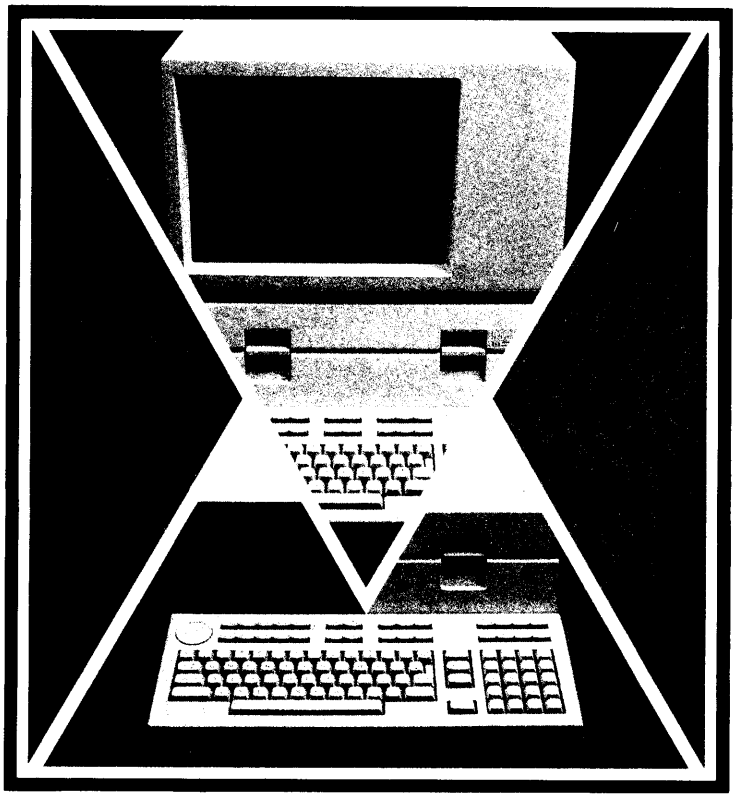
# 9826/36
# Basic Language

*Operating & Programming Course*



HEWLETT
PACKARD

# AGENDA

Series 200 Basic Operating and Programming Course

Day 1

- Documentation
- Basic Operations
- Data Representations and Operations

D1-1

NOTES

Day 2

- Program Structure and Control

Day 3

- Mass Storage Techniques

Day 4

- I/O Programming

Day 5

- Graphics Programming

D1-2

NOTES

NOTES

# DOCUMENTATION

OBJECTIVES:

Identify the correct manual to use

Locate critical information efficiently

D1-3

NOTES

# MANUALS TELL YOU HOW TO

Install the computer
Install Memory/Interface
Configure your system
Test system components
Write correct program lines
Control devices with a program
Correct program errors

D1-4

## SERIES 200 MANUAL STRUCTURE

BASIC Operating Manual

BASIC Programming Techniques

BASIC Language Reference

BASIC Interfaceing Techniques

Extensions

D1-5

NOTES

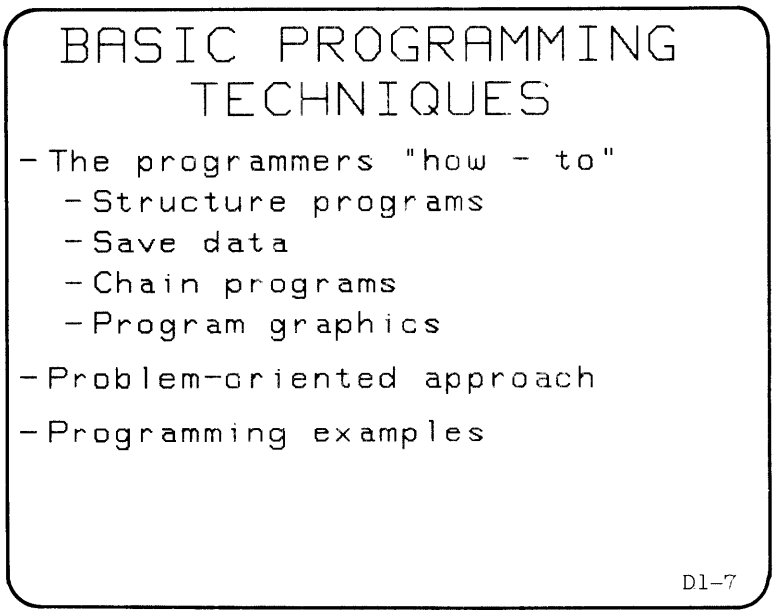
## BASIC OPERATING MANUAL

- First manual
- Shows how to install and set up the computer
- Used by operator thereafter
  - not the programmer -

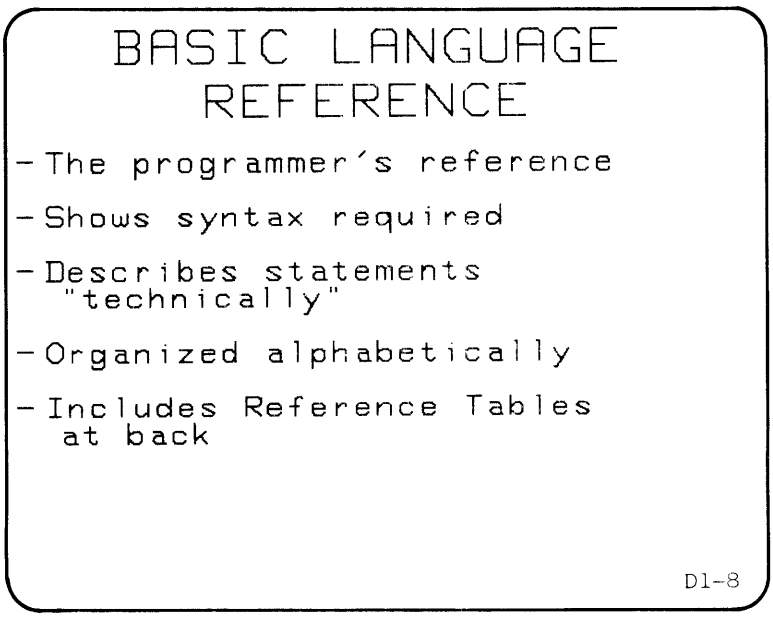
D1-6

NOTES

NOTES


BASIC PROGRAMMING TECHNIQUES
- The programmers "how - to"
- Structure programs
- Save data
- Chain programs
- Program graphics
- Problem-oriented approach
- Programming examples
D1-7

NOTES


BASIC LANGUAGE REFERENCE
- The programmer's reference
- Shows syntax required
- Describes statements "technically"
- Organized alphabetically
- Includes Reference Tables at back
D1-8

# BASIC INTERFACING TECHNIQUES

- The system designer's "how-to"
- Use to learn to collect data and control devices
- Describes how to
  - Program interface cards
  - Handle events
  - Control instruments
- Organized by Task & Interface

D1-9

NOTES

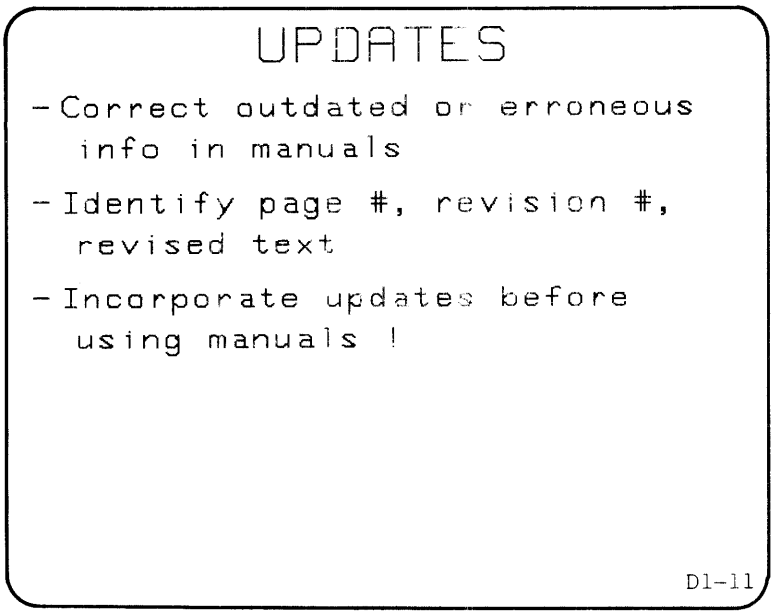
# EXTENSIONS

- Specific programmer "how-to" for language additions, user programs or both
- Organized by task

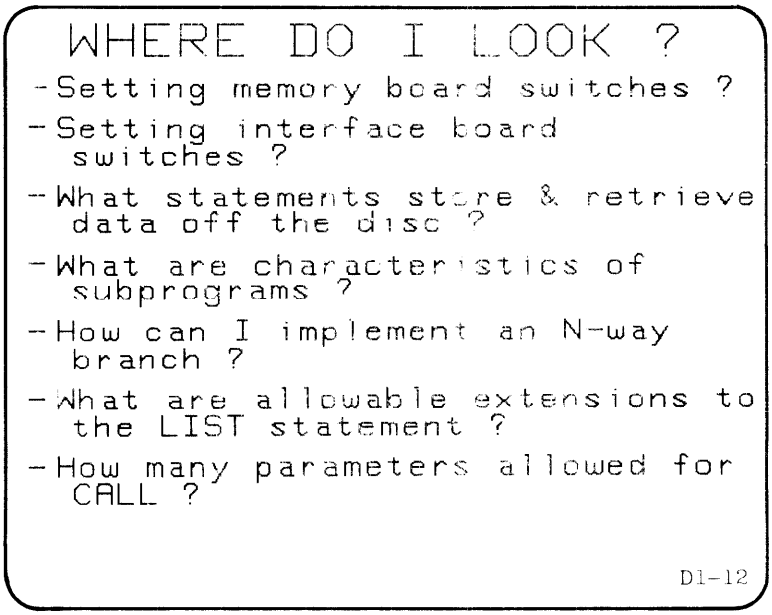
D1-10

NOTES

NOTES


UPDATES
- Correct outdated or erroneous info in manuals
- Identify page #, revision #, revised text
- Incorporate updates before using manuals !
D1-11

NOTES


WHERE DO I LOOK ?
- Setting memory board switches ?
- Setting interface board switches ?
- What statements store & retrieve data off the disc ?
- What are characteristics of subprograms ?
- How can I implement an N-way branch ?
- What are allowable extensions to the LIST statement ?
- How many parameters allowed for CALL ?
D1-12

# BASIC OPERATIONS

Objectives

1. Set up and use an Autostart file
2. Execute Live Keyboard Operations
3. Edit Programs

D1-13

NOTES

# AUTOSTART

Autostart: Automatically start running a user program at power-up time

Four Elements:

1. Power-up
2. "Boot" the Operating System
3. LOAD Program (AUTOST)
4. RUN

D1-14

NOTES

NOTES

## BOOT THE OPERATING SYSTEM

Prioritized:

1. DISC
2. ROM (built-in)
   - A. If more than one O.S. the operator can select
   - B. If no selection is made, priority by board address

D1-15

NOTES

## LIVE KEYBOARD OPERATIONS

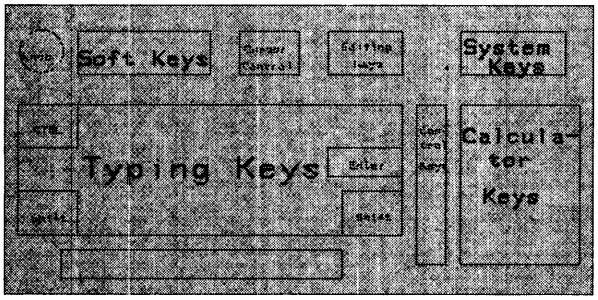
- While running a program the Series 200 still recognizes keypresses
- Program is normally unaffected and unaware of live Kbd.

5*LOG (1.85)/SIN (.77) EXECUTE

CAT EXECUTE

D1-16

# KEYBOARD FUNCTIONAL AREAS




D1-17

NOTES

# SYSTEM KEYS

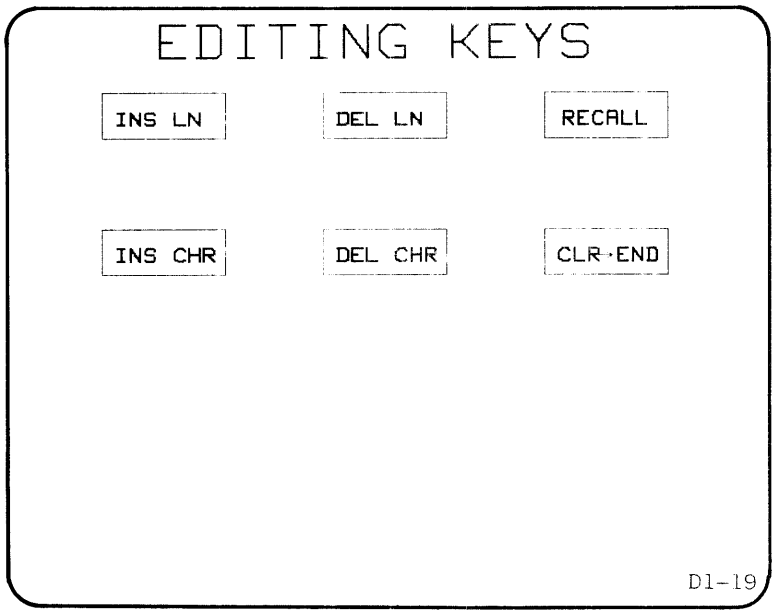
| DISPLAY FCTNS | DUMP ALPHA | DUMP GRAPHICS | ANY CHAR |
|---|---|---|---|
| EDIT | ALPHA | GRAPHICS | STEP |

| CLR SCR | SET TAB | CLR TAB | STOP |
|---|---|---|---|
| CLR LN | RESULT | PRT ALL | CLR I/O |

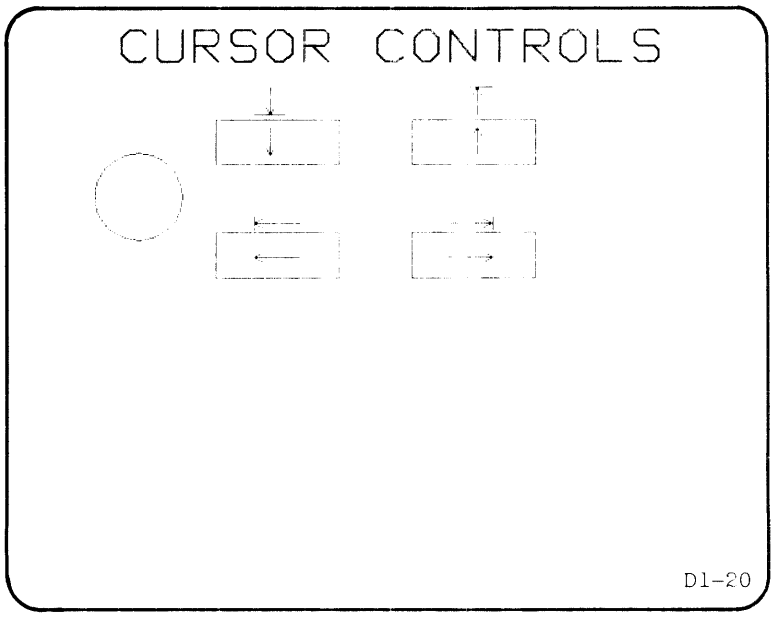
D1-18

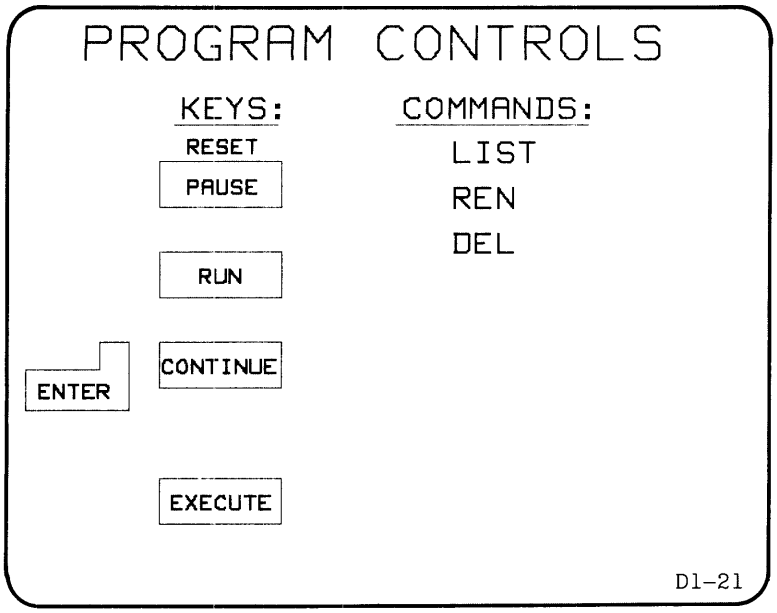
NOTES

**NOTES**

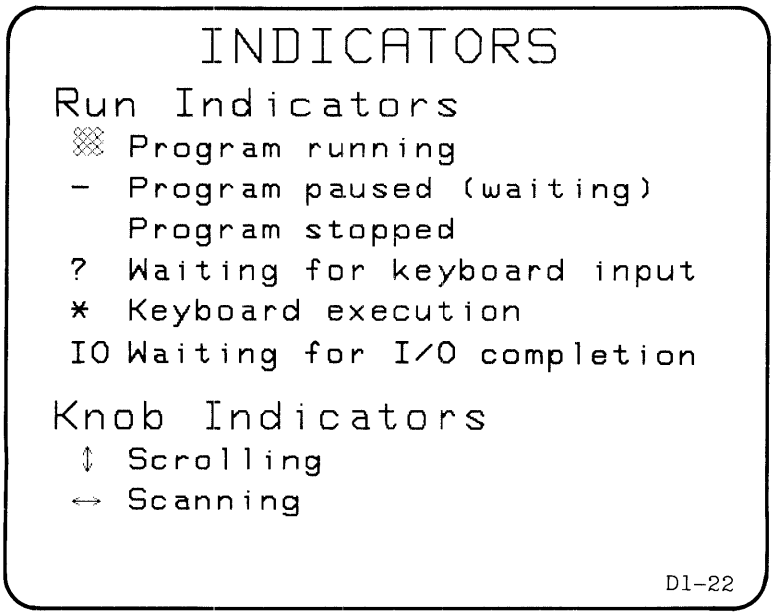

EDITING KEYS
INS LN
DEL LN
RECALL
INS CHR
DEL CHR
CLR·END
D1-19

**NOTES**


CURSOR CONTROLS
D1-20

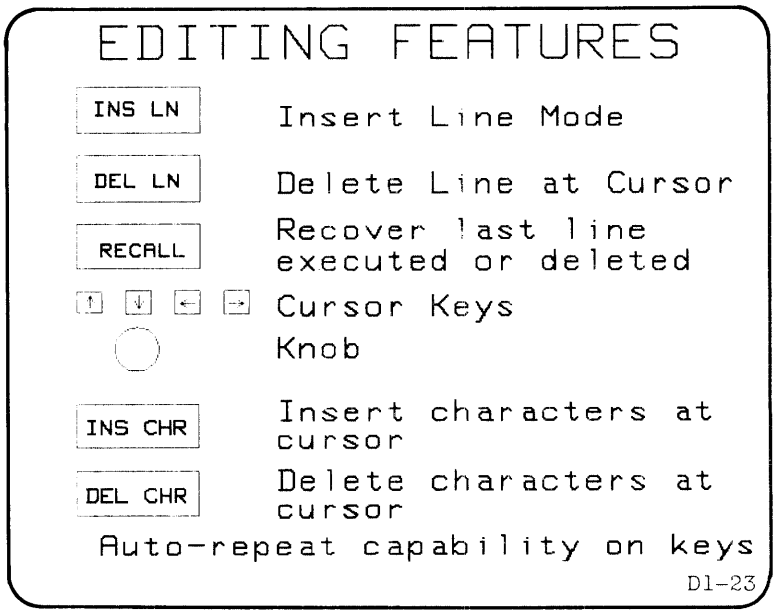

PROGRAM CONTROLS
KEYS:
COMMANDS:
RESET
PAUSE
RUN
CONTINUE
ENTER
EXECUTE
LIST
REN
DEL
D1-21

NOTES

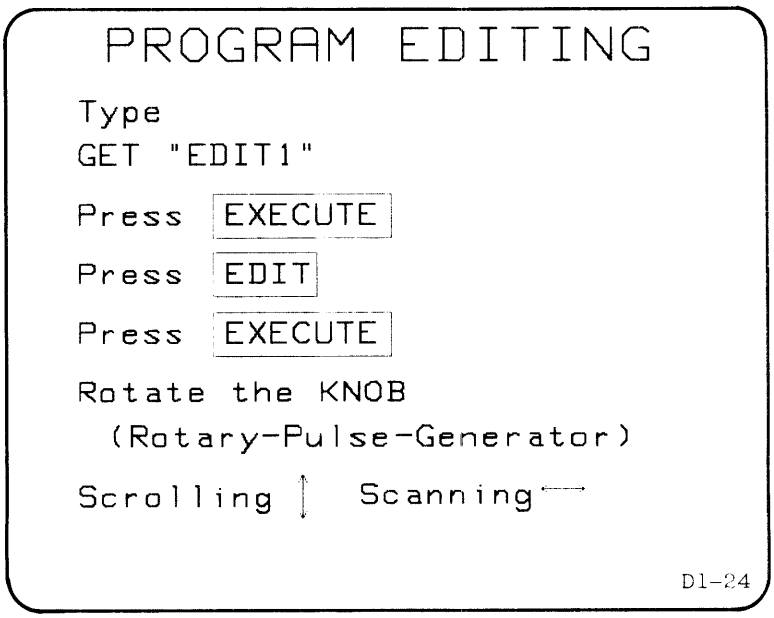

INDICATORS
Run Indicators
Program running
- Program paused (waiting)
Program stopped
? Waiting for keyboard input
* Keyboard execution
IO Waiting for I/O completion
Knob Indicators
↕ Scrolling
↔ Scanning
D1-22

NOTES

NOTES


EDITING FEATURES
INS LN
Insert Line Mode
DEL LN
Delete Line at Cursor
RECALL
Recover last line executed or deleted
Cursor Keys
Knob
INS CHR
Insert characters at cursor
DEL CHR
Delete characters at cursor
Auto-repeat capability on keys
D1-23

NOTES


PROGRAM EDITING
Type
GET "EDIT1"
Press EXECUTE
Press EDIT
Press EXECUTE
Rotate the KNOB
(Rotary-Pulse-Generator)
Scrolling
Scanning
D1-24

## PROGRAM EDITING

Line 100 should be
100 PRINT "Any text will do"

Line 110 should be
110 PRINT "Your NAME here"

Line 120 should be
120 PRINT"Your ADDRESS here"

Remove the exclamation marks,
then press ENTER

D1-25

NOTES

## PROGRAM EDITING

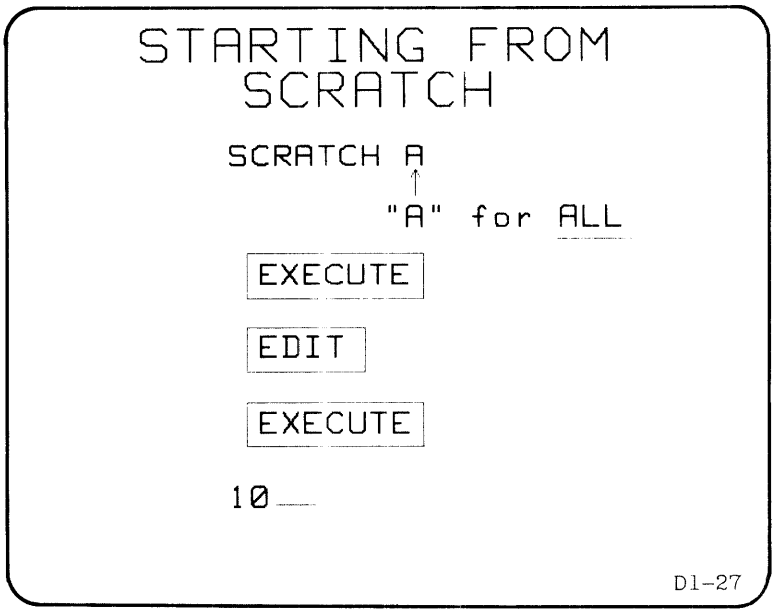
- Delete line 190,200 to eliminate the first run-time error

- Change the GOTO line number of line 150 to eliminate the second

D1-26

NOTES

**NOTES**




```
10     REM     Computation of Depreciation
20     REM     Three computation methods
30     !
40     PRINT
50     PRINT
60     PRINT "Enter a 1 for Straight-Line method"
70     PRINT "Enter a 2 for Decling Balance method"
80     PRINT "Enter a 3 for Sum-of-Years method"
90     INPUT "Method?",I
100    INPUT "Enter the value",V
110    INPUT "Enter the number of years",N
120    PRINT
130    !
140    ON I GOTO 160,190,220
150    !
160    PRINT "Straight-line method"
170      GOTO 240
180      !
190    PRINT "Declining balance method"
200      GOTO 240
210      !
220    PRINT "Sum-of-years method"
230    !
240    LET J=0
250    LET D1=V/N
260    LET F1=V/(N*(N+1)/2)
270    PRINT "Year","  Depr","  Value"
280    LET J=J+1
290    ON I GOTO 320,400,510
300    !
310    !
```

```
320   REM Straight-line method
330   !
340   LET V=V-D1
350   PRINT J,D1,V
360   IF J<N THEN 280
370   STOP
380   !
390   !
400   REM Double declining balance method
410   !
420   LET D2=(2/N)*V
430   LET D2=DROUND(D2,6)
440   LET V=V-D2
450   LET V=DROUND(V,6)
460   PRINT J,D2,V
470   IF J<N THEN 280
480   STOP
490   !
500   !
510   REM Sum-of-year's digits method
520   !
530   LET F2=N-J+1
540   LET D3=F1*F2
550   LET D3=DROUND(D3,6)
560   LET V=V-D3
570   LET V=DROUND(V,6)
580   PRINT J,D3,V
590   IF J<N THEN 280
600   END
```

## SPACE DEPENDENT

- Keywords recognized regardless of case (upper or lower case)
- Line Labels & Variable Names are converted to initial caps and lower case, automatically
- Beware of misspelled keywords !

D1-28

**NOTES**

NOTES

## NAMES

Character restrictions:

1. First character: uppercase letters, or characters with ASCII codes 161 thru 254
2. Remaining characters: lowercase letters, numerals underscore, or characters with ASCII codes 161 thru 254

15 Character names: Variables
Subprograms
Com labels
Line labels

13 Character names: Functions

10 Character names: Files

D1-29

NOTES

## BASIC TERMINOLOGY

Keywords
```
GOSUB
```

Statements
```
LIST 150,1000
```

Program lines
```
110 Loop:GOTO Loop
```

Functions
```
SIN(X)
```

Expressions
```
50*SIN(X)/360*(SQR(Y)+Z)
```

Commands
```
REN 100,5
```
EXECUTE

D1-30

# A BASIC REVIEW

GET "REVIEW" EXECUTE

LIST #701 EXECUTE

Discuss the program

Run the program

D1-31

NOTES

# BASIC DATA REPRESENTATIONS AND OPERATIONS

Objectives

Select the optimum data type for applications

Select the appropriate built-in operation or function that produces the desired results

D1-32

NOTES

NOTES

## THE FOUR DATA TYPES

| | |
|---|---|
| Real | (Floating Point Numbers) |
| Integer | (Whole Numbers) |
| String | (Characters) |
| @Name | (Data Paths) |

D1-33

NOTES

## NUMERIC REPRESENTATIONS

(Real and Integer)

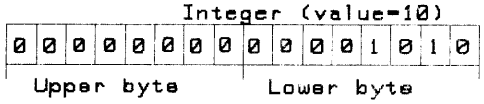
- Constants
  3.1415926
- Simple Variables
  X, Index_one, Value
- Array Variables
  A(I,J), Volts(Time,Range)

D1-34

# SIMPLE INTEGERS

10 INTEGER Alpha, Omega
(Explicit declaration)

- Whole numbers only
- 16 bit representation
- Two's complement number
- Range; -32768 to +32767
- Storage; 2 bytes
- Speed: Fastest math




D1-35

NOTES

# SIMPLE REALS

20 REAL Maximum, Minimum
(Explicit declaration)

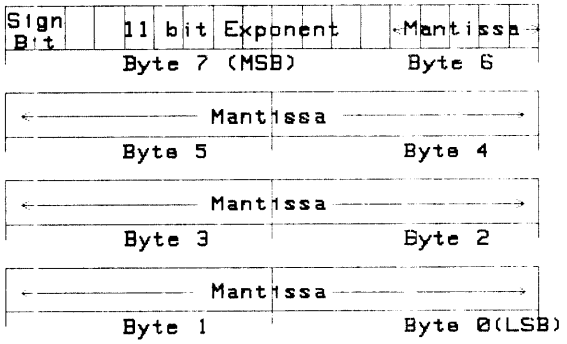
- Default numeric representation
- Whole number and fractional part
- 64 bit representation
  - 11 bit exponent (E±308)
  - 53 bit mantissa (15 plus digits)
- Range: Approx ±2E308, ±2E-308
- Precision: 15 decimal digits
- Speed: 2-5 times slower than integer

D1-36

NOTES

NOTES

# REAL NUMBER REPRESENTATION




D1-37

NOTES

# NUMERIC ARRAYS

```
25 OPTION BASE 1
30 DIM Big_array(100,1000)
35 INTEGER Twobit(-50:50,100)
```

- Default is OPTION BASE 0
- Maximum 6 dimensions
- Maximum 32767 elements/ dimension
- Implicit lower bound (OPTION BASE)
- Explicit lower bound (-50:50)
- Default 10 element, n-dimension REAL array unless explicity declared

D1-38

# ARRAY EXAMPLES

```
10 DIM A(2,3)
20 A(1,2)=3.14
30 END
```

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | 3.14 | |
| 2 | | | | |

```
10 OPTION BASE 1
20 DIM A(1981:
     1984,4)
30 A(1983,4)=31
40 END
```

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1981 | | | | |
| 1982 | | | | |
| 1983 | | | | 31 |
| 1984 | | | | |

D1-39

NOTES

# INTEGER OPERATIONS

Arithmetic Operators

Integer arguments, integer result

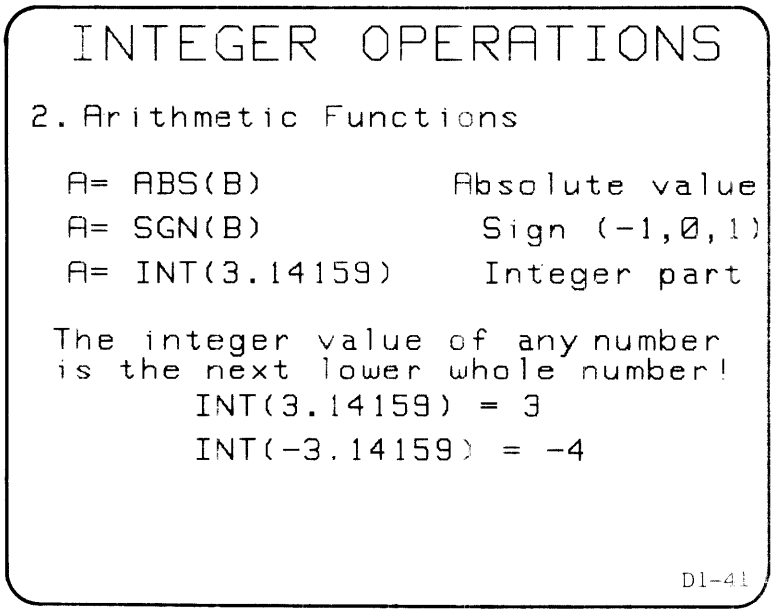
```
A= B+C       Addition
A= B-C       Subtraction
A= B*C       Multiplication
A= B DIV C   Division Quotient
A= B MOD C   Division Remainder
```

```
9 ÷ 2 = 4 + Remainder 1
9 DIV 2 = 4
9 MOD 2 = 1
```

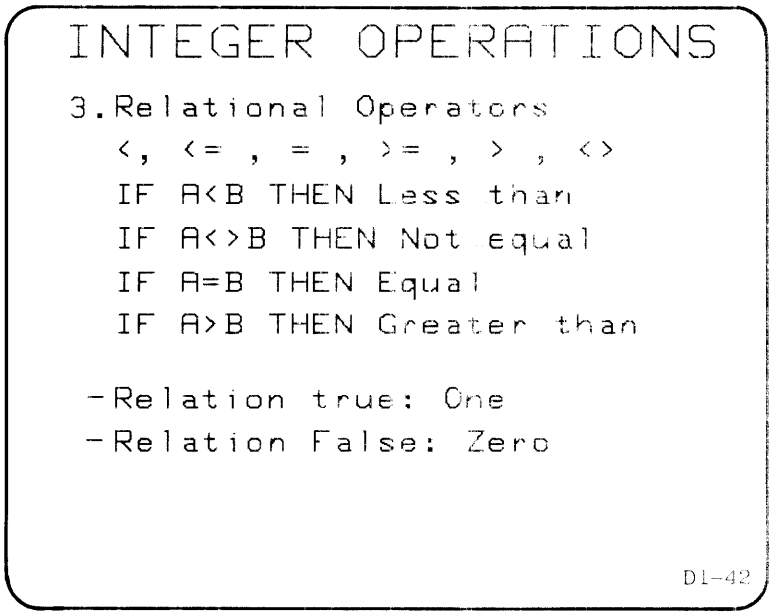
D1-40

NOTES

NOTES


INTEGER OPERATIONS
2. Arithmetic Functions
A= ABS(B) Absolute value
A= SGN(B) Sign (-1,0,1)
A= INT(3.14159) Integer part
The integer value of any number is the next lower whole number!
INT(3.14159) = 3
INT(-3.14159) = -4
D1-41

NOTES


INTEGER OPERATIONS
3. Relational Operators
<, <= , = , >= , > , <>
IF A<B THEN Less than
IF A<>B THEN Not equal
IF A=B THEN Equal
IF A>B THEN Greater than
- Relation true: One
- Relation False: Zero
D1-42

# INTEGER OPERATIONS

4. Logical Operators
   - AND, OR, NOT, EXOR

```
IF NOT(A>=B) THEN Less_than
IF (A<B) EXOR (C<D) THEN
                  Only_one_less
IF (A<B) AND (C<D) THEN
                       Both_less
```

AND

| A \ B | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

OR

| A \ B | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

EXOR

| A \ B | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

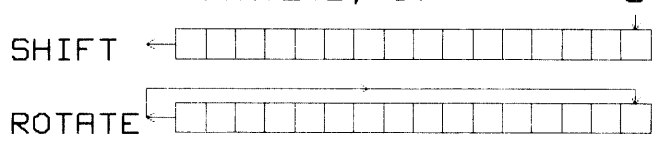
D1-43

NOTES

# INTEGER OPERATIONS

5. Binary Operations
   - BINAND, BINIOR, BINCMP, BINEOR, SHIFT, ROTATE, BIT

```
A = BINAND(B,C)
A = SHIFT(B,8)
IF BIT(A,2) THEN Bit_set
A = ROTATE(B,-8)
```




D1-44

NOTES

NOTES

# REAL NUMBER OPERATIONS

1. Arithmetic Operators
   Real arguments, real result
   +, -, *, /, ↑, MOD, DIV
   A = B↑C

2. Arithmetic Functions
   LOG, LGT, EXP, SQR, ABS, SGN,
   DROUND, RND, (RANDOMIZE)
   A = EXP(B)

D1-45

NOTES

3. Trigonometric Functions
   SIN, COS, TAN, ASN, ACS, ATN, PI,
   DEG, RAD

- Use DEG/RAD to set angular mode (degrees or radians) for trig functions.
  Default = RAD

```
10 DEG
20 IF ASN(1) = 90 THEN PRINT"YES"
30 END
```

D1-46

## 4.Relational Operators

- AND, OR, NOT, EXOR

```
IF NOT(A>=B) THEN Less_than
IF (A<B) EXOR (C<D) THEN
                  Only_one_less
IF (A<B) AND (C<D) THEN
                       Both_less
```

- Beware of tests for equality of two REALs !

AND

| A \ B | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

OR

| A \ B | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

EXOR

| A \ B | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

D1-47

NOTES

## TYPE-CONVERSION

Using Reals In Integer, Logical, & Binary Operations

- If the argument required is type INTEGER and a REAL is given, the number is automatically converted to an integer
- If the argument required is type REAL and an INTEGER is given, the number is converted to a REAL

From the keyboard, try:

```
400*400    EXECUTE
400.*400   EXECUTE
```

D1-48

NOTES

NOTES

## TYPE CONVERSION

How do type-conversions affect a program?

- They require time
- They may not be obvious
- If possible, perform necessary type-conversions explicitly, outside of program loops

D1-49

NOTES

## GET "LOOPTIME"

Modify the timed program line to see the effects of using Real vs Integer numbers for:

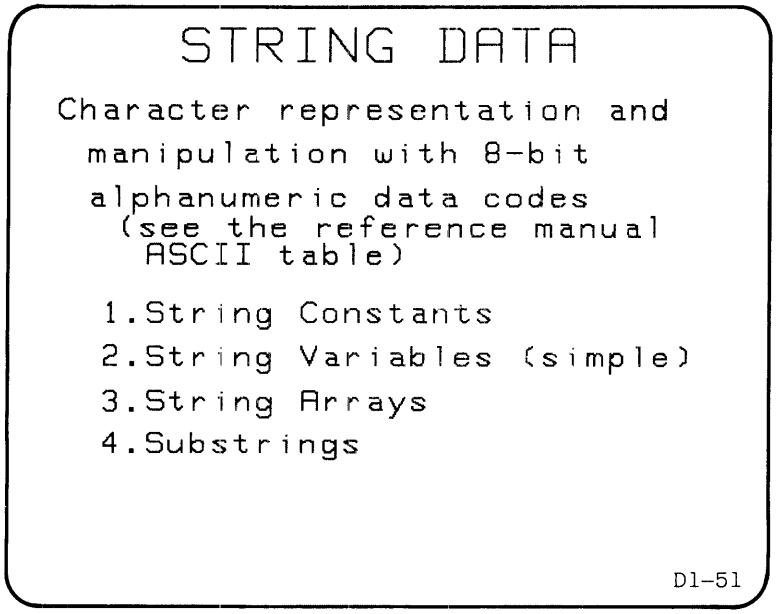
- Binary functions
- Array indexing
- Trig functions
- Integer operations (MOD)

D1-50

1. Modify Line 130 to time various statements. Some examples are:

```
130 A(One) = PI              !Integer array index.
130 A(Four) = PI             !Real array index
130 IF  One THEN 140         !Integer relational
130 IF Four THEN 140         !Real relational
130 Four = Five + Six        !Real add, no convert
130 One = Five + Six         !Real add,convert
130 One = BINAND (Two,Three) !Intgr op,Intgrs.
130 Four = BINAND (Five,Six) !Intgr op,Reals
```

2. Now GET "FINDMAX" and follow directions listed in the program.




**NOTES**

# US ASCII Character Codes

| ASCII Char. | EQUIVALENT FORMS | | | |
|---|---|---|---|---|
| | Binary | Oct | Hex | Dec |
| NULL | 00000000 | 000 | 00 | 0 |
| SOH | 00000001 | 001 | 01 | 1 |
| STX | 00000010 | 002 | 02 | 2 |
| ETX | 00000011 | 003 | 03 | 3 |
| EOT | 00000100 | 004 | 04 | 4 |
| ENQ | 00000101 | 005 | 05 | 5 |
| ACK | 00000110 | 006 | 06 | 6 |
| BELL | 00000111 | 007 | 07 | 7 |
| BS | 00001000 | 010 | 08 | 8 |
| HT | 00001001 | 011 | 09 | 9 |
| LF | 00001010 | 012 | 0A | 10 |
| VT | 00001011 | 013 | 0B | 11 |
| FF | 00001100 | 014 | 0C | 12 |
| CR | 00001101 | 015 | 0D | 13 |
| SO | 00001110 | 016 | 0E | 14 |
| SI | 00001111 | 017 | 0F | 15 |
| DLE | 00010000 | 020 | 10 | 16 |
| DC1 | 00010001 | 021 | 11 | 17 |
| DC2 | 00010010 | 022 | 12 | 18 |
| DC3 | 00010011 | 023 | 13 | 19 |
| DC4 | 00010100 | 024 | 14 | 20 |
| NAK | 00010101 | 025 | 15 | 21 |
| SYNC | 00010110 | 026 | 16 | 22 |
| ETB | 00010111 | 027 | 17 | 23 |
| CAN | 00011000 | 030 | 18 | 24 |
| EM | 00011001 | 031 | 19 | 25 |
| SUB | 00011010 | 032 | 1A | 26 |
| ESC | 00011011 | 033 | 1B | 27 |
| FS | 00011100 | 034 | 1C | 28 |
| GS | 00011101 | 035 | 1D | 29 |
| RS | 00011110 | 036 | 1E | 30 |
| US | 00011111 | 037 | 1F | 31 |

| ASCII Char. | EQUIVALENT FORMS | | | |
|---|---|---|---|---|
| | Binary | Oct | Hex | Dec |
| space | 00100000 | 040 | 20 | 32 |
| ! | 00100001 | 041 | 21 | 33 |
| " | 00100010 | 042 | 22 | 34 |
| # | 00100011 | 043 | 23 | 35 |
| $ | 00100100 | 044 | 24 | 36 |
| % | 00100101 | 045 | 25 | 37 |
| & | 00100110 | 046 | 26 | 38 |
| ' | 00100111 | 047 | 27 | 39 |
| ( | 00101000 | 050 | 28 | 40 |
| ) | 00101001 | 051 | 29 | 41 |
| * | 00101010 | 052 | 2A | 42 |
| + | 00101011 | 053 | 2B | 43 |
| , | 00101100 | 054 | 2C | 44 |
| - | 00101101 | 055 | 2D | 45 |
| . | 00101110 | 056 | 2E | 46 |
| / | 00101111 | 057 | 2F | 47 |
| 0 | 00110000 | 060 | 30 | 48 |
| 1 | 00110001 | 061 | 31 | 49 |
| 2 | 00110010 | 062 | 32 | 50 |
| 3 | 00110011 | 063 | 33 | 51 |
| 4 | 00110100 | 064 | 34 | 52 |
| 5 | 00110101 | 065 | 35 | 53 |
| 6 | 00110110 | 066 | 36 | 54 |
| 7 | 00110111 | 067 | 37 | 55 |
| 8 | 00111000 | 070 | 38 | 56 |
| 9 | 00111001 | 071 | 39 | 57 |
| : | 00111010 | 072 | 3A | 58 |
| ; | 00111011 | 073 | 3B | 59 |
| < | 00111100 | 074 | 3C | 60 |
| = | 00111101 | 075 | 3D | 61 |
| > | 00111110 | 076 | 3E | 62 |
| ? | 00111111 | 077 | 3F | 63 |

| ASCII Char. | EQUIVALENT FORMS | | | |
|---|---|---|---|---|
| | Binary | Oct | Hex | Dec |
| @ | 01000000 | 100 | 40 | 64 |
| A | 01000001 | 101 | 41 | 65 |
| B | 01000010 | 102 | 42 | 66 |
| C | 01000011 | 103 | 43 | 67 |
| D | 01000100 | 104 | 44 | 68 |
| E | 01000101 | 105 | 45 | 69 |
| F | 01000110 | 106 | 46 | 70 |
| G | 01000111 | 107 | 47 | 71 |
| H | 01001000 | 110 | 48 | 72 |
| I | 01001001 | 111 | 49 | 73 |
| J | 01001010 | 112 | 4A | 74 |
| K | 01001011 | 113 | 4B | 75 |
| L | 01001100 | 114 | 4C | 76 |
| M | 01001101 | 115 | 4D | 77 |
| N | 01001110 | 116 | 4E | 78 |
| O | 01001111 | 117 | 4F | 79 |
| P | 01010000 | 120 | 50 | 80 |
| Q | 01010001 | 121 | 51 | 81 |
| R | 01010010 | 122 | 52 | 82 |
| S | 01010011 | 123 | 53 | 83 |
| T | 01010100 | 124 | 54 | 84 |
| U | 01010101 | 125 | 55 | 85 |
| V | 01010110 | 126 | 56 | 86 |
| W | 01010111 | 127 | 57 | 87 |
| X | 01011000 | 130 | 58 | 88 |
| Y | 01011001 | 131 | 59 | 89 |
| Z | 01011010 | 132 | 5A | 90 |
| [ | 01011011 | 133 | 5B | 91 |
| \ | 01011100 | 134 | 5C | 92 |
| ] | 01011101 | 135 | 5D | 93 |
| ^ | 01011110 | 136 | 5E | 94 |
| _ | 01011111 | 137 | 5F | 95 |

| ASCII Char. | EQUIVALENT FORMS | | | |
|---|---|---|---|---|
| | Binary | Oct | Hex | Dec |
| ` | 01100000 | 140 | 60 | 96 |
| a | 01100001 | 141 | 61 | 97 |
| b | 01100010 | 142 | 62 | 98 |
| c | 01100011 | 143 | 63 | 99 |
| d | 01100100 | 144 | 64 | 100 |
| e | 01100101 | 145 | 65 | 101 |
| f | 01100110 | 146 | 66 | 102 |
| g | 01100111 | 147 | 67 | 103 |
| h | 01101000 | 150 | 68 | 104 |
| i | 01101001 | 151 | 69 | 105 |
| j | 01101010 | 152 | 6A | 106 |
| k | 01101011 | 153 | 6B | 107 |
| l | 01101100 | 154 | 6C | 108 |
| m | 01101101 | 155 | 6D | 109 |
| n | 01101110 | 156 | 6E | 110 |
| o | 01101111 | 157 | 6F | 111 |
| p | 01110000 | 160 | 70 | 112 |
| q | 01110001 | 161 | 71 | 113 |
| r | 01110010 | 162 | 72 | 114 |
| s | 01110011 | 163 | 73 | 115 |
| t | 01110100 | 164 | 74 | 116 |
| u | 01110101 | 165 | 75 | 117 |
| v | 01110110 | 166 | 76 | 118 |
| w | 01110111 | 167 | 77 | 119 |
| x | 01111000 | 170 | 78 | 120 |
| y | 01111001 | 171 | 79 | 121 |
| z | 01111010 | 172 | 7A | 122 |
| { | 01111011 | 173 | 7B | 123 |
| \| | 01111100 | 174 | 7C | 124 |
| } | 01111101 | 175 | 7D | 125 |
| ~ | 01111110 | 176 | 7E | 126 |
| DEL | 01111111 | 177 | 7F | 127 |

STD-1.1, 60061

1. String Constants (Literals):

   "ABCdef. . . 0123. . .#$%"

   "The quick brown fox"

2. String Variables:

   DIM Name$[80],Addr$[160]

   (Dollar sign=string) (Max length of string)

3. String Arrays:

   DIM City_State$(50)[132]

   50 element array of 132 character string elements

D1-52

NOTES

4. Substrings:

   Allow access to a specified segment of a string

   A$=Address$ [26, 50]

   Starting Character Position — Ending Character Position

   B$=Name$(30)[1, 15]

   Array Element Number

D1-53

NOTES

NOTES

Alternate substring access:

```
A$=Address$ [26; 25]
```

Starting Character Position

Substring Character Count

For example:

```
10 ADDRESS$="1611 W. Seventh"
20 PRINT Address$ [1,4]
30 PRINT Address$ [6;10]
40 END
```

```
1611
W. Seventh
```

D1-54

NOTES

Now add these lines:

```
31 Address$[6]="N. Forty"
32 PRINT Address$
33 Address$[9,11]="Fif"
34 PRINT Address$
```

Line 31: replaced entire remainder of string

Line 33: replaced only characters 9 through 11

D1-55

# STRING OPERATIONS

1. Concatenation(&):

   Building big strings from little ones

```
First$ = "John"
Last$  = "Barleycorn"
Name$  = First$&" "&Last$
```

concatenate

Now Name$ looks like

John Barleycorn

D1-56

NOTES

2. String Length (LEN):

   Returns the number of characters currently in the string or string expression

   Length = LEN(String$)

```
10 A$="1234567890123"
20 PRINT LEN(A$)
30 PRINT LEN(X$)
40 PRINT LEN(A$&"Text"&A$)
50 END
```

```
 13
  0
 30
```

D1-57

NOTES

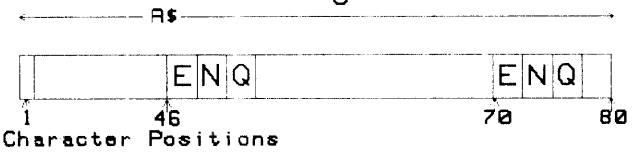
NOTES

3. String position (POS):

Returns the location of one string within another.

Syntax:
Position=POS(Look_in$,Look_for$)

Given a string - -




```
PRINT POS (A$,"ENQ")
    46
PRINT POS (A$[POS(A$,"ENQ")+1],"ENQ")
    23
```

D1-58

NOTES

POS Continued

Example

```
10 Answer$ = "NO"
20 INPUT "Please enter YES or NO
   (NO is default)",Answer$
30 IF POS(Answers$,"Y") OR
   POS(Answers$,"y") THEN 100
40 ! "NO"processing
        .
        .
        .
100 !"YES" processing
```

D1-59

4. String relationals

<, <=, =, >=, >, <>

Strings are compared character - by - character according to the numeric values of their ASCII codes (Refer to the ASCII table)

```
IF A$ > B$ THEN Greater

If Answer$ <> "NO" THEN Maybe
```

D1-60

NOTES

RELATIONALS Continued

Items of Note:

a. Two "identical" strings of of unequal length are not equal!

```
"YES" <> " YES"
```

b. Uppercase characters have lower values than lower case

```
"Computer" < "computer"
```

c. String sorts almost always surprise you.

D1-61

NOTES

NOTES

5. String-Numeric Conversions

a. ASCII code of a character:

X=NUM("A")

```
10 A$="ABCDEFabcdef"
20 FOR I=1 TO LEN(A$)
30 PRINT NUM(A$[I,I]),A$[I,I]
40 NEXT I
50 END
```

D1-62

NOTES

String-Numeric Conversions

b. Character of an ASCII code:

X$=CHR$(65)

```
10 FOR X=0 TO 255
20 PRINT CHR$(X)
30 NEXT X
40 END
```

Why the beep ?

D1-63

String-Numeric Conversions

c. Number from a string of numeric characters:
X=VAL("123.45 E10")

```
10 A$="May 15, 1973"
20 PRINT VAL(A$)
30 PRINT VAL(A$[POS(A$," ")])
40 PRINT VAL(A$[POS(A$,",")+1])
50 END
```

D1-64

NOTES

String-Numeric Conversions

d. String from a number:
Pi$=VAL$(PI)

```
10 PRINT PI;"X"
20 PRINT VAL$(PI);"X"
30 END
```

NOTE: no leading blanks,
no trailing blanks

D1-65

NOTES

# EXERCISE 1

Write a program that accepts an input of the form MM/DD/YY and converts it to the corresponding month, day, and year in the form Month Day,Year. For example:

Input= 06/21/81

Output= June 21, 1981

1. Use string array to hold the names of the months, then index into the array with the number of the month that was input.

2. Use the POS function and substrings to extract the day and the year information from the input string.

3. Consider the difference between the following two statements, where A$ is "ab/cd/ef" :

```
X = POS(A$, "/")
Y = POS(A$ [4] ,"/")
X = 3, Y = 2
```

# EXERCISE 2

Write a search-and-replace subroutine that replaces all occurrences of a "seek for" string with a "replace with" string. The subroutine will look in a string array for strings to replace. Print out any string elements your routine modifies.

Save your subroutine on disc (SAVE"TEMP"), then GET "SCH_RPL" , then append your subroutine to the end of the program SCH_RPL. Remember to include an END statement at the end of the program then try it out.

Your subroutine's name : Search_replace
The string array's name: Search_in$
Number of elements in Search_in $ : Num_elem

Example Subroutine format:

```
10 Search_replace : ! This subroutine searches for ...
20    INPUT "Search string?", A$
        .
        .
        .
110 RETURN
```

# PROGRAM STRUCTURE AND CONTROL

Objective:

- Select appropriate program structure for a given algorithm
- Write program segments to provide event response
- Utilize debugging tools to analyze errant programs

D2-1

NOTES

# AGENDA

Program Control

- Decisions
- Branching
- Looping

Event Response

Program Modules

- Subroutines
- Subprograms

Debugging Tools

D2-2

NOTES

NOTES

# STRUCTURED PROGRAMMING

Three Elements:

- Program Control
  Deciding what to execute

- Program Structure
  Separating tasks into modules
  One module (block)=1 problem

- Iteration
  Executing a block over and over

Select the right tool for the job

D2-3

NOTES

# PROGRAM CONTROL

- Normal Execution:
  Linear ascending-order line #'s

```
10 BEEP
20 WAIT 2
30 BEEP
40 END
```

D2-4

# BRANCHING

(Changing the sequence)

1. Unconditional
   GOTO 1040
   GOTO Forced_exit

2. Conditional (Decisions)
   IF Value>Limit THEN Too_big

3. Computed
   ON Command GOTO P_1,P_2,P_3

D2-5

NOTES

# COMPUTED BRANCH NOTES

```
50 ON Value GOTO 100,200,300,400,500
```

- The ON statement causes a branch to the appropriate line. In this example, if Value=1 then the program goes to line 100.

- What if Value=0 ? If VALUE=6 ?

D2-6

NOTES

NOTES

## CONDITIONAL BRANCH NOTES

- The IF statement tests for non-zero (True) or zero (False)
- The IF statement allows an executable statement or an implied GOTO

```
IF BIT(A,3) THEN PAUSE
        Executable statement

IF B>A THEN Greater
        Implied GOTO Greater
```

D2-7

NOTES

## CONDITIONAL PROGRAM BLOCKS

Two Conditions:

```
100  Bits:! Print a number
            in binary
110  FOR Count = 15 TO 0 STEP-1
120  IF BIT (Number,Count) THEN
130    PRINT "1";   !True block
140  ELSE
150    PRINT "0";   !False block
160  END IF
170  NEXT Count
180  PRINT
```

D2-8

# CONDITIONAL PROGRAM BLOCKS

Multiple Conditions:

```
100 SELECT Function
110 CASE 1
120   Y=SIN(X)      ! Function=1
130 CASE 2
140   Y=COS(X)      ! Function=2
150 CASE 3
160   Y=TAN(X)      ! Function=3
170 CASE 4
180   Y=EXP(X)      ! Function=4
190 CASE ELSE
200   Y=X           ! Anything else
210 END SELECT
```

Strings can be selected also

D2-9

NOTES

# CONDITIONAL PROGRAM BLOCKS

Multiple Ranges:

```
100 SELECT Reading
110 CASE 0
120   PRINT "Zero"
130 CASE 0 TO 10
140   PRINT "Range OK"
150 CASE 10 TO 2E300
160   PRINT "Overrange"
170 CASE ELSE
180   PRINT "Polarity error"
190 END SELECT
```

Character ranges can also be tested

D2-10

NOTES

## EXERCISE 3

Write a program that uses the SELECT CASE structure to write the binary representation of a number. You can take the basic contents of the IF/THEN/ELSE and modify it to be a SELECT/CASE if you wish.

## EXERCISE 4

Write a program to input a string of mixed uppercase and lowercase characters then convert that string to all uppercase characters. Use the SELECT/CASE construct to determine whether the character being analyzed is upper or lower case, and subtract 32 from the ASCII code of any characters in the range of "a" to "z:.

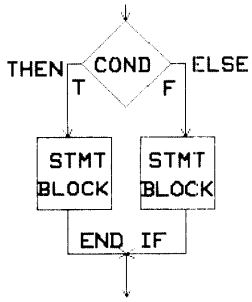
```
 Input : Yes
Output : YES
```

Modify the program to either convert the string to uppercase or to lowercase characters, depending upon the value of some variable. For example:

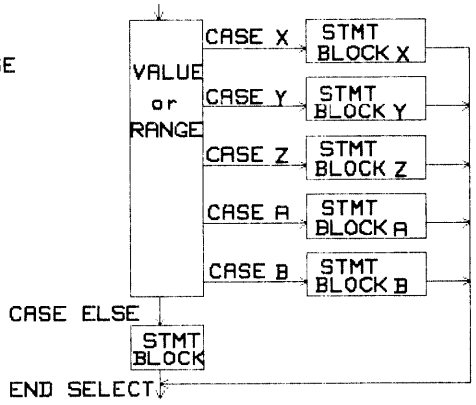
```
Function $ = "UPC"
   Input = no
  Output = NO

Function $ = "LWC"
   Input = WILDERNESS
  Output = wilderness
```

# CONDITIONAL FLOW CHARTS (Program flow)

IF-THEN-ELSE




SELECT-CASE




D2-11

NOTES

# ITERATION

- Iteration is repetitive loop execution
- Termination test can occur in one (or more) of 3 places

1. Test at end of loop (always executes once)
2. Test at beginning of loop (may or may not execute loop)
3. Test in middle of loop (at least part of loop is executed)

D2-12

NOTES

NOTES

## END-TEST ITERATION

(SPECIAL CASE)

FOR Counter = Init TO Final STEP Increment

Statement Block

NEXT Counter

1. An initial test is made to check if Init exceeds Final. If so, the loop is never executed !
2. Loop is terminated if Counter exceeds Final when NEXT is executed

D2-13

NOTES

## END-TEST ITERATION

REPEAT

Statement Block

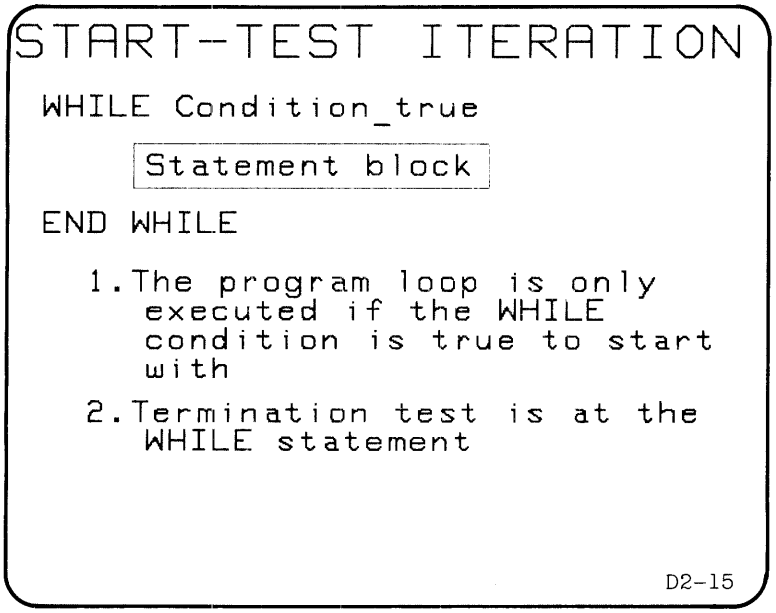
UNTIL Condition true

1. The program loop is always executed once
2. Termination test is at the UNTIL statement

D2-14

## EXERCISE 5

Rewrite the program that prints out the binary representation of a number, but use the REPEAT/UNTIL method of iteration instead of the FOR/NEXT loop.

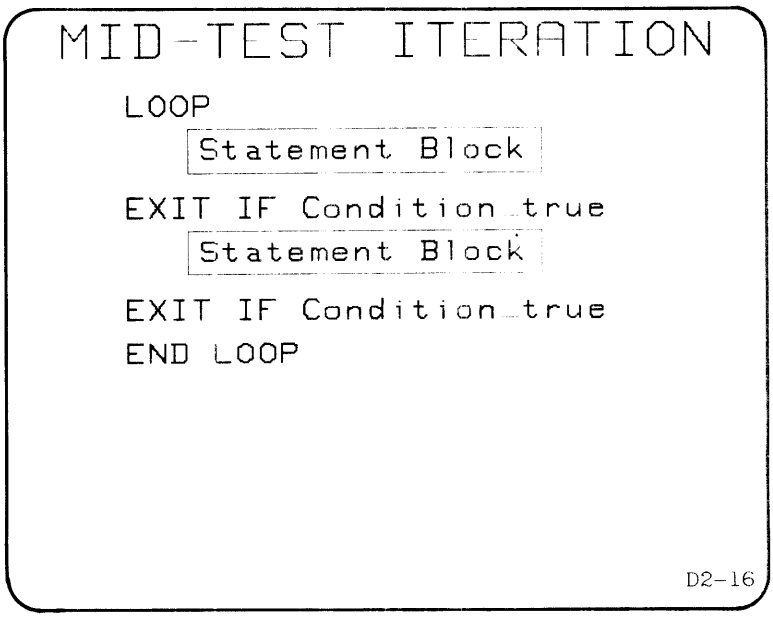



**NOTES**

## EXERCISE 6

Rewrite the program that prints out the binary representation of a number, but use the WHILE/ENDWHILE method of iteration instead of the FOR/NEXT loop. (Or simply modify the previous REPEAT program.)

**NOTES**



# LOOP STATEMENT

1. Part of the loop is always executed unless an EXIT IF statement precedes the loop statement

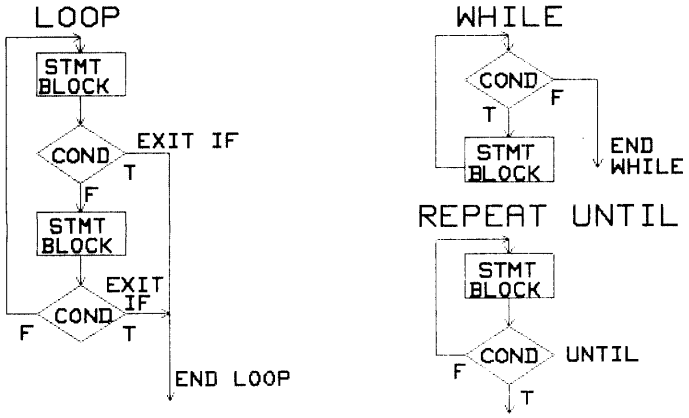
2. Termination test is at the EXIT IF statement

3. Multiple exit conditions can be programmed, as well as multiple exit points to break up the loop

D2-17

NOTES

# LOOPING CONSTRUCT FLOW CHARTS




D2-18

NOTES

NOTES

## PROGRAM MODULES

- Subroutine - a shared program segment not having a separate program environment (or context)
- Subprogram - a separate program segment with its own local program environment (context) isolated from the main program and other subprograms

D2-19

NOTES

## SUBROUTINES

- Segment of program lines ending with RETURN
- Called (invoked) by executing GOSUB statement

```
GOSUB 150
GOSUB Beep.sub
```

D2-20

```
        .
        .
        .
    100 GOSUB Count
```

Acts like inserted code

```
1400 Count: FOR I=1 TO 10
1410  Sum=Sum+I
1420  NEXT I
1430 RETURN
```

```
    110 Next_line: PRINT Sum
        .
        .
        .
```

D2-21

NOTES

## SUBROUTINES

- Subroutine calls can be nested to any depth
- A subroutine can call itself
- Subroutine variables are not local (private), but are global to the main program

D2-22

NOTES

NOTES

## SUBPROGRAMS AND USER FUNCTIONS

Complete, separate program

Local environment = "context"

- Variable names
- Key definitions
- Line labels
- Data storage

SUB AND DEF FNUser
(CALL) (FNUser)

D2-23

NOTES

## SUBPROGRAMS

- Program modules = "Top-down programming"
- Execute mini-tasks
- Modules are designed independent of calling program: interchangeable from program to program
  ∴ Libraries

D2-24

## SUBPROGRAM GEOGRAPHY

Main program

```
Var1 = FNUser_1
CALL Module_1
END
```

Function Subprogram

```
DEF FNUser_1
  RETURN Result
FNEND
```

"Sub" Subprogram

```
SUB Module_1
  SUBEXIT
SUBEND
```

D2-25

NOTES

## A SIMPLE FUNCTION

```
1! Print Numbers and Squares
10 FOR I = 1 TO 10
20 PRINT I, FNSqr(I)
30 NEXT I
40 END
50  DEF FNSqr(X)
60  RETURN X * X
70  FNEND
```

D2-26

NOTES

NOTES

## A SIMPLE SUBPROGRAM

```
1! Print the reverse of a string
10 INPUT A$
20 CALL Rev(A$)
30 END
40 SUB Rev(X$)
50  FOR I=LEN(X$) TO 1 STEP -1
60  PRINT X$[I,I];
70  NEXT I
80 SUBEND
```

D2-27

NOTES

## SUBPROGRAM COMMUNICATION

IN:
- Pass Parameters
- COM

OUT:
- Pass Parameters
- RETURN value
- COM

D2-28

# PASS PARAMETERS

- Pass Parameter List

```
CALL Sort_array (Num_elm,Array(*),INTEGER Max)
```

- Formal Parameter List

```
SUB Sort_array (Array_size,Srt_ary(*),INTEGER Hi)
```

- Parameter lists match:
  - Position
  - Type (REAL, INTEGER, Array, String)
  - Number

D2-29

NOTES

# PASS PARAMETERS

Pass by Value (expression)
- "Copy" = constant value
- Allows type-conversion
- One-way only: IN!

```
CALL Compare ((Original),(Final),Res)
             By Value          By Ref
```

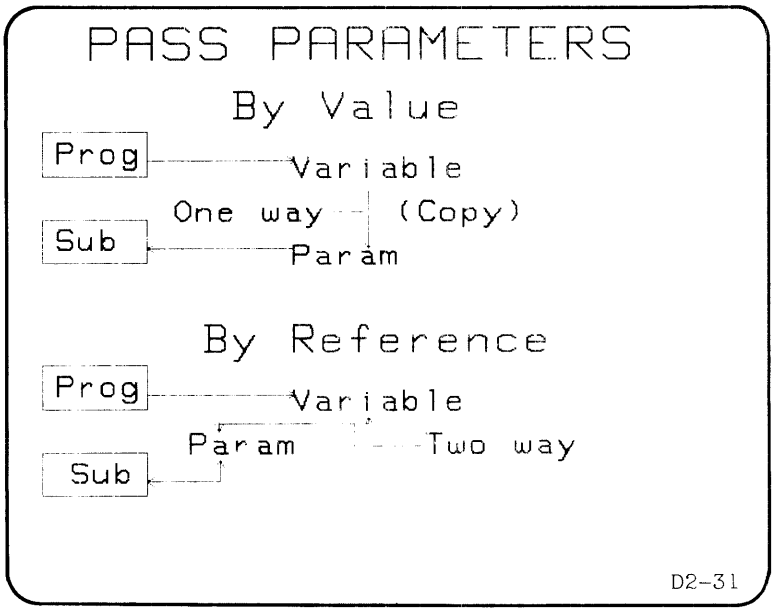
Pass by Reference
- Pointer to the variable itself
- Type-matching required
- Two-way: IN and OUT

```
CALL Sort (Array(*), (Numb_elem))
           By Ref      By Value
```

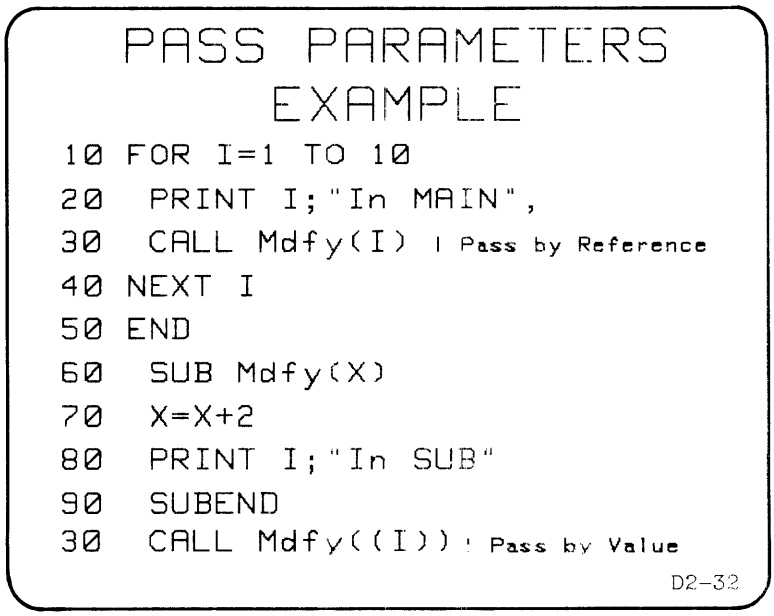
D2-30

NOTES

NOTES

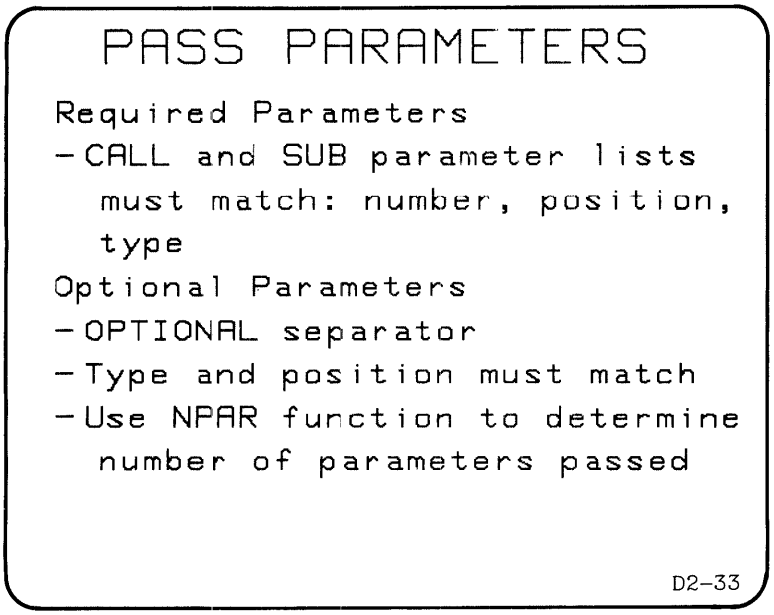

PASS PARAMETERS
By Value
Prog
Variable
One way
(Copy)
Sub
Param
By Reference
Prog
Variable
Param
Two way
Sub
D2-31

NOTES


PASS PARAMETERS
EXAMPLE
10 FOR I=1 TO 10
20 PRINT I;"In MAIN",
30 CALL Mdfy(I) ! Pass by Reference
40 NEXT I
50 END
60 SUB Mdfy(X)
70 X=X+2
80 PRINT I;"In SUB"
90 SUBEND
30 CALL Mdfy((I)) ! Pass by Value
D2-32

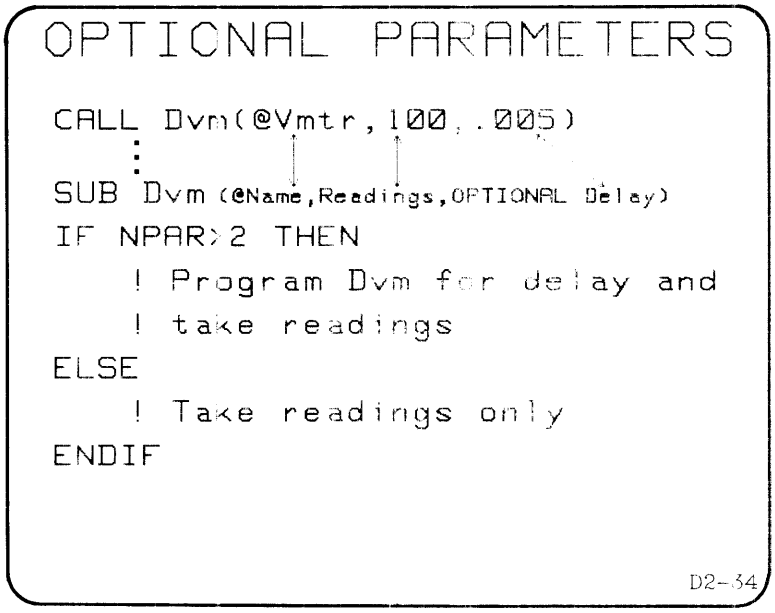
GET "VAL_REF" and run the program. See if you can figure out how the program looks by just studying its output. Can you explain the last block of output? What happened to I?

LIST the program and see if it looks the way you predicted. Check the pass-by-value and pass-by-reference function calls. Look at the two functions and analyze how they affect their pass parameters.




**NOTES**

**NOTES**




EXERCISE 8

GET "NPAR" and try running the program. See if you can analyze the problem. After you have the types matched, re-run the program. Be sure you understand the flow of the program before moving on to the next slide. (In the first call, C is an "optional" parameter. In the second call, the optional parameter was not passed, and the sub-program executed a different statement block.)

Which construct might be more appropriate for multiple optional parameters than the IF/THEN/ELSE construct?

## COMMON

Blank COM

- Declared in main program
- Accessible from subs only with matching COM statement
  - Types and position of variables must match
  - Names of variables needn't match

```
COM Initlzd,X$[300],INTEGER I,J
```

D2-35

NOTES

## LABELED COM

- Can be declared in subprograms not necessary in main
- COM/name/must match to have access
- Other requirements are same as blank COM
- Provides unique, "private" storage space for subprogram between invocations

```
COM/Private/X,Y,Z$
```

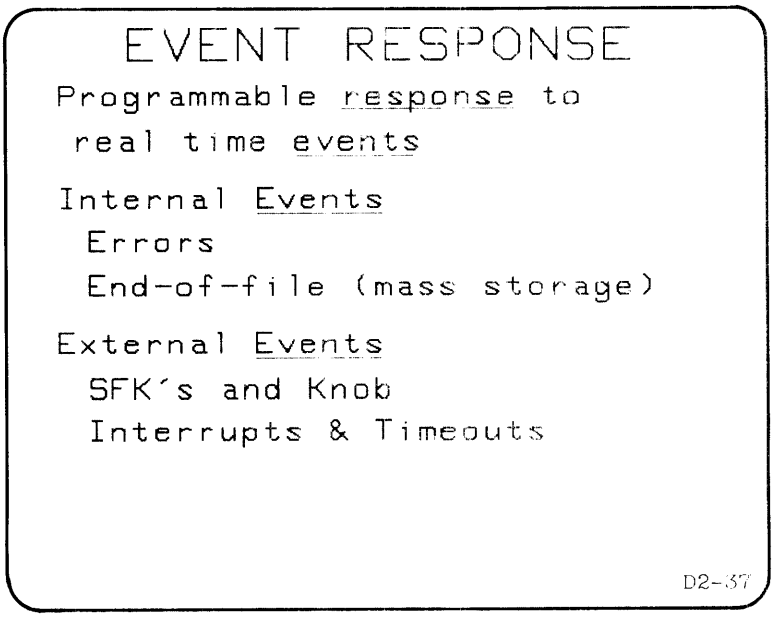
D2-36

NOTES

GET "LABEL_COM" and run the program. Note what happened to variables A,B, and C across the main program and the two subprograms. What happened to A,B, and C after calling the first subprogram?) Was the main program able to access variables L,M, and N of the labeled common? (If it could, the values assigned to L,M, and N in the first subprogram would be printed by main after the call.)

Was the second subprogram able to access all variables?

**NOTES**



# EVENT RESPONSE

End-of-Line Branching

A test is made by the operating system after EVERY program line to see if an event has occured requiring service

```
1050 Array(Index) = SIN(Index)
                          Test
1060 BETA = FNUser(Alpha)
                    Test
1070 NEXT Index
          Test
```

D2-38

NOTES

# EVENT RESPONSE

- If an event has occured, and the program has defined the end-of-line branch, that branch is taken

  ON ERROR GOSUB Recover

- ERROR is an event, and GOSUB is a response

- The user program defines and controls end-of-line branches

D2-39

NOTES

NOTES

## EVENT RESPONSE

Response Techniques
- GOTO
- GOSUB
- CALL
- RECOVER

Response Control
- ENABLE/DISABLE
- Priority
- Context

D2-40

NOTES

## EVENT RESPONSE

What is RECOVER ?

```
ON ERROR RECOVER Abortall
```

- RECOVER is a GOTO
- RECOVER restores the program context to the point where it was defined
- RECOVER remains active even though context changes to a subprogram

D2-41

# SYSTEM PRIORITY

- Normal (default) priority=0
- An event with a higher priority can cause an immediate branch (response)
- System priority becomes the priority assigned to the event
- System priority remains set at the new priority until exiting the service routine for the event
- A "GOTO" type response does not change system priority

D2-42

NOTES

# PRIORITY/CONTEXT TABLE

Will an immediate branch be taken?

| Branch type | Relative Event Priority | Executing Main Program | Executing Subprogram |
|---|---|---|---|
| GOTO | Lower | NO[1] | NO[2] |
| | Higher | YES | NO[2] |
| GOSUB | Lower | NO[1] | NO[2] |
| | Higher | YES | NO[2] |
| CALL | Lower | NO[1] | NO[1] |
| | Higher | YES | YES |
| RECOVER | Lower | NO[1] | NO[1] |
| | Higher | YES | YES |

1. Branch is deferred until system priority drops
2. Branch is deferred until main context is restored

D2-43

NOTES

NOTES

## SPECIAL FUNCTION KEYS

Define custom responses to SFK keypresses

Assign softlabels to key label areas

Very friendly user interface

D2-44

NOTES

## SPECIAL FUNCTION KEYS

Can assign system priority to key service

Example

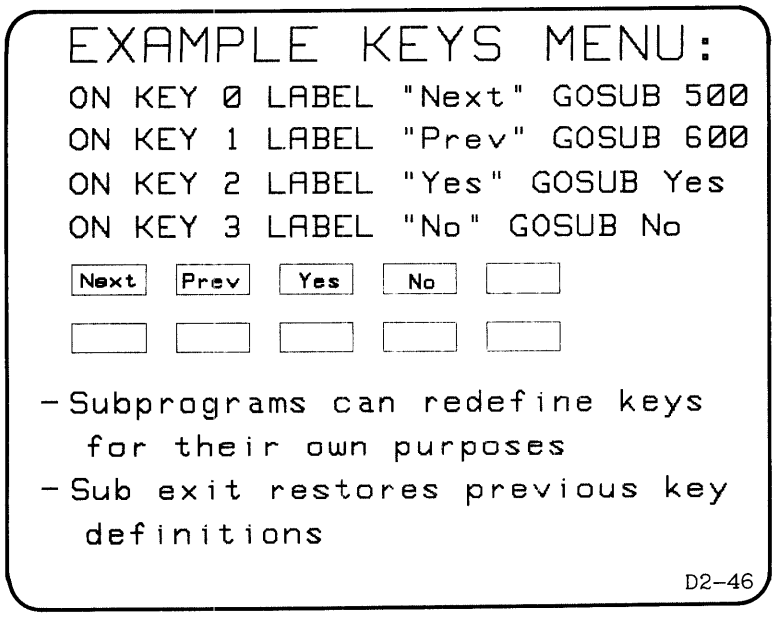
```
ON KEY 4 LABEL "Restart", 11
                    GOSUB Key_4
```

Significant programs should have a "bail-out" SFK

```
ON KEY 9 LABEL "Abort", 15
                    RECOVER Crash
```

Note: ON KEY service is temporarily disabled by INPUT, LINPUT!

D2-45






EXERCISE 10

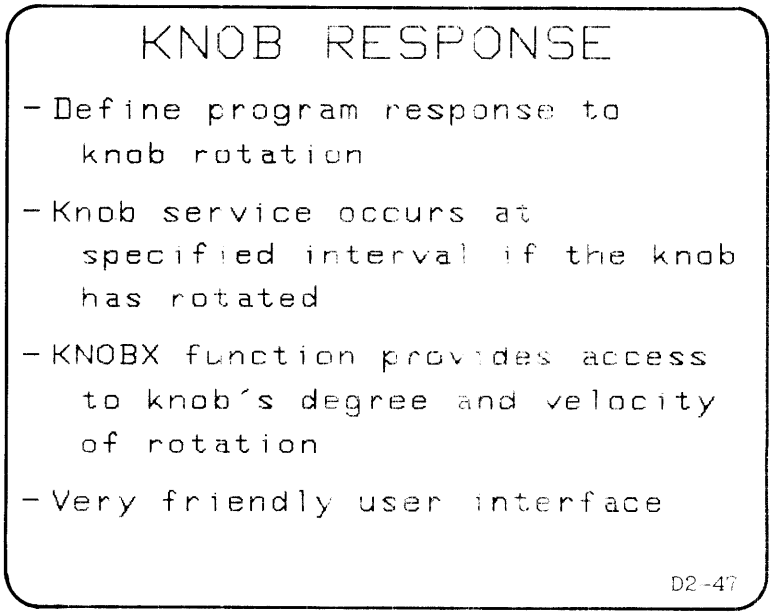
GET "KEYS1" and list the program. Note how the special keys are set up, labels defined, and priorities established. Note especially what happens in Sub3: a key is re-defined with a new priority, label, and response. Run the program and try various combinations of keys, noting which keypress get immediate response, and which ones are deferred. Try pressing a key twice to see if it's service routine is executed twice (watch the counter).

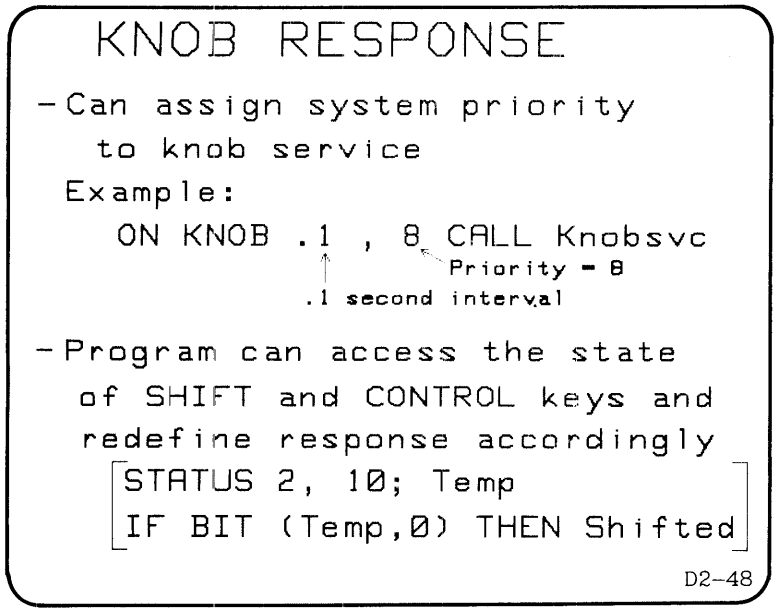
GET "PRIORITIES" and list the program. Note the key redefinition in Pri10. Will pressing K4 while Pri10 is executing cause an immediate branch to Pri 14?

Run the program. Press K0, K2, and K4 in rapid succession. Can you explain why K2 caused an immediate response while K4 did not? Why was the Priority 14 subprogram executed before returning back to the Priority 5 subprogram?

Try pressing the K5 "GOSUB" key from the main program. Now try pressing it from any of the subprograms. Why is K5 service deferred even though it has the highest priority? Can any of the subprograms interrupt the priority 15 subroutine service?

**NOTES**






NOTES

EXERCISE 12

Write a program that responds to knob rotation and executes BEEP with varying frequencies depending on the degree of rotation of the knob.

Hints:

1. Use absolute value of KNOBX.
2. You may need to multiply the result of #1 above by 10 or so to obtain suitable values for BEEP frequencies.
3. Use the reference manual to look up statement syntaxes, if necessary.

NOTES

## ERROR RECOVERY

- Set up a user-programmed response to errors
- Deal with operator mistakes in a "friendly" manner

ON ERROR GOTO Rcvr_1

Can be GOTO, GOSUB, CALL, RECOVER

D2-49

NOTES

## ERROR RECOVERY

ERRL: Boolean test for line number or line label of error

ERRN: Return the most recent error number

In Use: set up separate ON ERROR for each possible critical error location

(CREATE, ASSIGN, etc.)

D2-50

```
 10  ON ERROR GOTO Error
 20   INPUT "SELECT 1, 2, or 3",X
 30   ON X GOTO One,Two,Three
 40 Error:  IF ERRL(30) THEN
 50   PRINT "Please enter more
            carefully"
 60  ELSE
 70   PRINT "Unexpected error"
 80   STOP
 90  END IF
100  GOTO 10
110 One:
```

D2-51

NOTES

## ELEMENTAL CONTROL

Temporary Halts:

Timed: WAIT

```
WAIT 50 ! 50 seconds
```

Operator-controlled: PAUSE

```
PAUSE ! CONTINUE key
```

D2-52

NOTES

NOTES

## ELEMENTAL CONTROL

When I say "WHOA" I mean "Whoa!"

Program Temination:

- STOP requires RUN to restart program
- End is same as STOP but it also delimits main program

D2-53

NOTES

## DEBUGGING

When all else fails...

- Use ON ERROR for graceful program recovery
- Put PRINTs all over the place
  - Subprograms
  - FOR-NEXT loops
  - Key I/O statements
- Turn PRINTALL on

D2-54

## DEBUGGING

If you still cannot believe what you're seeing...

1. STOP/RESET and EDIT calls executing line into display

2. TRACE ALL 10,9999
   - Line numbers
   - Variable assignments
   - Display or PRINTALL printer

D2-55

NOTES

## DEBUGGING

3. TRACE PAUSE Line_label
   Executes PAUSE before executing specified line
   (Waits for CONTINUE)

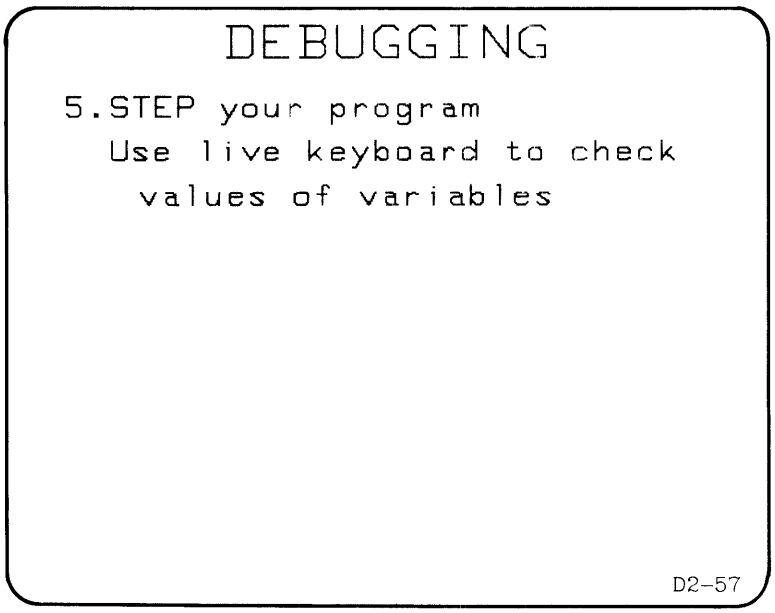
4. TRACE OFF
   Cancels all trace activity

Insert TRACEs as program lines or execute from keyboard

D2-56

NOTES

NOTES


DEBUGGING
5.STEP your program
Use live keyboard to check
values of variables
D2-57

# MASS STORAGE PROGRAMMING

Objectives:

- Save and retrieve programs, subprograms
- Save and retrieve data
- Emulate I/O devices

D3-1

NOTES

# MASS STORAGE

(An Electronic Filing Cabinet)

Mass Storage Devices

- Floppy disc drive
- Mag tape drive
- Hard disc drive

Mass Storage Media

- Floppy discs
- Magnetic tape
- Disc platters

D3-2

NOTES

NOTES

## MEDIA SPECIFIER

":INTERNAL,4,0"

- Indicates the device/media to use for a mass storage operation

String expression including colon and mass storage unit specifier (msus)

```
Media$ = ":" & "HP9895,700,0"
                 msus
```

D3-3

NOTES

## UNIFIED MASS STORAGE

- Identical statements can access different devices
- System designer has flexibility without risk of software incompatibility
- Requirement: one statement to redirect all mass storage operations

D3-4

## UNIFIED MASS STORAGE

MASS STORAGE IS Media$

- Defines default (implicit) mass storage device
- Can be overridden by an explicit device specifier

```
MASS STORAGE IS ":HP82901,707,0"
MASS STORAGE IS ":HP9895,700"
MASS STORAGE IS ":REMOTE"
MASS STORAGE IS ":CS80,700,1"
```

D3-5

NOTES

## THE FIRST OPERATION

- All magnetic media must be initialized before first use

```
INITIALIZE ":INTERNAL,4,0"
```

- A disc need be initialized only once. It is then ready to store programs and data

D3-6

NOTES

NOTES

## PROGRAM STORAGE STATEMENTS

| STORE | RETRIEVE |
|---|---|
| STORE | LOAD |
| RE-STORE | LOADSUB |
| SAVE | GET |
| RE-SAVE | |
| STORE BIN | LOAD BIN |
| RE-STORE BIN | |

D3-7

NOTES

## AN EXAMPLE STATEMENT

STORE"TRIANGLE"

Program file name

- The program file name identifies the program being stored in a "file"
  (Think of a file folder)
- The file name must be unique
  (Think of a file folder label)

D3-8

# FILES

(Electronic file folders)

- Used to store related information: program code mailing lists, instrument readings
- Each file is given a unique name of up to 10 characters

D3-9

NOTES

# FILE SPECIFIERS

- A file specifier consists of a file name plus an optional protect code plus a media specifier

Example:

"PROG1<SE>:INTERNAL"

file name  protect code  media specifier

```
Name$&"<"&Protect$&">"
```

D3-10

NOTES

NOTES

## FILE TYPES

| Type | Contents | To Construct | To Retrieve |
|---|---|---|---|
| PROG | Internal Program Code | STORE<br>RE-STORE | LOAD<br>LOADSUB |
| ASCII | Program ASCII source | SAVE<br>RE-SAVE | GET |
| | String data | CREATE ASCII OUTPUT | ENTER |
| BDAT | Data | CREATE BDAT OUTPUT | ENTER |
| BIN | Binary Program | STORE BIN | LOAD BIN |
| SYSTEM | Operating System | "Copy" Utility | Boot System |

D3-11

NOTES

## PROGRAM STORAGE SPECIFICS

- STORE creates a PROG file and writes program and binaries to the file in internal form
- RE-STORE does a STORE, then removes old file of same name from disc (used to update PROG files)

```
STORE "MAIN.1"
RE-STORE "MAIN.1"
STORE "SUB.1:HP9895,700,1"
```

D3-12

# PROGRAM STORAGE SPECIFICS

- SAVE creates an ASCII file and writes all or part of the program to the file as data
- RE-SAVE does a SAVE, then removes old file of same name from the disc (Update)

```
SAVE "EDITOR"
RE-SAVE "EDITOR"
SAVE File_name$ & Media$
```

D3-13

NOTES

# STORE vs. SAVE

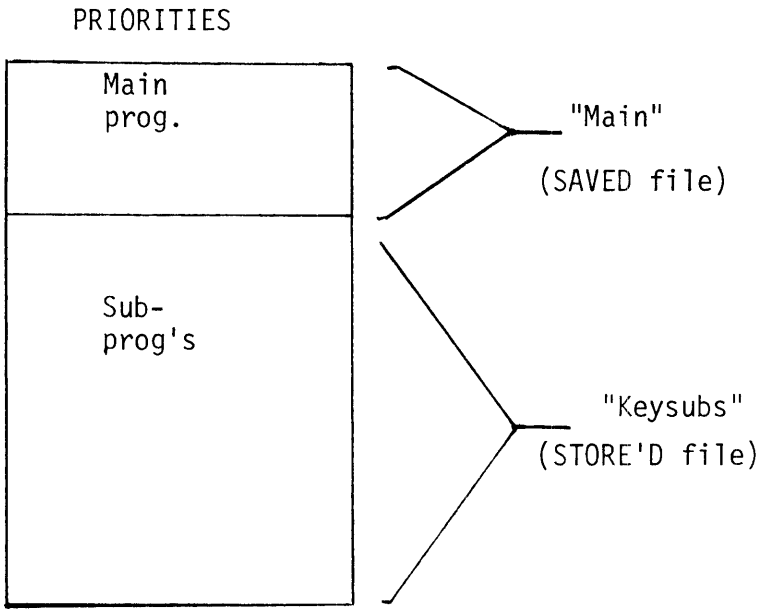
| STORE | SAVE |
|---|---|
| Internal format | ASCII source format |
| PROG file | ASCII file |
| Entire program plus binaries | All or part of program only |
| 9826 readable only | Compatible with other devices |
| Not accessible as data | Accessible just like ASCII data |
| Fast | Slow |

D3-14

NOTES

GET "PRIORITIES" and edit the program.

Use combinations of SAVE, GET, and STORE to build two files from the program: one file the main program, the other file the key-service subprograms.




The challenge here is to determine how to produce a STORE'd program file of just the key-service subprograms. You will be using these files in a short time.

## PROGRAM RETRIEVAL SPECIFICS

- LOAD brings a PROG file into memory replacing any program and variables already resident except for COM

- You can specify execution to automatically begin at any line

```
LOAD "MAIN_1", 10
LOAD "MAIN_2", Start_up
```

D3-15

NOTES

## PROGRAM RETRIEVAL SPECIFICS

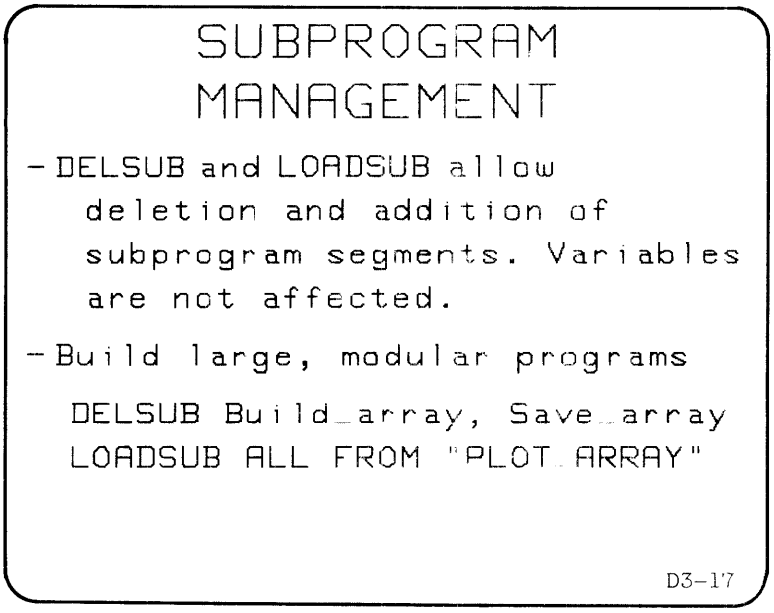
- GET reads, syntaxes, and stores an ASCII file into memory, replacing all variables except those in COM
- GET can overlay all or part of the resident program

```
GET "READ DATA", Add_code, Run_line
GET "NEXT prog", Last_line, Next_line
GET File$, 1,1
GET "TRIANGLE"
```

D3-16

NOTES

NOTES




EXERCISE 14

GET "Main" , the main-program file you saved previously, and modify it so that the program will run correctly. (You should need only to insert one line to be able to run it.) Be sure to re-save your main program if you want a running example for posterity.

## LOAD vs GET vs LOADSUB

| LOAD | GET | LOADSUB |
|---|---|---|
| PROG file | ASCII file | PROG file |
| Variables lost except COM | Variables lost except COM | All variables unchanged |
| Entire Program | All or part of program | Subprogram Segment(s) |
| Fast | Slow | Fast |

D3-18

**NOTES**

EXERCISE 15

GET "GETLOADSUB" and list the program

Before running the program, try to predict what the values of the variables will be when the PRINT statements are executed. Run the program. Did it do what you expected?

NOTES

# FILE MANAGEMENT

General-purpose file management and maintenance tools:

- File Directories (CAT)
- File Protection (PROTECT)
- File Deletion (PURGE)
- File Renaming (RENAME)
- File Copying (COPY)
- Disk Packing (Utility)

D3-19

NOTES

# FILE DIRECTORIES

Directory: a table of information of all existing files on an initialized media

| FILE NAME | PRO | TYPE | REC/FILE | BYTE/REC | ADDRESS |
|---|---|---|---|---|---|
| FBACKUP | * | PROG | 42 | 256 | 65 |
| CBACKUP | | ASCII | 56 | 256 | 107 |
| TRIANGLE | | PROG | 4 | 256 | 163 |
| READINGS | | BDAT | 150 | 256 | 167 |

D3-20

## ACCESSING THE DIRECTORY

- CAT lists media file directory information
- Unrecognized file types are listed as numbers

```
CAT
CAT ":INTERNAL" TO #701
```

D3-21

NOTES

## PROTECTING FILES

- PROTECT offers write-protect capabilities for PROG, BIN, BDAT files
- Old protect codes can be updated or removed

```
PROTECT "FINAL", Protect$
PROTECT "Main<OLD>", "NEW"
PROTECT "MAIN<OLD>",""
```

D3-22

NOTES

NOTES

## DELETING FILES

- PURGE deletes the directory entry for the specified file
- To purge protected files, include the file specifier's protect code

```
PURGE "REV_1:INTERNAL"
```

D3-23

NOTES

## RENAMING FILES

- RENAME changes the name of the specified file
- To rename protected files, include the old file specifier's protect code

```
RENAME "REV_2" TO "FINAL"
```

D3-24

# DATA STORAGE AND RETRIEVAL

File Structure
File Types
Access Methods
End-of-File Detection
Control and Status

D3-25

NOTES

# FILE STRUCTURE

- A file is composed of one or more records
- There are three record types:
  - Physical record
  - Defined record
  - Logical record

D3-26

NOTES

NOTES

## PHYSICAL RECORDS

- Length of a physical record is defined by the media when initialized
  DISC:256 bytes per record
- Physical record is minimum information transfer between operating system and media
- Fixed length records

D3-27

NOTES

## DEFINED RECORDS

- Length of a defined record is set by the user when creating a BDAT file
- Length of a defined record should be appropriate for accessing a convenient or logical "chunk" of data
- Fixed length records

D3-28

## LOGICAL RECORDS

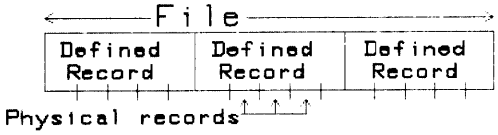
- Length of a logical record is set by the format and content of data being stored
- Length of a logical record may vary from record to record
- Variable length records

D3-29

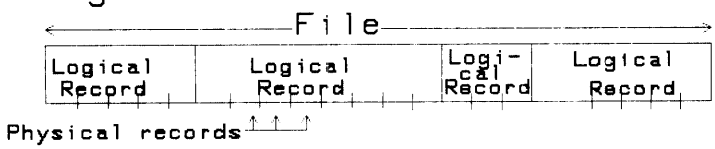
NOTES

## FILES AND RECORDS

Defined records:




Logical records:



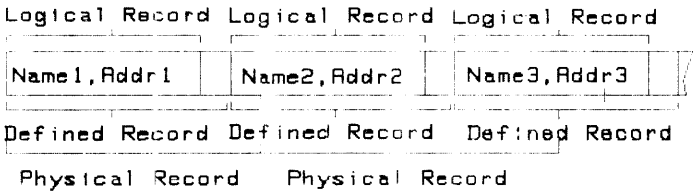

D3-30

NOTES

NOTES

# EXAMPLE RECORDS

- Data: Mailing list, 5 names and addresses
- Defined Record: 200 bytes (maximum entry length)
- Logical Record: One name and address




D3-31

NOTES

# FILE TYPES

Four BASIC file types:

| File Type | Record Types |
|---|---|
| PROG | Not meaningful |
| BIN | Not meaningful |
| ASCII | Logical Records |
| BDAT | Defined Records<br>Logical Records |

D3-32

## FILE ACCESS METHODS

- Sequential access: start at beginning of file and access each successive record in turn
- Random access: access any specified Defined Record

ASCII: Sequential access
BDAT: Random access

D3-33

NOTES

## FILE ACCESS

- I/O Path: a pathway for data to and from a data file
  Specified as an "@Name"
- Declaring an I/O path automatically creates an associated table of information to inform the operating system of I/O path characteristics:
  File type, File size, Pointers

D3-34

NOTES

NOTES

## FILE ACCESS

I/O Path Name:

- Typed variable - 100 bytes
- Can be passed to subprograms and user-functions: pass by reference only

```
CALL Test (@Dvm, Array(*))
```

- Can be allocated in COM:

```
COM @Dvm, Function, Range
```

D3-35

NOTES

## FILE ACCESS

1. Create file

| WRITE | READ |
|---|---|
| 2. Open file (Assign I/O path) | 2. Open file (Assign I/O path) |
| 3. Write data | 3. Read data |
| 4. Close file | 4. Close file |

D3-36

## FILE ACCESS: CREATE

- CREATE establishes a file of the desired type, length, and characteristics on the media

```
CREATE ASCII "Data1",100
CREATE BDAT "Data2",100
```

- More on CREATE and file types, later

D3-37

NOTES

## FILE ACCESS: OPEN

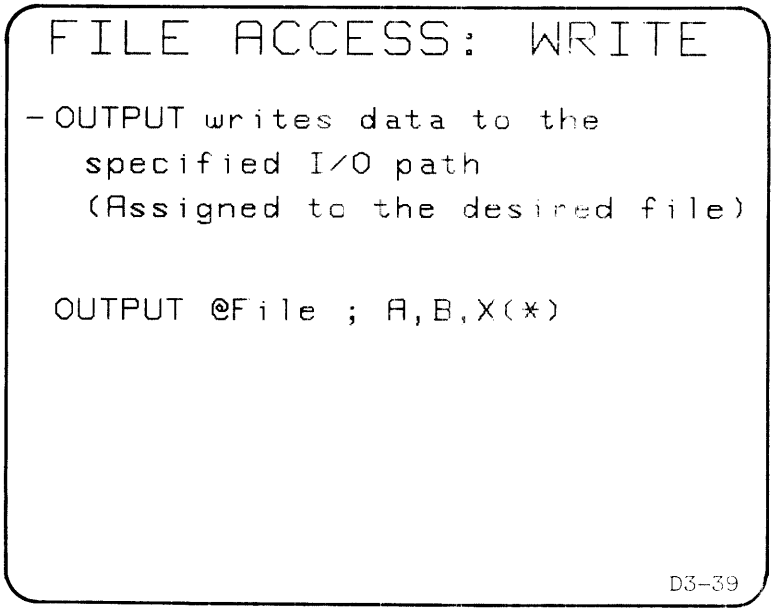
- ASSIGN establishes an I/O path for a specified file
- Multiple I/O paths can be set up for a given file (if necessary)

```
ASSIGN @File TO "DATA1"
```

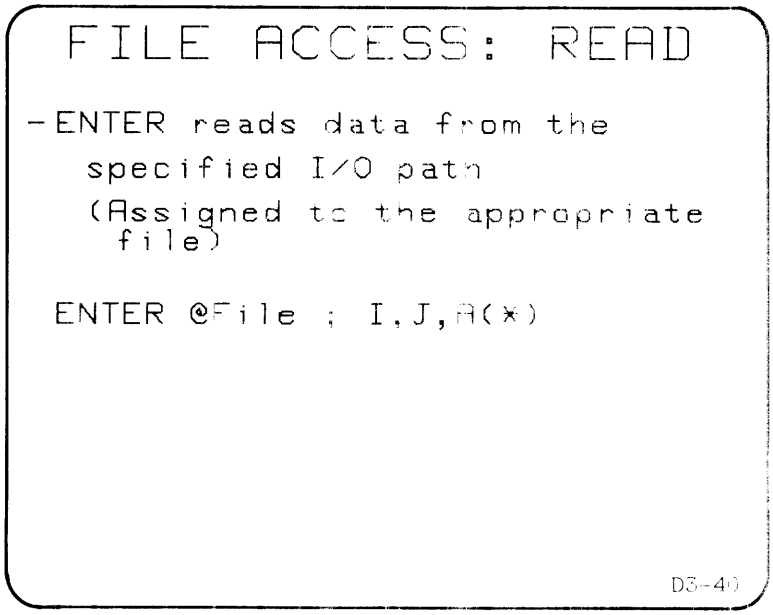
D3-38

NOTES

NOTES


FILE ACCESS: WRITE
- OUTPUT writes data to the specified I/O path (Assigned to the desired file)
OUTPUT @File ; A,B,X(*)
D3-39

NOTES


FILE ACCESS: READ
- ENTER reads data from the specified I/O path (Assigned to the appropriate file)
ENTER @File ; I,J,A(*)
D3-40

## FILE ACCESS: CLOSE

- ASSIGN TO * closes the I/O path for the specified file
- There are other implicit methods of closing the file (Language Reference p.13)

```
ASSIGN @File TO *
```

D3-41

NOTES

## ASCII FILES

- Created by:
  - (RE-)SAVE for programs
  - CREATE ASCII for data
- No defined records
- Sequential access
- End-of-File is a specific character
- Compatible and transportable

D3-42

NOTES

NOTES

## DATA COMPATIBILITY

- LIF = Logical Interchange Format
- LIF is an HP disc format standard that defines the structure of the disc directory and ASCII files
- Provides ASCII data transportability between computers, terminals

D3-43

NOTES

## ASCII DATA FILES

Storage requirements:

1. All data is converted to ASCII characters (string data)
2. Each data item requires
   2 bytes overhead (length)
   +1 byte per character (including all significant digits)
   +1 byte if number of characters is odd

D3-44

# CREATING ASCII DATA FILES

- CREATE ASCII reserves space on the disc for a file of the specified number of physical records (256 bytes each)

```
CREATE ASCII "DATA1", 100
```

D3-45

NOTES

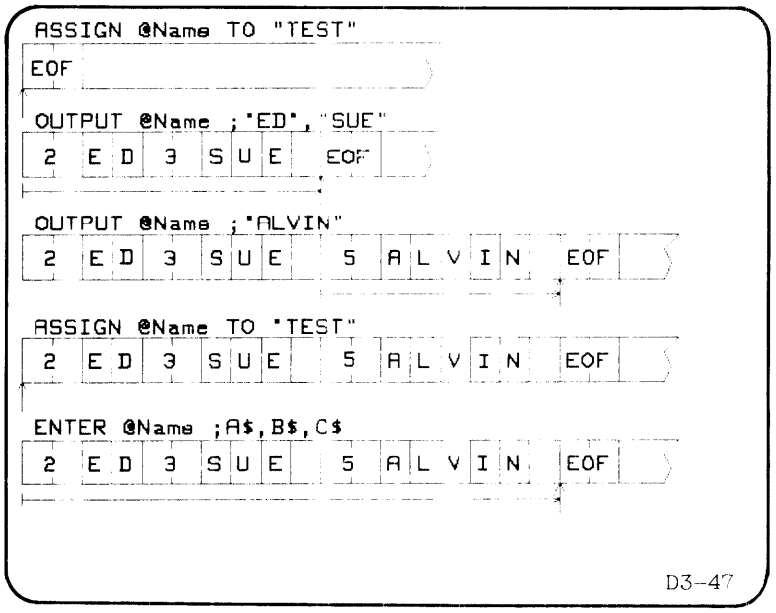
# USING ASCII DATA FILES

```
10 CREATE ASCII "TEST", 10
20 ASSIGN @Name TO "TEST"
30 OUTPUT @Name ; "ED", "SUE"
40 OUTPUT @Name ; "ALVIN"
50 ASSIGN @Name TO "TEST"
60 ENTER @Name ; A$, B$, C$
70 PRINT A$, B$, C$
80 END ! Implicit Close-File
     ED    SUE    ALVIN
```

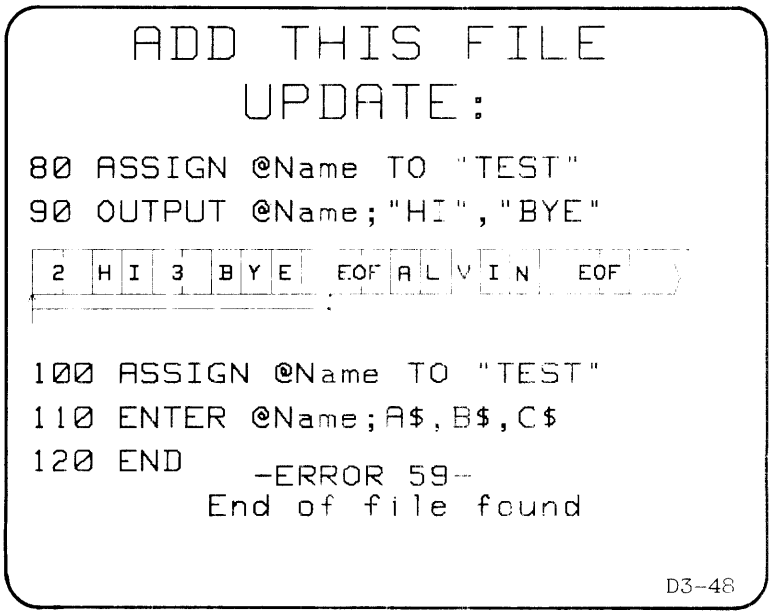
D3-46

NOTES

NOTES


ASSIGN @Name TO "TEST"
EOF
OUTPUT @Name ;"ED","SUE"
2 E D 3 S U E EOF
OUTPUT @Name ;"ALVIN"
2 E D 3 S U E 5 A L V I N EOF
ASSIGN @Name TO "TEST"
2 E D 3 S U E 5 A L V I N EOF
ENTER @Name ;A$,B$,C$
2 E D 3 S U E 5 A L V I N EOF
D3-47

NOTES


ADD THIS FILE
UPDATE:
80 ASSIGN @Name TO "TEST"
90 OUTPUT @Name;"HI","BYE"
2 H I 3 B Y E EOF A L V I N EOF
100 ASSIGN @Name TO "TEST"
110 ENTER @Name;A$,B$,C$
120 END
-ERROR 59-
End of file found
D3-48

## SERIAL UPDATE 1

- One-file update
- Read in, update, write out entire file

```
10 ASSIGN @Name TO "OLD"
20 ENTER @Name ; A$,B$,C$
30 B$ = "New name"
40 ASSIGN @Name TO "OLD"
50 OUTPUT @Name ; A$,B$,C$
60 END
```

D3-49

NOTES

## SERIAL UPDATE 2

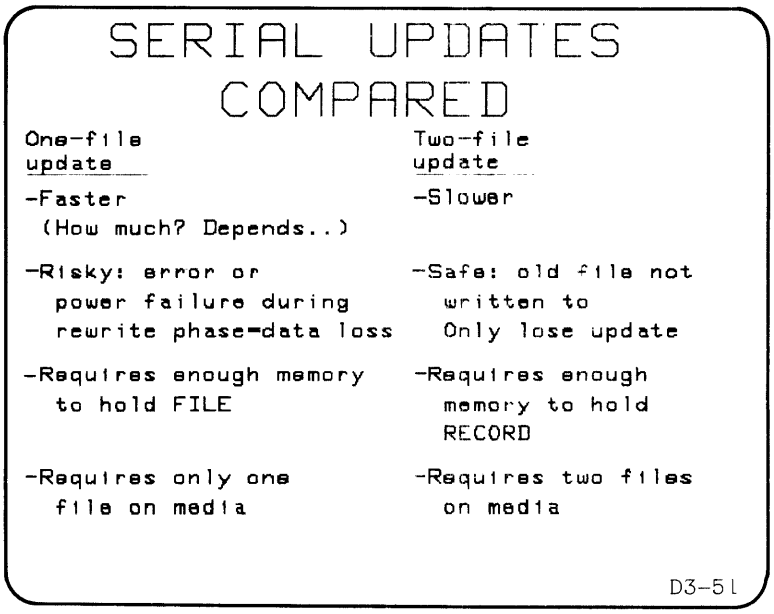
- Two-file update
- Read in logical records one at a time, update when appropriate rewrite each in turn to new file

```
10 ASSIGN @From TO "OLD"
20 ASSIGN @To TO "NEW"
30 FOR I=1 TO 3  ! Count of record
40 ENTER @From;A$
50 IF A$="ED"THEN A$="New name"
60 OUTPUT @To;A$
70 NEXT I
80 END
```

D3-50

NOTES

NOTES




EXERCISE 16

Write a program that updates the data on file "OLD_DATA" , using either update technique that was discussed.

- Data to be changed:
  1. Any occurrence of "Nabraska" to "Nebraska"
  2. Any occurrence of "Misissippi" to "Mississippi"

- Number of data items : 20
- Maximum length of any item : 20 characters

For your own information , have the program print out any data item that is updated, and the number of the data item.

(PRINT A$; TAB(20); I)

## BDAT FILES

- Created by CREATE BDAT
- Defined records
- Sequential or random access
- Formatted or unformatted data
- End-of-file is a pointer
- Not transportable or compatible

D3-52

NOTES

## BDAT FORMATTING

FORMAT ON/FORMAT OFF

- I/O Path attribute
- Defined by ASSIGN
- ASCII data representation

  FORMAT ON
- Internal binary data representation

  FORMAT OFF

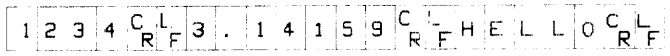
D3-53

NOTES

NOTES

# BDAT FORMATTING

```
ASSIGN @Name ; FORMAT ON
OUTPUT @Name ; Intgr,Real,Str$
```




```
ASSIGN @Name ; FORMAT OFF
OUTPUT @Name ; Intgr,Real,Str$
```


String var. overhead (count)

Internal representations!

D3-54

NOTES

# SELECTING BDAT FORMATTING

- Default: FORMAT OFF
- FORMAT ON: ASCII data representation
- FORMAT OFF: Internal data representation

```
ASSIGN @Name TO "TEST" ; FORMAT ON
ASSIGN @Name TO "FAST" ; FORMAT OFF
```

D3-55

# BDAT DATA FILE

Storage Requirements:

1. FORMAT ON:
   - 1 byte per character
   - + 2 bytes (CR/LF)

2. FORMAT OFF:
   - Real: 8 bytes
   - Integer: 2 bytes
   - String: 1 byte per character
     + 1 byte if odd # of characters
     + 4 bytes overhead

D3-56

NOTES

# CREATING BDAT FILES

- CREATE BDAT reserves space on the disc for a file of the specified number of defined records of optionally specified length (default = 256 bytes)

```
CREATE BDAT "DATA2",100,512
```

D3-57

NOTES

NOTES

## BDAT SERIAL ACCESS

- Use (semantics) and syntax is the same as ASCII file serial access. Data is written to and read from file at the current file data pointer
- No EOF is put on the file at the end of the data. Instead, an EOF pointer is maintained in the I/O path table and on the disc

```
OUTPUT @File;Data$
```

D3-58

NOTES

## DIRECTED (RANDOM) ACCESS

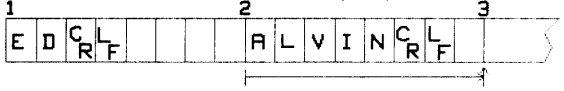
- Directed access postitions the file pointer at the beginning of the specified record for OUTPUT or ENTER
  (Must specify record number!)
- No EOF is put on the file. There is an EOF pointer instead in the I/O path table

```
OUTPUT @File,Rec_num;Data$
```

D3-59

```
10 CREATE BDAT "TEST1",10,8
20 ASSIGN @File TO "TEST1";
   FORMAT ON
30 OUTPUT @File,1;"ED"
```


```
40 OUTPUT @File,2;"ALVIN"
```



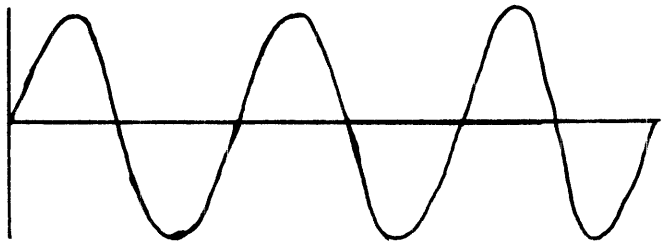

```
50 ENTER @File,2;A$
60 PRINT A$
70 END
```

D3-60

NOTES

# EXERCISE 17

Write a program that puts 100 numeric data items on a BDAT data file. The values for these data items are from a 3-cycle sine wave that your program will calculate. Pictorially, the data will look like this:



1. Create a data file "SIN-DAT", either
    a. 100 records of 8 bytes/record (Directed Access), or
    b. 1 record of 800 bytes (Serial Access)
2. Generate the data
    a. Range of -10 to +10 (10*SIN(X) )
    b. Use a FOR/NEXT LOOP?

        Array(I)=SIN(I*360*3/100)*10
3. Write the data to your file using the appropriate access method.

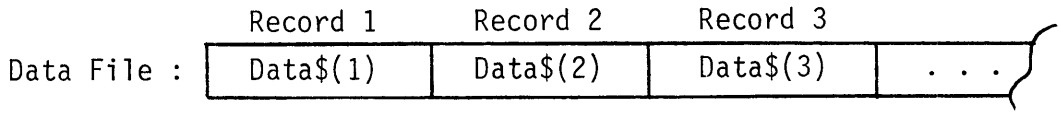
Note: The solution program creates a serial access file.

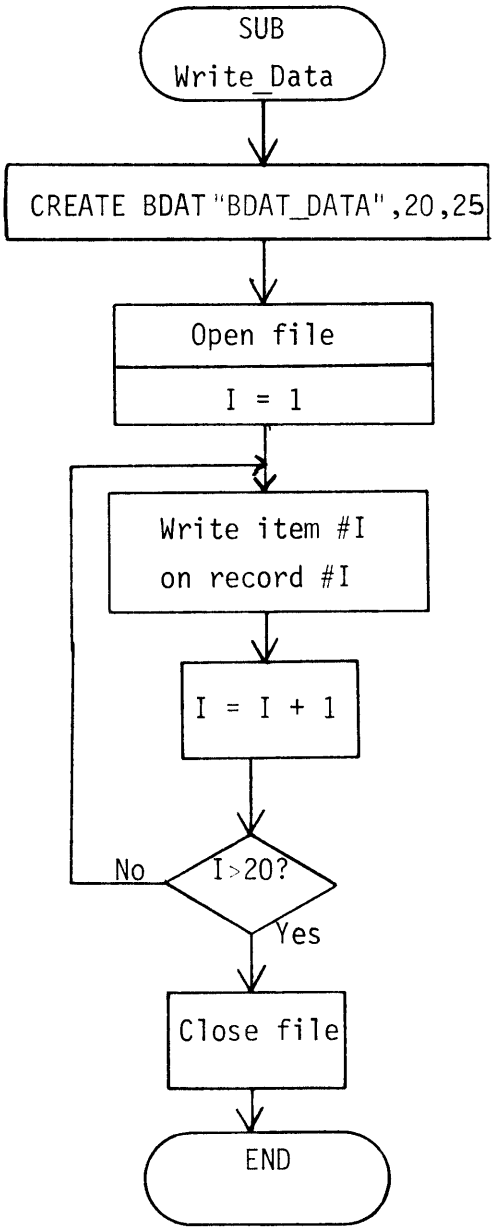
# EXERCISE 18

Write a subprogram that creates a BDAT data file, then outputs 20 string data items (one per record) to the file. The data items will be passed to your subprogram as a string array in the parameter list.

- Name of your subprogram: Write_data
- File name of your subprogram: "WRITEBDAT"
- Create a BDAT file of: 20 records, 25 bytes/record
- Name of BDAT file:: "BDAT_DATA"
- Parameter list: (Data$(*) )

After you write the subprogram and store it on the file, GET "BLD_DATA" and run it. Your subprogram will be loaded and called. When everything has completed, the data file should look like this:



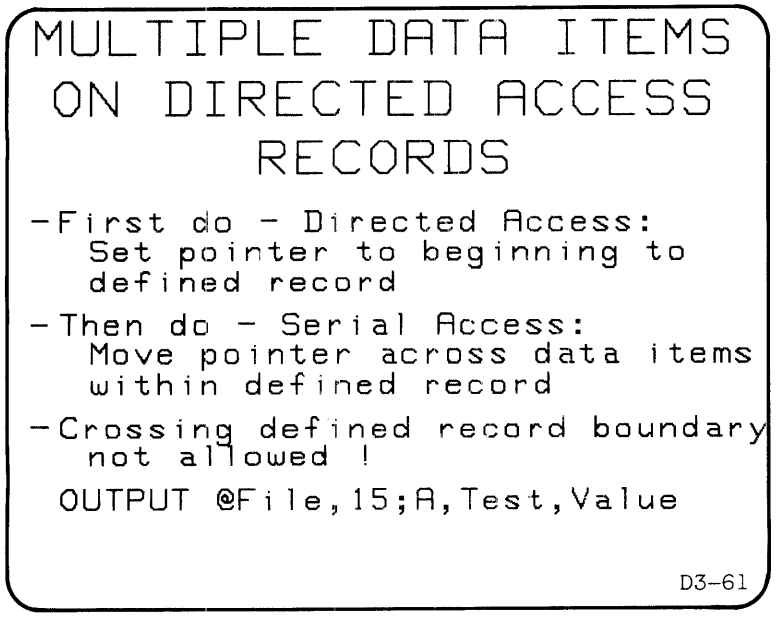

SUB
Write_Data
CREATE BDAT "BDAT_DATA",20,25
Open file
I = 1
Write item #I
on record #I
I = I + 1
No
I>20?
Yes
Close file
END

Write a program to retrieve and print the 20 data items on file "BDAT_DATA" (A FOR-NEXT loop would be OK). Your program should then ask which items need to be modified, input the corrected items, and rewrite them to the file. (Don't rewrite all the items, only the corrected ones). Remember, this file is a directed-access file so your program will have to keep track of the associated record number for each data item.

Some records you might wish to update are:

1
2
9
14
15

(They are misspelled.)




**NOTES**

NOTES

## SERIAL VS DIRECTED ACCESS

| Serial | Directed |
|---|---|
| Conserves space | Can be wastful if logical record lengths vary |
| Hard to update | Easy to update |
| Slow access to last records in file | Constant access times across file |

D3-62

NOTES

## BDAT END-OF-FILE

Updated whenever:

1. Current EOF moves past EOF record (Register 7)
2. END keyword is specified in an OUTPUT statement (OUTPUT @File;END)
3. CONTROL statement sets register 7 or 8

D3-63

## CONTRAST OF ASCII AND BDAT FILES

| | ASCII | BDAT |
|---|---|---|
| Transportable | YES | NO |
| HPL Compatible | YES | NO |
| Serial access | YES | YES |
| Directed access | NO | YES |
| FORMAT ON | YES | YES |
| FORMAT OFF | NO | YES |
| Record length | 256 | Defined |
| Use IMAGE? | NO | YES |

D3-64

NOTES

## TRAPPING END-OF-FILE

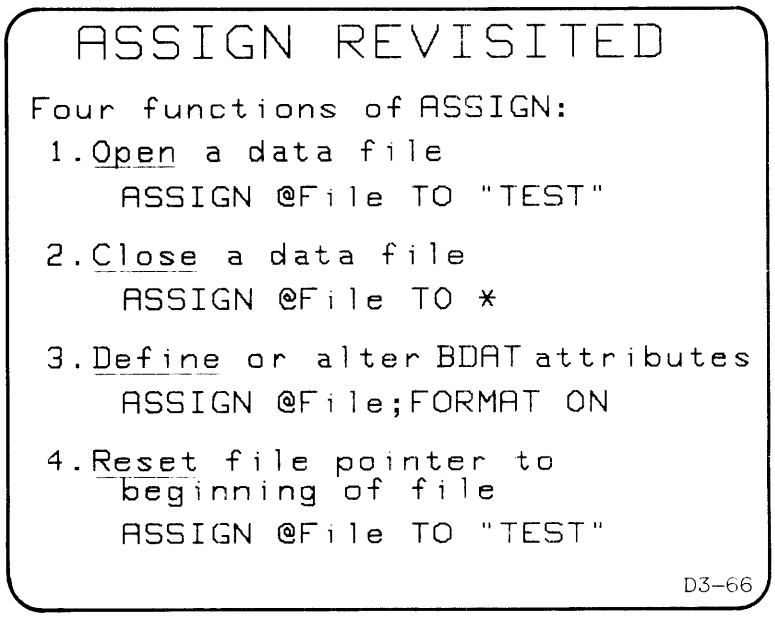
```
10 ASSIGN @Dvm TO "DATA_FILE"
20 ON END @Dvm GOTO Done
30 LOOP
40  ENTER @Dvm;DATA(I)
50  I = I+1
60 END LOOP
70 Done : CALL Plot(Data(*))
80 END
```

D3-65

NOTES

Modify the program "RAND_UPDT" to use ON END and LOOP rather than the FOR-NEXT loop to read in the data. Check your solution by running the program, but you needn't bother updating any data items (enter a 0 record number).

**NOTES**



# FILE CONTROL AND STATUS REGISTERS

- BASIC Language Reference p.283
- STATUS reads I/O path registers

```
STATUS @File,1;Type,Dev,No_rec,Len
```

- CONTROL writes I/O path registers

```
CONTROL @File,7;100
```

D3-67

NOTES

# SPEED CONSIDERATIONS

- BDAT files, FORMAT OFF are significantly faster than either
  - BDAT/FORMAT ON or ASCII files
  - Exception: Strings
- Fastest BDAT write is obtained by using CONTROL to set EOF to last record in file (especially directed access)

D3-68

NOTES

GET "SPEED" and run the program. The program records numeric data on a BDAT file using various combinations of FORMAT ON/OFF, writing the entire array vs one element at a time, and forcing EOF to the end of file.

Analyze the printed results.

Are the relative performances what you expected? What other factors influence this application? (File size, updatability...)

1. Can you explain why Number 1 is faster than Number 3?
2. How is Number 4 so much faster than Number 2?
3. Contrast Number 5 with Number 1. Does the fewer number of bytes transferred explain the speed difference? How about formatting time?
4. Why is Number 6 only slightly faster than number 2, even though only half as many bytes are being transferred?

# I/O PROGRAMMING

Objectives:

1. Relate Unified I/O and Mass Storage Programming
2. Control a minimal instrument system
3. Control 9826 internal peripherals

D4-1

NOTES

# I/O IS:

- Data Input and Output
- Stated relative to the computer
- A data source being transferred to a data destination
- Implemented by Interface Cards

D4-2

NOTES

NOTES

# INTERFACE CARDS

Provide electrical and mechanical compatibility between the computer and external peripherals

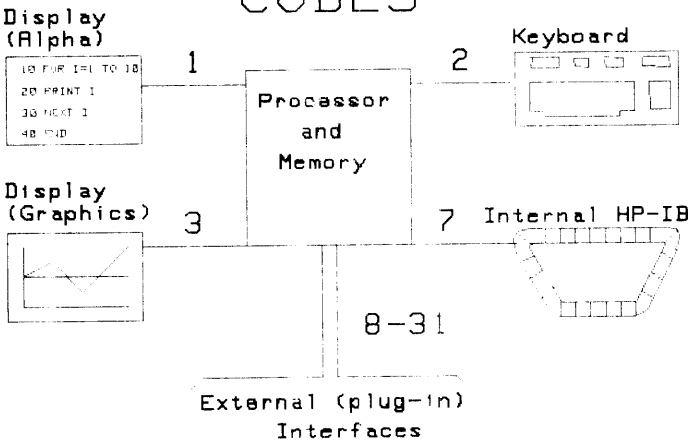
Examples:

- 98622A Parallel Interface
- 98623A BCD Interface
- 98624A HP-IB Interface
- 98626A RS-232C Interface
- 98628A Datacomm Interface

D4-3

NOTES

# INTERFACE SELECT CODES




D4-4

# I/O HIERARCHY

| | |
|---|---|
| ↑High Level | I/O Statement |
| | @Name |
| | Select Code |
| | Formatting |
| | Conversion |
| | Firmware Drivers |
| | Registers |
| Low Level | Hardware |

D4-5

NOTES

# VERY BASIC I/O

```
10 LET Volts = 2.51
       Destination  Source

20 INPUT A , B , C
          Destinations
       Operator=Source

30 PRINT A , B , C
          Source
       Printer=Destination

40 DISP A , B , C
          Source
       CRT=Destination
```

D4-6

NOTES

NOTES

# INPUT VS LINPUT

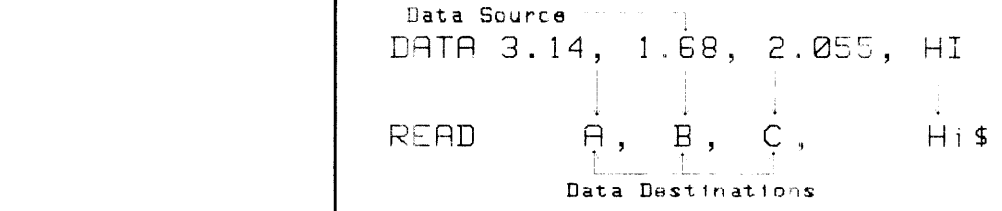
The Problem of Commas & Quotes

```
INPUT "Address",Addr$
  -operator enters-
  Houston,Texas [ENTER]
  Addr$ contains Houston

LINPUT "Address",Addr$
  -same entry-
  Addr$ contains Houston,Texas
```

D4-7

NOTES

# MORE BASIC I/O




```
RESTORE Dvm_data
  Pointer control (like ASSIGN)
```

D4-8

# USER-DIRECTED I/O

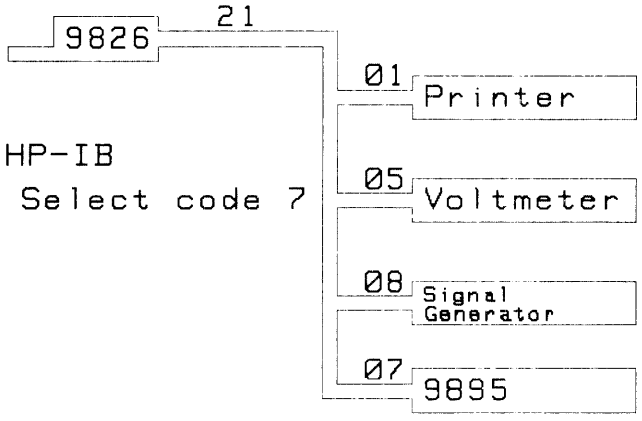
- Explicit designation of data source/destination
- Device Selector: interface select code (and device address)

```
PRINTER IS 701
           ↑↑
           ||Device Address
           |Select Code
```

D4-9

NOTES

# DEVICE ADDRESS (HP-IB)




D4-10

NOTES

NOTES

## USER-DIRECTED I/O

PRINTER IS device selector
  Directs all PRINT data to the specified device
  Default=CRT=1

PRINTALL IS device selector
  Directs PRT ALL messages to the specified device
  Default=701
  (Governed by PRT ALL is ON/OFF)

D4-11

NOTES

## USER-DIRECTED I/O

LIST # device selector
  Lists all or part of the program to the specified device

DUMP DEVICE IS device selector
  Directs contents of alpha or graphics dump to specified device

  (DUMP ALPHA, DUMP GRAPHICS)

D4-12

# USER-DIRECTED I/O

ENTER and OUTPUT
I/O Workhorses

OUTPUT Device selector ; A,B,C

Destination Source

ENTER Device selector ; A,B,C

Source Destination

D4-13

NOTES

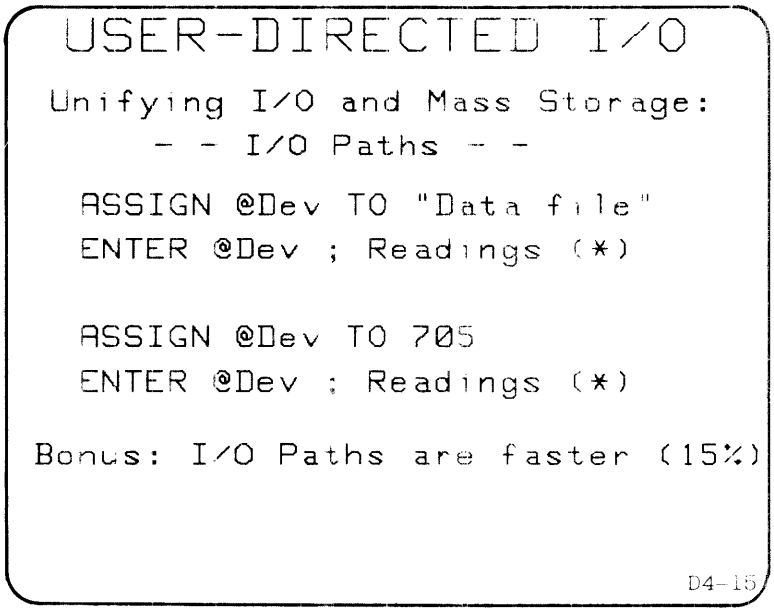
# EXAMPLES

```
ENTER 2;Long_string$
OUTPUT 1;Operator_prompt$
OUTPUT 2;"BEEP K X"
OUTPUT 1;1.8,2.8,3.8
OUTPUT 701;"Printer test"
```

D4-14

NOTES

NOTES




EXERCISE 22

Write a program that takes 100 "instrument" readings and outputs those readings to the CRT, then to the printer. Your program can take the "instrument" readings off of file "SINDAT" for simulation purposes. Use only ASSIGN, OUTPUT, and ENTER to read and write the data items.

Don't forget to use the appropriate file access statement for the structure of the records on the file (sequential is 1 record of 800 bytes, directed is 100 records of 8 bytes).

Save your file as "Read_sin" (lowercase to avoid clobbering "READ_SIN").

## DATA FORMATTING

Two levels:

Low level: Translation of internal binary data representation into characters

000000101100100 ——→ 356

Internal Formatting External

High level: Arrangement of data into optimal human or machine-readable formats

160500 ——→ +16.05 E+04

Unformatted Formatting Formatted

D4-16

NOTES

## DATA FORMATTING

Low level formatting: I/O path attribute

```
FORMAT ON        FORMAT OFF
80 ASSIGN @Pipe TO 705;FORMAT OFF
```

High level formatting:

User-defined "images"

This topic is addressed in the following slides

D4-17

NOTES

NOTES

## DATA FORMATTING

FORMAT ON/FORMAT OFF

- I/O Path Attribute
- Defined by ASSIGN
- ASCII data representation

FORMAT ON

- Internal binary data representation

FORMAT OFF

D4-18

NOTES

## DATA FORMATTING

In the beginning.....
There were punched cards

- Card image: what the data looked like on the punched card (Format)

- Record: one card image
"Physical" unit of data

D4-19

## DATA FORMATTING

- Record Input: ENTER
  Reads data items until encountering a record delimiter

- Record Output: OUTPUT
  Sends data items then sends a record delimiter

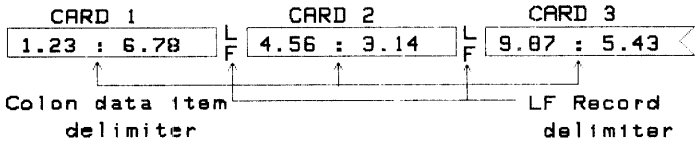
D4-20

NOTES

## DATA FORMATTING

Data item delimiter: a control or special character separating individual data items

Record delimiter: a control character separating records (card images)




D4-21

NOTES

NOTES

## DATA FORMATTING

Data Delimiters

| | Numeric | String | Record |
|---|---|---|---|
| OUTPUT (3) | , | CR/LF | CR/LF[1] |
| OUTPUT (4) | Pos. ' '<br>Neg. '-' | nothing | CR/LF[1] |
| ENTER | non-numeric | LF OR CR/LF | LF or CR/LF[1] |
| PRINT (3) | Blank Pads[2] | Blank Pads[2] | CR/LF[1] |
| PRINT (4) | 1 blank | 1 blank | CR/LF[1] |

1. EOL (end-of-line) sequence
2. Blank fill to end of 10 character field
3. Commas for data list separators
4. Semicolons for data list separators

D4-22

NOTES

## DATA FORMATTING

Try each of these:

```
OUTPUT 1; 123,456
PRINT 123,456
OUTPUT 1; 123;456
PRINT 123;456
OUTPUT 1;-123,-456
OUTPUT 1;-123;-456
```

D4-23

## DATA FORMATTING

Default Numeric Formats:

- Numbers in the range of
  1E-4 ≤ |Number| ≤ 1E6, sent as rounded 12 digit floating point
- All others - scientific notation

```
OUTPUT 1;123456.7891,1234567.891
```

D4-24

NOTES

## DATA FORMATTING

Custom Data Formats

- IMAGE explicity specifies the data format used for OUTPUT, ENTER, PRINT, DISP, and LABEL data items

```
GET "IMG_EXPLS"
LIST the program
RUN the program
```

D4-25

NOTES

NOTES

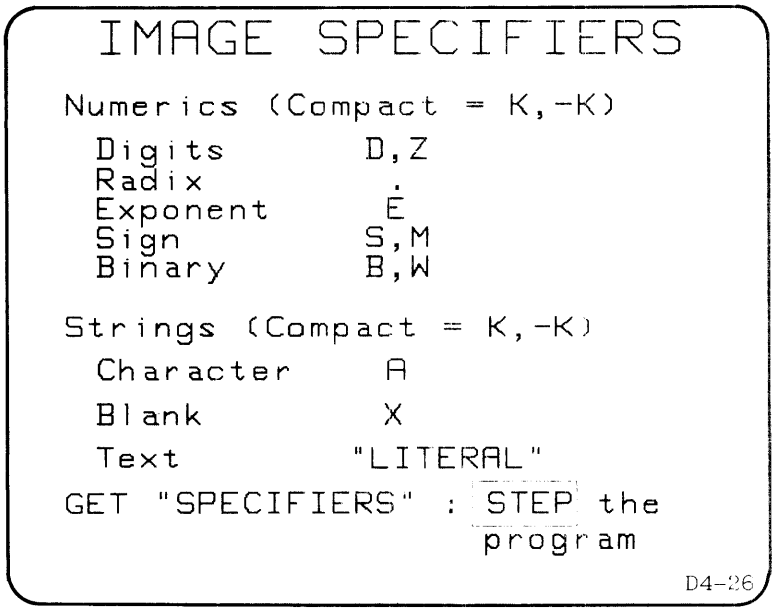


IMAGE SPECIFIERS
Numerics (Compact = K,-K)
Digits D,Z
Radix .
Exponent E
Sign S,M
Binary B,W
Strings (Compact = K,-K)
Character A
Blank X
Text "LITERAL"
GET "SPECIFIERS" : STEP the program
D4-26

NOTES

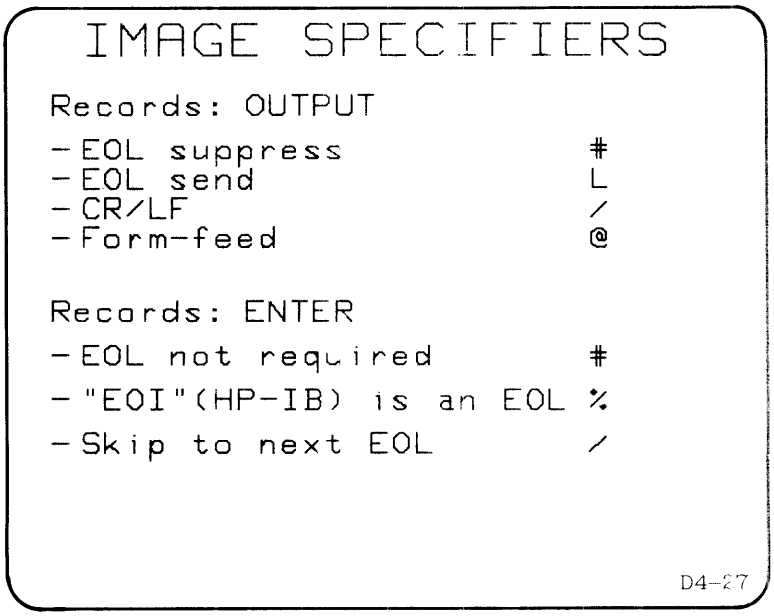
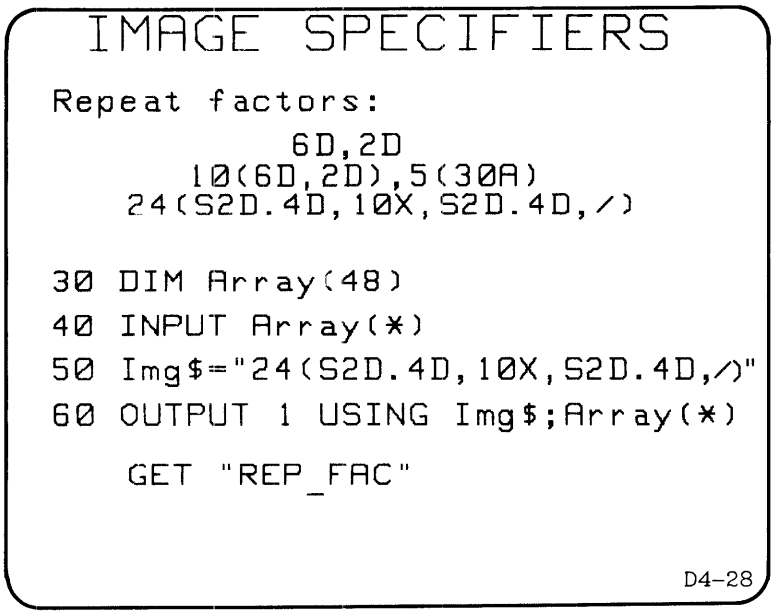


IMAGE SPECIFIERS
Records: OUTPUT
-EOL suppress #
-EOL send L
-CR/LF /
-Form-feed @
Records: ENTER
-EOL not required #
-"EOI"(HP-IB) is an EOL %
-Skip to next EOL /
D4-27




NOTES

EXERCISE 23

Modify program "READ_SIN" (or your program "Read_sin") to write the data items as a 5 column printout to the CRT, and a 10 column printout to the printer. Some things to be aware of:

1. You won't need an exponent.
2. Allow two digits to the left of the decimal point.
3. Three digits to the right of the decimal point are plenty (our DVM has limited accuracy).
4. Add some extra spaces between numbers on a line.
5. Remember to specify an EOL sequence with each line.

Picture one line of printout and the data images on it, then try to construct an IMAGE that will produce it.

For example:

-1.345 2.843 8.152 6.001 .055

Then add a repeat factor that will produce the desired number of lines of that format.

NOTES

## INTERNAL I/O

OUTPUT to the keyboard?!

1. Control the computer by pressing keys, "typing".

2. Allow operator editing of string without re-typing:

```
OUTPUT 2 USING "#,K";A$
ENTER 2 ; A$
```

D4-29

NOTES

## INTERNAL I/O

ENTER from the CRT?!

Read in system messages, CATalogs, etc

Position the cursor:

```
PRINT TABXY (1,1);
```

Read the screen:

```
ENTER 1 ; Screen$
```

D4-30

## INTERNAL I/O

ENTER, OUTPUT to the disc
-Mass Storage-

The Real-Time Clock
SET TIME Seconds
SET TIMEDATE Seconds
Seconds= TIMEDATE

D4-31

NOTES

## INTERNAL I/O

- The keyboard
  Execute a service routine
  whenever a key is pressed

ON KBD, 8 CALL Key_svc
↑
priority

- KBD$ function returns keys
  pressed and clears the buffer

Keys$=Keys$&KBD$

D4-32

NOTES

Write a program that uses ON KBD to trap keypresses and display the character and its numeric value for the pressed key. Try adding features to disable the STOP and PAUSE key functions. Look up in the Language Reference manual the function of the following statement:

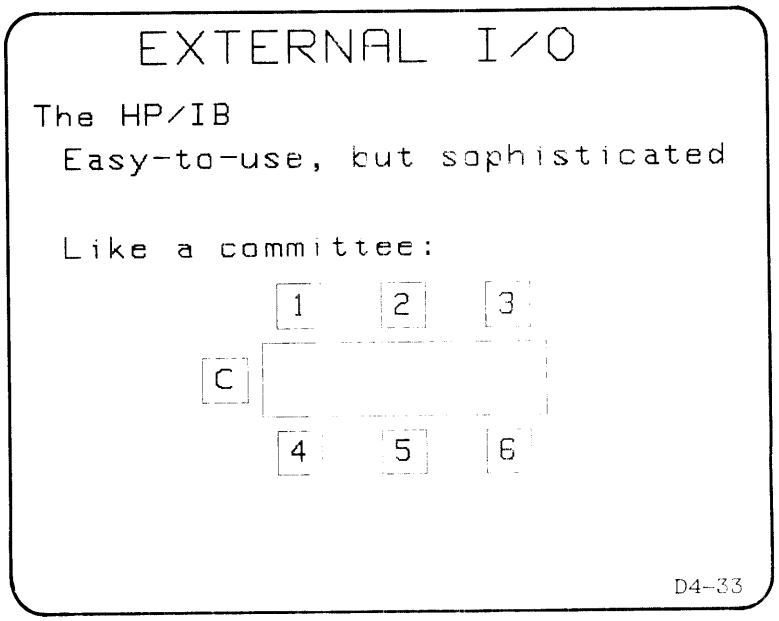
```
CONTROL 2 , 7 ; 2
```

(Hint - look at the "Keyboard Status and Control Registers" table in the back of the manual.)

What effect would this statement have on the operator and the program? How might an operator stop the program if necessary? Perhaps a sequence of Control-Shift-Key(X) to signal an extraordinary condition to the program?

Use CTRL - SHIFT - STEP - to stop "ON_KBD" after you run it.

**NOTES**



## HP-IB

- Committee chairman
  HP-IB=System controller
- Speaking member
  HP-IB=Active Talker
- Listening members(s)
  HP-IB=Active Listener(s)

D4-34

NOTES

## HP-IB

- Acting chairman
  HP-IB=Active Controller
- To address the committee, a member must be "given the floor" by acting chairman
  HP-IB=Addressed-to-talk
  Done by Active Controller

D4-35

NOTES

NOTES

## HP-IB

- The acting chairman may direct members to take notes
  HP-IB=Addressed-to-listen
  Done by Active Controller
- Slow members may raise their hand to slow down the speaker
  HP-IB="Handshake"

D4-36

NOTES

## HP-IB

- The Gavel (restoring order)
  HP-IB=Interface Clear (IFC)
  Only done by System Controller
- Summary:
  A committee is organized, orderly, and has protocols
  HP-IB is well-defined, predictable, and has protocols

D4-37

# HP-IB "BEHIND THE SCENES"

```
OUTPUT 701 ; "HI"
Commands:ATN True
  1.TALK ADDRESS 21     (MTA)
  2.UNLISTEN            (UNL)
  3.LISTEN ADDRESS 01   (LAG)
Data: ATN False
  4.DATA  "H"
  5.DATA  "I"
  6.DATA  CR
  7.DATA  LF
```

D4-38

NOTES

# HP-IB "BEHIND THE SCENES"

```
ENTER 702 ; A$
Commands: ATN True
  1.TALK ADDRESS 02    (TAG)
  2.UNLISTEN           (UNL)
  3.LISTEN ADDRESS 21  (MLA)

Data: ATN False
  4.DATA  "B"
  5.DATA  "Y"
  6.DATA  LF
```

D4-39

NOTES

NOTES

# HP-IB "BEHIND THE SCENES"

"Order in the court!"

"Order on the bus!"

ABORT 7 (IFC)
-The Gavel!-

CLEAR 7 (DCL)
-The Shoe?-

D4-40

NOTES

# HP-IB

Setting up instruments
-Commands as data-

| Instrument Functions | ASCII Commands |
|---|---|
| Range 1,2,3 | R1, R2, R3 |
| Function 1,2 | F1, F2 |
| Trigger | T1 |

D4-41

## HP-IB

Sending instrument commands

```
OUTPUT 705 ; "F1R2T1"
```

Reading instrument data

```
ENTER 705 ; Volts
```

D4-42

NOTES

## HP-IB

```
10 ASSIGN @Todvm TO 705
20 ASSIGN @Fromdvm TO 705
30 OUTPUT @Todvm;"F1D40R7T1"
40 FOR I=0 TO 9
50 ENTER @Fromdvm;Volts(I)
60 NEXT I
70 PRINT Volts(*)
80 END
```

D4-43

NOTES

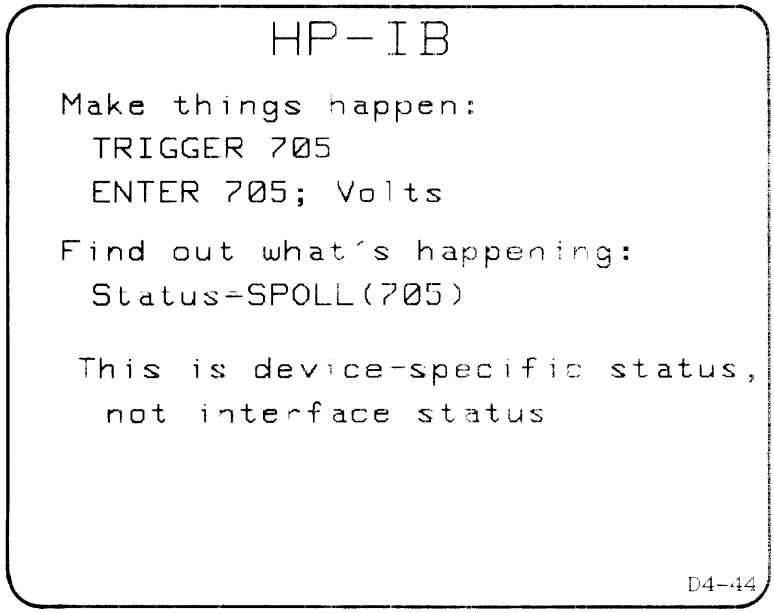
## EXERCISE 25 (Optional)

Modify your program "Read_sin" or the solution program "READ_SIN" to work with a voltmeter. Take the readings from the device (instead of the file) and write them to the file, CRT, and printer.

1. You will need to insert some program lines similar to those on the previous slide to take the readings from the instrument.
2. You may need to modify the instrument commands depending on the particular set-up available to your class.
3. Output the 100 reading array back to the file "SINDAT". This should simply be a matter of changing ENTER to OUTPUT.

Note: If an instrument is not available, assign the "Todvm" I/O path to a temporary file (one record of 256 bytes is sufficient) and the "Fromdvm" I/C path to "SINDAT".

**NOTES**



## HP-IB

Put devices under remote control:

```
REMOTE 7
```

Prevent operator intervention:

```
LOCAL LOCKOUT 7
```

Return devices to local control:

```
LOCAL 7
```

D4-45

NOTES

## HP-IB INTERFACE

Status: Lang. Ref. pgs.287-290

```
Type: STATUS 7,0;Type
Address: STATUS 7,3;Ad
  Ad=BINAND(Ad,31)
State: STATUS 7,6;State
Lines: STATUS 7,7;Lines
```

D4-46

NOTES

NOTES

## HP-IB INTERFACE

Control

Reset: CONTROL 7,0;1

Set Address: CONTROL 7,3;Adr

Interrupt

SRQ: ENABLE INTR 7; 2

Anything else-refer to Lang. Ref.

D4-47

NOTES

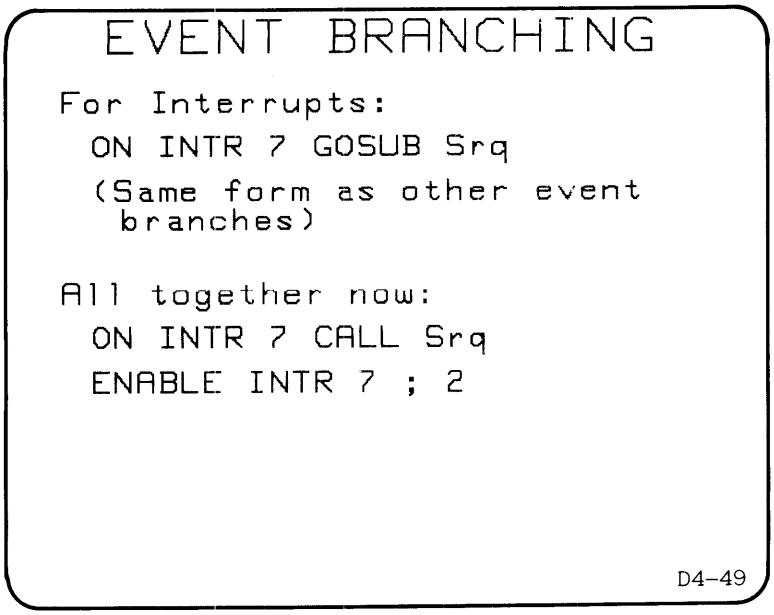
## EVENT BRANCHING -REVISITED-

Myriad of possible external causes. One "enable" statement for all

The "mask" specifies which interrupt causes are desired

ENABLE INTR 7 ; Mask

D4-48




NOTES

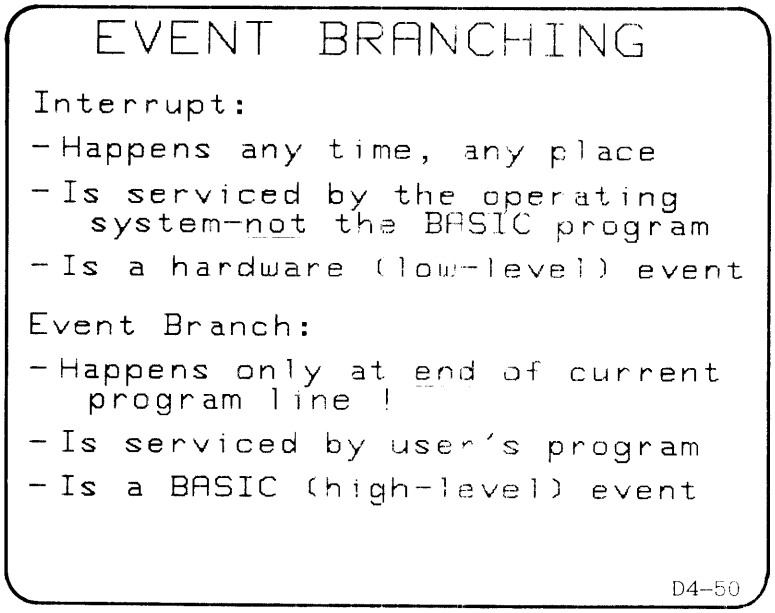
EXERCISE 26

Write a program that enables and services a service request. The program should contain a main loop that displays a counter, and a service routine. In your service routine, obtain the device's status and print the result. (Use SPOLL).

1. To generate a service request, press the SRQ button on the HP-IB box.
2. To deal with SPOLL using the HP-IB box, you need to accept several command bytes, then when the ATN line goes false, send one response byte, then you must accept two more command bytes. At this point, SPOLL is complete.

Your program must re-execute ENABLE INTR if it is to continue servicing SRQ interrupts.

NOTES




EXERCISE 27

Modify your service-request program so that the main program is executing a WAIT 5 in the loop. Change the SPOLL function in your service routine to a PRINT "HERE" statement. Watch how long it takes your program to service the SRQ.

Try pressing SRQ twice in rapid succession. Does the service routine get executed twice? What does this mean?

What effect would this have on your programming efforts if you required a short response time to interrupts? What can happen to interrupts coming in at too high a frequency?

# EVENT BRANCHING

Bailing Out

If an OUTPUT or ENTER gets "hung" by a device, the program dies

```
ON TIMEOUT 7,2 CALL Bail_out
```

Sets a two second time limit to an otherwise infinite wait for ENTER or OUTPUT on select code 7 to complete

D4-51

NOTES

# EVENT BRANCHING REVIEW

| Event | Priority | ENABLE DISABLE? | Local Interrupts Logged? |
|---|---|---|---|
| ON END | 16 | NO | NO |
| ON ERROR | 16 | NO | NO |
| ON TIMEOUT | 16 | NO | NO |
| ON INTR | 1 - 15 | YES[1] | YES |
| ON KEY | 1 - 15 | YES | YES |
| ON KNOB | 1 - 15 | YES | YES |
| ON KBD | 1 - 15 | YES | YES |

1. DISABLE only. Use ENABLE INTR.

D4-52

NOTES

NOTES

## EVENT BRANCHING REVIEW

| Branch | Scope | System Priority Becomes: |
|---|---|---|
| GOTO | Local | No change |
| GOSUB | Local | Specified priority of ON<event> |
| CALL | Global | Specified priority of ON<event> |
| RECOVER | Global | Priority of context[1] that defined ON<event> |

[1] Dynamic priority

D4-53

NOTES

## EVENT BRANCHING REVIEW

Priority

- Only event branches of specified priority higher than current system priority are taken
- Default system priority=0
- Default event priority=1

D4-54

# EXERCISE 28

Write a program with service routines for errors, special function keys, the knob, and external (SRQ) interrupts. Define priorities and branch types to ensure that SRQ is always serviced immediately, regardless of current program context.

1. K1, K2, and K3 invoke subroutines of priority levels 1, 2, and 3 respectively, and display operating priority.
2. K 19 is an "abort" key that returns program execution to the main program loop - inspite of current context and operating conditions.
3. The knob causes all but SRQ service to be disabled for the duration of knob service then reenabled at the end of the routine.
4. An error has the same effect as pressing K 19.

In each service routine, display the current priority, the routine's name and a counter that is incremented to 500 before exiting the routine.

Try various combinations of knob rotation, keypresses, and SRQ to determine if the logic of your program is correct. Be sure that SRQ is always serviced, regardless of what the computer is doing.

After your program is running correctly, add a subprogram that displays the current nesting level ( how many times it has called itself), increments the nesting level counter, allocates a 100 element array, waits for .05 seconds, then calls itself. Add a call to this subprogram in the main program and run your program again. Which interrupts will get serviced? Which won't? How many levels of nesting do you get before you get an error? (How can you tell you got an error?) What happens if you press K19 before then? Try pressing K1, K2, K3, then K19. Are you able to explain the sequence of events that occurred?

# GRAPHICS PROGRAMMING

Objectives:

- Display data graphically.
  Curves, Bars, Pies
- Create and manipulate objects.
  Draw, Move, Rotate

D5-1

NOTES

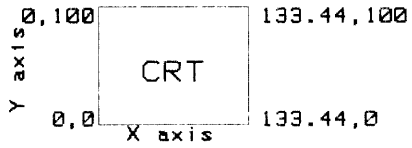
# INTRODUCTION

Graphics = Plotting = Drawing

The pencil: Electronic or Mechanical

The paper: The CRT or Plotter bed

The ruler: Graphic Display Units (GDU scale versus inches or millimeters)




D5-2

NOTES

NOTES

# INTRODUCTION

10 GRAPHICS ON = GRAPHICS

20 ALPHA OFF = GRAPHICS (2nd press)

30 GINIT = Initialize parameters.
(Refer to GINIT in ref. manual)

40 GCLEAR = Clear graphics.

50 MOVE 0,0 = Lift "pen", move it to lower left of CRT (X=0, Y=0).

60 DRAW 10,10 = Put pen down, draw a line from current (X,Y) to (X=10, Y=10).

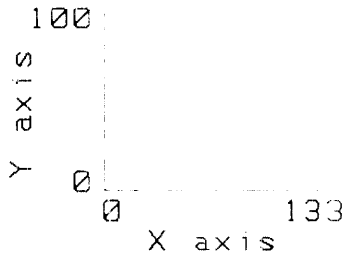
D5-3

NOTES

# DRAW A LINE

DRAW 50,50

X Y



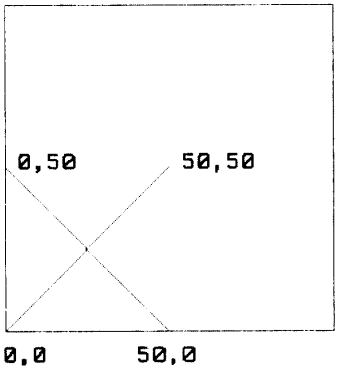

D5-4

# DRAW AN X

Visualize the action:

Draw - Move - Draw

```
10 GINIT
20 GRAPHICS ON
30 DRAW 50,50
40 MOVE 0,50
50 DRAW 50,0
60 END
```




D5-5

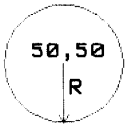
NOTES

# DRAW A CIRCLE

Visualize the action:

Move - Draw - Draw - Draw...

```
 5 DEG
10 GINIT
20 GRAPHICS ON
30 X=50   ! Center of
40 Y=50   ! Circle
60 R=40   ! Radius of circle
70 MOVE X,Y
80 FOR I=0 TO 360
90 DRAW X+R*COS(I),Y+R*SIN(I)
100 NEXT I
110 END
```




D5-6

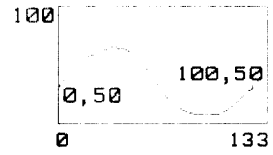
NOTES

NOTES

## DRAW A SINE WAVE

This can be used to plot most any function of X. Y=f(X) = SIN (X)

```
 5 DEG
10 GINIT
20 GRAPHICS ON
30 MOVE 0,50
40 FOR X=0 TO 360
50 DRAW X,SIN(X)
60 NEXT X
70 END
```




Try scaling...

```
50 DRAW X/360*100,SIN(X)*40+50
```

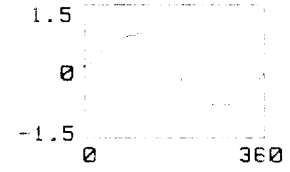
D5-7

NOTES

## GRAPHICS SCALING

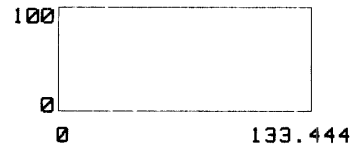
You can scale the display to the range of the data you wish to represent.

```
 5 DEG
10 GINIT
20 GRAPHICS ON
30 WINDOW 0,360,-1.5,1.5
40 MOVE 0,0!
50 FOR X=0 TO 360
60 DRAW X,SIN(X)
70 NEXT X
80 END
```



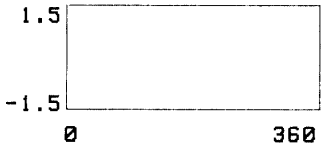

D5-8

## DEFAULT SCALING




GDU's
Graphic Display
Unit

## SIN(X) SCALING




UDU's
User-Defined
Units

D5-9

NOTES

## DEFAULT SCALING

(What, and why is it?)

1. It is arbitrary.
2. It is independent of actual plotter size.
3. Plotter short side = 100 units.
4. One X unit=One Y unit=One GDU
5. Aspect Ratio (X len/Y len) of plot area = RATIO function.

D5-10

NOTES

NOTES

## DEFAULT SCALING
(How can I use it?)

1. Scale your data appropriately.
2. Ignore it. (Scale display to match your data range.)
3. Use it with VIEWPORT to allow the same routine & same range of data draw on different areas of plotter.

D5-11

NOTES

## LOCATING & SIZING THE PLOTTING AREA

- Use VIEWPORT to specify where and how big the plotting area is.
- VIEWPORT uses GDU's to specify points on the plotting device: Xleft, Xright, Ylower, Yupper

VIEWPORT 40,65,40,60

VIEWPORT Left, Right, Bottom, Top

D5-12

# MULTIPLE PLOTS

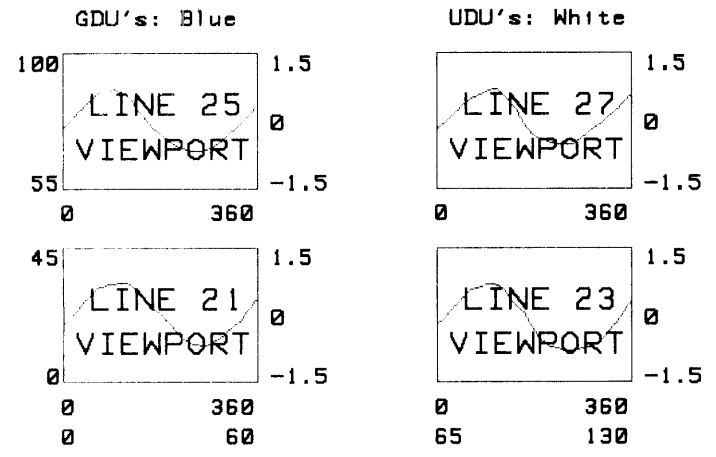
Insert these lines:

```
21 VIEWPORT 0,60,0,45  !Lower Left
22 GOSUB 30
23 VIEWPORT 65,130,0,45  !Lower Right
24 GOSUB 30
25 VIEWPORT 0,60,55,100  !Upper Left
26 GOSUB 30
27 VIEWPORT 65,130,55,100  !Upper Right
28 GOSUB 30
29 STOP

71 RETURN
```

D5-13

NOTES

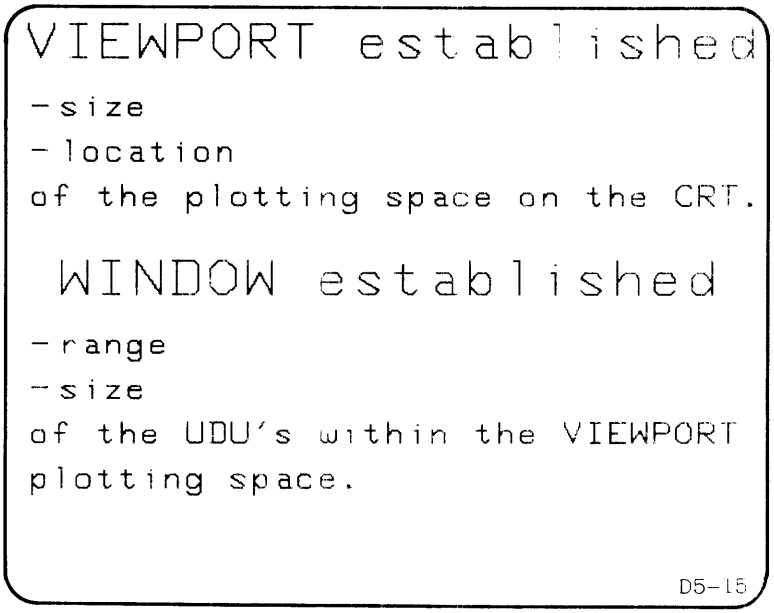
# WHAT YOU JUST DID:




D5-14

NOTES

NOTES




EXERCISE 29

Modify the program "CIRCLE" so that the circle is drawn inside a window of 100 X 100.

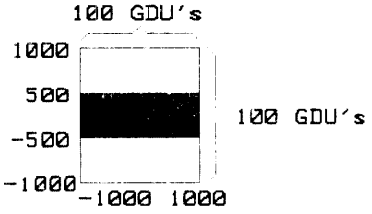
What happens to the circle? Can you explain why? Be sure you understand what happened before continuing on to the following topics.

# YET ANOTHER SCALE

Isotropic UDU's!

X unit length = Y unit length
(Like GDU's, only different)

```
VIEWPORT 0,100,0,100
SHOW -1000,1000,-500,500
```




D5-16

NOTES

VIEWPORT establishes the plotting area. (in GDU's)

Within the plotting area:

WINDOW defines the range of UDU's on X and Y axes.
(X units ≠ Y units)

SHOW determines size of UDU so that X unit = Y unit.

D5-17

NOTES

NOTES

WINDOW scales the plotting area in units appropriate to plot data. (seconds vs. volts)

SHOW scales the plotting area in units appropriate to draw objects. (inches, feet, meters)

D5-18

NOTES

## BACK TO BASICS

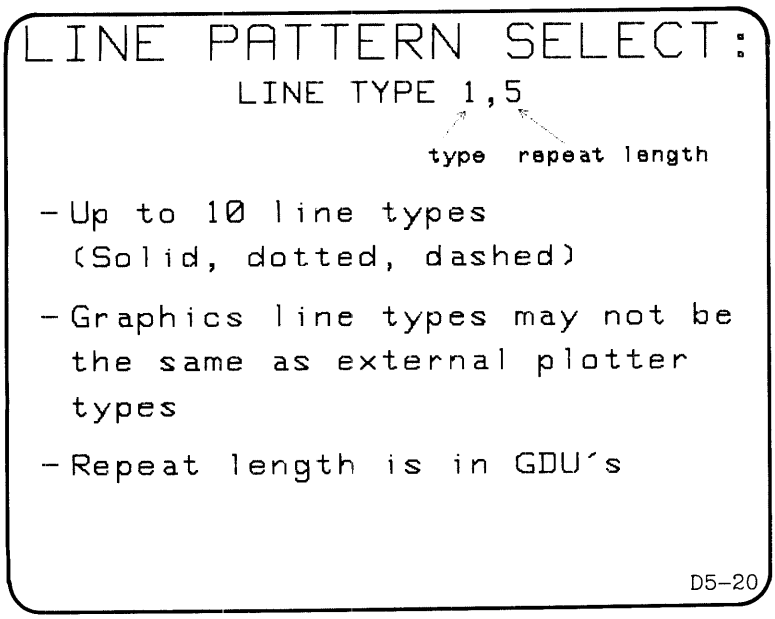
Pen control:

- MOVE vs. DRAW
- PENUP (external plotter)
- PEN

Graphics Pens

- Draw line : $\geq 1$
- Erase line : $\leq -1$
- Complement line : 0
- External : selects pen #

D5-19




NOTES

EXERCISE 30

Modify the program "FOUR_SIN" so that each curve is drawn with a different line type. Choose line-types appropriate for drawing continuous curves.

Be aware that line type affects the frame drawn around the plotting area (via the FRAME statement). How might the program be modified so that the frame is always drawn with line type 1?

NOTES

## SOME MORE BASICS

For interpolation and extrapolation you use graph paper.

Graph Characteristics:
- Grid pattern
- X axis
- Y axis
- Tick marks (on the axes)
- Graph and axis labels

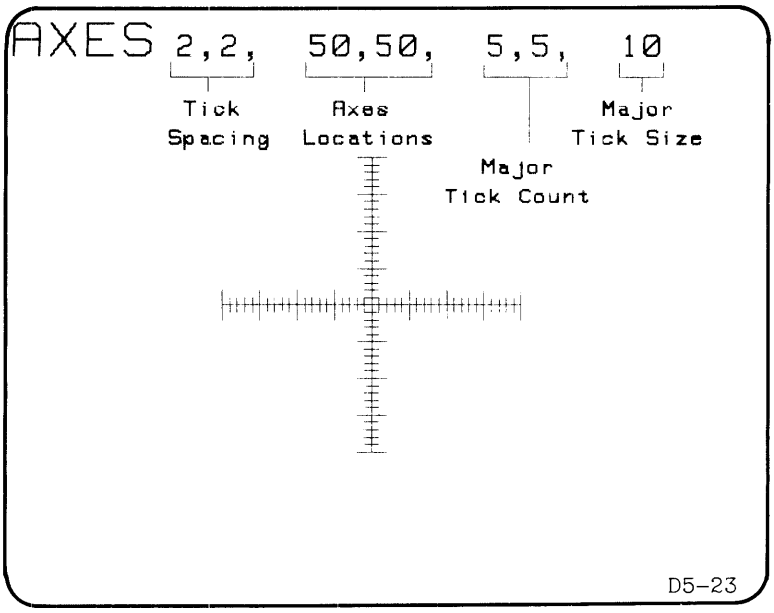
D5-21

NOTES

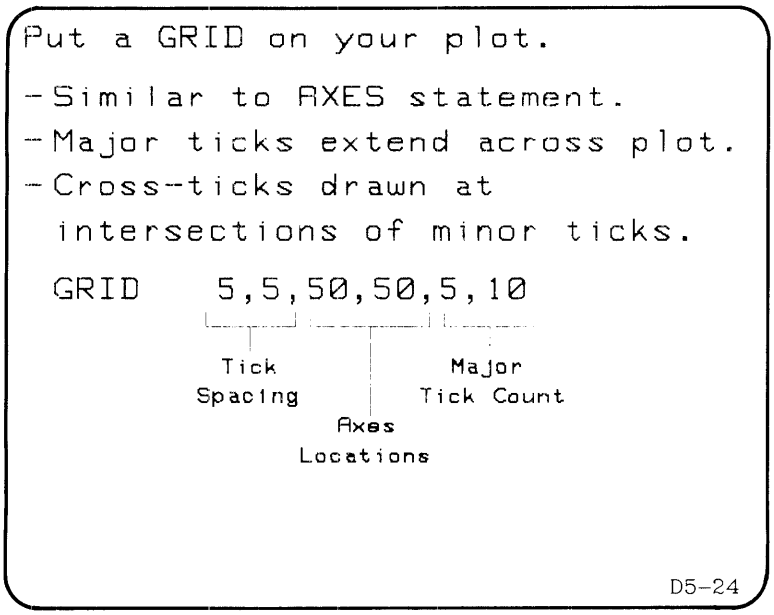
FRAME your plot.
- Uses current line type.
- Frames defined plotting area. (set by VIEWPORT)

Put AXES on your plot.
Specifies X and Y axes.
- Location
- Tick-mark spacing
- Large tick interval

D5-22

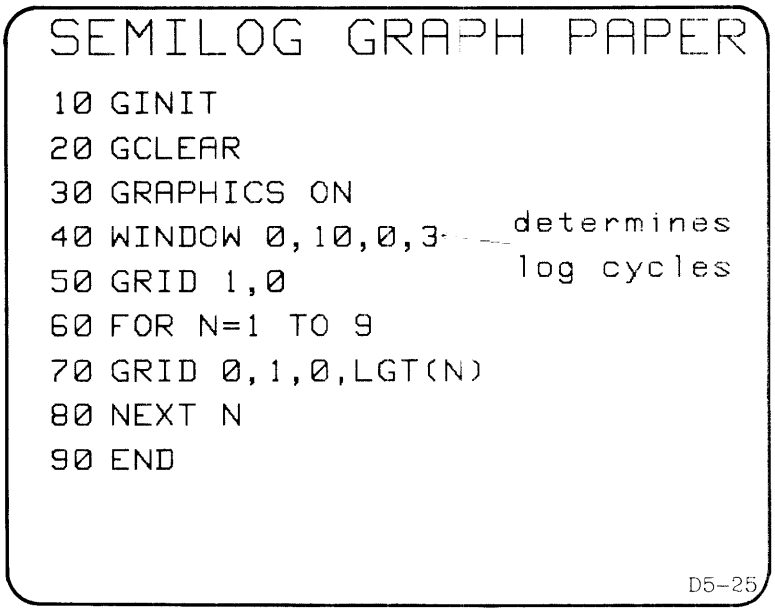

AXES 2,2, 50,50, 5,5, 10
Tick Spacing
Axes Locations
Major Tick Count
Major Tick Size
D5-23

NOTES


Put a GRID on your plot.
-Similar to AXES statement.
-Major ticks extend across plot.
-Cross-ticks drawn at intersections of minor ticks.
GRID 5,5,50,50,5,10
Tick Spacing
Axes Locations
Major Tick Count
D5-24

NOTES

NOTES




EXERCISE 31

Modify the semilog grid program to produce a 4 cycle Y axis loglog grid. You could either add another FOR-NEXT loop or modify the WINDOW and GRID statements to draw lines on both the X and Y axes on an LGT basis.

# LABELING YOUR GRAPHS

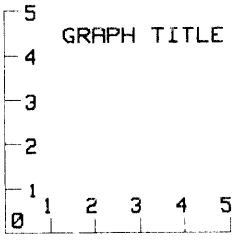
- LABEL/CSIZ/LDIR/LORG
- Labels are drawn at current pen position.
- Character strings, numbers, or arrays can be labels.
- The program can specify:
  - Label character size (CSIZ)
  - Orientation (angle) of label (LDIR)
  - Relative origin of label (LORG)

D5-26

NOTES

# TITLES AND AXES LABELS

```
 10 GINIT
 20 GRAPHICS ON
 30 WINDOW 0,5.5,0,5.5
 40 AXES 1,1
 50 MOVE 2,5
 60 LABEL "GRAPH TITLE"
 70 FOR X=0 TO 5
 80 MOVE X,0
 90 LABEL X
100 NEXT X
110 FOR Y=0 TO 5
120 MOVE 0,Y
130 LABEL Y
140 NEXT Y
150 END
```




D5-27

NOTES

NOTES

## LABEL CHARACTER SIZE

CSIZE

- Label Characters Have:
  - Default Height (5 GDUs)
  - Default Width (3 GDUs)
  - Aspect Ratio = 3/5 = .6
- Height and Aspect Ratio set by CSIZE
- Change "Graph Title" to:
  - 8 GDUs high, 3 wide
  - CSIZE 8, 3/8

D5-28

NOTES

## LABEL ORIENTATION

LDIR

- Labels can be oriented in any direction, as appropriate.
- Label direction is specified in current angular units
  - RAD or DEG
- Add to end of program:

```
MOVE 0,0
DRAW 4,4
MOVE 2,2
DEG
LDIR 45
LABEL "Bisected Angle"
```

D5-29

# RELATIVE ORIGIN OF LABELS

LORG

- Normally, labels are drawn above and to the right of the current pen position.
- Use LORG to specify other relative placements:
  - below-left, centered-above, etc.
  - LORG 7

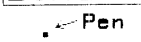
D5-30

NOTES

3. 6. 9.
2. LABEL 8. DEFAULT = LORG 1
1. 4. 7.

Normal pen position/label placement

LORG 9 (Below-left)




LORG 4 (Center-above)


D5-31

NOTES

## EXERCISE 32

Modify the program "LABELS" so that the X axis labels are below the axis and the Y axis labels are to the left of the axis.

## EXERCISE 33

Modify the program "READ_SIN" to plot the data after it is read off the file "SINDAT". Your plot routine should be written as a subprogram, with the number of data elements and the data array passed as formal parameters.

The plot should be framed, with labels and axes as appropriate. Contrast these graphic results with the printout of the data you performed previously.

# PLOTTING BOUNDARIES

Device-defined boundaries
- Hard-clip limits
  - Edge of CRT
  - Edge of plotter bed

User-defined boundaries
- Soft-clip boundaries
  - VIEWPORT boundaries : GDU's
  - CLIP boundaries : UDU'S/GDU's

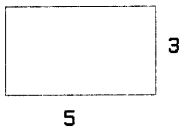
```
CLIP Left, Right, Bottom, Top
CLIP ON
CLIP OFF
```

D5-32

NOTES

# RELATIVE PLOTTING

Useful for drawing objects:




IMOVE, IDRAW specify ΔX, ΔY
(± X units across, ±Y units up)

```
IDRAW 5,0    (5 units across)
IDRAW 0,3    (3 units up)
IDRAW -5,0   (5 units back)
IDRAW 0,-3   (3 units down)
```

D5-33

NOTES

NOTES

## ROTATIONAL PLOTTING

Used primarily to rotate objects:

PIVOT specifies new orientation of X and Y axes for MOVE, DRAW, IMOVE, IDRAW.

```
PIVOT  45 or PIVOT PI/4
PIVOT 135 or PIVOT 3*PI/4
```

D5-34

NOTES

## ESOTERICS & REVIEW

Clipping: Limits & boundaries

1. Hard-clip limits
   - Absolute limits of plotting area
   - Physical device boundaries
   - ALWAYS 100 GDUs on short side
2. Soft-clip boundaries
   - User-defined plotting area boundaries
   - Established by VIEWPORT or CLIP
   - Soft-clipping regulated by CLIP ON/CLIP OFF
   - If CLIP ON, no plotting outside soft-clip boundaries

D5-35

Plotting units:

1. Graphic Display Units (GDUs)
   - Defined as 1/100 of device's short side
   - Isotropic:
     X unit size = Y unit size
   - Locates plotting area on plotting device
2. User Defined Units (UDUs)
   - Plotting area scaled to range of data
   - Can be Isotropic (X unit = Y unit) or Nonisotropic (X unit ≠ Y unit)

D5-36

NOTES

3. Isotropic UDUs
   - Defined by SHOW statement
   - Centered in VIEWPORT area
   - Necessary for drawing objects
     X units wide by Y units high

4. Nonisotropic UDUs
   - Defined by WINDOW statement
   - Mapped into VIEWPORT (soft-clip) area
   - Useful for graphing relationships [Y=f(X)]

D5-37

NOTES

NOTES

## Plotting: making lines

1. Absolute Plotting
   - Moves the pen to specified X,Y point
     (MOVE: pen up; DRAW: pen down)
   - Location specified in current units
     (GDUs or UDUs)
2. Relative Plotting
   - Moves the pen by specified X,Y distance
     (IMOVE: pen up; IDRAW: pen down)
   - Distance specified in current units
     (GDUs or UDUs)

D5-38

NOTES

3. Physical Pen
   - Restricted by clipping limits
   - Moved by DRAW, IDRAW
   - Lifted by PENUP (useful on plotter)
   - Not moved by DRAW/IDRAW outside clip limits!
     (Logical pen is moved)
4. Logical Pen
   - Not restricted to clip limits
   - Updated by MOVE, IMOVE
     Also DRAW, IDRAW
   - Physical pen is relocated to Logical Pen coordinates by DRAW, IDRAW

D5-39

5. Plot Coordinates
   - Always measured in current units
   - Always along strictly horizontal and vertical axes for Logical Pen placement, AXES, and LABEL
   - Can be "rotated" by PIVOT for drawing lines. Labels & axes not affected
     - PIVOT specifies angle in current angular measure (DEG/RAD)
     - PIVOT can be used to rotate objects being drawn
     - Use with care!

D5-40

NOTES

## INTERNAL GRAPHICS CONTROL

- Initialize: GINIT
- Clear screen: GCLEAR
- Turn off alpha: ALPHA OFF
  (also ALPHA ON)
- Turn on graphics: GRAPHICS ON
  (also GRAPHICS OFF)
- Hardcopy graphics: DUMP GRAPHICS
  (DUMP GRAPHICS #701)
- Save graphics: GSTORE Array (*)
- Recall graphics: GLOAD Array (*)

D5-41

NOTES

NOTES

# EXTERNAL PLOTTING

## PLOTTER IS

-Default plotting device is CRT:

```
PLOTTER IS 3, "INTERNAL"
```

-Programs can also be directed to external plotters (except for graphics control operations)

```
PLOTTER IS 705, "HPGL"
```

D5-42

NOTES

# EXTERNAL PLOTTING

- Explicit Plotter Control
  - What is HPGL?
  - Hewlett-Packard Graphics Language
- Select pen speed (fast/slow):
- Select alternate character sets:

D5-43

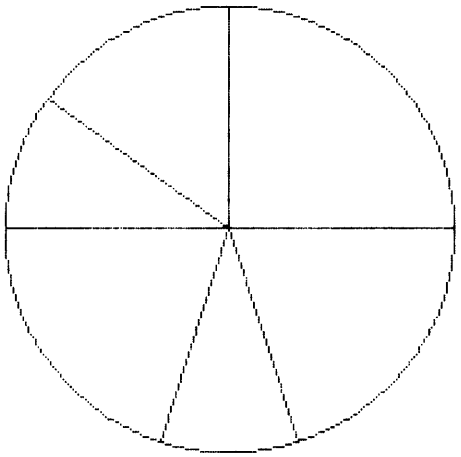
# EXERCISE 34

Write a program that plots percentage data as a pie-chart graph.

There are a number of ways to attack the problem, but one way is to read the next percent and call a circle-drawing subprogram that draws the given percent of a circle.

Example data : 25, 15, 10, 20, 10, 20,

Resulting Pie Chart :



Write a program that implements a bar-meter type readout of a changing signal.

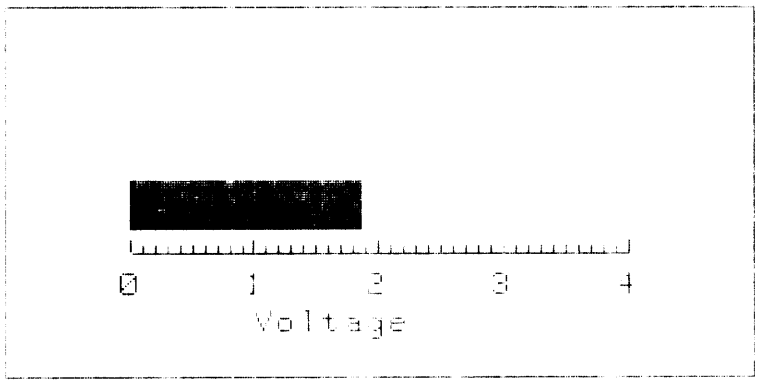
Incoming signal values should be serviced by an interrupt service routine - in this case, a knob-service routine.

While in the routine, update the bar-meter by drawing to the right or erasing to the left, depending on the new signal value (greater to or less than the old signal value).

(Although this program uses the knob as its signal source, that source could easily be data from an HP-IB instrument.)

Your main program here should consist only of set-up and an endless loop. In a more complex application the main program could be executing various other tasks, since updating the display is the responsibility of the service routine.

Example readout :



D1-24

```
10    !  "EDIT1"
20    !  The following lines (100,110,120) are for
30    !  editing.
40    !  The exclamation marks are equivalent to
50    !  the standard BASIC REM(ARK) statement.
60    !
70    !  Remove the exclamation mark, correct the
80    !  line, then press ENTER to store it.
90    !
100!  PRENT "This keyword is misspelled."
110!  PRINT "Your NAME here.
120!  PRINT  Your ADDRESS here"
130 !
140 !
150 GOTO 190 ! This is an illegal GOTO at RUN time
160 !
170 !
180  END
190  PRINT   ! This line must be deleted.
200  GOTO 180! So must this one.
```

D1-31

```
10 ! SAVE "REVIEW"
20 REM  Program to compute Mean, Variance, and
30 REM  Standard Deviation
40 !
50 !
60 REM  Initialization section
70   OPTION BASE 1
80   DIM X(10)
90   DATA 20,45,13,64,85,97,59,34,72,6
100  FOR I=1 TO 10
110     READ X(I)
120  NEXT I
130  Sum=0
140  Sumofsquares=0
150  Numofitems=10
160  !
170  !
180  REM  Compute sum and sum of squares
190  FOR I=1 TO Numofitems
200     Sum=Sum+X(I)
210     Sumofsquares=Sumofsquares+X(I)*X(I)
220  NEXT I
230  !
240  !
250  Mean=Sum/Numofitems
260  Variance=(Sumofsquares-Sum*Sum/Numofitems)/(Numofitems-1)
270  Std_dev=SQR(Variance)
280  GOSUB 320
290  STOP
300  !
310  !
320  PRINT "Index ";TAB(11);"Data"
330  FOR I=1 TO Numofitems
340     PRINT TAB(2);I;TAB(11);X(I)
350  NEXT I
360  !
370  PRINT
380  PRINT "Mean= ";DROUND(Mean,5)
390  PRINT "Variance= ";DROUND(Variance,5)
400  PRINT "Standard Deviation= ";DROUND(Std_dev,5)
410  RETURN
420  END
```

```
10 ! SAVE "LOOPTIME"
20 ! This program can be used to measure the execution time of BASIC program sta
tements.
30 ! To use it, replace line 130  with the program line to be timed.
40     INTEGER One,Two,Three    ! Some integer vars
50     REAL Four,Five,Six       ! Some real vars
60     SET TIME 0
70 ! Measure FOR-NEXT loop time.
80       FOR I=1 TO 10000
90       NEXT I
100    Time1=TIMEDATE
110 ! Now try different statements in line 130.
120      FOR I=1 TO 10000
130      ! ****** Test statement goes here. ******
140      NEXT I
150    Time2=TIMEDATE
160 ! The difference between the two times, divided by 10000 is the statement tim
e.
170    Statement_time=(Time2-Time1) MOD 86400/10000
180    PRINT "Execution time is",Statement_time*1000;" milliseconds."
190    END
```

```
10! SAVE "MONTH_DAY"
20!         This program takes an input of the form       mm/dd/yy and produces a
n output of the form
30!     Month Day, Year
40!
50     OPTION BASE 1          ! Lower bound=1
60     DIM Month$(12)[10]
70     Month$(1)="JANUARY"   ! Initialize names
80     Month$(2)="FEBRUARY"  ! of months.
90     Month$(3)="MARCH"
100    Month$(4)="APRIL"
110    Month$(5)="MAY"
120    Month$(6)="JUNE"
130    Month$(7)="JULY"
140    Month$(8)="AUGUST"
150    Month$(9)="SEPTEMBER"
160    Month$(10)="OCTOBER"
170    Month$(11)="NOVEMBER"
180    Month$(12)="DECEMBER"
190!
200    INPUT "Enter mm/dd/yy",String$
210!
220!     Now find the number of the month.
230    Answer$=Month$(VAL(String$))
240!
250!     Set a pointer past the first "/".
260    Temp=POS(String$,"/")+1
270!
280!     Start building the output as Month & day.
290    Answer$=Answer$&" "&String$[Temp;POS(String$[Temp],"/")-1]
300    Answer$=Answer$&","
310!
320!     Now set a pointer past the second "/".
330    Temp=Temp+POS(String$[Temp],"/")
340!
350!     Add the year information as "19yy".
360    Answer$=Answer$&"19"&String$[Temp;2]
370    PRINT Answer$
380    END
```

```
10 ! SAVE "SCH_RPL"
20    DIM Search_in$(150)[100]
30    !  The next line defines the name of the                ASCII file to be sear
ched.
40    Program$="CIRCLE"
50    !  Read in the ASCII file until end-of-file
60    ASSIGN @Pipe TO Program$
70    ON END @Pipe GOSUB Done
80    I=0
90    REPEAT
100      I=I+1
110      ENTER @Pipe;Search_in$(I)
120   UNTIL Eof
130   Num_elem=I
140 GOSUB Search_replace
150   INPUT "Shall I print the entire array?",S$
160     IF POS(S$,"y") OR POS(S$,"Y") THEN
170        FOR I=1 TO Num_elem
180        PRINT Search_in$(I)
190        NEXT I
200     ELSE
210     END IF
220   STOP
230 Done:    ASSIGN @Pipe TO *    ! Stop reading
240          Eof=1
250          RETURN
260 !*******************************************
270 ! Append your subroutine starting here.
280 ! GET "TEMP"
```

```
10! SAVE "SCHRPL_SUB"
20 Search_replace: ! One possible solution.
30        INPUT "Enter the search string",A$
40        INPUT "Enter the replace string",B$
50      FOR I=1 TO Num_elem
60        X=POS(Search_in$(I),A$)
70        IF X THEN
80           PRINT "The line was:"
90           PRINT Search_in$(I)
100          Search_in$(I)=Search_in$(I)[1,X-1]&B$&Search_in$(I)[X+LEN(A$)]
110          PRINT "The line now is:"
120          PRINT Search_in$(I)
130          PRINT
140          WAIT .3
150       ELSE
160       END IF
170     NEXT I
180       PRINT
190       PRINT
200 RETURN
210     END
```

```
10 ! SAVE "IFTHENELSE"
20    INPUT "Please enter some number",Number
30   FOR Count=15 TO 0 STEP -1
40   !
50    IF BIT(Number,Count) THEN
60        PRINT "1";
70    ELSE
80        PRINT "0";
90    END IF
100  !
110  NEXT Count
120   PRINT
130   GOTO 20
140   END
```

Solution 3

```
10! SAVE "SELECTCASE"
20  INPUT "Enter a number",Number
30  FOR Count=15 TO 0 STEP -1
40       SELECT BIT(Number,Count)
50          CASE 0
60          PRINT "0";
70          CASE 1
80          PRINT "1";
90       END SELECT
100 NEXT Count
110 END
```

Solution 4a

```
10 ! SAVE "UPPERCASE"
20    INPUT "ENTER A STRING",A$
30    !
40    !  Look at one character at a time
50  FOR I=1 TO LEN(A$)
60    SELECT A$[I,I]
70    !
80    !  If the character is in the range:
90    CASE "a" TO "z"
100   !  then subtract 32 from it's ASCII value
110    A$[I,I]=CHR$(NUM(A$[I,I])-32)
120   END SELECT
130 NEXT I
140   PRINT A$
150   END
```

Solution 4b

```
10 ! SAVE "UPC_LWC"
20    !  Function$ determines whether this program          does uppercase or low
ercase.
30    Function$="LWC"    ! Either UPC or LWC
40    !
50    DIM A$[80]
60    INPUT "ENTER A STRING",A$
70    !
80    !  Look at one character at a time
90  FOR I=1 TO LEN(A$)
100   SELECT A$[I,I]
110    !
120   CASE "a" TO "z"
130    !  If the character is in range and the                function is UPC,
140    !  then subtract 32 from it's ASCII value
150      IF Function$="UPC" THEN
160         A$[I,I]=CHR$(NUM(A$[I,I])-32)
170      ELSE
180      END IF
190    CASE "A" TO "Z"
200    !  Now try to see if function is LWC:
210      IF Function$="LWC" THEN
220         A$[I,I]=CHR$(NUM(A$[I,I])+32)
230      ELSE
240      END IF
250   END SELECT
260 NEXT I
270   PRINT A$
280   END
```

Solution 5

```
10! SAVE "REPEAT"
20  INPUT "NUMBER?",Number
30  Count=15
40  REPEAT
50    IF BIT(Number,Count) THEN
60       PRINT "1";
70    ELSE
80       PRINT "0";
90    END IF
100   Count=Count-1
110   UNTIL Count<0
120 PRINT
130   END
```

Solution 6

```
10! SAVE "WHILE"
20  INPUT "NUMBER?",Number
30  Count=15
40    WHILE Count>=0
50      IF BIT(Number,Count) THEN
60         PRINT "1";
70      ELSE
80         PRINT "0";
90      END IF
100   Count=Count-1
110   END WHILE
120  PRINT
130  END
```

```
1 ! SAVE "VAL_REF"
10    PRINT " PASS BY VALUE"
20    PRINT " I","FNSq1(I)","<Does not alter passed param>"
30    PRINT "--","--------"
40    FOR I=1 TO 10
50      PRINT I,FNSq1((I))
60    NEXT I
70    PRINT
80    !
90    !
100   PRINT " PASS BY REFERENCE"
110   PRINT " I","FNSq1(I)","<Does not alter passed param>"
120   PRINT "--","--------"
130   FOR I=1 TO 10
140     PRINT I,FNSq1(I)
150   NEXT I
160   PRINT
170   !
180   !
190   PRINT " PASS BY VALUE"
200   PRINT " I","FNSq2(I)","<Tries to alter passed param>"
210   PRINT "--","--------"
220   FOR I=1 TO 10
230     PRINT I,FNSq2((I))
240   NEXT I
250   PRINT
260   !
270   !
280   PRINT " PASS BY REFERENCE"
290   PRINT " I","FNSq2(I)","<Tries to alter passed param>"
300   PRINT "--","--------"
310   FOR I=1 TO 10
320     PRINT I,FNSq2(I)
330   NEXT I
340   END
350   !
360   !
370   DEF FNSq1(Num)  ! Does not alter pass params
380     RETURN Num^2
390   FNEND
400   !
410   !
420   DEF FNSq2(Num)  ! Alters passed parameters
430   Num=Num^2
440     RETURN Num
450   FNEND
```

```
1 ! SAVE "NPAR"
10    INTEGER A
20    A=1
30    B=2     ! This is a REAL number
40    C=3     ! This is a REAL number
50    !
60    !
70    PRINT "Main program values"
80    PRINT " A, B, C"
90    PRINT A;B;C
100   PRINT
110   !
120   !
130   PRINT "Pass by value:  A,B,C"
140   CALL Printout(1*A,1*B,(C)) ! Pass by value
150   PRINT
160   !
170   !
180   PRINT "Pass by reference:  A,B"
190   CALL Printout(A,B)    ! Pass by reference
200   END
210   !
220   !
230   !
240   SUB Printout(X,Y,OPTIONAL Z)
250  IF NPAR=3 THEN
260   PRINT "In subprogram, got optional param Z"
270   PRINT " A, B, C, X, Y, Z"
280   PRINT A;B;C;X;Y;Z
290   PRINT
300  ELSE
310   PRINT "In subprogram, no optional param"
320   PRINT " A, B, C, X, Y, no Z"
330   PRINT A;B;C;X;Y
340   PRINT
350  END IF
360   SUBEND
```

```
1 ! SAVE "LABEL_COM"
10    A=1
20    B=2
30    C=3
40    COM A,B,C
50    PRINT "Values in Main before call"
60    PRINT " A, B, C, L, M, N"
70    PRINT A;B;C;L;M;N
80    !
90    !
100   CALL Printout1
110   PRINT "Values in Main after call"
120   PRINT " A, B, C, L, M, N"
130   PRINT A;B;C;L;M;N
140   !
150   CALL Printout2
160   END
170   !
180   !
190   !
200   SUB Printout1
210   COM A,B,C
220   COM /Crypto/ L,M,N
230   A=4
240   B=5
250   C=6
260   L=7
270   M=8
280   N=9
290   PRINT
300   PRINT "First Sub COM  values "
310   PRINT " A, B, C, L, M, N"
320   PRINT A;B;C;L;M;N
330   PRINT
340   SUBEND
350   !
360   SUB Printout2
370   COM /Crypto/ L,M,N
380   PRINT
390   PRINT "Second Sub COM  values "
400   PRINT " A, B, C, L, M, N"
410   PRINT A;B;C;L;M;N
420   PRINT
430   SUBEND
```

```
1 ! SAVE "KEYS1"
10     ON KEY 0 LABEL "PRIO_1",1 CALL Sub1
20     ON KEY 1 LABEL "PRIO_5",5 CALL Sub5
30     ON KEY 2 LABEL "PRIO_14",14 CALL Sub14
40     ON KEY 3 LABEL "ABORT",15 RECOVER Abort
50 Loop:    !
60          DISP I,"Main"
70          I=I+1
80          GOTO Loop
90          !
100 Abort: PRINT "Aborted operations"
110         STOP
120         END
130         !
140         !
150 SUB Sub1
160         PRINT "SUB: Priority 1"
170         FOR I=1 TO 20000
180         NEXT I
190         SUBEND
200         !
210  SUB Sub5
220         PRINT "SUB: Priority 5"
230         FOR I=1 TO 15000
240         NEXT I
250         SUBEND
260         !
270  SUB Sub14
280       ON KEY 3 LABEL "STOP",15 GOTO 330
290         PRINT "SUB: Priority 14"
300         FOR I=1 TO 15000
310         NEXT I
320         SUBEXIT
330         STOP
340         SUBEND
```

Solution 11

```
1 ! SAVE "PRIORITIES"
10    ON KEY 0 LABEL "CALL P5",5 CALL Pri5
20    ON KEY 2 LABEL "CALL P10",10 CALL Pri10
30    ON KEY 4 LABEL "CALL P14",14 CALL Pri14
40    ON KEY 5 LABEL "GSUB P15",15 GOSUB Pri15
50    C=C+1
60    DISP " IN MAIN, COUNTER =";C
70    WAIT .1
80    GOTO 50
90 Pri15:    !
100          FOR X=1 TO 20
110          DISP "IN SUBROUTINE, PRIORITY 15";X
120          WAIT .1
130          NEXT X
140 RETURN
150   END
160   !
170   !
180   !
190 SUB Pri5
200       FOR X=1 TO 20
210          DISP "IN Subprogram, PRIORITY 5:";X
220       WAIT .15
230       NEXT X
240 SUBEND
250 !
260 SUB Pri10
270    ON KEY 4 LABEL "*********",1 CALL Pri14
280       FOR X=1 TO 20
290       DISP "          In Subprogram,PRIORITY 10:";X
300       WAIT .15
310       NEXT X
320 SUBEND
330 !
340 SUB Pri14
350 !   Note that almost any priority here can-          cels the key service.
360    ON KEY 0 LABEL "*********",1 CALL Pri5
370    ON KEY 2 LABEL "*********",1 CALL Pri10
380       FOR X=1 TO 20
390       DISP "                    In subprogram, PRIORITY 14:";X
400       WAIT .15
410       NEXT X
420 SUBEND
```

Solution 12

```
1 ! SAVE "BEEP"
10    ON KNOB .5 GOSUB Beeper
20    DISP "Main: ";X
30    X=X+1
40    GOTO 20
41    !
50 Beeper:    !
60            Y=ABS(KNOBX)
61            DISP "Service: ";Y
70            BEEP Y*10,.2
71            WAIT .2
80            RETURN
90    END
```

```
1 ! SAVE "MAIN"
2     LOADSUB ALL FROM "KEYSUBS"
10    ON KEY 0 LABEL "CALL P5",5 CALL Pri5
20    ON KEY 2 LABEL "CALL P10",10 CALL Pri10
30    ON KEY 4 LABEL "CALL P14",14 CALL Pri14
40    ON KEY 5 LABEL "GSUB P15",15 GOSUB Pri15
50    C=C+1
60    DISP " IN MAIN, COUNTER =";C
70    WAIT .1
80    GOTO 50
90 Pri15:   !
100         FOR X=1 TO 20
110         DISP "IN SUBROUTINE, PRIORITY 15";X
120         WAIT .1
130         NEXT X
140 RETURN
150   END
```

```
10 ! STORE "KEYSUBS"
190 SUB Pri5
200       FOR X=1 TO 20
210          DISP "IN Subprogram, PRIORITY 5:";X
220       WAIT .15
230       NEXT X
240 SUBEND
250 !
260 SUB Pri10
270     ON KEY 4 LABEL "********",1 CALL Pri14
280       FOR X=1 TO 20
290       DISP "          In Subprogram,PRIORITY 10:";X
300       WAIT .15
310       NEXT X
320 SUBEND
330 !
340 SUB Pri14
350 !   Note that almost any priority here can-          cels the key service.
360     ON KEY 0 LABEL "********",1 CALL Pri5
370     ON KEY 2 LABEL "********",1 CALL Pri10
380       FOR X=1 TO 20
390       DISP "                    In subprogram, PRIORITY 14:";X
400       WAIT .15
410       NEXT X
420 SUBEND
```

```
1 ! SAVE "GETLOADSUB"
10    ! Put some variables in common so they don't get clobbered by
 GET
20    !
30    COM /A/ One,Two
40    COM Three,Four
50    !
60    ! Next assign some values to six variables (some not in COM)
70    !
80    DATA 1,2,3,4,5,6
90    READ One,Two,Three,Four,Five,Six
100   !
110   !
120   PRINT "Before CALL",One;Two;Three;Four;Five;Six
130   CALL Sub_first
140   !
150   !
160   PRINT "After CALL  ",One;Two;Three;Four;Five;Six
170   !
180   !
190   DELSUB Sub_first
200   PRINT "After DELETE",One;Two;Three;Four;Five;Six
210   !
220   !
230   LOADSUB ALL FROM "SUB"
240   CALL Sub_load
250   PRINT "After LOADSUB",One;Two;Three;Four;Five;Six
260   !
270   !
280   ! Now GET a sub at End and execute next line
290   GET "GETSUB",End,Next_line
300 Next_line:  ! This line is executed next.
310   CALL Sub_get
320   PRINT "After GET   ",One;Two;Three;Four;Five;Six
330   END
340   !
350   !
360 End:  SUB Sub_first
370       PRINT "IN SUB Sub_first"
380       SUBEND
```

```
10 ! RE-SAVE "GETSUB"
20 End:  SUB Sub_get
30    PRINT "In SUB Sub_get"
40    SUBEND
```

```
10 ! RE-STORE "SUB"
20 End:SUB Sub_load
30    PRINT "In SUB Sub_load"
40    SUBEND
```

```
1 ! SAVE "ASCII_WRT"
2 ! Writes to "TEST" and reads back in.
3 ! This version is more enhanced than the one
4 ! on the slides...more room.
5 !
6 !
10      CREATE ASCII "TEST",10
11     !
12     !
20      ASSIGN @Name TO "TEST"
30      OUTPUT @Name;"ED","SUE"
40      OUTPUT @Name;"ALVIN"
41     !
50      ASSIGN @Name TO "TEST"
60      ENTER @Name;A$,B$,C$
70      PRINT A$,B$,C$
80      END
```

Solution 16

```
1 ! SAVE "UPDATE"
10     ! This program updates the file "OLD_DATA"
20     DIM A$[20]
30     ! Create the destination file
40     CREATE ASCII "NEW_DATA",10
50     !
60     !
70     ASSIGN @Old TO "OLD_DATA"
80     ASSIGN @New TO "NEW_DATA"
90     !
100    FOR I=1 TO 20
110    ENTER @Old;A$    ! Read the data item
120      SELECT A$
130        CASE "MISISSIPPI"  ! Check for update
140          A$="MISSISSIPPI"
150          PRINT A$;TAB(20);I
160        CASE "NABRASKA"     ! Check for update
170          A$="NEBRASKA"
180          PRINT A$;TAB(20);I
190        CASE ELSE
200      END SELECT
210    !
220    OUTPUT @New;A$    ! Write the updated data
230    NEXT I
240    !
250    !
260    ASSIGN @Pipe TO *
270    ASSIGN @New TO *
280   END
```

Solution 17

```
10! SAVE "WRITE_SIN"
20    DEG   ! Set degrees mode
30    ON ERROR GOTO 50  ! In case file exists...
40    CREATE BDAT "SINDAT",1,800
50    OFF ERROR
60    !
70    ASSIGN @Pipe TO "SINDAT"
80    OPTION BASE 1
90    DIM Volts(100)
100   !
110   !
120   CALL Compute(Volts(*))
130   !
140   !
150   OUTPUT @Pipe;Volts(*)
160   ASSIGN @Pipe TO *
170   END
180   !
190   !
200   SUB Compute(Array(*))
210   !  Compute 100 values of 3 cycles of a sine
220   !  Results go to Array
230     I=1  ! I is the 100 step counter
240   REPEAT
250     X=(I)*360*3/100  ! 3 cycles, 100 points
260     Array(I)=SIN(X)*10
270     I=I+1
280   UNTIL I>100
290   SUBEND
```

Exercise 18

```
10! SAVE "BLD_DATA"
20!
30    OPTION BASE 1
40    DIM A$(20)[20]
50    !
60    !
70    DATA MISISSIPPI,NABRASKA,COLORADO,OREGON,KANSAS,MISSOURI,OHIO
80    DATA OREGON,MISISSIPPI,NORTH DAKOTA,SOUTH DAKOTA,WYOMING,NEVADA
90    DATA MISISSIPPI,NABRASKA,NEBRASKA,MISSISSIPPI,PENNSYLVANIA,VIRGINIA
100   DATA NEW YORK,FLORIDA
110   !
120   !
130      DISP "Preparing Data"
140   FOR I=1 TO 20
150   READ A$(I)
160   NEXT I
170   !
180      DISP "Loading subprogram"
190   LOADSUB ALL FROM "WRITEBDAT"
200      DISP "Calling subprogram"
210   CALL Write_data(A$(*))
220  END
```

```
10! STORE "writebdat"
20 !
30    SUB Write_data(Data$(*))
40 !
50    OPTION BASE 1
60 !
70 !
80 !  This segment attempts to set up the file
90       DISP "Setting up data file"
100   ON ERROR GOTO 120
110   PURGE "BDAT_DATA"
120   CREATE BDAT "BDAT_DATA",20,25
130   OFF ERROR
140!
150!
160!  This segment writes the data to the file
170      DISP "Writing data to file"
180   ASSIGN @Pipe TO "BDAT_DATA"
190   FOR I=1 TO 20
200      OUTPUT @Pipe,I;Data$(I)
210      PRINT I,Data$(I)
220   NEXT I
230   ASSIGN @Pipe TO *
240 !
250   SUBEND
```

```
10! SAVE "RAND_UPDT"
20!
30     OPTION BASE 1
40     DIM A$(20)[20]
50     !
60     !
70        DISP "Reading Data"
80     PRINT "Record";TAB(10);"Item"
90     ASSIGN @Pipe TO "BDAT_DATA"
100    FOR I=1 TO 20
110       ENTER @Pipe,I;A$(I)
120       PRINT I;TAB(10);A$(I)
130    NEXT I
140    !
150    CALL Update(A$(*),@Pipe)
160    END
170    !
171    !
172    !
180 SUB Update(Data$(*),@File)
200    OPTION BASE 1
220!
230!
240!   This segment updates the file.
241    LOOP
250          INPUT "Record number to update? (0 when done)",Num
260    EXIT IF Num=0
270          INPUT "Enter the new item:",Data$(Num)
280!
290!   Write the new data to the file
300          DISP "Writing data to file"
310          OUTPUT @File,Num;Data$(Num)
320          PRINT Num,Data$(Num)
330          INPUT "Next record number to update? (0 when done)",Num
340    END LOOP
350          ASSIGN @File TO *
360 !
370    SUBEND
```

Solution 20

```
10! SAVE "TRAP_END"
20!
30    OPTION BASE 1
40    DIM A$(20)[20]
50    !
60    !
70       DISP "Reading Data"
80    PRINT "Record";TAB(10);"Item"
90    ASSIGN @Pipe TO "BDAT_DATA"
100 ON END @Pipe GOTO Continue !*****************
110       I=1                  !*****************
120 LOOP                       !*****************
130      ENTER @Pipe,I;A$(I)
140      PRINT I;TAB(10);A$(I)
150       I=I+1                !*****************
160 END LOOP                   !*****************
170   !
180 Continue:  CALL Update(A$(*),@Pipe)  !******
190   END
200   !
210   !
220   !
230 SUB Update(Data$(*),@File)
240   OPTION BASE 1
250!
260!  This segment updates the file.
270   LOOP
280        INPUT "Record number to update? (0 when done)",Num
290   EXIT IF Num=0
300        INPUT "Enter the new item:",Data$(Num)
310!
320!  Write the new data to the file
330        DISP "Writing data to file"
340        OUTPUT @File,Num;Data$(Num)
350        PRINT Num,Data$(Num)
360   END LOOP
370        ASSIGN @File TO *
380 !
390   SUBEND
```

Solution 21

```
10 ! SAVE "SPEED"
20   OPTION BASE 1
30 !  This Prog demo's a BDAT file with & w/o
40 !  an EOF at the end of data file "NUM_DAT".
50 !  Also, FORMAT ON/OFF is illustrated.
60 !
70 !  The variable Pass_num indicates whether we
80 !  have set up the disc and computed the array
90 !  previously.  It is in COM so its value is
100!  preserved from run to run.
110 !
120   DIM Array(1200)
130   INTEGER Pass_num
140 !
150   IF Pass_num=0 THEN Pass_num=1
160 !
170 Loop_again:!
180    PRINT "Number ";Pass_num,
190 !
200 !  This loop sets up the data file on disk.
210 !
220 IF Pass_num=1 THEN
230      DISP "Setting up disc, please wait "
240 !
```

```
250       ON ERROR GOTO 270
260       PURGE "NUM_DAT"
270       OFF ERROR
280 !
290      CREATE BDAT "NUM_DAT",1,20000
300 END IF
310 !
320 !
330 !
340 ! This section toggles FORMAT ON/OFF
350 !
360   IF Pass_num<=4 THEN
370      ASSIGN @Pipe TO "NUM_DAT";FORMAT ON
380      PRINT "  FORMAT ON"
390   ELSE
400      ASSIGN @Pipe TO "NUM_DAT";FORMAT OFF
410      PRINT "  FORMAT OFF"
420   END IF
430   CONTROL @Pipe,7;1,1 !Set EOF to byte # 1
440 !
450 !
460 IF Pass_num=1 THEN
470   DISP "Calculating data items"
480     FOR I=1 TO 1200
490        Array(I)=RND*10^(RND*10)
500     NEXT I
510 END IF
520 !
530 !
540  SELECT Pass_num
550   CASE 3,4
560     CONTROL @Pipe,7;1,20000
570     PRINT "Moved EOF to end of file"
580          ! Reg 7&8 of @Pipe puts an EOF mark
590          ! at the 20000th byte of NUM_DAT
600   END SELECT
610 !
620 !
630   DISP "Writing data to disc"
640   T0=TIMEDATE
650       IF Pass_num MOD 2=1 THEN  ! (It's ODD)
660           PRINT "Outputting entire array (1200 elements)"
670           OUTPUT @Pipe,1;Array(*),
680       ELSE
690           PRINT "Outputting 120 elements one at a time. (1200 takes too long
)"
700           FOR I=1 TO 120
710             OUTPUT @Pipe;Array(I),
720           NEXT I
730       END IF
740   !
750       STATUS @Pipe,6;I
760       PRINT "Total bytes on data file =";I
770       ASSIGN @Pipe TO *
780   !
790   !
800   T1=TIMEDATE
810   BEEP 3500,.5
820   DISP
830   PRINT "Disc Transfer Time =";DROUND(T1-T0,4);"sec."
840   BEEP
850   Pass_num=Pass_num+1
860   PRINT
870   IF Pass_num<=6 THEN GOTO Loop_again
880   END
```

Solution 22

```
10 ! SAVE "READ_SIN"
20    ON ERROR GOTO Problem  ! Just in case...
30    OPTION BASE 1
40    DIM Volts(100)
50 !
60 !
70 ! Read from disc file
80    ASSIGN @Pipe TO "SINDAT"
90    ENTER @Pipe;Volts(*)
100   ASSIGN @Pipe TO *
110!
120! Write to CRT
130   ASSIGN @Pipe TO 1
140   OUTPUT @Pipe;Volts(*)
150!
160! Write to Printer
170   ASSIGN @Pipe TO 701
180   OUTPUT @Pipe;Volts(*)
190   STOP
200!
210 Problem:  BEEP
220   PRINT "You probably didn't create file SINDAT."
230   PRINT "Run program WRITE_SIN to build the file."
240   END
```

D4-25

```
10!   SAVE "IMG_EXPLS"
20!
30! This "program" illustrates several methods
40! of using IMAGE with I/O statements.
50!
60!
70    IMAGE 10(S6D.2D)
80    OUTPUT 1 USING 70;1,2,3,4,5,6,7;8;9;10
90 !    Note either commas or semis are OK.
100!
110!
120   PRINT USING Image1;6.08
130 Image1:      IMAGE /,"Volts = ",K,/
140!  Note that the IMAGE statement can be
150!  referred to by line label...anywhere.
160!
170!
180   Image$="/,6X,2D.2DE,/"
190   OUTPUT 1 USING Image$;1,2,3,4,5
200!  Note that a string variable can be used
210!  to hold the image specifications.
220!  Note also that the data was forced to
230!  conform to the image: it doesn't look
240!  at all like 1,2,3,4, or 5!
250   END
```

```
10!  SAVE "SPECIFIERS"
20  PRINT "*** Digit specifier examples:"
30  PRINT "Number=1234: IMAGE 9Z",
40  PRINT USING "9Z";1234
50  PRINT "Number=1234: IMAGE 9D",
60  PRINT USING "9D";1234
70  PRINT
80  PRINT "*** Radix examples:"
90  PRINT "Number=1234: IMAGE 4Z.4D",
100 PRINT USING "4Z.4D";1234
110 PRINT "Number=1.236: IMAGE 4D.2D",
120 PRINT USING "4D.2D";1.236
130 PRINT "  (notice rounding)"
140 PRINT
150 PRINT "*** Exponent examples:"
160 PRINT "Number=1234: IMAGE 3D.DE",
170 PRINT USING "3D.DE";1234
180 PRINT "Number=1234: IMAGE 3D.DESZ",
190 PRINT USING "3D.DESZ";1234
200 PRINT "Number=1234: IMAGE 4D.DESZ",
210 PRINT USING "4D.DESZ";1234
220 PRINT "Number=1234: IMAGE 2D.DESZZZ",
230 PRINT USING "2D.DESZZZ";1234
240 PRINT "  (notice rounding)"
250 PRINT
260 PRINT "*** Sign examples:"
270 PRINT "Number=1234: IMAGE S2D.DE",
280 PRINT USING "S2D.DE";1234
290 PRINT "Number=1234: IMAGE M2D.DE",
300 PRINT USING "M2D.DE";1234
310 PRINT "Number=-1234: IMAGE M2D.DE",
320 PRINT USING "M2D.DE";-1234
330 PRINT
340 PRINT "*** Binary examples"
350 PRINT "Numbers= 94,126: IMAGE B,B",
360 PRINT USING "B";94,126
370 PRINT "Number= 19887: IMAGE W",
380 PRINT USING "W";19887
390 PRINT "Number= 12 (formfeed): IMAGE B",
400 PRINT USING "B";12
410 !
420 PRINT "*** String examples"
430 PRINT "String= 'text text'"
440 PRINT "IMAGE 3A ",TAB(31),
450 PRINT USING "3A";"text text"
460 PRINT "  (note truncation)"
470 PRINT
480 PRINT "String= 'text text'"
490 PRINT "IMAGE 20A",TAB(31),
500 PRINT USING "20A";"text text"
510 PRINT
520 PRINT "String= 'text','text'"
530 PRINT "IMAGE 4A,7X,4A ",TAB(31),
540 PRINT USING "4A,7X,4A";"text","text"
550 PRINT
560 PRINT "Number= 1.23"
570 PRINT USING "#,6A,B,6A,B,9A,8X";"IMAGE ",34,"Volts=",34,",3X,D.3D"
580 PRINT USING 590;1.23
590 IMAGE "Volts=",3X,D.3D
600 PRINT
610 PRINT "*** Record control examples:"
620 PRINT "SEND EOL: Numbers= 123.4, 5.678"
630 PRINT "IMAGE 4D.2D,2/,4D.2D   ";CHR$(10)
640 PRINT USING "4D.2D,2/,4D.2D";123.4,5.678
```

```
650 PRINT
660 PRINT "SUPPRESS EOL: Number1= 1.234"
670 PRINT "Number2= 5.678"
680 PRINT "Number1 IMAGE #,D.3D,3X"
690 PRINT "Number2 IMAGE D.3D"
700 PRINT
710 PRINT USING "#,D.3D,3X";1.234
720 PRINT USING "D.3D";5.678
730 PRINT
740 PRINT "Form-feed: IMAGE @"
750 PRINT USING "@"
760 END
```

D4-28

```
10!  SAVE "REP_FAC"
20    OPTION BASE 1
30    DIM Array(48),Image$[50]
40    DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20
50    DATA 21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40
60    DATA 41,42,43,44,45,46,47,48
70    FOR I=1 TO 48
80    READ Array(I)
90    NEXT I
100   Image$="24(S2D.4D,10X,S2D.4D,/)"
110   OUTPUT 1 USING Image$;Array(*)
120   END
```

Solution 23

```
10 ! SAVE "SIN_IMAGE"
20     ON ERROR GOTO Problem  ! Just in case...
30     OPTION BASE 1
40     DIM Volts(100)
50 Image1:     IMAGE 20(5(2D.3D,3X),/)
60 Image2:     IMAGE 10(10(2D.3D,2X),/)
70 !
80 !
90 ! Read from disc file
100    ASSIGN @Pipe TO "SINDAT"
110    ENTER @Pipe;Volts(*)
120    ASSIGN @Pipe TO *
130!
140! Write to CRT
150    ASSIGN @Pipe TO 1
160    OUTPUT @Pipe USING Image1;Volts(*)
170!
180! Write to Printer
190    ASSIGN @Pipe TO 701
200    OUTPUT @Pipe USING Image2;Volts(*)
210    STOP
220!
230 Problem:  BEEP
240    PRINT "You probably didn't create file SINDAT."
250    PRINT "Run program WRITE_SIN to build the file."
260    END
```

D4-29

```
10!  SAVE "KBD_OUT"
20    DIM A$[50]
30    OUTPUT 2 USING "#,K";"Edit and add to this text that you 'typed'."
40    ENTER 2;A$
50    PRINT A$
60    END
```

D4-30

```
10! SAVE "READ_CAT"
20    DIM A$[50]
30    OUTPUT 2 USING "#,K";"CAT%X" !Execute CAT
40    OUTPUT 2 USING "#,K";"%T" !Push top of form.
50    REPEAT
60    ! Read the display and scroll down one line.
70      PRINT TABXY(1,1);     !Set read position.
80    ENTER 1;A$              !Read the screen.
90      Count=Count+1
100     DISP A$
110     OUTPUT 2 USING "#,K";"%^" !Scroll down.
120     WAIT .15
130   UNTIL Count=60
140   END
```

Solution 24

```
10! SAVE "ON_KBD"
20   DIM Key$[20]
30   CONTROL 2,7;2
40   ON ERROR GOTO 60
50   !
60   ON KBD ALL GOSUB Keysvc
70   Key$=" "
80 Main:    I=I+1
90          DISP I,Key$,NUM(Key$[1])
100         GOTO Main
110  !
120 Keysvc:    Key$=KBD$
130            IF Key$="%%$" THEN STOP
140            RETURN
150   END
```

Solution 25

```
10! SAVE "INSTRUMENT"
20    ON ERROR GOTO Problem  ! Just in case...
30    OPTION BASE 1
40    DIM Volts(100)
50 !
60 !
70 ! Inserted lines to read the voltmeter.
80    ASSIGN @Fromdvm TO 705
90    ASSIGN @Todvm TO 705
100   OUTPUT @Todvm;"F1 R7 D40 T1"
110   FOR I=1 TO 100
120   ENTER @Fromdvm;Volts(I)
130   NEXT I
140!
150!
160! Write to disc file
170   ASSIGN @Pipe TO "SINDAT"
180   OUTPUT @Pipe;Volts(*)  ! Rewrite to file.
190   ASSIGN @Pipe TO *
200!
210! Write to CRT
220   ASSIGN @Pipe TO 1
230   OUTPUT @Pipe;Volts(*)
240!
250! Write to Printer
260   ASSIGN @Pipe TO 701
270   OUTPUT @Pipe;Volts(*)
280   STOP
290!
300 Problem:  BEEP
310   PRINT "You may have a problem with the instrument."
320   PRINT "Error Number",ERRN
330   END
```

Solution 26

```
10! SAVE "SRQ_SVC"
20  ON INTR 7 GOSUB Srqsvc
30  ENABLE INTR 7;2 ! Enable SRQ interrupts
31  !
32  !
40 Main:  DISP I
50        I=I+1
60        GOTO Main
70  !
80  !
90 Srqsvc:  Dev_status=SPOLL(701)
100    PRINT "Status returned:",Dev_status
110    RETURN
120 END
```

```
10! SAVE "POTPOURRI"
20    ON KEY 1 LABEL "GSB P1",1 GOSUB Pri1
30    ON KEY 2 LABEL "GSB P2",2 GOSUB Pri2
40    ON KEY 3 LABEL "GSB P3",3 GOSUB Pri3
50    ON KEY 19 LABEL "ABORT",15 RECOVER Pri15
60    ON ERROR RECOVER Pri15
70    ON KNOB .1,4 CALL Knobsvc
80    ON INTR 7,15 CALL Srqsvc
90    ENABLE INTR 7;2
100   !
110   !
120 Main:  C=C+1
130   DISP " IN MAIN, COUNTER =";C
140   WAIT .1
150   CALL Nest(1)
160   GOTO Main
170   !
180 Pri15:  !
190         FOR X=1 TO 500
200         DISP "RECOVERED to Main";X
210         NEXT X
220   GOTO Main
230   !
240   !
250 Pri1:   FOR I=1 TO 500
260         DISP "AT PRIORITY 1",I
270         NEXT I
280         RETURN
290         !
300 Pri2:   FOR I=1 TO 500
310         DISP "AT PRIORITY 2",I
320         NEXT I
330         RETURN
340         !
350 Pri3:   FOR I=1 TO 500
360         DISP "AT PRIORITY 3",I
370         NEXT I
380         RETURN
390   END
400   !
410   !
420 SUB Srqsvc
430      FOR X=1 TO 500
440         DISP "IN Subprogram, PRIORITY 15:";X
450      NEXT X
460 SUBEND
470 !
480 !
490 SUB Knobsvc
500      DISABLE
510      FOR X=1 TO 500
520      DISP "IN Knobsvc, interrupts disabled",X
530      NEXT X
540      ENABLE
550 SUBEND
560 !
570 !
580 SUB Nest(Nest)
590      DISP "Nesting level",Nest
600      ALLOCATE Temp(100)
610      Nest=Nest+1
620      WAIT .05
630      CALL Nest(Nest)
640 SUBEND
```

D5-6

```
1 ! SAVE "CIRCLE"
10    DEG
20    GINIT
30    GRAPHICS ON
40    Y=50
50    X=50
60    R=40
70    MOVE X,Y
80    FOR I=0 TO 360
90    DRAW X+R*COS(I),Y+R*SIN(I)
100   NEXT I
110   END
```

D5-7

```
1 ! SAVE "SIN_1"
10    DEG
20    GINIT
30    GRAPHICS ON
60    MOVE 0,50
70    FOR X=0 TO 360
80    DRAW X,SIN(X)
90    NEXT X
100   END
```

D5-13

```
1 ! SAVE "FOUR_SIN"
10    DEG
20    GINIT
30    GRAPHICS ON
40    VIEWPORT 0,60,0,45
50    GOSUB 130
60    VIEWPORT 65,130,0,45
70    GOSUB 130
80    VIEWPORT 0,60,55,100
90    GOSUB 130
100   VIEWPORT 65,130,55,100
110   GOSUB 130
120   STOP
130   WINDOW 0,360,-1.5,1.5
140   FRAME
150   MOVE 0,0
160   FOR X=0 TO 360
170   DRAW X,SIN(X)
180   NEXT X
190   RETURN
200   END
```

D5-15

```
1 ! SAVE "FOUR_SIN1"
10    DEG
20    GINIT
30    GRAPHICS ON
40    VIEWPORT 0,60,0,45
50    GOSUB 130
60    VIEWPORT 65,130,0,45
70    GOSUB 130
80    VIEWPORT 0,60,55,100
90    GOSUB 130
100   VIEWPORT 20,100,20,80
110   GOSUB 130
120   STOP
130   WINDOW 0,360,-1.5,1.5
140   FRAME
150   MOVE 0,0
160   FOR X=0 TO 360
170   DRAW X,SIN(X)
180   NEXT X
190   RETURN
200   END
```

Solution 29

```
1 ! SAVE "EGG"
10    DEG
20    GINIT
30    GRAPHICS ON
31    WINDOW 0,100,0,100
40    Y=50
50    X=50
60    R=40
70    MOVE X+R*COS(I),Y+R*SIN(I)
80    FOR I=0 TO 360
90    DRAW X+R*COS(I),Y+R*SIN(I)
100   NEXT I
110   END
```

D5-20

```
10! SAVE "LINES"
20    GINIT
30    GRAPHICS ON
40    CSIZE 4
50    N=4.5   ! Fudge factor for shrinking frames.
60    !
70    ! Step through 10 line types
80    FOR L_type=10 TO 1 STEP -1
90    !
100   LINE TYPE L_type,10
110   !
120   VIEWPORT L_type*N,133-L_type*N,L_type*N,100-L_type*N
130   FRAME
140   IMOVE 1,0   ! Scoot the label over a bit.
150   LINE TYPE 1 ! Select line type 1 for labels
160   LABEL "LINE TYPE ";L_type
170   NEXT L_type
180   !
190   END
```

Solution 30

```
10! SAVE "SIN_LINE"
20    DEG
30    GINIT
40    GRAPHICS ON
50    !
60    VIEWPORT 0,60,0,45
70    L_type=1
80    Rpt=5
90    GOSUB 290
100   !
110   VIEWPORT 65,130,0,45
120   L_type=3
130   Rpt=8
140   GOSUB 290
150   !
160   VIEWPORT 0,60,55,100
170   L_type=8
180   Rpt=11
190   GOSUB 290
200   !
210   VIEWPORT 65,130,55,100
220   L_type=5
230   Rpt=9
240   GOSUB 290
250   !
260   STOP
270   !
280   !
290   WINDOW 0,360,-1.5,1.5
300   LINE TYPE 1
310   FRAME
320   MOVE 0,0
330   LINE TYPE L_type,Rpt
340   !
350   FOR X=0 TO 360
360   DRAW X,SIN(X)
370   NEXT X
380   RETURN
390   END
```

D5-25

```
10! SAVE "SEMILOG"
20    GINIT
30    GCLEAR
40    GRAPHICS ON
50    ! 10 units X axis, 3 units Y axis
60    WINDOW 0,10,0,3
70    GRID 1,0
80    ! Step up Y axis by LGT units
90    ! You get 3 Y lines every GRID stmt.
100   ! You get 10 X lines every GRID stmt.
110   FOR N=1 TO 9
120   GRID 0,1,0,LGT(N)
130   NEXT N
140   END
```

Solution 31

```
10! SAVE "LOGLOG"
20    GINIT
30    GCLEAR
40    GRAPHICS ON
50    WINDOW 0,4,0,3
60    FOR N=1 TO 9
70    GRID 1,1,LGT(N),LGT(N)
80    NEXT N
90    END
```

05-27

```
1 ! SAVE "LABELS"
10    GCLEAR
20    GINIT
30    GRAPHICS ON
40    WINDOW 0,5.5,0,5.5
50    AXES 1,1
60    MOVE 2,5
70    LABEL "Graph Title"
80    FOR X=0 TO 5
90    MOVE X,0
100   LABEL X
110   NEXT X
120   FOR Y=0 TO 5
130   MOVE 0,Y
140   LABEL Y
150   NEXT Y
160   ALPHA OFF
170   PAUSE
180   END
```

Solution 32

```
1 ! SAVE "LABELS1"
10    GCLEAR
20    GINIT
30    GRAPHICS ON
40    WINDOW -1,5.5,-1,5.5
50    AXES 1,1
60    MOVE 2,5
70    LABEL "Graph Title"
71    LORG 6
80    FOR X=0 TO 5
90    MOVE X,-.1
100   LABEL X
110   NEXT X
111   LORG 8
120   FOR Y=0 TO 5
130   MOVE 0,Y
140   LABEL Y
150   NEXT Y
160   ALPHA OFF
170   PAUSE
180   END
```

```
10! SAVE "PLOT_SIN"
20    ON ERROR GOTO Problem  ! Just in case...
30    OPTION BASE 1
40    DIM Volts(100)
50 !
60 !
70 ! Read from disc file
80    ASSIGN @Pipe TO "SINDAT"
90    ENTER @Pipe;Volts(*)
100   ASSIGN @Pipe TO *
110   CALL Plot_sin(Volts(*),100)
120   PAUSE
130!
140 Problem:  BEEP
150   PRINT "You probably didn't create file SINDAT."
160   PRINT "Run program WRITE_SIN to build the file."
170   STOP
171   END
180   !
190   !
200 SUB Plot_sin(Dat(*),N)
210   GCLEAR     ! Initialize and scale
220   ALPHA OFF
230   GINIT
240   GRAPHICS ON
250   WINDOW -20,110,-12,12
260   !
270   !       Draw axes
280      CLIP 0,110,-10,10
290      AXES 10,1,0,0
300      CLIP OFF
310   !
320   !       Graph label
330   MOVE 40,10
340   LABEL "Sin Wave Data"
350   !
360   !       Label X axis
370   LORG 6
380   FOR X=0 TO N STEP N/5
390      MOVE X,0
400      LABEL X
410   NEXT X
420   !
430   !       Label Y axis
440   LORG 8
450   FOR Y=-10 TO 10 STEP 2
460      MOVE 0,Y
470      LABEL Y
480   NEXT Y
490   !
500   !       Draw N data points
510   MOVE 0,0
520   FOR I=1 TO N
530      DRAW I,Dat(I)
540   NEXT I
550   !
560   SUBEXIT
570 SUBEND
```

```
10! SAVE "ROTATE"
20    ! Set up the plotting space
30    GINIT
40    DEG
50    GRAPHICS ON
60    ALPHA OFF
70    A=SIN(60)*20
80    SHOW 0,50,0,65
90    !
100   MOVE 15,25
110   IMOVE 10,A/2
120   Pen=1
130   !
140   !
150   LOOP   ! Forever
160     Angle=Angle+5
170     PIVOT Angle
180     CALL Triangle(Pen,A)  ! Draw
190   !
200     Pen=-Pen
210     CALL Triangle(Pen,A)  ! Erase
220     Pen=-Pen
230   END LOOP
240   !
250   END
260   !
270   !
280   SUB Triangle(I,A)
290     PEN I
300     X=A/2.5  ! Approximately the center...
310   !
320     IMOVE -10,-X ! Move to angle A
330     IDRAW 20,0   ! Draw to angle B
340     IDRAW -10,A  ! Draw to angle C
350     IDRAW -10,-A ! Draw to angle A
360     IMOVE 10,X   ! Move back to center
370   SUBEND
```

```
10! SAVE "PIE_CHART"
20    COM Cur_deg,X,Y,Radius
30    Cur_deg=0
40    X=65  ! X axis location of circle center.
50    Y=50  ! Y axis location of circle center.
60    ON ERROR GOTO Done
70    GRAPHICS ON
80    GINIT
90    GCLEAR
100   DEG
110   !
120   !
130   ! Set Circle radius
140   DATA 30
150   READ Radius
160   !
170   ! Percent (total=100)
180   DATA 25,15,10,20,10,20
190   !
200   LOOP
210       READ Percent
220       CALL Circle(Percent)
230   END LOOP
240 Done: END
250   !
260   !
270   !
280 SUB Circle(P)
290   COM Cur_deg,X,Y,R
300   MOVE X,Y
310   !
320   FOR I=Cur_deg TO Cur_deg+(P*360/100)
330      DRAW X+R*COS(I),Y+R*SIN(I)
340   NEXT I
350   !
360   !  Following is an attempt to label graph.
370      Z=I-(Cur_deg+(P*360/100)-Cur_deg)/2
380      MOVE X+R*1.2*COS(Z),Y+R*1.2*SIN(Z)
390      SELECT Z      ! Change LORG according to
400                    ! current quadrant.
410         CASE 90 TO 180
420           LORG 7
430         CASE 180 TO 270
440           LORG 9
450         CASE 270 TO 360
460           LORG 3
470         CASE ELSE
480           LORG 1
490      END SELECT
500      LABEL USING "DD,A";P,"%"
510   !
520   Cur_deg=I-1   ! Remember I is one greater!
530   !
540 SUBEND
```

```
10! SAVE "BAR_METER"
20    GINIT
30    GRAPHICS ON
40    VIEWPORT 17,117,25,75
50    FRAME
60    SHOW -1,5,-1,2
70    CLIP 0,4,0,.1
80    AXES .1,0,0,0,10,0,5
90    CLIP OFF
100   !
110   !
120   FOR J=0. TO 4.
130     PEN -1
140     DRAW J-.1,-.4
150     PEN 1
160     LABEL VAL$(J)
170   NEXT J
180   MOVE 1,-.7
190   LABEL "Voltage"
200   !
210   ON KNOB .01 GOSUB Bar
220   Xold=0
230   Xnew=0
240   MOVE 0,.2
250   CLIP ON
260   CLIP 0,4,0,1
270 Main:  GOTO Main
280   !
290   !
300 Bar:    ! Service Knob interrupts and update
310         ! the display.
320         !
330    Xold=Xnew        ! Save the old pointer
340    Xnew=Xnew+KNOBX*.01   ! Make a new one
350    Sign=SGN(Xnew-Xold)
360      PEN Sign
370    IF NOT Sign THEN Sign=1
380    !
390    !
400    ! Beep for overrange and underrange.
410    SELECT Xnew
420       CASE >4
430         BEEP 1500,.2
440         Xnew=4
450       CASE <0
460         BEEP 150,.2
470         Xnew=0
480    END SELECT
490    !
500    !
510    ! Now draw a "bar" of vertical lines from
520    ! the old position to the new one.
530    FOR I=Xold TO Xnew STEP (Sign*.02)
540      MOVE I,.2
550      DRAW I,.6
560    NEXT I
570    !
580    !
590    RETURN
600   !
610   !
620    END
```


Part No. 11141-70381

Printed in U.S.A.
January 1982