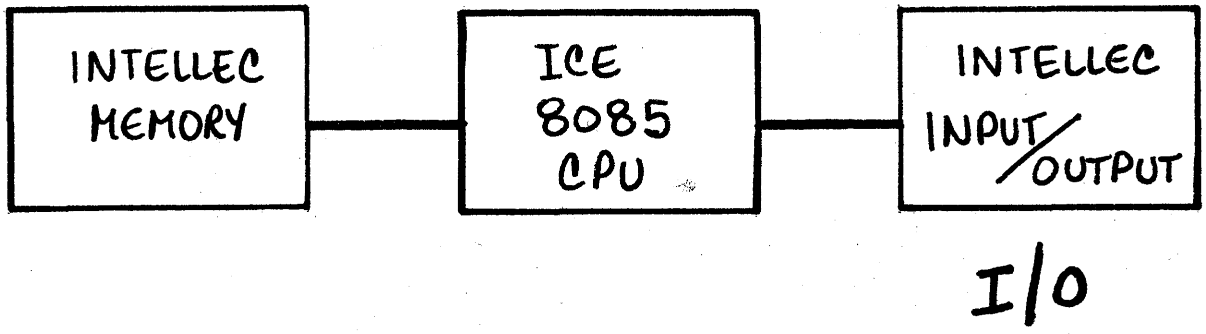


WITH ICE PRESENT, WE HAVE THIS AT FIRST:



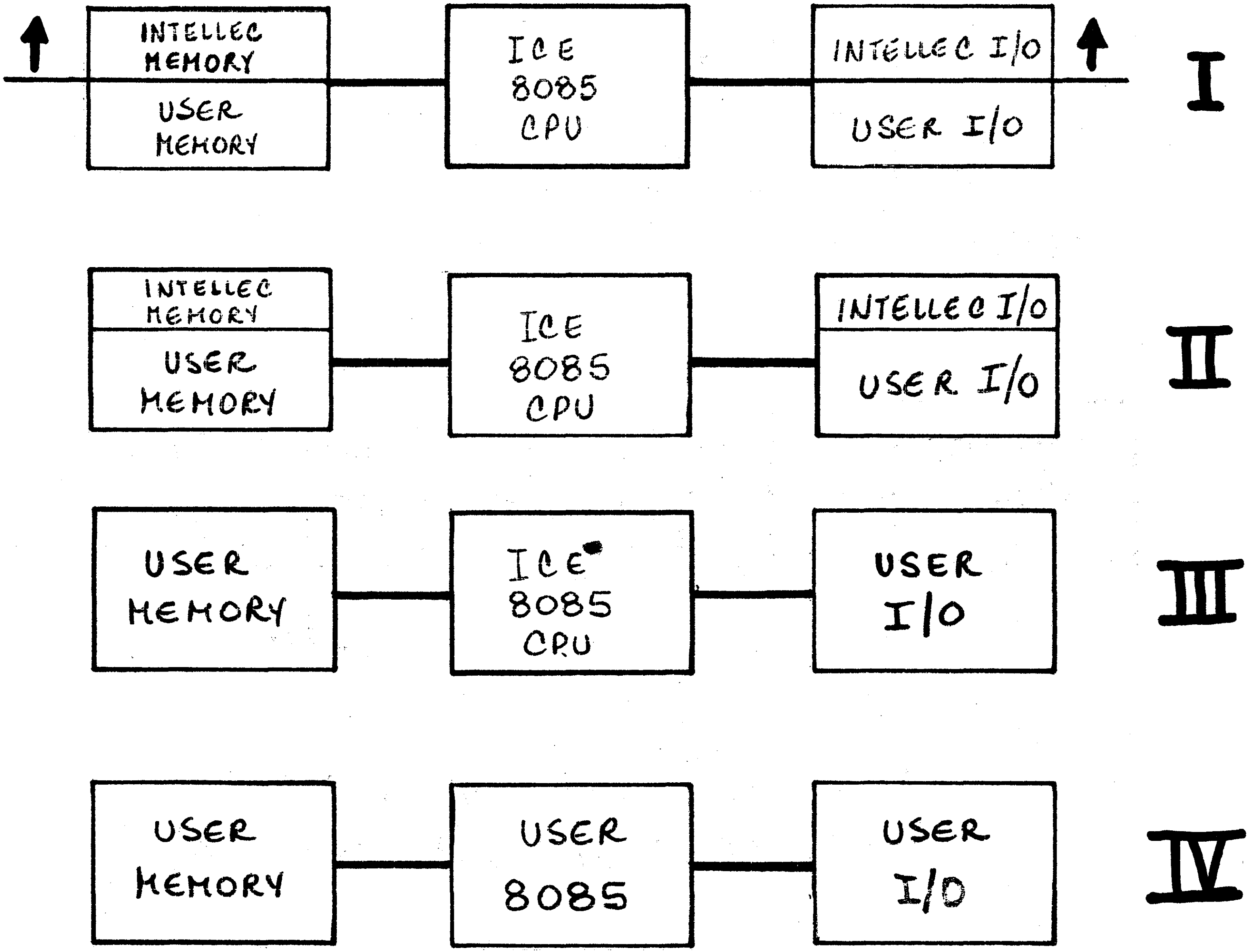
ICE ALLOWS US TO BORROW RESOURCES FROM THE DEVELOPMENT SYSTEM





NOW WE HAVE "HARDWARE" ON WHICH TO TEST OUR SOFTWARE!

AS USER HARDWARE BECOMES AVAILABLE, WE CAN USE IT DIRECTLY AND CHECK OUT ITS FUNCTION WITH THE REST OF OUR "HARDWARE"



ICE 85 STEPS

1. READY ANY REAL HARDWARE.

2. INVOKE ICE-85.

3. BORROW NEEDED RESOURCES FROM INTELLEC WITH MAP COMMANDS.

4. LOAD USER SOFTWARE INTO ICE.

5. USING ICE COMMANDS, RUN, STEP, DISPLAY, AND MODIFY UNTIL PROGRAM FUNCTIONS CORRECTLY.

6. IF PROGRAM MODIFICATIONS ARE NECESSARY, MAKE THEM IN THE SOURCE PROGRAM. RECOMPILE, LINK, LOCATE, AND RETEST.

CHAPTER 5

— MAINTENANCE —

MAINTENANCE

- PREVENTATIVE

- UNSCHEDULED

PREVENTATIVE MAINTENANCE PHILOSOPHY:

IF IT WORKS;

Leave it alone!

MECHANICAL CHECKS:

1. FILTERS
2. FANS

ELECTRONIC CHECKS:

1. BUILT-IN CONFIDENCE CHECK
2. Z$ IN MONITOR
3. DISKETTE CONFIDENCE CHECK

REMEMBER:

THESE ARE CONFIDENCE CHECKOUTS, NOT COMPLETE DIAGNOSTICS.

SOFTWARE CHECKS:

- IS YOUR SYSTEM AND SYSTEM SOFTWARE ALL PROPERLY REGISTERED?

- DO YOU HAVE THE LATEST VERSIONS OF THE SOFTWARE YOU ARE USING?

UNSCHEDULED MAINTENANCE;

- OR -

WHAT TO DO **BEFORE** YOU CALL THE HOTLINE.

1 - RESEAT CARDS
2 - CHECK CONNECTORS
3 - CHECK SOCKETED CHIPS

CHAPTER 6

PROGRAMMING AIDS

PROGRAMMING AIDS

COMPILER (ASSEMBLER) LEVEL

$INCLUDE

ISIS LEVEL

SUBMIT

$INCLUDE

THE INCLUDE FEATURE IS COMMON TO ALL INTEL SUPPLIED COMPILERS AND ASSEMBLERS.

PERMITS THE INCLUSION OF BLOCKS OF SOURCE CODE WITHIN ANY PROGRAM.

$INCLUDE

FIRST POSSIBILITY

I NEED A COPYRIGHT NOTICE IN THE BEGINNING OF **EVERY** PROGRAM I WRITE.

1. CREATE A FILE CALLED

   COPYRI.GHT (CLEVER, HUH?)

2. CONTENTS WOULD BE

```
C   THE FOLLOWING PROGRAM IS
C   COPYRIGHTED. THE UNAUTHORIZED
C   DUPLICATION OF THIS PROGRAM,
C   OR ANY PART BY ANY MEANS,
C   ELECTRONICAL, MECHANICAL, etc.
```

3. TO USE THE NOTICE PLACE

```
$INCLUDE (COPYRI.GHT)
```

   AS THE FIRST LINE OF YOUR PROGRAM ($ GOES IN COLUMN 1).

4. WHEN THE SOURCE CODE IS COMPILED, THE FILE COPYRI.GHT WILL BE READ AND PROCESSED AS IF IT WERE PART OF YOUR SOURCE CODE!

$INCLUDE

SECOND POSSIBILITY

I WANT TO INCLUDE THE SAME SET OF DATA DECLARATIONS IN MANY DIFFERENT PROGRAMS.

1. CREATE A FILE CALLED

   COMMON.DAT

2. CONTENTS WOULD BE

```
DIMENSION   AX(12), BINT(1Ø)
DIMENSION   INT (25)
INTEGER     IJACK, B17, CCHAR
REAL        KJ, ZCHAR
            etc.
```

3. TO USE THIS FILE, PLACE
   $INCLUDE (COMMON.DAT)
   WHEREVER YOU HAVE DATA DECLARATIONS IN YOUR MAINSTREAM CODE.

SUBMIT

THE SUBMIT FACILITY OF ISIS ALLOWS BATCH PROCESSING OF ISIS COMMANDS.

SUBMIT
A TYPICAL JOB STREAM

```
— FORT80     :F1: JACK.FOR

— FORT80     :F1: JILL.FOR

— FORT80     :F1: HILL.FOR

— COPY     :F1: JACK.LST  TO  :LP:

— COPY     :F1: JILL.LST  TO  :LP:

— COPY     :F1: HILL.LST  TO  :LP:

— LINK     :F1: JACK.OBJ, :F1: JILL.OBJ,  &
           :F1: HILL.OBJ  TO  :F1: TOTAL.LNK

— LOCATE   :F1: TOTAL.LNK
```

SUBMIT

CREATE A FILE CALLED

3COMP.CSD

```
FORT 8Ø       :F1: JACK.FOR
FORT 8Ø       :F1: JILL.FOR
FORT 8Ø       :F1: HILL.FOR
COPY     :F1: JACK.LST  TO  :LP:
COPY     :F1: JILL.LST  TO  :LP:
COPY     :F1: HILL.LST  TO  :LP:
LINK     :F1: JACK.OBJ, :F1: JILL.OBJ,   &
         :F1: HILL.OBJ  TO :F1: TOTAL.LNK
LOCATE   :F1: TOTAL.LNK
```

THEN AFTER READYING ALL FILES ON DRIVE 1,

SUBMIT 3COMP

SUBMIT

THE FILE 3COMP.CSD WOULD ONLY COMPILE, LINK, AND LOCATE JACK, JILL, AND HILL. WE WANT A GENERAL PURPOSE FILE, SO CREATE 3ACOMP.CSD LIKE:

```
FORT8Ø    %Ø.FOR
FORT8Ø    %1.FOR
FORT8Ø    %2.FOR
COPY  %Ø.LST  TO  :LP:
COPY  %1.LST  TO  :LP:
COPY  %2.LST  TO  :LP:
LINK %Ø.OBJ, %1.OBJ, %2.OBJ TO &
                        %3.LNK
LOCATE %3.LNK
```

TO USE:

```
                   %Ø         %1        %2          %3
SUBMIT  3ACOMP(:FI:JACK, :FI:JILL, :FI: HILL, :FI:TOTAL)
```

SUBMIT

WOULDN'T IT BE NICE IF WE COULD PAUSE AFTER THE COMPILE STEP TO CHECK THE RESULTS? NO USE PRINTING OUT LISTINGS WITH LOTS OF ERRORS, SO MODIFY THE FILE 3ACOMP.CSD

```
FORT 8Ø     %Ø.FOR
FORT 8Ø     %1.FOR
FORT 8Ø     %2.FOR
↑E(CR)
COPY  %Ø.LST  TO  :LP:
COPY  %1.LST  TO  :LP:
COPY  %2.LST  TO  :LP:
         etc.
```

USE AS BEFORE.

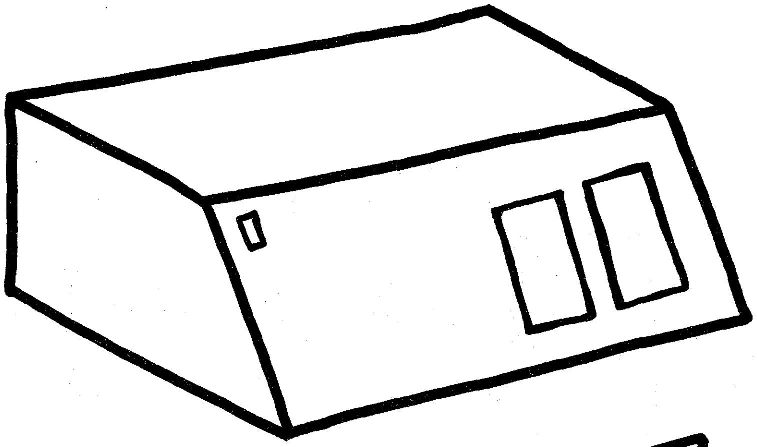
WHEN THE ↑E IS ENCOUNTERED, CONTROL REVERTS TO THE CONSOLE. IF COMPILATIONS ARE OK, TYPE ↑E AND AUTOMATIC OPERATION RESUMES. IF NOT, PRESS INTERRUPT 1 TO CANCEL AUTOMATIC OPERATION.

CHAPTER 7

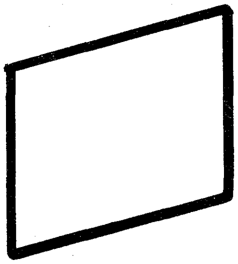
UPM

UPP HARDWARE

THE UPP HARDWARE CONSISTS OF:

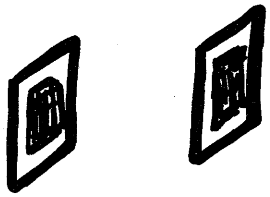


UPP CHASSIS



UPP CONTROL CARD (4004 μP)

PERSONALITY CARDS



ADAPTOR SOCKETS (FOR 40-PIN EPROMS)

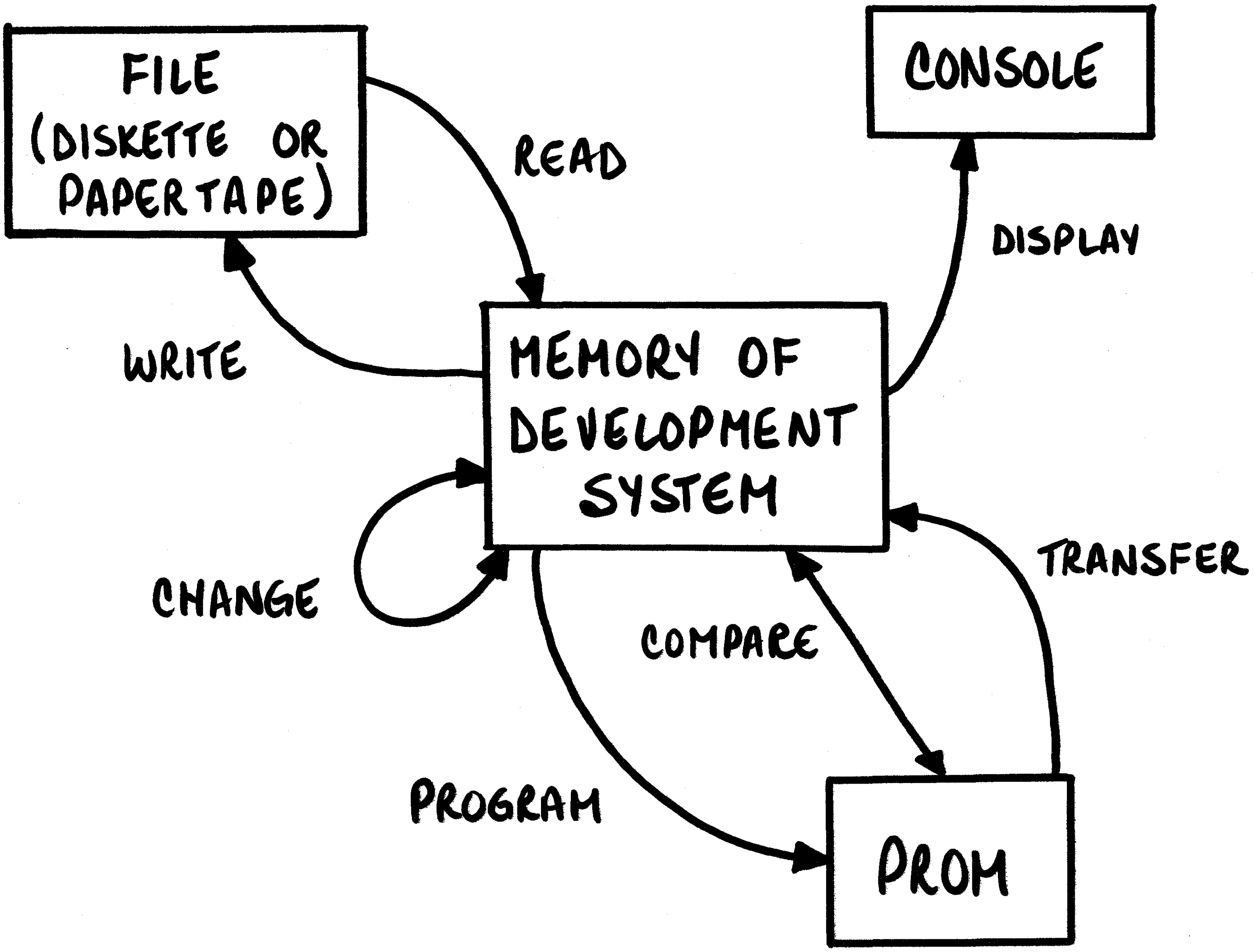
UPP SOFTWARE

THE UPP SOFTWARE CONSISTS OF:

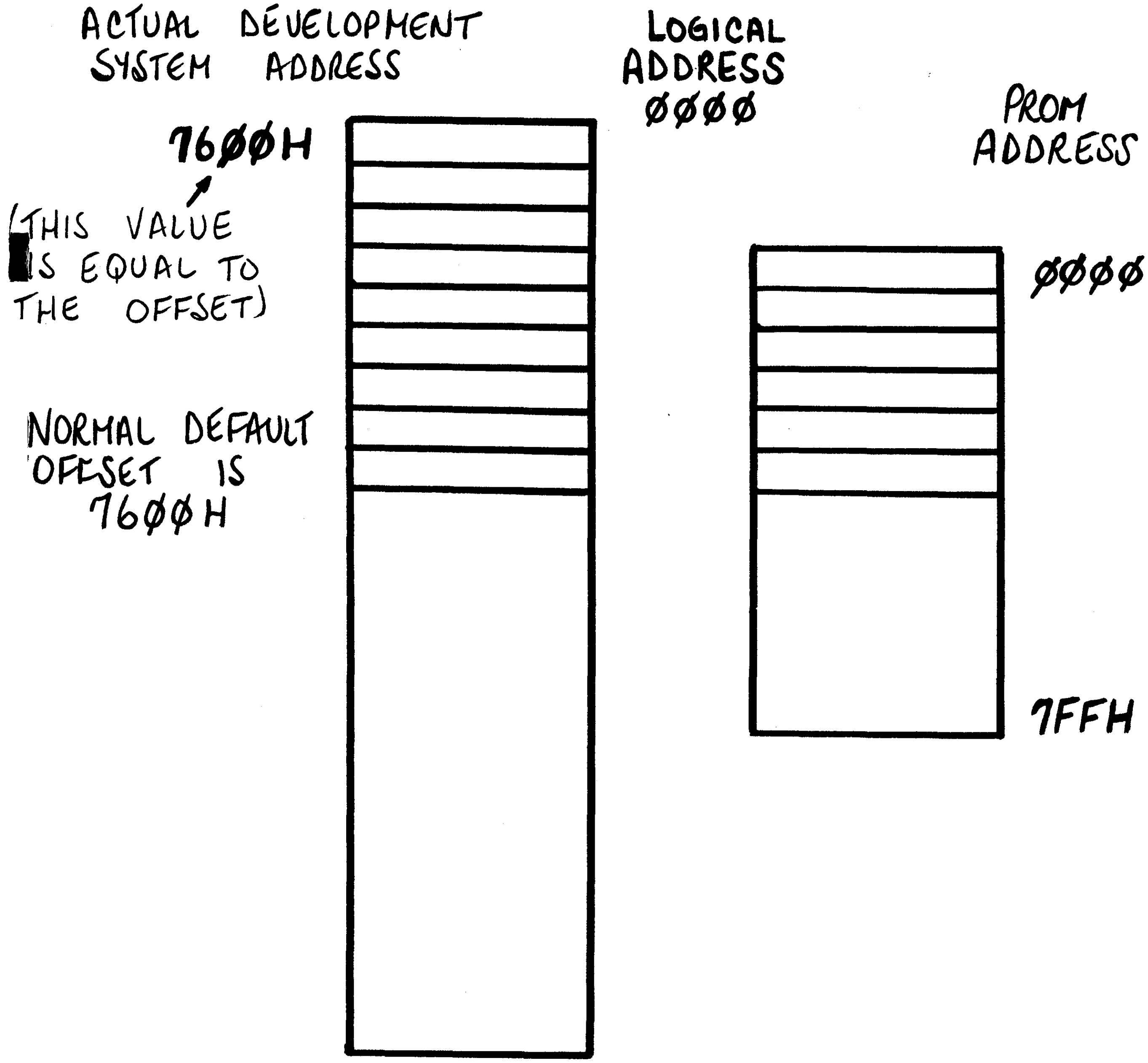
UPM

USAGE OF UPP

COMMANDS AND DATA FLOW



UPP MEMORY USAGE


ACTUAL DEVELOPMENT SYSTEM ADDRESS
7600H
(THIS VALUE IS EQUAL TO THE OFFSET)
NORMAL DEFAULT OFFSET IS 7600H
LOGICAL ADDRESS 0000
PROM ADDRESS
0000
7FFH

READ COMMAND

```
READ filetype FILE filename INTO bias
```

filetype — OBJECT
86 HEX
HEX default
BNPF

filename — FILE WHICH CONTAINS DATA

bias — USUALLY Ø

EXAMPLE:

```
READ OBJ FIL :F1:TOM INTO Ø
READ FILE :F3:JACK.HEX INTO Ø
```

DISPLAY COMMAND

DISPLAY FROM start TO finish

BOTH START AND FINISH ADDRESSES ARE <u>LOGICAL</u> ADDRESSES

EXAMPLE:

```
DIS  FRO  Ø  TO  7FH
DIS  FROM 1ØØH TO 3ØØH
```

TRANSFER COMMAND

TRANSFER FROM start TO finish

BOTH START AND FINISH
ARE LOGICAL ADDRESSES

THE UPM SOFTWARE ASSUMES DATA TO BE TRANSFERRED STARTS WITH LOCATION Ø IN THE ROM!

EXAMPLE:

```
TRA FRO Ø TO 3FFH

TRA FRO 4ØØH TO 7FFH
```

COMPARE COMMAND

COMPARE FROM start TO finish

BOTH START AND FINISH
ARE <u>LOGICAL</u> ADDRESSES

COMPARE, LIKE TRANSFER,
<u>ASSUMES</u> A STARTING
ADDRESS IN THE PROM OF Ø

EXAMPLE:

COMPARE FRO Ø TO 3FFH
COMPARE FRO 4ØØH TO 7FFH

PROGRAM COMMAND

```
PROGRAM FROM start TO finish;
        START prom start
```

BOTH START AND FINISH ARE LOGICAL ADDRESSES

prom start IS A PROM ADDRESS

EXAMPLE:

```
PRO FRO ∅ TO 3FFH START ∅
PRO FRO 8∅∅H TO 87∅H START ∅
```

(NOTE: SOME PROMS CANNOT BE PARTIALLY PROGRAMMED; SEE UPP USERS' MANUAL)

CHANGE COMMAND

CHANGE start = new, new2, new3, etc.

WHERE start IS A LOGICAL ADDRESS

new, new2, etc., ARE THE NEW DATA TO BE PLACED IN SUCCESSIVE LOCATIONS

EXAMPLE:

CHANGE Ø = 3EH, ØEH, ØD3H, etc.

WRITE COMMAND

WRITE FROM start to finish FILE filename filetype

Where start and finish
are LOGICAL ADDRESSES

filename is the file TO BE WRITTEN

filetype is the same as READ COMMAND

EXAMPLE:

```
WRITE FROM Ø TO 7FFH FILE :F1: JACK.OBJ
WRI FRO 8ØØH TO ØFFFH FILE :F1: JILL.HEX
```

SAMPLE SESSION

COMMAND	COMMENT
-UPM	CALL IN UPM SOFTWARE
ISIS-II PROM MAPPER VX.X	SOFTWARE SIGNS ON
TYPE * 8755	ASKS FOR TYPE; WE GIVE 8755
* SOCKET =2	WE HAVE PERSONALITY MODULE IN SOCKET 2
* TRANSFER FROM Ø TO 7FFH	WE PLACE 8755 IN SOCKET 2 AND READ TO VERIFY IT'S ALL ERASED
* DISPLAY FROM Ø TO ØFFH	SHOULD SEE ØFFH FROM EACH POSITION

SAMPLE SESSION

(CONTINUED)

```
COMMAND                                 COMMENT

ØØØØ  FF  FF  FF  FF  FF etc.
           :
ØØFØ  FF  FF  FF  FF  etc.


* READ OBJECT FILE                      READ OUR FILE TO
    :F1:TEST INTO Ø                       BE PROGRAMMED


* DISPLAY FROM                          CHECK FOR
    Ø TO ØFH                              PROPER DATA


ØØØØ  3E  ØE  D3  2Ø etc.


* PROGRAM FROM Ø TO                     PROGRAM THE FIRST
    7FH START Ø                           7FH LOCATION
```

SAMPLE SESSION

(CONTINUED)

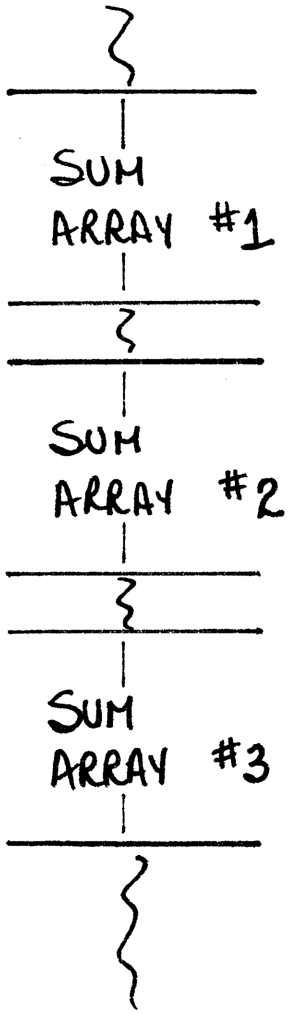
COMMAND	COMMENT
* COMPARE FROM Ø TO 7FH	CHECK ONE MORE TIME
* EXIT	DONE; EXIT TO ISIS
_	

APPENDIX

A NEW TOPIC

-THE SUBROUTINE

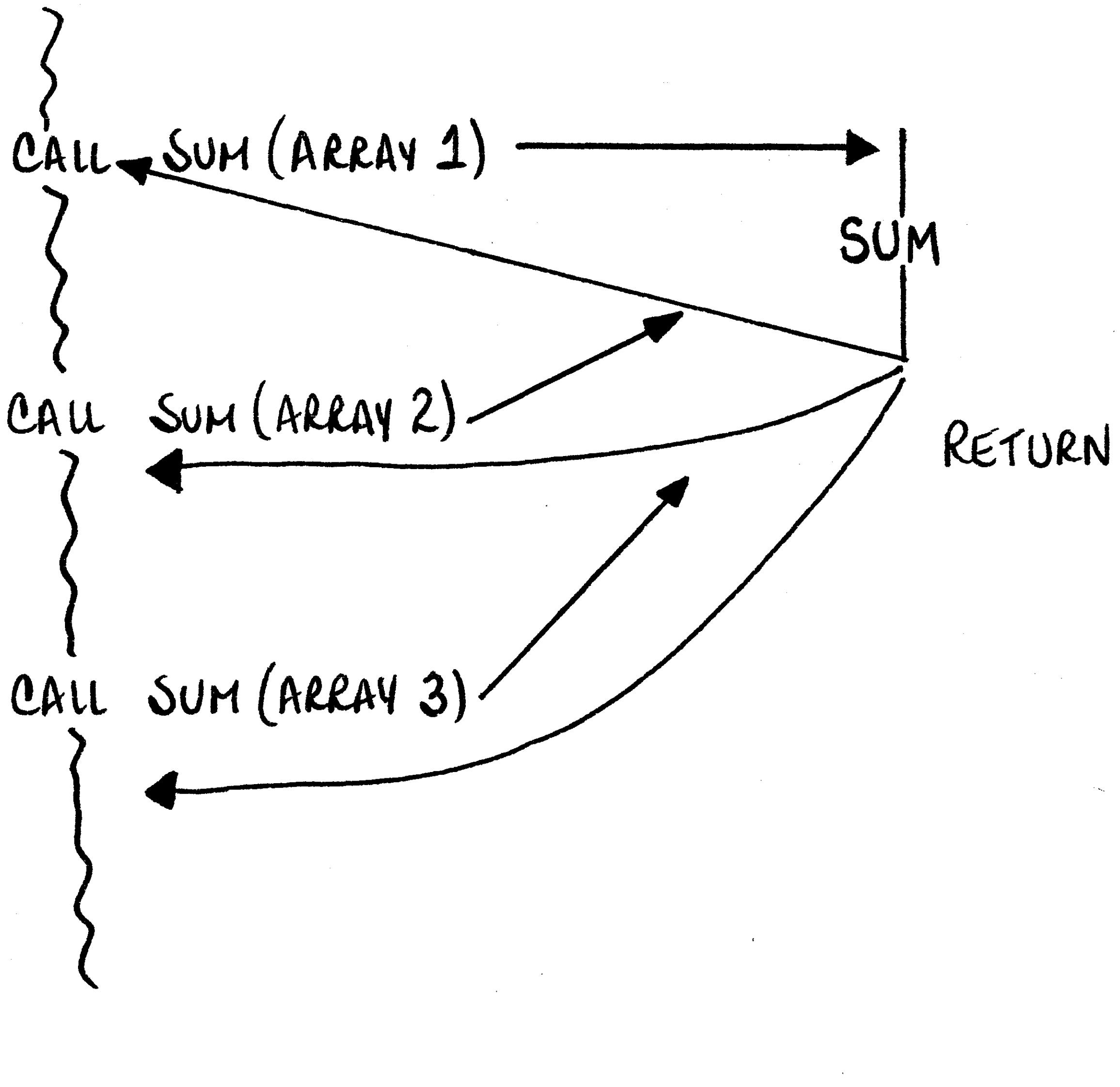
SOMETIMES WE FIND OURSELVES DOING THE SAME ROUTINE (SUCH AS SUMMING AN ARRAY) OVER AND OVER; LIKE THIS:



SUBROUTINES

WOULDN'T IT BE NICE TO USE A COMMON PROGRAM TO DO THE SUMS?

LIKE THIS:



SUBROUTINES
- AN EXAMPLE

```
      PROGRAM FOUR
      INTEGER A, B, C, ATOT, BTOT, CTOT, Z, I
      DIMENSION A(4), B(4), C(4)
10    READ(5,*) A(1), A(2), A(3), A (4)
      READ(5,*) B(1), B(2), B(3), B(4)
      READ (5,*) C(1), C(2), C(3), C(4)
      CALL SUMTOT (A)
      ATOT = Z
      WRITE (6,*)ATOT
      CALL SUMTOT (B)
      BTOT = Z
      WRITE (6,*)BTOT
      CALL SUMTOT (C)
      CTOT= Z
      WRITE (6,*) CTOT
      STOP
      SUBROUTINE SUMTOT(ALPHA)
      Z=Ø
      DO I= 1,4
20    Z = Z + ALPHA (I)
      RETURN
      END
```

PROGRAM DEBUGGING (MONITOR STYLE)

OCCASIONALLY, YOUR PROGRAM MAY NOT WORK THE VERY FIRST TIME.

TO FIND THE POINT WHERE IT FAILS, WE CAN USE THE DEBUGGING CAPABILITIES OF THE MONITOR THAT IS PART OF THE DEVELOPMENT SYSTEM'S SOFTWARE.

SEQUENCE

1. LOAD YOUR PROGRAM.
2. STEP THROUGH THE PROGRAM USING MONITOR COMMANDS UNTIL THE FAILURE IS DETECTED.
3. CORRECT THE <u>SOURCE</u> PROGRAM; TRANSLATE, LINK, AND LOCATE.
4. TRY AGAIN.

PROGRAM DEBUGGING
(MONITOR STYLE)

1. LOAD PROGRAM.

```
-DEBUG :F3:JACK1
```

THIS CAUSES THE SYSTEM TO LOAD THE PROGRAM STORED IN :F3:JACK1 TO THEN TURN CONTROL OF THE COMPUTER OVER TO THE MONITOR.

MONITOR RESPONDS WITH

```
#3680

.
```

WHERE 3680 IS THE PROGRAMS STARTING ADDRESS, AND . IS THE MONITOR PROMPT CHARACTER.

PROGRAM DEBUGGING
(MONITOR STYLE)

2. STEP THROUGH YOUR PROGRAM USING MONITOR COMMANDS.

G [XXXX] [,YYYY] [,ZZZZ]   GO[AND SET BREAK POINT]

XXXX = START ADDRESS. IF OMITTED, CONTINUE WHERE YOU LEFT OFF.
YYYY = BREAK POINT (STOPPING POINT) #1
ZZZZ = " " " " #2

THE START AND BREAK ADDRESSES ARE OBTAINED FROM THE LOCATE LISTING.

EXAMPLES:

G3680	START AT 3680; DO NOT STOP.
G3680, 3752	START AT 3680; STOP IF YOU HIT 3752.
G3680, 3752, 3790	START AT 3680; STOP IF YOU HIT 3752 OR 3790.
G,37A7	CONTINUE FROM WHERE WE LAST STOPPED. STOP IF YOU HIT 37A7.

PROGRAM DEBUGGING
(MONITOR STYLE)

MONITOR COMMANDS (CON'T.)

Dxxxx,yyyy DISPLAY A RANGE OF MEMORY

XXXX = STARTING ADDRESS
YYYY = ENDING ADDRESS (MAY BE SAME AS START FOR A SINGLE BYTE)

XXXX AND YYYY CAN BE OBTAINED FROM THE SYMBOL INFORMATION ON THE LOCATE LISTING.

EXAMPLES:

D4277, 4310 DISPLAY THE BLOCK OF DATA FROM 4277 TO 4310.

DA7BB, A7BB DISPLAY THE SINGLE BYTE AT A7BB.

PROGRAM DEBUGGING
(MONITOR STYLE)

YOU SEE	ACTION
#3680	START PROGRAM (MONITOR HAS START ADDRESS ALREADY)
•	AND RUN TO SELECTED LINE IN
G,3690	PROGRAM (LINE 4 OR 3690)
#3690	HAVE MADE IT TO 3690 LOOK
.D3701,3701	AT TEMPDT (3701)
3701 ØF	IF HAS A ØF WHICH WAS ON THE SWITCHES
.G, 3698	NOW RUN TO LINE 6 (3698)
# 3698	NOTE LITES NOW HAVE ØF
.G	PROGRAM IS OK!
	LET IT RUN

ICE 85 COMMANDS

STEP 2 INVOKE ICE 85

a) MAKE SURE ICE 85 HARDWARE IS SET UP CORRECTLY

b) IF USER HARDWARE IS AVAILABLE, PLUG ICE 85 UMBILICAL CORD INTO USER 8085 SOCKET

c) ON INTELLEC TYPE

-ICE 85

ICE 85 COMMANDS

3. BORROW NEEDED RESOURCES

MEMORY

```
MAP [MEMORY] partition = | GUARDED
                         | USER [NOVERIFY]
                         | INTELLEC exp
                         |      [NO VERIFY]
```

WHERE *partition* IS ONE OR MORE <u>CONTIGUOUS</u> BLOCKS (2048 BYTES) OF MEMORY AND *exp* IS A STARTING ADDRESS IN INTELLEC MEMORY (MULTIPLE OF 2048)

EXAMPLES:

```
MAP 0 TO 2047 = GUARDED
MAP MEMORY 0 TO 8K = USER
MAP MEM 8000H TO A000H = USER NOVERIF
MAP 4000H LEN 4K = USER
MAP 4000H LEN 4K = INTELLEC 6000H
```

TO CHECK MAP STATUS:

```
MAP
```

ICE 85 COMMANDS

3. (CONTINUED)

INPUT/OUTPUT

```
MAP IO partition =  | GUARDED  |
                    | USER     |
                    | INTELLEC |
```

WHERE partition IS ONE OR MORE <u>CONTIGUOUS</u> BLOCKS (8 PORTS/BLOCK)

EXAMPLES:

```
MAP IO  Ø   TO  7   = USER
MAP IO  Ø   TO  7   = INTELLEC
MAP IO  Ø   TO  FFH = INTELLEC
MAP IO  56T TO  63T = USER
```

TO CHECK MAP STATUS:

```
MAP IO
```

ICE 85 COMMANDS

4. LOAD USER SOFTWARE

```
LOAD filename
```

WHERE filename IS THE FILE THE USER MACHINE CODE IS STORED IN.

ICE 85 COMMANDS

5a) RUN USER PROGRAM (THE GO COMMAND)

```
GO [FROM addr] | [FOREVER]
               | [TILL break cond [OR break cond2]
               |                  [OR SYØ]]
               | [TILL SYØ]
```

WHERE addr IS THE STARTING ADDRESS AND breakcondition IS

```
Stopaddr | READ
         | WRITTEN
         | EXECUTED
         | INPUT
         | OUTPUT
```

EXAMPLES:

```
GO FROM .START FOREVER
GO FROM .START TILL .TOTAL WRITTEN
GO TILL .TOTAL 2 WRITTEN
GO TILL .TOTAL 3 WRITTEN OR .MAX READ
GO TILL ..ONE #35 EXECUTED OR .MIN WRITTEN
GO TILL 35XXH EXECUTED
```

ICE 85 COMMANDS

DISPLAY/MODIFY MEMORY

```
BYTE addrange          WORD addrange
```

WHERE addrange IS EITHER A SINGLE ADDRESS OR A RANGE OF ADDRESSES

FOR INSTANCE:

```
BYTE  .MAX
BYTE  .ARRAY TO .JACK
BYTE  3ØØØH  LEN 5ØH

WORD  .MIN
WORD  .ARRAY TO .JACK
WORD  951ØH LEN  3ØØH
```

TO MODIFY

```
BYTE addr  =  val [, val]...
WORD addr  =  val [, val]...
```

WHERE addr IS A SINGLE ADDRESS AND val IS THE BYTE OR WORD VALUE TO BE STORED.

```
BYTE .MIN = 35H
BYTE .ARRAY = 2H, 37H, 1ØT, 31, 29, 2A
WORD .SAM = 4A77
WORD .WARRAY = 4B22, AA77, 251ØT, 31Q
```

ICE 85 COMMANDS

DISPLAY|MODIFY REGISTERS

Rx

WHERE x IS A REGISTERED NAME
(I.E., A, B, C, D, E, H, L, HL, DE, etc.)

```
RA
ØAH ←———————— ICE RESPONSE
                   ↓
REG
P=ØØ18H  S=Ø7FEH  A=ØFH  F=ØØH  B=ØØH  etc.
```

TO MODIFY A REGISTER:

Rx = val

WHERE val IS THE VALUE TO BE
PLACED IN THE REGISTER

```
RA = 23
RBC = 1234T
RPC = 3Ø1Ø
```

ICE 85 COMMANDS

DISPLAY INPUT PORT CONTENTS

```
PORT portnum
```

WHERE portnum IS THE INPUT PORT DESIRED

```
PORT 35
PORT 1ØT
```

MODIFY OUTPUT PORT CONTENTS

```
PORT portnum = val
```

WHERE val IS THE VALUE TO BE PLACED ON THE OUTPUT PORT

```
PORT 22H = 19T
PORT 1Ø = 1Ø
PORT 7BH = 1Ø11Ø1Ø1Y
```

ICE 85 COMMANDS

DISPLAY TRACE MEMORY

TRACE DISPLAY MODE

$$\text{TRACE} = \left|\begin{array}{l}\text{INSTRUCTIONS}\\ \text{CYCLES}\end{array}\right|$$

TRACE DISPLAY

$$\text{PRINT} \left|\begin{array}{l}\text{ALL}\\ \pm n\end{array}\right|$$

WHERE n = NUMBER OF ENTRIES
TO DISPLAY

```
PRINT  ALL
PRINT  -1Ø
```

OLDEST, NEWEST

OLDEST	MOVE TO FIRST ENTRY IN TRACE BUFFER
NEWEST	MOVE TO LAST ENTRY IN TRACE BUFFER

ICE 85 COMMANDS

5b) STEP USER PROGRAM
(THE STEP COMMAND)

```
STEP [FROM addr] [COUNT exp-10] [TILL cond1 |AND/OR| [cond1]..]
```

WHERE addr IS THE START ADDRESS
exp-10 IS A <u>DECIMAL</u> COUNT OF INSTRUCTIONS TO BE EXECUTED.
cond1 AND cond2 ARE CONDITIONS TO STOP EMULATION WHEN THEY ARE ENCOUNTERED IN THE PROPER LOGICAL COMBINATION SPECIFIED

EXAMPLES:

```
STEP FOREVER
STEP FROM .START FOREVER
STEP COUNT 1Ø
STEP FROM 32ØØ COUNT 27
STEP FROM ..ONE #35 TILL BYTE .MAX>55
STEP TILL PC=..TWO#35 OR WORD .LIMIT=17ØØ
```

ICE 85 COMMANDS

THE DUMP

WHILE STEPPING, WE WANT TO SEE WHAT IS HAPPENING IN THE REGISTERS OF THE 8085

```
| ENABLE  |
|         |   DUMP
| DISABLE |
```

FOR ENABLE DUMP, WE CAN SELECT THE AREAS OF OUR PROGRAM WHERE DUMPING WILL OCCUR WITH THE FOLLOWING:

```
ENABLE DUMP | partition |
            | CALL      |
            | JUMP      |
            | RETURN    |
```

WHERE *partition* IS ANY ADDRESS RANGE. i.e.,

```
        1ØØØH   TO   2ØØØH
  OR    386Ø    LEN  3ØØ
```

CALL INDICATED A DUMP WILL OCCUR EACH TIME A CALL INSTRUCTION IS EXECUTED. JUMP AND RETURN BOTH FUNCTION IN THE SAME WAY.

LABORATORIES

LABORATORY 2

ISIS AND INTEL SUPPLIED FILE MANIPULATION PROGRAMS

PURPOSE: TO BECOME FAMILIAR WITH INTEL SUPPLIED FILE MANIPULATION PROGRAMS.

1. TURN ON SYSTEM AND INSERT SYSTEM DISKETTE IN DRIVE Ø. PRESS RESET TO LOAD ISIS.

THE SYSTEM DISKETTE HAS SEVERAL FILES ON IT THAT WE WILL COPY ONTO THE USER DISKETTE AND THEN MODIFY. BEFORE WE CAN COPY FILES ONTO THE USER DISKETTE WE MUST INITIALIZE IT. NORMALLY THIS IS NOT NECESSARY, BUT THIS DISKETTE IS BLANK.

2. INSERT USER DISKETTE INTO DRIVE 1.

3. TYPE: IDISK :F1:mmmdd.yy ↲ ← CARRIAGE RETURN

   WHERE mmm = CURRENT MONTH (ie. SEP, MAY etc)
   dd = CURRENT DAY (ie. 27, Ø3 etc)
   yy = CURRENT YEAR (ie. 79, 80 etc)

4. WHEN THE "–" PROMPT RETURNS, THE INITIALIZATION IS COMPLETE

TO SEE WHAT FILES ARE RECORDED ON THE DISKETTE WE WILL USE THE DIRECTORY PROGRAM DIR.

Laboratory 2 (continued)

5. Type:

```
DIR 1 ↲
```

Note that there seem to be no files on the diskette yet some space has been used. The files must be "invisible" so try:

```
DIR I 1 ↲
```

Aha! There is the directory, map, label and TØ boot.

One of the most frequently used programs is COPY. To use it we will copy files from the system diskette to the user diskette.

6. Type:

```
COPY FILE1.DAT TO :F1:FILE1.DAT ↲
```

to see what happened:

```
DIR 1 ↲
```

It's there! Now what?

COPY can copy several files at a time if they all have something in common in their names.

7. Try:

```
COPY FILE2.* TO :F1: C ↲
```

Notice we don't need the destination file name if it is to be the same as the source. The C option copies the files attributes as well as the file.

Laboratory 2 (continued)

Lets see what we got.

```
DIR 1 ↲
```

COPY also has the ability to copy from one diskette to another even if only one drive is available. Let's copy a file from the system diskette to the user diskette using only drive Ø.

8. Follow carefully!

   a) Type: `COPY FILE3 TO FILEA P ↲`

   b) When "INSERT SOURCE DISKETTE" message appears we can type the carriage return since the system diskette is in place.

   c) When "INSERT DESTINATION DISKETTE" appears remove the system diskette and place the user diskette in drive Ø. Type carriage return.

   d) When "INSERT SYSTEM DISKETTE" appears place the system diskette in drive Ø. Put the user diskette back in drive 1.

Let's see if it worked

```
DIR 1 ↲
```

RENAME allows us to rename a file without altering the file in any other way.

9. Try:

```
RENAME :F1:FILEA TO :F1:FILE3 ↲
```

Laboratory 2 (continued)

Sooner or later we will want to delete a file.

10. Try:

```
        DELETE   :F1:FILE3 ↵
```

To see the effect

```
      DIR   1 ↵
```

If a file is write protected it may not be deleted or renamed.

11. Try:

```
        DELETE   :F1:FILE2.AAA ↵
```

Didn't work did it?

To unwrite protect a file use the attribute changing program.

12. Type:

```
          ATTRIB   :F1:FILE2.AAA   WØ ↵
```

Now the directory should show a non-write protected status for this file.

```
        DIR  1 ↵
```

We can now delete it

```
        DELETE   :F1:FILE2.AAA ↵
```

LABORATORY 2 (CONTINUED)

WE SHOULD TRY A CONTROLLED WILD CARD DELETE.

13. TYPE:

```
DELETE   :F1:*.*   Q ↵
```

THE Q ALLOWS US TO DECIDE ON A FILE BY FILE BASIS WETHER A FILE IS TO REMAIN OR NOT. KEEP SOME AND DELETE 2. (REMEMBER THE ISIS FILES ARE PROTECTED!) SEE THE RESULTS WITH:

```
DIR   1 ↵
```

AS A FINAL CLEAN UP:

```
DELETE   :F1:*.*   ↵
```

WILL DELETE ALL NON-WRITE PROTECTED FILES.

LABORTORY 3
CREDIT

PURPOSE: TO FAMILIARIZE THE STUDENT WITH THE CREDIT TEXT EDITOR.

1. TURN ON THE SYSTEM AND THE DISKETTE DRIVES. INSERT THE SYSTEM DISKETTE IN DRIVE Ø AND THE USER DISKETTE IN DRIVE 1. RESET THE SYSTEM.

   YOU SHOULD NOW SEE:

```
ISIS V3.4
_
```

2. TO INVOKE THE TEXT EDITOR, TYPE:

```
CREDIT    filename ↲
```

   WHERE filename IS ANY VALID FILENAME ON DRIVE 1 SUCH AS :F1:TSLØ1.FOR. (CREDIT CAN BE USED TO CREATE A FILE ON ANY DRIVE BUT WE WILL ALWAYS BE USING DRIVE 1.)

NOW YOU ARE GOING TO CREATE A TEXT FILE USING THE TEXT EDITOR. ALTHOUGH THE FILE YOU CREATE WILL BE A TEXT FILE ANY DATA OR PROGRAM CAN BE CREATED AS EASILY.

LABORATORY 3 (CONTINUED)

THE SCREEN SHOULD LOOK LIKE:

```
ISIS CRT-BASED TEXT EDITOR V1.0
NEW FILE                     xxxx BLOCKS LEFT



----
 I
 _ ← BLINKING CURSOR
```

3. TO ENTER TEXT SIMPLY TYPE AS YOU WOULD ON A TYPEWRITER. TABS ARE SET AT 8, 16, 24, 32 etc. (THIS CAN BE CHANGED.) IF A MISTAKE IS MADE, POSITION THE CURSOR UNDER THE ERROR AND TYPE THE CORRECT CHARACTER. THE FOLLOWING KEYS ARE NOW OPERATIONAL:

↑ ← → ↓ CURSOR CONTROLS	POSITION CURSOR
↑D	DELETE CHARACTER AT CURSOR POSITION.
↑C	INSERT CHARACTER AT CURRENT CURSOR POSITION. TO USE TYPE: ↑C THEN CHARACTER DESIRED ↑CA WOULD INSERT AN A

LABORATORY 3 (CONTINUED)

NOW TYPE THE FOLLOWING:

PERFECTION IN TECHNICAL RATIONALITY REQUIRES COMPLETE ↲
KNOWLEDGE OF CAUSE/EFFECT RELATIONS ↲
PLUS CONTROL OVER ALL OF THE RELEVANT ↲
VARIABLES OR CLOSURE. THEREFORE, ↲
UNDER NORMS OF RATIONALITY ↲
ORGANIZATIONS SEEK TO SEAL OFF THEIR ↲
CORE TECHNOLOGIES FROM ENVIORNMENTAL INFLUENCES. ↲

REMEMBER, IF YOU MAKE ANY MISTAKES USE THE CURSOR CONTROLS, ↑D AND ↑C. (↲ IS CARRIAGE RETURN.)

5. TO END THE EDIT AND STORE THIS BLOCK OF TEXT ON THE DISKETTE WE MUST GO INTO COMMAND MODE. TO DO THIS TYPE:

$\boxed{\text{HOME}}$

THE TOP OF THE SCREEN SHOULD NOW LOOK LIKE:

```
*



----
PERFECTION IN TECHNICAL RATIONALITY REQUIRES etc
```

LABORATORY 3 (CONTINUED)

6. TO EXIT TYPE:

```
EX ↲
```

THE TEXT EDITOR WILL UPDATE THE FILE ON DISKETTE THEN ISIS WILL RESUME CONTROL.

7. FOR LARGER ADDITIONS AND DELETIONS TO AN EXISTING FILE THERE ARE SEVERAL SCREEN MODE COMMANDS AND COMMAND MODE COMMANDS THAT MAY BE EMPLOYED. WE WILL FURTHER MODIFY THE FILE WE HAVE JUST CREATED USING THESE COMMANDS. FIRST RE-ENTER THE EDITOR WITH:

```
CREDIT filename ↲
```

WHERE filename IS THE SAME AS BEFORE.

THE SCREEN SHOULD LOOK LIKE:

```
ISIS CRT-BASED TEXT EDITOR V1.0



----
PERFECTION IN TECHNICAL RATIONALITY REQUIRES COMPLETE ↑
KNOWLEDGE etc.
```

LABORATORY 3 (CONTINUED)

8. TO ENTER A LARGE BLOCK OF TEXT IN THE TEXT USE ↑A text ↑A. IN THIS CASE LET'S ENTER A LINE OF TEXT.

a) POSITION THE CURSOR UNDER THE P OF PLUS.

b) TYPE ↑A. NOTICE THE REST OF THE FILE "DISAPPEARS"

c) NOW TYPE:

THE QUICK BROWN FOX JUMPED TOO HIGH ↲

IF A TYPING MISTAKE IS MADE IT CAN BE CORRECTED WITH THE RUBOUT AND ↑X COMMANDS. THE CURSOR CONTROLS WILL NOT WORK INSIDE A ↑A INSERT.

d) TO END THE INSERT TYPE ↑A. THE REST OF THE FILE SHOULD REAPPEAR.

9. TO REMOVE A LARGE BLOCK OF TEXT USE ↑Z. WE SHALL REMOVE THE NEXT TO LAST LINE IN THIS MANNER.

a) POSITION THE CURSOR UNDER THE O OF <u>O</u>RGANIZATIONS. TYPE ↑Z.

b) MOVE THE CURSOR UNDER THE ↑ AT THE END OF THE SAME LINE.

c) TYPE A SECOND ↑Z. THE LINE SHOULD DISAPPEAR.

SO FAR, WITH THE EXCEPTION OF THE EXIT COMMAND, WE HAVE REMAINED IN SCREEN MODE. CREDIT HAS MANY POWERFUL COMMANDS THAT ARE EMPLOYED IN COMMAND MODE. .SINCE THERE ARE SO MANY WE WILL ONLY TRY A FEW. THROUGHOUT THE REST OF THE WEEK YOU SHOULD TRY ALL OF THEM.

Laboratory 3 (continued)

10. Cursor movement.

The cursor is almost at the end of our file. We can position it to the beginning with the cursor controls, but lets try command mode. Type

```
HOME
```

to get to command mode.

Now type:

```
JTT ↵
```

To return to screen mode type:

```
↑V ↵
```

Notice the cursor is now at the top of the file.

We can move the cursor to the end of the file with the JTE command like:

```
Type      HOME
then      JTE ↵
finally   ↑V ↵
```

The cursor should be at the end of the file. Remember, it is not always necessary to go back to screen mode after executing a command mode command. In fact it is possible to have an entire editing session in the command mode!

Laboratory 3 (continued)

11. As long as we are in screen mode let's add some more text to the file. Continue with the following

```
The NOAA report calls for a ↲
national policy to recognize ↲
that aquaculture is in the ↲
national interest and to ↲
encourage private farming of ↲
fish and shell fish. ↲
If Congress eventually ↲
includes aqua culture in ↲
such appropriations, ↲
the emphasis probably ↲
will go to research ↲
on unromantic species ↲
like talapia and carp. ↲
```

12. In the command mode it is possible to insert, delete, find substitute, move and copy text. We have already used command mode commands to move the cursor to the extremes of the file. Since most of these commands can also be accomplished in screen mode, we will concentrate on some things not easily done with screen mode. First, mass substitution. CREDIT provides two ways, with and without query. A mass substitution with query goes as follows:

a) Move the cursor to the top of the file.

$\boxed{\text{HOME}}$ JTT ↲

b) Replace all occurances of "to" with "xxx"

```
!<SQ/TO/xxx/>↲
```

For each query respond with Y for yes and N or carriage return for no. If you want to quit the command before you finish type ESC (escape).

ESC will abort any CREDIT command!

13. Last, but not least is the block move and block copy of text. The block move removes the text from the source area while the block copy does not. Let's try a block copy. The block move is identical except for the final command.

a) Set a tag at the beginning of the text to be copied. Use the screen mode to position the cursor under the "P" of perfection in the first line. Type HOME to get to command mode. To set the tag type:

```
TS4↲
```

b) Go back to screen mode (↑V) and move the cursor to the ↑ after "influences" on the sixth line. Type HOME to get to command mode and set the tag by typing:

```
TS5↲
```

LABORATORY 3 (CONTINUED)

c) THE FIRST TWO TAGS DEFINE THE BLOCK OF TEXT TO BE COPIED. WE NOW MOVE THE CURSOR TO THE PLACE WHERE THE TEXT IS TO BE INSERTED. TO DO THIS GO BACK TO SCREEN MODE (↑V) AND MOVE THE CURSOR TO THE "I" IN "IF CONGRESS EVENTUALLY"

d) LAST STEP. BACK TO COMMAND MODE (HOME) AND TYPE:

```
XC T4,T5 ↵
```

e) TO SEE THE RESULTS GO TO THE BEGINNING OF THE FILE WITH JTT THEN GO TO SCREEN MODE (↑V). NOTICE THAT THE ENTIRE FILE WILL NO LONGER FIT ON THE SCREEN. TO SEE THE NEXT PAGE:

↑N (NEXT PAGE)

TO GO BACK TO A PREVIOUS PAGE:

↑P (PREVIOUS PAGE)

YOU HAVE A CREDIT USERS MANUAL TRY MORE OF THE COMMANDS LISTED WHEN YOU HAVE TIME.

LABORATORY 4

FORTRAN

PURPOSE: TO WRITE, TRANSLATE, LINK, LOCATE AND RUN
A FORTRAN PROGRAM.

THIS LABORATORY CAN BE APPROACHED ON THREE LEVELS:

A - MANAGEMENT OVERVIEW

B - SYSTEM USER, NOVICE PROGRAMMER

C - EXPERIENCED PROGRAMMER/ENGINEER

A.

1. FOLLOW STEPS 1 & 2 OF LABORATORY 2.
2. COPY THE FILE CHECK.FOR FROM DRIVE Ø TO DRIVE 1.
3. TRANSLATE THE PROGRAM WITH:

```
FORT80   :F1:CHECK.FOR ↲
```

4. GET A LISTING OF THE TRANSLATED PROGRAM:

```
COPY   :F1:CHECK.LST   TO   :LP: ↲
```

(MAKE SURE THE LINE PRINTER IS ATTACHED TO YOUR SYSTEM WHEN YOU DO THIS!)

5. GET A COPY OF THE LINK COMMAND FILE:

```
COPY   :F1:FORTL   TO   :F1: ↲
```

LABORATORY 4 (CONTINUED)

6. RUN THE COMMAND FILE TO LINK YOUR PROGRAM:

```
SUBMIT  :F1:FORTL(:F1:CHECK) ↵
```

7. LOCATE THE FINAL ASSEMBLY:

```
LOCATE  :F1:CHECK.LNK ↵
```

8. RUN THE PROGRAM AFTER READING THE LISTING (TO SEE WHAT IT DOES!)

```
:F1:CHECK ↵
```

B.

1. RATHER THAN COPY A PROGRAM FROM THE SYSTEM DISKETTE, THE SYSTEM USER OR NOVICE PROGRAMMER SHOULD CREATE A FILE USING CREDIT. THIS PROGRAM CAN BE COPIED FROM ONE OF THE PROGRAMS SHOWN IN LECTURE OR ONE OF THE PROGRAMS GIVEN IN THE APPENDIX.

2. ONCE THE FILE IS CREATED, FOLLOW STEPS 3 THRU 8 OF SECTION A. REMEMBER TO USE YOUR FILE NAME INSTEAD OF :F1:CHECK.FOR etc!

LABORATORY 4 (CONTINUED)

C.

1. THE EXPERIENCED PROGRAMMER WILL HAVE ENOUGH TIME TO CREATE THE PROGRAM DESCRIBED IN THIS SECTION.

LEVEL I - BALANCE A CHECKBOOK.

LEVEL II - a) TAKE UP TO 100 ENTRIES. EACH ENTRY SHOULD TAKE PLACE AS FOLLOWS:

```
ENTER C(CHECK), D(DEPOSIT) OR Q(QUIT)       D
ENTER DEPOSIT NUMBER (4 DIGITS MAX)         1375
ENTER DEPOSIT AMOUNT (UP TO XXXX.XX)        379.52

ENTER C(CHECK), D(DEPOSIT) OR Q(QUIT)       C
ENTER CHECK NUMBER (4 DIGITS MAX)           1799
ENTER CHECK AMOUNT (UP TO XXXX.XX)          39.40

ENTER C(CHECK), D(DEPOSIT) OR Q(QUIT)       Q
```

b) AFTER TAKING THE DATA THE PROGRAM SHOULD SORT THE TRANSACTIONS BY CHECK OR DEPOSIT NUMBER THEN CREATE A BALANCE SHEET.

c) The program should then print the balance sheet on the line printer as follows:

```
              SORTED  CHECKS
 CHECK NO.          AMOUNT            BALANCE
DEPOSIT NO.

   101              3275.00           3275.00
   103                50.00           3225.00
   112                70.00           3155.00

                    etc.
```

Level III - The programmer should prepare a flowchart then write the program from the flowchart. Compile, link,locate and run the program with steps 3 thru 8 of section A.

(Remember to use your own file name in place of :F1:CHECK.FOR!)

Laboratory 4 (continued)

Other useful programs the novice or experienced programmer/engineer might attempt.

I. Direct Reduction Loan Amortization Schedule

Program would calculate a table of interest paid, payment to principle and present value of mortgage. As an option it can also find yearly accumulated interest for tax purposes.

Program should ask (thru the console) for

a) Monthly Payment

b) Yearly Interest

c) Beginning Principle

Program should then produce the following table on the line printer.

Payment No.	Period Interest	Payment to Principle	Remaining Principle	Yearly Interest
1	175.00	25.00	29975.00	
2	174.85	25.15	29949.85	
3	174.71	25.29	29924.56	
⋮	⋮	⋮	⋮	
12	173.35	26.25	29690.19	2090.17
⋮	⋮	⋮	⋮	⋮

LABORATORY 4 (CONTINUED)

THE EQUATIONS TO CALCULATE THESE VALUES ARE:

$$\text{PERIOD INTEREST}_{K+1} = i \times \text{PRINCIPLE}_{K}$$

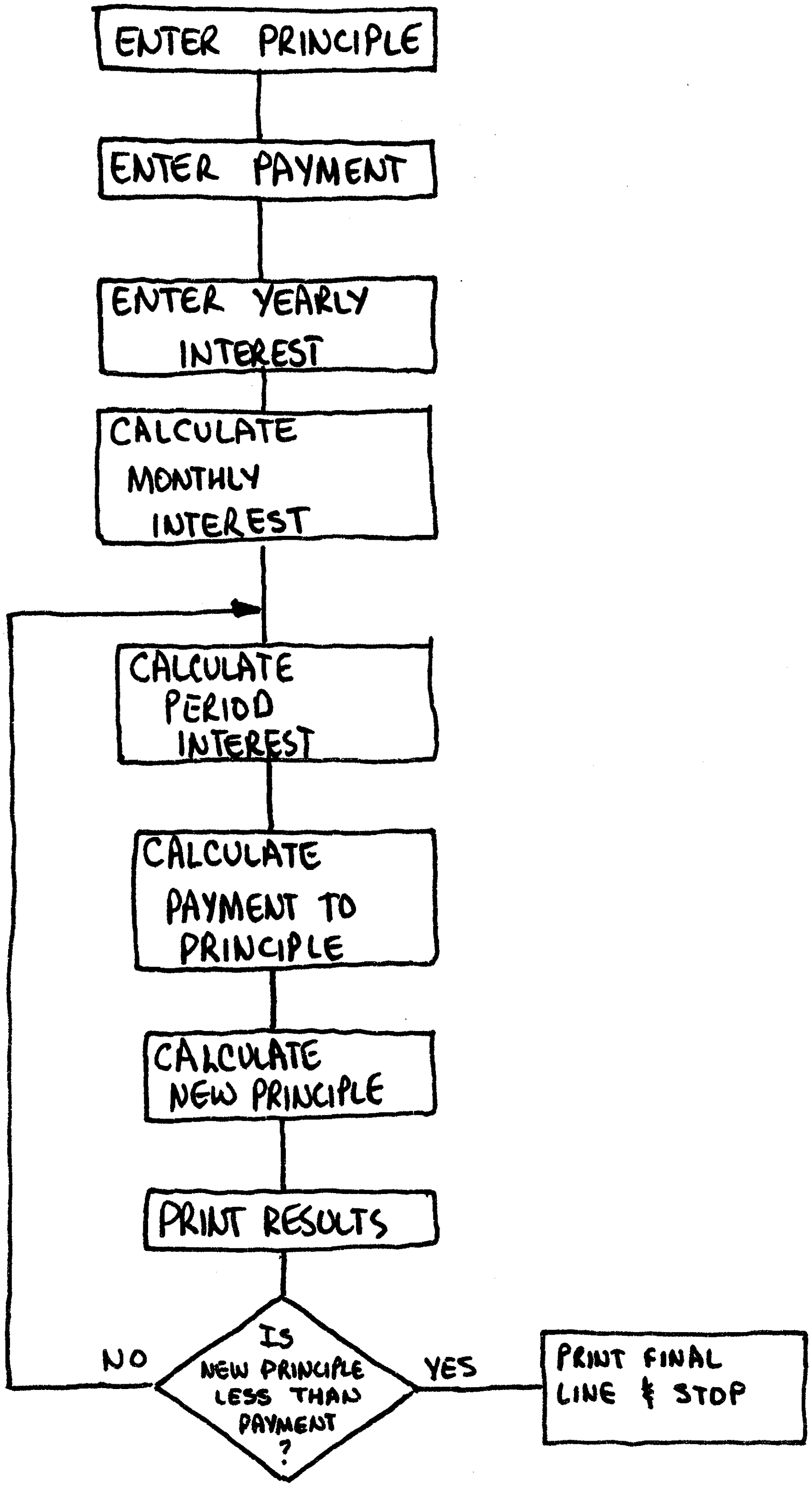
WHERE i = PERIODIC INTEREST = $\dfrac{\text{YEARLY INTEREST}}{12}$

$$\text{PAYMENT TO PRINCIPLE}_{K+1} = \text{PAYMENT} - \text{PERIOD INTEREST}_{K+1}$$

$$\text{(NEW)PRINCIPLE}_{K+1} = \text{PRINCIPLE}_{K} - \text{PAYMENT TO PRINCIPLE}_{K+1}$$

THE FLOW CHART FOR THIS PROGRAM IS ON THE NEXT PAGE

LABORATORY 4 (CONTINUED)


ENTER PRINCIPLE
ENTER PAYMENT
ENTER YEARLY INTEREST
CALCULATE MONTHLY INTEREST
CALCULATE PERIOD INTEREST
CALCULATE PAYMENT TO PRINCIPLE
CALCULATE NEW PRINCIPLE
PRINT RESULTS
IS NEW PRINCIPLE LESS THAN PAYMENT ?
NO
YES
PRINT FINAL LINE & STOP

LABORATORY 4 (CONTINUED)

II DOLLAR BILL CHANGER

WRITE A PROGRAM WHICH WILL PROVIDE CHANGE FOR A DOLLAR FOR ANY ITEM PURCHASED THAT COSTS $1.00 OR LESS. PRINT OUT THE UNIT OF CHANGE (HALF, QUARTER, DIME, NICKLE OR PENNY) PROVIDED. ALWAYS DISPENSE THE BIGGEST DENOMINATION POSSIBLE. FOR EXAMPLE 37 CENTS INCHANGE WOULD RESULT IN

1 - QUARTER
1 - DIME
2 - PENNYS

LABORATORY 5

ICE85

PURPOSE: TO ACQUAINT THE STUDENT WITH SOME OF THE FACILITIES OF ICE-85.

1. COPY THE DEMONSTRATION CODE AND SUBMIT FILE TO THE USER DISKETTE

```
COPY ICETST.FOR TO :F1:
COPY ICELL.CSD TO :F1:
```

2. COMPILE (AND DIRECTLY PRINT THE LISTING!)

```
FORT80 :F1:ICETST.FOR DEBUG PRINT(:LP:)
```

3. LINK WITH FORTRAN LIBRARIES. NOTICE THE LIBRARIES THAT ARE BEING USED.

```
SUBMIT :F1:ICELL(:F1:ICETST)
```

4. OUR "TARGET" SYSTEM WILL HAVE MEMORY IN THE FOLLOWING PATTERN:

ROM	Ø TO $7FF_{16}$
RAM	2ØØØ TO 2ØFF
I/O	PORTS 2Ø TO 26

LABORATORY 5 (CONTINUED)

4. (CONTINUED)

```
LOCATE :F1:ICETST.LNK &
       CODE(Ø) DATA(2ØØØH) STACK(2BØØH) &
       MAP LINES SYMBOLS PRINT(:LP:)
```

5. NOW WE ARE READY FOR ICE-85

```
ICE85
```

6. FIRST WE MUST BORROW RAM FROM THE DEVELOPMENT SYSTEM.

```
MAP Ø TO 7FF = INT 7ØØØ
```

MEANS BORROW 2048_{10} BYTES OF INTELLEC MEMORY (7ØØØ TO 77FF) AND CALL IT Ø TO 7FF USER

```
MAP 2ØØØ TO 27FF = USER
```

ADDRESSES 2ØØØ TO 27FF ACTUALLY EXIST IN THE USER SYSTEM.

7. INPUT/OUTPUT RESOURCES ALREADY EXISTS IN THE USER SYSTEM SO WE USE THEM

```
MAP IO 2Ø TO 2F = USER
```

LABORATORY 5(CONTINUED)

8. NOW WE HAVE THE NEEDED RESOURCES, LOAD THE PROGRAM.

```
LOAD :F1:ICETST
```

9. LET'S SEE IF ICE85 REALLY KNOWS ABOUT OUR SYMBOL TABLE.

```
SYMBOLS
```

AHA! THEY ARE THERE!

10. NOW TRY RUNNING THE PROGRAM (AFTER READING)

```
GO FROM #2
```

TRY FLICKING THE TINY SWITCHES ON THE SDK TO TRY TO CHANGE THE LITE PATTERN ON THE TINY LEDS.

11. TO STOP EXECUTION PRESS $\boxed{\text{ESC}}$ (ESCAPE) ON THE DEVELOPMENT SYSTEM KEYBOARD.

12. WE CAN STOP EXECUTION ON A MEMORY WRITE FOR INSTANCE WHEN THE PROGRAM SETS THE INPUT DATA INTO VAL.

LABORATORY 5 (CONTINUED)

12. (CONTINUED)

TO DO THIS TYPE

```
GO FRO #2 TILL .VAL WRITTEN
```

13. WHAT WAS JUST WRITTEN INTO VAL?

```
BYTE .VAL
```

AH, THERE IS THE SWITCH DATA AS WE EXPECTED.

14. LET'S SEE IF ANYTHING CHANGES WITH DIFFERENT SWITCH INPUT. SET THE SWITCHES TO SOME VALUE. NOW TRY;

```
GO FRO #2 TILL .VAL WRITTEN
```

TO SEE THE DATA AGAIN

```
BYTE .VAL
```

15. NOW LET'S CONTINUE TILL WE DO THE OUTPUT

```
GO TILL # 12 EXEC
```

DID THE LITES CHANGE? IF NOT WE CAN FORCE DATA INTO THE PROGRAM

LABORATORY 5 (CONTINUED)

16. SELECT A VALUE OF VAL THAT SHOULD TURN THE OTHER PATTERN OF LITES ON. (3ØH ⇒ ØF$_{16}$) (ØH ⇒ 55$_{16}$). SET .VAL TO THAT VALUE:

```
BYTE .VAL = ____ ← SELECTED VALUE
```

17. NOW TRY PART OF THE PROGRAM.

```
GO FROM #5 TILL #12 EXEC
```

HOW DID THE LITES DO?

18. WE CAN SINGLE STEP THE PROGRAM:

```
STEP FROM #2 TILL BYTE .VAL >3ØH
```

NOTE HOW SLOWLY THE SYSTEM RESPONDS!

19. LETS WATCH THE TRACE FEATURE:

```
GO FROM #2 TILL #12 EXEC
PRINT -2Ø
```

WHAT YOU SEE IS AN INSTRUCTION BY INSTRUCTION "RECORDING" OF YOUR PROGRAM RUNNING

LABORATORY 5(CONTINUED)

20. WOULD YOU LIKE "HARD COPY"?

```
LIST :LP:
```

NOW REPEAT

```
GO FROM #2 TILL #12 EXEC
PRINT -20
```

21. TO EXIT ICE85

```
EXIT
```

22. FOR FUN FOLLOW:

```
ICE85
MAP ø LEN 2K = INT 7øøø
MAP IO 2ø TO 2F = USER
LOAD     WEIRD
GO FROM .START
```

LABORATORY 6

PROGRAMMING AIDS

PURPOSE: TO BECOME FAMILIAR WITH THE $INCLUDE AND SUBMIT FACILITIES OF INTEL SUPPLIED SOFTWARE.

A. MANAGEMENT OVERVIEW

1. COPY THE PROGRAM SUBSOR.FOR FROM THE SYSTEM DISKETTE TO YOUR USER DISKETTE.

2. COMPILE THE PROGRAM.

3. PRINT THE LISTING. NOTICE, THE COPYRIGHT NOTICE IS INCLUDED.

4. TO LINK AND LOCATE, USE SUBMIT AS FOLLOWS
   a) COPY FORCLL TO :F1:
   b) SUBMIT :F1: FORCLL(:F1:MAIN, :F1:SUBSOR)

5. WHEN THE MESSAGE

```
SUBMIT RESTORE ...
```

APPEARS THE SUBMIT IS FINISHED AND THE RESULTING PROGRAM -MAIN- CAN BE RUN WITH:

```
:F1:MAIN
```

LABORATORY 6 (CONTINUED)

B. SYSTEM USER, NOVICE PROGRAMMER

1. CREATE THE FOLLOWING FILE (CAREFULLY) USING CREDIT.

```
COLUMN     COLUMN     COLUMN
  1          8          16

       SUBROUTINE SORT (M,COUNT)
C   SORT AN ARRAY OF REAL DATA IN ASCENDING ORDER
C   COPYRIGHT NOTICE FOLLOWS
$INCLUDE(COPYRI.GHT)
C   PARAMETER DEFINITIONS
C   M- TABLE TO BE SORTED
C   COUNT - NUMBER OF ELEMENTS
       INTEGER COUNT
       REAL  M
       DIMENSION  M(COUNT)
C   LOCAL VARIABLES
       INTEGER  INDEX, NEXLAS
       LOGICAL  MORE
       REAL     TEMP
```

LABORATORY 6 (CONTINUED)

2. FOLLOW STEPS 2 THRU 5 OF THE A SECTION.
3. TO GET AN IDEA OF WHAT THE FORCLL.CSD FILE LOOKS LIKE, PRINT IT ON THE LINE PRINTER. NOTICE THE USE OF PARAMETERS.
4. CREATE A "SUPER" SUBMIT THAT
   - a) COMPILES A FILE (PASSED BY PARAMETER)
   - b) LINKS & LOCATES IT.
   - c) PRINTS THE LISTING.
5. COPY THE FILE TEST.FOR ONTO YOUR USER DISKETTE FROM THE SYSTEM DISKETTE
6. TRY YOUR "SUPER" SUBMIT ON THE NEW FILE :F1:TEST.FOR.

LABORATORY 6 (CONTINUED)

C. EXPERIENCED PROGRAMMER, ENGINEER

1. CREATE A SUBMIT FILE WHICH WILL

   a) LINK TWO FORTRAN FILES TO THE LIBRARIES

   b) LOCATE THE RESULT

2. COMPILE THE SUBROUTINE SUBFOR.FOR AFTER ENTERING IT (OR COPYING IT FROM THE SYSTEM DISKETTE IF THERE IS NO TIME)

3. USE YOUR SUBMIT FILE TO LINK & LOCATE THE RESULT.

4. MODIFY THE FILE OF YOUR SUBMIT TO MAKE IT PAUSE (AND MAYBE RING THE CONSOLE BELL) BETWEEN THE LINK & LOCATE STEP. TRY IT AGAIN.

LABORATORY 6 (CONTINUED)

```
COLUMN COLUMN COLUMN
  1      8      16

C  PERFORM BUBBLE SORT
       NEXLAS = COUNT -1
5      MORE = , FALSE.
       DO 30 INDEX= 1,NEXLAS
       IF( M(INDEX) .GT. M(INDEX+1) ) THEN
              TEMP = M(INDEX)
              M(INDEX)   = M(INDEX +1)
              M(INDEX +1) = TEMP
              MORE = .TRUE.
       ENDIF
30     CONTINUE
       IF (MORE) THEN  GO TO 5
       ENDIF
C  SORT IS FINISHED RETURN
       RETURN
       END
```

LABORATORY 7

UPM

PURPOSE: TO ACQUAINT THE STUDENT WITH THE PROCESS OF COMPILING, TESTING AND TRANSFERRING A PROGRAM INTO A ROM FOR EXECUTION.

1. MODIFY THE ICETST PROGRAM IN THE FOLLOWING MANNER:

   ```
   CREDIT ICETST.FOR TO :F1:UPMTST.FOR
   ```

   NOW CHANGE THE STATEMENT

   ```
   " IF (TSTVAL .GT. 1ØØ) THEN "
   ```

   TO

   ```
   IF (TSTVAL .GT. 1Ø) THEN
   ```

   USING CREDIT COMMANDS.

2. COMPILE THE PROGRAM. DON'T FORGET THE DEBUG AND CODE OPTIONS!

LABORATORY 7 (CONTINUED)

3. LINK THE NEW FILE WITH

```
SUBMIT :F1:ICELL(:F1:UPMTST)
```

4. THE TARGET SYSTEM WILL HAVE

ROM 8ØØH TO ØFFFH
RAM 2ØØØH TO 2ØFFH
I/O 2ØH TO 23H

THE LOCATE STEP WILL THUS BE:

```
LOCATE :F1:UPMTST.LNK &
   CODE(8ØØH) DATA(2ØØØH) STACK(2ØBØH) &
   MAP SYMBOLS LINES PRINT(:LP:)
```

5. TEST THE RESULT WITH ICE 85

```
ICE85

MAP 800 = INT 7000
MAP 2000 = USER
MAP IO 20 = USER
```

WE HAVE RAM AND IO ON THE BOARD, BUT OUR ROM SOCKET IS EMPTY.

LABORATORY 7(CONTINUED)

6. LOAD THE PROGRAM

   LOAD :F1:UPMTST

7. RUN IT

   GO FROM #2

   IS IT WORKING? (IT SHOULD!)

8. ESCAPE FROM ICE EMULATION AND EXIT FROM ICE.

9. MOVE TO A SYSTEM WITH A PROM MAPPER.

10. GET AN 8755A FROM THE INSTRUCTOR. ERASE IT ACCORDING TO HIS DIRECTIONS.

11. TURN ON MDS SYSTEM AND UNIVERSAL PROM MAPPER.

12. RESET UPM (RESET BUTTON ON UPM ITSELF)

13. CALL UP THE UPM SOFTWARE WITH

    UPM

LABORATORY 7 (CONTINUED)

14. PLACE THE 8755A IN THE PROM MAPPER SOCKET WITH THE NOTCH ON THE END OF THE 8755A MATCHING THE NOTCH IN THE SOCKET. (USUALLY UP)

15. THE PROM MAPPER SOFTWARE ASKS FOR TYPE. YOU RESPOND WITH:

TYPE * 8755

16. YOU TELL THE SOFTWARE WHICH SOCKET YOU ARE USING WITH:

SOCKET = 2

17. NOW CHECK THE PROM FOR FULL ERASURE.

TRANSFER FROM ϕ TO 7FFH

DISPLAY FROM ϕ TO 1ϕϕH

THE TRANSFER READS THE PROM. THE DISPLAY DISPLAYS THE DATA. IT SHOULD BE ALL ϕFFH.

LABORATORY 7 (CONTINUED)

18. NOW LOAD THE OBJECT CODE INTO MEMORY

```
READ OBJECT FILE :F1:UPMTST INTO Ø
```

19. DISPLAY THE FIRST 10 LOCATIONS.

```
DISPLAY FROM 8ØØH TO 8Ø9H
```

DOES THAT LOOK LIKE THE LISTING ?
(REMEMBER SOME OF YOUR ADDRESSES WEREN'T
FILLED IN IN THE LISTING.)

20. OK. NOW PROGRAM THE PROM.

```
PROGRAM FROM 8ØØH TO ØFFFH START Ø
```

21. NOW WAIT. IT TAKES ABOUT 2 MINUTES. THE PROGRAM LITE WILL BE ON DURING THIS PERIOD.

22. CHECK THE RESULTS

```
COMPARE FROM 8ØØH TO 87ØH
```

23. REMOVE THE PROM FROM THE SOCKET.

LABORATORY 7 (CONTINUED)

24. MOVE TO A SYSTEM WITHOUT A PROM MAPPER TO GIVE SOMEONE ELSE A CHANCE.

25. TURN OFF THE POWER SUPPLY TO THE SDK-85. AND SWITCH THE ZIF SOCKET TO _OFF_.

    INSERT THE PROM WITH THE NOTCH FACING THE SAME DIRECTION AS THE OTHER LARGE CHIPS ON THE BOARD. SWITCH THE SOCKET TO _ON_. (MAKE SURE THE ROM IS _FULLY_ INSERTED.)

26. TURN ON THE SDK-85.

27. BRING UP ICE85 AND MAP ALL MEMORY TO THE SDK-85 AS WELL AS ALL IO.

```
ICE85

MAP Ø TO FFFF = USER

MAP IO Ø TO FF = USER
```

28. LOAD THE SYMBOL TABLE _ONLY_ SO WE CAN STILL USE SYMBOLIC DEBUGGING.

```
LOAD :F1:UPMTST NOCODE
```

LABORATORY 7 (CONTINUED)

29. NOW TRY IT OUT:

GO FROM #2

OK? OK!

30. ONCE YOU HAVE SATISFIED YOURSELF THAT YOU STILL HAVE FULL ICE 85 CAPABILITIES (GO, STEP, DISPLAY, ETC.)

RETURN THE 8755A TO THE INSTRUCTOR!