Preface

This student Study Guide is intended for participants in the Intellec Development Systems Operations Workshop.

Table of Contents

Preface
Chapter 1: System Overview and Setup
Chapter 2: Normal Use; Diskette Care
Chapter 3: Introduction to Fortran-80
Chapter 4: ICE-85
Chapter 5: Maintenance
Chapter 6: Programming Aids
Chapter 7: PROM Programming
Appendix
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
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 INTERLEC
DEVELOPMENT SYSTEM
fit into
the development process?
Block Diagram
of
System Hardware

MULTIBUS SLOTS USED

1
0
1
2
2 or 3
Block Diagram of Integrated Processor Board

- Master System Processor (8080 A-Z)
- Memory 32K RAM 10K Rom*

* Switch selected (diagnostic usage)

IPB Local Bus

- Input/Output System #1
  - Teletype
  - External CRT
  - Front Panel

- Input/Output System #2 (8041)
  - Line printer
  - Paper tape reader
  - Paper tape punch
  - PROM programmer
SOFTWARE OVERVIEW

![Diagram of software components]

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 (3)
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

- Read/Write Access Slot
- Stress Relief Notch
- Write Protect Notch
- Drive Spindle Hole
- Index Hole (in Magnetic Medium)
- Index Access Hole
- Outer Jacket
- Liner
- Magnetic Medium

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Single Density</th>
<th>Double Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Tracks</td>
<td>77</td>
<td>77</td>
</tr>
<tr>
<td>#Sectors Per Track</td>
<td>26</td>
<td>52</td>
</tr>
<tr>
<td>#Bytes Per Sector</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Total # Sectors</td>
<td>2002</td>
<td>4004</td>
</tr>
<tr>
<td>Total # Bytes</td>
<td>256,256</td>
<td>512,512</td>
</tr>
</tbody>
</table>
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 DRESSING POWER SWITCH.

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

3. INSERT SYSTEM DISKETTE INTO DRIVE #1; LABEL TO THE LEFT, READ/WRITE SLOT HORIZONTAL AS SHOWN. CLOSE DRIVE DOOR.

4. PRESS RESET.

   SYSTEM HAS FINISHED LOADING WHEN

   **ISIS V3.4**

   APPEARS ON THE SCREEN.
System Memory
With
ISIS Loaded

Monitor (ROM) 64K

48K

32K

Non-Resident

ISIS.CLI 16K

Buffer Space

ISIS.BIN

3680H

3180H
TYING ON A DEVELPMENT SYSTEM

SPECIAL CHARACTERS EASE TYING TASK:

**Rubout** - \(\text{(Ro)}\) deletes last character and echoes it on the screen

I.E. Typed: JACQUE \(\text{Ro} \quad \text{Ro} \quad \text{Ro}\)

Screen: JACQUE E U Q

At this point memory contains JAC

\(\uparrow \text{x} - (\text{Control-x})\) deletes entire line

I.E. Typed: JACK AND JILL WENT \(\uparrow \text{x}\)

Screen: JACK AND JILL WENT #

\(\text{← blinking cursor}\)

\(\uparrow \text{r} - (\text{Control-r})\) reviews current contents of line being typed (corrections already made)

I.E. Typed: JACK AND JILL FL NE EN WEN

Screen: JACK AND JILL FL NE EN WEN

JACK AND JILL WENT \(\text{← blinking cursor}\)
Output Control

Two More Special Characters to Control System Output 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.

-:FI: 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) - :F0:, :F1:, ..., :F5:
- Root - 1 to 6 characters (A-z, 0-9)
  A, JACK, 0123, A3BZ, etc.
- Extension - 1 to 3 characters
  (A-z, 0-9)

OK ☺️
:F1: JACK.PLH
:F4: J32. FOR
Editor
SAMMY.ASM

Illegal 😞
:F6: JACK.PLH
:F4: . J32. FOR
Editor
:F4: PRESENTR

Notes:
1. If no drive # is present, system assumes :F0:
2. Extension is optional (but recommended)
Sometimes we want to work with groups of files which have similar names. For instance:

:FI: JACK. FOR  
:FI: JACK. LST  
:FI: JACK. OBJ  
:FI: JACK. LNK

Are all related files with the common root jack. We can treat them as a group with

:FI: 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.TFO - "" "" ""
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
  I DISK

- Cross Reference Listing
  Generation (Inter-Program)
  ASX REF
INTEL SUPPLIED PROGRAMS

-DIR-

WHAT - THE DIR PROGR AM 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 0 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 :FI: MAR12.79 s (a system diskette)

IDISK :FI: 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 TEOFL. ASM
DELETE SAM 1.*
DELETE *.* Q
INTEL SUPPLIED PROGRAMS

- RENAME -

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

RENAM E filename1 TO filename2

EXAMPLES:

RENAME    Tom to Jerry
RENAME    :FI: Prob1.LST to :FI: Grad. LST
RENAME    :F2: Jack.* to :F2: Jill.*

NOTE: BOTH FILES MUST BE ON THE SAME DISKETTE
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.
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

<table>
<thead>
<tr>
<th>KEY</th>
<th>EFFECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cursor Controls</td>
<td>Position Cursor 1 space or line. Use repeat key for movement over long distances</td>
</tr>
<tr>
<td>HOME</td>
<td>Change to command mode</td>
</tr>
</tbody>
</table>

Letter, number, or special character replaces character at current cursor position

Note: The cursor controls and home key work only with Credit!

Escape (Esc) will cancel any command.
## Screen Mode Commands

<table>
<thead>
<tr>
<th>Key</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>↑C</td>
<td>Insert a character at current cursor position. Example: ↑CA would insert an A.</td>
</tr>
<tr>
<td>↑D</td>
<td>Delete the character at the current position.</td>
</tr>
<tr>
<td>↑A (string of characters) ↑A</td>
<td>Insert a group of characters. (String may be any length) in this mode Rubout and ↑X are functional.</td>
</tr>
<tr>
<td>↑Z</td>
<td>Delete a string of characters (position cursor at beginning, type ↑Z; position cursor at end, type ↑Z).</td>
</tr>
</tbody>
</table>
CREDIT

Screen Mode Commands

The screen moves as text is entered.

a)

<table>
<thead>
<tr>
<th>text</th>
<th>text</th>
<th>text</th>
<th>text</th>
<th>text</th>
</tr>
</thead>
</table>

b)

<table>
<thead>
<tr>
<th>text</th>
<th>text</th>
<th>text</th>
<th>text</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>text</td>
<td>text</td>
<td>text</td>
<td>text</td>
</tr>
</tbody>
</table>
The screen can be positioned with

\[ \uparrow N \text{ (Next Screen)} \]
\[ \uparrow P \text{ (Previous Screen)} \]

<table>
<thead>
<tr>
<th>Previous Screen</th>
<th>Current Screen Position</th>
<th>Next Screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>text 1</td>
<td>text 2</td>
<td>text 14</td>
</tr>
<tr>
<td>text 2</td>
<td>text 3</td>
<td>text 15</td>
</tr>
<tr>
<td>text 3</td>
<td>text 4</td>
<td>text 16</td>
</tr>
<tr>
<td>text 4</td>
<td>text 5</td>
<td>text 17</td>
</tr>
<tr>
<td>text 5</td>
<td>text 6</td>
<td>text 18</td>
</tr>
<tr>
<td>text 6</td>
<td>text 7</td>
<td>text 19</td>
</tr>
<tr>
<td>text 7</td>
<td>text 8</td>
<td>text 20</td>
</tr>
<tr>
<td>text 8</td>
<td>text 9</td>
<td></td>
</tr>
<tr>
<td>text 9</td>
<td>text 10</td>
<td></td>
</tr>
<tr>
<td>text 10</td>
<td>text 11</td>
<td></td>
</tr>
<tr>
<td>text 11</td>
<td>text 12</td>
<td></td>
</tr>
<tr>
<td>text 12</td>
<td>text 13</td>
<td></td>
</tr>
<tr>
<td>text 13</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CREDIT

Screen Mode Commands

The screen can be positioned with

↑V (Set position of screen according to cursor)

---

**BEFORE**

<table>
<thead>
<tr>
<th>Text 1</th>
<th>Text 2</th>
<th>Text 3</th>
<th>Text 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text 5</td>
<td>Text 6</td>
<td>Text 7</td>
<td>Text 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text 9</td>
<td>Text 10</td>
<td>Text 11</td>
<td>Text 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SCREEN POSITION**

**AFTER**

<table>
<thead>
<tr>
<th>Text 1</th>
<th>Text 2</th>
<th>Text 3</th>
<th>Text 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text 5</td>
<td>Text 6</td>
<td>Text 7</td>
<td>Text 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text 9</td>
<td>Text 10</td>
<td>Text 11</td>
<td>Text 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**CURSOR**
CREDIT

Command Mode Commands

To get to Command Mode: \[ \text{HOME} \]

To get back to Screen Mode: \[ \uparrow \mathbf{V} \]

Help Command

\[ \mathbf{H} \] Credit will print a menu of commands

Cursor Positioning

\[ \mathbf{L} \] Move by the line

\[ \mathbf{L} = \mathbf{L1} \] Move forward 1 line

\[ \mathbf{L3} \] Move forward 3 lines

\[ \mathbf{L-17} \] Move backward 17 lines

\[ \mathbf{JTT} \] Jump to top of file

\[ \mathbf{JTE} \] Jump to end of file
CREDIT
Command Mode Commands

Text Altering
(can use Rubout and 1X)

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.

2-31
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 OCCURRENCE
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

<table>
<thead>
<tr>
<th>Difficulty to People</th>
<th>Language</th>
<th>Difficulty to 8080</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Problem</td>
<td>English, Spanish, Greek, etc. (Are you kidding?)</td>
<td></td>
</tr>
<tr>
<td>Understandable</td>
<td>FORTRAN, BASIC, PL/M, etc. (A TRANSLATABLE LANGUAGE)</td>
<td></td>
</tr>
<tr>
<td>More Mysterious</td>
<td>ASSEMBLER CODE                    (A TRANSLATABLE LANGUAGE)</td>
<td></td>
</tr>
<tr>
<td>Digital Hieroglyphics</td>
<td>Binary Machine Code               (A NATURAL FOR THE 8080)</td>
<td></td>
</tr>
</tbody>
</table>

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.

![Diagram showing the process of translating FORTRAN to machine code](chart)

5. Finish processing machine code with LINK and LOCATE programs.

![Diagram showing the process of linking and locating programs](chart)

6. Run the finished program.

![Diagram showing the executable program](chart)
FORTRAN

- THE LANGUAGE

FORMULA TRA\textcolor{red}{NSLATION}

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 (20)  
Real SAM_3, BINT
A SIMPLE PROGRAM

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:

- Simple Sequence of Statements
- \( x = A + B \)
- \( y = \sin(x) \)
EXECUTABLE STATEMENTS

IF - THEN - ELSE

IF (A > B) THEN
    BILL = BILL + 1
ELSE
    SAM = 32
ENDIF
DO LOOP

I > 20

A = A + I

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 (6,20)
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
DIMENSION X.(5)

10           READ (5,*) X(1), X(2), X(3), X(4), X(5)
Y = 0

20           DO 20 I = 1, 5
Y = Y + X(I)
WRITE (6,*) Y
           GO TO 10
20           END
SUMMARY

The Assignment Statement:

\[ X = A + B \quad \text{MEANS} \]

ADD A TO B AND PLACE
THE RESULT IN X.

\[ X = X + 1 \quad \text{MEANS} \]

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

More Complicated Assignment Statements:

\[ X(I) = (B \times 75.0)^2 \]

or \((75.0 \times B)^2\)

\[ JHAWK = (\sqrt{B^2 - 4.0A \times C})/2.0A \]
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 149, 149, 151. The next number would be 153, but it is bigger than 152, so it is not done.
EXTENSIONS

To accommodate 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 constant.

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 (0, TEMPDT)
CALL OUTPUT (0, 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

NOTE: F0RT80 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(TSL01.OBJ, TSL01.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 **locate** the program.

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

---

**Diagram:**

- COMPLETE MACHINE CODE (TSLØ1.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:

TSL Ø1
The Development Steps (A Review)

1. Describe Problem Completely
2. Describe Solution Completely
3. Partition Solution into Hardware & Software
4. Implement Hardware Part of Solution
   - Debug Hardware
5. Implement Software Part of Solution
   - Debug Software
6. Integrate Hardware and Software
7. 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

MEMORY -> 8085 CPU -> INPUT/OUTPUT PORTS
IN THE BEGINNING OF HARDWARE DEVELOPMENT, OUR SYSTEM LOOKS LIKE THIS:
With ICE present, we have this at first:

```
---
| ICE 8085 CPU |
---
```
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
$\texttt{include}$

ISIS Level
Submit
The include feature is common to all Intel supplied compilers and assemblers.

Permits the inclusion of blocks of source code within any program.
First Possibility

I need a copyright notice in the beginning of every program I write.

1. Create a file called Copyright (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 electronic, mechanical, etc.
```

3. To use the notice place

```
$INCLUDE (COPYRIGHT)
```

as the first line of your program ($ goes in column 1).

4. When the source code is compiled, the file COPYRIGHT 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(16)
   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

- FORT 8φ :FI: JACK. FOR
- FORT 8φ :FI: JILL. FOR
- FORT 8φ :FI: HILL. FOR
- COPY :FI: JACK. LST TO :LP:
- COPY :FI: JILL. LST TO :LP:
- COPY :FI: HILL. LST TO :LP:
- LINK :FI: JACK. OBJ, :FI: JILL. OBJ, & :FI: HILL. OBJ TO :FI: TOTAL.LNK
- LOCATE :FI: TOTAL. LNK
Submit

Create a file called

3Comp. CSD

Fort 80 :Fi: Jack. For
Fort 80 :Fi: Jill. For
Fort 80 :Fi: Hill. For
Copy :Fi: Jack. LST To :lp:
Copy :Fi: Jill. LST To :lp:
Copy :Fi: Hill. LST To :lp:
Locate :Fi: Total. Lnk

Then after reading 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:

```
FORT 80  % 1.FOR
FORT 80  % 1.FOR
FORT 80  % 2.FOR
COPY  % 1.LST TO :LP:
COPY  % 2.LST TO :LP:
LINK  % 1.OBJ, % 2.OBJ, % 3.OBJ TO &
        % 3.LNK
LOCATE % 3.LNK
```

To use:

```
% 0  % 1  % 2  % 3
Submit 3acomp(:F1:JACK,:F1:JILL,:F1:HILL,:F1:TOTAL)
```
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 80   % 0. FOR
Fort 80   % 1. FOR
Fort 80   % 2. FOR
\$E(CR)
COPY   %0. 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.
UPP HARDWARE

The UPP hardware consists of:

- UPP CHASSIS
- UPP CONTROL CARD (4004 μP)
- PERSONALITY CARDS
- 2708
- 8748
- 8755
- ADAPTOR SOCKETS (FOR 40-PIN EPROMS)
UPP SOFTWARE

The UPP Software consists of:

UPM
USAGE OF UPP

COMMANDS AND DATA FLOW

FILE
(DISKETTE OR PAPERTAPE)

MEMORY OF DEVELOPMENT SYSTEM

CONSOLE

TRANSMIT

PROGRAM

COMPARE

CHANGE

WRITE

READ

DISPLAY
UPP MEMORY USAGE

Actual Development System Address

7600H

Logical Address 0000

Prom Address

0000

Normal Default Offset is 7600H

(This value is equal to the offset)
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 Logical Addresses

Example:

DIS FROM 0 TO 7FH
DIS FROM 100H TO 300H
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 φ IN THE ROM!

EXAMPLE:

TRA FRO φ TO 3FFH
TRA FRO 4ΦΦH TO 7FFH
COMPARE COMMAND

COMPARE FROM start TO finish

Both START AND FINISH ARE LOGICAL ADDRESSES

COMPARE, LIKE TRANSFER, ASSUMES A STARTING ADDRESS IN THE PROM OF Ø

EXAMPLE:

COMPARE FROM Ø TO 3FFH
COMPARE FROM 4FFH 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 $\phi$ TO 3FFH START $\phi$
PRO FRO 8000H TO 8700H START $\phi$

(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 φ = 3EH, φEH, φD3H, 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 0 TO 7FFFH FILE :F1: JACK.OBJ
WRI FRO 8000H TO 0FFFFH FILE :F1: JILL.HEX
Sample Session

**Command**

- UPM
- ISIS- II PROM MAPPER Vx.x
- Type * 8755
- Socket = 2
- Transfer from Ø to 7FFH
- Display from Ø to ØFFH

**Comment**

- Call in upm software
- Software signs on
- Asks for type; we give 8755
- We have personality module in socket 2
- We place 8755 in socket 2 and read to verify it's all erased
- Should see ØFFH from each position

7-12
## Sample Session

(Continued)

<table>
<thead>
<tr>
<th>Command</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF FF FF FF FF FF FF etc.</td>
<td></td>
</tr>
<tr>
<td>FF FF FF FF FF FF FF etc.</td>
<td></td>
</tr>
</tbody>
</table>

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

* DISPLAY FROM  
  Ø TO ØFH  
  CHECK FOR PROPER DATA

| ØØØØ 3E ØE D3 2Ø etc. | |

* PROGRAM FROM Ø TO PROGRAM THE FIRST 7FH START Ø 7FH LOCATION
SAMPLE SESSION (CONTINUED)

COMMAND

* COMPARE FROM Ø TO 7FH

* EXIT

COMMENT

CHECK ONE MORE TIME

DONE; EXIT TO ISIS
A NEW TOPIC

-THE SUBROUTINE

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

```
{  
    SUM ARRAY #1
    __________
    3

    SUM ARRAY #2
    __________
    3

    SUM ARRAY #3
    __________
    3
}
```
Subroutines

Wouldn't it be nice to use a common program to do the sums?
Like this:

CALL SUM (ARRAY 1) → SUM
CALL SUM (ARRAY 2) → SUM
CALL SUM (ARRAY 3) → SUM

RETURN
**Subroutines**

- **An Example**

Program Four

```plaintext
INTEGER A, B, C, ATOT, BTOT, CTOT, Z, I
DIMENSION A(4), B(4), C(4)
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 = φ
DO  I = 1, 4
   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 source program; translate, link, and locate.
4. Try again.
Program Debugging
(Monitor Style)

1. Load Program.

-DEBG :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 PROGRAM'S STARTING ADDRESS, AND . IS THE MONITOR PROMPT CHARACTER.
PROGRAM DEBUGGING  
(MONITOR STYLE)

2. STEP THROUGH YOUR PROGRAM USING  
MONITOR COMMANDS.

\[G[\text{XXX}], \text{YYY}, \text{ZZZ}]\]  
\[\text{GO[AND SET BREAK POINT]}\]

\text{XXX} = START ADDRESS. IF OMITTED,  
CONTINUE WHERE YOU LEFT OFF.  
\text{YYYY} = BREAK POINT (STOPPING POINT) \#1  
\text{ZZZZ} = " " " " \#2

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

EXAMPLES:

\[G3680\]  
\[G3680, 3752\]  
\[G3680, 3752, 3790\]  
\[G, 37A7\]

\text{START AT 3680; DO NOT STOP.}  
\text{START AT 3680; STOP IF YOU}  
\text{HIT 3752.}  
\text{START AT 3680; STOP IF YOU}  
\text{HIT 3752 OR 3790.}  
\text{CONTINUE FROM WHERE WE}  
\text{LAST STOPPED. STOP}  
\text{IF YOU HIT 37A7.}
PROGRAM DEBUGGING
(MONITOR STYLE)

MONITOR COMMANDS (CON'T.)

Dxxx,yyy DISPLAY A RANGE OF MEMORY

xxx = STARTING ADDRESS
yyy = ENDING ADDRESS (MAY BE SAME AS START FOR A SINGLE BYTE)

xxx AND yyy 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)

<table>
<thead>
<tr>
<th><strong>You See</strong></th>
<th><strong>Action</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>#3680</td>
<td>START PROGRAM (MONITOR HAS START ADDRESS ALREADY) AND RUN TO SELECTED LINE IN PROGRAM (LINE 4 OR 3690)</td>
</tr>
<tr>
<td></td>
<td>HAVE MADE IT TO 3690 LOOK AT TEMPDAT (3701)</td>
</tr>
<tr>
<td>G, 3690</td>
<td>IF HAS A ØF WHICH WAS ON THE SWITCHES</td>
</tr>
<tr>
<td>#3690</td>
<td>NOW RUN TO LINE 6 (3698)</td>
</tr>
<tr>
<td>.D 3701, 3701</td>
<td>NOTE LITES NOW HAVE ØF</td>
</tr>
<tr>
<td>3701 ØF</td>
<td>PROGRAM IS OK!</td>
</tr>
<tr>
<td>.G, 3698</td>
<td>LET IT RUN</td>
</tr>
<tr>
<td>#3698</td>
<td></td>
</tr>
<tr>
<td>.G</td>
<td></td>
</tr>
</tbody>
</table>
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

\[ \text{MAP [MEMORY] partition = GUARDED}\]

\[ \text{USER [NOVERIFY]} \]

\[ \text{INTELLEC EXP [NO VERIFY]} \]

\[ \text{WHERE partition is one or more contiguous blocks (2048 bytes) of memory and EXP is a starting address in INTELLEC memory (multiple of 2048)} \]

Examples:

\[ \text{MAP } \emptyset \text{ TO } 2φ47 \text{ = GUARDED} \]

\[ \text{MAP MEM MEMORY } \emptyset \text{ TO } 8K \text{ = USER} \]

\[ \text{MAP MEM } 8φφφH \text{ TO } AφφφH \text{ = USER [NOVERIFY]} \]

\[ \text{MAP } 4φφφH \text{ LEN } 4K \text{ = USER} \]

\[ \text{MAP } 4φφφH \text{ LEN } 4K \text{ = INTELLEC } 6φφφH \]

To check MAP status:

\[ \text{MAP} \]
ICE 85 Commands

3. (CONTINUED)

INPUT/OUTPUT

MAP IO partition = | GUARDED USER INTELLEC |

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

EXAMPLES:

MAP IO 0 TO 7 = USER
MAP IO 0 TO 7 = INTELLEC
MAP IO 0 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]
[UNTIL break cond] [OR break cond2]
[OR STOP]
[UNTIL STOP]

WHERE addr is the starting address
AND break condition is

StopAddr READ
WRITE EXECUTED
INPUT OUTPUT

Examples:
GO FROM .START FOREVER
GO FROM .START UNTIL .TOTAL2 WRITTEN
GO UNTIL .TOTAL3 WRITTEN OR .MAX READ
GO UNTIL ..ONE 35 EXECUTED OR ..MIN WRITTEN
GO UNTIL 35 XXH EXECUTED
ICE 85 COMMANDS

DISPLAY/MODIFY MEMORY

BYTE address

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

FOR INSTANCE:

BYTE .MAX
BYTE .ARRAY TO .JACK
BYTE 3000H LEN 50H
WORD .MIN
WORD .ARRAY TO .JACK
WORD 95100H LEN 300H

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, 10T, 31, 29, 2A
WORD .SAM = 4A77
WORD .WARRAY = 4622, AA77, 2510T, 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
\[\phi \text{AH} \]

ICE RESPONSE

REG
\[p=\phi\phi18H \quad s=\phi7\text{FEH} \quad a=\phi\phi H \quad f=\phi\phi H \quad b=\phi\phi H \quad \text{etc.}\]

TO MODIFY A REGISTER:

Rx = \text{val}
WHERE \text{val} IS THE VALUE TO BE
PLACED IN THE REGISTER

RA = 23
RBC = 1234H
RDC = 3\phi1\phi
ICE 85 COMMANDS

DISPLAY INPUT PORT CONTENTS

Port 

Port portnum

WHERE portnum IS THE INPUT PORT DESIRED

Port 35
Port 1φT

MODIFY OUTPUT PORT CONTENTS

Port 

Port portnum = val

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

Port 2Ah = 19T
Port 1φ = 1φ
Port 1BH = 1φ1φ1φ1φ
ICE 85 Commands

Display Trace Memory

Trace Display Mode

\[ \text{Trace} = \begin{array}{c}
   \text{Instructions} \\
   \text{Cycles}
\end{array} \]

Trace Display

Print \[ \begin{array}{c}
   \text{All} \\
   \pm n
\end{array} \]

Where \( n \) = Number of Entries to Display

Print All
Print \(-1\phi\)

Oldest, Newest

Oldest \quad \text{Move to first entry in trace buffer}

Newest \quad \text{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 decimal 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 10
STEP FROM 3200 COUNT 27
STEP FROM ..ONE #35 TILL BYTE .MAX>55
STEP TILL PC=..TWO#35 OR WORD .LIMIT=1700
ICE 85 COMMANDS

THE DUMP

While stepping, we want to see what is happening in the registers of the 8085.

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>DUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISABLE</td>
<td></td>
</tr>
</tbody>
</table>

For Enable Dump, we can select the areas of our program where dumping will occur with the following:

<table>
<thead>
<tr>
<th>Enable Dump</th>
<th>partition</th>
<th>call</th>
<th>jump</th>
<th>return</th>
</tr>
</thead>
</table>

Where partition is any address range, i.e.,

$1000h$ to $2000h$

or $3860h$ len $300h$

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 \carrige

   WHERE  mmm = current month (i.e. SEP, MAY etc.)
   dd = current day (i.e. 27, 03 etc.)
   yy = current year (i.e. 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.
5. Type:

```
DIR 12
```

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 12
```

Aha! There is the directory, map, label and to 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 12
```

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)

Let's see what we got.

DIR 1 2

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 2
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 2

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

9. Try:

RENAME :FI:FILEA TO :FI:FILE3 2
Laboratory 2 (continued)

Sooner or later we will want to delete a file.

10. Try:

   DELETE :F1:FILE3 2

   To see the effect

   DIR 1 2

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

11. Try:

    DELETE :F1:FILE2.AAA 2

    Didn't work did it?

To unwrite protect a file use the attribute changing program.

12. Type:

    ATTRIB :F1:FILE2.AAA W0

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

    DIR 1 2

    We can now delete it

    DELETE :F1:FILE2.AAA 2
LAbORATORY 2 (CONTINUED)

WE SHOULD TRY A CONTROLLED WILDCARD DELETE.

13. Type:

    DELETE :F1:*** Q 2

The Q allows us to decide on a file by file basis whether a file is to remain or not. Keep some and delete 2. (Remember the ISIS files are protected!) See the results with:

    DIR 1 2

As a final clean up:

    DELETE :F1:*** 

Will delete all non-write protected files.
LABORATORY 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 2

   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.
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**:
  - Up
  - Down
  - Left
  - Right

- **POSITION CURSOR**

- **DELETE CHARACTER AT CURSOR POSITION**

- **INSERT CHARACTER AT CURRENT CURSOR POSITION**
  To use type:
  - ↑C then
  - CHARACTER DESIRED
  - ↑CA would insert an A
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 environmental 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:

HOME

The top of the screen should now look like:

*  

---  

Perfection in technical rationality requires etc.
LABORATORY 3 (CONTINUED)

6. TO EXIT TYPE:

    EX 2

    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 1
    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 organizations.
      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)


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

```
[HOME]
```

to get to command mode.

Now type:

```
JTE 2
```

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 2 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!
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 shellfish. If Congress eventually includes aquaculture in such appropriations, the emphasis probably will go to research on unromantic species like tilapia 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.

```
HOME JTT 2
```
b) Replace all occurrences of "to" with "xxx"

\[
!<sQ/to/xxx/>2
\]

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 \texttt{ESC} (escape).

\texttt{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 \texttt{HOME} to get to command mode. To set the tag type:

\texttt{TS42}

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

\texttt{TS52}
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:

\[
\text{↑N (next page)}
\]

To go back to a previous page:

\[
\text{↑P (previous page)}
\]

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 $ to drive 1.
3. Translate the program with:
   
   FORT80 :FI:CHECK.FOR 2

4. Get a listing of the translated program:
   
   COPY :FI:CHECK.LST TO :LP: 2
   
   (Make sure the line printer is attached to your system when you do this!)

5. Get a copy of the link command file:
   
   COPY :FI:FORTL TO :FI: 2
LAbORATORY 4 (CONTINUED)

6. Run the command file to link your program:
   
   **SUBMIT :FI:FORTL (:FI:CHECK) 2**

7. Locate the final assembly:
   
   **LOCATE :FI:CHECK.LNK 2**

8. Run the program **AFTER** reading the listing
   (to see what it does!)
   
   **:FI:CHECK 2**

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 :FI:CHECK.FOR etc!
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 XXX.XX)  379.52

   ENTER C(check), D(deposit) or Q(quit)  C
   ENTER CHECK NUMBER (4 DIGITS MAX)  1799
   ENTER CHECK AMOUNT (UP TO XXX.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:

<table>
<thead>
<tr>
<th>Check No.</th>
<th>Amount</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>3275.00</td>
<td>3275.00</td>
</tr>
<tr>
<td>103</td>
<td>50.00</td>
<td>3225.00</td>
</tr>
<tr>
<td>112</td>
<td>70.00</td>
<td>3155.00</td>
</tr>
</tbody>
</table>

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 :FI: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.

<table>
<thead>
<tr>
<th>Payment No.</th>
<th>Period Interest</th>
<th>Payment to Principle</th>
<th>Remaining Principle</th>
<th>Yearly Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>175.00</td>
<td>25.00</td>
<td>29975.00</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>174.85</td>
<td>25.15</td>
<td>29949.85</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>174.71</td>
<td>25.29</td>
<td>29924.56</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>173.35</td>
<td>26.25</td>
<td>29690.19</td>
<td>2090.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Laboratory 4 (continued)

The equations to calculate these values are:

\[ \text{Period Interest}_{k+1} = i \times \text{Principle}_k \]

where

\[ i = \text{Periodic Interest} = \frac{\text{Yearly Interest}}{12} \]

\[ \text{Payment to Principle}_{k+1} = \text{Payment} - \text{Period Interest}_{k+1} \]

\[ \text{(New) Principle}_{k+1} = \text{Principle}_k - \text{Payment to Principle}_{k+1} \]

The flow chart for this program is on the next page.
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, nickel or penny) provided. Always dispense the biggest denomination possible. For example, 37 cents in change would result in

1 - quarter
1 - dime
2 - pennies
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 :FI:
   COPY ICELL.CSD TO :FI:

2. Compile (and directly print the listing!)

   FORTB0 :FI:ICETST.FOR DEBUG PRINT(:LP:)

3. Link with FORTRAN LIBRARIES. Notice the libraries that are being used.

   SUBMIT :FI:ICELL(:FI:ICETST)

4. Our "TARGET" system will have memory in the following pattern:

   ROM φ TO 7FF₁₆
   RAM 2ΦΦΦ TO 2ΦFF
   I/O Ports 2φ TO 26
LABORATORY 5 (CONTINUED)

4. (CONTINUED)

LOCATE :F1: ICESTST.LNK &
CODE ( ) DATA (2000H) STACK (2B00H) &
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 0 TO 7FF = INT 7600

MEANS BORROW 20480 BYTES OF INTELLEC MEMORY
(7600 TO 77FF) AND CALL IT 0 TO 7FF USER

MAP 2000 TO 27FF = USER

ADDRESSES 2000 TO 27FF ACTUALLY EXIST IN
THE USER SYSTEM.

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

MAP 10 20 TO 2F = USER
LABORATORY 5 (CONTINUED)

8. Now we have the needed resources, load the program.

    LOAD :F1:ICETST

9. Let's see if ice8s 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 [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. \(30h \rightarrow 6F_{16} \)
\( \phi h \rightarrow 55_{16} \). SET .VAL TO THAT VALUE:

```
BYTE .VAL = ___ ← SELETED VALUE
```

17. NOW TRY PART OF THE PROGRAM.

```
GO FROM #5 TILL #12 EXEC
HWW DID THE LITES DO?
```

18. WE CAN SINGLE STEP THE PROGRAM:

```
STEP FROM #2 TILL BYTE .VAL > 30h
NOTE HOW SLOWLY THE SYSTEM RESPONDS!
```

19. LET'S WATCH THE TRACE FEATURE:

```
GO FROM #2 TILL #12 EXEC
PRINT -2Φ
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 ICEBS

    EXIT

22. FOR FUN FOLLOW:

    ICEBS
    MAP $ LEN 2K = INT 7660
    MAP IO 2$ TO Z2$ = 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.

<table>
<thead>
<tr>
<th>COLUMN</th>
<th>COLUMN</th>
<th>COLUMN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>16</td>
</tr>
</tbody>
</table>

SUBROUTINE SORT (M, COUNT)

C SORT AN ARRAY OF REAL DATA IN ASCENDING ORDER
C COPYRIGHT NOTICE FOLLOWS
$INCLUDE(COPYRIGHT)

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
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 :FI:TEST.FOR.
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 STEPs. 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. 100) THEN "

   to

   IF (TSTVAL .GT. 10) 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

\textbf{SUBMIT :FI:ICELL(:FI:UPMTST)}

4. THE TARGET SYSTEM WILL HAVE

\begin{itemize}
  \item [\textbf{ROM}] 800H to 9FFFH
  \item [\textbf{RAM}] 2000H to 2FFFH
  \item [\textbf{I/O}] 20H to 23H
\end{itemize}

THE LOCATE STEP WILL THEREFORE BE:

\textbf{LOCATE :FI:UPMTST, LINK &}
\textbf{CODE(800H) DATA(2000H) STACK(20B0H) &}
\textbf{MAP SYMBOLS LINES PRINT(:LP:)}

5. TEST THE RESULT WITH ICE8S

\textbf{ICE8S}

\begin{itemize}
  \item [\textbf{MAP}] 800 = INT 7000
  \item [\textbf{MAP}] 2000 = USER
  \item [\textbf{MAP}] 10 20 = USER
\end{itemize}

WE HAVE RAM AND I/O ON THE BOARD, BUT OUR ROM SOCKET IS EMPTY.
LAboratory 7 (continued)

6. Load the program

   load :fi: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 $0$ to $1FFH$

   Display from $0$ to $1FFH$

The transfer reads the PROM. The display displays the data. It should be all $FFH$. 
LABORATORY 7 (CONTINUED)

18. Now load the object code into memory

Read object file :FI:UPMTST into $f$

19. Display the first 10 locations.

Display from $800H$ to $809H$

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 $800H$ to $0FFFH$ start $f$

21. Now wait, it takes about 2 minutes. The program light will be on during this period.

22. Check the results

Compare from $000H$ to $87FH$

23. Remove the PROM from the socket.
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 φ to FFFF = USER

MAP IO φ to FF = USER

28. Load the symbol table only so we
can still use symbolic debugging.

LOAD :FI: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 ICEBS CAPABILITIES (GO, STEP, DISPLAY, ETC.)

    RETURN THE 8755A TO THE INSTRUCTOR!