HEWLETT PACKARD

HP 3000 SERIES II

SPL

&

FILE SYSTEM

INTRODUCTION

STUDENT WORKBOOK

22804 - 90001

SPL

HEWLETT hp PACKARD

SPL/3000

SYSTEMS PROGRAMMING LANGUAGE

FOR

HP3000

SPL DEFINITION

SPL IS A PROCEDURE-ORIENTED SYSTEMS PROGRAMMING LANGUAGE DESIGNED FOR THE IMPLEMENTATION OF ALL STANDARD HP SYSTEM 3000 SOFTWARE.

SPL CHARACTERISTICS

- PROCEDURE ORIENTED ..... MODULAR
- HIGH-LEVEL SYNTAX ..... PRODUCTIVE
- TRANSPARENT ..... PREDICTABLE
- ASSEMBLY-LEVEL SYNTAX ..... FLEXIBLE
- MACHINE DEPENDENT ..... INTERFACEABLE

INTRODUCTORY EXAMPLES

THE FOLLOWING SLIDES PRESENT A SET OF 6 EXAMPLE PROGRAMS WRITTEN IN SPL.

THEY ARE DESIGNED TO GIVE THE STUDENT A GENERAL OVERVIEW OF THE CONCEPTS, STRUCTURE, AND OPERATION OF SPL PROGRAMS.

PLEASE NOTE THAT THE EXAMPLES ARE RELATED AND THAT EACH EXAMPLE EXPANDS ON THE IDEAS PRESENTED IN THE PREVIOUS ONES.

EXAMPLE 1

WHAT IT DOES

- PRINTS A MESSAGE
- READS SOME INPUT
- ECHOS BACK INPUT

WHAT IT SHOWS

- COMPLETE SPL PROGRAM
- DATA DECLARATIONS
- ELEMENTARY UTILITY ROUTINES

```
PAGE 0001   HEWLETT-PACKARD 32100A.06.2  SPL  TUE, SEP  7, 1976, 12:07
```

EXAMPLE 1

```
00001000  00000 0   $CONTROL USLINIT
00002000  00000 0   BEGIN
00003000  00000 1
00004000  00000 1   ARRAY BUFFER(0:35):="ENTER NAME: ";
00005000  00006 1   INTEGER LEN;
00006000  00006 1
00007000  00006 1   INTRINSIC PRINT,READ;
00008000  00006 1
00009000  00006 1   PRINT(BUFFER,-12,%320);      (1)
00010000  00004 1   LEN:=READ(BUFFER,-30);        (2)
00011000  00011 1
00012000  00011 1   IF LEN=0 THEN RETURN;         (3)
00013000  00015 1   PRINT(BUFFER,-LEN,0);         (4)
00014000  00021 1
00015000  00021 1   END.
 PRIMARY DB STORAGE=%002;    SECONDARY DB STORAGE=%00044
 NO. ERRORS=0000;            NO. WARNINGS=0000
 PROCESSOR TIME=0:00:01;     ELAPSED TIME=0:00:07
```

(1) *Ask for an ASCII string*

(2) *Read input*

(3) *If no input, stop the program*

(4) *If input, echo back string*

EXAMPLE 2

WHAT IT DOES

- REQUESTS AND INPUTS A DATA FILE NAME
- TRIES TO OPEN THE FILE
- IF FILE OPENED, PRINTS A MESSAGE THAT THE OPEN WORKED

WHAT IT SHOWS

- USE OF FOPEN TO OPEN A FILE
- CHECK OF CONDITION CODE RETURNED BY FOPEN
- HOW TO PRINT A "TOMBSTONE" AND ABORT A PROGRAM
- ARRAY EQUIVALENCING

EXAMPLE 2

```
00001000  00000 0   $CONTROL USLINIT
00002000  00000 0   BEGIN
00003000  00000 1
00004000  00000 1   ARRAY BUFFER(0:35):="ENTER FILE NAME: ";
00005000  00011 1   INTEGER LEN,FILENO;
00006000  00011 1
00007000  00011 1   BYTE ARRAY FILENAME(*)=BUFFER;
00008000  00011 1
00009000  00011 1   INTRINSIC PRINT,READ,FOPEN,PRINT'FILE'INFO,QUIT;
00010000  00011 1
00011000  00011 1   PRINT(BUFFER,-17,%320);
00012000  00004 1   LEN:=READ(BUFFER,-30);
00013000  00011 1
00014000  00011 1   IF LEN=0 THEN RETURN;
00015000  00015 1
00016000  00015 1   FILENAME(LEN):=%15;
00017000  00020 1   FILENO:=FOPEN(FILENAME,1,0);      (1)
00018000  00030 1
00019000  00030 1   IF <                               (2)
00020000  00030 1   THEN BEGIN
00021000  00031 2        PRINT'FILE'INFO(FILENO);
00022000  00033 2        QUIT(1);
00023000  00035 2        END;
00024000  00035 1
00025000  00035 1   MOVE BUFFER:="FILE OPENED OK";     (3)
00026000  00052 1   PRINT(BUFFER,-14,0);
00027000  00056 1
00028000  00056 1   END.
 PRIMARY DB STORAGE=%004;    SECONDARY DB STORAGE=%00044
 NO. ERRORS=0000;            NO. WARNINGS=0000
 PROCESSOR TIME=0:00:01;     ELAPSED TIME=0:00:05
```

(1) *If a filename is input, open the file*

(2) *If an error occurred during the FOPEN, print a "tombstone" and abort the program*

(3) *If file opened OK, notify user*

EXAMPLE 3

WHAT IT DOES

- OPENS THE DATA FILE
- READS ALL RECORDS FROM THE FILE
- PRINTS OUT EACH RECORD TO $STDLIST

WHAT IT SHOWS

- READING FROM A FILE
- CHECKING FOR FILE ERROR AND END-OF-FILE CONDITIONS
- BRANCHING TO A LABEL

```
PAGE 0001   HEWLETT-PACKARD 32100A.06.2  SPL  TUE, SEP  7, 1976, 12:08 PM
```

EXAMPLE 3

```
00001000  00000 0   $CONTROL USLINIT
00002000  00000 0   BEGIN
00003000  00000 1
00004000  00000 1   ARRAY BUFFER(0:35):="ENTER FILE NAME: ";
00005000  00011 1   INTEGER LEN,FILENO;
00006000  00011 1   BYTE ARRAY FILENAME(*)=BUFFER;
00007000  00011 1
00008000  00011 1   INTRINSIC PRINT,READ,FOPEN,PRINT'FILE'INFO,QUIT,FREAD,
00009000  00011 1           FCLOSE;
00010000  00011 1
00011000  00011 1   PRINT(BUFFER,-17,%320);
00012000  00004 1   LEN:=READ(BUFFER,-30);
00013000  00011 1   IF LEN=0 THEN RETURN;
00014000  00015 1   FILENAME(LEN):=%15;
00015000  00020 1   FILENO:=FOPEN(FILENAME,1,0);
00016000  00030 1   IF <
00017000  00030 1   THEN BEGIN
00018000  00031 2        PRINT'FILE'INFO(FILENO);
00019000  00033 2        QUIT(1);
00020000  00035 2        END;
00021000  00035 1
00022000  00035 1   READ'A'RECORD;
00023000  00035 1                LEN:=FREAD(FILENO,BUFFER,-72);  (1)
00024000  00043 1            (2) IF <
00025000  00043 1                THEN BEGIN
00026000  00044 2                     PRINT'FILE'INFO(FILENO);
00027000  00046 2                     QUIT(3);
00028000  00050 2                     END
00029000  00050 1            (3) ELSE IF >
00030000  00052 1                     THEN BEGIN
00031000  00053 2                          MOVE BUFFER:="EOF FOUND";
00032000  00065 2                          PRINT(BUFFER,-9,0);
00033000  00071 2                          FCLOSE(FILENO,0,0);
00034000  00074 2                          RETURN;
00035000  00075 2                          END;
00036000  00075 1            (4) PRINT(BUFFER,-LEN,0);
00037000  00101 1                GO READ'A'RECORD;
00038000  00102 1
00039000  00102 1   END.
 PRIMARY DB STORAGE=%004;    SECONDARY DB STORAGE=%00044
 NO. ERRORS=0000;            NO. WARNINGS=0000
 PROCESSOR TIME=0:00:01;     ELAPSED TIME=0:00:06
```

(1) *Read a record from the file*

(2) *If the read failed because of an error, print a tombstone and abort the program*

(3) *If the end of file is found, notify user, close file, end program*

(4) *If a record is read successful, print it out to $STDLIST and go read another record*

EXAMPLE 4

WHAT IT DOES

- OPENS THE DATA FILE
- OPENS $STDLIST AS A FILE NAMED "OUTFILE"
- READS ALL RECORDS FROM THE DATA FILE
- WRITES THE RECORDS READ TO "OUTFILE" (THIS REPLACES THE PRINT INTRINSIC USED IN EXAMPLE 3.)

WHAT IT SHOWS

- OPENING $STDLIST AS A FILE
- WRITING TO $STDLIST OPENED AS A FILE

EXAMPLE 4

```
00001000  00000 0   $CONTROL USLINIT
00002000  00000 0   BEGIN
00003000  00000 1
00004000  00000 1   ARRAY BUFFER(0:35):="ENTER FILE NAME: ";
00005000  00011 1
00006000  00011 1   INTEGER LEN,FILENO,OUTF;
00007000  00011 1
00008000  00011 1   BYTE ARRAY FILENAME(*)=BUFFER;
00009000  00011 1
00010000  00011 1   INTRINSIC PRINT,READ,FOPEN,PRINT'FILE'INFO,QUIT,FREAD,
00011000  00011 1             FCLOSE,FWRITE;
00012000  00011 1
00013000  00011 1   PRINT(BUFFER,-17,%320);
00014000  00004 1   LEN:=READ(BUFFER,-30);
00015000  00011 1   IF LEN=0 THEN RETURN;
00016000  00015 1   FILENAME(LEN):=%15;
00017000  00020 1   FILENO:=FOPEN(FILENAME,1,0);
00018000  00030 1   IF <
00019000  00030 1   THEN BEGIN
00020000  00031 2        PRINT'FILE'INFO(FILENO);
00021000  00033 2        QUIT(1);
00022000  00035 2        END;
00023000  00035 1
00024000  00035 1   MOVE FILENAME:="OUTFILE ";
00025000  00050 1   OUTF:=FOPEN(FILENAME,%414,1);
00026000  00060 1   IF <
00027000  00060 1   THEN BEGIN
00028000  00061 2        PRINT'FILE'INFO(OUTF);
00029000  00063 2        QUIT(2);
00030000  00065 2        END;
00031000  00065 1
00032000  00065 1   READ'A'RECORD:
00033000  00065 1                 LEN:=FREAD(FILENO,BUFFER,-72);
00034000  00073 1                 IF <
00035000  00073 1                 THEN BEGIN
00036000  00074 2                      PRINT'FILE'INFO(FILENO);
00037000  00076 2                      QUIT(3);
00038000  00100 2                      END
00039000  00100 1                 ELSE IF >
00040000  00103 1                      THEN BEGIN
00041000  00104 2                           MOVE BUFFER:="EOF FOUND";
00042000  00116 2                           PRINT(BUFFER,-9,0);
00043000  00122 2                           FCLOSE(FILENO,0,0);
00044000  00125 2                           RETURN;
00045000  00126 2                           END;
00046000  00126 1
00047000  00126 1                 FWRITE(OUTF,BUFFER,-LEN,0);
00048000  00133 1
00049000  00133 1                 GO READ'A'RECORD;
00050000  00134 1
00051000  00134 1   END.
 PRIMARY DB STORAGE=%005;   SECONDARY DB STORAGE=%00044
 NO. ERRORS=0000;           NO. WARNINGS=0000
 PROCESSOR TIME=0:00:02;    ELAPSED TIME=0:00:06
```

(1) *Open $STDLIST as a file with the formal designator OUTFILE*

(2) *Write record to OUTFILE*

EXAMPLE 5

WHAT IT DOES

- OPENS THE DATA FILE
- OPENS $STDLIST AS A FILE NAMED "OUTFILE"
- READS ALL RECORDS FROM THE DATA FILE
- COUNTS THE NUMBER OF RECORDS READ
- WRITES THE RECORDS READ TO "OUTFILE"
- PRINTS THE NUMBER OF RECORDS READ WHEN THE END OF THE DATA FILE IS FOUND

WHAT IT SHOWS

- INITIALIZATION OF VARIABLES
- USE OF A POINTER
- CONVERTING A BINARY NUMBER TO ASCII REPRESENTATION

EXAMPLE 5

```
00001000  00000 0   $CONTROL USLINIT
00002000  00000 0   BEGIN
00003000  00000 1
00004000  00000 1   ARRAY BUFFER(0:35):="ENTER FILE NAME: ";
00005000  00011 1
00006000  00011 1   INTEGER LEN,FILENO,OUTF,NREC:=0;
00007000  00011 1
00008000  00011 1   BYTE ARRAY FILENAME(*)=BUFFER;
00009000  00011 1
00010000  00011 1   BYTE POINTER BUF:=@FILENAME;
00011000  00011 1
00012000  00011 1   INTRINSIC PRINT,READ,FOPEN,PRINT'FILE'INFO,QUIT,FREAD,
00013000  00011 1             FCLOSE,FWRITE,ASCII;
00014000  00011 1
00015000  00011 1   PRINT(BUFFER,-17,%320);
00016000  00004 1   LEN:=READ(BUFFER,-30);
00017000  00011 1   IF LEN=0 THEN RETURN;
00018000  00015 1   FILENAME(LEN):=%15;
00019000  00020 1   FILENO:=FOPEN(FILENAME,1,0);
00020000  00030 1   IF <
00021000  00030 1   THEN BEGIN
00022000  00031 2        PRINT'FILE'INFO(FILENO);
00023000  00033 2        QUIT(1);
00024000  00035 2        END;
00025000  00035 1   MOVE FILENAME:="OUTFILE ";
00026000  00050 1   OUTF:=FOPEN(FILENAME,%414,1);
00027000  00060 1   IF <
00028000  00060 1   THEN BEGIN
00029000  00061 2        PRINT'FILE'INFO(OUTF);
00030000  00063 2        QUIT(2);
00031000  00065 2        END;
00032000  00065 1   READ'A'RECORD:
00033000  00065 1                 LEN:=FREAD(FILENO,BUFFER,-72);
00034000  00073 1                 IF <
00035000  00073 1                 THEN BEGIN
00036000  00074 2                      PRINT'FILE'INFO(FILENO);
00037000  00076 2                      QUIT(3);
00038000  00100 2                      END
00039000  00100 1                 ELSE IF >
00040000  00103 1                      THEN BEGIN
00041000  00104 2
00042000  00104 2                           MOVE BUFFER:="EOF FOUND AFTER ";
00043000  00122 2                           LEN:=ASCII(NREC,10,BUF(16));
00044000  00131 2                           MOVE BUF(16+LEN):=" RECORDS";
00045000  00146 2                           PRINT(BUFFER,-24-LEN,0);
00046000  00153 2
00047000  00153 2                           FCLOSE(FILENO,0,0);
00048000  00156 2                           RETURN;
00049000  00157 2                           END;
00050000  00157 1
00051000  00157 1                 NREC:=NREC+1;
00052000  00160 1
00053000  00160 1                 FWRITE(OUTF,BUFFER,-LEN,0);
00054000  00165 1                 GO READ'A'RECORD;
00055000  00166 1
00056000  00166 1   END.
 PRIMARY DB STORAGE=%007;   SECONDARY DB STORAGE=%00044
```

(1) *When EOF is found, report the number of records read*

(2) *Count number of records read*

EXAMPLE 6

WHAT IT DOES

EXAMPLE 6 PERFORMS EXACTLY THE SAME OPERATIONS AS EXAMPLE 5 BUT HAS BEEN RESTRUCTURED TO ILLUSTRATE DIFFERENT SPL STRUCTURES.

WHAT IT SHOWS

- PROCEDURE TO HANDLE ERROR CONDITIONS
- FUNCTION PROCEDURE TO READ, COUNT, AND WRITE RECORDS
- LOCAL DECLARATION
- VALUE DECLARATION
- BYTE ADDRESS TO WORD ADDRESS CONVERSION
- ASSIGNMENT WITHIN A CONDITIONAL EXPRESSION

EXAMPLE 6

```
00001000  00000 0   $CONTROL USLINIT
00002000  00000 0   BEGIN
00003000  00000 1
00004000  00000 1   ARRAY BUFFER(0:35):="ENTER FILE NAME: ";
00005000  00011 1   INTEGER LEN,FILENO,OUTF;
00006000  00011 1   BYTE ARRAY FILENAME(*)=BUFFER;
00007000  00011 1   BYTE POINTER BUF:=@FILENAME;
00008000  00011 1
00009000  00011 1   INTRINSIC PRINT,READ,FOPEN,PRINT'FILE'INFO,QUIT,
00010000  00011 1             FREAD,FCLOSE,FWRITE,ASCII;
00011000  00011 1
00012000  00011 1   PROCEDURE PRINT'ERROR(FNO,QNO);
00013000  00000 1   VALUE FNO,QNO; INTEGER FNO,QNO;
00014000  00000 1   BEGIN
00015000  00000 2     PRINT'FILE'INFO(FNO);
00016000  00002 2     QUIT(QNO);
00017000  00004 2   END;
00018000  00000 1
00019000  00000 1   INTEGER PROCEDURE RECORD'COUNT;
00020000  00000 1   BEGIN
00021000  00000 2     INTEGER NREC:=0;
00022000  00000 2     READ'A'RECORD;
00023000  00001 2     LEN:=FREAD(FILENO,BUFFER,-72);
00024000  00007 2     IF < THEN PRINT'ERROR(FILENO,3)
00025000  00012 2          ELSE IF >
00026000  00014 2               THEN BEGIN
00027000  00015 3                    RECORD'COUNT:=NREC;
00028000  00017 3                    RETURN;
00029000  00020 3                    END;
00030000  00020 2     NREC:=NREC+1;
00031000  00021 2     FWRITE(OUTF,BUFFER,-LEN,0);
00032000  00026 2     GO READ'A'RECORD;
00033000  00027 2   END;
00034000  00000 1
00035000  00000 1   PRINT(BUFFER,-17,%320);
00036000  00004 1   IF (LEN:=READ(BUFFER,-30)) = 0 THEN RETURN;
00037000  00015 1
00038000  00015 1   FILENAME(LEN):=%15;
00039000  00020 1   FILENO:=FOPEN(FILENAME,1,0);
00040000  00030 1   IF < THEN PRINT'ERROR(FILENO,1);
00041000  00034 1
00042000  00034 1   MOVE FILENAME:="OUTFILE ";
00043000  00047 1   OUTF:=FOPEN(FILENAME,%414,1);
00044000  00057 1   IF < THEN PRINT'ERROR(OUTF,2);
00045000  00063 1
00046000  00063 1   LEN:=ASCII(RECORD'COUNT,10,BUF(16));
00047000  00072 1
00048000  00072 1   MOVE BUF:="EOF FOUND AFTER ";
00049000  00112 1   MOVE BUF(16+LEN):=" RECORDS";
00050000  00127 1   PRINT(BUF,-24-LEN,0);

***** WARNING *****  ARITHMETIC RIGHT SHIFT EMITTED

00051000  00135 1   FCLOSE(FILENO,0,0);
00052000  00140 1
00053000  00140 1   END.
 PRIMARY DB STORAGE=%006;   SECONDARY DB STORAGE=%00044
```

(1) *Error Routine*

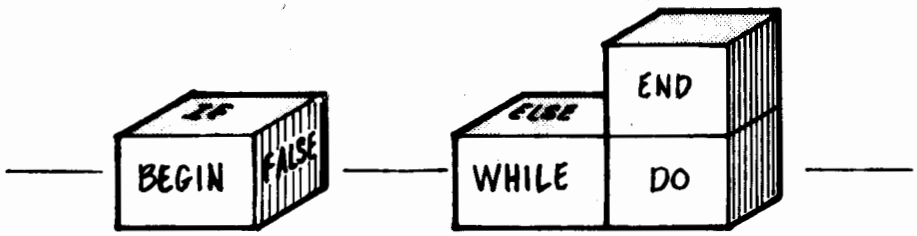
(2) *Function Procedure*

(3) *Function returns the number of records read*

(4) *Byte address converted to a word address*

SPL

PROGRAM STRUCTURE and LANGUAGE CONSTRUCTIONS



PROGRAM =

BEGIN

DATA DECLARATIONS

PROCEDURE DECLARATIONS

STATEMENTS

END

- FREE - FORM

- **IDENTIFIERS**

  A, B23, DATA, GET'CARD

- **DECLARATIONS**

  INTEGER I, J, B9;

  REAL I', J', B92;

- **RESERVED WORDS**

  BEGIN, ELSE, GOTO, FOR

- **STATEMENTS**

  A:=0; GOTO L; IF A< B THEN.

• BEGIN – END

1. ENCLOSE PROGRAM
2. COMPOUND STATEMENT

• SEMI-COLON

1. TERMINATES EVERY DECLARATION
2. TERMINATES EVERY STATEMENT EXCEPT THOSE FOLLOWED BY **END**, **ELSE** or **UNTIL**

COMMENTS

COMMENT

ANY NUMBER OF LINES OF DATA ENDED BY ;

<< ANY NUMBER OF LINES ENDED BY >>

CONSTANTS

- **INTEGER**

  DECIMAL 125, –93

  OCTAL %37, %77

- **DOUBLE**

  DECIMAL 125D, –93D

  OCTAL %37D, %77D

- **REAL**

  DECIMAL 1.25, –93.6

  1.25E2

- **LONG**

  DECIMAL 1.25L0, –93.6L0

  1.25L2

VARIABLES

- **DECLARATION**

  TYPE IDENTIFIER,. . . ;

- **INITIAL VALUES**

  TYPE IDENTIFIER := CONSTANT;

ASSIGNMENT STATEMENTS

VARIABLE := EXPRESSION ;

ASSIGNMENT OPERATOR

A := 5 + B;

REPLACES VARIABLE WITH THE VALUE OF AN EXPRESSION.

MULTIPLE ASSIGNMENT ALLOWED.

TYPE CONVERSION

REAL (INTEGER EXPRESSION)

INTEGER(FIXR (REAL EXPRESSION))

INTEGER(FIXT (REAL EXPRESSION))

ARITHMETIC EXPRESSIONS

OPERATORS

+ – * / MOD ∧

PRECEDENCE

∧
* / MOD
+ –

A + B * C

PARENTHESES

(A + B) * C

ABSOLUTE VALUE

\ A – B \

NO TYPE MIXING

LABELS

LABEL:

L: A:=Ø;

GOTO STATEMENT

GOTO L;

{ GO
GOTO
GO TO } LABEL;

LOGIC EXPRESSIONS

ARITH–EXPR	REL	ARITH–EXPR
	=	
	<	
	>	
	< =	
	> =	
	< >	

- A < 10

  B > = C

  D < > E

LOGICAL CONSTANTS

TRUE (-1) FALSE (0) INTEGERS

LOGICAL VARIABLES

```
LOGICAL L:= TRUE;

L:= A < 3;
```

LOGICAL OPERATORS

NOT LAND XOR LOR

IF STATEMENTS

IF CONDITION THEN STATEMENT;

IF CONDITION THEN STATEMENT

ELSE STATEMENT;

CONDITIONS

LOGICAL EXPRESSIONS COMBINED

BY NEW OPERATORS:

OR

AND

NOT THE SAME AS LOR, LAND

SIDE - EFFECTS

IF EXPRESSIONS

IF CONDITION THEN EXPRESSION

ELSE EXPRESSION

SPL WORKSESSION 1

WRITE A SINGLE ASSIGNMENT STATEMENT WHICH SETS THE VALUE OF THE INTEGER VARIABLE "LIMIT" BASED ON THE VALUE OF THE INTEGER VARIABLE "NUMBER".

IF THE VALUE OF "NUMBER" EXCEEDS 5, SET "LIMIT" TO 5.

IF THE VALUE OF " NUMBER" IS LESS THAN OR EQUAL TO 5, SET "LIMIT" EQUAL TO "NUMBER".

ARRAYS

TYPE		NAME	LOWER	UPPER
INTEGER	ARRAY	A	(0	: 10);

BOUNDS

INTEGER ARRAY B(1:10),C(5:10),D(-5:5);

ARRAY E(-100:-50);

INITIALIZATION

```
INTEGER     ARRAY     A(0:10):=

        1, 2, 3, 4, 5, 6, 7;

INTEGER     ARRAY     A(0:10):=

        "ABCDEF";

INTEGER     ARRAY     A(0:10):= 1, 2, 3, B(0:10); BAD
```

ACCESSING ELEMENT

```
INTEGER ARRAY A(0:10);

            A(1)

            A(22)

            A

            A(5) + A(6) * 2
A(2):= A(5)/2;
A(I:= I+1):= 0;
A(I + 1):=1;
```

BYTE ARRAYS

BYTE = 8 BITS = 1/2 WORD = CHARACTER

```
BYTE ARRAY DATA (0:20);

BYTE ARRAY BUF (1:80):=

80(%40);
```

DO UNTIL STATEMENT

```
DO
      STATEMENT
UNTIL
      CONDITION;
```

```
BYTE ARRAY STRING(0:71);
INTEGER SUB:= -1;

      DO SUB:= SUB + 1 UNTIL STRING (SUB) <>"0";
```

WHILE DO STATEMENT


```
WHILE CONDITION
 DO STATEMENT;


 INTEGER ARRAY     A(0:10);
 INTEGER SUB:=0;


      WHILE SUB < =  10
      DO BEGIN   A(SUB) := SUB ∧ 2;
                 SUB:= SUB + 1;
      END;
```

SPL WORKSESSION 2

WRITE THE SPL STATEMENTS TO DECLARE AN INTEGER ARRAY "DATA" OF 100 ELEMENTS AND SEARCH THE ARRAY TO FIND THE FIRST ELEMENT EQUAL TO ZERO. SET A LOGICAL VARIABLE NONE TO TRUE IF NONE WERE ZERO, FALSE IF ONE OR MORE WAS ZERO. (ASSUME THAT THE DATA ARRAY CONTAINS VALID INFORMATION. IN ACTUAL FACT, ITS CONTENTS ARE UNINITIALIZED.)

MAKE YOUR SOLUTION A COMPLETE PROGRAM

```
EQUATE          A = 2, B = A + 1;

INTEGER ARRAY DATA (0:A);
```

```
DEFINE I = A, B, C, D, E#;   <---  declaration

INTEGER I;                   <---  invocation
   |
   v
INTEGER A, B, C, D, E;       <---  effect
```

SWITCH

- DECLARATION

```
SWITCH SW:= L1, L2, L3;
```

- INVOCATION

```
GOTO SW(N);
```

<< SW(0)= L1, SW(1)= L2, . . . >>

CASE STATEMENT

```
CASE INDEX OF COMPOUND STATEMENT;

    CASE  I OF

    BEGIN  A:= 0;

           B:= A;

           IF G = 23 THEN GOTO L;

           GO TO M'

    END;
```

PROCEDURES

- BASIC SUB-PROGRAM OF SPL
- DECLARED IN GLOBAL HEAD
- INVOKED BY PROCEDURE CALL STATEMENT
- PARAMETERS
- LOCAL VARIABLES
- FUNCTIONS

(PROCEDURES ARE THE EQUIVALENT OF A SUBROUTINE IN FORTRAN OR A SECTION IN COBOL.)

PROCEDURE DECLARATION

PROCEDURE HEAD

TYPE **PROCEDURE** NAME (FORMAL PARAMETER LIST);

VALUE VALUE PARAMETER LIST;

SPECIFICATION PART;

PROCEDURE BODY

BEGIN

LOCAL DECLARATIONS;

STATEMENTS;

END;

LOCAL DECLARATIONS

```
PROCEDURE P;

BEGIN
    INTEGER A:=0;
    ARRAY B(0:10);
    INTRINSIC PRINT;
    _________
    _________
    _________
END;
```

LOCAL DECLARATION

STATEMENTS

PROCEDURE BODY

PROCEDURE CALL

```
PROCEDURE N(A):
    VALUE A; INTEGER A;
    BEGIN
                                  DECLARATION
        STATEMENT;
        STATEMENT;
        STATEMENT;
    END;

N(5);
N(B);
N(B:=B+1);                        CALLS
N(B*5+OUT);
N(0);
```

PARAMETERS

```
PROCEDURE N(A, B, C, D, E);
    VALUE A;
    INTEGER A;
    REAL B;
    PROCEDURE C;
    LABEL D;
    ARRAY E;
```

FUNCTION PROCEDURES

RETURNS A RESULT IN PLACE OF ITS NAME

<< DECLARATION >>

```
INTEGER NUM:=108,NIX;
INTEGER PROCEDURE VAL (A,B,C);
      VALUE A,B,C;
      INTEGER A,B,C;
            VAL:=(A+B)*C;
```

<< INVOCATION >>

```
NIX := NUM/VAL(4,5,6);
```

SPL WORKSESSION 3

WRITE A PROCEDURE WITH TWO PARAMETERS: A,B.

RETURN THE SUM OF **A** AND **B** **(A + B)** IN **A.**

RETURN THE PRODUCT OF **A** AND **B** **(A x B)** IN **B.**

USE INTEGER VARIABLES FOR BOTH **A** AND **B.**

IGNORE THE POSSIBILITY OF INTEGER OVERFLOW DURING ARITHMETIC OPERATIONS.

BINARY INTRINSIC

VARIABLE := **BINARY** (ARRAY, LENGTH);

CONVERTED RESULT	STRING OF CHARACTERS	NUMBER OF CHARACTERS

OCTAL IF STARTS WITH %

%137

DECIMAL OTHERWISE

- INTEGERS ONLY.

ASCII INTRINSIC

VARIABLE := **ASCII** (WORD, BASE, ARRAY);

VALUE TO CONVERT

RESULT

8, 10 OR -10

(DECIMAL, NUMBER OF CHARACTERS)

(OCTAL, NUMBER OF SIGN, DIGITS)

PLACE TO PUT CHARACTERS

OCTAL ALWAYS PRODUCES 6 DIGITS

PRINT INTRINSIC

```
PRINT     (ARRAY, LENGTH, CONTROL);

     LENGTH   = + WORDS
                - BYTES

     CONTROL = Ø FOR CR,LF
               "+" FOR NO LF
               %320 FOR NO CR, NO LF

PRINT (DATA, -72,Ø);
```

READ INTRINSIC

VARIABLE:= **READ** (ARRAY, MAXLGTH);

MAXLGTH = + WORDS

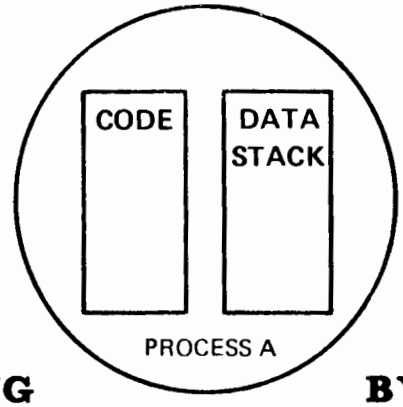
- BYTES

VARIABLE = ACTUAL LGTH (ALWAYS+)

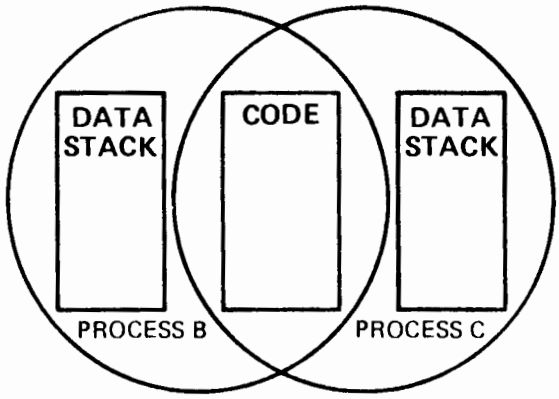
A := **READ** (DATA, −72);

WARNING MESSAGES

```
BYTE ARRAY B(0:10);

PRINT (B,-5,0);

*WARNING* ARITHMETIC RIGHT SHIFT EMITTED
```

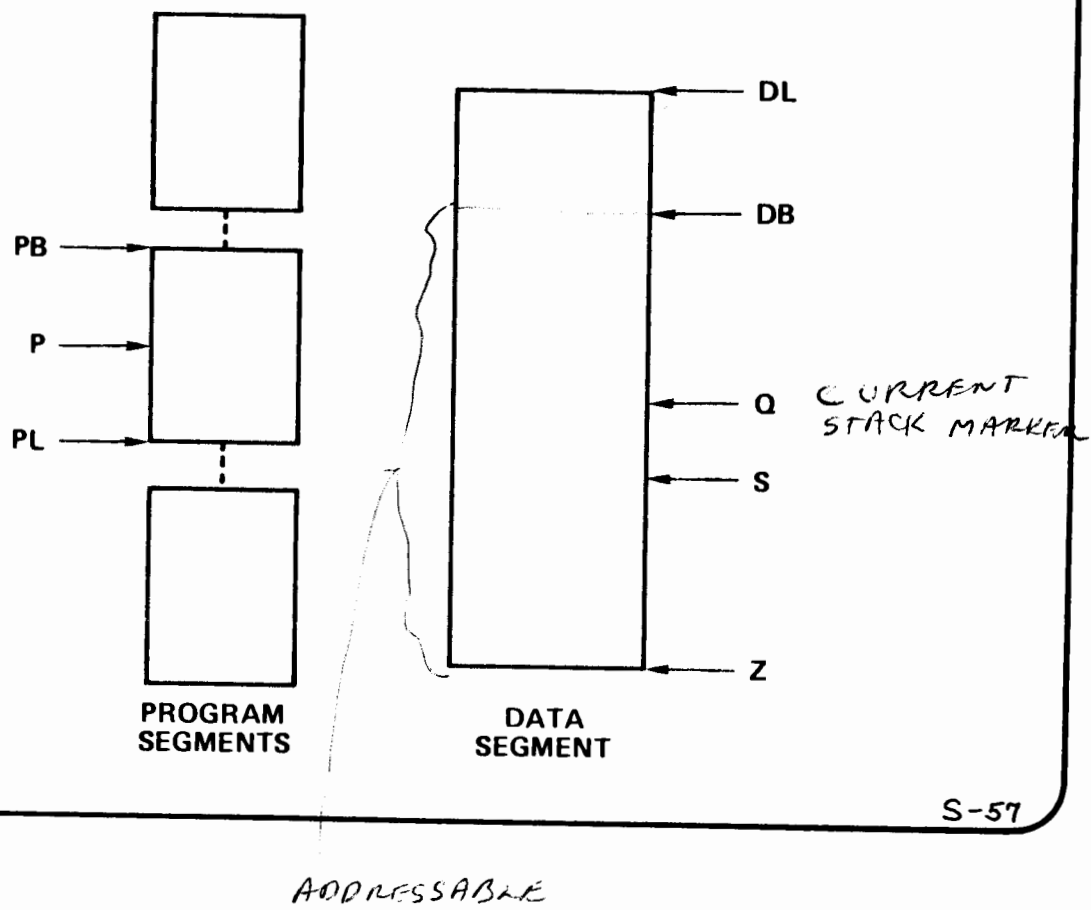


CODE SHARING BY PROCESSES



HEWLETT hp PACKARD

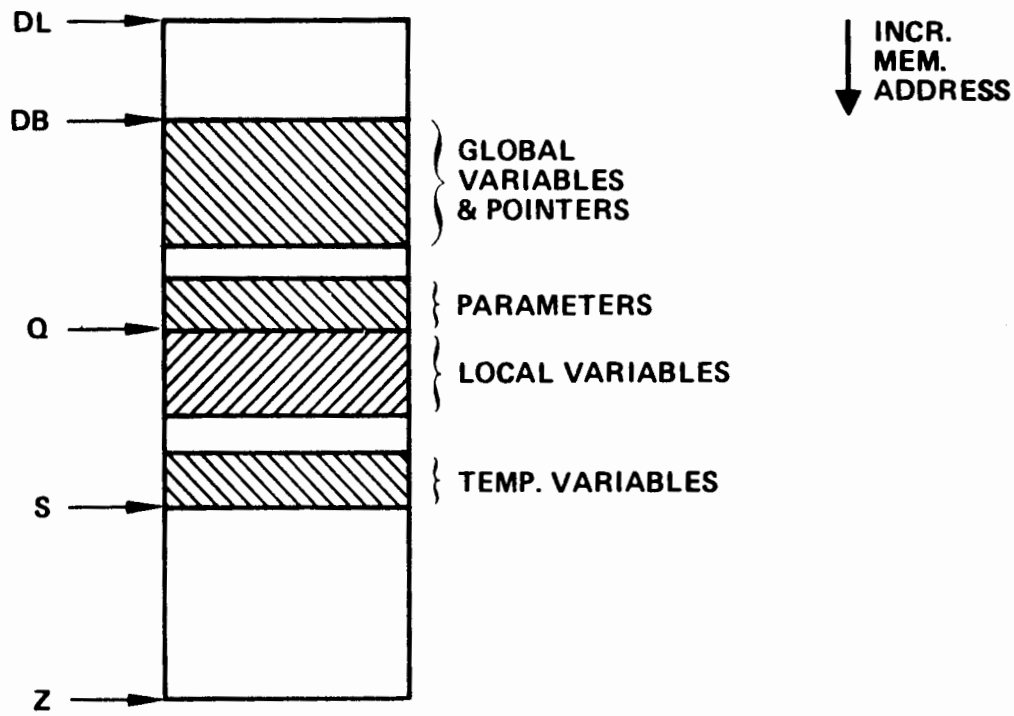
PROCESS UNIVERSE



S-57

ADDRESSING IS NOT PB RELATIVE, USE AN ADDRESS BIT THUS ± 63 (NOT 255) FOR Q, S addressing.

DATA STACK


DL
DB
Q
S
Z
GLOBAL VARIABLES & POINTERS
PARAMETERS
LOCAL VARIABLES
TEMP. VARIABLES
INCR. MEM. ADDRESS

$CONTROL USLINIT, ADR, CODE, MAP

```
PAGE 0001   HEWLETT-PACKARD 32100A.06.2  SPL  FRI, SEP 10, 1976,  1:45 PM  (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1976

00001000  00000 0   $CONTROL USLINIT,ADR,CODE,MAP
00002000  00000 0   BEGIN INTEGER A,B;
                          DB+000
                          DB+001
00003000  00000 1         REAL R;
                            DB+002
00004000  00000 1   L:    R:=REAL(A+B);
00005000  00004 1         A:=A+B*(12/A);
00006000  00012 1         R:=R+1.0;
00007000  00023 1         IF R>100.0 THEN GO TO L;
00008000  00027 1   END.
```

ADDR

SOURCE STATEMENTS

```
 00000  041000 071001 004700 161002 021014 041000 002340 111001   00010  071000 051000 151002 034002 140005 040000 000000 040644
 00020  000000 005100 161002 151002 034405 005000 141466 000000
```

ADDRESS

INSTRUCTION

IDENTIFIER	CLASS	TYPE	ADDRESS
A	SIMP. VAR.	INTEGER	DB+000
B	SIMP. VAR.	INTEGER	DB+001
L	LABEL		PB+000
R	SIMP. VAR.	REAL	DB+002
TERMINATE*	PROCEDURE		

MAP

```
PRIMARY DB STORAGE=%004;    SECONDARY DB STORAGE=%00000
NO. ERRORS=0000;            NO. WARNINGS=0000
PROCESSOR TIME=0:00:00;     ELAPSED TIME=0:00:04
```

$CONTROL USLINIT, ADR, INNERLIST

```
PAGE 0001   HEWLETT-PACKARD 32100A.06.2  SPL  FRI, SEP 10, 1976,  1:46 PM

00001000  00000 0   $CONTROL USLINIT,ADR,INNERLIST
00002000  00000 0   BEGIN INTEGER A,B;
                            DB+000
                            DB+001
00003000  00000 1         REAL R;
                            DB+002
00004000  00000 1   L;    R:=REAL(A+B);
                                        00000    LOAD  DB 000      041000    02.28
                                        00001    ADDM  DB 001      071001    02.63
                                        00002    FLT , NOP         004700    06.65
                                        00003    STD   DB 002      161002    04.03
00005000  00004 1         A:=A+B*(12/A);
                                        00004    LDI ,014          021014    01.05
                                        00005    LOAD  DB 000      041000    02.28
                                        00006    DIV , DEL         002340    09.45
                                        00007    MPYM  DB 001      111001    08.23
                                        00010    ADDM  DB 000      071000    02.63
                                        00011    STOR  DB 000      051000    02.63
00006000  00012 1         R:=R+1.0;
                                        00012    LDD   DB 002      151002    03.85
                                        00013    LDPP,000          034000    03.68
                                        00021    FADD, NOP         005100    13.90
                                        00022    STD   DB 002      161002    04.03
00007000  00023 1         IF R>100.0 THEN GO TO L;
                                        00023    LDD   DB 002      151002    03.85
                                        00024    LDPP,000          034000    03.68
                                        00025    FCMP, NOP         005000    04.70
                                        00026    BLE   P+ 000      141300    03.50
                                        00027    BR    P+ 000      140000    03.50
                                        00026    INSERT OR FIXUP   140000
00008000  00027 1   END.
                                        00027    PCAL,052          000000    25.00
 PRIMARY DB STORAGE=%004;    SECONDARY DB STORAGE=%00000
 NO. ERRORS=0000;            NO. WARNINGS=0000
 PROCESSOR TIME=0:00:01;     ELAPSED TIME=0:00:04
```

PAGE NUMBER

SEQUENCE FIELD

P COUNTER

BEGIN/END COUNTER

INNERLIST

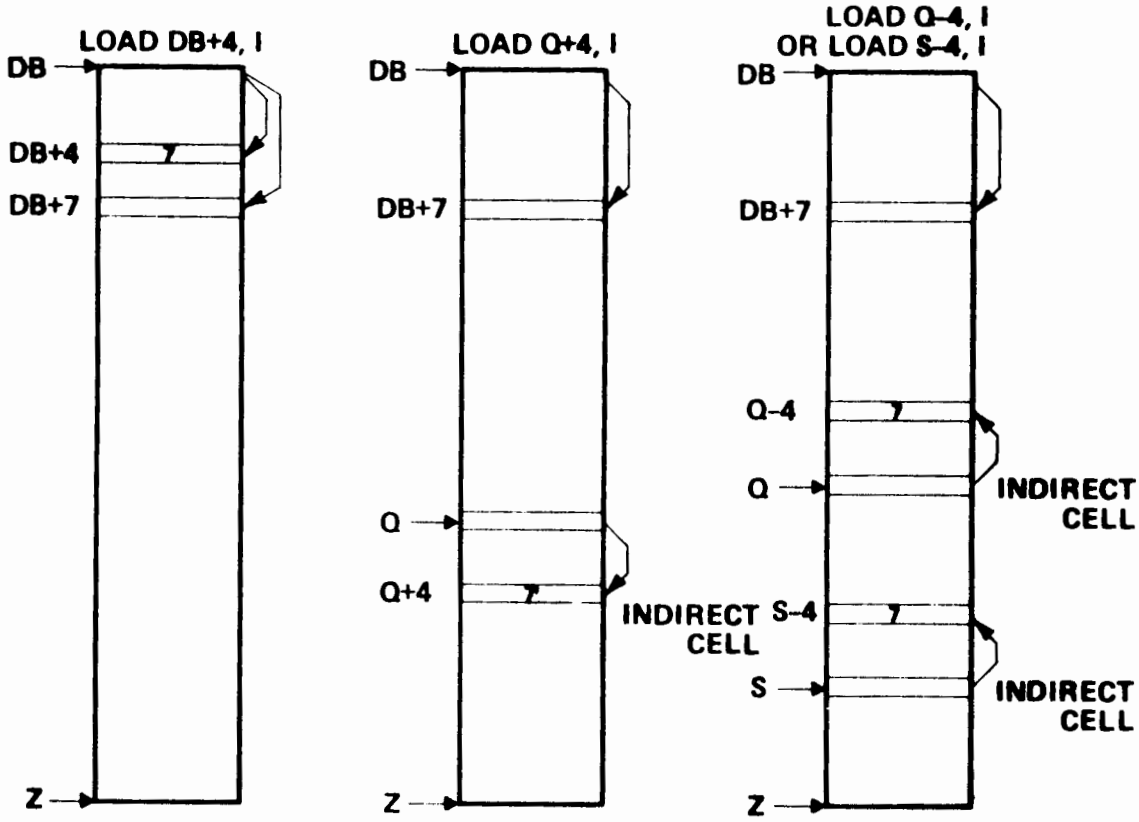
~TIME

1 MEMORY CYCLE PER INSTRUCTION

LOAD FROM BB nelab to STACK

ONLY 255 WORDS

EXAMPLES OF INDIRECT ADDRESSING

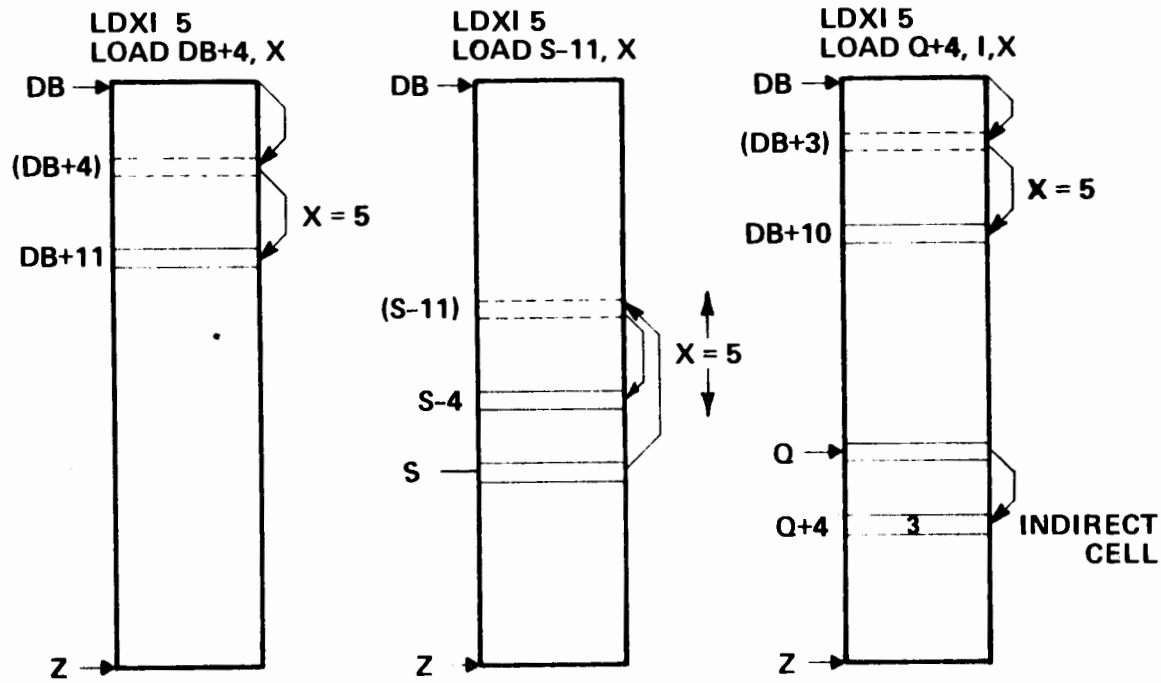


S-61

ALL INDIRECT ADDRESSING IS DB RELATIVE

EXAMPLES OF INDEXING

DATA, INDEXED

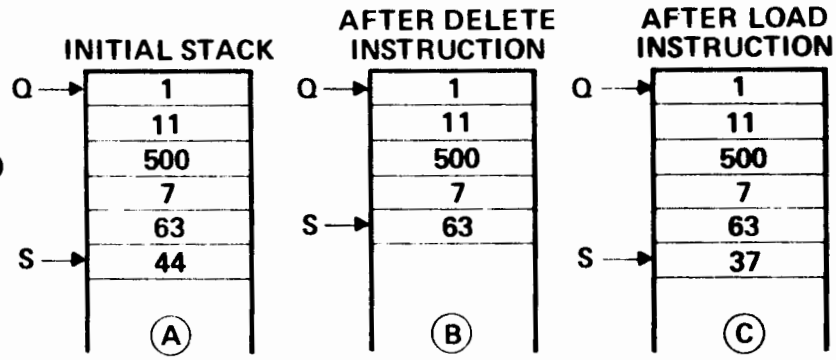


NOTE: ADDRESS CALCULATIONS IN OCTAL

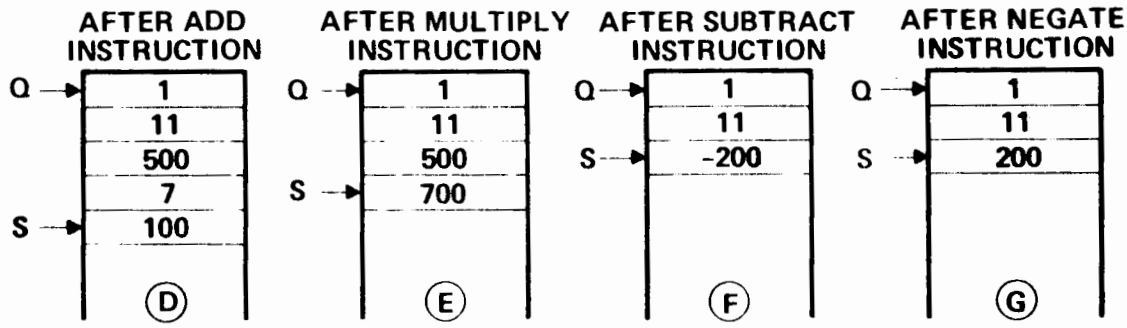
S-62

FASTER TO MOVE WORD THAN BYTE
SINCE FOR BYTE HAS TO MOVE MASKED WORD

DELETE/LOAD



ADD/MULTIPLY/SUBTRACT/NEGATE



SPL WORKSESSION 4

FILL IN THE STACK AFTER EACH OPCODE IS EXECUTED:

ASSEMBLE (LDI 3;	3	
LDI 5;	5 3	
ADDI 2;	7 5	
LDNI 10;	-10 7 5	UPSIDE DOWN
MPY, XCH);	3 -70	

DATA CONSTRUCTS

THE FOLLOWING TYPES OF DATA STRUCTURES MAY BE DECLARED IN THE DATA GROUP OF AN SPL/3000 PROGRAM.

• INTEGER	16 BITS 2'S COMPLEMENT FORM
• DOUBLE INTEGER	32 BITS 2'S COMPLEMENT FORM
• REAL	32 BIT SIGN + MAGNITUDE FORM
• LONG	64 BIT SIGN + MAGNITUDE FORM
• BYTE	8 BIT PACKED, WORD ALLOCATED
• LOGICAL	16 BIT POSITIVE INTEGER

WORKS ONLY AT LOW ORDER BIT

TYPE TRANSFER FUNCTION TABLE

FROM	TO					
	LONG	REAL	DOUBLE	INTEGER	LOGICAL	BYTE
Long		REAL				
Real	LONG		FIXR FIXT			
Double	LONG	REAL		INTEGER	LOGICAL	
Integer		REAL	DOUBLE		LOGICAL	BYTE
Logical		REAL	DOUBLE	INTEGER		BYTE
Byte		REAL	DOUBLE	INTEGER	LOGICAL	

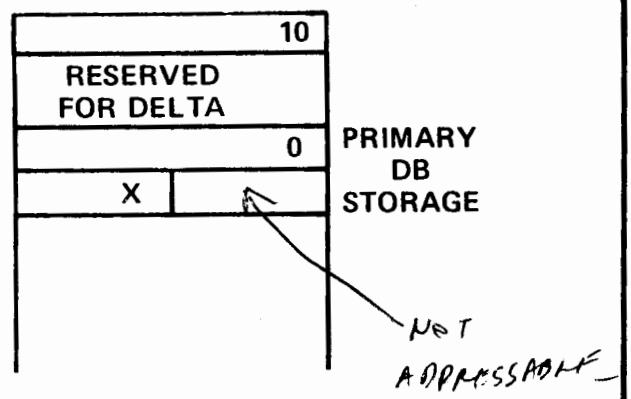
EMPTY SPACES DO NOT HAVE A FUNCTION. USE 2 OR MORE.

e.g.
```
LOGICAL LL;
LONG    XY;

XY := (LONG(REAL(LL)))
```

GLOBAL STORAGE ALLOCATION

–SIMPLE VARIABLES–

DECLARATION	ALLOCATION
INTEGER COUNT: = 10;	DB + 0
REAL DELTA;	1, 2
LOGICAL DONE: = FALSE;	3
BYTE CHAR: = "X";	4
	. . .
	255



ASSIGNMENT STATEMENTS

TYPICALLY GENERATE STACK OPS AND MEMORY REFERENCES.
IMMEDIATES ARE GENERATED TO OPTIMIZE INTEGER ARITHMETIC.

```
00001000  00000 0   $CONTROL USLINIT,ADR,INNERLIST
00002000  00000 0   BEGIN
00003000  00000 1   INTEGER A,B,C;
                            DB+000
                            DB+001
                            DB+002
00004000  00000 1   REAL R,S;
                            DB+003
                            DB+005
00005000  00000 1   R:=R+S;
                     00000    LDD    DB 003          151003     03.85
                     00001    LDD    DB 005          151005     03.85
                     00002    FADD, NOP              005100     13.90
                     00003    STD    DB 003          161003     04.03
00006000  00004 1   A:=A+B;
                     00004    LOAD   DB 000          041000     02.28
                     00005    ADDM   DB 001          071001     02.63
                     00006    STOR   DB 000          051000     02.63
00007000  00007 1   A:=A+1;
                     00007    INCM   DB 000          120000     03.50
00008000  00010 1   R:=R+1.0;
                     00010    LDD    DB 003          151003     03.85
                     00011    LDPP,000               034000     03.68
                     00015    FADD, NOP              005100     13.90
                     00016    STD    DB 003          161003     04.03
00009000  00017 1   R:=R+56
00010000  00017 1   END.
                     00017    LOAD   DB 001          041001     02.28
                     00020    ADDI,070               022470     01.05
                     00021    STOR   DB 001          051001     02.63
                     00022    PCAL,052               000000     25.00
```

LOGICAL OPERATORS

+	ADD	=	EQUALS
-	SUBTRACT	<>	NOT EQUALS
*	MULTIPLY	>	GREATER THAN
/	DIVIDE	<	LESS THAN
MOD	MODULO	>=	GREATER THAN OR EQUALS
NOT	BOOLEAN NOT-ONES COMPL.	<=	LESS THAN OR EQUALS
		XOR	EXCLUSIVE OR
LAND	BOOLEAN AND	LOR	INCLUSIVE OR
		A<=B<=C	RANGE CHECK (INTEGER EXPR)

- HIERARCHY:

NOT	(HIGHEST)
* / MOD	
+ -	
= < > < = > = < >	
LAND	
XOR	
LOR A<=B<=C	(LOWEST)

LOGICAL EXPRESSIONS

- GENERATE CODE LIKE REGULAR EXPRESSIONS.
- NOTE SPECIAL CASES LIKE CPRB.

```
INTEGER A,B,C;
LOGICAL L,M,N,O:
L:=NOT L;
L:=A<=B<=C; <<CPRB >>
L:=L+1;
```

ONE HARDWARE INSTRUCTION.

IF STATEMENTS

- CALCULATES ON THE STACK.
- BASIC TEST.
- CONDITIONAL BRANCHES WHICH TEST TOS DELETE OPERAND.
- NOTE DIFFERENCE BETWEEN AND/LAND.

```
INTEGER A,B,C;
REAL R,S;
LOGICAL FLAG,LESS;
IF A > B THEN A:=A+1;
IF FLAG THEN A:=A+1;
IF FLAG LAND A<B THEN A:=A+1;
IF FLAG AND LESS:=A<B THEN A:=A+1;
```


LESS DOES NOT GET ASSIGNED
IF FLAG NOT TRUE
IF LAND IT WOULD

IF STATEMENTS CONTINUED

- BRANCH WORDS

OVERFLOW
NOVERFLOW
= , < ' > , < > , < = , > = (CC)
CARRY
NOCARRY

```
IF <> THEN A:=A+1;
IF A<=B<=C THEN A:=A+1;
IF CARRY THEN A:=A+1;
```

GLOBAL STORAGE ALLOCATION

ARRAYS

DECLARATION

INTEGER ARRAY ADATA (0:15):

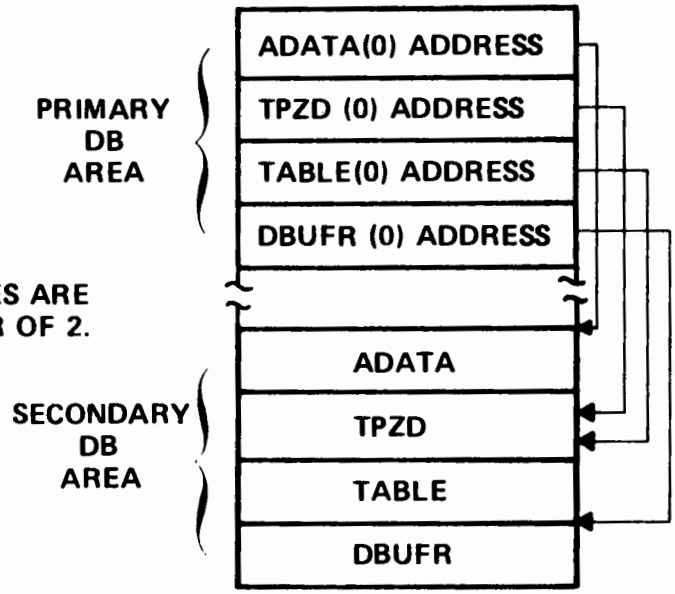
REAL ARRAY TPZD (-5:10);

LOGICAL ARRAY TABLE (10:20);

* BYTE ARRAY DBUFR (0:80);

NOTE – BYTE ARRAY ADDRESSES ARE INFLATED BY A FACTOR OF 2.

ALLOCATION



ARRAYS

- USE MEMORY REFERENCE WITH INDEXING
- DO NO BOUNDS CHECKING ON SUBSCRIPT

```
INTEGER ARRAY IBUF (0:10);
INTEGER   A:=6,
          B:=2,
          Z;

Z : = IBUF (6);
Z : = IBUF (A);
Z : = IBUF (B*3);
Z : = IBUF (B:=B*3);
IBUF := IBUF (A);
IBUF (IBUF(B)) := Z;
Z : = IBUF (-A);
```

PROCEDURES

- MUST BE DECLARED IN GLOBAL AREA.
- CALLED BY PROCEDURE CALL STATEMENT FROM WITHIN THE MAIN PROGRAM OR OTHER PROCEDURES.
- CAN BE FUNCTIONS.
- CAN HAVE LOCAL DECLARATIONS OF THEIR OWN.

PROCEDURE DECLARATION

```
TYPE PROCEDURE NAME
( FORMAL PARAMETER LIST ) ;
VALUE VALUE PARAMETER LIST ;        } PROCEDURE HEAD
SPECIFICATION PART ;
OPTION PART ;

BEGIN
        LOCAL DECLARATIONS ;
                                    } PROCEDURE BODY
        STATEMENTS ;

END;
```

- **VALUE PARAMETERS**

  COMPILER PASSES ACTUAL VALUE OF PARAMETER IN STACK.

  PARAMETER CAN BE ANY EXPRESSION.

  ADDRESSED DIRECTLY (Q-) BY PROCEDURE.

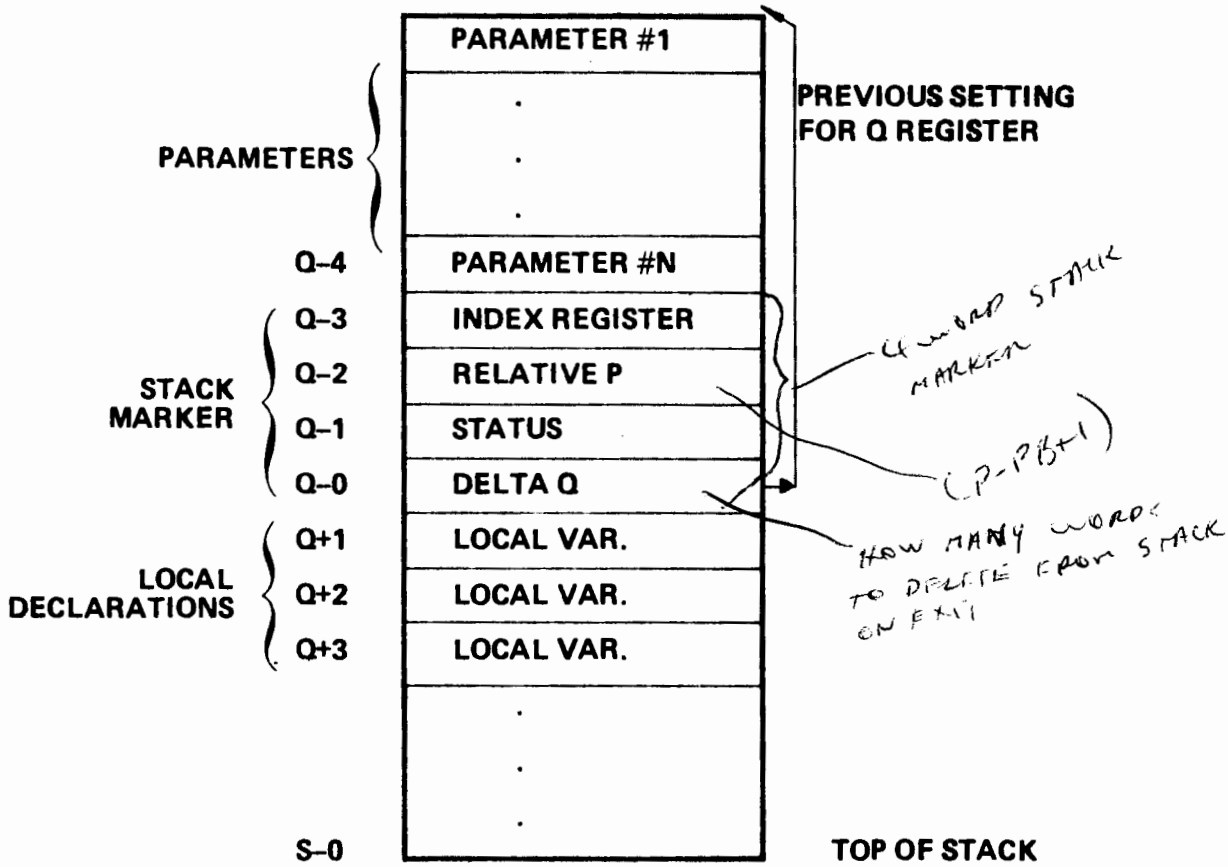
- **REFERENCE PARAMETERS** (DEFAULT)

  COMPILER PASSES ADDRESS OF ACTUAL PARAMETER.

  PARAMETER MUST BE AN IDENTIFIER.

  ADDRESSED INDIRECTLY (Q-) BY PROCEDURE.

PCAL INSTRUCTION SETS Q REGISTER AND SAVES THE ENVIRONMENT



PROCEDURE PARAMETERS/LOCAL VARIABLES

- PARAMETERS ARE Q MINUS.
- LOCAL VARIABLES ARE Q PLUS.
- PROCEDURES CAN REFERENCE GLOBAL VARIABLES (DB PLUS).
- PROCEDURE TEMPORARIES, STACK MARKER, AND PARAMETERS ARE DELETED BY EXIT
- RETURN η CREATES AN EXTRA EXIT

```
00000 0   BEGIN INTEGER A,B;
                  DB+000
                  DB+001
00000 1   PROCEDURE P(N,M);
00000 1     VALUE N;INTEGER N,M;
00000 1   BEGIN INTEGER TEMP;
                  Q +001
00000 2     TEMP:=N;
                                  00000    ADDS,001
                                  00001    LOAD  Q- 005
                                  00002    STOR  Q+ 001
00003 2     M:=TEMP;
                                  00003    LOAD  Q+ 001
                                  00004    STOR  Q- 004,I
00005 2     TEMP:=N+M;
                                  00005    LOAD  Q- 005
                                  00006    ADDM  Q- 004,I
                                  00007    STOR  Q+ 001
00010 2     N:=TEMP;
                                  00010    LOAD  Q+ 001
                                  00011    STOR  Q- 005
00012 2     M:=N*TEMP;
                                  00012    LOAD  Q- 005
                                  00013    MPYM  Q+ 001
                                  00014    STOR  Q- 004,I
00015 2   END;
                                  00015    EXIT,002
00000 1   <<NULL MAIN>>
00000 1   END.
```

PROCEDURE CALL STATEMENT

- VALUE IS PASSED FOR VALUE PARAMETERS; CAN BE EXPRESSIONS.
- ADDRESS IS PASSED FOR REFERENCE PARAMETERS; CAN BE NAME ONLY. (LRA)
- CONSTANT IS NOT REFERENCE PARAMETER, BUT COMPILER PASSES IT AS ADDRESS.

```
00000 0   $CONTROL ADR,INNERLIST
00000 0   BEGIN INTEGER A,B;
                  DB+000
                  DB+001
00000 1   PROCEDURE P(N,M);
00000 1     VALUE N;INTEGER N,M;
00000 1   BEGIN <<NULL>> END;
                                    00000    EXIT,002
00000 1   P(A,B);
                                    00000    LOAD  DB 000
                                    00001    LRA   DB 001
                                    00002    PCAL,000
00003 1   P(A+B,A);
                                    00003    LOAD  DB 000
                                    00004    ADDM  DB 001
                                    00005    LRA   DB 000
                                    00006    PCAL,000
00007 1   P(5,B);
                                    00007    LDI ,005
                                    00010    LRA   DB 001
                                    00011    PCAL,000
00012 1   P(5,10);
                                    00012    LDI ,005
                                    00013    LDI ,012
                        ^
***** WARNING *****  EXPECTS REFERENCE PARAMETER

                                    00014    PCAL,000
00015 1   END.
```

SPL WORKSESSION 5

DRAW THE STACK MARKER THAT WILL BE CREATED IF A PCAL INSTRUCTION IS EXECUTED AT THIS MOMENT:

DL = 34630

DB = 34630

Q = 40200

S = 40206

Z = 42316

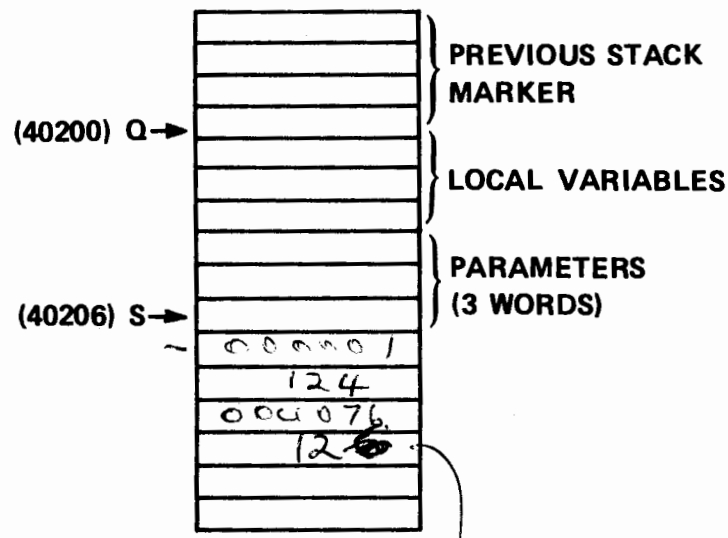
PB = 35000

P = 35123 (PCAL)

PL = 357621

X = 000001

STATUS = 004076



S-81

NO OF WORDS BACK TO Q

FUNCTION PROCEDURES

RETURNS A RESULT OF A SPECIFIED TYPE IN PLACE OF ITS NAME.

CALLED FROM WITHIN AN EXPRESSION.

EX:

```
BEGIN INTEGER NUM:=108,NIX;
    INTEGER PROCEDURE VAL(A,B,C);
        VALUE A,B,C;
        INTEGER A,B,C;
            VAL:=(A+B)*C;

    NIX:=NUM/VAL(4,5,6);
```

FUNCTION RESULTS

COMPILER PUSHES A ZERO ONTO THE STACK BEFORE PUSHING THE PARAMETERS.

THIS EXTRA LOCATION IS USED FOR THE RESULT.

Q-7	0	RESULT (VAL)
Q-6	A	
Q-5	B	PARAMETERS
Q-4	C	
Q-3	INDEX REGISTER	
Q-2	RELATIVE P	STACK
Q-1	STATUS	MARKER
Q-0	DELTA Q	

FUNCTIONS

A:= P(P2(5 * N) + I);

<< NESTED >>

P(6); << DELETES RESULT >>

STACK OPERATIONS

EXAMPLE PROGRAM:

MAIN CODE CALLS PROCEDURE ADDER

ADDER ADDS TWO NUMBERS AND CONVERTS THEIR SUM TO AN ASCII STRING BY CALLING "ASCII"

ADDER RETURNS THE STRING TO THE CALLING CODE

MAIN CODE TERMINATES

```
$CONTROL USLINIT
BEGIN
 INTEGER ONE:=1, TWO:=2;
 BYTE ARRAY ANSWER(0:5):="      ";

 INTRINSIC ASCII;

 PROCEDURE ADDER(A,B,STRING);
  VALUE A,B;
  INTEGER A,B;
  BYTE ARRAY STRING;

  BEGIN
   INTEGER SUM;
   SUM:=A+B;
   ASCII(SUM,10,STRING);
  END;

<<START OF MAIN CODE>>

      ADDER(ONE,TWO,ANSWER);
END.
```

INITIAL STACK

```
$CONTROL USLINIT
BEGIN
 INTEGER ONE:=1, TWO:=2;
 BYTE ARRAY ANSWER(0:5):="      ";

 INTRINSIC ASCII;

 PROCEDURE ADDER(A,B,STRING);
  VALUE A,B;
  INTEGER A,B;
  BYTE ARRAY STRING;

  BEGIN
   INTEGER SUM;
   SUM:=A+B;
   ASCII(SUM,10,STRING);
  END;

<<START OF MAIN CODE>>

      ADDER(ONE,TWO,ANSWER);
END.
```

```
DB→000001 ONE
   000002 TWO
   000006 ANSWER
   020040 }
   020040 } ANSWER (DATA)
   020040 }
   000000 PARM
   000000 }
   000000 } INITIAL
   140015 } STACK
Q,S→000004 } MARKER
```

BEFORE CALL TO ADDER

```
$CONTROL USLINIT
BEGIN
 INTEGER ONE:=1, TWO:=2;
 BYTE ARRAY ANSWER(0:5):="      ";

 INTRINSIC ASCII;

 PROCEDURE ADDER(A,B,STRING);
  VALUE A,B;
  INTEGER A,B;
  BYTE ARRAY STRING;

  BEGIN
   INTEGER SUM;
   SUM:=A+B;
   ASCII(SUM,10,STRING);
  END;

<<START OF MAIN CODE>>

      ADDER(ONE,TWO,ANSWER);
END.
```

```
DB→000001  ONE
   000002  TWO
   000006  ANSWER
   020040 }
   020040 } ANSWER (DATA)
   020040 }
   000000  PARM
   000000 }
   000000 } INITIAL
   140015 } STACK
Q→ 000004 } MARKER
   000001  ONE      }
   000002  TWO      } PARAMETERS
S→ 000006  ANSWER   } FOR ADDER
```

AFTER CALL TO ADDER

```
$CONTROL USLINIT
BEGIN
 INTEGER ONE:=1, TWO:=2;
 BYTE ARRAY ANSWER(0:5):="      ";

 INTRINSIC ASCII;

 PROCEDURE ADDER(A,B,STRING);
  VALUE A,B;
  INTEGER A,B;
  BYTE ARRAY STRING;

  BEGIN                    ⇦
   INTEGER SUM;
   SUM:=A+B;
   ASCII(SUM,10,STRING);
  END;

<<START OF MAIN CODE>>

      ADDER(ONE,TWO,ANSWER);
END.
```

DB→	000001	ONE
	000002	TWO
	000006	ANSWER
	020040	ANSWER (DATA)
	020040	
	020040	
	000000	PARM
	000000	INITIAL STACK MARKER
	000000	
	140015	
	000004	
	000001	ONE — PARAMETERS FOR ADDER
	000002	TWO
	000006	ANSWER
	000000	X
	000004	REL P
	060301	STATUS
Q,S→	000007	ΔQ

BEFORE CALL TO ASCII

```
$CONTROL USLINIT
BEGIN
 INTEGER ONE:=1, TWO:=2;
 BYTE ARRAY ANSWER(0:5):="      ";

 INTRINSIC ASCII;

 PROCEDURE ADDER(A,B,STRING);
  VALUE A,B;
  INTEGER A,B;
  BYTE ARRAY STRING;

  BEGIN
   INTEGER SUM;
   SUM:=A+B;
   ASCII(SUM,10,STRING);
  END;

<<START OF MAIN CODE>>

       ADDER(ONE,TWO,ANSWER);
END.
```

```
DB→000001  ONE
   000002  TWO
   000006  ANSWER
   020040 }
   020040 } ANSWER (DATA)
   020040 }
   000000  PARM
   000000 }
   000000 } INITIAL
   140015 } STACK
   000004 } MARKER
   000001  ONE   }
   000002  TWO   } PARAMETERS
   000006  ANSWER } FOR ADDER
   000000 }
   000004 } STACK MARKER
   060301 } CALL TO ADDER
Q→ 000007 }
   000003  SUM (LOCAL)
   000000  FUNCTION RTN (ASCII)
   000003  SUM    }
   000012  10     } PARAMETERS
S→ 000006  STRING } FOR ASCII
```

AFTER RETURN FROM ASCII

```
$CONTROL USLINIT
BEGIN
 INTEGER ONE:=1, TWO:=2;
 BYTE ARRAY ANSWER(0:5):="      ";

 INTRINSIC ASCII;

 PROCEDURE ADDER(A,B,STRING);
  VALUE A,B;
  INTEGER A,B;
  BYTE ARRAY STRING;

  BEGIN
   INTEGER SUM;
   SUM:=A+B;
   ASCII(SUM,10,STRING);
  END;

<<START OF MAIN CODE>>

      ADDER(ONE,TWO,ANSWER);
END.
```

```
DB→000001  ONE
   000002  TWO
   000006  ANSWER
★  031440 }
   020040 } ANSWER (DATA)
   020040 }
   000000  PARM
   000000 }
   000000 } INITIAL
   140015 } STACK
   000004 } MARKER
   000001  ONE   }
   000002  TWO   } PARAMETERS
   000006  ANSWER } FOR ADDER
   000000 }
   000004 } STACK MARKER
   060301 } CALL TO ADDER
Q→ 000007 }
   000003  SUM (LOCAL)
S→ 000001  FUNCTION RTN (ASCII)
```

★ STRING RESULT HAS BEEN RETURNED.

AFTER RETURN FROM ADDER

```
$CONTROL USLINIT
BEGIN
 INTEGER ONE:=1, TWO:=2;
 BYTE ARRAY ANSWER(0:5):="      ";

 INTRINSIC ASCII;

 PROCEDURE ADDER(A,B,STRING);
  VALUE A,B;
  INTEGER A,B;
  BYTE ARRAY STRING;

  BEGIN
   INTEGER SUM;
   SUM:=A+B;
   ASCII(SUM,10,STRING);
  END;

<<START OF MAIN CODE>>

      ADDER(ONE,TWO,ANSWER);
END.
```

```
DB→ 000001  ONE
    000002  TWO
    000006  ANSWER
    031440 }
    020040 } ANSWER(DATA)
    020040 }
    000000  PARM
    000000 }
    000000 } INITIAL
    140015 } STACK
Q,S→ 000004 } MARKER
```

STACK MARKERS

Q-3	INDEX REG.
Q-2	RELATIVE P
Q-1	STATUS REG.
Q-0	DELTA Q

PCAL -

PRESERVE THE CALLER'S ENVIRONMENT

EXIT -

RESTORE THE CALLER'S ENVIRONMENT

USING Q-RELATIVE ADDRESSING, THE CALLED PROCEDURE CAN MODIFY THE STACK MARKER TO -

- PASS BACK A VALUE TO THE CALLER IN THE INDEX REGISTER
- SET CONDITION CODE VALUES IN THE CALLER'S STATUS REGISTER

PROCEDURE OPTION PART

- **EXTERNAL REFERENCES**

```
PROCEDURE P (A,B); VALUE A; INTEGER A,B;
    OPTION EXTERNAL;
    <<NO BODY>>
```

- **VARIABLE NUMBER OF PARAMETERS**

```
PROCEDURE P (A,B,C,D); INTEGER A,B,C,D;
    OPTION EXTERNAL, VARIABLE;

P(N);        P(N, ,M);        P(N, , ,M);
```

NOTE: COMPILER PASSES A BIT MAP IN Q-4

DEFAULT FOR RUN IS TO SEARCH
SL·PUB·SYS FOR UNSATISFIED EXTERNALS

AT PREPTIME EXTERNALS FROM RL's ARE SATISFIED

i.e. IF Qt addressing SL
DB " RL

Hence observability of passing values by reference

PROCEDURE OPTIONS (CONT.)

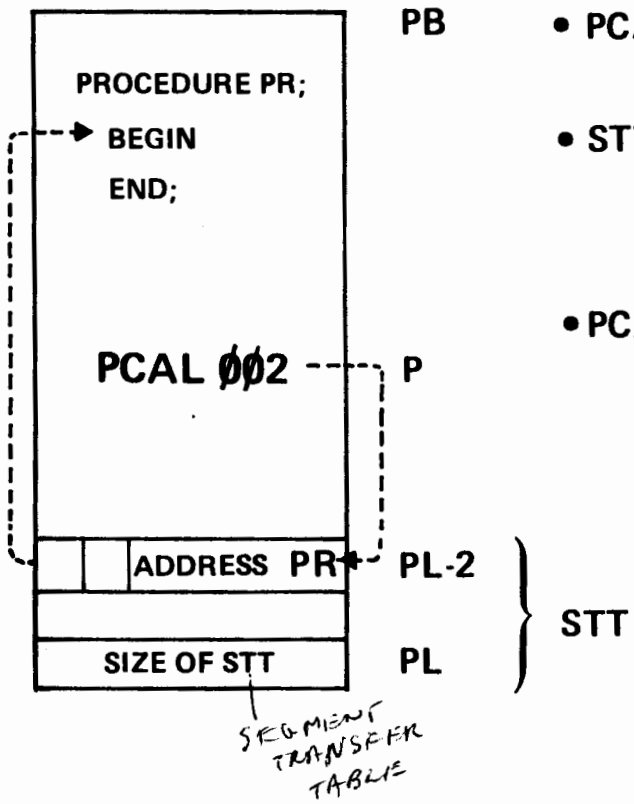
- FORWARD REFERENCES

```
PROCEDURE B (P);
VALUE P; INTEGER P;
OPTION FORWARD;
PROCEDURE A (N, M);
VALUE N, M;
INTEGER N, M;
BEGIN
        •
        •
        •
IF M > 0 THEN B (M-1);
        •
        •
        •
END;
PROCEDURE B (P);
VALUE P; INTEGER P;
BEGIN
        •
        •
        •
        A (4, P);
        •
        •
        •
END;
```

BECAUSE WE HAVEN'T DECLARED A YET

RESPECIFY PROCEDURE B

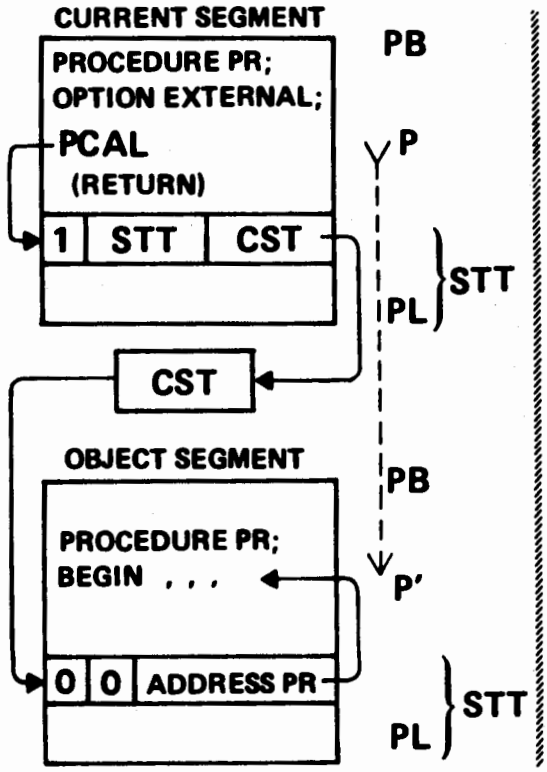
INTERNAL PCAL



- PCAL SPECIFIES STT ENTRY
- STT ENTRY CONTAINS LOCAL LABEL WHICH SPECIFIES THE PB RELATIVE ENTRY POINT
- PCAL SAVES THE RETURN P ADDRESS AND OTHER INFORMATION IN THE STACK MARKER AND SETS Q.

PCAL CAN BE INTERNAL (INTO SAME SEGMENT)
OR EXTERNAL

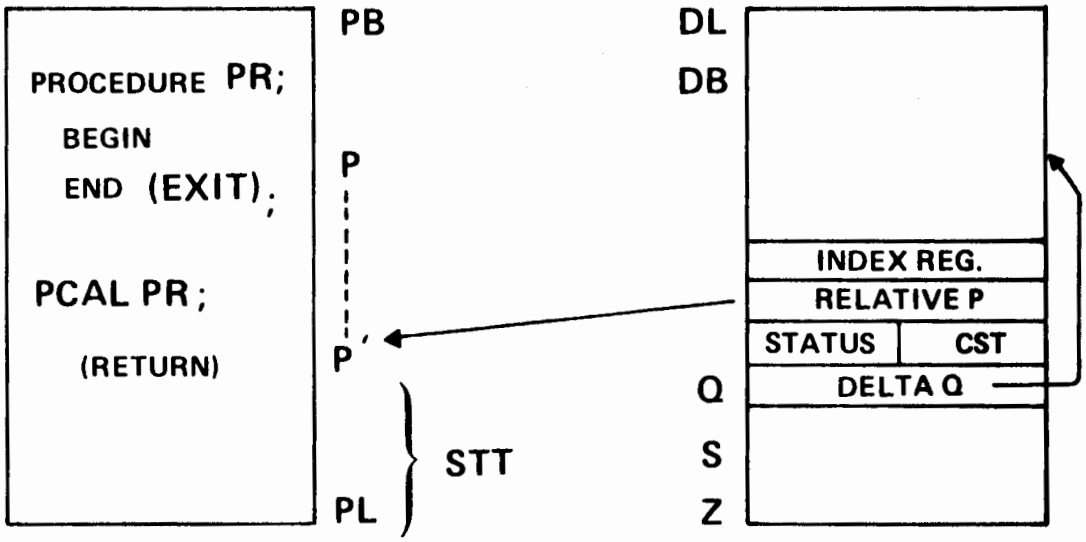
EXTERNAL PCAL



- **PCAL** SPECIFIES STT ENTRY.
- **STT** ENTRY CONTAINS EXTERNAL LABEL WHICH SPECIFIES THE CST ENTRY OF THE SEGMENT CONTAINING THE PROCEDURE AND THE STT ENTRY WITHIN THAT SEGMENT OF ENTRY POINT.
- **PCAL** USES THE CST ENTRY TO SET PB/PL TO THE NEW SEGMENT.
- **PCAL** USES THE STT ENTRY IN THE OBJECT SEGMENT TO SET P.
- **PCAL** SAVES THE RETURN ADDRESS AND OLD CST NUMBER IN THE STACK MARKER.

INTERNAL EXIT

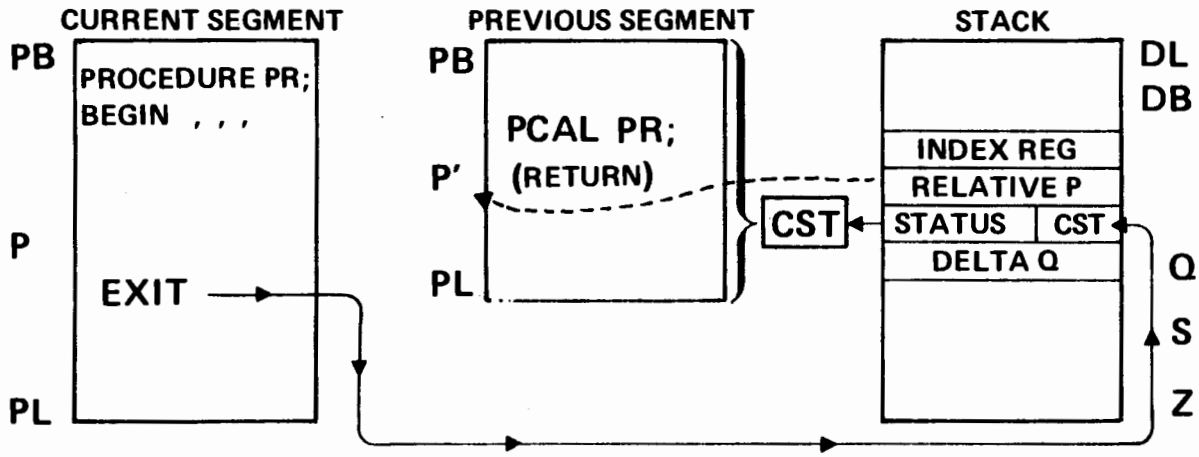
- USES STACK MARKER TO SET X, Q, AND P, PLUS STATUS
- EXIT CHECKS THAT CST NUMBER IN THE STACK MARKER IS THE SAME AS THE CURRENT CST BEFORE THE EXIT.



EXTERNAL EXIT

- OCCURS WHEN THE CST NUMBER IN THE STACK MARKER DOES NOT MATCH THE CURRENT CST NUMBER.

EXIT CHECKS THE CST NUMBER FROM THE STACK MARKER TO SET PB/PL; SETS P TO THE P RETURN ADDRESS SAVED IN THE STACK MARKER; AND RESETS THE Q REGISTER.



CODE NEVER GOES TO VIRTUAL MEMORY (ONLY STACK)

WORKING SET

BYTES

BYTE = 1/2 WORD = 8 BITS = 1 ASCII CHAR.

SPL/3000 HAS:

- BYTE VARIABLES
- BYTE ARRAYS
- BYTE POINTERS
- BYTE MOVE STATEMENTS
- BYTE SCAN STATEMENTS
- BYTE COMPARISONS

BYTE VARIABLE/BYTE ARRAYS

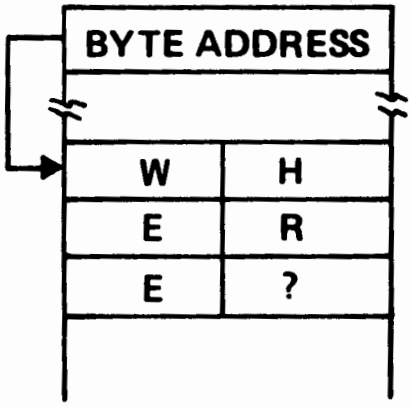
```
BYTE CHAR,
  BYT   : = "A",
  AB    : = %63;
```

ALLOCATED ONE WORD,
USES LEFT BYTE

		CHAR
"A"		BYT
%63		AB

```
BYTE ARRAY
  QUESTION (0:5) : = "WHERE?";
```

ALLOCATES BYTE ADDRESS LOCATION AND SPACE FOR 6 BYTES (3 WORDS):



BYTES

BYTE = 1/2 WORD = 8 BITS = 1 ASCII CHAR.

SPL/3000 HAS:

- BYTE VARIABLES
- BYTE ARRAYS
- BYTE POINTERS
- BYTE MOVE STATEMENTS
- BYTE SCAN STATEMENTS
- BYTE COMPARISONS

WORD MUCH FASTER

BYTE VARIABLE/BYTE ARRAYS

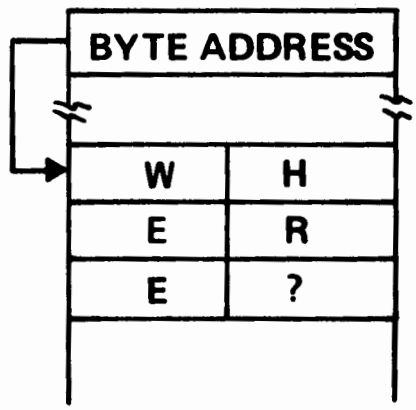
```
BYTE CHAR,
   BYT   : = "A",
   AB    : = %63;
```

ALLOCATED ONE WORD,
USES LEFT BYTE

		CHAR
"A"		BYT
%63		AB

```
BYTE ARRAY
  QUESTION (0:5) : = "WHERE?";
```

ALLOCATES BYTE ADDRESS LOCATION AND SPACE FOR 6 BYTES (3 WORDS):



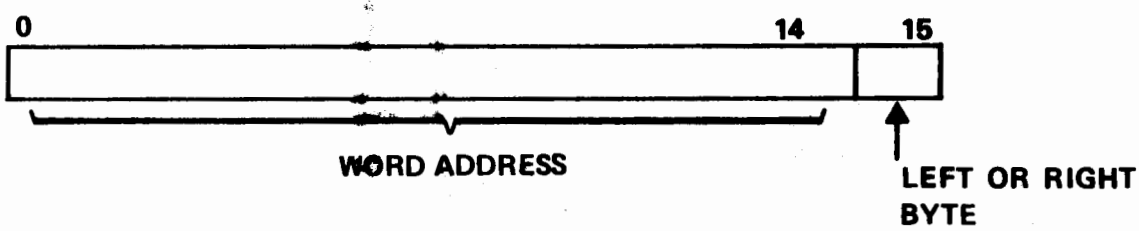
BYTES

- LDB/STB MOVE 8 BITS.
- LDB LOADS 8 BITS ONTO TOS, RIGHT-JUSTIFIED WITH LEADING ZEROES.
- STB TAKES RIGHT-MOST 8 BITS OF TOS.
- DIRECT REFERENCES SPECIFY WORD ADDRESS (E.G., DB+1); ASSUME LEFT HALF OF WORD (EVEN BYTE ADDRESS).
- INDIRECT REFERENCES USE A SPECIAL TYPE OF ADDRESS.

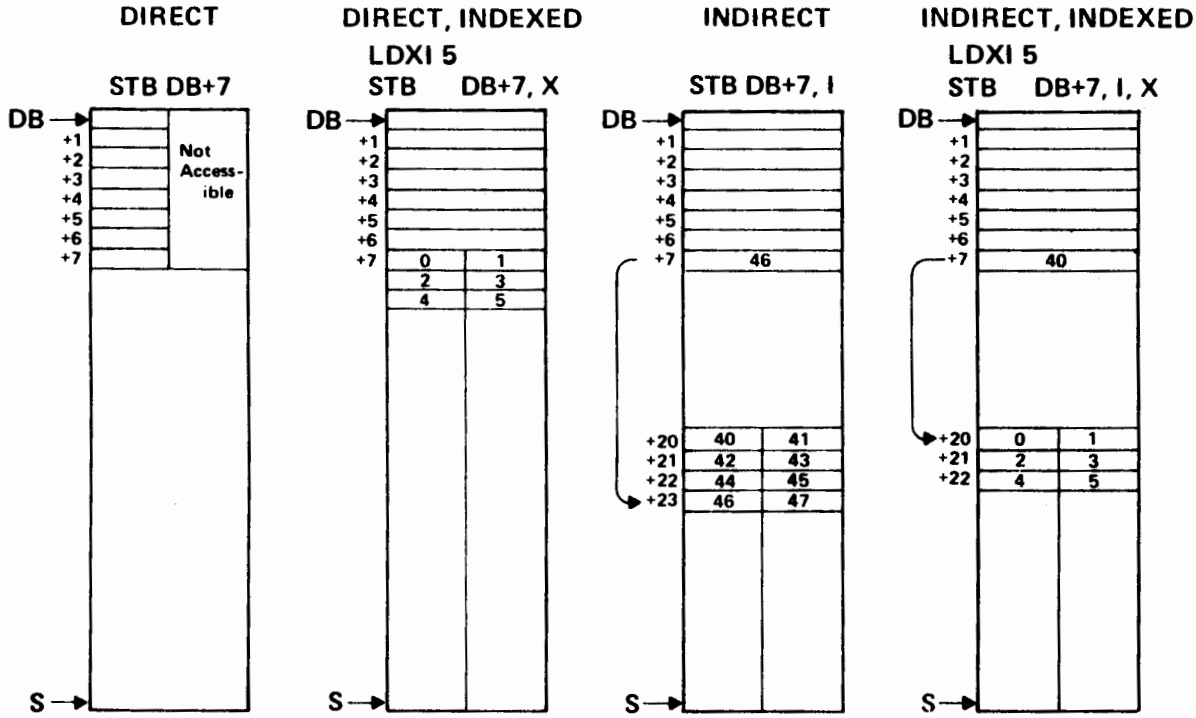
```
00000 0   $CONTROL ADR,INNERLIST
00000 0   BEGIN
00000 1   BYTE A:=1,B:=2,C:=3;
                  DB+000
                  DB+001
                  DB+002
00000 1   INTEGER I:=10;
                  DB+003
00000 1   A:=B+C;
                                  00000   LDB    DB 001
                                  00001   LDB    DB 002
                                  00002   ADD ,  NOP
                                  00003   STB    DB 000
00004 1   A:=1;
                                  00004   LOAD   DB 003
                                  00005   STB    DB 000
00006 1   I:=A;
                                  00006   LDB    DB 000
                                  00007   STOR   DB 003
00010 1   END.
```

BYTE ADDRESS

BYTE ADDRESSES ARE INFLATED BY A FACTOR OF TWO, THUS, THE WORD ADDRESS DB + 256 (%400) IS EQUIVALENT TO THE BYTE ADDRESS DB + 512 (%1000).



EXAMPLES OF BYTE ADDRESSING



S-103

LOAD ~~GOES~~ PUTS ONTO STACK AS FULL WORD WITH LEADING ZEROES (TAKES RIGHT BYTE OF DB+7)

MUST BE INDEXED TO LOOK AT RIGHT BYTE

POINTER VARIABLES

A POINTER VARIABLE CONTAINS THE ADDRESS OF A VARIABLE. WHEN THE POINTER IS USED IN AN EXPRESSION IT CREATES AN AUTOMATIC INDIRECT REFERENCE TO THE CONTENTS OF THE WORD OR BYTE WHICH IT DESIGNATES.

ASSIGN TO TOS THE ~~ADDRESS IN IP~~ CONTENTS OF THE ADDRESS IN IP.

INTEGER POINTER IP;

TOS; =IP; → IP ADDR OF DATA → DATA DATA VALUE

INDIRECT LOAD <u>THROUGH</u> IP OF DATA CONTENTS

ASSIGN TO TOS THE CONTENTS OF IP.

TOS:=@IP; → IP ADDR OF DATA

DIRECT LOAD OF IP CONTENTS

NOTE: BOTH DATA AND POINTER ARE ASSUMED TO HAVE THE SAME TYPE.

POINTER DECLARATION

POINTERS MUST BE DECLARED AND ASSIGNED A VALID VARIABLE TYPE. DECLARATION CAN ALSO INCLUDE INITIALIZATION – SPECIFICATION OF THE CONTAINED ADDRESS.

```
BYTE POINTER CHAR,
        FIRST: = @BARY,
        LAST:  = @BARY(79);
INTEGER POINTER TESTVAL: = @IARY(5),
    NEXT NO;
POINTER B00L1,      <<LOGICAL ASSUMED>>
            B00LZ: = @LSIMPVAR;
  REAL POINTER X: = @RARY,
                Y,
                Z;
```

The address of the first element of BARY

The address of the last element of BARY

POINTERS

- @ P GENERATES DIRECT REFERENCE.
- P GENERATES INDIRECT REFERENCE.
- @ VARIABLE GENERATES LRA

```
00000 0   $CONTROL ADR,INNERLIST
00000 0   BEGIN
00000 1   POINTER P;
                  DB+000
00000 1   LOGICAL L;
                  DB+001
00000 1   P:=L;
                                  00000    LOAD   DB 001
                                  00001    STOR   DB 000,I
00002 1   @P:=@L;
                                  00002    LRA    DB 001
                                  00003    STOR   DB 000
00004 1   L:=P;
                                  00004    LOAD   DB 000,I
                                  00005    STOR   DB 001
00006 1   L:=@P;
                                  00006    LOAD   DB 000
                                  00007    STOR   DB 001
00010 1   END.
```

HEWLETT hp PACKARD

SPL WORKSESSION 6

GIVEN THESE DECLARATIONS:

```
INTEGER POINTER PNTR;
INTEGER ARRAY ARRY (0:100);
INTEGER IVAR;
```

EXPLAIN THESE STATEMENTS:

1. IVAR:=PNTR+15; — LOAD, STOR
2. IVAR:=@PNTR+15; — LOAD STOR
3. @PNTR:=@ARRY(5); — LRA STOR
4. @PNTR:=@PNTR+1; — LOAD STOR
5. PNTR:=ARRY(4);

S-107

1. ASSIGN TO IVAR (CONTENTS OF ADDRESS IN PNTR) +15 ✓

2. " " " (CONTENTS OF PNTR) +15 ✓

3. ~~" " THE CONTENTS OF THE ADDRESS IN~~

~~PT~~ ASSIGN TO PNTR THE CONTENTS OF THE ~~ADDRESS~~ IN THE 5TH ELEMENT OF ARRY

4. ~~ASSIGN TO PNTR THE CONTENTS OF~~ INCREMENT POINTER BY 1

5. ASSIGN TO THE ADDRESS IN POINTER THE CONTENTS OF ELEMENT 4 OF THE ARRAY

COMPATABILITY WITH POINTERS

DURING DECLARATION, BYTE AND WORD ADDRESSING CAN BE MIXED, BECAUSE THE COMPILER CONVERTS ADDRESSES:

```
BYTE ARRAY B(0:71);
LOGICAL POINTER LP := @B(2);
BYTE POINTER BP;
```

DURING EXECUTION, THE PROGRAM MUST EXPLICITLY CONVERT:

```
<<WRONG>>     @LP :=@B(2);
<<RIGHT>>     @LP :=@B(2) & ASR(1);
<<RIGHT>>     @BP :=@LP & ASL(1);
```

DIVIDE BY 2

MOVE STATEMENTS

MOVE DESTINATION:= SOURCE, (COUNT);

- DESTINATION AND SOURCE CAN BE ARRAYS/POINTERS, WITH/WITHOUT INDEXES.
- COUNT IS AN INTEGER EXPRESSION (+ MEANS INCREMENT ADDRESSES, – DECREMENT).

MOVE BUF:= AR2, (20);

MOVE BUF(10):= AR2(3),(-4);

STACK DECREMENT OPERAND

- STACKED VALUES

DESTINATION	S - 2
SOURCE	S - 1
COUNT	S

- MOVE BUF: = AR2, (20), SDEC;

  SDEC = 0,1,2,3.

  NUMBER OF WORDS TO DELETE AFTER MOVE.

  DEFAULT IS DELETE ALL THREE.

- ADDRESSES POINT TO NEXT ITEM NOT MOVED UPON COMPLETION.

- COUNT ALWAYS EQUALS ZERO UPON COMPLETION.

IF NOT SDEC LEAVES DIRTY STACK

■ CONSTANTS

```
MOVE BUF:=(5, 10, 23, 5(0,1,2));

MOVE BUF:= "ABCDEFG";
```

NO COUNT IS REQUIRED.
ALL CONSTANTS TO BE MOVED
ARE STORED IN THE CODE
SEGMENT.

■ STACKED ADDRESS

```
TOS:=@BUF;

MOVE * := AR2,(20),1;

MOVE * := *, (20);      <<MOVE BYTES ONLY>>
```


WHATEVER IS CURRENTLY TOS

MOVE WORDS/BYTES

- WORDS

  (INTEGER,LOGICAL,REAL,DOUBLE,OR LONG)

  WORD COUNT.

  ALLOWS ONLY ONE STACKED ADDRESS.

- BYTES

  (BYTE ARRAY OR POINTERS ONLY.)

  BYTE ADDRESSES.

  BYTE COUNT.

  ALLOWS TWO STACKED ADDRESSES.

MOVING BYTES TO WORDS OR VICE VERSA IS ILLEGAL.
(ADDRESS TYPES MUST MATCH.)

MOVE BYTES WHILE STATEMENT

MOVES BYTES WHILE THEY ARE A CERTAIN TYPE.

MOVE DESTINATION := SOURCE WHILE CONDITION , SDEC ;

- DESTINATION AND SOURCE ARE BYTE ARRAYS OR POINTERS (OR *).
- CONDITION IS A, N, AN, AS, ANS
  (A=ALPHA, N=NUMERIC, S=SHIFT TO UPPER)

```
BYTE ARRAY B(0:10);BYTE POINTER PB;
MOVE B := PB(5) WHILE AN, 1;
```

ON COMPLETION:

- BOTH SOURCE AND DESTINATION ADDRESSES POINT TO THE NEXT BYTE (NOT MOVED)
- THE CONDITION CODE IS SET ON THE LAST CHARACTER EXAMINED
- THE SDEC OPERAND IS USED TO DECREMENT THE STACK

STACK

DESTINATION ADDRESS	S - 1
SOURCE ADDRESS	S

HENCE MIXED U/L CASE COMMANDS ALLOWED IN MPE

SCAN STATEMENTS

- SCAN A BYTE ARRAY UNTIL A TEST OR TERMINAL CHARACTER IS FOUND OR WHILE A TEST CHARACTER IS FOUND:

  SCAN BYTE ADDRESS UNTIL TESTWORD, SDEC;
  SCAN BYTE ADDRESS WHILE TESTWORD, SDEC;

TESTWORD =

0 — 7	8 — 15
TERMINAL CHARACTER	TEST CHARACTER

- SCAN SETS CARRY IF TERMINAL CHARACTER IS FOUND, OR SETS NOCARRY IF TEST CHARACTER IS FOUND.

- SCAN SETS CONDITION CODE ON LAST CHARACTER:

  <(SPECIAL) =(ALPHA) > (NUMERIC)

STACKED VALUES

BYTE ADDRESS	S-1
TESTWORD	S

SDEC = 0, 1, 2. DEFAULT = 2.

- BYTE ADDRESS MUST BE SAVED TO KNOW HOW FAR SCAN WENT.

```
BYTE POINTER BP;
SCAN BP WHILE %6440,1;
@BP:=(TOS: = TOS);  << DUPLICATE TOS >>
SCAN * UNTIL TEST, 1;
```


ie. DON'T POP

COMPARE BYTES

```
BYTE ARRAY BUF(0:100), INBUF(0:20);

IF BUF = "END OF DATA" THEN . . . ;

IF BUF = INBUF , (10) THEN . . . ;

IF * = * , (5),1 THEN . . . ;
```

STACK

FIRST ADDRESS	S - 2
SECOND ADDRESS	S - 1
COUNT	S

EQUIVALENCE

REUSE, RELABEL, RETYPE STORAGE WHICH IS ALREADY ALLOCATED

```
INTEGER A;
LOGICAL B=A;

INTEGER ARRAY A'(0:100);
BYTE ARRAY B'(*)=A';

LOGICAL C=DB+7;

LOGICAL IND=X;
```

BASE REGISTER REFERENCE

= DB+N	(N= 0 . . . 255)
= Q+N	(N= 0 . . . 127)
= Q-N	(N= 0 . . . 63)
= S-N	(N= 0 . . . 63)

- USED TO ASSIGN LOCATIONS TO POINTERS, SIMPLE-VARIABLES.
- ARRAYS CAN ALSO BE BASE REGISTER REFERENCED.
- NO SPACE ALLOCATED.

EQUIVALENCING TO REGISTER—RELATIVE ADDRESSES

- ARRAY A(*) = DB + 5;

  <<DIRECT ARRAY, ZERO ELEMENT AT DB+5>>

- ARRAY B(@) = DB + 5;

  <<INDIRECT ARRAY, ZERO ELEMENT POINTED AT BY DB+5>>

INDEXED IDENTIFIER REFERENCE

= ARRAYNAME (INDEX)

= POINTERNAME (INDEX)

USED TO SPECIFY THAT THE ZERO ELEMENT OF A NEW ARRAY IS TO COINCIDE WITH SOME PREVIOUSLY-DEFINED ARRAY (OR POINTER) ELEMENT.

```
INTEGER ARRAY REUSE (*) = POINT(6);
```

NO SPACE ALLOCATED, BUT NEW POINTER MAY BE ALLOCATED.

OWN

- LOCAL DATA ITEMS CAN BE DECLARED "OWN":

```
OWN INTEGER A;
OWN ARRAY B(0:10);
OWN POINTER P;
```

- BELONG TO PROCEDURE, BUT ALLOCATED PERMANENT SPACE IN THE DB AREA.

- THUS, OWN VARIABLES DO NOT DISAPPEAR EACH TIME THE PROCEDURE EXITS.

AN IDENTIFIER IN A FORTRAN DATA STATEMENT IS THE SAME AS AN OWN VARIABLE IN AN SPL PROGRAM.

IF "OWN" CANNOT BE
SL PROCEDURE

OWN LOCAL ARRAYS

OWN INTEGER ARRAY SAVE (LOWER:UPPER);

- CAN BE INITIALIZED, BUT IS ONLY INITIALIZED AT THE START OF THE PROGRAM EXECUTION.
- SPACE FOR ARRAY REMAINS THROUGHOUT LIFE OF PROGRAM.

Basic 19(5)
10(5)

FOR STATEMENT

- REPEAT A STATEMENT UNTIL AN INDEXING VARIABLE EXCEEDS THE TERMINATION VALUE.

- FORM:

  FOR INTEGER VAR:= ARITHMETIC EXPRESSION **STEP** ARITHMETIC EXPRESSION **UNTIL** ARITHMETIC EXPRESSION

  DO STATEMENT;

- NOTES: DEFAULT STEP IS 1.

  FOR *I:=1 UNTIL 10 DO A(I):=I*1;

  * MEANS DO IT ONCE ALWAYS.

  THE INCREMENT AND LIMIT ARE DETERMINED ONLY ONCE.

THIS STATEMENT IS EQUIVALENT TO A FORTRAN DO STATEMENT.

FOR STATEMENT CAUTIONS

- CONTROL VALUES RESIDE IN THE STACK DURING FOR STATEMENT EXECUTION.

- WITHIN THE RANGE OF **FOR**

  AVOID ANY EXPLICIT MODIFICATION OF THE STACK EG.

  TOS:=ABC;

  ABC:=TOS;

- IF THE INDEX REGISTER IS USED AS THE INDEX VARIABLE, AVOID USING SUBSCRIPTED VARIABLES OF THE FORM **ARRAY (5)** AND A<=B<=C CONSTRUCTIONS.

FOR STATEMENT EXAMPLE

```
INTEGER I, MAX, RANGE, SUM:=0,
    A:=2, B:=4, C:=5, FOFI;

    FOR I:=MAX STEP-RANGE/4 UNTIL MAX-RANGE
        DO  BEGIN
                FOFI:=A*I + 2*I+C;
                SUM:=SUM+FOFI;

                END;
```

FOR STATEMENTS

VARIABLE ADDRESS	S - 2
STEP VALUE	S - 1
LIMIT VALUE	S

- **TBA** (TEST AND BRANCH ON A)
  MTBA (MODIFY, TEST AND BRANCH)

- * DELETES INITIAL **TBA**

- **X** USES **TBX, MTBX**

STEP VALUE	S - 1
LIMIT VALUE	S

```
00000 0   BEGIN
00000 1   INTEGER A,B,C,X=X;
                  DB+000
                  DB+001
                  DB+002
                  X  000
00000 1   FOR A:=B STEP C UNTIL 100 DO B:=A+B;
                                 00000    LOAD  DB 001
                                 00001    STOR  DB 000
                                 00002    LRA   DB 000
                                 00003    LOAD  DB 002
                                 00004    LDI ,144
                                 00005    TBA   P+ 002
                                 00006    BR    P+ 000
                                 00007    LOAD  DB 000
                                 00010    ADDM  DB 001
                                 00011    STOR  DB 001
                                 00012    MTBA  P- 000
00013 1   FOR *A:=1 UNTIL 10 DO B:=B+1;
                                 00013    LDI ,001
                                 00014    STOR  DB 000
                                 00015    LRA   DB 000
                                 00016    LDI ,001
                                 00017    LDI ,012
                                 00020    INCM  DB 001
                                 00021    MTBA  P- 000
00022 1   FOR X:=1 UNTIL 10 DO A:=B*A;
                                 00022    LDXI,001
                                 00023    LDI ,001
                                 00024    LDI ,012
                                 00025    TBX   P+ 002
                                 00026    BR    P+ 000
                                 00027    LOAD  DB 001
                                 00030    MPYM  DB 000
                                 00031    STOR  DB 000
                                 00032    MTBX  P- 000
00033 1   END.
```


A ≡ 105
B ≡ S - 1

BIT FUNCTIONS

- EXTRACT
- DEPOSIT

ALL BIT OPERATIONS CAN BE USED IN PLACE OF VARIABLES OR CONSTANTS WITHIN

ARITHMETIC EXPRESSIONS

LOGICAL EXPRESSIONS

BIT EXTRACT

EXTRACTS A CONTIGUOUS BIT FIELD FROM A WORD. THE RESULT IS RIGHT JUSTIFIED WITH LEADING ZEROES.

FORM:

VARIABLE. (LEFT EXTRACT BIT: EXTRACT FIELD LENGTH)

EXAMPLE:

ZVALUE. (7:4)

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

THE ORIGINAL VALUE (ZVALUE) REMAINS UNALTERED.

MAXIMUM EXTRACT FIELD IS 15 BITS.

BIT DEPOSIT

DEPOSIT A CONTIGUOUS BIT FIELD IN AN ASSIGNMENT VARIABLE.

FORM:

VARIABLE. (LEFT DEPOSIT BIT : DEPOSIT FIELD LENGTH) := EXPRESSION

EXAMPLE:

PAT.(0:8) := %40516

0	1	0	0	1	1	1	0	0	1	0	0	1	1	1	1,	PAT
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	1	0	0	0	0	0	1	0	1	0	0	1	1	1	0,	EXPRESSION (%40516)
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	

THE REST OF THE DEPOSIT WORD (PAT) IS UNCHANGED.

SUBROUTINES

LIKE PROCEDURES, BUT

- NO LOCAL DECLARATIONS
- NO "OPTION" PART
- DOES NOT CHANGE Q
- USES S- ADDRESSING
- MUST BE IN SAME SEGMENT

RULES/CAUTIONS/RESTRICTIONS

- DON'T UPSET STACK.
- GLOBAL ARE GLOBAL ONLY.
- SUBROUTINES CAN BE DECLARED INSIDE PROCEDURES.
- OK TO ADDRESS GLOBAL VARIABLES AND PROCEDURE LOCAL VARIABLES.

SUBROUTINES

- ADDRESS PARAMETERS S MINUS (CHANGES DYNAMICALLY).
- ASSUMES RETURN ADDRESS ON TOS FOR SXIT

```
00000 0   $CONTROL ADR,INNERLIST
00000 0   BEGIN
00000 1   INTEGER I;
                DB+000
00000 1   SUBROUTINE S(A,B,C);
00000 1     VALUE A,B;
00000 1     REAL A;INTEGER B,C;
00000 1   BEGIN
00000 2     C:=INTEGER(FIXR(A));
                                  00000   LDD    S- 004
                                  00001   FIXR,  DTST
                                  00002   DELB,  NOP
                                  00003   STOR   S- 002,I
00004 2     C:=C+1;
                                  00004   INCM   S- 001,I
00005 2     B:=B-1;
                                  00005   DECM   S- 002
00006 2     A:=A+1.0;
                                  00006   LDD    S- 004
                                  00007   LDPP,000
                                  00010   FADD,  NOP
                                  00011   STD    S- 006
00012 2     B:=C+I;
                                  00012   LOAD   S- 001,I
                                  00013   ADDM   DB 000
                                  00014   STOR   S- 003
00015 2   END;
                                  00015   SXIT,004
00020 1   S(4.5,2,I);
                                  00020   LDPP,000
                                  00021   LDI ,002
                                  00022   LRA    DB 000
                                  00023   LRA    P+ 002
                                  00024   BR     P- 024
00027 1   END.
```

ADDITIONAL SPL TOPICS

- ASSEMBLE STATEMENT
- PUSH & SET STATEMENTS
- DEL, DELB, DDEL STATEMENTS
- BIT SHIFT
- BIT CONCATENATE
- BRANCH CONDITIONS
  [DXBZ, DABZ, IXBZ, IABZ]

- DIRECT ARRAYS
- TOS CONSTRUCTIONS
- STACKED PARAMETERS
- X REGISTER VARIABLES
- BASED INTEGERS
- COMPOSITE INTEGERS

LOAD : SPL FILE' — NEDSL SEGMENT NAME
: SEGMENTER — LISTUSL
SL SL [— NDEL SEG'
USL — EXIT

See MPE SEGMENTER

Slide S133 is really a catchall for omitted materials in SPL in the course.

We said at the very beginning of the class that there was a great deal of syntactic material that we simply would not be able to explain in the 2 - 2½ days that are allotted to the course - this portion of the course.

What we're going to do is we are going to walk through with you this list of additional topics and take a little time to discuss each. You might choose to take a very similar approach during the class if you have time and if the students are interested, however it's also possible to treat this slide very, very lightly and just indicate that these are things that we did not cover in any great depth in the SPL materials.

Let's begin with the ASSEMBLE statement. In the first place we have alluded to the fact that if we had high level constructions to generate all the possible machine instructions available on the 3000, the syntax for SPL would be probably quite complicated indeed. You would have to have statements for linked list searches and you would have to have statements for I/O instructions. for reading the switch register. You would have to be able to emit all Assembly language with a machine level instructions from high level SPL. This obviously is impractical, but we mentioned that we do have the ability to assemble as well as to compile with this particular piece of software, with SPL. So the ASSEMBLE statement is typically used or required for generating specialised code, writing things like the I/O system in MPE or portions of the I/O System, where we're doing specialised instruction like START I/O or TEST I/O, READ I/O, WRITE I/O which generally require privileged mode anyway. So the ASSEMBLY statement is indeed useful to emit these constructions which are not generally available from high level language.

But it is used very scarcely in MPE, very scarcely, and students should be reminded about this at this point, and if they feel that they need to actually get down on instruction for instruction code generation level, they should see the SPL Reference Manual for the format of the individual instructions. They've seen a great number of mnemonics or Assembly language or Assembly level instructions in the innerlist that we presented in the class and quite a few of them will be familiar with those instructions. But probably not with some of the more esoteric offerings of the instruction set.

The next item is PUSH and SET. These are high level statements in SPL which we simply haven't covered. Basically the PUSH statement copies the registers into the stack and the SET statement takes registers from stack and copies them back into the hardware registers. You might, for example, use a PUSH statement if you wanted to determine at execution time the actual size of your stack by pushing both DL and Z into the stack and then performing the appropriate arithmetic computations to determine the difference between the two. Some people forget the order in which these registers are pushed into the stack and the order in which they are required to be in the stack in order to set properly. If you forget that, of course, the order is well documented in the instruction manual, and if you have the students turn to the formatted machine instruction set portion of the MPE Pocket Guide or the SPL Pocket Guide you can show them the PUSH registers instruction PSHR or the SET registers instruction SETR PUSH and SET registers and in each of those instructions is a small sub-field, a bit-field. Each bit corresponds to a register, i.e. either pushed or set, and the registers are pushed into the stack in the same order as they appear in the bit map going left to right. There are certain

things which you cannot do to the status register, certain things which you cannot do to other registers unless you are operating in privileged mode. For example, it is tempting for a beginning student to think since these are high level constructions that he can simply push or copy the status register into the stack, set on the privileged mode bit and set the status register back into the hardware register, thus enabling privileged mode for his executing program. This of course is the sort of thing that's trapped out by the microcode checks in the set register instruction.

The next thing on the list is the set up DELETE instructions available from high level SPL. These are really used to clean up the stack after you may have left something on it, e.g. with an SDEC, other than default, in a move or a scan, or a byte compare. DEL deletes a single word from the top of stack and DDEL deletes a double word on the top of stack. These are both high level statements in SPL and Assembly language mnemonics which you can find in the instruction documentation.

SPL also has quite an ability to generate SHIFT operations and these can be done from high level SPL. The format that you use is the variable identifier and an ampersand, the shift operation which is really the Assembly language mnemonic, and followed by, enclosed in parenthesis, a shift count. There is great variety in terms of the number of shift instructions that are actually available on the 3000. There are both arithmetic and shift instructions and logical shift instructions. There are left shifts and right shifts and circular shifts, single word shifts, double word shifts, even relatively unusual shifts - triples that in the past have been used for normalising floating point instructions. So there is just terrific variety in terms of shifts. You have seen some already in the class, in connection with bytes and pointers with adjusting addresses effectively multiplying addresses by two, and dividing addresses by two to convert

word and byte addresses. You can use these shift operations in much the same way that you can use the extracts and deposits, in that you can use the shift operations where you can use variables, generally speaking on the right hand side of an assignment statement.

Concatinate is the next instruction, or next set of constructions we want to talk about. The key word that cues SPL to this construction is the word CAT for concatinate. This is the key word. Concatinate basically uses an extract field instruction to isolate one bit subfield from a word and a deposit field instruction to deposit it in a corresponding word. All this is considered really one operation as far as CAT is concerned. It is not very often used and really in our opinion it is quite a bit clearer if you simply use the extract field instructions and deposit field instructions, or extract and deposit constructions we've talked about. On a separate basis they accomplish exactly the same thing and the code generated would probably be about the same.

We said earlier in the class too that we were going to continually expand or add to the logical expression type constructions. This next item on additional SPL topics really expands logical expressions. It is the instructions DXBZ and DABZ and IXBZ and IABZ. Standing on their own these are really Assembly language mnemonics and they stand for Decriment and Index Register and branch on zero, Decriment the Top of Stack or A and branch on zero, increment the index register and branch on zero and increment the top of stack and branch on zero. These may be used in logical expressions such that you may say, if DXBZ then, some statement, else, some other statement. The logical expression is considered to be true if the branch would be taken, when the Assembly language instruction is executed. In other words, if the result after the increment or decriment is zero the branch would be taken. These form quite efficient logical expressions in that they are themselves branching and testing constructions.

Next we'll talk for a moment about direct arrays. Again, there are some cautions to beware of if bounds are specified for sub scripts and direct arrays. Let's take an example and try and illustrate this. Suppose we had an array which was in local storage, it was inside a procedure, and its subscripts went from 2000 to 2002. Now basically we have plenty of room to store the data Q-relative because of course it takes only three locations if it is integer array or logical array. But the problem is that the zero element of the array is some 2000 locations from the data and the trouble is the zero element cannot be directly addressed. If the zero element of the direct array cannot be directly addressed then the declaration of the direct array is considered illegal. This is a good time to review with students the summary chart on arrays, declarations, and such. In the SPL Pocket Guide is a very concise chart and it is presented in tabular form and it talks about when pointers are allocated and whether arrays can be initialized and all that sort of thing. It is a very good summary chart - students should certainly be acquainted with it.

We are going to continue now by talking about top of stack constructions. Top of stack constructions offer an almost unlimited potential for trouble, for students who really are unfamiliar with the machine. Top of stack takes on the predominant type of the expression in SPL, i.e. if top of stack is used in an integer expression it takes on default the type integer, if it is used in a logical expression it takes on default type logical, if it is used in a double integer it takes on default type double. So it is very flexible that way and you never need to use type transfer functions with the identifier TOS. The thing that you have to realise is that when TOS is encountered in the evaluation of an expression it is used very literally and that is it

means take whatever happens to be on the top of stack at this point in the expression and use that as a variable. Now, in a very complicated expression, this can be quite dangerous if you don't understand the way code has been generated and you don't understand what happens to be on the top of stack at all times, using TOS can be quite a dangerous construction. There is a rule that will generally predict the code that is generated when you use TOS type constructions, and here is the rule. Write the expression you are trying to evaluate with the TOS's in it, then generate code as if TOS were a variable and use LOADS and STORES to deal with it. Then step three is eliminate all the LOAD and STORE code from your statement, from your expression, and the code remaining would be the code that would be produced if TOS was used and the compiler had its way. For example, say TOS is replaced by TOS + TOS. The second step requires that you generate the code, load TOS, load TOS, add, and store TOS. Now eliminating all the LOADS and the STORES of TOS the net result is ADD. The stack operation ADD and that's what would be generated for the statement TOS replaced by TOS plus TOS.

The next topic is stack parameters. You may recall from our discussion of MOVE and SCAN statements that we were able from time to time to put in an asterisk in place of an amperand to indicate that we had already placed the amperand in the stack and that SPL was not required to load it. We have a very analogous situation for parameters in a procedure call, i.e. when we use an asterisk in place of the parameter identifier to indicate to SPL that we have already loaded that parameter onto the stack. Caution students that if they use stack parameters with function procedures they are responsible for setting up or for allocating return space on the stack for the functional value. Another very important consideration is that you cannot have SPL put a

parameter on the stack for you for procedure call and then specify that paramaters after that one are already stacked. This would of course put the parameter on the stack in the wrong order and SPL does no exchange instructions, implements a no code to move the parameters and reorder them on the stack prior to the procedure call.

The next topic is X register variables or index register equivalencing. Again, we simply want you to caution students that when they are equivalencing anything to the index register that one change of an equivalenced identifier simply changes all other identifiers. And also caution them that certain constructions like constants in subscripts, or any subscripting compare ranging branches and other such things also cause the index register to assume different values.

The next topic is based integers. We use based integers a great deal in SPL, in fact they're very common, and the common one is the default base, and that is base 8, although you can use bases from 2 to 16. Based integers are written % sign and the left parenthesis, then the base then the right parenthesis and then the number following that. The digits of the number of course must be valid in that base, in other words a 9 would certainly be an invalid digit in a binary number. If you are using the default base, i.e. just octal, you would put in a % sign, leave out the base and its enclosing parenthesis, and just specify the octal number. An example of a number written in binary of course is %(2)101101 and so on. We may also generate constants of bases up to 16, a hexadecimal constant might look like %(16)AFED. One caution regarding the hexadecimal constants based integer constants is that the convention - the standard convention, that to turn an integer constant into a double integer constant you

follow it by a capital d does not quite hold for hexadecimal based integers. Because D of course is a valid digit in hexadecimal base, you need to precede the final D, or the D that means double integer, by a space.

Composite integers is the next topic and the final topic on the slide. Composite integers are enclosed in square brackets and they are composed of, fields, or field lengths, and values that are placed in the fields. The syntax for each field is length/value, and you could have several fields specified in the single number, the length/value portions being separated by commas inside the square brackets. Composite integers can be quite useful where you are dealing with bit subfields and where these are natural like, for example, the bit subfields in F options or in A options in working with the file system, or the subfields that determine the speed and the terminal type when you issue an F control 37 to speed sense a terminal. We want to make sure that in dealing with this material and particularly in dealing with this Slide, 133, that you treat it with a great deal of flexibility.

Please feel free to add or to delete from this material as you see fit and to enhance your own presentation and customize the material to your own style as an instructor.

This slide, S133, ends the SPL section of the course. Students should now be ready to begin Lab Exercise 2 and as an instructor you should be aware that Lab Exercise 2 is significantly more complex than Lab Exercise 1. If a student cannot finish it in one Lab period, then you should encourage him to work on it in background mood during the File System session of this class.

- - - -

FILE SYSTEM

FILE SYSTEM

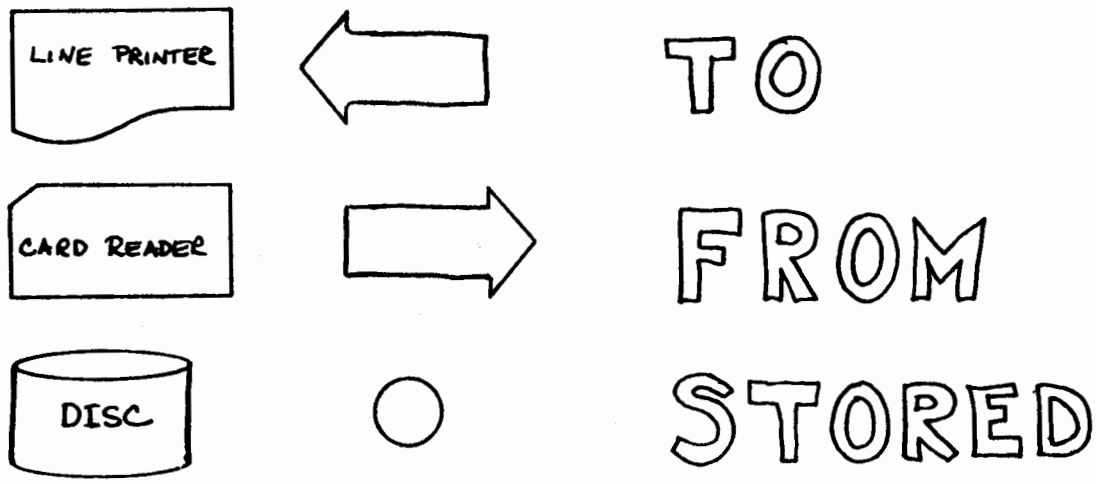
HEWLETT PACKARD

AN INTRODUCTION TO

THE MPE FILE SYSTEM

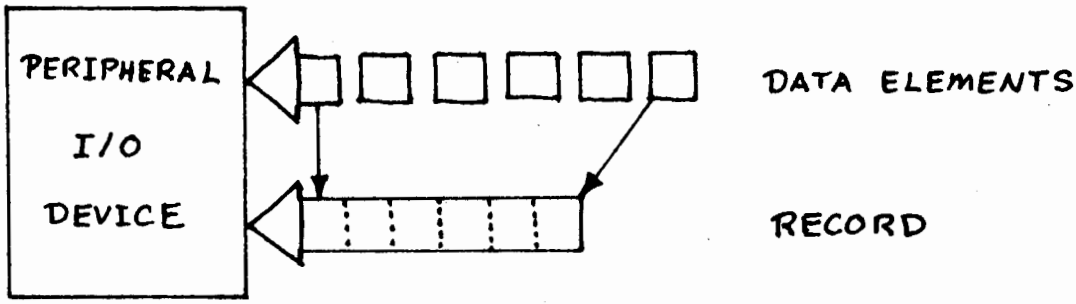
DEFINITION

FILE SYSTEM: THE PORTION OF MPE WHICH MANAGES COLLECTIONS OF DATA (FILES) BEING TRANSFERRED OR STORED WITH PERIPHERAL DEVICES



DATA TRANSFERS

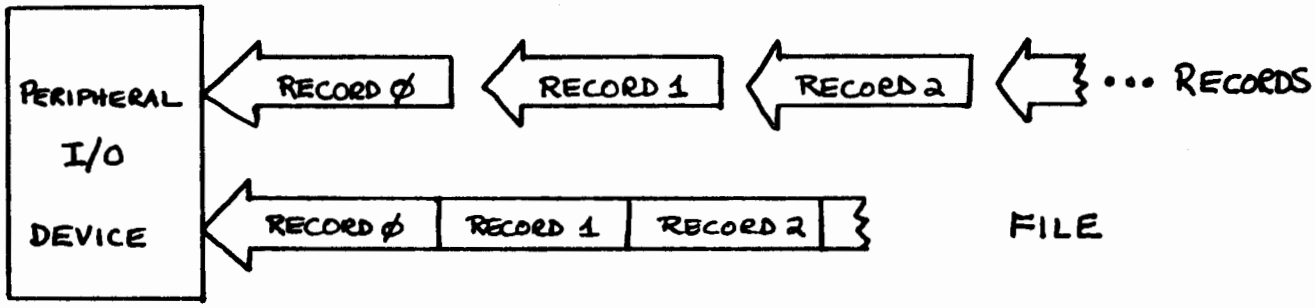
CONCEPTUALLY, DATA TRANSFERS ARE QUITE SIMPLE.



LOGICALLY RELATED DATA ELEMENTS MAY BE GROUPED INTO A RECORD FOR PURPOSES OF TRANSFER TO OR FROM THE DEVICE.

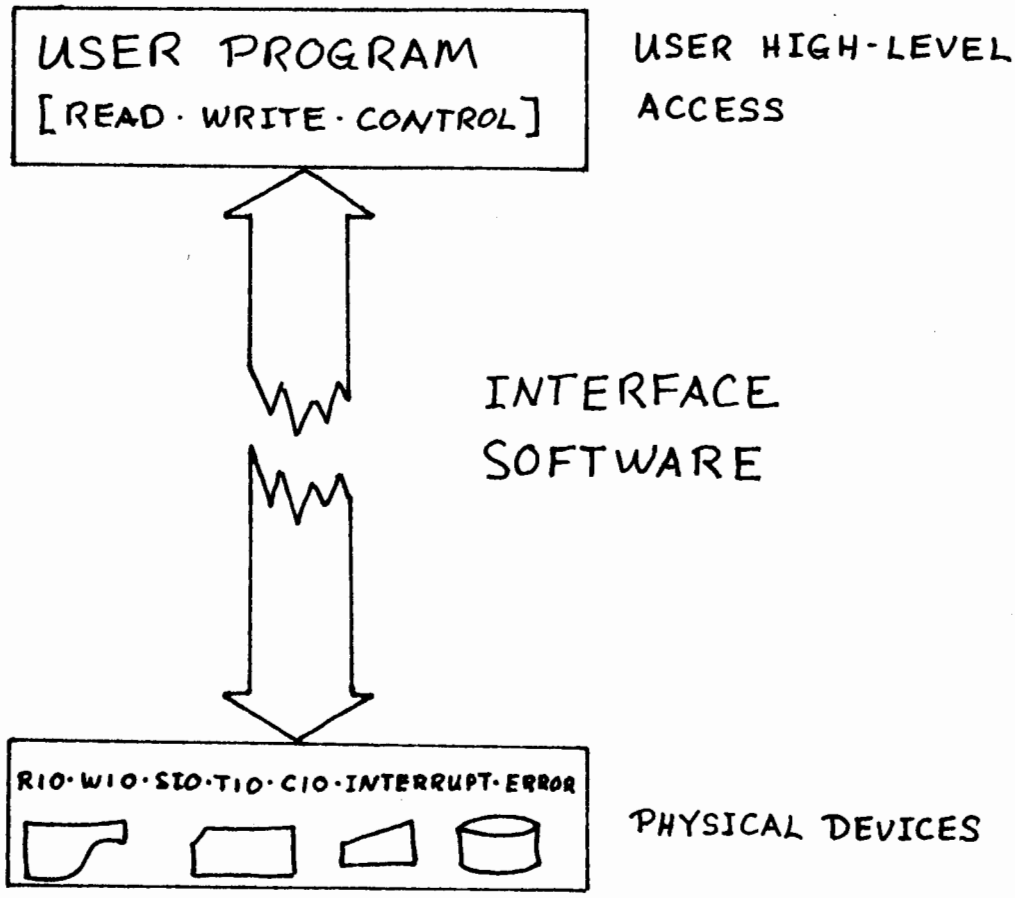
RECORD HANDLING

LOGICALLY RELATED RECORDS FORM A SET KNOWN TO THE FILE SYSTEM AS A **FILE**

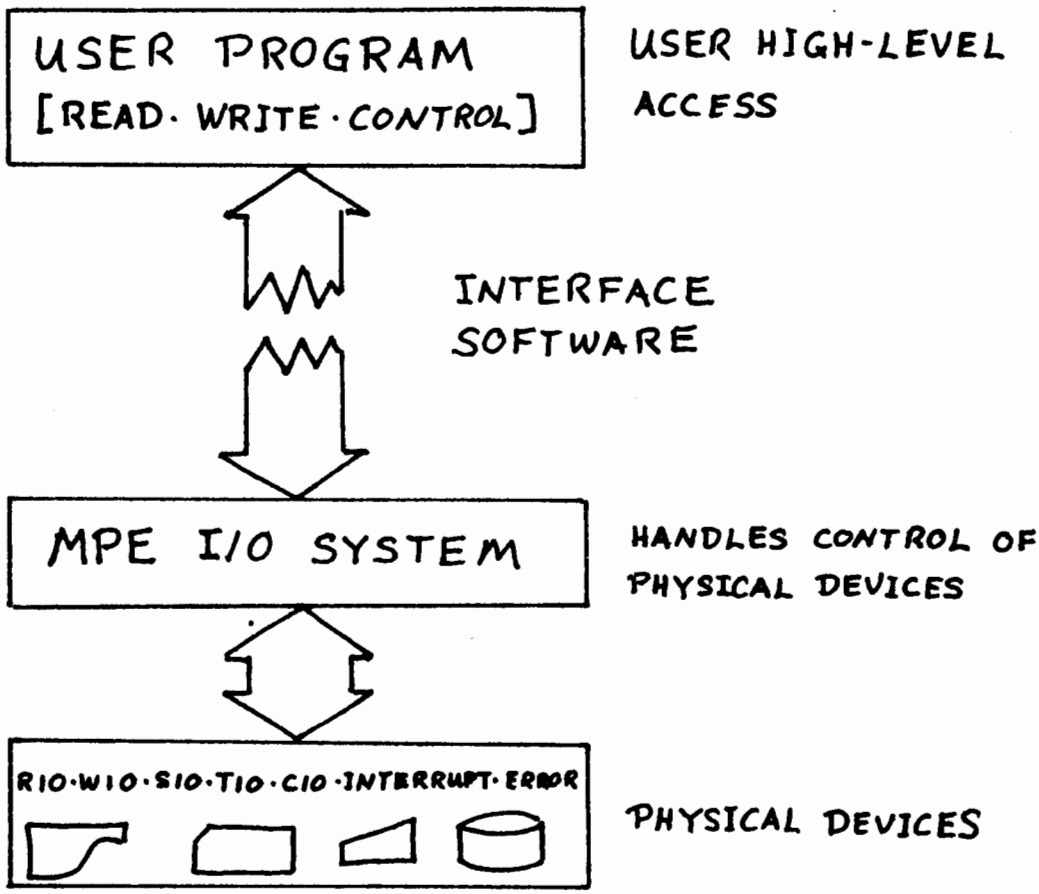


SINCE ALL I/O IS DONE THROUGH THE MECHANISM OF FILES, THE USER MAY ACCESS VERY DIFFERENT DEVICES IN A STANDARD, CONSISTENT WAY.

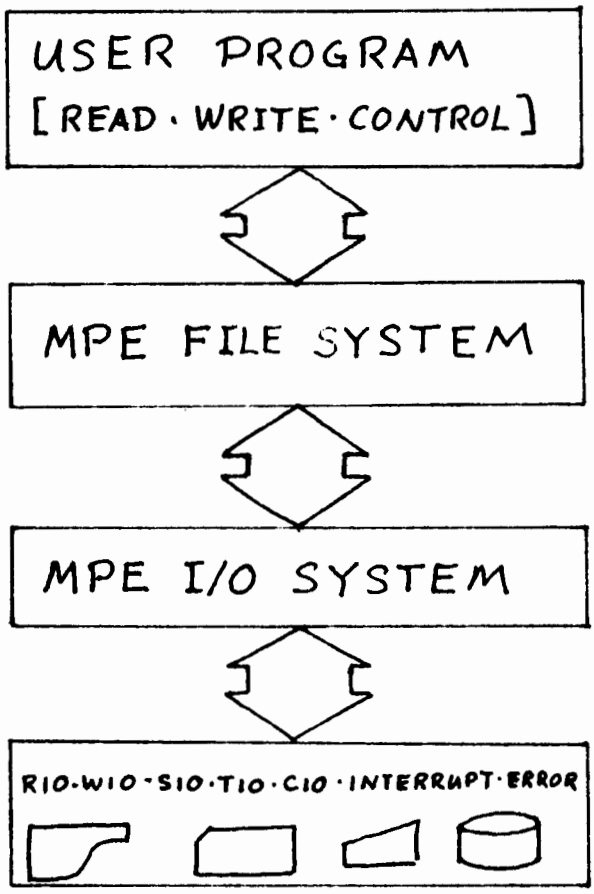
INTERFACE FUNCTION



I/O SYSTEM INTERFACE



FILE SYSTEM INTERFACE



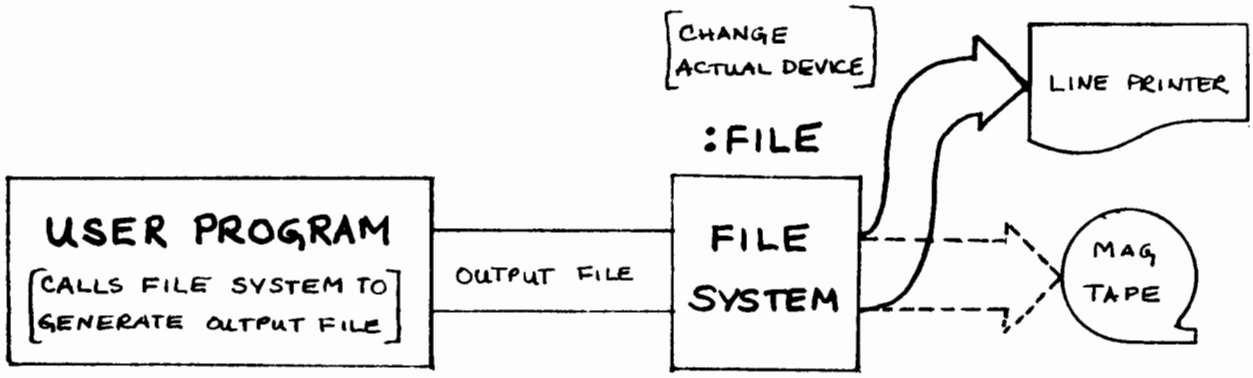
USER HIGH-LEVEL ACCESS

PERMITS USER TO DEAL WITH ALL I/O AT THE FILE LEVEL (DEVICE INDEPENDENT)

HANDLES CONTROL OF PHYSICAL DEVICES

PHYSICAL DEVICES

DEVICE INDEPENDENCE



- DATA TRANSFERS TO A LINE PRINTER ARE CODED THE SAME AS IF THEY WERE BEING MADE TO A MAG TAPE
- THE MAJOR CHARACTERISTICS OF THE DEVICE AND TYPE OF ACCESS MUST ALSO BE CONSIDERED. (E.G. CAN'T READ FROM A LINE PRINTER)

FILE SYSTEM CALLS

USER PROGRAMS DIRECT THEIR I/O OPERATIONS BY CALLING VARIOUS FILE SYSTEM ROUTINES KNOWN AS INTRINSICS

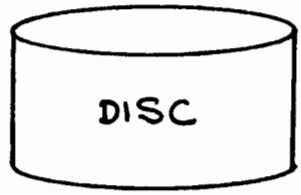
FOPEN	FCONTROL
FREAD	FCLOSE
FWRITE	ETC . . .

IF THE DEVICE CHARACTERISTICS PERMIT, THE SAME INTRINSICS MAY BE USED REGARDLESS OF

TYPE OF DEVICE -OR- CLASS OF FILE

CLASSES OF FILES

DISC FILES



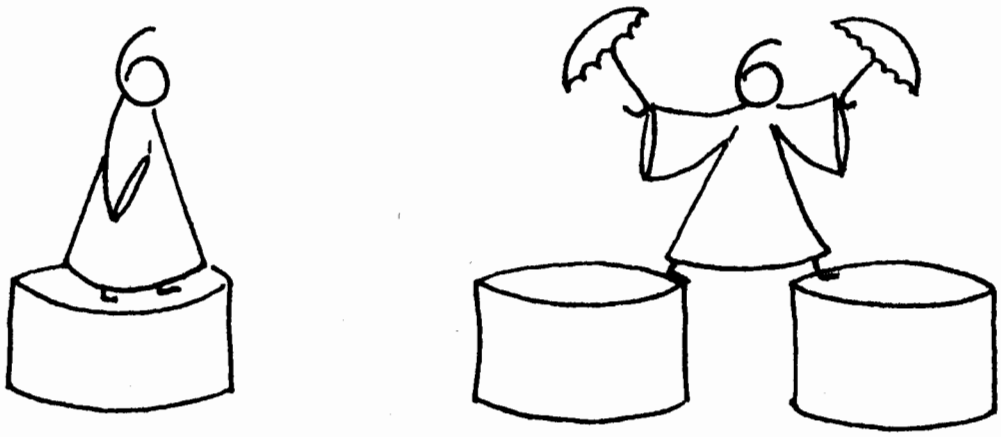
$ FILES

SPECIALLY DEFINED SYSTEM FILES [INPUT TEMP OUTPUT NULL]

DEVICE FILES

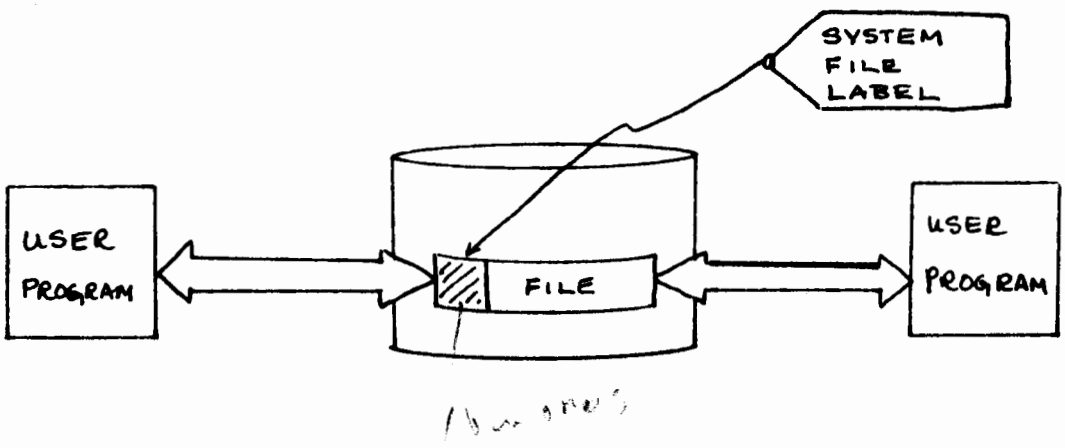
THE DISC FILE

- A SET OF LOGICALLY RELATED RECORDS WHICH RESIDES ON ONE OR MORE DEVICES

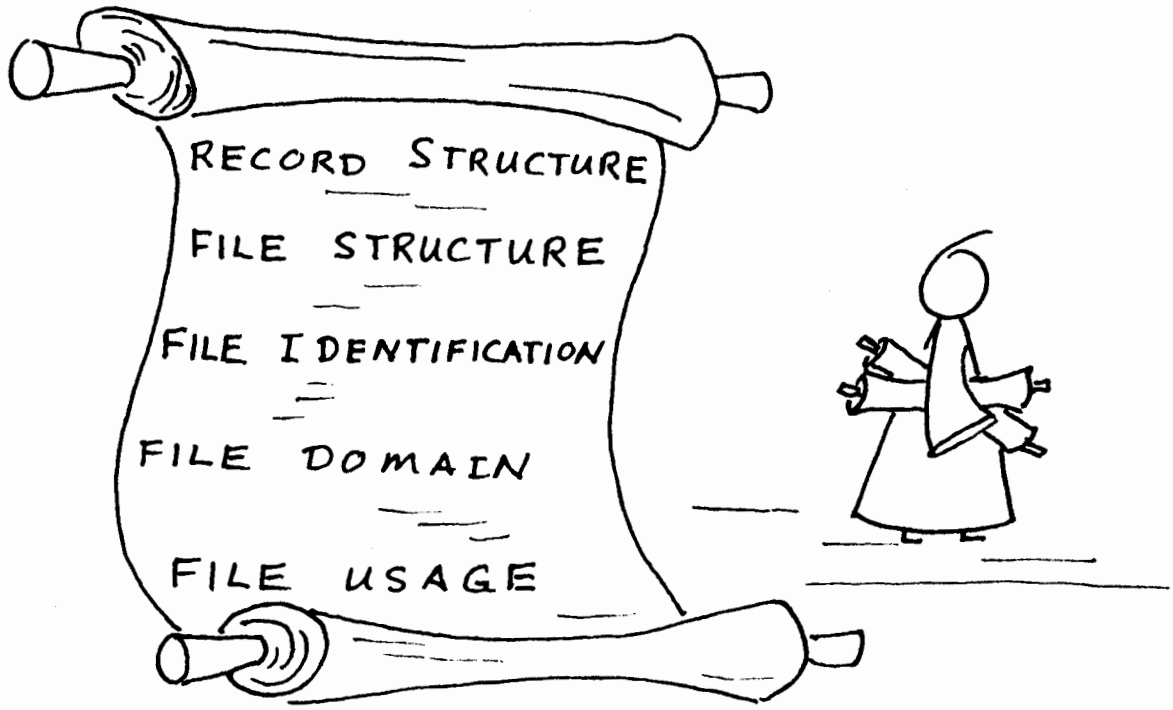


PROPERTIES OF A DISC FILE

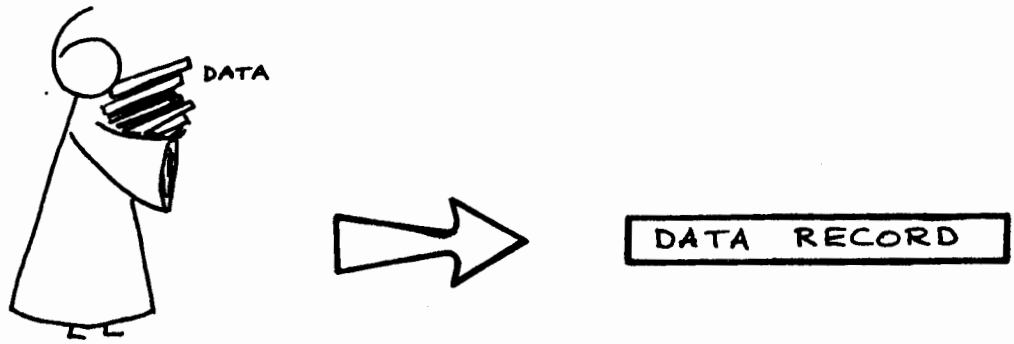
- HAS A SYSTEM LABEL WHICH IDENTIFIES PHYSICAL CHARACTERISTICS
- CAN BE ACCESSED FOR INPUT AND/OR OUTPUT
- CAN BE SHARED



DISC FILE CHARACTERISTICS



RECORD STRUCTURE



- HOW SHOULD THE DATA BE REPRESENTED?
- WILL ALL THE RECORDS BE THE SAME SIZE?
- HOW LONG WILL THE RECORDS BE?
- SHOULD THEY BE GROUPED TOGETHER FOR TRANSFER?

RECORD STRUCTURE

DATA REPRESENTATION

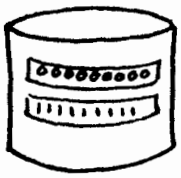
ASCII - BINARY

SOURCE FILE

- ASCII FILE PADDED WITH BLANKS (" ", %40)
- BINARY FILE PADDED WITH ZEROS (%0)

PROGRAM FILE

DATA FILES

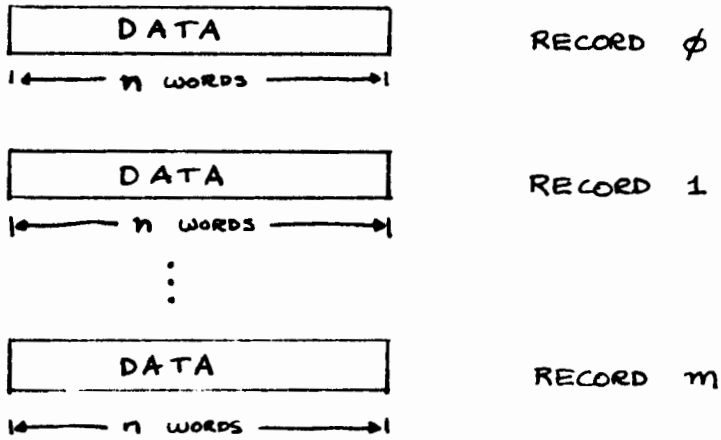


KINDS OF RECORDS

FIXED -- VARIABLE -- UNDEFINED

FIXED:

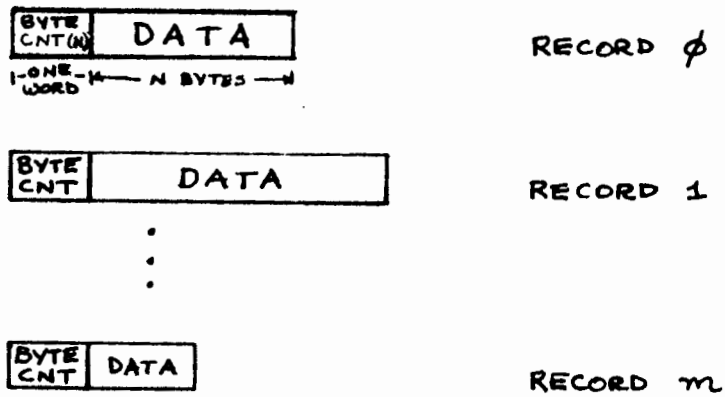
RECORD LENGTH KNOWN TO FILE SYSTEM
SAME LENGTH FOR EACH RECORD
RECORD CONSISTS OF DATA ONLY



KINDS OF RECORDS

VARIABLE:

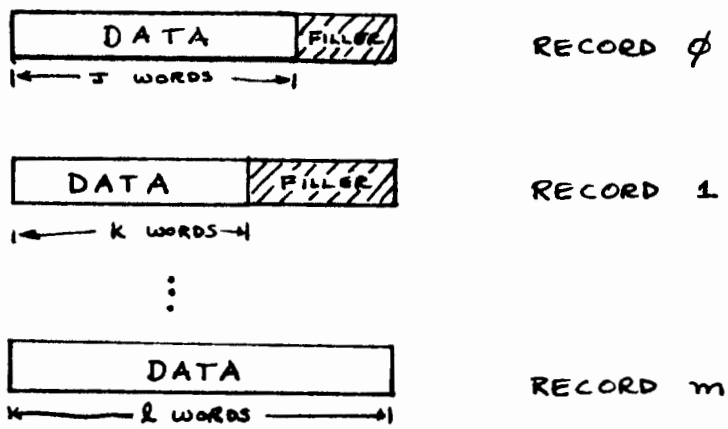
RECORD LENGTH KNOWN TO FILE SYSTEM
LENGTH VARIES FROM RECORD TO RECORD
RECORD CONTAINS LENGTH PLUS DATA



KINDS OF RECORDS

UNDEFINED:

ACTUAL RECORD LENGTH UNKNOWN TO FILE SYSTEM
LENGTH MAY VARY FROM RECORD TO RECORD
RECORD CONTAINS DATA (PLUS "FILLER")



NOTE: "FILLER" IS LAST WORD OF RECORD REPEATED TO END OF SECTOR
REMAINING SECTORS LEFT BLANK (OR ZERO)

RECORD SIZE

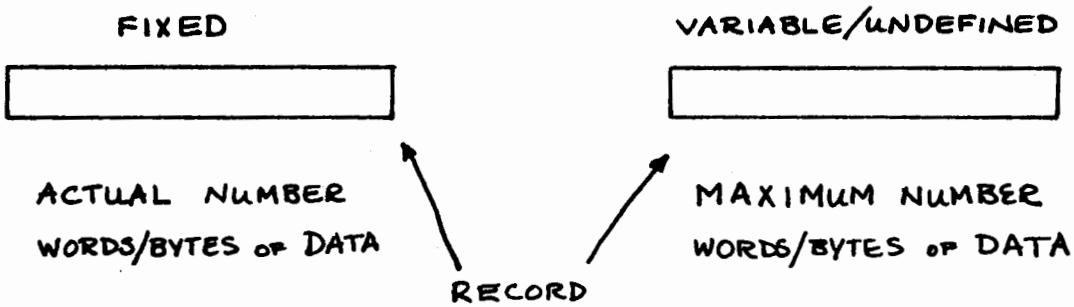
BYTES -- WORDS

- ASCII FILES: **BYTES** PER RECORD
- BINARY FILES: **WORDS** PER RECORD

72 BYTES/REC	128 WORDS/REC
BEGIN	010010111 010000100 001
⋮	
TIME:= CLOCK;	011101110111 0001100101
⋮	
END.	000110011100010101010
ASCII	BINARY

RECORD SIZE

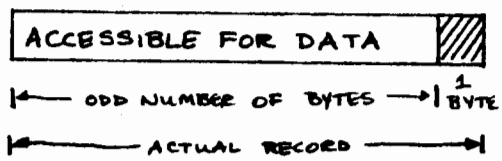
- FIXED : **ACTUAL** RECORD LENGTH
- VARIABLE : **MAXIMUM** ALLOWABLE RECORD LENGTH
- UNDEFINED : **MAXIMUM** ALLOWABLE RECORD LENGTH



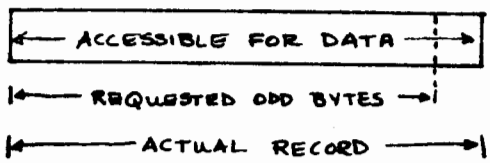
RECORD SIZE

- RECORDS **ALWAYS** BEGIN ON **WORD** BOUNDARIES
- ODD BYTE LENGTHS ROUNDED UP

FIXED/UNDEFINED



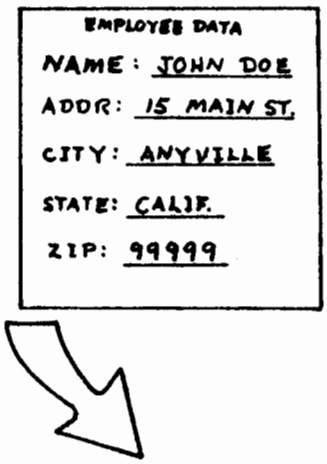
VARIABLE

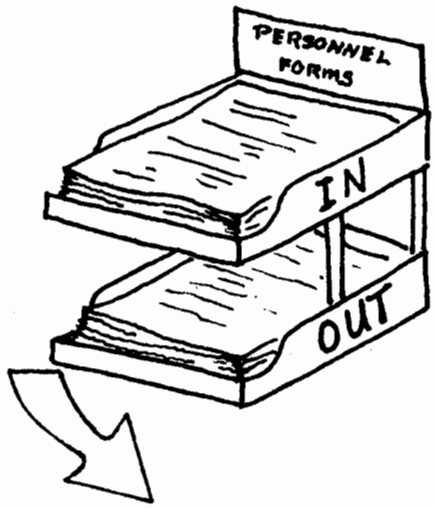


RECORDS PER BLOCK

LOGICAL GROUP OF DATA

PHYSICAL TRANSFER OF DATA





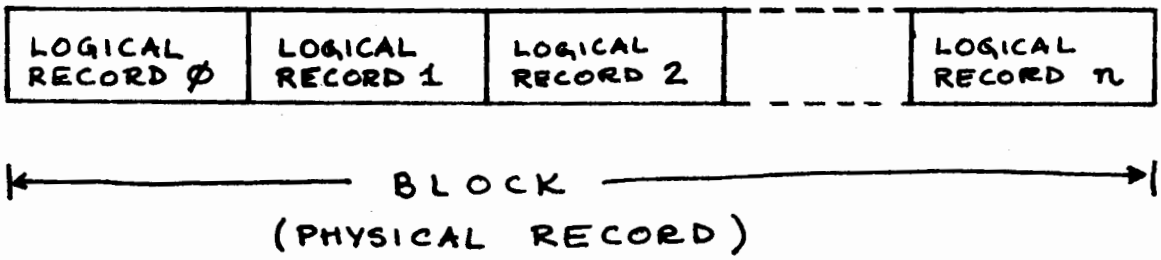
LOGICAL RECORD

PHYSICAL RECORD (BLOCK)

RECORDS PER BLOCK

PHYSICAL RECORD (BLOCK)

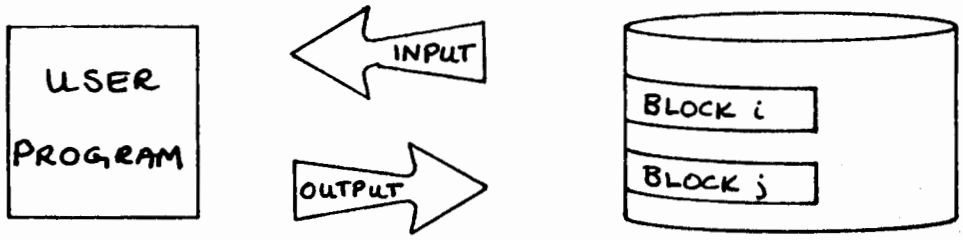
CONSISTS OF ONE OR MORE LOGICAL RECORDS



RECORDS PER BLOCK

PHYSICAL TRANSFER OF DATA

INPUTS/OUTPUTS A PHYSICAL RECORD (BLOCK)



PROPER GROUPING OF LOGICAL RECORDS INTO BLOCKS RESULTS IN

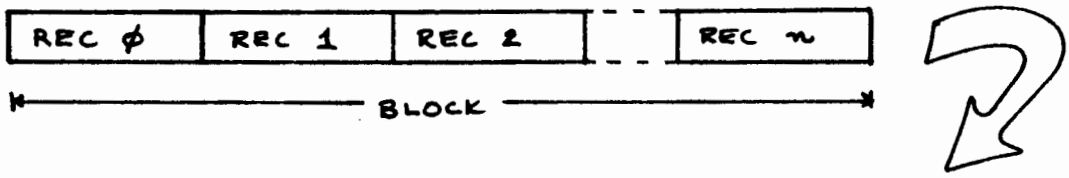
- FEWER DISC ACCESSES
- BETTER DISC SPACE UTILIZATION

IF 16 (MAX) BLOCKS SPECIFIED
FILES THEN [illegible] PHYSICAL [illegible] AS SOON
AS [illegible] BLOCKS
BUFFER [illegible] THEN [illegible].

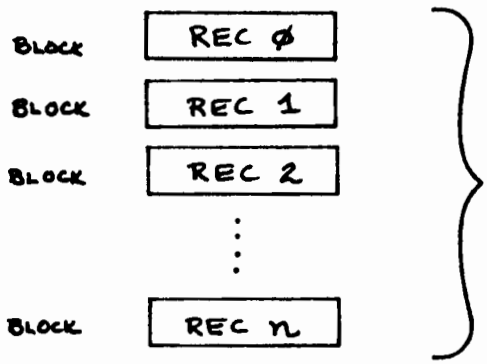
(ALL THE [illegible] ARE [illegible])

RECORDS PER BLOCK

DISC ACCESS CONSIDERATIONS



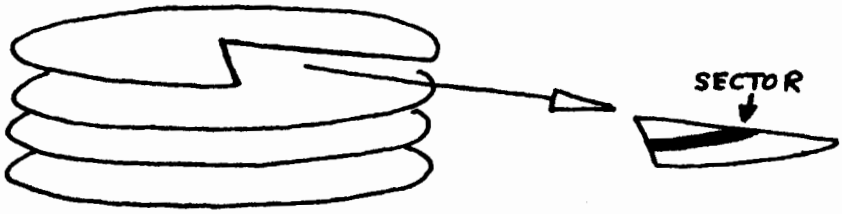
1 PHYSICAL RECORD
1 DISC TRANSFER



n PHYSICAL RECORDS
n DISC TRANSFERS

RECORDS PER BLOCK

DISC SPACE UTILIZATION



- DISCS ARE NOT WORD ADDRESSABLE
- SMALLEST ADDRESSABLE PIECES ARE CALLED SECTORS
- EACH SECTOR IS 128 WORDS LONG
- ALL PHYSICAL TRANSFERS BEGIN ON A SECTOR BOUNDARY

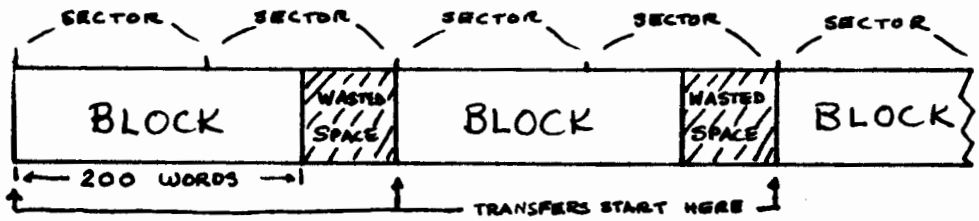
NOTE- A HARDWARE WRITE MUST BE A MULTIPLE OF 128 WORDS
A HARDWARE READ MAY BE ANY LENGTH

RECORDS PER BLOCK

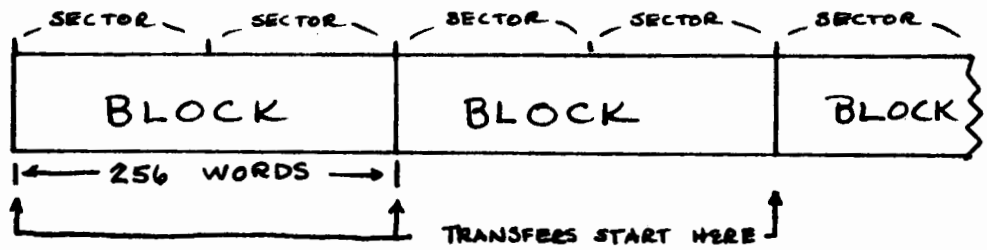
DISC SPACE UTILIZATION

MOST EFFICIENT BLOCK LENGTHS: MULTIPLE OF 128 WORDS

CASE 1: 200 WORD BLOCK (SPACE WASTED)



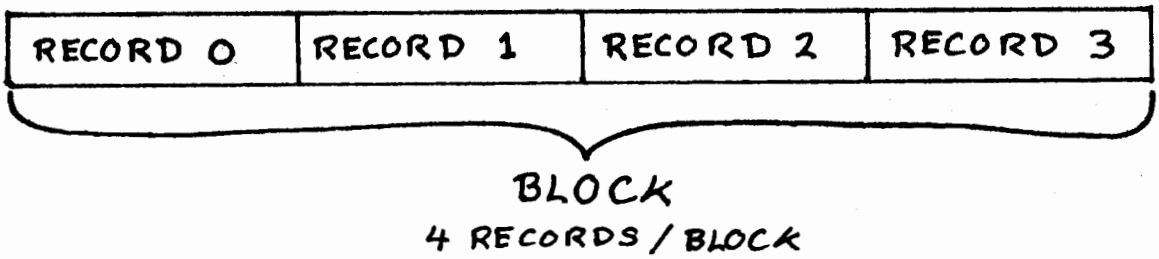
CASE 2: 256 WORD BLOCK (NO SPACE WASTED)



RECORDS PER BLOCK

FIXED

BLOCK SIZE = RECORD SIZE × RECORDS PER BLOCK



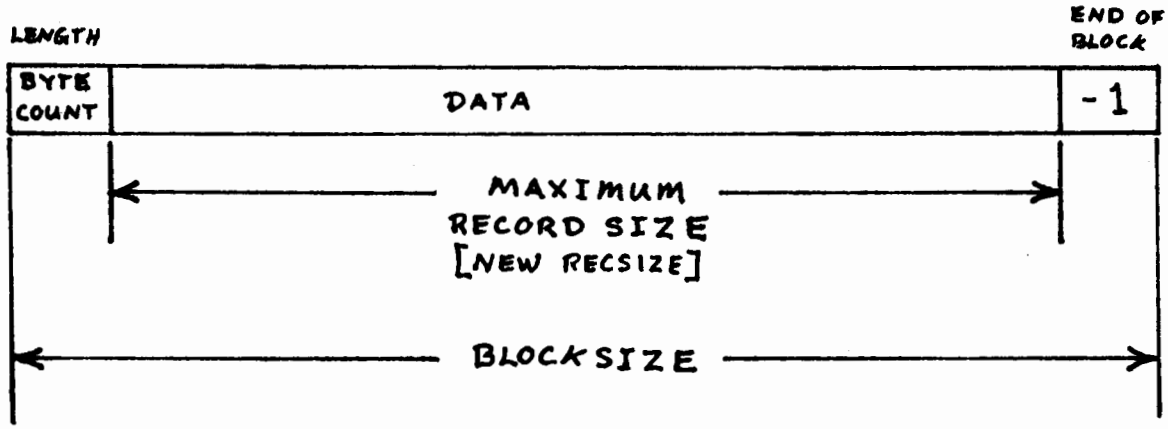
BLOCKSIZE SHOULD BE AS CLOSE TO (BUT LESS THAN OR EQUAL TO) A MULTIPLE OF 128 WORDS.

RECORDS PER BLOCK

VARIABLE

FOPEN MAY CHANGE RECSIZE AND BLOCKFACTOR

- NEW RECSIZE = RECSIZE × BLOCKFACTOR
- NEW BLOCKFACTOR = 1

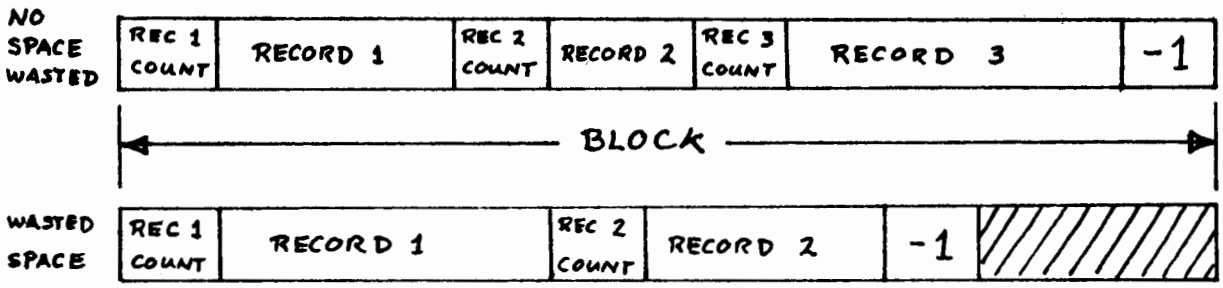


BLOCKSIZE = (NEW) RECSIZE + 2 [BLOCKSIZE IN WORDS]

RECORDS PER BLOCK

VARIABLE

- BLOCKSIZE SHOULD BE MULTIPLE OF 128 WORDS TO AVOID PERMANENT WASTE OF SECTOR SPACE IN FILE
- RECORD SPACE IN BLOCK MAY BE WASTED IF RECORDS FIT POORLY



SPACE WAS WASTED BECAUSE RECORD 3 WAS TOO BIG TO BE CONTAINED IN THE REMAINING BLOCK SPACE.

RECORDS PER BLOCK

UNDEFINED

FOPEN CHANGES BLOCKSIZE AND BLOCK FACTOR

- NEW BLOCKSIZE = RECSIZE
- NEW BLOCK FACTOR = 1

LOGICAL RECORDS ARE IDENTICAL TO PHYSICAL RECORDS

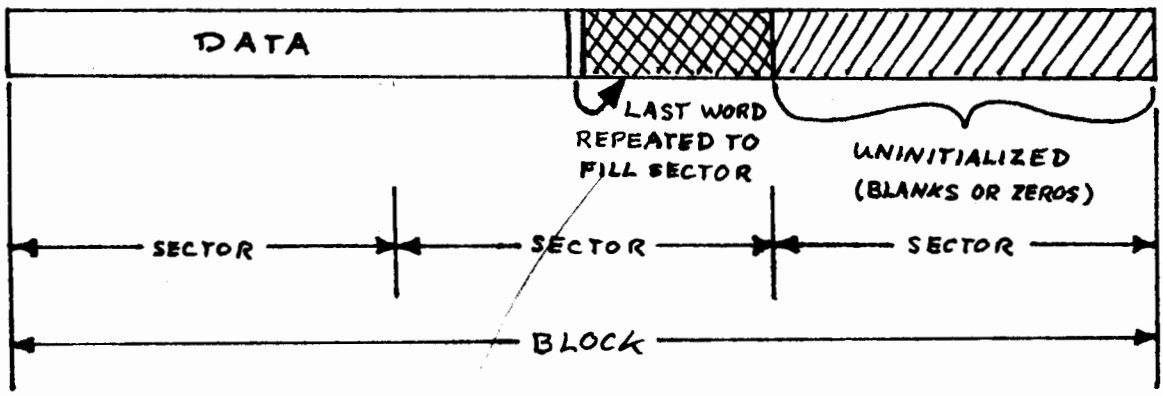
[UNKNOWN RECORD LENGTH MAKES BLOCKING IMPOSSIBLE]

PHYSICAL RECORD

RECORDS PER BLOCK

UNDEFINED

- BLOCKSIZE (RECSIZE) SHOULD BE MULTIPLE OF 128 WORDS TO AVOID PERMANENT WASTE OF SECTOR SPACE IN FILE
- RECORD SPACE IN BLOCK WASTED BY SHORT RECORDS



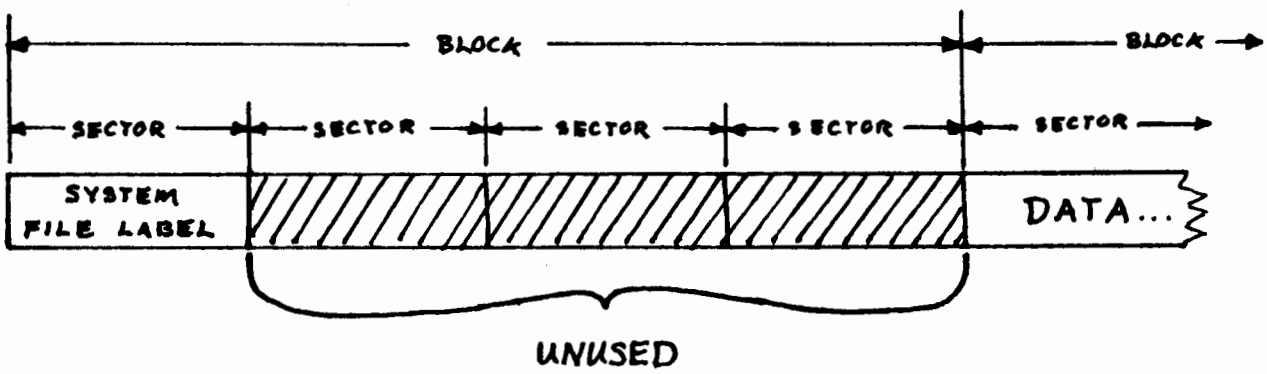
RECORDS PER BLOCK - FILE LABEL

SYSTEM FILE LABEL

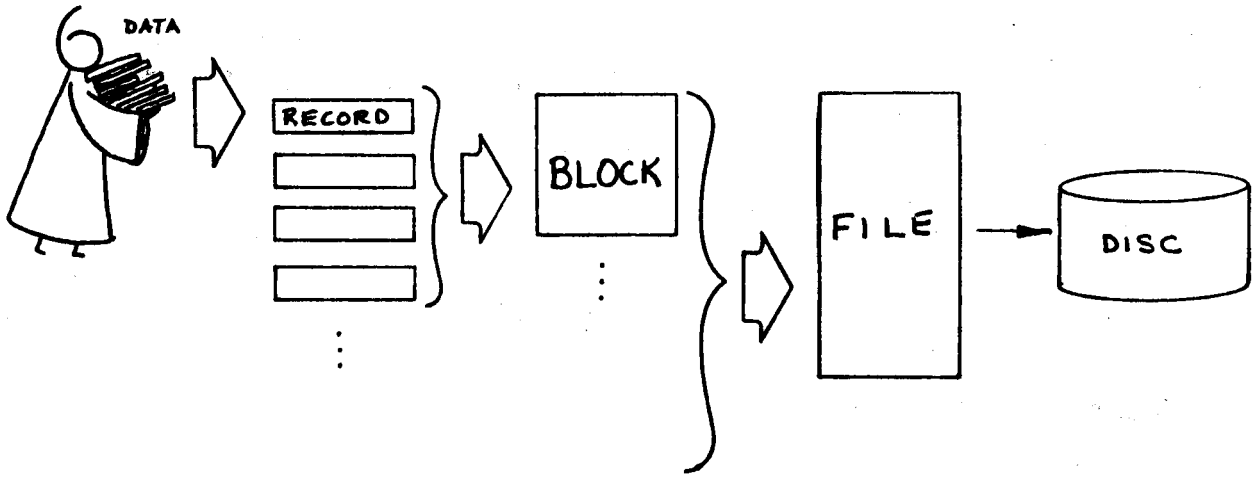
- OCCUPIES 1 BLOCK
- FILE LABEL SIZE = 128 WORDS

EXAMPLE:

BLOCKSIZE = 512 WORDS

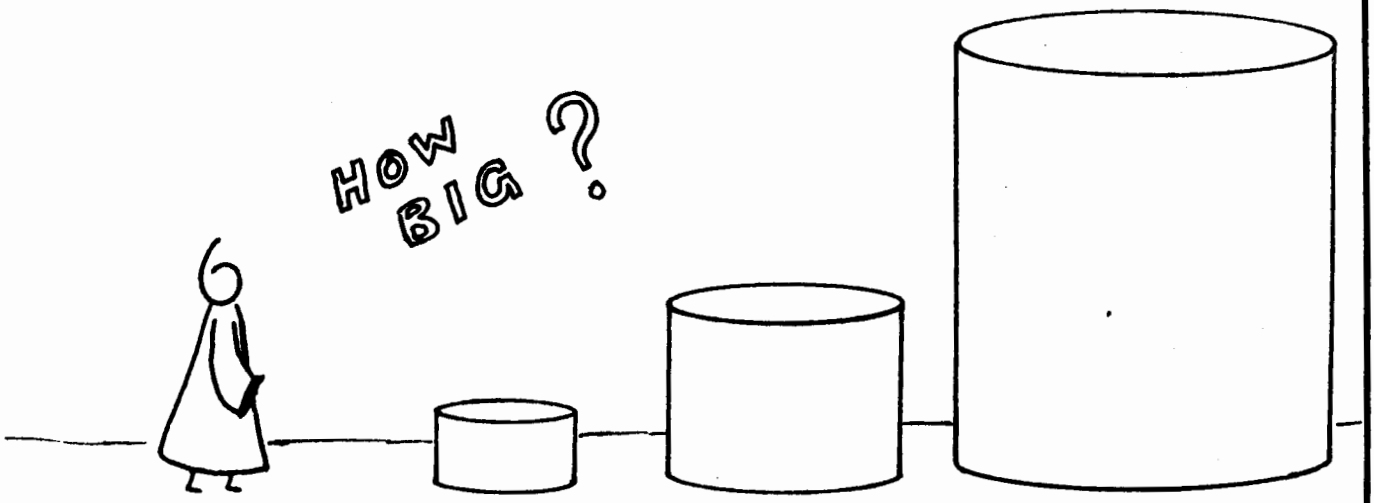


FILE STRUCTURE



- HOW BIG WILL THE FILE BE ?
- WHICH DISC SHOULD BE USED ?
- HOW SHOULD THE FILE BE DISTRIBUTED ON THE DISC?
- HOW MUCH SPACE IS NEEDED AT THE START ?

RECORDS PER FILE



- MAKE FILE AS LARGE (OR LARGER) THAN NEEDED
- CAN MAKE EXISTING FILE SMALLER, NOT LARGER

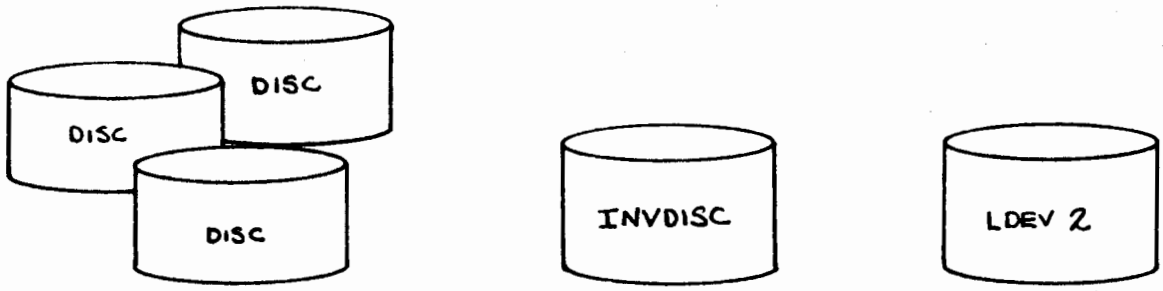
CAN SHRINK FILE

BY FORCE.

[illegible] (MOKUP)

ONLY USING [illegible]

DEVICE SELECTION

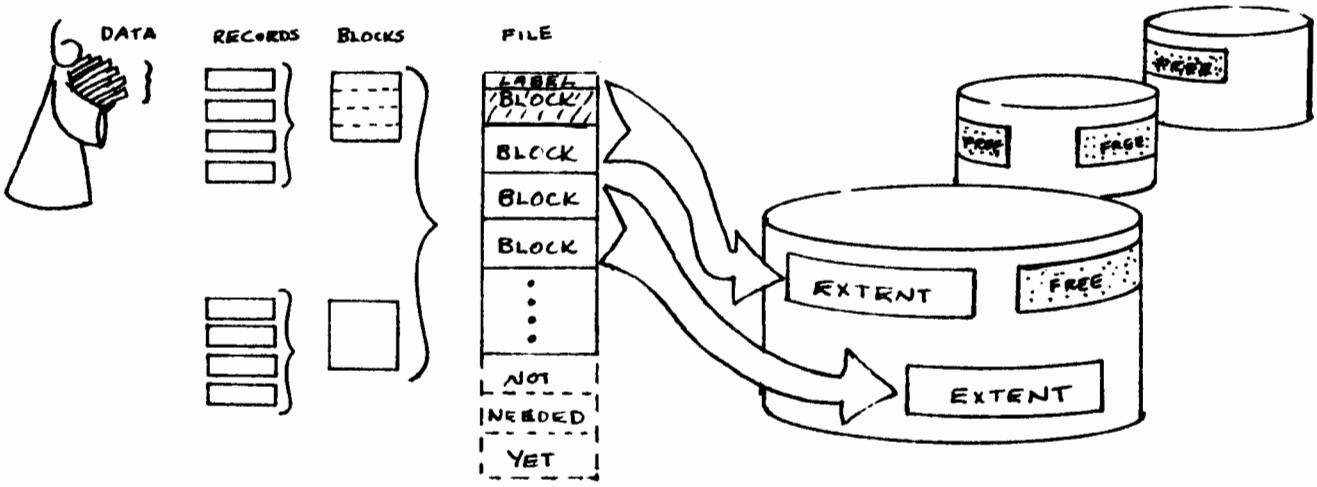


CLASS OF DEVICES
"DISC", "INVDISC"

LOGICAL DEVICE NUMBER (2)

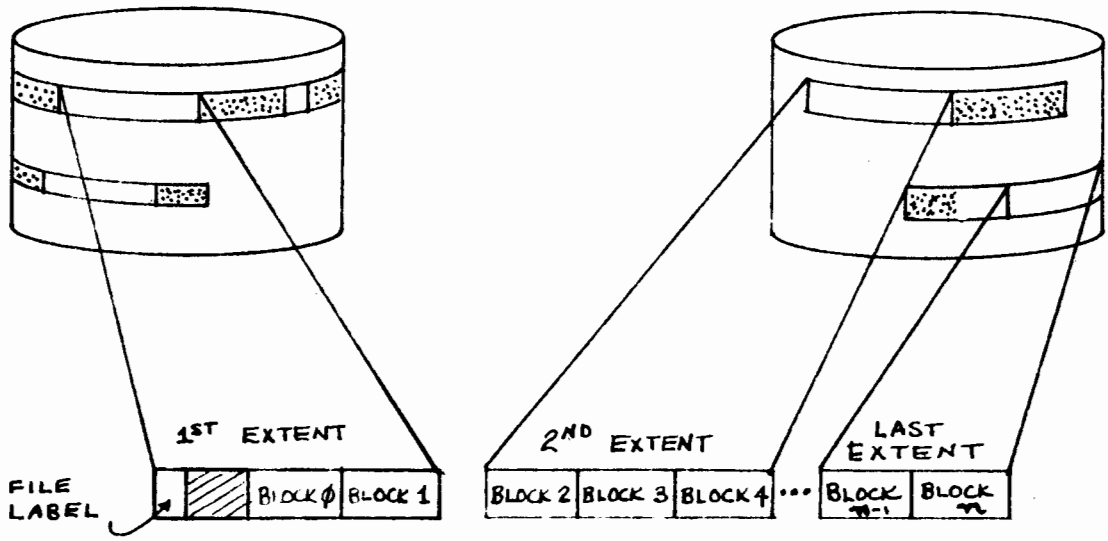
DEVICE SPECIFICATION	DEVICE SELECTION	FILE LOCATION
DEVICE CLASS	BASED ON LAST DISC IN CLASS USED BY ANYONE IN THE SYSTEM, SELECT THE NEXT DISC IN THE CLASS AND BEGIN SPACE ALLOCATION. [CYCLE TO NEXT DISC IF NECESSARY TO CONTINUE INIT. ALLOC.]	SPACE ALLOCATIONS MAY SPREAD FILE OVER ONE OR MORE DISCS IN CLASS
LOGICAL DEVICE NUMBER	USE DEVICE SPECIFIED	FILE MUST RESIDE ENTIRELY ON THE SPECIFIED DEVICE.

FILE PLACEMENT



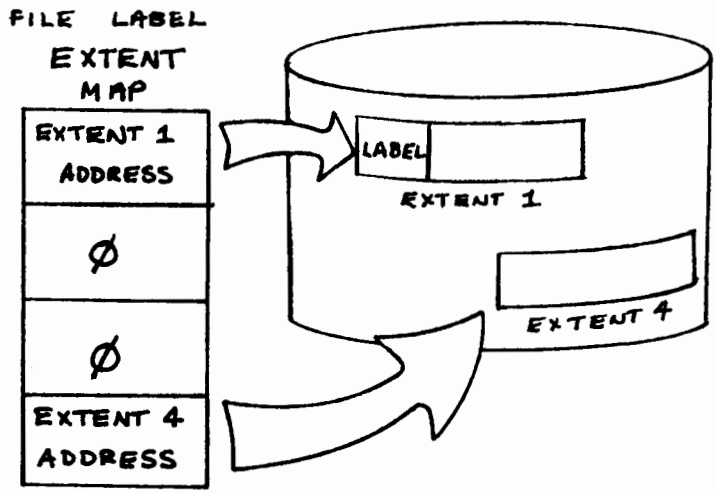
- FREE DISC SPACE NOT CONTIGUOUS
- FILE BROKEN INTO INDIVIDUALLY PLACEABLE PIECES (EXTENTS)
- SPACE CAN BE ALLOCATED AS REQUIRED

INDIVIDUALLY PLACEABLE PIECES



- CAN SPECIFY MAXIMUM NUMBER OF EXTENTS (TO 32)
- EXTENT WILL BE INTEGRAL NUMBER OF BLOCKS
- LAST EXTENT MAY BE SHORTER THAN REST

EXTENT ALLOCATION



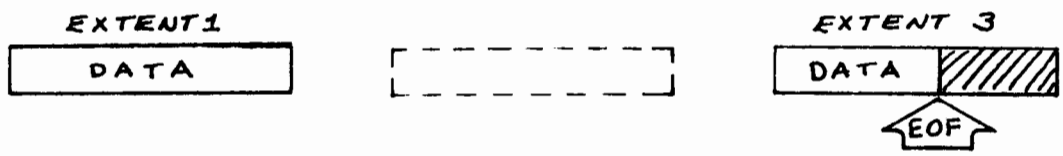
EXTENTS NEED NOT BE ALLOCATED IN ORDER

- INITIAL ALLOCATION : EXTENTS ALLOCATED IN ORDER (1,2,3...)
- OUTPUT TO UNALLOCATED BLOCK FORCES ALLOCATION OF THAT EXTENT
- INPUT FROM UNALLOCATED BLOCK FORCES ALLOCATION AND INITIALIZATION OF THAT EXTENT

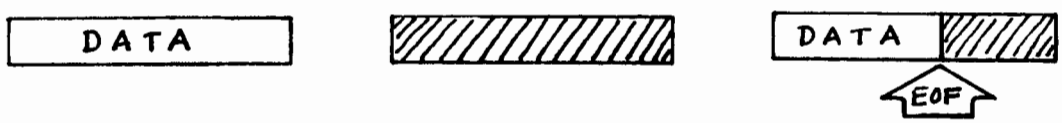
EXTENT INITIALIZATION

THE FILE SYSTEM MUST NOT ALLOW UNINITIALIZED SPACE TO BE READ

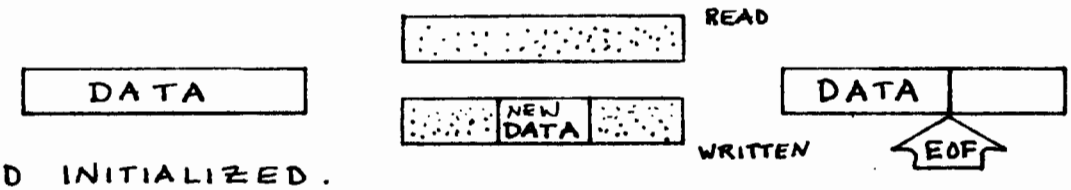
CASE I.



WHEN A RECORD IS READ FROM OR WRITTEN TO AN UNALLOCATED EXTENT .....



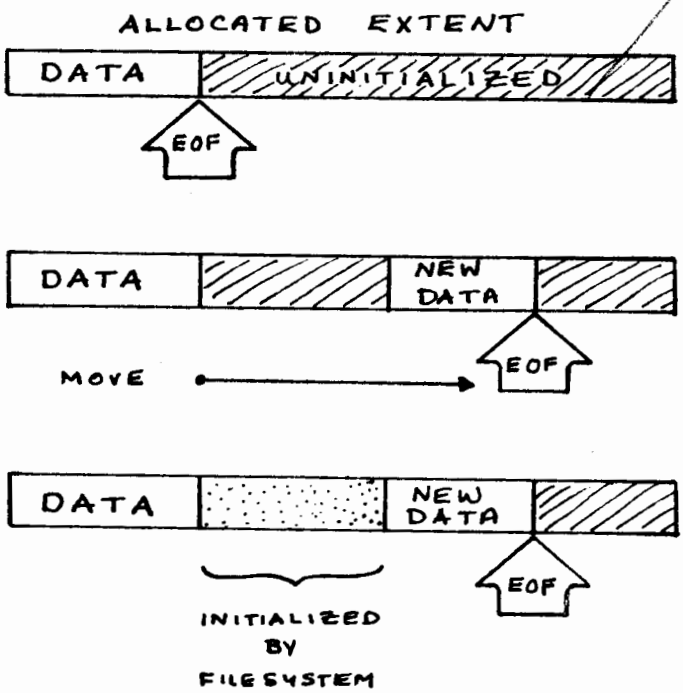
THE EXTENT IS ALLOCATED .....



AND INITIALIZED.

EXTENT INITIALIZATION

CASE II.

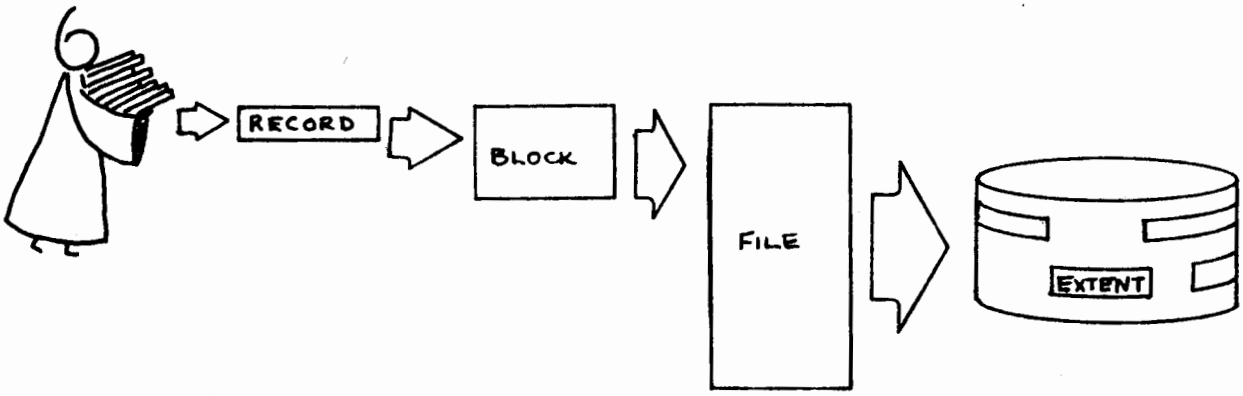


WHEN EOF IS MOVED ....

UNINITIALIZED BLOCKS NOW INCLUDED IN FILE (< EOF) ....

MUST BE INITIALIZED.

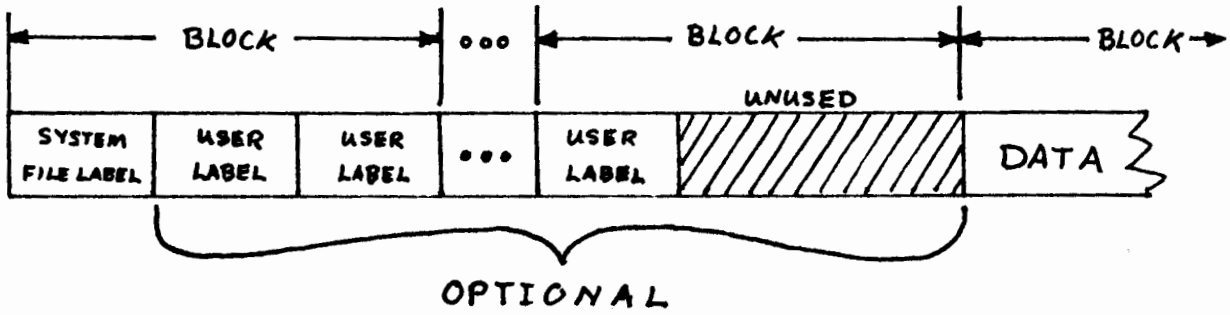
FILE IDENTIFICATION



- IS SPECIAL NON-DATA STORAGE REQUIRED?
- DOES THE FILE NEED A SPECIAL TYPE CODE ASSOCIATED WITH IT?
- SHOULD THE FILE HAVE A NAME?

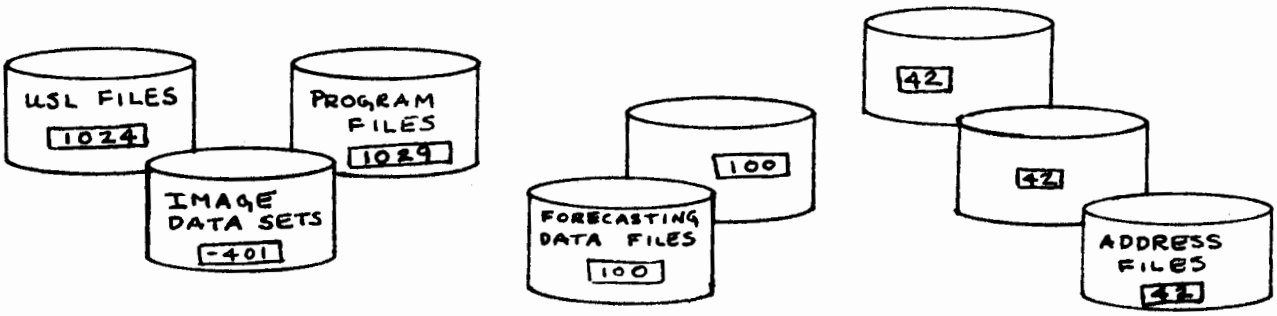
NON-DATA STORAGE - USER LABELS

- MAXIMUM OF 255 USER LABELS
- EACH LABEL IS 128 WORDS LONG
- BEGIN IN SAME BLOCK AS SYSTEM FILE LABEL
- LABELS AND DATA CANNOT SHARE A BLOCK

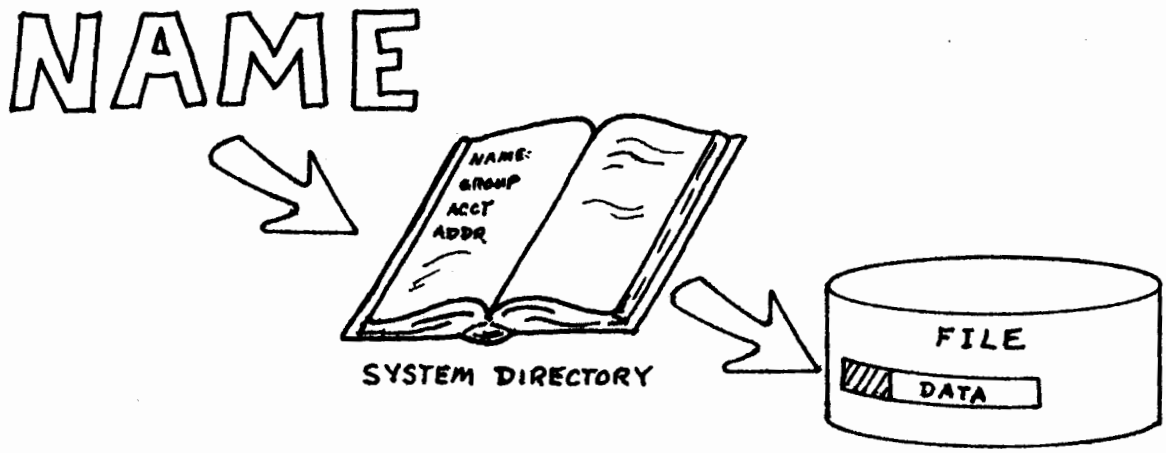


TYPE CODE (FILE CODE)

- IDENTIFIES GROUP OR CLASS OF FILES
- CODE KEPT IN SYSTEM FILE LABEL
- USER CODE CAN BE IN RANGE 0 TO 1023
- PRIVILEGED USERS CAN ASSIGN NEGATIVE CODE (E.G. IMAGE)



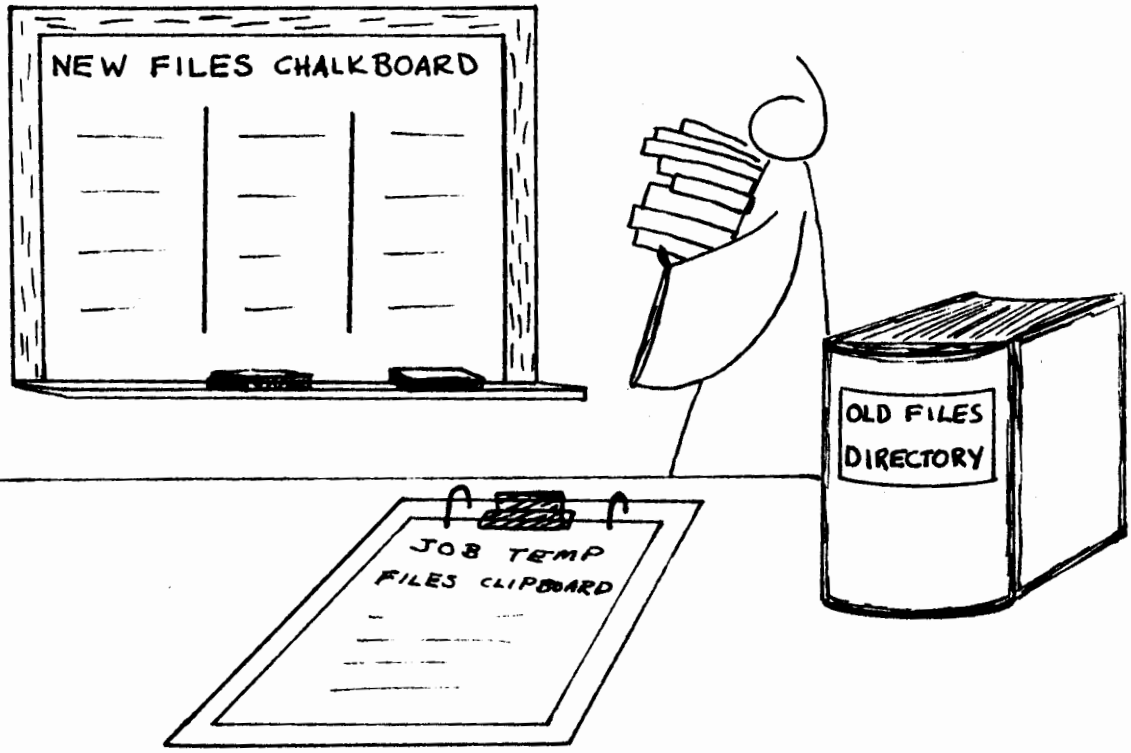
FILE NAME



- USEFUL FOR FILE IDENTIFICATION IN SYSTEM
- REQUIRED IF :FILE COMMANDS ARE TO BE USED
- UP TO 8 ALPHANUMERIC CHARACTERS (START WITH ALPHA)
- MAY BE QUALIFIED WITH GROUP, ACCOUNT, LOCKWORD

FILE DOMAIN

WHERE DOES THE FILE "LIVE"?



DOMAIN

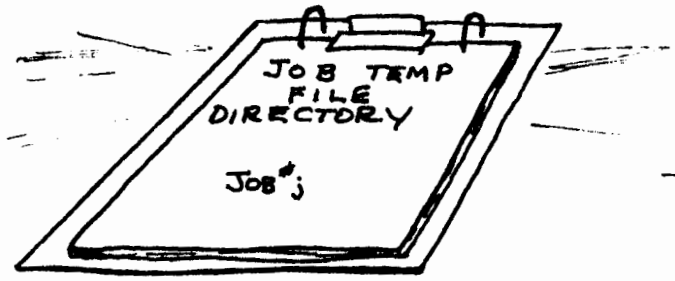
(PROCESS TEMPORARY UNTIL CLOSED)

NEW: DOES NOT YET EXIST
SPACE TO BE ALLOCATED
PHYSICAL CHARACTERISTICS TO BE DEFINED

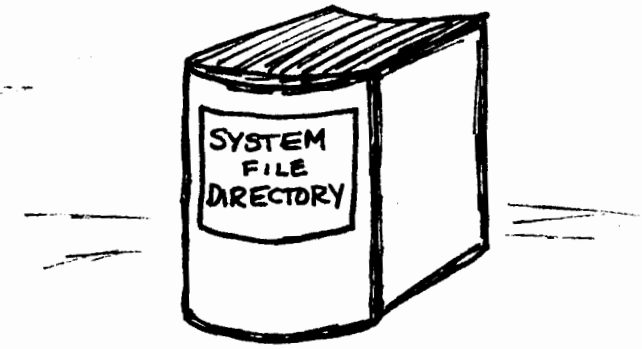
OLD: EXISTS AS PERMANENT FILE IN SYSTEM
SPACE ALLOCATED ALREADY (SOME OR ALL)
PHYSICAL CHARACTERISTICS DEFINED

TEMP: EXISTS AS JOB TEMPORARY FILE
SPACE ALLOCATED ALREADY (SOME OR ALL)
PHYSICAL CHARACTERISTICS DEFINED

DOMAIN -- DIRECTORY SEARCH



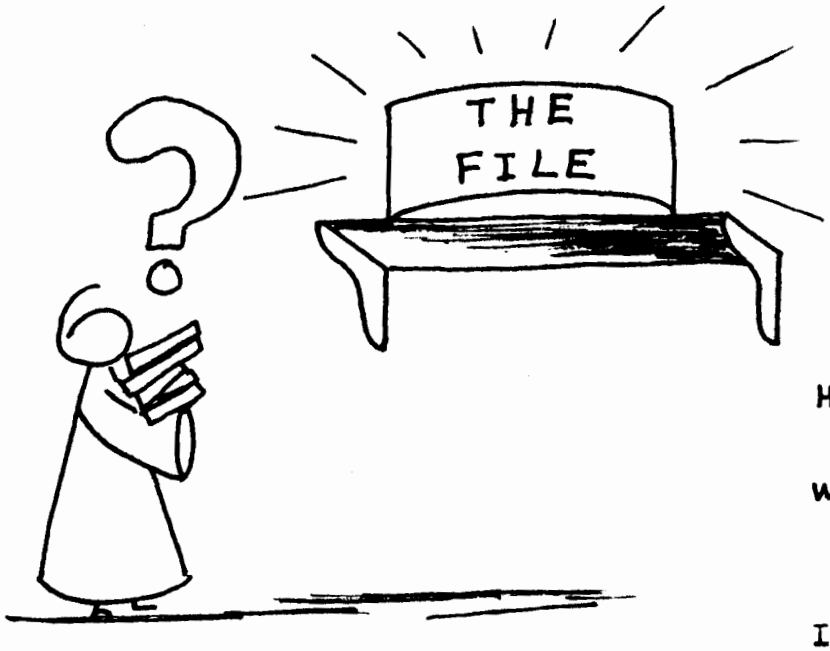
SEARCH FOR
TEMP FILES



SEARCH FOR
OLD FILES

- NO DIRECTORY SEARCH FOR NEW FILES
- WHEN BOTH DIRECTORIES SEARCHED, **JOB TEMPORARY** FILE DIRECTORY IS SEARCHED **FIRST**

FILE USAGE



HOW WILL THE FILE BE USED?

WILL THERE BE ANY FILE BUFFERING?

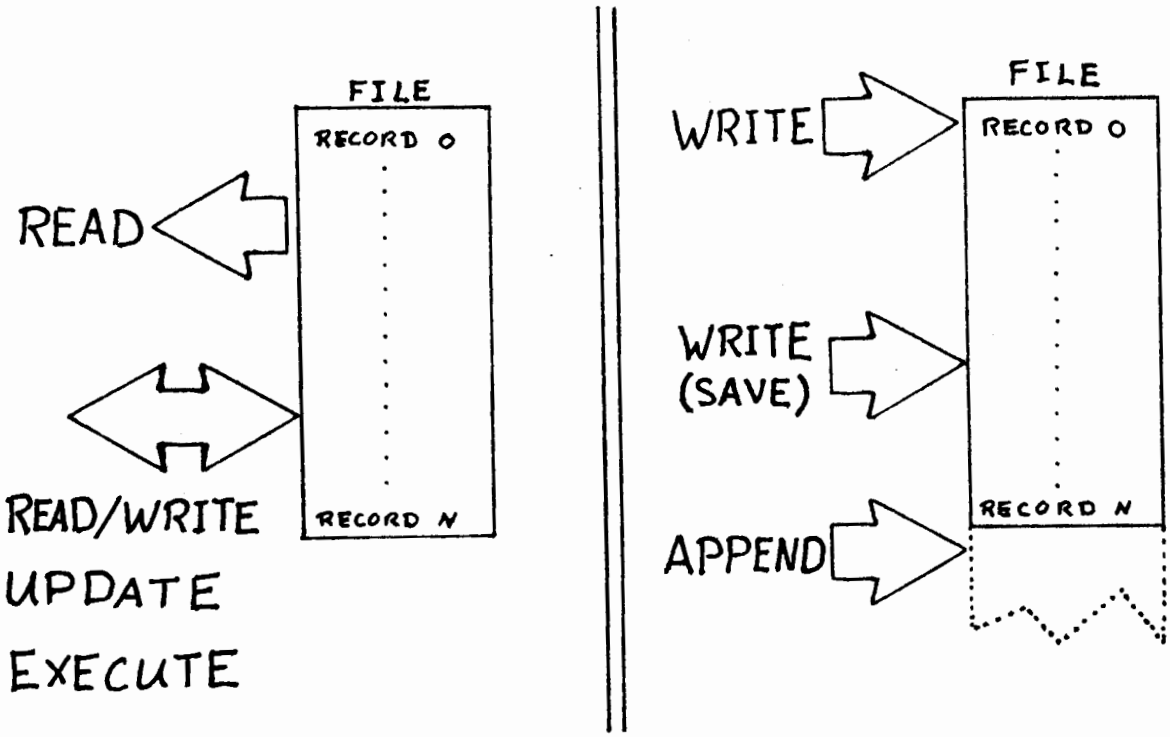
IF SO, HOW MANY BUFFERS WILL BE USED?

WILL OTHERS BE ALLOWED CONCURRENT ACCESS?

WILL THE CONCURRENT ACCESS NEED SPECIAL MANAGEMENT?

ARE THERE SPECIAL FEATURES REQUIRED TO ACCESS THE FILE?

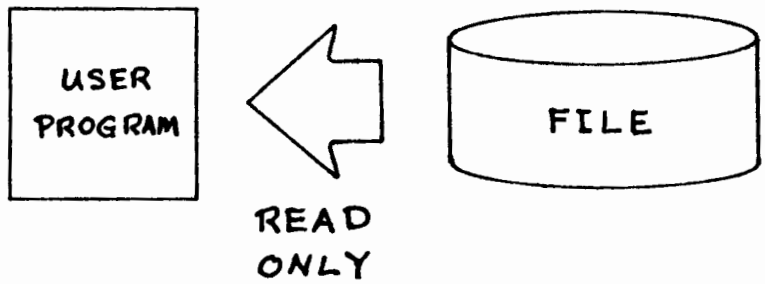
TYPE OF OPERATION



TYPE OF OPERATION

READ

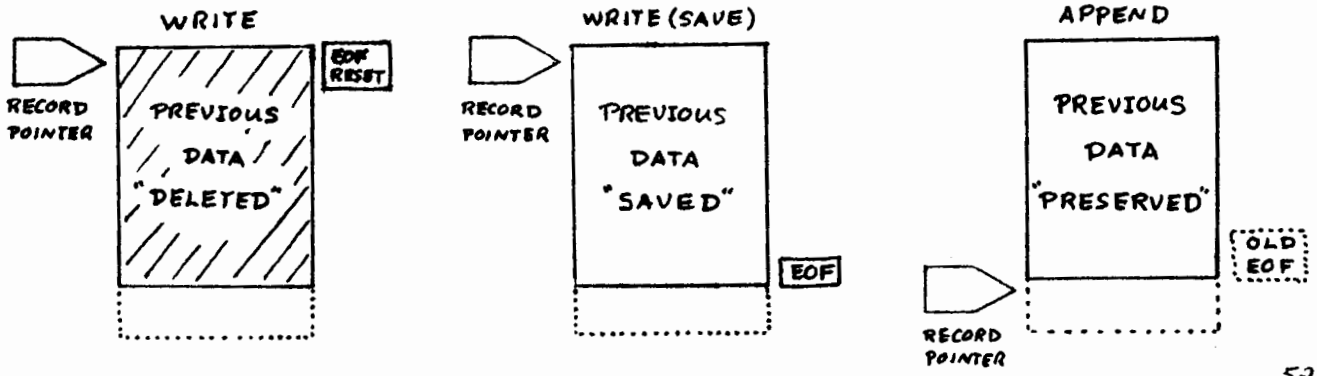
- FILE CAN BE USED FOR INPUT ONLY
- NOT USEFUL FOR A NEW (EMPTY) FILE
- CANNOT CHANGE END OF FILE



TYPE OF OPERATION

WRITE – WRITE (SAVE) APPEND

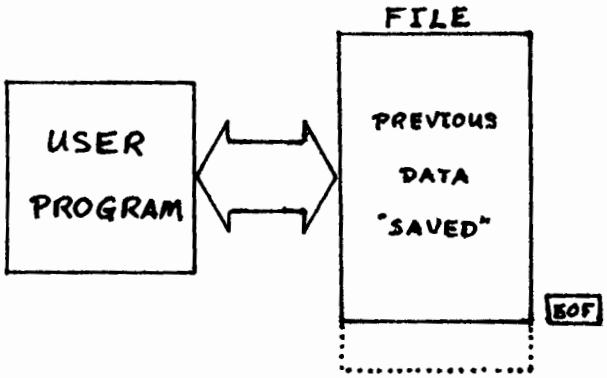
- FILE CAN BE USED FOR OUTPUT ONLY
- CAN BE USED WITH NEW, OLD, OR TEMP FILES
- END OF FILE CAN BE CHANGED



TYPE OF OPERATION

READ/WRITE-UPDATE

- FILE CAN BE USED FOR INPUT AND OUTPUT
- CAN BE USED WITH NEW, OLD, OR TEMP FILES
- END OF FILE CAN BE CHANGED



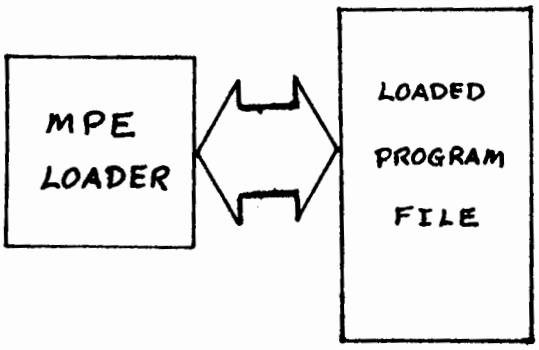
NOTE:

<u>ONLY</u> UPDATE OPTION ALLOWS USE OF FUPDATE INTRINSIC

TYPE OF OPERATION

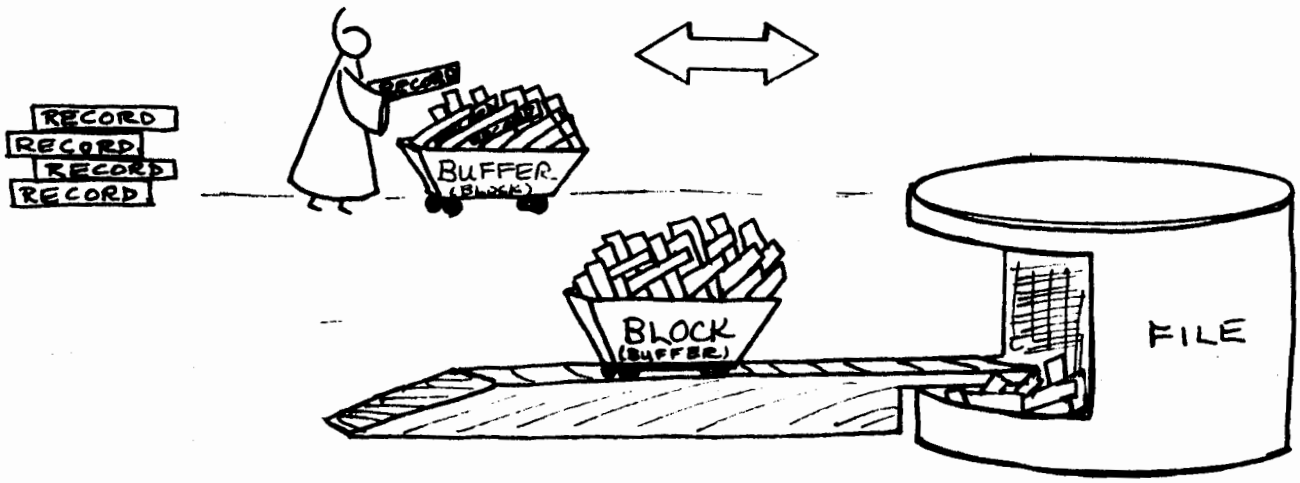
EXECUTE

- SPECIAL OPTION REQUIRED BY MPE LOADER
- ALLOWS READ/WRITE OPERATION e.g. [illegible]
- GRANTS ACCESS TO A LOADED PROGRAM FILE (REQUIRES PRIVILEGED MODE)



DESIGNED TO MAINTAIN INTEGRITY OF AN EXECUTING PROGRAM FILE

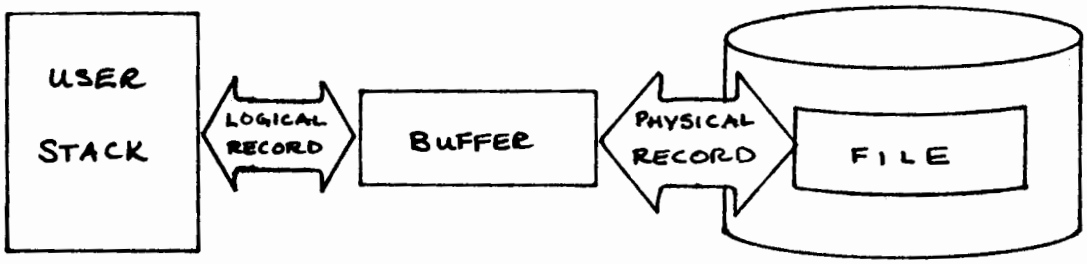
BUFFERING CONSIDERATIONS



BUFFERED I/O

BUFFERING CONSIDERATIONS

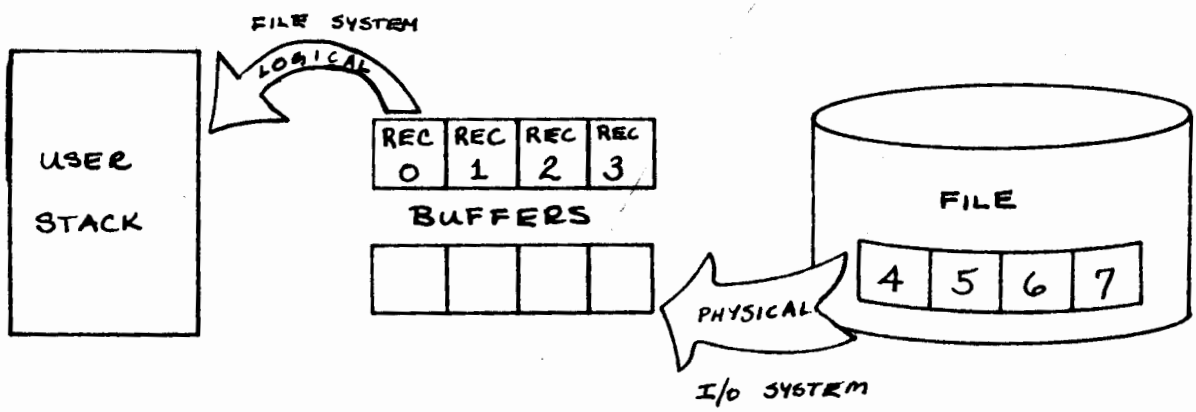
- BUFFER : AREA NOT IN USER STACK
  MAINTAINED BY FILE SYSTEM
- I/O TRANSFERS MADE TO/FROM BUFFER AREA



BUFFERING CONSIDERATIONS

WHY "BUFFER" TRANSFERS?

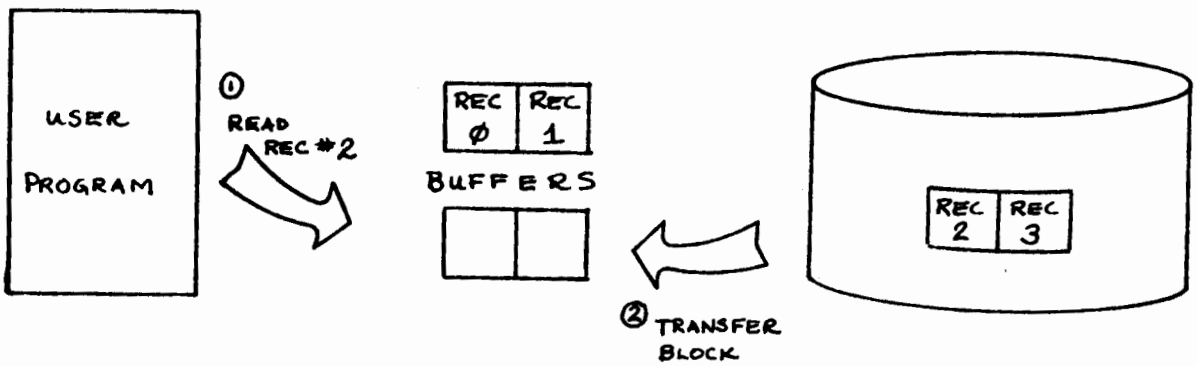
- AUTOMATIC BLOCKING/DEBLOCKING (LOGICAL RECORDS)
- ANTICIPATORY READS (SEQUENTIAL)
- PARALLELISM (USER PROGRAM, I/O PROCESS)



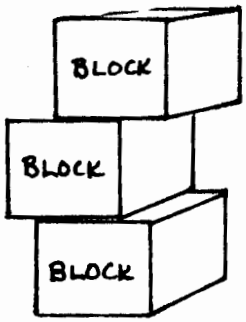
BUFFERING CONSIDERATIONS

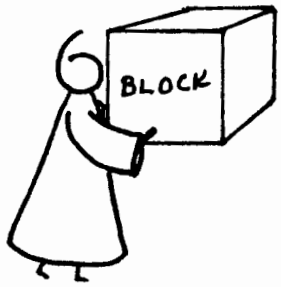
RANDOM ACCESS

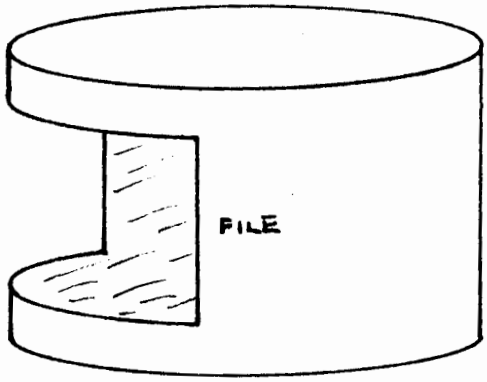
- AUTOMATIC BLOCKING/DEBLOCKING (LOGICAL RECORDS)
- ANTICIPATORY READS NOT AUTOMATIC



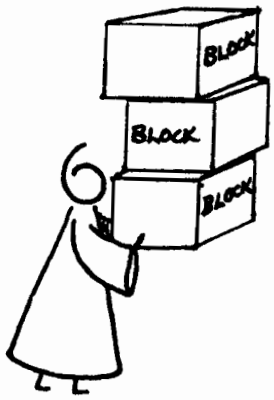
BUFFERING CONSIDERATIONS







UNBUFFERED I/O
[BLOCK TRANSFERS]

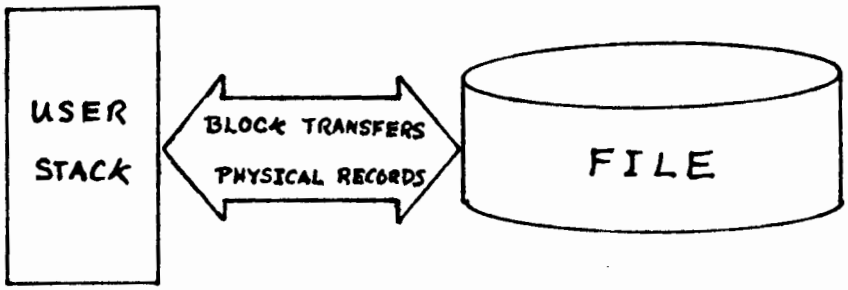


MULTI RECORD I/O
[MULTIPLE BLOCK TRANSFERS]

BUFFERING CONSIDERATIONS

DIRECT TRANSFERS

- NO BUFFERS - TRANSFER BETWEEN USER STACK AND DISC
- USER SUSPENDS - WAIT FOR TRANSFER COMPLETION
- PHYSICAL RECORDS ONLY - FILE SYSTEM WILL NOT BLOCK OR UNBLOCK LOGICAL RECORDS



BUFFERING CONSIDERATIONS

HOW MANY BUFFERS?

0 (NOBUF) : SUSPEND ON EVERY TRANSFER
STACK FROZEN IN MEMORY
CAN ONLY TRANSFER PHYSICAL RECORDS

1 : SUSPEND WHEN LOGICAL RECORD NOT IN BUFFER
STACK NOT FROZEN IN MEMORY

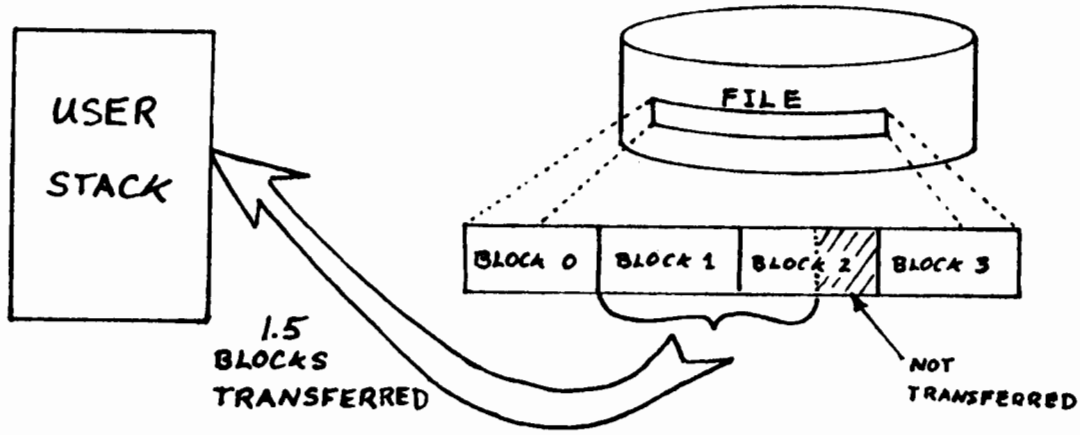
2 : MAY NOT SUSPEND - ALLOWS PARALLEL PROCESSING
BUFFER USAGE ALTERNATES

3 (OR MORE) : MAY NOT SUSPEND EVEN UNDER HEAVY I/O LOAD
USEFUL FOR LOCAL SET OF FREQUENTLY ACCESSED RECORDS

BUFFERING CONSIDERATIONS

MULTI RECORD OPTION

- ALLOWS READING/WRITING MULTIPLE PHYSICAL RECORDS
- NOBUF IS REQUIRED
- NO AUTOMATIC BLOCKING/UNBLOCKING DONE



WRITTEN AS [illegible]

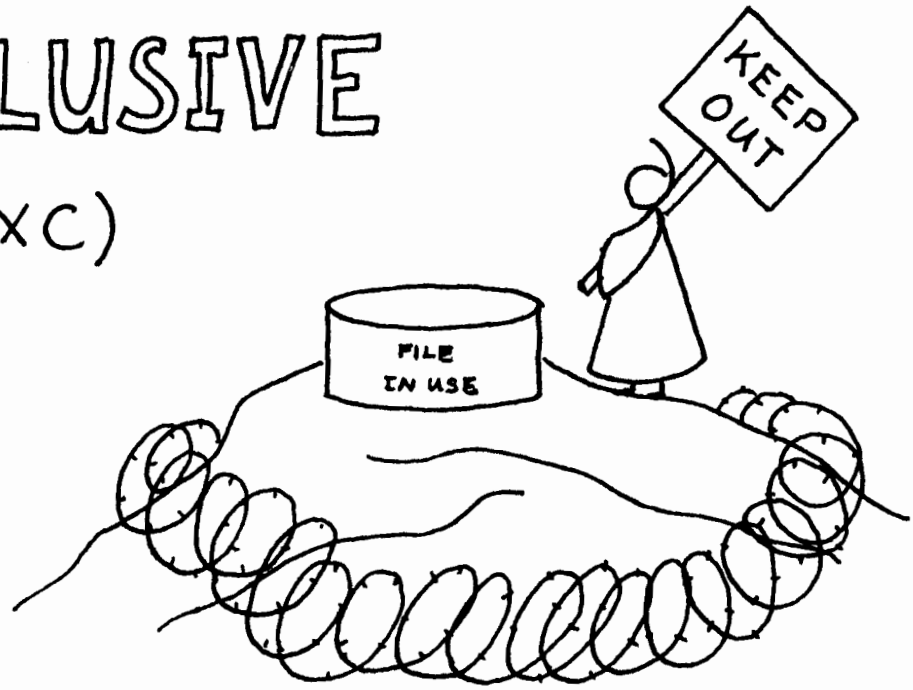
: FILE X ; R/C = -20, [illegible]
[illegible]
[illegible]
[illegible]
2nd [illegible]

: FILE X ; R/C = -20, 4, [illegible] ; MR
1st record = |A A|
2nd record = |B-B|

FILE ACCESSIBILITY

EXCLUSIVE

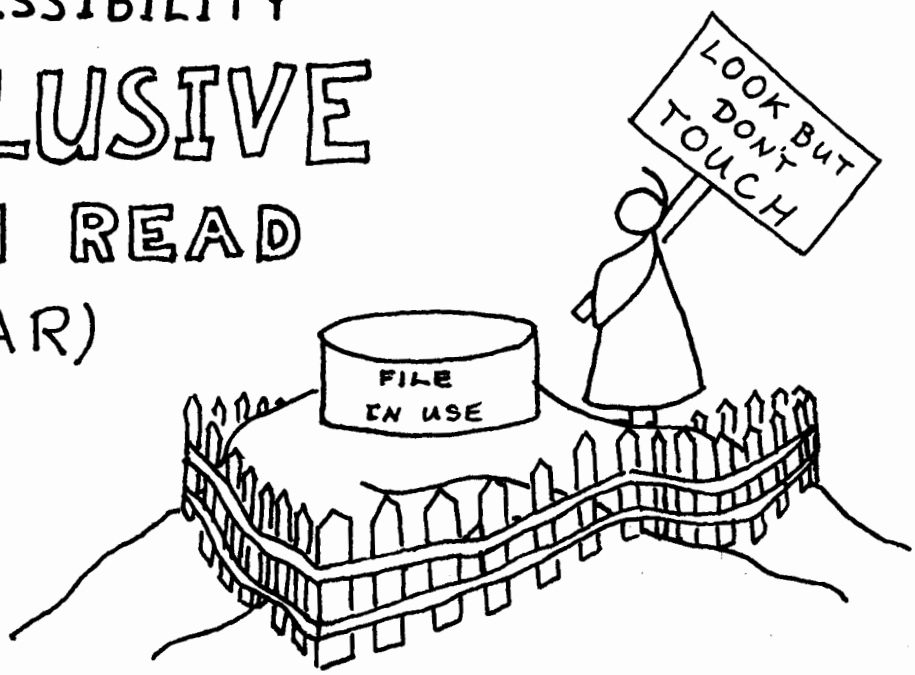
(EXC)



- NO OTHER USER CAN ACCESS A FILE WHILE IT IS OPEN WITH EXCLUSIVE ACCESS.
- NORMALLY USED WITH OUTPUT OPERATIONS

FILE ACCESSIBILITY

EXCLUSIVE WITH READ (EAR)

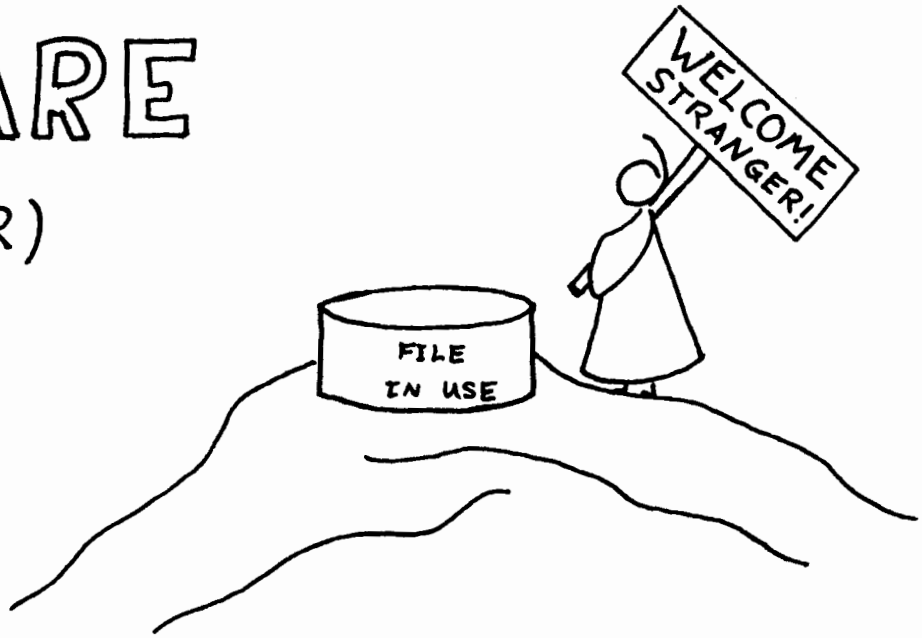


- FILE MAY BE READ BUT NOT WRITTEN BY OTHER ACCESSORS
- USE WITH INPUT AND/OR OUTPUT OPERATIONS

FILE ACCESSIBILITY

SHARE

(SHR)



- FILE DATA SHARED
- ACCESS TO FILE IS SHARED
- AVAILABLE TO ANY PROGRAM IN THE SYSTEM

FILE ACCESSIBILITY

(SHARED FILE)

DYNAMIC LOCKING

SPECIAL FEATURES

MULTI ACCESS

- FILE DATA SHARED
- ACCESS TO FILE IS SHARED
- RESTRICTED TO RELATED PROCESSES (FATHER/SONS)

e.g. TRANSACTION LOGGING OF RECORDS

SPECIAL FEATURES

NO WAIT IO

ALLOWS USER PROGRAM

TO CONTINUE PROCESSING

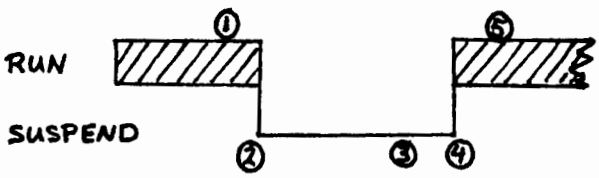
DURING I/O TRANSFERS

- AVAILABLE WITH NOBUF (ONLY)
- PRIVILEGE MODE REQUIRED TO ESTABLISH ACCESS
- USEFUL WITH 'MULTIPLE TERMINAL' APPLICATIONS

NOBUF I/O SEQUENCE

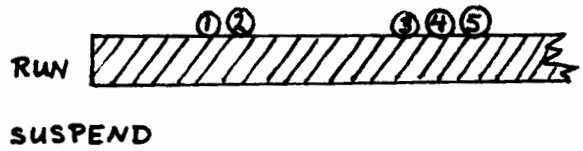
NORMAL "WAIT"

1. PROCESS REQUESTS I/O
2. PROCESS SUSPENDS UNTIL TRANSFER IS DONE
3. I/O COMPLETES
4. PROCESS REACTIVATED
5. EXECUTION CONTINUES



"NO WAIT"

1. PROCESS REQUESTS I/O
2. PROCESS CONTINUES EXECUTION DURING TRANSFER
3. I/O COMPLETES
4. PROCESS REQUESTS COMPLETION INFORMATION
5. EXECUTION CONTINUES



DISC FILE CHARACTERISTICS

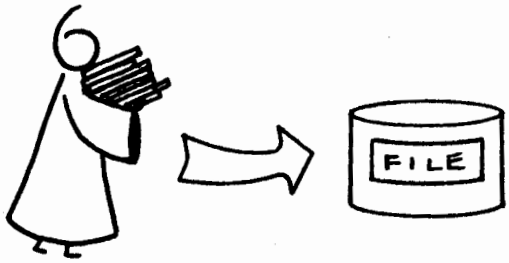
RECORD STRUCTURE

FILE STRUCTURE

FILE IDENTIFICATION

FILE DOMAIN

FILE USAGE



- WHERE WILL THIS INFORMATION COME FROM?
- HOW CAN IT BE SUPPLIED TO THE FILE SYSTEM?

DEFINING FILE CHARACTERISTICS

- INTRINSIC

FOPEN

(RECORD STRUCTURE
FILE STRUCTURE
FILE IDENTIFICATION
FILE DOMAIN
FILE USAGE)

- PARAMETER INFORMATION

DEFINING FILE CHARACTERISTICS

```
   I                    BA      LV      LV     IV    BA      BA
filenum:=FOPEN(formaldesignator,foptions,aoptions,recsize,device,formmsg,

                IV          IV          IV    DV          IV
          userlabels,blockfactor,numbuffers,filesize,numextents,

                IV      IV   0-V
          initialloc,filecode);
```

RECORD STRUCTURE	FILE STRUCTURE	FILE IDENTIFICATION	FILE DOMAIN	FILE USAGE
FOPTIONS	BLOCKFACTOR	USERLABELS	FOPTIONS	AOPTIONS
RECSIZE	FILESIZE	FILECODE		NUMBUFFERS
	NUMEXTENTS	FORMAL-DESIGNATOR		FOPTIONS
	INITALLOC	FOPTIONS		
	DEVICE			
	FOPTIONS			

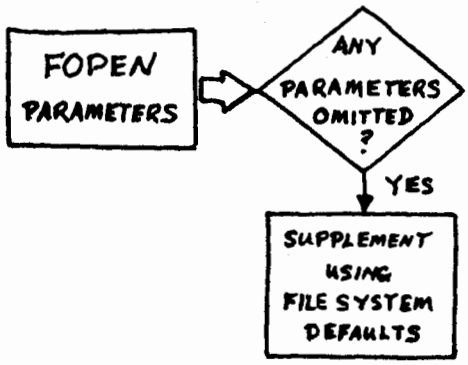
OPENING THE DISC FILE

IS THIS TRIP REALLY NECESSARY?

- ARE ALL FOPEN PARAMETERS REQUIRED?
- CAN I OVERRIDE THE FOPEN PARAMETERS SPECIFIED?

OPENING THE DISC FILE

PARAMETER DEFAULTS



RECORD

BINARY
FIXED
RECSIZE = 128

FILE

BLOCKFACTOR = n*
FILESIZE = 1023
DEV = DISC
NUMEXTS = 8
INITALLOC = 1

IDENTIFICATION

UNNAMED
USERLABELS = 0
FILECODE = 0

DOMAIN

NEW

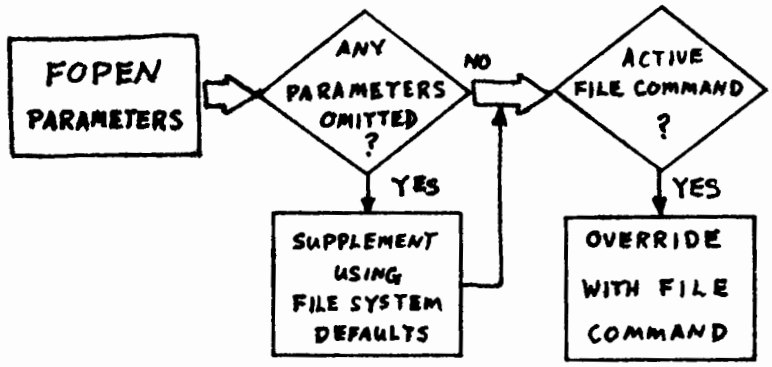
USAGE

READ ONLY
ACCESS = { SHR (READ ONLY) / EXC (ALL OTHER) }
NO FLOCK
BUFFERED
NUMBUF = 2
NO MULTIREC
NO MULTIACC
WAIT FOR I/O

* $n = \lfloor 128/\text{RECSIZE} \rfloor$ (ROUNDED DOWN)

OPENING THE DISC FILE

:FILE OVERRIDES



A :FILE COMMAND CAN SPECIFY ALL DISC FILE CHARACTERISTICS EXCEPT THE FOLLOWING:

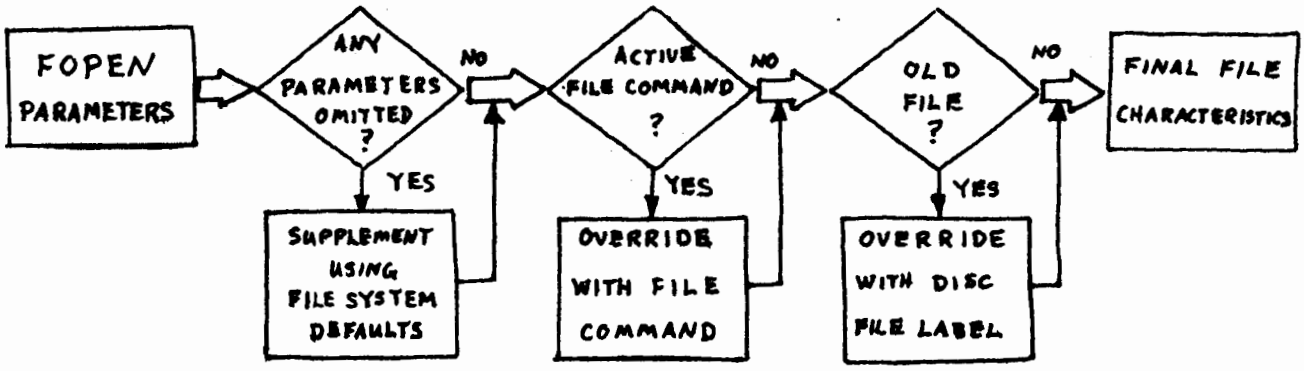
ALL FILES
- USERLABELS
- EXECUTE ACCESS
- DYNAMIC LOCKING

OLD DISC FILES
- RECORD STRUCTURE (REC=)
  [ASCII/BINARY, F/V/U, RECSIZE, BLOCKFACTOR]
- FILE STRUCTURE (DISC=, DEV=)
  [FILE SIZE, NUMEXTENTS, INITALLOC, DEVICE]
- FILE CODE (CODE=)

- FILE EQUATIONS TAKE EFFECT AT FOPEN TIME - NOT AT THE TIME THEY ARE ENTERED.
- IF FOPEN DISALLOWS FILE EQUATIONS <u>AND</u> THE FORMALDESIGNATOR DOES NOT BEGIN WITH *, NO :FILE OVERRIDE TAKES PLACE.

OPENING THE DISC FILE

FILE LABEL OVERRIDES

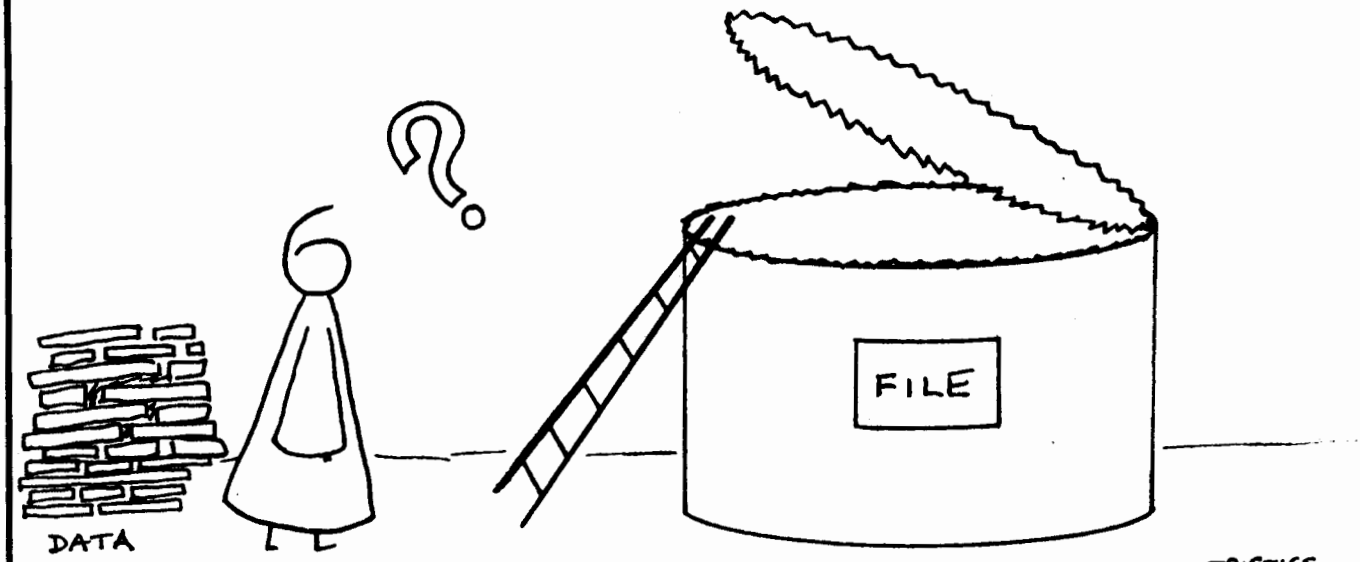


FILE LABEL INFORMATION

RECORD	FILE	IDENTIFICATION
ASCII/BINARY	BLOCKFACTOR	USERLABELS
F, V, U	FILESIZE	FILECODE
RECORD SIZE	NUMEXTS	
	INITALLOC	

NOTE: DOMAIN AND USAGE ARE DETERMINED BEFORE THE FILE LABEL IS CHECKED.

THE OPEN DISC FILE



FOPEN: DETERMINES PHYSICAL CHARACTERISTICS
BUILDS ACCESS PATH
DOES NOT TRANSFER DATA

- ONCE THE FILE IS OPEN, HOW DO WE MOVE THE DATA?

FILE ACCESS INTRINSICS

DATA TRANSFER

INPUT:	OUTPUT:
FREAD	FWRITE
FREADDIR	FWRITEDIR
	FUPDATE

OTHER INTRINSIC FUNCTIONS:

ACCESS USER LABELS
LOCK/UNLOCK FILES
PERFORM CONTROL OPERATIONS

NOTE: THESE INTRINSICS CAN ONLY BE USED ON OPEN FILES.

CLOSING THE DISC FILE

WHEN ACCESS IS NO LONGER REQUIRED,

THE FILE MUST BE "CLOSED"

- WHAT HAPPENS TO THE FILE CHARACTERISTICS?
- WILL THE STORAGE SPACE BE SAVED? WHERE?
- HOW MUST THE FILE BE FOPEN'D NEXT TIME?

CLOSING THE DISC FILE

- INTRINSIC

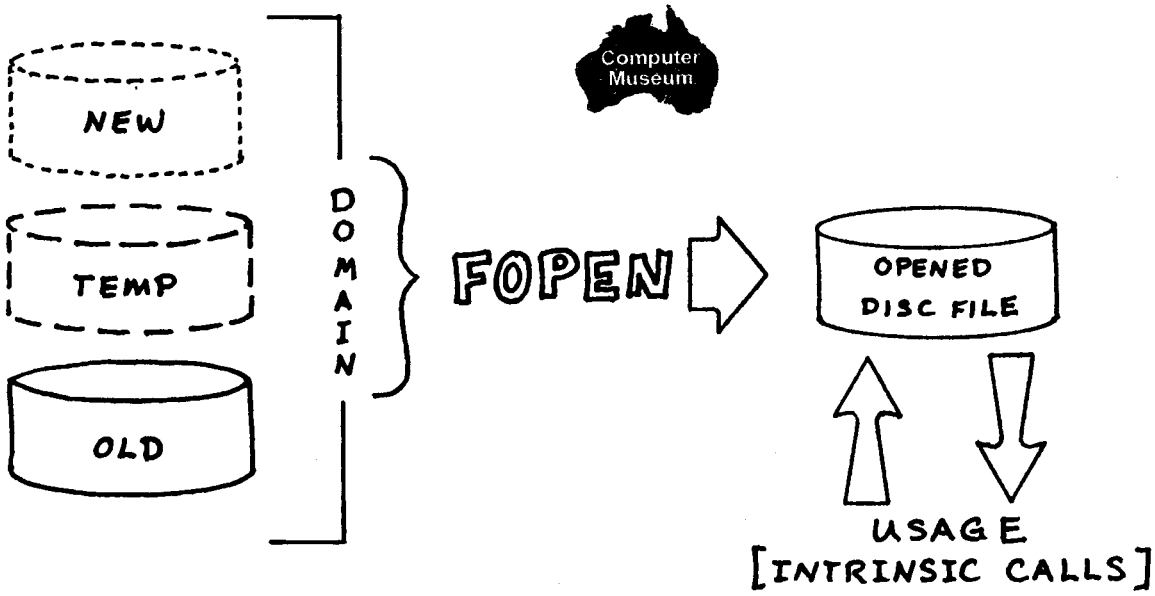
FCLOSE

(FILE DOMAIN DISPOSITION
FILE SPACE DISPOSITION
FILE SECURITY DISPOSITION)

```
                IV          IV          IV
FCLOSE(filenum,disposition,seccode);
```

- PARAMETER INFORMATION

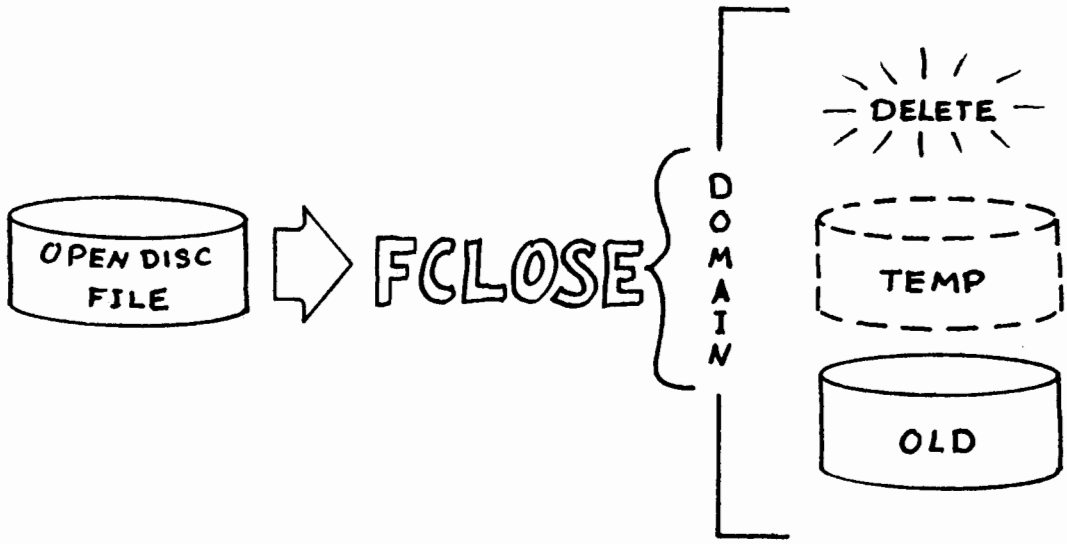
FILE DOMAIN DISPOSITION



FOPEN USES DOMAIN TO LOCATE THE FILE. AND SETS UP ACCESS PATH TO THE FILE.

FILE USAGE IS ACCOMPLISHED BY CALLING INTRINSICS.

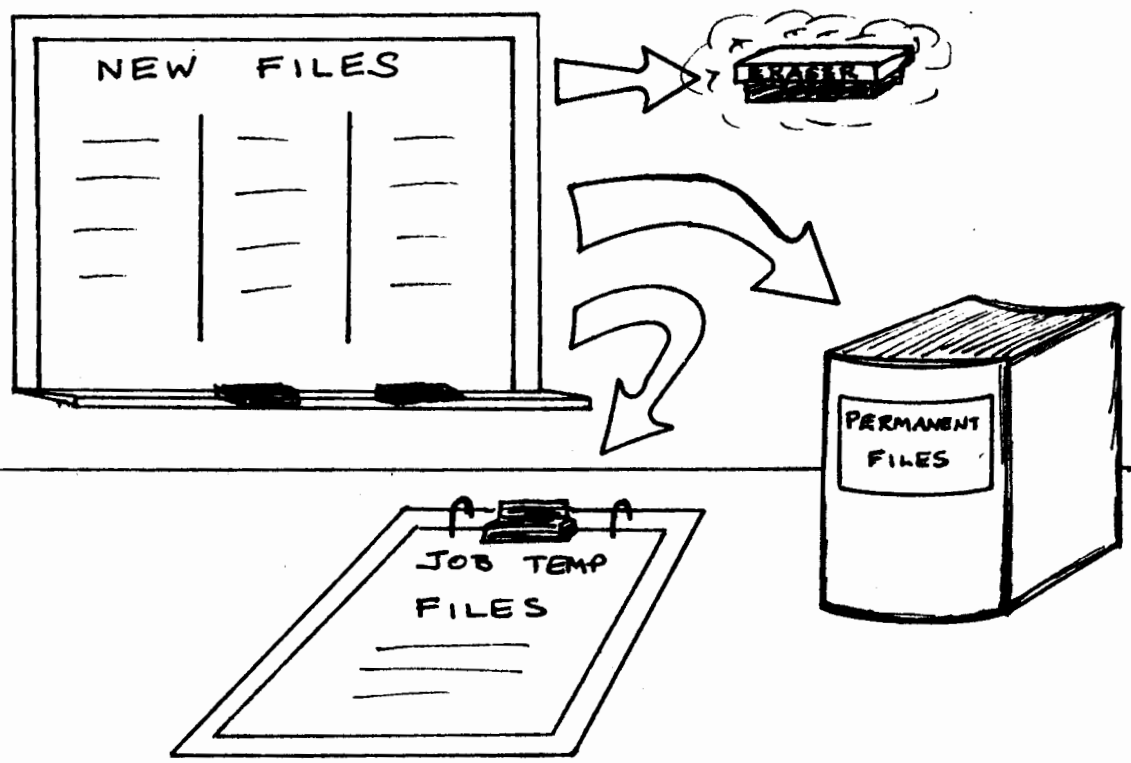
FILE DOMAIN DISPOSITION



FCLOSE CLOSES ACCESS PATH SET UP BY FOPEN AND DETERMINES DOMAIN FILE WILL BE PLACED (OR LEFT) IN

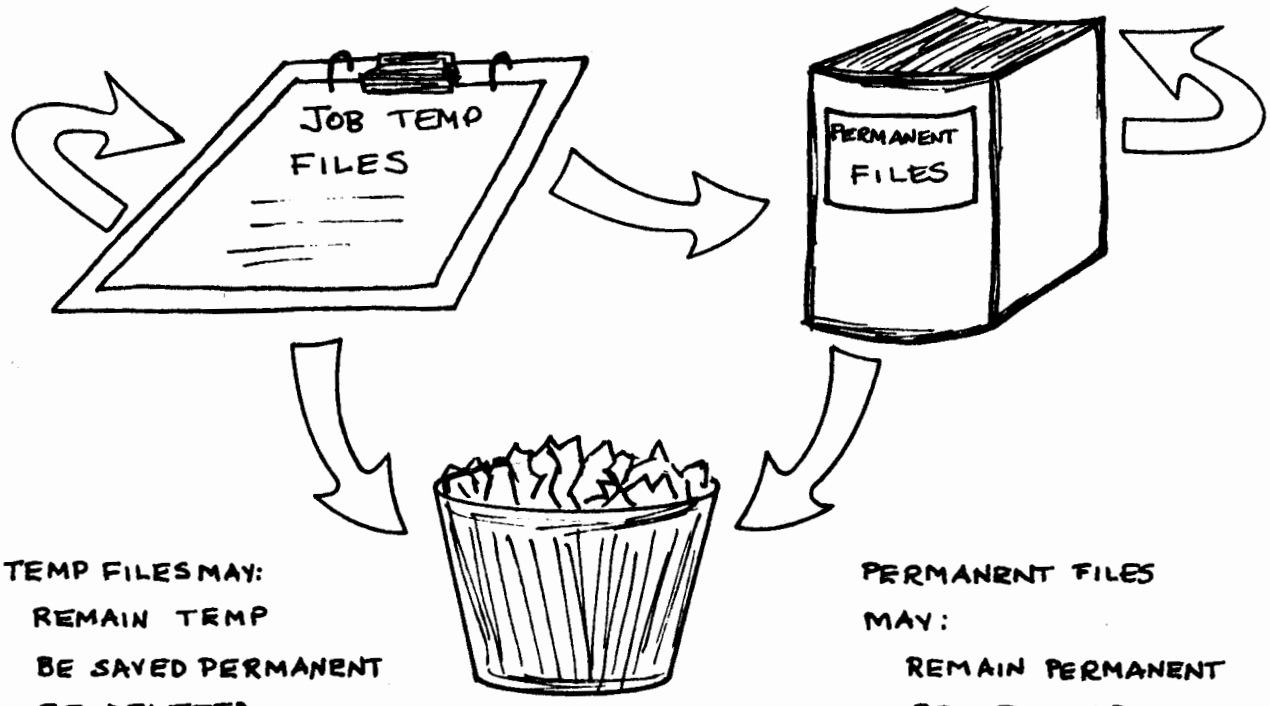
(DOMAIN DISPOSITION CAN BE OVERRIDDEN WITH :FILE)

FILE DOMAIN DISPOSITION



- NEW FILES MAY BE DELETED, SAVED PERMANENT OR SAVED JOB TEMPORARY

FILE DOMAIN DISPOSITION

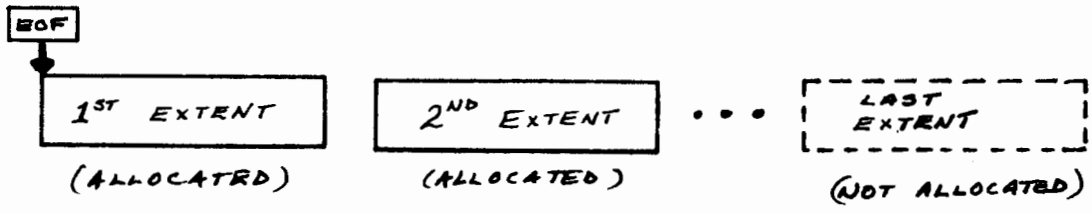


TEMP FILES MAY:
- REMAIN TEMP
- BE SAVED PERMANENT
- BE DELETED

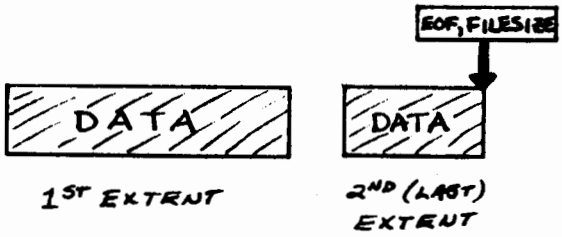
PERMANENT FILES MAY:
- REMAIN PERMANENT
- BE DELETED

FILE SPACE DISPOSITION

AT FOPEN (NEW)

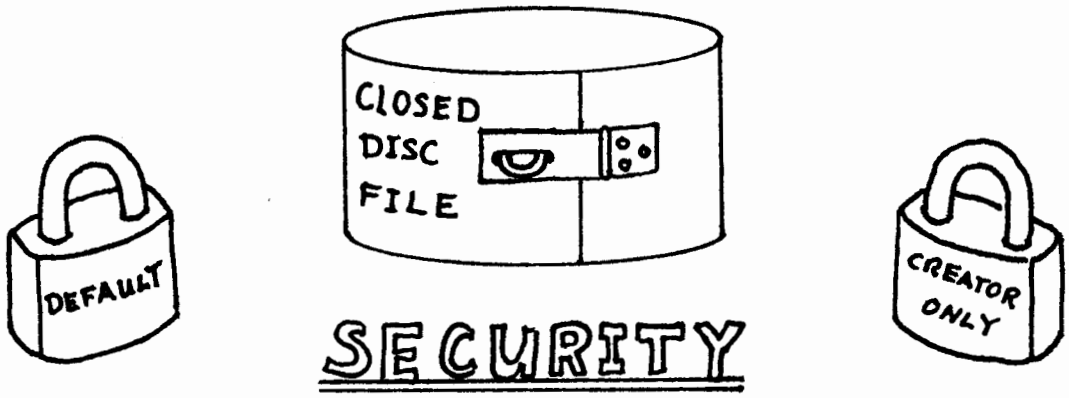


AT FCLOSE (OLD OR TEMP)



- SPACE BEYOND EOF MAY BE RETURNED TO SYSTEM AS FREE SPACE
- NO FURTHER FILE EXPANSION IS POSSIBLE

FILE SECURITY DISPOSITION

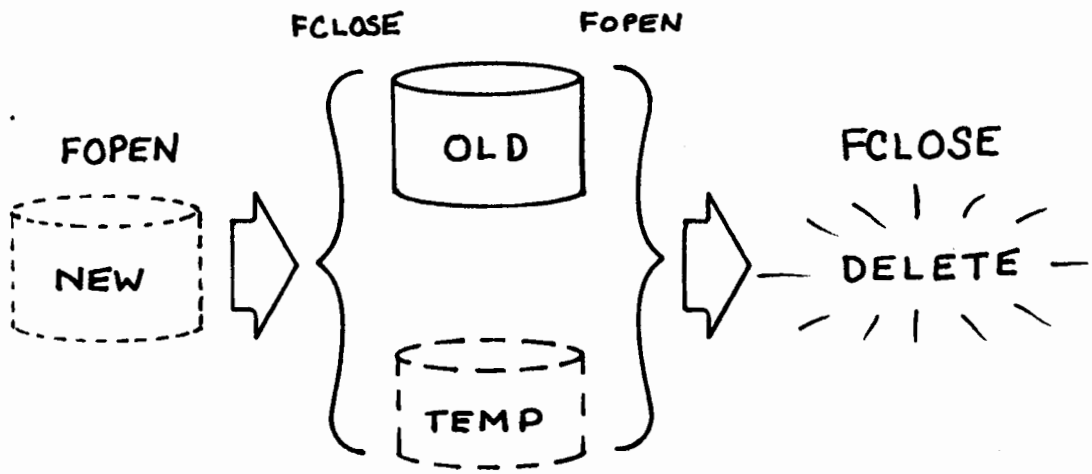


- SECURITY DISPOSITION APPLIES ONLY WHEN NEW OR TEMP FILES ARE SAVED PERMANENT

- CREATOR ONLY - ACCESS LIMITED TO CREATOR AND ACCOUNT/SYSTEM MANAGERS

FOPEN/FCLOSE

EXAMPLES



FOPEN/FCLOSE

EXAMPLE 1

WHAT IT DOES

- REQUESTS AND INPUTS DATA FROM USER
- WRITES DATA TO A NEW FILE
- SAVES FILE AS PERMANENT OR JOB TEMPORARY

WHAT IT SHOWS

- HOW TO FOPEN A NEW FILE
- HOW TO FCLOSE A FILE PERMANENT OR JOB TEMPORARY

FOPEN/FCLOSE - EXAMPLE 1

```
PAGE 0001   HEWLETT-PACKARD 32100A.06.2  SPL  MON, DEC 27, 1976, 10:34 AM

00001000  00000 0   $CONTROL USLINIT,MAIN=FOPENX1
00002000  00000 0   BEGIN
00003000  00000 1
00004000  00000 1     BYTE ARRAY FILENAME(0:5):="FILE1*";
00005000  00004 1     LOGICAL    FOPTIONS:=4;              <<NEW,ASCII,FIXED>>
00006000  00004 1     LOGICAL    AOPTIONS:=1;              <<WRITE ONLY>>
00007000  00004 1     INTEGER    RECSIZE:=-80;             <<80 BYTES PER RECORD>>
00008000  00004 1     <<DEFAULT  DEVICE="DISC ">>
00009000  00004 1     <<DEFAULT  NO FORMS MESSAGE>>
00010000  00004 1     <<DEFAULT  NO USERLABELS>>
00011000  00004 1     INTEGER    BLOCKFACTOR:=16;          <<16 LOG. RECS PER PHYS. REC>>
00012000  00004 1     <<DEFAULT  NUMBUFFERS:=2>>
00013000  00004 1     DOUBLE     FILESIZE:=16D;            <<16 RECORDS IN THE FILE>>
00014000  00004 1     INTEGER    NUMEXTENTS:=1;
00015000  00004 1     <<DEFAULT  INITALLOC:=1>>
00016000  00004 1     <<DEFAULT  NO FILE CODE (0)>>
00017000  00004 1
00018000  00004 1     INTEGER    FILENUM;                  <<RETURNED BY FOPEN>>
00019000  00004 1
00020000  00004 1
00021000  00004 1     ARRAY DATA(0:79),PROMPT(0:5):="DATA> ";
00022000  00003 1     ARRAY DISPOSITION(0:12):="OLD OR TEMP? ";
00023000  00007 1     BYTE ARRAY DISPSTN(*)=DISPOSITION;
00024000  00007 1     INTEGER LEN,DISP;
00025000  00007 1
00026000  00007 1     INTRINSIC FOPEN,FCLOSE,FWRITE,PRINT,READ,PRINT'FILE'INFO,QUIT;
00027000  00007 1
00028000  00007 1
00029000  00007 1     SUBROUTINE CCL(ERRNO);
00030000  00000 1     VALUE ERRNO; INTEGER ERRNO;
00031000  00000 1     BEGIN
00032000  00000 2       PRINT'FILE'INFO(FILENUM);
00033000  00002 2       QUIT(ERRNO);
00034000  00004 2     END;
00035000  00005 1
00036000  00005 1   << ****   START OF MAIN CODE   **** >>
00037000  00005 1
00038000  00005 1   FILENUM:=FOPEN(FILENAME,FOPTIONS,AOPTIONS,RECSIZE,,,,  << OPEN FILE >>
00039000  00012 1                 BLOCKFACTOR,,FILESIZE,NUMEXTENTS);
00040000  00023 1   IF < THEN CCL(1);                                     <<ERROR CHECK>>
00041000  00030 1
00042000  00030 1   REQUEST'DATA: PRINT(PROMPT,-6,%320);                  <<ASK FOR DATA>>
00043000  00034 1                 IF(LEN:=READ(DATA,-80))=0               <<FOR FILE>>
00044000  00042 1                    THEN GO CLOSE'FILE;
00045000  00044 1                 FWRITE(FILENUM,DATA,-LEN,0);            <<WRITE TO FILE>>
00046000  00051 1                 IF < THEN CCL(3);
00047000  00055 1                 IF > THEN GO CLOSE'FILE;                <<EOF DETECTED>>
00048000  00056 1                 GO REQUEST'DATA;                        <<LOOP TO REQ>>
00049000  00057 1   CLOSE'FILE:   PRINT(DISPOSITION,-13,%320);            <<INPUT CLOSING>>
00050000  00063 1                 READ(DISPOSITION,-4);                   <<DISPOSITION>>
00051000  00070 1                 DISP:=IF DISPSTN="OLD" THEN 1 ELSE 2;
00052000  00105 1                 FCLOSE(FILENUM,DISP,0);                 <<CLOSE & SAVE>>
00053000  00111 1                 IF < THEN CCL(5);                       <<TEMP OR PERM>>
00054000  00115 1   END.
 PRIMARY DB STORAGE=%017;   SECONDARY DB STORAGE=%00146
 NO. ERRORS=0000;           NO. WARNINGS=0000
 PROCESSOR TIME=0:00:02;    ELAPSED TIME=0:00:15
```

FOPEN/FCLOSE - EXAMPLE 1

SAMPLE OUTPUT

(SAVE PERM)

```
:LISTF FILE1,2
ERR 108
NON-EXISTENT FILE

:RUN FOPENX1

DATA> RECORD 1
DATA> RECORD 2
DATA> RECORD 3
DATA> THIS IS THE LAST RECORD.....
DATA>
OLD OR TEMP? OLD

 END OF PROGRAM
:LISTF FILE1,2
ACCOUNT=  MAL        GROUP=  CLASS

FILENAME  CODE ------------LOGICAL RECORD-----------  ----SPACE---- ACC
                 SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE1             80B  FA           4         16  16       10  1  1 ???
```

FOPEN/FCLOSE - EXAMPLE 1

```
PAGE 0001   HEWLETT-PACKARD 32100A.06.2  SPL  MON, DEC 27, 1976, 10:34 AM

00001000  00000 0   $CONTROL USLINIT,MAIN=FOPENX1
00002000  00000 0   BEGIN
00003000  00000 1
00004000  00000 1     BYTE ARRAY FILENAME(0:5):="FILE1*";
00005000  00004 1     LOGICAL    FOPTIONS:=4;              <<NEW,ASCII,FIXED>>
00006000  00004 1     LOGICAL    AOPTIONS:=1;              <<WRITE ONLY>>
00007000  00004 1     INTEGER    RECSIZE:=-80;             <<80 BYTES PER RECORD>>
00008000  00004 1     <<DEFAULT  DEVICE="DISC ">>
00009000  00004 1     <<DEFAULT  NO FORMS MESSAGE>>
00010000  00004 1     <<DEFAULT  NO USERLABELS>>
00011000  00004 1     INTEGER    BLOCKFACTOR:=16;          <<16 LOG. RECS PER PHYS. REC>>
00012000  00004 1     <<DEFAULT  NUMBUFFERS:=2>>
00013000  00004 1     DOUBLE     FILESIZE:=16D;            <<16 RECORDS IN THE FILE>>
00014000  00004 1     INTEGER    NUMEXTENTS:=1;
00015000  00004 1     <<DEFAULT  INITALLOC:=1>>
00016000  00004 1     <<DEFAULT  NO FILE CODE (0)>>
00017000  00004 1
00018000  00004 1     INTEGER    FILENUM;                  <<RETURNED BY FOPEN>>
00019000  00004 1
00020000  00004 1
00021000  00004 1     ARRAY DATA(0:79),PROMPT(0:5):="DATA> ";
00022000  00003 1     ARRAY DISPOSITION(0:12):="OLD OR TEMP? ";
00023000  00007 1     BYTE ARRAY DISPSTN(*)=DISPOSITION;
00024000  00007 1     INTEGER LEN,DISP;
00025000  00007 1
00026000  00007 1     INTRINSIC FOPEN,FCLOSE,FWRITE,PRINT,READ,PRINT'FILE'INFO,QUIT;
00027000  00007 1
00028000  00007 1
00029000  00007 1     SUBROUTINE CCL(ERRNO);
00030000  00000 1     VALUE ERRNO; INTEGER ERRNO;
00031000  00000 1     BEGIN
00032000  00000 2       PRINT'FILE'INFO(FILENUM);
00033000  00002 2       QUIT(ERRNO);
00034000  00004 2     END;
00035000  00005 1
00036000  00005 1   << ****   START OF MAIN CODE   **** >>
00037000  00005 1
00038000  00005 1   FILENUM:=FOPEN(FILENAME,FOPTIONS,AOPTIONS,RECSIZE,,,,  << OPEN FILE >>
00039000  00012 1                 BLOCKFACTOR,,FILESIZE,NUMEXTENTS);
00040000  00023 1   IF < THEN CCL(1);                                     <<ERROR CHECK>>
00041000  00030 1
00042000  00030 1   REQUEST'DATA: PRINT(PROMPT,-6,%320);                  <<ASK FOR DATA>>
00043000  00034 1                 IF(LEN:=READ(DATA,-80))=0               <<FOR FILE>>
00044000  00042 1                    THEN GO CLOSE'FILE;
00045000  00044 1                 FWRITE(FILENUM,DATA,-LEN,0);            <<WRITE TO FILE>>
00046000  00051 1                 IF < THEN CCL(3);
00047000  00055 1                 IF > THEN GO CLOSE'FILE;                <<EOF DETECTED>>
00048000  00056 1                 GO REQUEST'DATA;                        <<LOOP TO REQ>>
00049000  00057 1   CLOSE'FILE:   PRINT(DISPOSITION,-13,%320);            <<INPUT CLOSING>>
00050000  00063 1                 READ(DISPOSITION,-4);                   <<DISPOSITION>>
00051000  00070 1                 DISP:=IF DISPSTN="OLD" THEN 1 ELSE 2;
00052000  00105 1                 FCLOSE(FILENUM,DISP,0);                 <<CLOSE & SAVE>>
00053000  00111 1                 IF < THEN CCL(5);                       <<TEMP OR PERM>>
00054000  00115 1   END.
 PRIMARY DB STORAGE=%017;   SECONDARY DB STORAGE=%00146
 NO. ERRORS=0000;           NO. WARNINGS=0000
 PROCESSOR TIME=0:00:02;    ELAPSED TIME=0:00:15
```

FOPEN/FCLOSE - EXAMPLE 1

SAMPLE OUTPUT

(SAVE PERM)

```
:LISTF FILE1,2
ERR 108
NON-EXISTENT FILE

:RUN FOPENX1

DATA> RECORD 1
DATA> RECORD 2
DATA> RECORD 3
DATA> THIS IS THE LAST RECORD.....
DATA>
OLD OR TEMP? OLD

 END OF PROGRAM
:LISTF FILE1,2
ACCOUNT=  MAL        GROUP=  CLASS

FILENAME  CODE ------------LOGICAL RECORD-----------  ----SPACE---- ACC
                SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE1            80B  FA           4         16  16       10  1  1 ???
```

FOPEN/FCLOSE - EXAMPLE 1

```
:RUN LISTEQ2.PUB.SYS

LISTEQ2  (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1976.

***NO TEMP FILES

***NO FILE EQUATIONS

 END OF PROGRAM
```

SAMPLE OUTPUT

(SAVE TEMP)

```
:RUN FOPENX1

DATA> FIRST RECORD
DATA> SECOND RECORD
DATA> THIRD RECORD
DATA> LAST BUT NOT LEAST......
DATA>
OLD OR TEMP? TEMP

 END OF PROGRAM
:RUN LISTEQ2.PUB.SYS

LISTEQ2  (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1976.

***TEMP FILES

FILE1.CLASS.MAL

***NO FILE EQUATIONS

 END OF PROGRAM
```

EXAMPLE 2

WHAT IT DOES

- READS DATA FROM AN OLD FILE (PERM. OR TEMP.)
- PRINTS DATA AT JOB/SESSION OUTPUT DEVICE
- CLOSES AND DELETES OLD FILE

WHAT IT SHOWS

- HOW TO FOPEN AN OLD FILE (SEARCH BOTH JOB TEMPORARY AND SYSTEM DIRECTORIES)
- HOW TO FCLOSE AN OLD FILE WITH DELETE DISPOSITION

FOPEN/FCLOSE - EXAMPLE 2

```
PAGE 0001   HEWLETT-PACKARD 32100A.06.2  SPL  MON, DEC 27, 1976, 10:34 AM

00001000  00000 0   $CONTROL USLINIT,MAIN=FOPENX2
00002000  00000 0   BEGIN
00003000  00000 1     BYTE ARRAY FILENAME(0:5):="FILE1*";
00004000  00004 1     LOGICAL    FOPTIONS:=3;                    <<OLD TEMP OR PERM>>
00005000  00004 1     LOGICAL    AOPTIONS:=0;                    <<READ ONLY>>
00006000  00004 1
00007000  00004 1     INTEGER    FILENUM;                        <<RETURNED BY FOPEN>>
00008000  00004 1
00009000  00004 1     ARRAY DATA(0:79);
00010000  00004 1     INTEGER DELETE:=4;                         <<USED BY FCLOSE>>
00011000  00004 1
00012000  00004 1     INTRINSIC FOPEN,FCLOSE,FREAD,PRINT,PRINT'FILE'INFO,QUIT;
00013000  00004 1
00014000  00004 1     SUBROUTINE CCL(ERRNO);
00015000  00000 1     VALUE ERRNO; INTEGER ERRNO;
00016000  00000 1     BEGIN
00017000  00000 2       PRINT'FILE'INFO(FILENUM);
00018000  00002 2       QUIT(ERRNO);
00019000  00004 2     END;
00020000  00005 1
00021000  00005 1   <<   ****    START OF MAIN CODE    ****   >>
00022000  00005 1
00023000  00005 1   FILENUM:=FOPEN(FILENAME,FOPTIONS,AOPTIONS);
00024000  00015 1   IF < THEN CCL(1);
00025000  00022 1
00026000  00022 1   READ'REC: FREAD(FILENUM,DATA,-80);            <<READ FILE RECORD>>
00027000  00030 1             IF < THEN CCL(3);                   <<CHECK FOR ERROR>>
00028000  00034 1             IF > THEN GO END'PGM;               <<EOF DETECTED>>
00029000  00035 1             PRINT(DATA,-80,0);                  <<OUTPUT TO $STDLIST>>
00030000  00041 1             GO READ'REC;                        <<LOOP TO NEXT RECORD>>
00031000  00042 1
00032000  00042 1   END'PGM:  FCLOSE(FILENUM,DELETE,0);           <<DELETE FILE>>
00033000  00046 1
00034000  00046 1   END.
 PRIMARY DB STORAGE=%006;   SECONDARY DB STORAGE=%00123
 NO. ERRORS=0000;           NO. WARNINGS=0000
 PROCESSOR TIME=0:00:01;    ELAPSED TIME=0:00:10
```

FOPEN/FCLOSE - EXAMPLE 2

SAMPLE OUTPUT (PERM FILE FOUND)

```
:LISTF FILE1,2
ACCOUNT=  MAL        GROUP=  CLASS

FILENAME  CODE ------------LOGICAL RECORD-----------  ----SPACE---- ACC
                 SIZE  TYP        EOF      LIMIT R/B  SECTORS #X MX

FILE1             80B  FA          4          16  16       10  1  1 ???
```

```
:RUN FOPENX2

RECORD 1

RECORD 2

RECORD 3

THIS IS THE LAST RECORD.....


 END OF PROGRAM
```

```
:LISTF FILE1,2
ERR 108
NON-EXISTENT FILE
```

FOPEN/FCLOSE - EXAMPLE 2

```
:RUN LISTEQ2.PUB.SYS

LISTEQ2  (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1976.

***TEMP FILES

FILE1.CLASS.MAL

***NO FILE EQUATIONS
 END OF PROGRAM
:RUN FOPENX2

FIRST RECORD

SECOND RECORD

THIRD RECORD

LAST BUT NOT LEAST......


 END OF PROGRAM
:RUN LISTEQ2.PUB.SYS
LISTEQ2  (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1976.

***NO TEMP FILES

***NO FILE EQUATIONS
 END OF PROGRAM
```

SAMPLE OUTPUT

(TEMP FILE FOUND)

FOPEN NEW

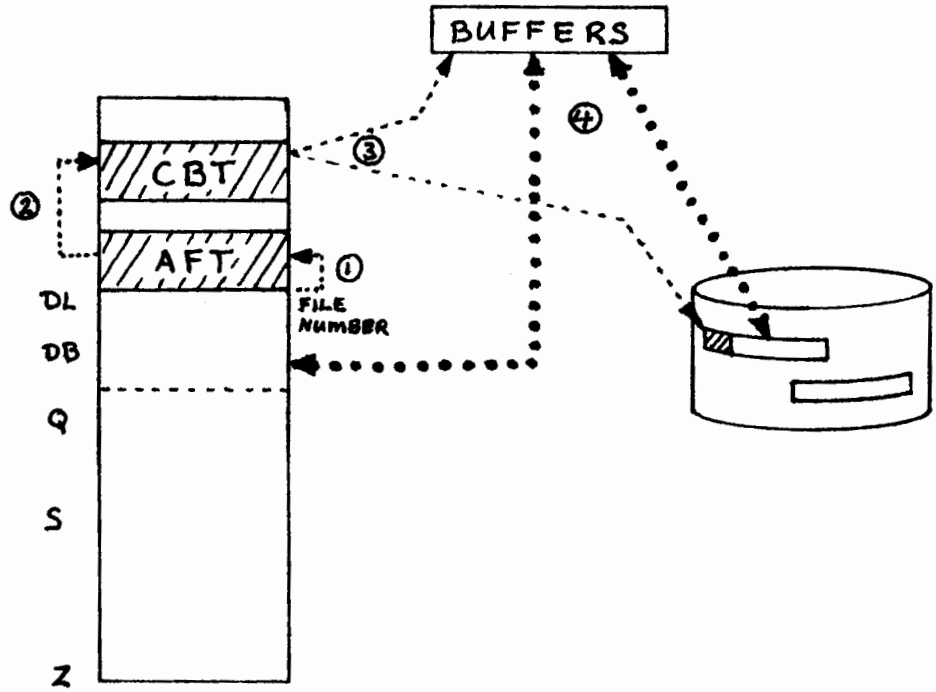
IMPLICATIONS:

- FILE IS DEFINED ONLY FOR PROCESS ACCESSING IT.
- THERE IS NO DIRECTORY ENTRY FOR THE FILE
- OPENING PROCESS HAS ONLY COPY OF FILE'S ADDRESS
- THE FILE IS NOT SHARABLE
- DEFAULT FILE CLOSE (NO SPECIAL ACTION TAKEN)

  FILE ADDRESS IS LOST

  FILE BECOMES DISC FREE SPACE

FILE ACCESS MECHANISM



1) FILE SYSTEM INTRINSIC USES FILE NUMBER TO INDEX INTO AVAILABLE FILE TABLE (AFT).

2) AFT ENTRY POINTS TO CONTROL BLOCKS FOR THE FILE.

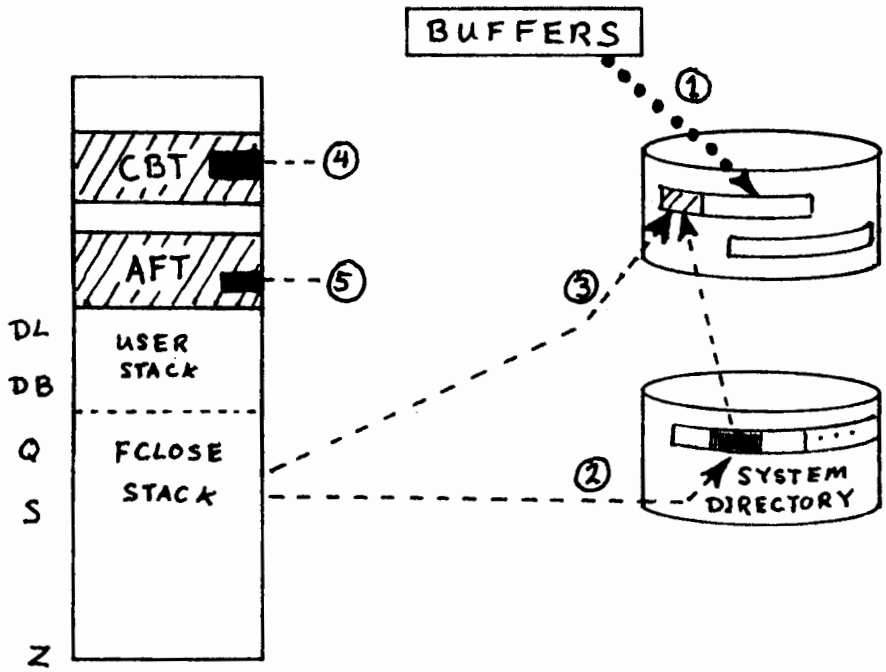
3) CONTROL BLOCK ENTRIES CONTAIN FILE INFORMATION AND POINTERS TO FILE LABEL AND BUFFERS.

4) FILE SYSTEM MANAGES DATA TRANSFERS BETWEEN:
   FILE - BUFFERS
   BUFFERS - STACK

NOTE: IF FILE IS <u>NOBUF</u> TRANSFERS GO DIRECTLY BETWEEN DISC AND STACK.

FCLOSE

SAVE



1) POST BUFFERS TO THE DISC FILE.

2) BUILD AN ENTRY IN THE SYSTEM DIRECTORY WHICH POINTS TO THE FILE LABEL.

3) UPDATE EOF IN THE FILE LABEL

4) REMOVE CONTROL BLOCK ENTRIES FROM CBT IN STACK OR FILE SEGMENT (IF NOCB).

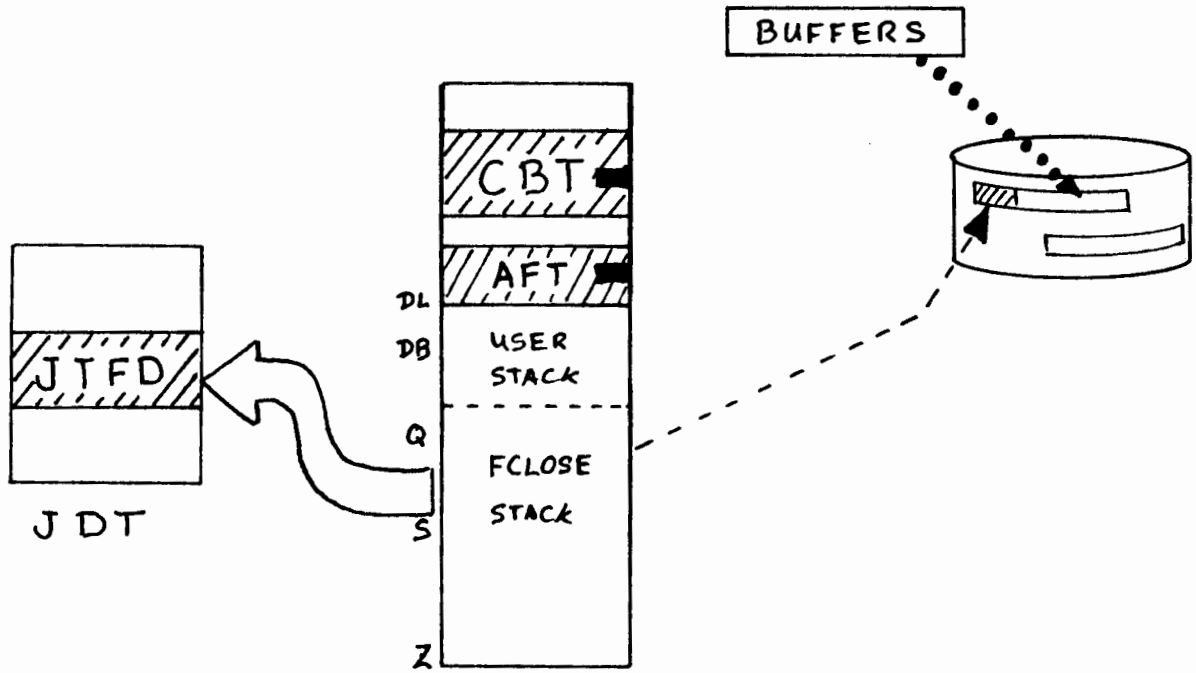
5) RESET AFT ENTRY FOR THE FILE.

FCLOSE SAVE

IMPLICATIONS:

- FILE IS NOW DEFINED IN THE SYSTEM
- IT HAS A SYSTEM DIRECTORY ENTRY
- ANY PROCESS WHICH PASSES FILE SECURITY RESTRICTIONS CAN OPEN AND ACCESS IT.
- FILE IS OLD OR PERMANENT
- DEFAULT FCLOSE (NO SPECIAL ACTION TAKEN) FILE WILL REMAIN ON THE SYSTEM

FCLOSE TEMP



ALL OPERATIONS ARE THE SAME AS FCLOSE (SAVE)

EXCEPT:

THE DIRECTORY ENTRY IS PLACED IN A JOB LOCAL EXTRA DATA SEGMENT - THE JOB DIRECTORY TABLE (JDT)

ENTRIES FOR TEMPORARY FILES RESIDE IN THE JOB TEMPORARY FILE DIRECTORY (JTFD) PORTION OF THE JDT.

FCLOSE TEMP

IMPLICATIONS:

- FILE IS NOW DEFINED ONLY WITHIN THE JOB/SESSION. PROCESSES OUTSIDE THE JOB/SESSION HAVE NO ACCESS TO THE JTFD WHERE THE FILE IS LISTED.

- FILE IS TEMP. WITHOUT SPECIAL ACTION FILE IS DELETED AT END OF JOB/SESSION. [JDT SEGMENT IS DELETED.]

- A FILE IN JTFD MAY HAVE THE SAME NAME AS A FILE IN SYSTEM DIRECTORY

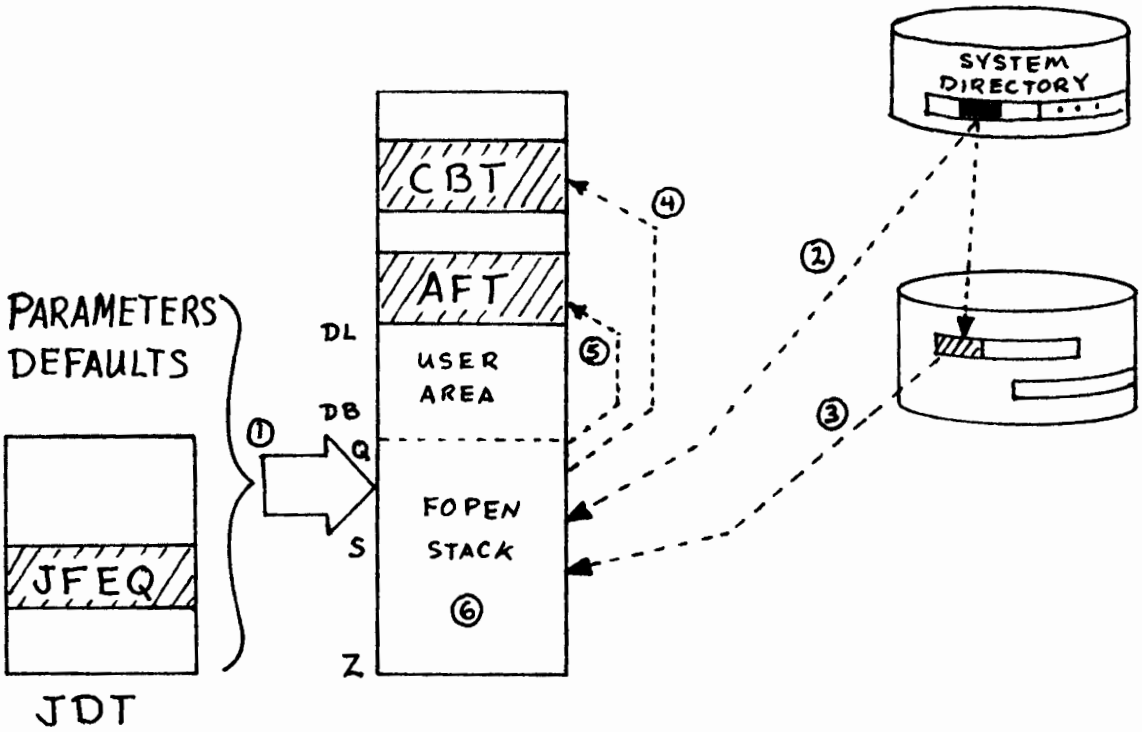
  - IF THAT FILENAME IS FOPENED (OLD/TEMP) TEMP FILE IS FOUND FIRST AND OPENED

  - IT IS POSSIBLE TO HAVE 3 FILES WITH THE SAME NAME OPEN AT ONE TIME:
    1 IN SYSTEM DIRECTORY (OLD)
    1 IN JTFD (TEMP)
    1 IN NO DIRECTORY (NEW)

FOPEN

OLD

- FILE ALREADY EXISTS
- IT HAS AN ENTRY IN THE SYSTEM DIRECTORY



1) DETERMINE FILE'S CHARACTERISTICS USING FOPEN PARAMETERS, FILE SYSTEM DEFAULTS, AND FILE COMMAND (JFEQ) IF SUPPLIED AND ALLOWED.

2) SEARCH SYSTEM DIRECTORY USING GROUP AND ACCOUNT STRUCTURE. (USE DIRECTORY ENTRY TO GET FILE LABEL ADDRESS.)

3) READ FILE LABEL - ESTABLISH PHYSICAL CHARACTERISTICS OF FILE

4) BUILD CONTROL BLOCK ENTRIES (CBT) - USED TO MANAGE FILE

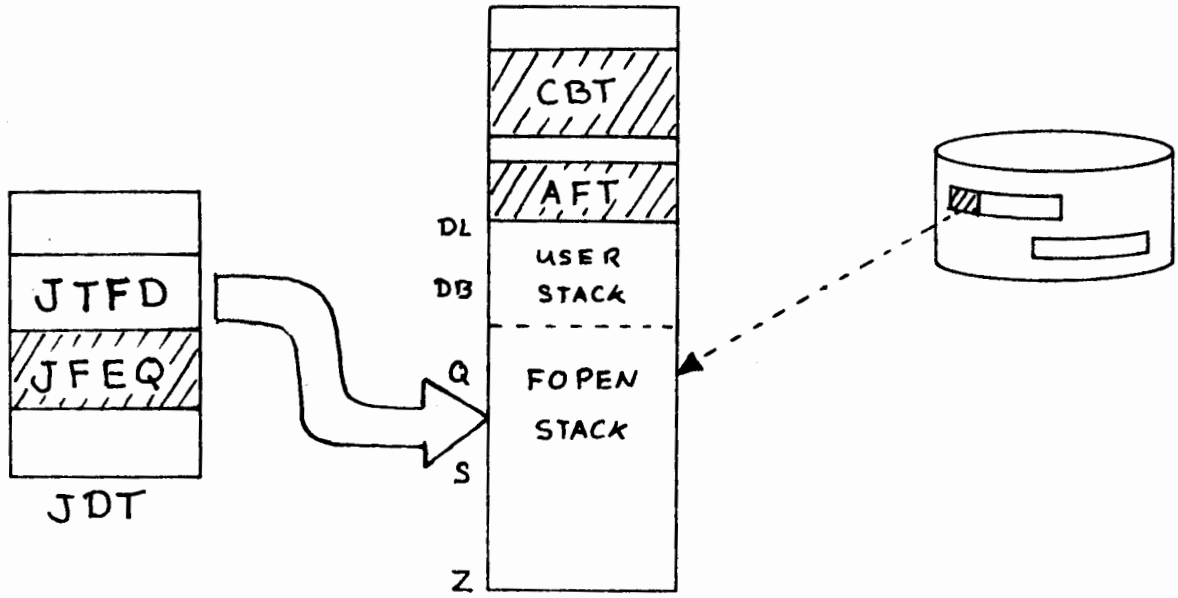
5) BUILD AFT ENTRY TO ACCESS CONTROL BLOCK BY FILE NUMBER

6) RETURN FILE NUMBER (AFT INDEX) TO CALLER

FOPEN

TEMP

- FILE ALREADY EXISTS
- IT HAS AN ENTRY IN THE JOB/SESSION JTFD



ALL OPERATIONS ARE THE SAME AS FOPEN (OLD)

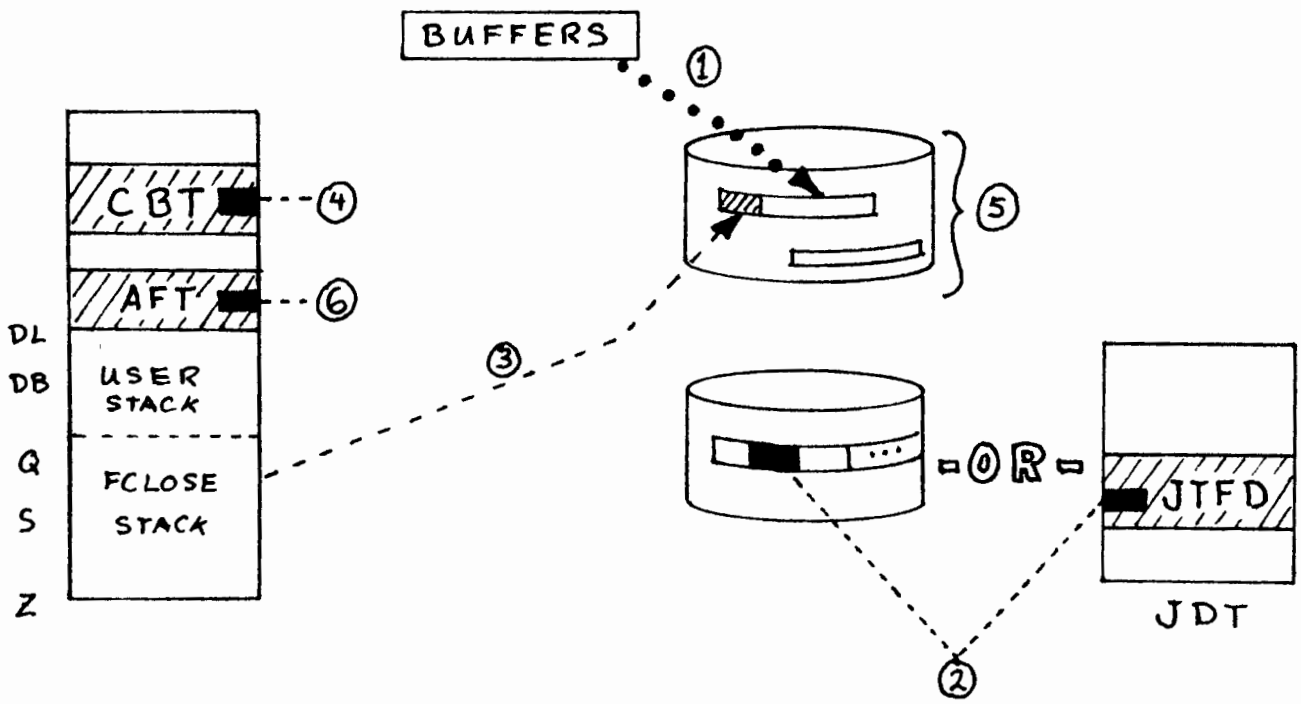
EXCEPT:

THE DIRECTORY ENTRY IS FOUND IN A JOB LOCAL EXTRA DATA SEGMENT - THE JOB DIRECTORY TABLE (JDT).

ENTRIES FOR TEMPORARY FILES RESIDE IN THE JOB TEMPORARY FILE DIRECTORY (JTFD) PORTION OF THE JDT.

FCLOSE

DELETE



1) POST BUFFERS TO THE DISC FILE

2) IF: OLD REMOVE ENTRY FROM SYSTEM DIRECTORY
TEMP REMOVE ENTRY FROM JTFD

3) UPDATE EOF IN THE FILE LABEL

4) REMOVE CONTROL BLOCK ENTRIES FROM CBT IN STACK
OR FILE SEGMENT (IF NOCB)

5) RETURN FILE SPACE TO DISC FREE SPACE

6) RESET AFT ENTRY FOR THE FILE

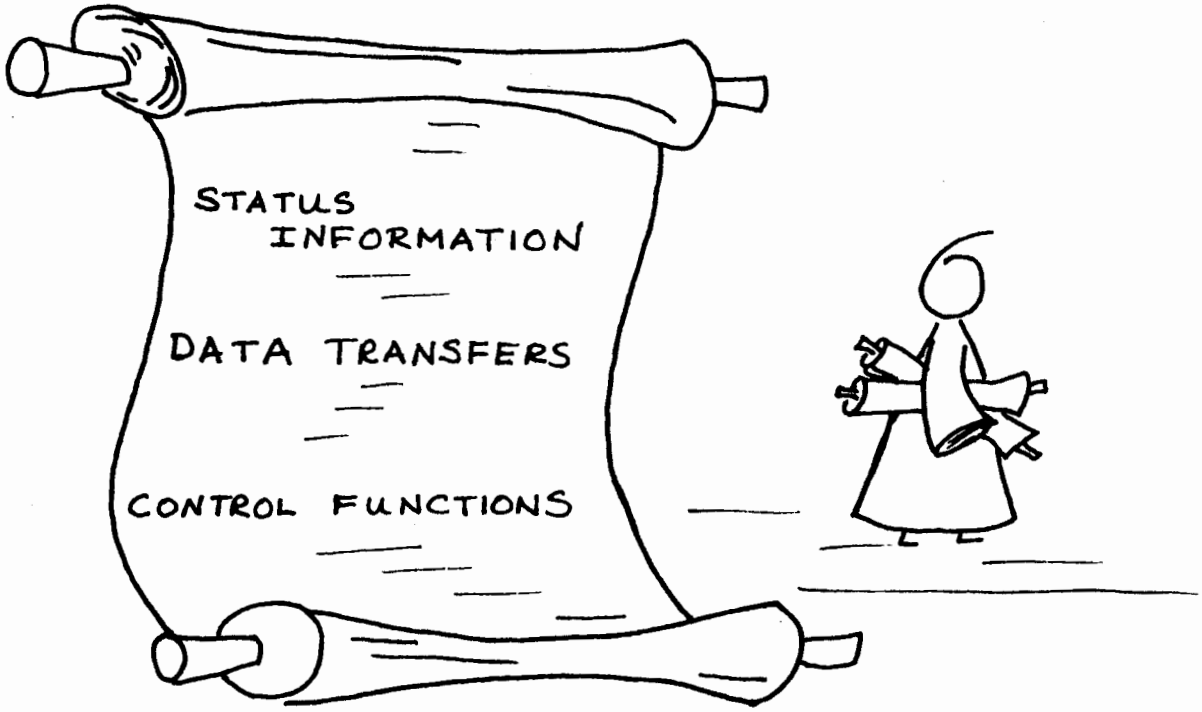
DEFAULT DISPOSITION

FILES FCLOSED WITH DEFAULT DISPOSITION REVERT TO THEIR PREVIOUS STATE (DISPOSITION)

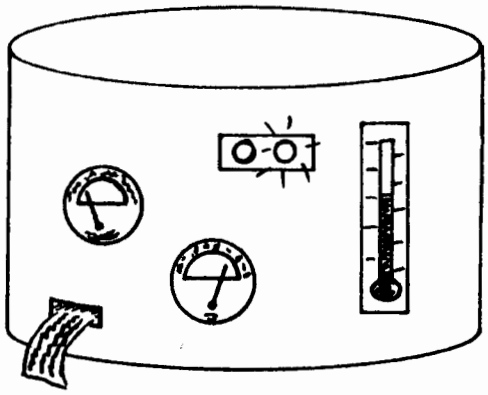
FOPEN		FCLOSE		
(UNDEFINED)	⇨	NEW	⇨	(UNDEFINED)
TEMP	⇨	TEMP	⇨	TEMP
OLD*	⇨	OLD	⇨	OLD

* PERMANENT FILE

DISC FILE OPERATIONS



STATUS INFORMATION



- WHAT KIND OF INFORMATION IS AVAILABLE?
- HOW CAN YOU GET THE INFORMATION?
- WHERE DOES IT COME FROM?

STATUS INFORMATION AVAILABLE

- **ACTUAL** FILE CHARACTERISTICS

  PHYSICAL
  OPERATIONAL } FOPEN + [:FILE + [LABEL]]

- **CURRENT** FILE INFORMATION

  EOF
  RECORD POINTER
  LOGICAL, PHYSICAL TRANSFER COUNT

- **ERROR** INFORMATION

  LAST ERROR FOR FILE
  LAST FOPEN ERROR

HEWLETT hp PACKARD

OBTAINING STATUS INFORMATION

PRINT, FILE, INFO

- ACTUAL FILE CHARACTERISTICS
- CURRENT FILE INFORMATION

} FGETINFO

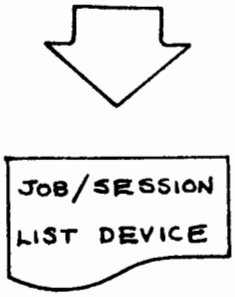
- ERROR INFORMATION

} FCHECK

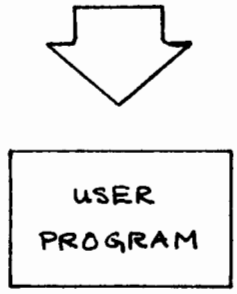
THE SAME STATUS INFORMATION MAY BE OBTAINED THROUGH DIFFERENT INTRINSIC CALLS

OBTAINING STATUS INFORMATION

PRINT' FILE' INFO



FGETINFO, FCHECK



STATUS INFORMATION:

- FORMATTED AND OUTPUT TO JOB/SESSION LIST DEVICE

-OR-

- RETURNED (UNFORMATTED) DIRECTLY TO USER PROGRAM

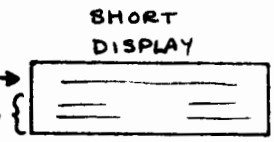
PRINT'FILE'INFO

IV
PRINT'FILE'INFO(*fnum*);

- REQUIRES FILE NUMBER RETURNED BY FOPEN (USE ZERO FOR FOPEN FAILURE)
- OUTPUT WILL ONLY BE PRINTED TO JOB/SESSION LIST DEVICE
- TWO FORMATS ("TOMBSTONES") POSSIBLE

FILE INFORMATION DISPLAY - FULL

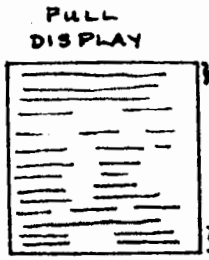
```
+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
!  FILE NAME IS SPL.PUB.SYS                       !
!  FOPTIONS: SYS,B,*FORMAL*,F,N,FEQ               !
!  AOPTIONS:  IN/OUT,SREC,NOLOCK,DEF,BUFFER       !
!  DEVICE TYPE: 0       DEVICE SUBTYPE: 3         !
!  LDEV: 2         DRT: 5          UNIT: 0        !
!  RECORD SIZE: 128     BLOCK SIZE: 128   (WORDS) !
!  EXTENT SIZE: 360     MAX EXTENTS: 1            !
!  RECPTR: 0            RECLIMIT: 359             !
!  LOGCOUNT: 0             PHYSCOUNT: 0           !
!  EOF AT: 359          LABEL ADDR: %00200262753  !
!  FILE CODE: 1029    ID IS MANAGER   ULABELS: 0  !
!  PHYSICAL STATUS: 1111000000000000              !
!  ERROR NUMBER: 42     RESIDUE: 0                !
!  BLOCK NUMBER: 0              NUMREC: 1         !
+-----------------------------------------------+
```



FILE IS OPEN

- FILE NUMBER REPRESENTS A CURRENTLY OPEN FILE
- ERROR INDICATES LAST ERROR ON FILE
- FULL DISPLAY CONDENSED WHEN FILE IS NOT OPEN

FILE INFORMATION DISPLAY - SHORT




```
+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
!  FILE NUMBER -1      IS UNDEFINED.              !
!  ERROR NUMBER: 52     RESIDUE: 0                !
!  BLOCK NUMBER: 0              NUMREC: 0         !
+-----------------------------------------------+
```

FILE NOT OPEN

- FILE NUMBER IS ZERO OR INVALID
- FOPEN FAILURE ASSUMED IF ZERO FILE NUMBER
  (FIRST LINE NOT PRINTED)
- ERROR IS ALWAYS LAST FOPEN ERROR
  (ALL OTHER FIELDS ARE ZERO)
- FIELDS MATCH FULL DISPLAY
  (FILE NAME REPLACED WITH FILE NUMBER)

FILE INFORMATION DISPLAY

```
+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
!                                                 !
! ① FILE NAME IS SPL.PUB.SYS                      !
!   FOPTIONS: SYS,B,*FORMAL*,F,N,FEQ              !
!   AOPTIONS:  IN/OUT,SREC,NOLOCK,DEF,BUFFER      !
!                                                 !
!   DEVICE TYPE: 0        DEVICE SUBTYPE: 3       !
!   LDEV: 2          DRT: 5          UNIT: 0      !
!   RECORD SIZE: 128      BLOCK SIZE: 128  (WORDS) ②
!   EXTENT SIZE: 360      MAX EXTENTS: 1          !
!                                                 !
! ③ RECPTR: 0           RECLIMIT: 359             !
!   LOGCOUNT: 0            PHYSCOUNT: 0             !
!   EOF AT: 359                                     !
!                                 LABEL ADDR: %00200262753
!   FILE CODE: 1029   ID IS MANAGER   ULABELS: 0  ④
!   PHYSICAL STATUS: 1111000000000000               !
!                                                 !
! ⑤ ERROR NUMBER: 42    RESIDUE: 0                !
!   BLOCK NUMBER: 0            NUMREC: 1          !
!                                                 !
+-------------------------------------------------+
```

① NAME AND OPTIONS

② DEVICE AND DATA STRUCTURE

③ TRANSFER INFORMATION

④ LABELS AND PHYSICAL STATUS

⑤ ERROR INFORMATION

NAME AND OPTIONS

```
FILE NAME IS SPL.PUB.SYS
FOPTIONS: SYS,B,*FORMAL*,F,N,FEQ
AOPTIONS:  IN/OUT,SREC,NOLOCK,DEF,BUFFER
```

- FILE NAME : FULLY QUALIFIED (NAME, GROUP, ACCOUNT)
- FOPTIONS : ACTUAL FOPTIONS IN EFFECT
- AOPTIONS : CURRENT AOPTIONS IN EFFECT

FOPTIONS KEYWORDS

DOMAIN	ASCII/ BINARY	DEFAULT DESIGNATOR	RECORD FORMAT	CARRIAGE CONTROL	DISALLOW : FILE
NEW	A	*FORMAL*	F	N	FEQ
SYS	B	$STDLIST	V	C	DEQ
JOB		$NEWPASS	U		
ALL		$OLDPASS	?		
		$STDIN			
		$STDINX			
		$NULL			

AOPTIONS KEYWORDS

ACCESS TYPE	MULTI-RECORD	DYNAMIC LOCKING	EXCLUSIVE ACCESS	INHIBIT BUFFERING
INPUT	SREC	NOLOCK	DEF	BUFFER
OUTPUT	MREC	LOCK	EXC	NOBUFF
OUTKEEP			SEA*	
APPEND			SHR	
IN/OUT				
UPDATE			* SEMI-EXCLUSIVE ACCESS (EAR)	

NOTE: MULTI ACCESS, NO WAIT FIELDS NOT REPRESENTED

DEVICE AND DATA STRUCTURE

```
DEVICE TYPE: 0        DEVICE SUBTYPE: 3
LDEV: 2          DRT: 5          UNIT: 0
RECORD SIZE: 128      BLOCK SIZE: 128    (WORDS)
EXTENT SIZE: 360      MAX EXTENTS: 1
```

- DEVICE TYPE, SUBTYPE
  LDEV, DRT, UNIT } HARDWARE INFORMATION (SET AT CONFIGURATION)
- RECORD SIZE : LOGICAL RECORD SIZE (WORDS/BYTES)
  [FOR VARIABLE, DOES NOT INCLUDE 2 WORDS ADDED]
- BLOCK SIZE : PHYSICAL RECORD SIZE (WORDS/BYTES)
  [DOES NOT INCLUDE 2 WORDS ADDED FOR VARIABLE]
- EXTENT SIZE : NUMBER OF **SECTORS** PER EXTENT
- MAX EXTENTS : MAXIMUM ALLOWED FOR FILE

TRANSFER INFORMATION

```
RECPTR: 0                    RECLIMIT: 359
LOGCOUNT: 0                     PHYSCOUNT: 0
EOF AT: 359
```

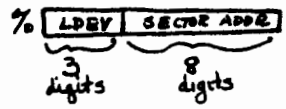
- RECPTR: CURRENT RECORD POINTER (LOGICAL OR PHYSICAL) POINTS TO NEXT RECORD TO BE TRANSFERRED
- RECLIMIT: MAXIMUM NUMBER OF RECORDS IN FILE
- LOGCOUNT: NUMBER OF LOGICAL RECORD TRANSFERS TO/FROM USER STACK SINCE FOPEN
- PHYSCOUNT: NUMBER OF PHYSICAL RECORD TRANSFERS TO/FROM FILE (DISC) SINCE FOPEN
- EOF: CURRENT EOF POINTER (ONE PLUS LARGEST LOGICAL RECORD NUMBER EVER USED TO WRITE DATA TO THE FILE)

NOTE: RECPTR, LOGCOUNT, PHYSCOUNT, EOF START AT Ø

LABELS AND PHYSICAL STATUS

```
                          LABEL ADDR: %00200262753
FILE CODE: 1029    ID IS MANAGER    ULABELS: 0
PHYSICAL STATUS: 1111000000000000
```

- LABEL ADDR : SECTOR ADDRESS AND LDEV NO. FOR FILE LABEL



- FILE CODE : USER OR SYSTEM DEFINED (BLANK IF Ø)
- ID : USER NAME OF CREATOR
- ULABELS : MAXIMUM NUMBER OF USER LABELS ALLOWED
- PHYSICAL STATUS : STATUS OF DISC AT TIME OF LAST INTERRUPT [MEANINGLESS FOR DISC IN MULTIPROGRAMMING ENVIRONMENT]

ERROR INFORMATION

```
ERROR NUMBER: 42     RESIDUE: 0
BLOCK NUMBER: 0            NUMREC: 1
```

- ERROR NUMBER : LAST ERROR FOR FILE (Ø MEANS EOF DETECTED; IF NO ERROR HAS OCCURRED, ERROR CODE IS RANDOM)

- RESIDUE : 1) NUMBER OF WORDS/BYTES **NOT** TRANSFERRED AFTER ERROR WAS DETECTED
  2) IN CASE OF EOF, NUMBER OF WORDS/BYTES TRANSFERRED **BEFORE** EOF WAS DETECTED

- BLOCK NUMBER: ERROR DETECTED IN THIS BLOCK

- NUMREC : NUMBER OF LOGICAL RECORDS IN 'ERROR' BLOCK

NOTE: BLOCK NUMBER STARTS AT ZERO

SOURCES FOR FILE INFORMATION DISPLAY

FGETINFO

```
+-F-I-L-E---I-N-F-O-R-M-A-T-I-O-N---D-I-S-P-L-A-Y+
!  FILE NAME IS SPL.PUB.SYS                        !
!  FOPTIONS: SYS,B,*FORMAL*,F,N,FEQ                !
!  AOPTIONS:  IN/OUT,SREC,NOLOCK,DEF,BUFFER        !
!  DEVICE TYPE: 0       DEVICE SUBTYPE: 3          !
!  LDEV: 2         DRT: 5          UNIT: 0         !
!  RECORD SIZE: 128     BLOCK SIZE: 128    (WORDS) !
!  EXTENT SIZE: 360     MAX EXTENTS: 1             !
!  RECPTR: 0            RECLIMIT: 359              !
!  LOGCOUNT: 0             PHYSCOUNT: 0            !
!  EOF AT: 359          LABEL ADDR: %00200262753   !
!  FILE CODE: 1029    ID IS MANAGER    ULABELS: 0  !
!  PHYSICAL STATUS: 1111000000000000                !
!  ERROR NUMBER: 42     RESIDUE: 0                 !
!  BLOCK NUMBER: 0              NUMREC: 1          !
+-------------------------------------------------+
```

FCHECK

STATUS INFORMATION DISPLAYED BY PRINT'FILE'INFO
OBTAINED THROUGH PARAMETERS OF
FGETINFO , FCHECK

NOTE: PHYSICAL STATUS (NOT USEFUL FOR DISC) OBTAINED THROUGH FCONTROL

FGETINFO

```
                IV      BA      L      L      I      I      L      L
FGETINFO(filenum,filename,foptions,aoptions,recsize,devtype,ldnum,hdaddr,

                I    D  D    D        D        D      I      L
        filecode,recpt,eof,flimit,logcount,physcount,blksize,extsize,

                I        I      BA      D  0-V
        numextents,userlabels,creatorid,labaddr);
```

- REQUIRES FILE NUMBER FOR OPEN FILE
- INFORMATION RETURNED THROUGH PARAMETERS
- ZERO OR INVALID FILE NUMBER RESULTS IN CCL

FCHECK

```
                IV        I   I      D       I   0-V
FCHECK(filenum,errorcode,tlog,blknum,numrecs);
```

- USES FILE NUMBER RETURNED BY FOPEN
  (ASSUMES FOPEN FAILURE IF *filenum* IS ZERO OR OMITTED)
- INFORMATION RETURNED THROUGH PARAMETERS
  (*errorcode* IS **LAST** ERROR FOR FILE OR **LAST** FOPEN ERROR)
- INVALID FILE NUMBER : CCL SET AND 72 RETURNED IN *errorcode*
  (FILE NUMBER IS INVALID IF ≠0 AND THERE IS NO CORRESPONDING OPEN FILE)

FGETINFO/PRINT'FILE'INFO

PARAMETER / FIELD RELATIONSHIP

FGETINFO PARAMETERS	PRINT'FILE'INFO FIELDS
FILENAME ----	FILE NAME
FOPTIONS	FOPTIONS
AOPTIONS	AOPTIONS
RECSIZE ----	RECORD SIZE
DEVTYPE	DEVICE TYPE, SUBTYPE
LDNUM	LDEV
HDADDR	DRT, UNIT
FILECODE ----	FILE CODE
RECPTR	RECPTR
EOF	EOP AT
FLIMIT	REC LIMIT

FGETINFO PARAMETERS	PRINT'FILE'INFO FIELDS
LOGCOUNT ----	LOGCOUNT
PHYSCOUNT	PHYSCOUNT
BLKSIZE	BLOCK SIZE
EXTSIZE	EXTENT SIZE
NUMEXTENTS	MAX EXTENTS
USERLABELS ----	ULABELS
CREATORID	ID IS
LABADDR	LABEL ADDR

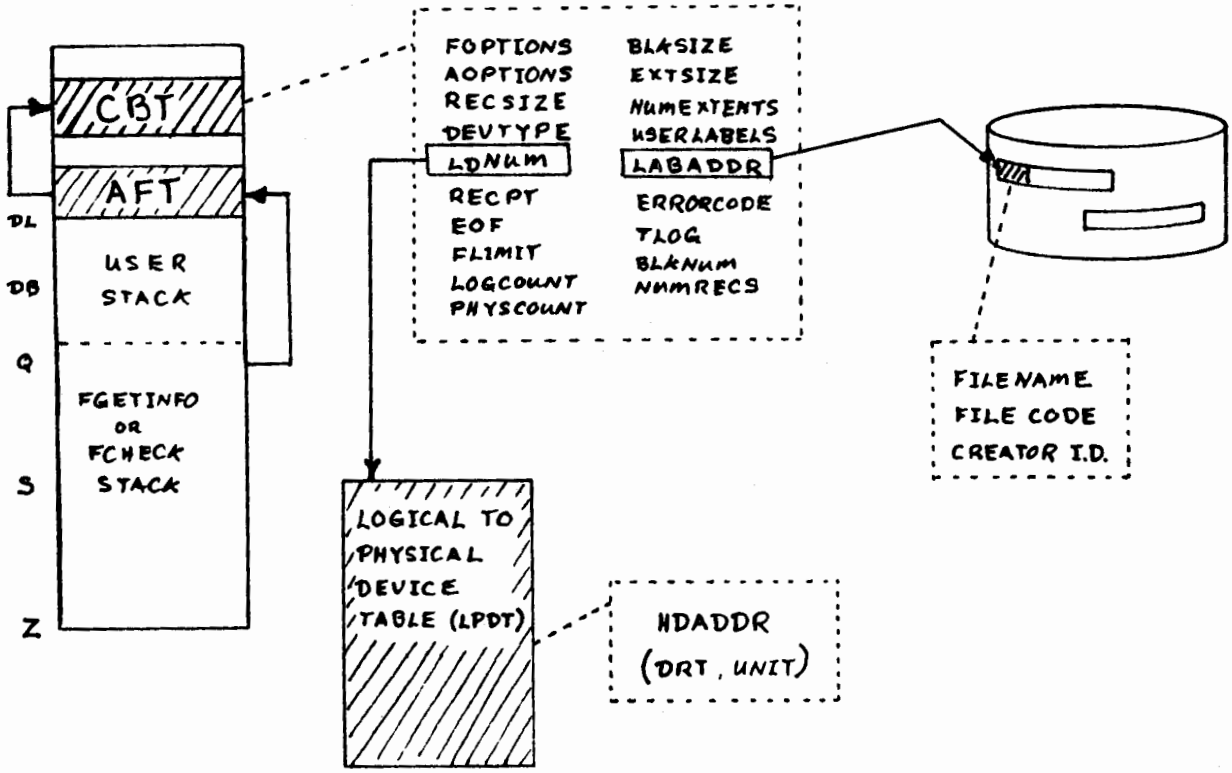
FCHECK / PRINT'FILE'INFO

PARAMETER / FIELD RELATIONSHIP

FCHECK PARAMETERS	PRINT'FILE'INFO FIELDS
ERRORCODE	ERROR NUMBER
TLOG	RESIDUE
BLKNUM	BLOCK NUMBER
NUMRECS	NUMREC

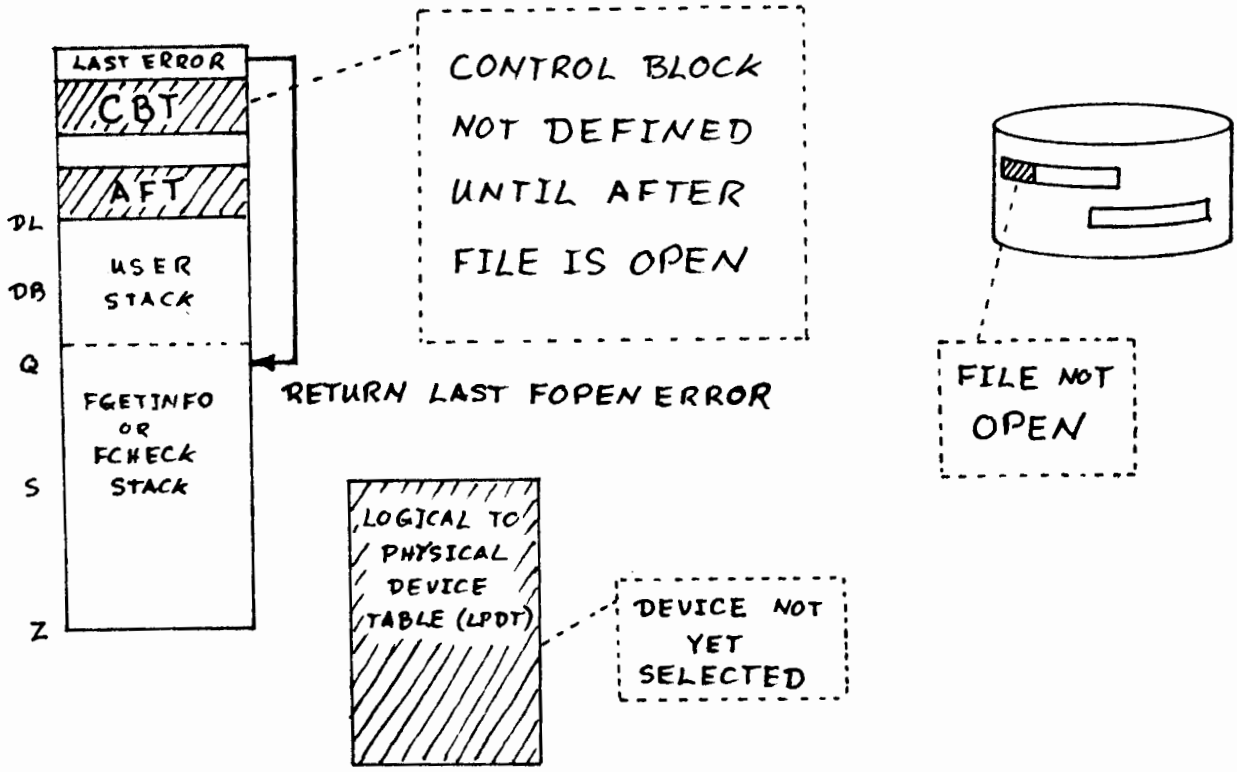
FGETINFO/FCHECK - SOURCES

FILE IS OPEN



FGETINFO/FCHECK - SOURCES

FILE IS NOT OPEN



FILE SYSTEM LAB #1

WRITE AN SPL PROGRAM TO:

1. OPEN A NEW FILE AS FOLLOWS:

NAME:	LAB1F
FOPTIONS:	ASCII
AOPTIONS:	READ/WRITE
RECSIZE:	40
USERLABELS:	2
BLOCKFACTOR:	16
FILESIZE:	16
FILECODE:	200

2. FOLLOWING FOPEN, CALL PRINT'FILE'INFO TO DETERMINE:
   a. FINAL FOPTIONS, AOPTIONS
   b. EXTENT SIZE
   c. EOF

3. CLOSE THE FILE (DEFAULT DISPOSITION). WAS THE FILE SAVED? WHY OR WHY NOT?

4. CALL PRINT'FILE'INFO; WHAT FORM IS PRINTED? WHY?

FILE SYSTEM LAB #2

USING THE PROGRAM FROM FILE SYSTEM LAB #1:

1. OPEN FILE LAB1F AS BEFORE. TEST CONDITION CODE AFTER FOPEN; CALL PRINT'FILE'INFO AND QUIT <u>ONLY</u> IF CCL IS RETURNED.

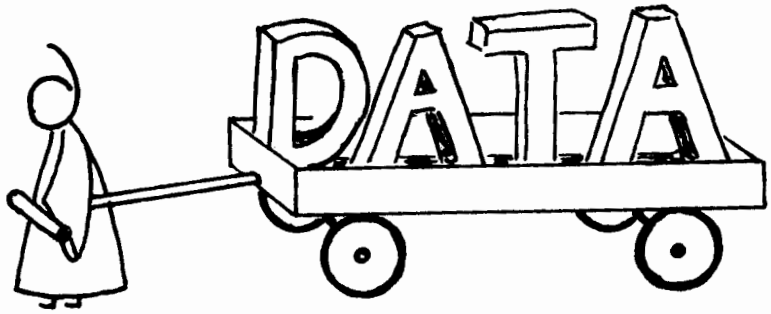
2. WRITE A BUFFER ("ABCDEFGHIJKLMNOPQRSTUVWXYZ") TO EACH RECORD IN THE FILE (TEST FOR END OF FILE).

3. WHEN EOF IS DETECTED, CALL PRINT'FILE'INFO. WHERE ARE THE POINTERS SET TO?

4. CLOSE THE FILE, PERMANENT DISPOSITION.

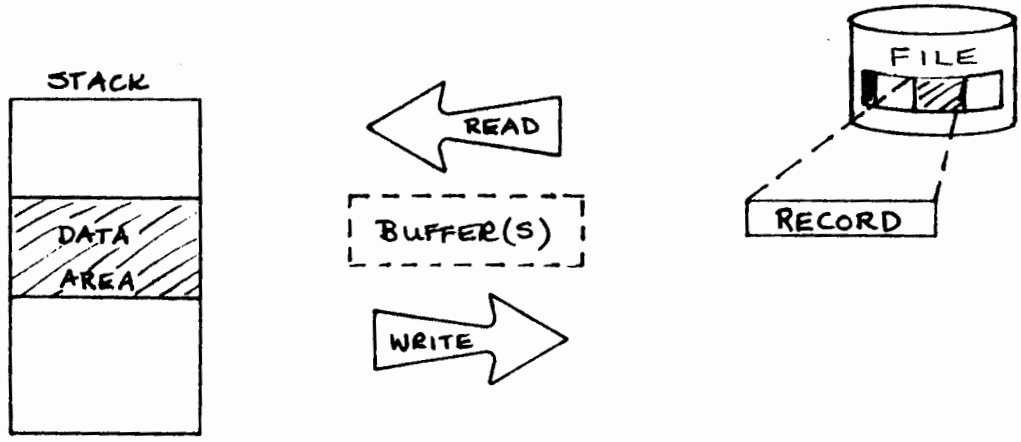
5. DO A LISTF,2 ON THE FILE.

DATA TRANSFER



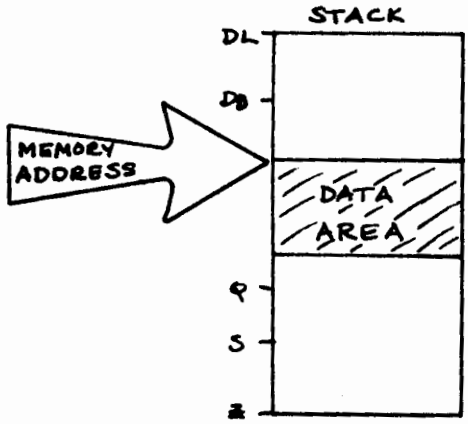
- WHAT ARE THE ELEMENTS OF A DATA TRANSFER?
- HOW ARE RECORDS SELECTED FOR TRANSFER?
- WHAT ARE THE INTRINSICS USED FOR DATA TRANSFER OPERATIONS? (HOW ARE THEY USED?)

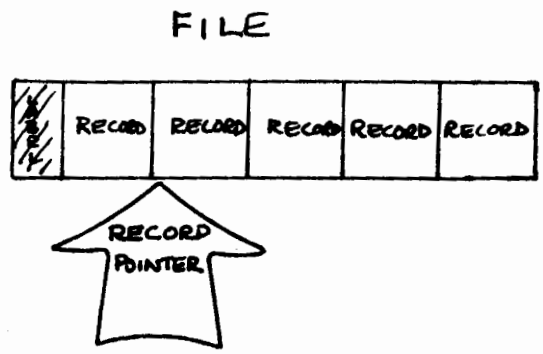
TRANSFER ELEMENTS



- READ : SOURCE IS RECORD, DESTINATION IS DATA AREA
- WRITE : SOURCE IS DATA AREA, DESTINATION IS RECORD
- ADDRESS OF SOURCE AND DESTINATION REQUIRED

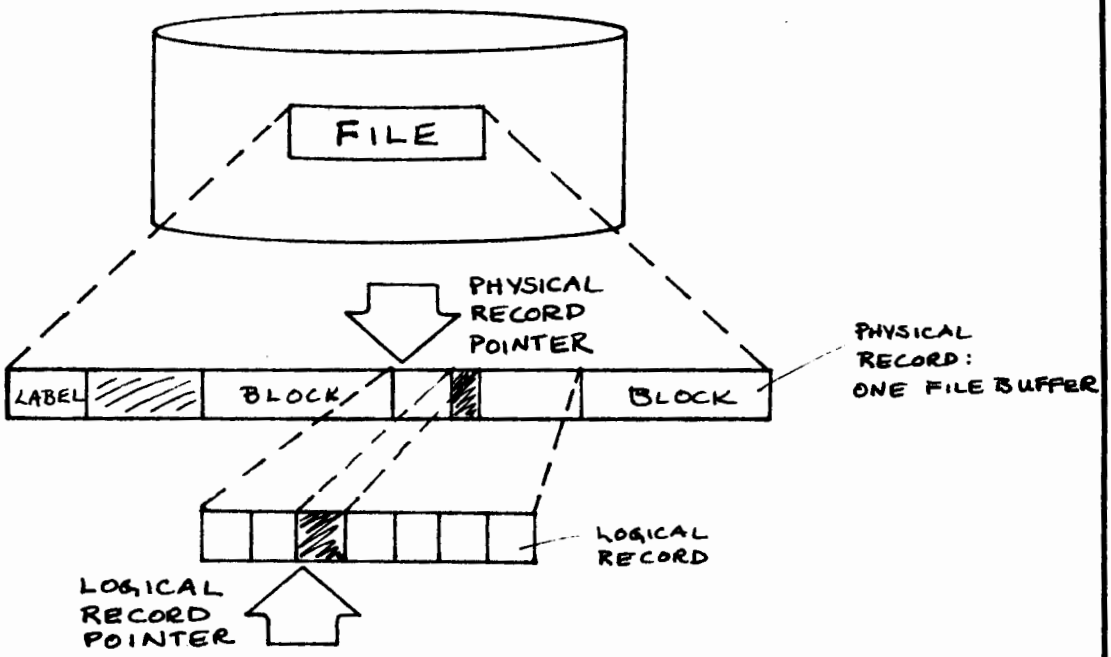
SOURCE/DESTINATION ADDRESSING





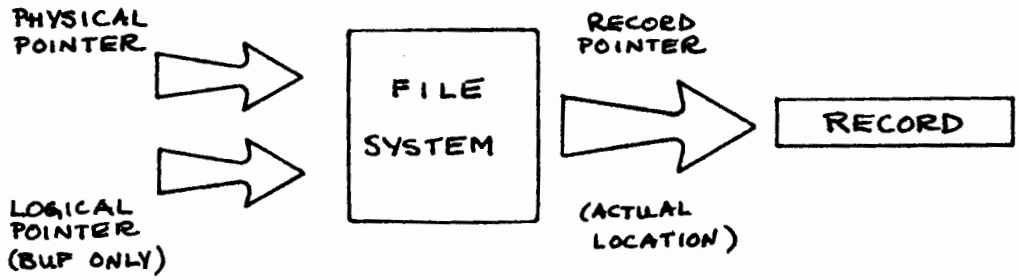
- MEMORY ADDRESS : LOCATES AN AREA IN USER STACK BETWEEN DL AND S
- RECORD POINTER : LOCATES RECORD RELATIVE TO START OF FILE

RECORD POINTERS



- PHYSICAL POINTER: USED TO LOCATE PHYSICAL RECORD ON DISC
- LOGICAL POINTER: USED FOR BLOCKING/DEBLOCKING LOGICAL RECORDS IN FILE BUFFER
- NOBUF FILES: PHYSICAL POINTERS (ONLY)

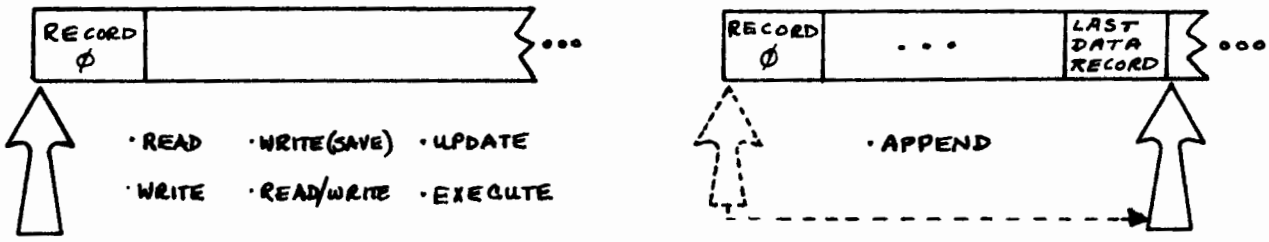
POINTER ADDRESSING



- FILE SYSTEM COMBINES PHYSICAL AND LOGICAL POINTERS TO LOCATE RECORD

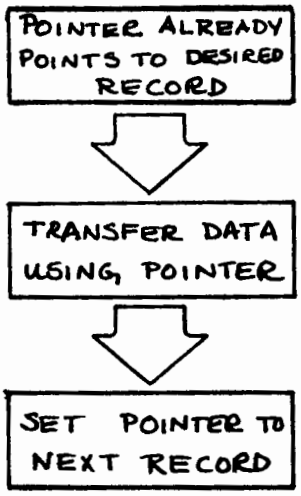
- FUTURE REFERENCES (IN THIS MATERIAL) TO "RECORD POINTER" : IMPLIES COMBINATION

POINTER INITIALIZATION (FOPEN)



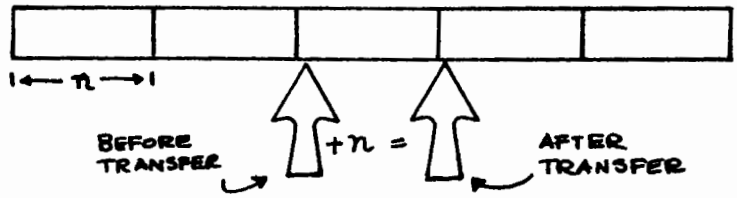
- FOPEN SETS THE RECORD POINTER TO RECORD Ø FOR ALL OPERATIONS
- APPEND ONLY: POINTER MOVED TO END OF FILE PRIOR TO WRITING DATA
- POINTER READY TO BE MODIFIED BY INTRINSICS USED TO SELECT RECORDS

RECORD SELECTION (DEFAULT)

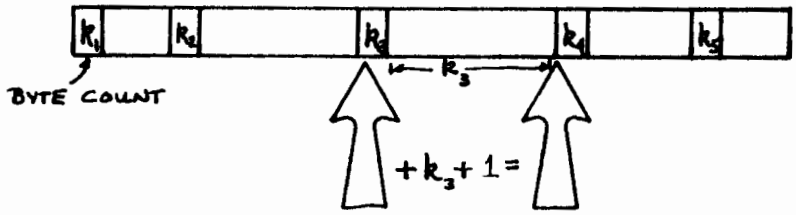


- POINTER UPDATE DEPENDS ON RECORD TYPE

FIXED : UNIFORM RECORD LENGTH
UNDEFINED : ADDED TO POINTER



VARIABLE : NO UNIFORM LENGTH; LENGTH TAKEN FROM TRANSFERRED RECORD - ADDED TO POINTER



- INTRINSICS USED FOR TRANSFER WITH DEFAULT RECORD SELECTION ARE FREAD, FWRITE

RECORD SELECTION (CONTROLLED)

SPECIFY RELATIVE RECORD NUMBER

↓

CALCULATE NEW POINTER SETTING

↓

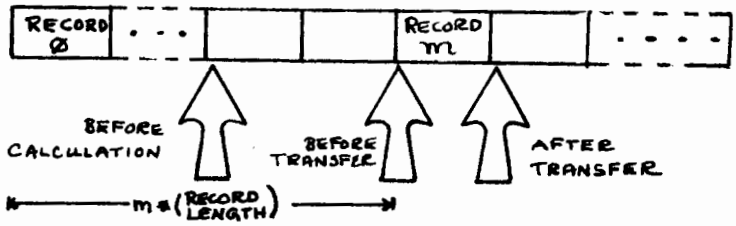
TRANSFER DATA USING POINTER

↓

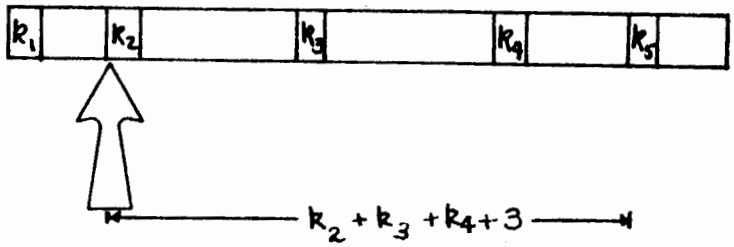
SET POINTER TO NEXT RECORD

- NOT AVAILABLE FOR ALL RECORD TYPES
- POINTER UPDATE SAME AS DEFAULT

FIXED : UNIFORM RECORD LENGTH
UNDEFINED : USED WITH RELATIVE RECORD NUMBER TO SET NEW POINTER VALUE



VARIABLE : NO UNIFORM LENGTH
RECORD LENGTH ONLY WHEN RECORD IS ACTUALLY TRANSFERRED



- INTRINSICS USED FOR TRANSFERS WITH SPECIFIED RECORD NUMBER ARE FREADDIR, FWRITEDIR, FREADSEEK

RECORD SELECTION (UPDATE)

SET POINTER TO PREVIOUS RECORD

↓

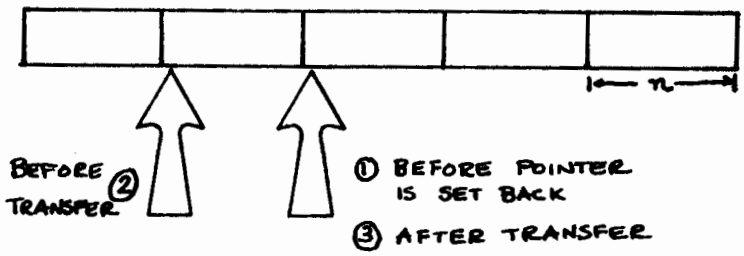
WRITE DATA USING POINTER

↓

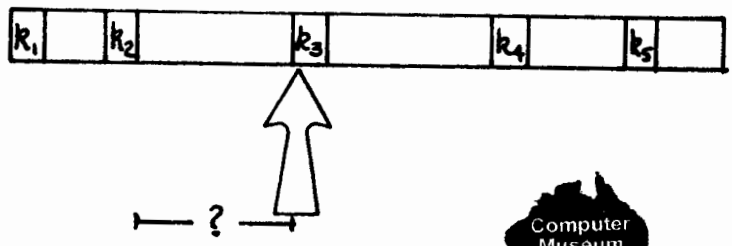
SET POINTER TO NEXT RECORD

- NOT AVAILABLE FOR ALL RECORD TYPES
- POINTER UPDATE SAME AS FOR DEFAULT
- PREVIOUS READ IMPLIED

FIXED / UNDEFINED : UNIFORM RECORD LENGTH SUBTRACTED FROM POINTER



VARIABLE : NO UNIFORM LENGTH
PREVIOUS RECORD LENGTH NOT RETAINED



- INTRINSIC USED FOR UPDATE SELECTION IS FUPDATE

RECORD SELECTION SUMMARY

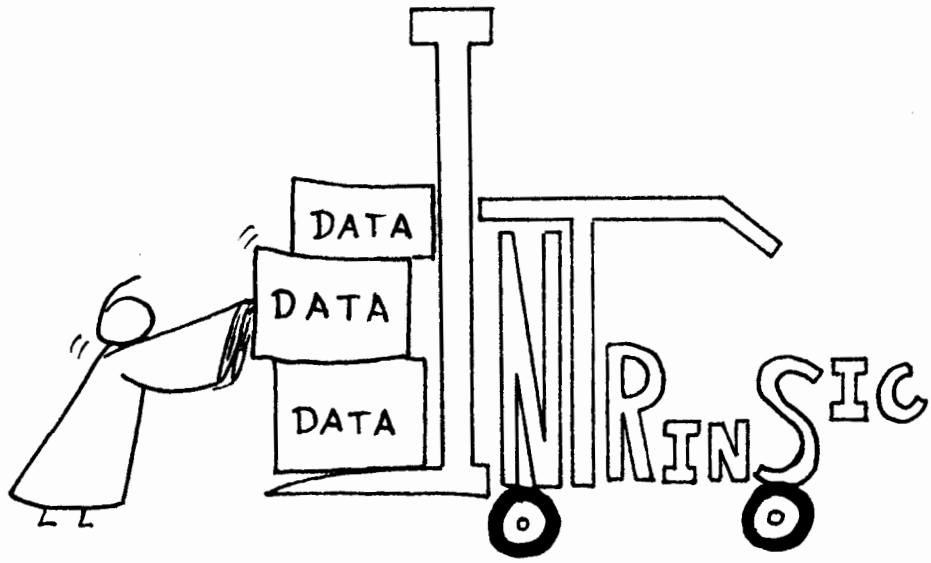
SELECTION METHOD	PERMISSIBLE RECORD TYPE	FOPEN ACCESS ALLOWED	ASSOCIATED INTRINSIC(S)
DEFAULT (SEQUENTIAL)	F, V, U	ANY	FREAD, FWRITE
CONTROLLED (DIRECT)	F, U	ANY EXCEPT APPEND	FREADDIR FWRITEDIR FREADSEEK
UPDATE	F, U	UPDATE (ONLY)	FUPDATE

F: FIXED, V: VARIABLE, U: UNDEFINED

RECORD SELECTION SUMMARY

FOPEN ACCESS SPECIFIED	UPDATE	APPEND	READ, WRITE, WRITE(SAVE) READ/WRITE, EXECUTE		
RECORD TYPE ALLOWED	F, U	F, U	F, U	V	
	ALL INTRINSICS	FWRITE ONLY	ALL AS APPROPRIATE EXCEPT FUPDATE	FREAD, FWRITE AS APPROPRIATE	INTRINSICS ALLOWED
	SEQUENTIAL DIRECT UPDATE	SEQUENTIAL ONLY	SEQUENTIAL DIRECT	SEQUENTIAL ONLY	SELECTION METHODS ALLOWED

INTRINSICS FOR DATA TRANSFER



- SEQUENTIAL
- UPDATE
- DIRECT
- ANTICIPATORY

FREAD · FWRITE · FREADDIR · FWRITEDIR · FUPDATE · FREADSEEK

FREAD

```
   I                 IV   LA     IV
lgth:=FREAD(filenum,target,tcount);
```

- SEQUENTIAL READ
- USE WITH F, V, U RECORDS
- FOPEN ACCESS MUST BE READ, READ/WRITE, OR UPDATE
- SUCCESSFUL READ RETURNS CCE AND TRANSFER LENGTH
- FILE ERROR RESULTS IN CCL
- END OF FILE RESULTS IN CCG, (ZERO LENGTH RETURNED)

FWRITE

```
                IV   LA    IV    LV
FWRITE(filenum,target,tcount,control);
```

- SEQUENTIAL WRITE
- USE WITH F, V, U RECORDS
- FOPEN ACCESS MUST BE WRITE, WRITE (SAVE), APPEND, READ/WRITE, OR UPDATE
- SUCCESSFUL WRITE RESULTS IN CCE.
- FILE ERROR RESULTS IN CCL, EOF IN CCG

FREADDIR

```
                    IV   LA    IV    DV
FREADDIR(filenum,target,tcount,recnum);
```

- DIRECT READ
- USE ONLY WITH F,U RECORDS
- FOPEN ACCESS MUST BE READ, READ/WRITE, OR UPDATE
- SPECIFY LOGICAL RECORD NUMBER (BUF FILES) OR PHYSICAL RECORD NUMBER (NOBUF FILES)
- SUCCESSFUL READ: CCE, FILE ERROR: CCL
  EOF: CCG

RECNUM

WITH NOBUF WOULD REFER

TO PHYS RECNUM

FWRITEDIR

HEWLETT PACKARD

```
                    IV   LA    IV    DV
FWRITEDIR(filenum,target,tcount,recnum);
```

- DIRECT WRITE
- USE ONLY WITH F,U RECORDS
- FOPEN ACCESS MUST BE WRITE, WRITE(SAVE), READ/WRITE OR UPDATE (NOT APPEND)
- LOGICAL RECORD NUMBER (BUF FILES) OR PHYSICAL RECORD NUMBER (NOBUF FILES)
- SUCCESSFUL WRITE: CCE, FILE ERROR: CCL, EOF: CCG

SPECIAL EOF CONSIDERATIONS

MULTIRECORD TRANSFERS

- FREAD, FREADDIR : LIMITED TO READ UP TO EOF
- FWRITE, FWRITEDIR: LIMITED TO WRITE UP TO FILE LIMIT
- WHEN TRANSFER EXCEEDS LIMIT (EOF/FILE)
  - CCG IS RETURNED
  - DATA UP TO LIMIT IS TRANSFERRED
  - FREAD RETURNS ZERO LENGTH (USE FCHECK TO GET ACTUAL TRANSFER LENGTH.)

CAN CHANGE LIMIT USING "DISKEDIT2"

FUPDATE

```
                    IV    LA    IV
FUPDATE(filenum,target,tcount);
```

- UPDATE PREVIOUS RECORD (LOGICAL/PHYSICAL)
- USE WITH F,U RECORDS
- FOPEN ACCESS MUST BE UPDATE
- SUCCESSFUL UPDATE: CCE , FILE ERROR: CCL
  EOF: CCG
- NO MULTIRECORD UPDATE ALLOWED

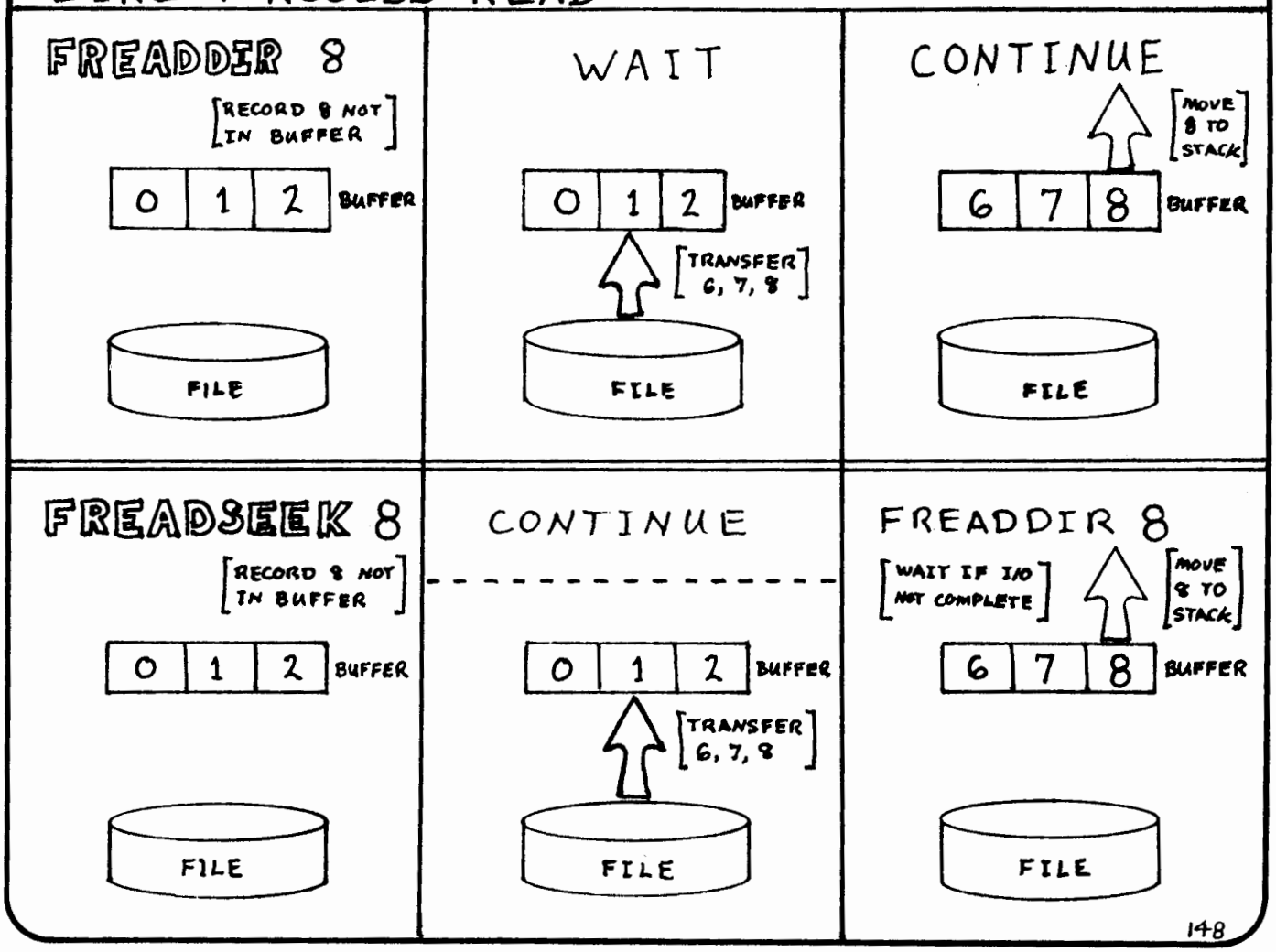
FREADSEEK

IV DV
FREADSEEK(*filenum,recnum*);

- ANTICIPATORY DIRECT READ INTO FILESYSTEM BUFFERS (BUF FILES ONLY)
- USE WITH F,U RECORDS
- FOPEN ACCESS MUST BE READ, READ/WRITE, OR UPDATE
- SPECIFY <u>LOGICAL</u> RECORD
- SUCCESSFUL READ: CCE, FILE ERROR: CCL, EOF: CCG

LIKE READ NO WAIT

DIRECT ACCESS READ



EXAMPLE: DATA TRANSFER INTRINSICS

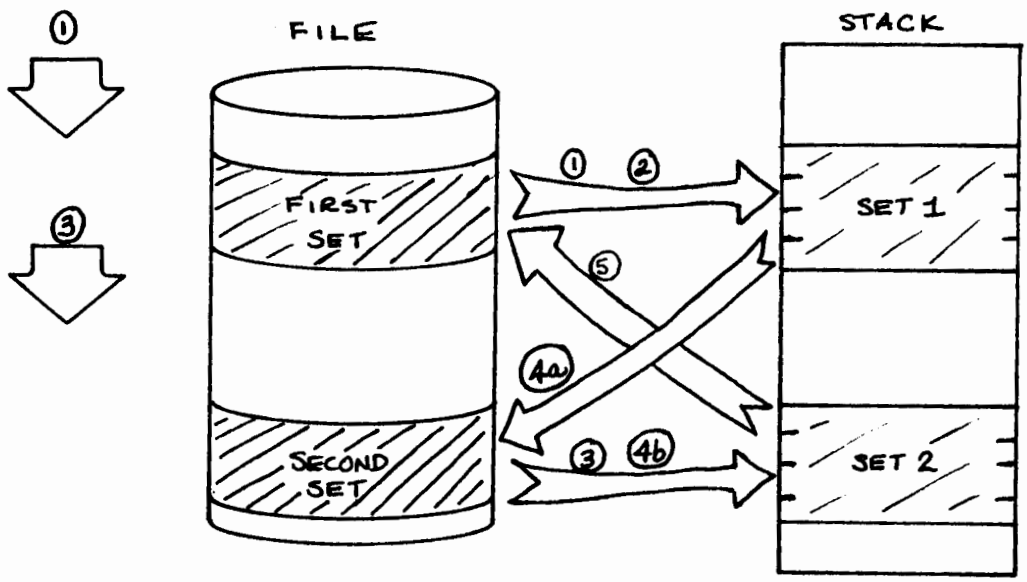
WHAT IT DOES

- ALLOWS USER TO IDENTIFY TWO SETS OF RECORDS IN A FILE
- INTERCHANGES POSITIONS OF THE SETS IN THE FILE

WHAT IT SHOWS

- SEQUENTIAL, DIRECT, AND UPDATE RECORD SELECTION
- MIXING RECORD SELECTION MODES
- INTRINSICS FREAD, FWRITE, FREADDIR, FWRITEDIR, FUPDATE, FREADSEEK

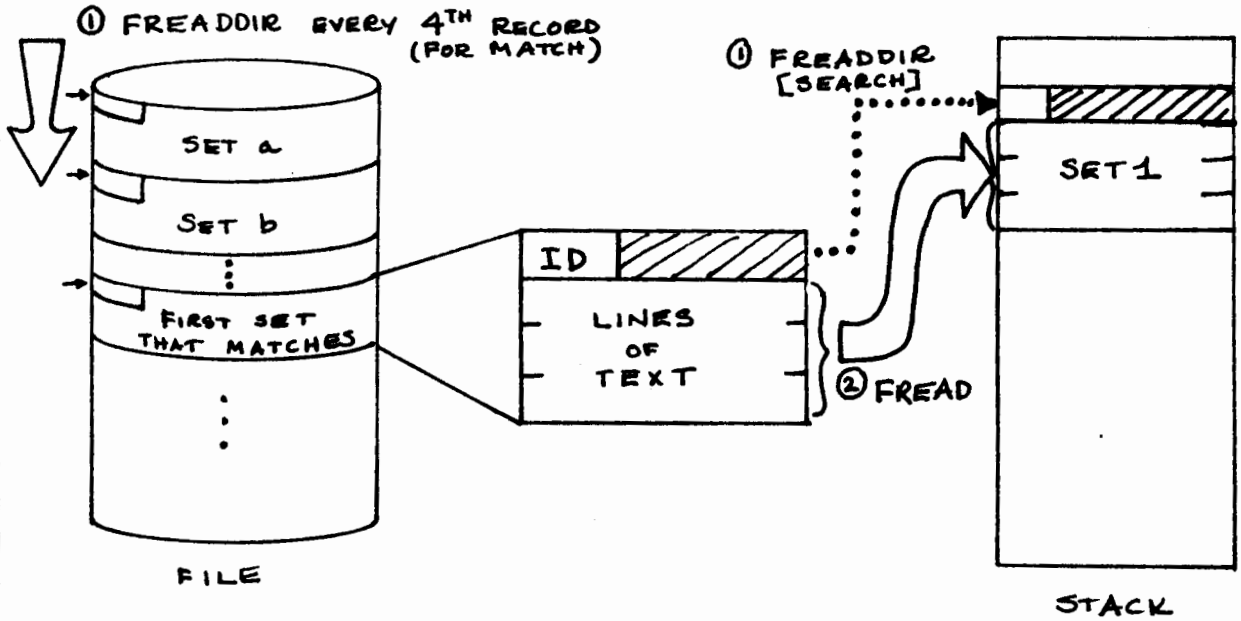
OVERVIEW OF EXAMPLE



1. SEARCH FOR FIRST SET
2. COPY FIRST SET TO STACK
3. SEARCH FOR SECOND SET
4. COPY SECOND SET INTO STACK AND UPDATE WITH SET 1 (FROM STACK) RECORD BY RECORD
5. COPY SET 2 FROM STACK TO ORIGINAL POSITION OF FIRST SET IN FILE

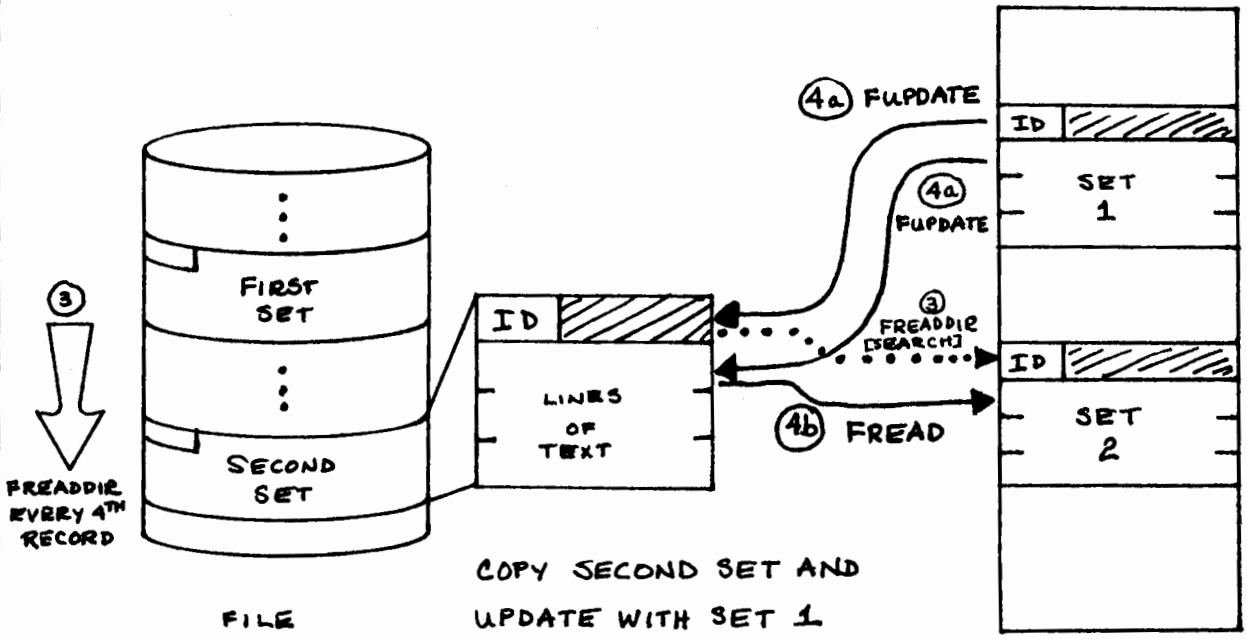
OVERVIEW OF EXAMPLE

① & ② : SEARCH FOR FIRST SET AND COPY INTO STACK



OVERVIEW OF EXAMPLE

③ & ④ SEARCH FOR SECOND SET AND COPY INTO STACK
UPDATE SET 1 FROM STACK INTO POSITION OF
SECOND SET



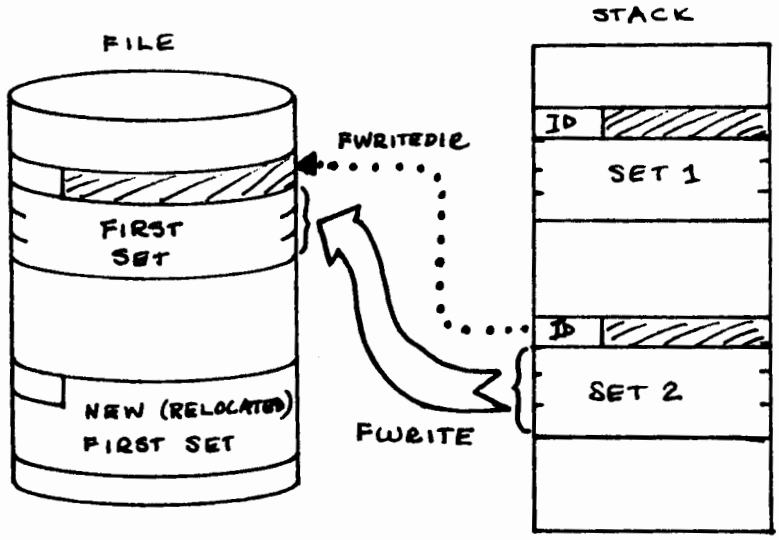
FILE

STACK

COPY SECOND SET AND
UPDATE WITH SET 1
(FREADDIR, FUPDATE, FREAD, FUPDATE,
FREAD, ...ETC.)

OVERVIEW OF EXAMPLE

(5) : COPY SET 2 TO ORIGINAL POSITION OF FIRST SET IN THE FILE



EXAMPLE - LISTING (1)

```
PAGE 0001   HEWLETT-PACKARD 32100A.06.3  SPL[4W]  MON, JAN 31, 1977,  2:55 PM  (C) HEWLETT-

00001000  00000 0   $CONTROL USLINIT
00002000  00000 0   BEGIN
00003000  00000 1   EQUATE UPDATE=5,OLDFILE=3;
00004000  00000 1   LOGICAL ID1,ID2,IDLEFT;
00005000  00000 1   INTEGER NF,I:=0;
00006000  00000 1   DOUBLE RECNO:=-40,FIRSTSET;
00007000  00000 1
00008000  00000 1   ARRAY SET2(0:159),SET1(*)=SET2(32),MSG(*)=SET2,
00009000  00000 1         ID1Q(0:2):="ID1:= ";
00010000  00003 1   ARRAY ID2Q(0:2):="ID2:= ";
00011000  00003 1   BYTE ARRAY FILENAME(0:8):="FORMS ";
00012000  00004 1
00013000  00004 1   DEFINE CCL=IF < THEN ERROR#,
00014000  00004 1          EOF=>#;
00015000  00004 1
00016000  00004 1   INTRINSIC PRINT'FILE'INFO,FOPEN,FREAD,FREADDIR,FREADSEEK,
00017000  00004 1             FWRITE,FWRITEDIR,FUPDATE,QUIT,PRINT,READ;
00018000  00004 1
00019000  00004 1   SUBROUTINE ERROR(NUM);
00020000  00000 1   VALUE NUM; INTEGER NUM;
00021000  00000 1   BEGIN
00022000  00000 2     PRINT'FILE'INFO(NF);
00023000  00002 2     QUIT(NUM);
00024000  00004 2   END; << ERROR >>
00025000  00005 1
00026000  00005 1   <<.............................................................>>
00027000  00005 1
00028000  00005 1   ASK'FOR'ID1:
00029000  00005 1      PRINT(ID1Q,-5,%320);                    <<REQUEST 1ST ID>>
00030000  00011 1      IF READ(ID1,-2)=0 THEN RETURN;          <<INPUT 1ST ID>>
00031000  00020 1   ASK'FOR'ID2:
00032000  00020 1      PRINT(ID2Q,-5,%320);                    <<REQUEST 2ND ID>>
00033000  00024 1      IF READ(ID2,-2)=0 THEN GO ASK'FOR'ID1; <<INPUT 2ND ID>>
00034000  00032 1
00035000  00032 1   IF ID1=ID2
00036000  00033 1
00037000  00033 1      THEN BEGIN
00038000  00036 2           MOVE MSG:="ID'S MUST NOT BE EQUAL";
00039000  00056 2           PRINT(MSG,-21,0);
00040000  00062 2           GO ASK'FOR'ID2;
00041000  00063 2           END;
00042000  00063 1
00043000  00063 1   NF:=FOPEN(FILENAME,OLDFILE,UPDATE);        <<OLD FILE,BUFFERED,UPDATE>>
00044000  00073 1   CCL(1);                                    <<CHECK CONDITION CODE>>
00045000  00100 1
```

EXAMPLE - LISTING (2)

```
PAGE 0002   HEWLETT-PACKARD

00046000  00100 1   FIND'1ST'SET:                                  <<LOOK FOR EITHER SET>>
00047000  00100 1      FREADDIR(NF,SET1,32,RECNO:=RECNO+4D);       <<READ ID RECORD>>
00048000  00110 1      CCL(5);
00049000  00116 1      IF EOF                                      <<EARLY (UNEXPECTED) EOF>>
00050000  00117 1         THEN BEGIN
00051000  00117 2               EARLY'EOF:
00052000  00117 2               MOVE MSG:="IDS NOT ON FILE";
00053000  00134 2               PRINT(MSG,-16,0);
00054000  00140 2               GO ASK'FOR'ID1;
00055000  00141 2               END;
00056000  00141 1
00057000  00141 1      FREADSEEK(NF,RECNO+4D);                     <<GET NEXT SET>>
00058000  00146 1      IF SET1=ID1 THEN IDLEFT:=ID2
00059000  00152 1         ELSE IF SET1=ID2 THEN IDLEFT:=ID1
00060000  00163 1               ELSE GO FIND'1ST'SET;               <<DOESN'T MATCH EITHER>>
00061000  00167 1
00062000  00167 1      DO BEGIN                                    <<READ IN TEXT RECORDS>>
00063000  00167 2         FREAD(NF,SET1(I:=I+32),32); CCL(8);
00064000  00206 2         IF EOF THEN ERROR(9);                    <<MUST HAVE 3 TEXT RECS>>
00065000  00212 2         END
00066000  00212 1      UNTIL I=96;
00067000  00215 1      FIRSTSET:=RECNO;                            <<SAVE REC NO OF 1ST SET>>
00068000  00217 1   FIND'2ND'SET:                                  <<LOOK FOR REMAINING SET>>
00069000  00217 1      FREADDIR(NF,SET2,32,RECNO:=RECNO+4D);       <<READ NEXT ID>>
00070000  00227 1      CCL(15);
00071000  00235 1      IF EOF THEN GO EARLY'EOF;
00072000  00236 1      IF SET2<>IDLEFT THEN GO FIND'2ND'SET;       <<NOT A MATCH>>
00073000  00242 1      FREADSEEK(NF,FIRSTSET); CCL(17);            <<BRING 1ST SET BACK TO BUF>>
00074000  00252 1
00075000  00252 1      I:=0;                                       <<UPDATE LAST REC, READ NEXT>>
00076000  00254 1      DO BEGIN
00077000  00254 2
00078000  00254 2         FUPDATE(NF,SET1(I),32); CCL(20);         <<UPDATE LAST RECORD READ>>
00079000  00265 2         FREAD(NF,SET2(I:=I+32),32); CCL(21);<<READ NEXT RECORD>>
00080000  00304 2         IF EOF THEN ERROR(22);                   <<UNEXPECTED EOF>>
00081000  00310 2         END
00082000  00310 1      UNTIL I=96;
00083000  00313 1      FUPDATE(NF,SET1(I),32); CCL(23);            <<UPDATE LAST OF OLD 2ND SET>>
00084000  00324 1
00085000  00324 1      FWRITEDIR(NF,SET2,32,FIRSTSET);CCL(25);<<REPLACE 1ST SET WITH 2ND>>
00086000  00335 1      I:=0;
00087000  00337 1      DO BEGIN
00088000  00337 2         FWRITE(NF,SET2(I:=I+32),32,0); CCL(30);
00089000  00355 2         IF EOF THEN ERROR(32);                   <<UNEXPECTED EOF>>
00090000  00361 2         END
00091000  00361 1      UNTIL I=96;
00092000  00364 1   END.
 PRIMARY DB STORAGE=%016;    SECONDARY DB STORAGE=%00253
 NO. ERRORS=0000;            NO. WARNINGS=0000
 PROCESSOR TIME=0:00:03;     ELAPSED TIME=0:00:40
```

EXAMPLE RUN

FILE "POEMS" **BEFORE**

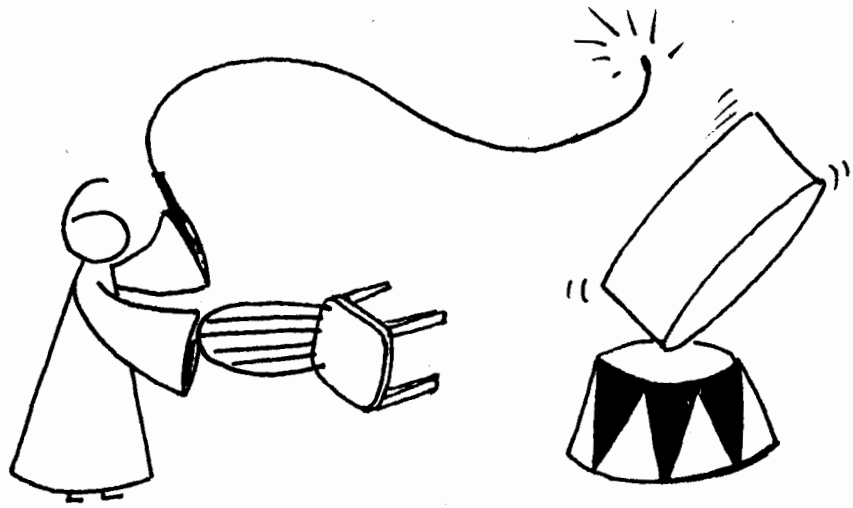
```
P3
Its fleece was white as snow;
And everywhere that Mary went,
Her lamb was sure to go.
P2
And pretty maids all in a row.

Mary had a little lamb,
P1
Mary, Mary, quite contrary,
How does your garden grow?
With silver bells and cockle shells
```

```
:RUN DATATRAN

ID1:=P3
ID2:=P1

 END OF PROGRAM
:
```

```
P1
Mary, Mary, quite contrary,
How does your garden grow?
With silver bells and cockle shells
P2
And pretty maids all in a row.

Mary had a little lamb,
P3
Its fleece was white as snow;
And everywhere that Mary went,
Her lamb was sure to go.
```

FILE "POEMS" **AFTER**

CONTROL FUNCTIONS



- CAN RECORDS BE SELECTED WITHOUT TRANSFERRING DATA?
- HOW CAN YOU MAKE CERTAIN A TRANSFER IS COMPLETE?
- WHAT CONTROL IS THERE OVER FILE LABELS?

RECORD SELECTION - NO TRANSFER

SPACE	MOVE POINTER BACKWARD/ FORWARD
POINT	RESET POINTER
REWIND	POINT TO RECORD 0

FSPACE / FPOINT

```
                IV          IV
FSPACE(filenum,displacement);
```

DISPLACEMENT IS NUMBER OF RECORDS TO SPACE FROM CURRENT POINTER [+ FORWARD , - BACKWARD]

```
                IV      DV
FPOINT(filenum,recnum);
```

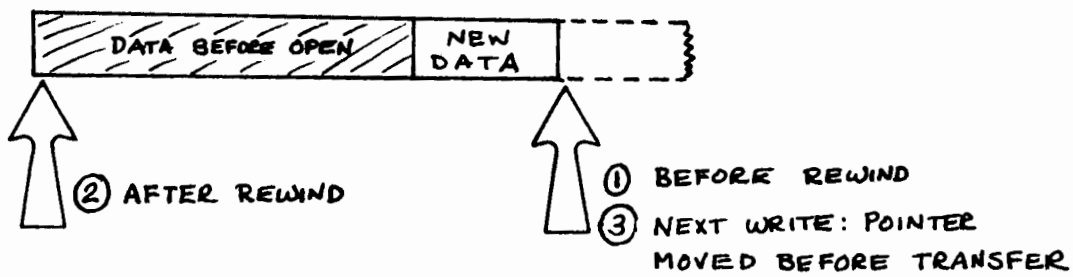
RECNUM IS RECORD NUMBER RELATIVE TO THE START OF THE FILE (RECORD 0) [+ ONLY]

- F, U RECORD TYPES ONLY
- BUF: LOGICAL / NOBUF: PHYSICAL
- NOT ALLOWED FOR APPEND ACCESS (CCL RETURNED)
- SPACE/POINT BEYOND EOF RESULTS IN CCG (POINTER NOT CHANGED)

FCONTROL (REWIND)

FCONTROL (filenum[IV], 5, dummy'param[L])

- RESETS POINTER TO RECORD Ø [LOGICAL:BUF/PHYSICAL:NOBUF]
- USE ON F, V, U FILES
- ANY ACCESS (R, W, R/W, W(S), U, A, X)
- SPECIAL MEANING FOR APPEND

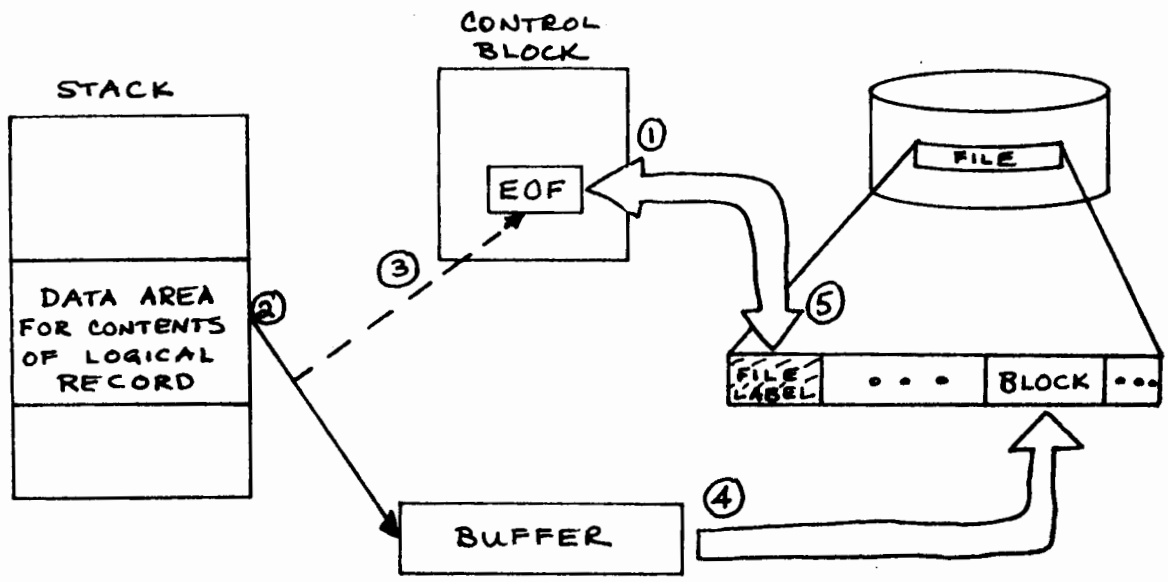


COMPLETION OF TRANSFERS

DATA ⇄ BUFFER ⇄ FILE

END OF FILE ⟷ ✱

TRANSFER OVERVIEW - OUTPUT



1. FOPEN READS FILE LABEL (EOF)
2. LOGICAL RECORD SENT TO BUFFER
3. LOGICAL EOF UPDATED IN CONTROL BLOCK (RECORD NUMBER > EOF)
4. PHYSICAL RECORD WRITTEN TO DISC (BUFFER POSTED)
5. EOF IN FILE LABEL WRITTEN AT FCLOSE

NOTE: EOF IS NOT A PHYSICAL MARK ON DISC FOLLOWING LAST RECORD

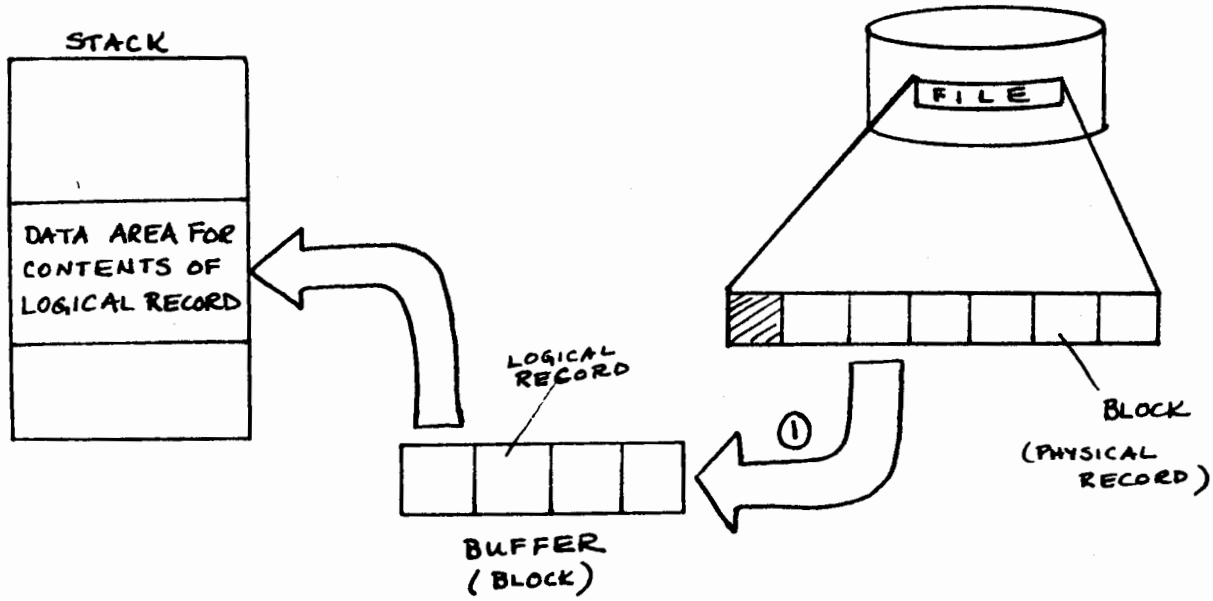
IF ABORTED OK

BUT IF SYS FAILURE OR REBOOT

IN MIDDLE COULD HAVE DIRTY

BUFFER AND CB EOF ≠ FILE LABEL EOF

TRANSFER OVERVIEW--INPUT



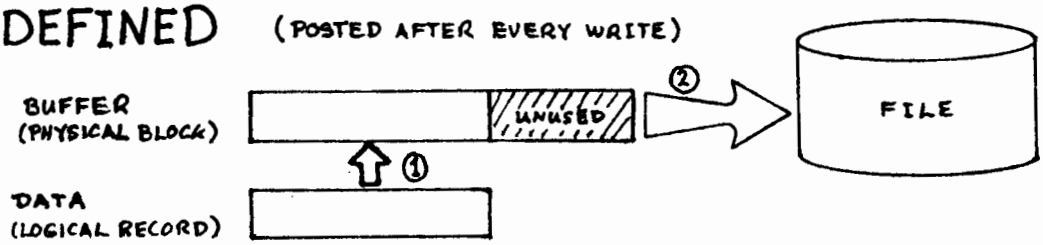
① INPUT PHYSICAL RECORD FROM FILE

② DEBLOCK AND RETURN LOGICAL RECORD

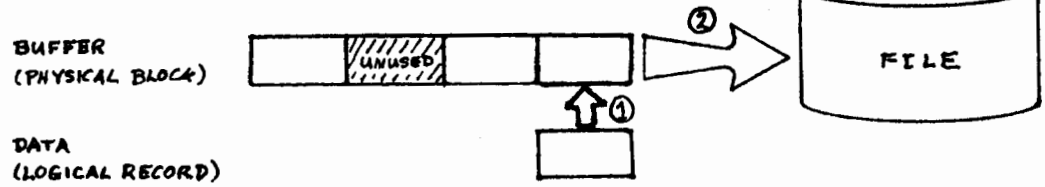
BUFFER MANAGEMENT - OUTPUT

FWRITE-FWRITEDIR-FUPDATE

UNDEFINED (POSTED AFTER EVERY WRITE)



FIXED (POSTED AFTER WRITE TO LAST RECORD IN BLOCK)



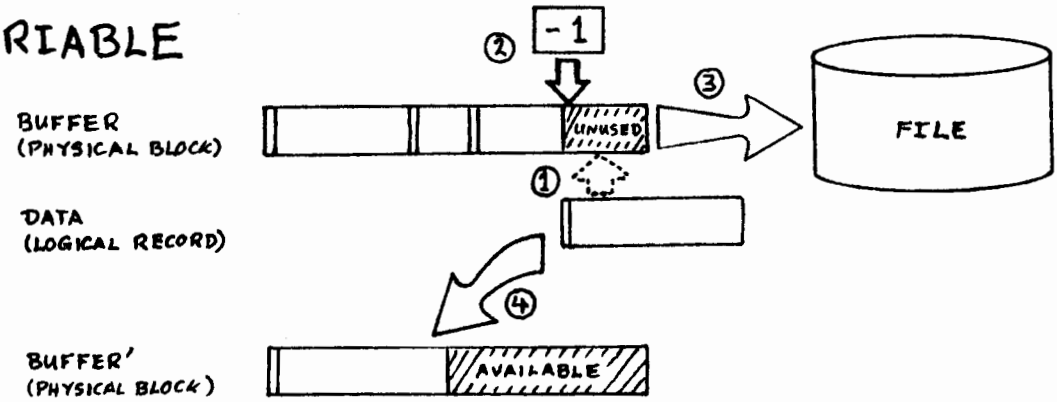
1) PUT LOGICAL RECORD INTO BUFFER

2) POST BUFFER TO DISC (NO WAIT)

BUFFER MANAGEMENT - OUTPUT

FWRITE

VARIABLE



① WILL RECORD FIT INTO THE CURRENT BLOCK?

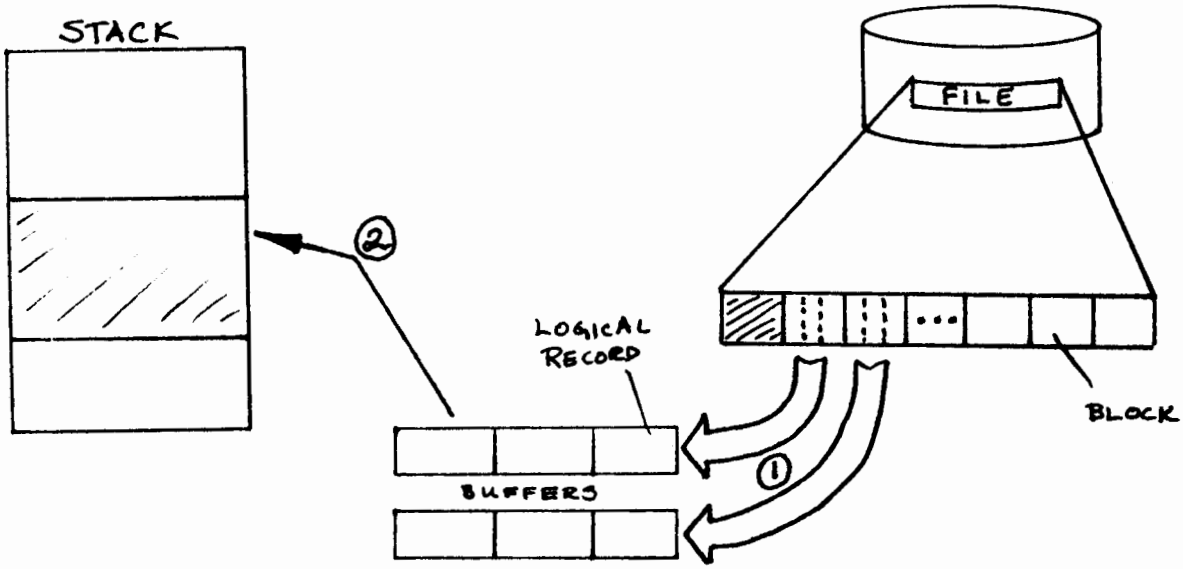
② NO: SET -1 IN BUFFER (END OF BLOCK DATA)

③ POST CURRENT BUFFER (BLOCK) TO DISC (NO WAIT)

④ WRITE LOGICAL RECORD INTO NEXT BUFFER (BECOMES CURRENT)

BUFFER MANAGEMENT - INPUT

FREAD (FIRST)

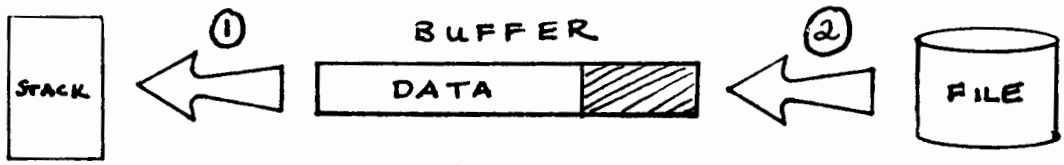


1. FIRST FREAD: FILL ALL EMPTY BUFFERS (NO WAIT)
2. RETURN 1ST LOGICAL RECORD

BUFFER MANAGEMENT - INPUT

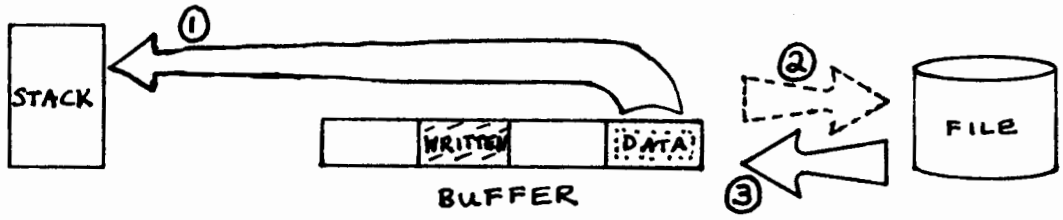
FREAD (SUBSEQUENT)

- UNDEFINED



1. LOGICAL RECORD SENT TO STACK
2. PRE-READ NEXT BLOCK (NO WAIT) INTO SAME BUFFER

- FIXED

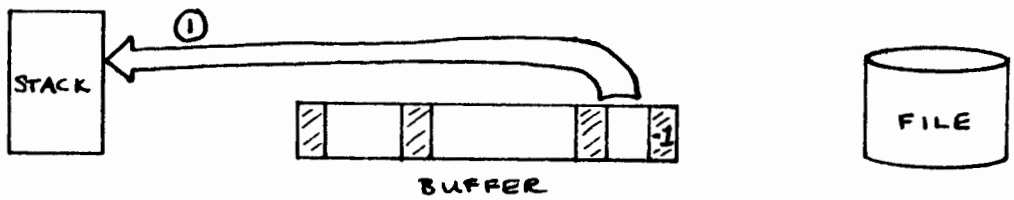


1. LAST LOGICAL RECORD OF BLOCK IS READ AND SENT TO STACK
2. IF ANY LOGICAL RECORD HAS BEEN WRITTEN TO BLOCK, POST BUFFER TO DISC
3. PRE-READ NEXT BLOCK IN FILE (NO WAIT) INTO SAME BUFFER

BUFFER MANAGEMENT - INPUT

FREAD (SUBSEQUENT)

- VARIABLE



1 LAST LOGICAL RECORD OF BLOCK SENT TO STACK

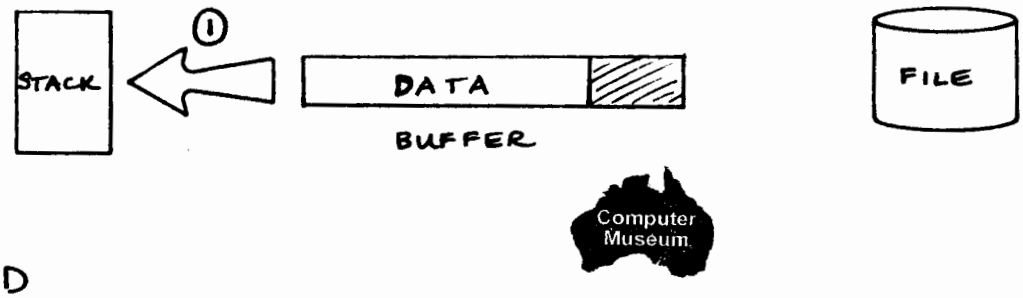
[NO PRE-READ IS DONE. NEXT BLOCK IS READ FROM DISC AT <u>NEXT</u> FREAD.]

HENCE VARIABLE SLOWER THAN F,V

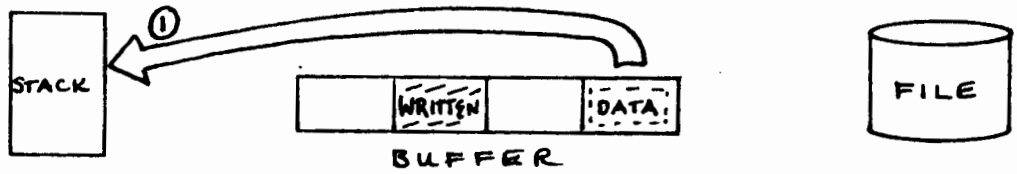
BUFFER MANAGEMENT – INPUT

FREADDIR

• UNDEFINED



• FIXED



1 LOGICAL RECORD SENT TO STACK
[NO PRE-READ IS DONE. BUFFER NOT POSTED AFTER FREADDIR.]

SUMMARY OF BUFFER POSTING/PRE-READ

	FWRITE (FWRITEDIR/FUPDATE)	FREAD	FREADDIR
UNDEFINED	POSTED AFTER EVERY OUTPUT	PRE-READ AFTER EVERY READ	NO ACTION
FIXED	POSTED AFTER OUTPUT OF LAST LOGICAL RECORD OF THE BLOCK	AFTER LAST LOGICAL RECORD IN BLOCK IS READ: 1) POST BUFFER IF CHANGED 2) PRE-READ	NO ACTION
VARIABLE	POSTED IF OUTPUT RECORD DOESN'T FIT INTO CURRENT BLOCK [FWRITE ONLY]	NO ACTION	

- ALL POSTING/PRE-READING IS NO WAIT
- CURRENT EOF ONLY MAINTAINED IN CONTROL BLOCK [FILE LABEL UPDATED AT FCLOSE]

BUFFER/EOF CONTROL - INTRINSICS

FSETMODE

[CRITICAL OUTPUT VERIFICATION]

FCONTROL

[COMPLETE I/O, WRITE EOF]

FSETMODE

FSETMODE (filenum^IV, 2)

CRITICAL OUTPUT VERIFICATION:

- FORCES WAIT WHEN A BUFFER IS POSTED
- DOES NOT EFFECT THE DECISION TO POST THE BUFFER
- SINGLE CALL SETS OR RESETS VERIFICATION MODE
- USEFUL ONLY FOR BUFFERED FILES

FCONTROL (COMPLETE I/O)

```
FCONTROL (filenum^IV, 2, dummy'param^L)
```

- FORCES POSTING OF ALL CHANGED BUFFERS
- WAITS FOR POSTING TO COMPLETE
- MARKS ALL BUFFERS AS EMPTY
- CALL AFTER EVERY OUTPUT (FWRITE, FWRITEDIR, FUPDATE) WHERE POSTING IS DESIRED
- USEFUL ONLY FOR BUFFERED FILES

FCONTROL (WRITE EOF)

FCONTROL (filenum[IV], 6, dummy'param[L])

- BUFFERED FILES ONLY
  - POST CHANGED BUFFERS; WAIT FOR COMPLETION
  - MARK BUFFERS EMPTY
- ALL FILES (BUF / NOBUF)
  - UPDATES USER LABEL COUNT IN FILE LABEL
  - UPDATES EOF IN FILE LABEL
- NOT ALLOWED FOR READ ONLY FILES

MEANS OF PROTECTING AGAINST SYS FAILURE WHEN OUTPUTTING TO CRITICAL FILES

ADDITIONAL INTRINSICS

FSPACE

FPOINT

FCONTROL (REWIND)

- POST BUFFERS IF CHANGED
- MARK ALL BUFFERS EMPTY

HEWLETT hp PACKARD

FILE LABEL CONTROL

FRENAME — FILE NAME (SYSTEM LABEL)

FREADLABEL
FWRITELABEL
} USER LABELS

FRENAME FAST WAY OF MOVING FILES BETWEEN GROUPS
AS IT ONLY AFFECTS SYSTEM DIRECTORY

FRENAME

```
                IV                BA
FRENAME(filenum,newfilereference);
```

- REQUIRES WRITE AND EXCLUSIVE ACCESS, SF CAPABILITY
- PERMANENT FILES
  - PURGES OLD NAME FROM SYSTEM DIRECTORY
  - INSERT NEW NAME INTO DIRECTORY
  - IF ERROR, OLD NAME RESTORED
- TEMPORARY FILES
  - ENTER NEW NAME INTO JTFD
  - DELETE OLD ENTRY
- UPDATE FILE LABEL (NAME, LAST MODIFICATION DATE, LOCKWORD)
- CAN CHANGE FILE GROUP NAME BUT NOT ACCOUNT NAME

FREADLABEL / FWRITELABEL

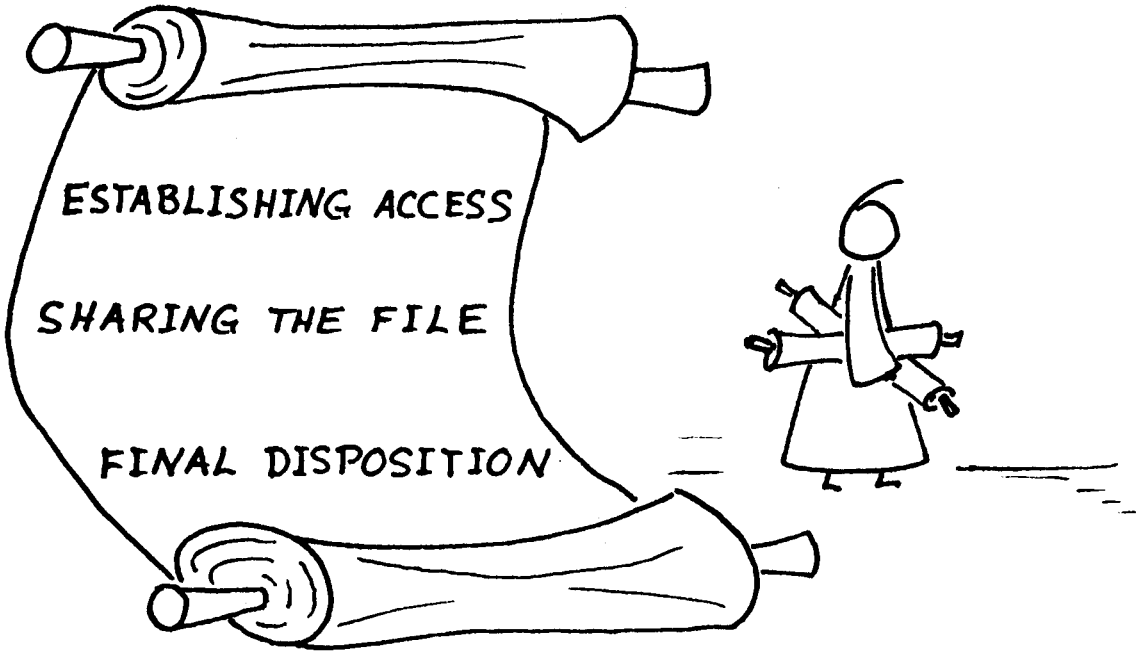
```
                    IV   LA    IV    IV   0-V
FREADLABEL(filenum,target,tcount,labelid);
```

```
                    IV   LA    IV    IV   0-V
FWRITELABEL(filenum,target,tcount,labelid);
```

- CAN READ OR WRITE REGARDLESS OF ACCESS
- RECORD POINTER IS NOT EFFECTED
- WAITS FOR COMPLETION OF LABEL TRANSFER

HEWLETT PACKARD

SHARED FILE CONSIDERATIONS



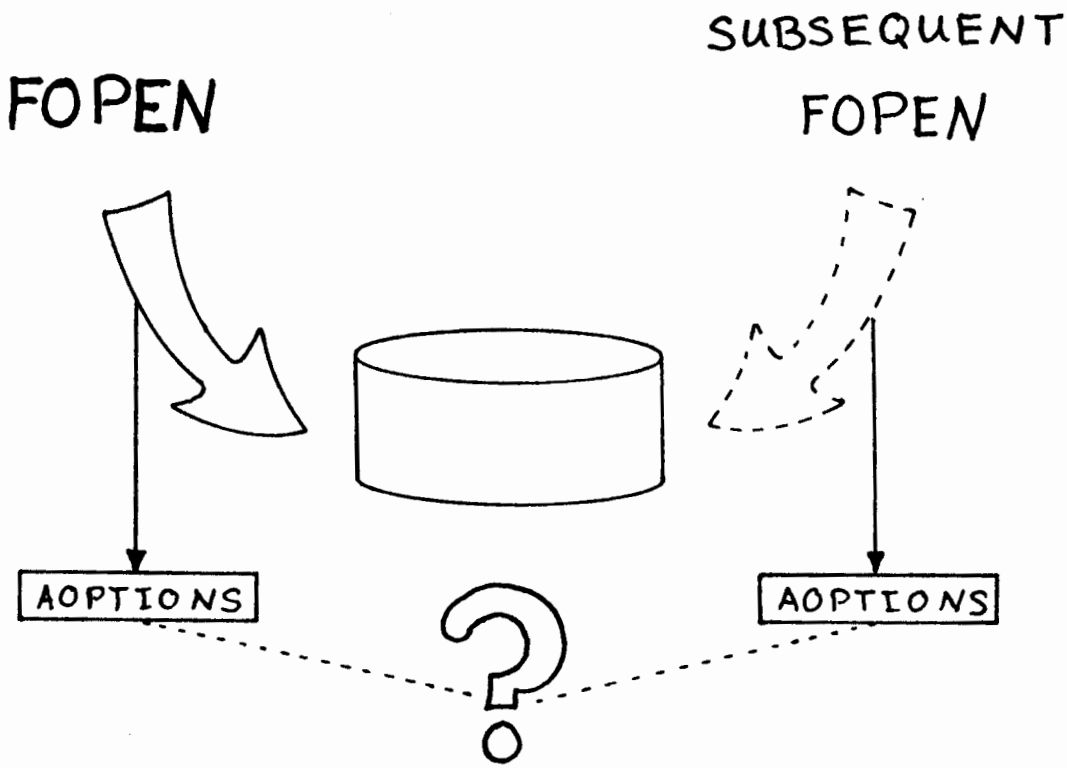
ESTABLISHING ACCESS

FOPEN [FILE NOT ALREADY OPEN]

SETS → ALLOWABLE ACCESS (AOPTIONS)

DETERMINES WHETHER ACCESS TO THIS **OPEN** FILE IS GRANTED TO SUBSEQUENT FOPEN REQUESTS

SUBSEQUENT FOPENS



AOPTIONS MUST BE COMPATIBLE BEFORE ACCESS IS GRANTED TO NEXT FOPEN

COMPATIBLE AOPTIONS - EXC

FIRST FOPEN: **EXC** [EXCLUSIVE ACCESS]

ALL SUBSEQUENT FOPENS <u>FAIL</u>

COMPATIBLE AOPTIONS – SHR

FIRST FOPEN: **SHR**

MODES ALLOWED IN SUBSEQUENT FOPENS

	SHR	EAR	EXC
INPUT	I, O, I/O	I, O, I/O	NONE
OUTPUT	I, O, I/O	NONE	NONE
INPUT/OUTPUT	I, O, I/O	NONE	NONE

- MODES: I (INPUT: READ ONLY)
  O (OUTPUT: WRITE, WRITE (SAVE), APPEND)
  I/O (INPUT/OUTPUT: READ/WRITE, UPDATE, EXECUTE)
- EAR REQUIRES ALL OTHER ACCESS TO BE INPUT ONLY

COMPATIBLE OPTIONS — EAR

MODES ALLOWED IN SUBSEQUENT FOPENS

FIRST FOPEN: EAR

	SHR	EAR	EXC
INPUT	I, I/O→I	I, I/O→I	NONE
OUTPUT	I, I/O→I	NONE	NONE
INPUT/OUTPUT	I, I/O→I	NONE	NONE

(I/O→I : INPUT/OUTPUT RESET TO INPUT BY FOPEN)

COMPATIBLE AOPTIONS - LOCKING

FIRST **FOPEN** : DYNAMIC LOCKING

ALL SUBSEQUENT FOPENS MUST REQUEST DYNAMIC LOCKING

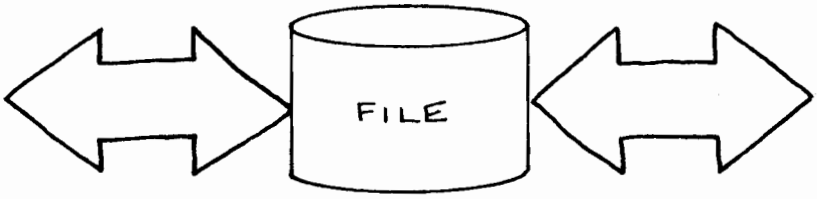
FIRST **FOPEN** : NO DYNAMIC LOCKING

NO SUBSEQUENT FOPENS MAY REQUEST DYNAMIC LOCKING

- IF CONDITIONS NOT MET, FOPEN FAILS (CCL)

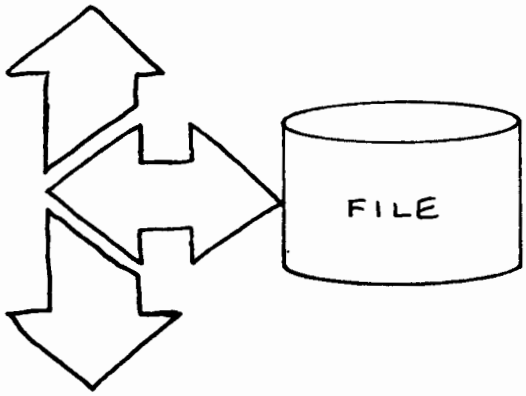
HEWLETT hp PACKARD

ESTABLISHING ACCESS – PATH SELECTION



SHARE DATA

(NO MULTI)



SHARE DATA, POINTERS, BUFFERS

(MULTI)

- EVERY FOPEN DETERMINES WHETHER THE ACCESS PATH MAY BE SHARED

IN MULTI ONE CLOSES NOT CLOSED FOR OTHERS

PATH SELECTION – AOPTIONS

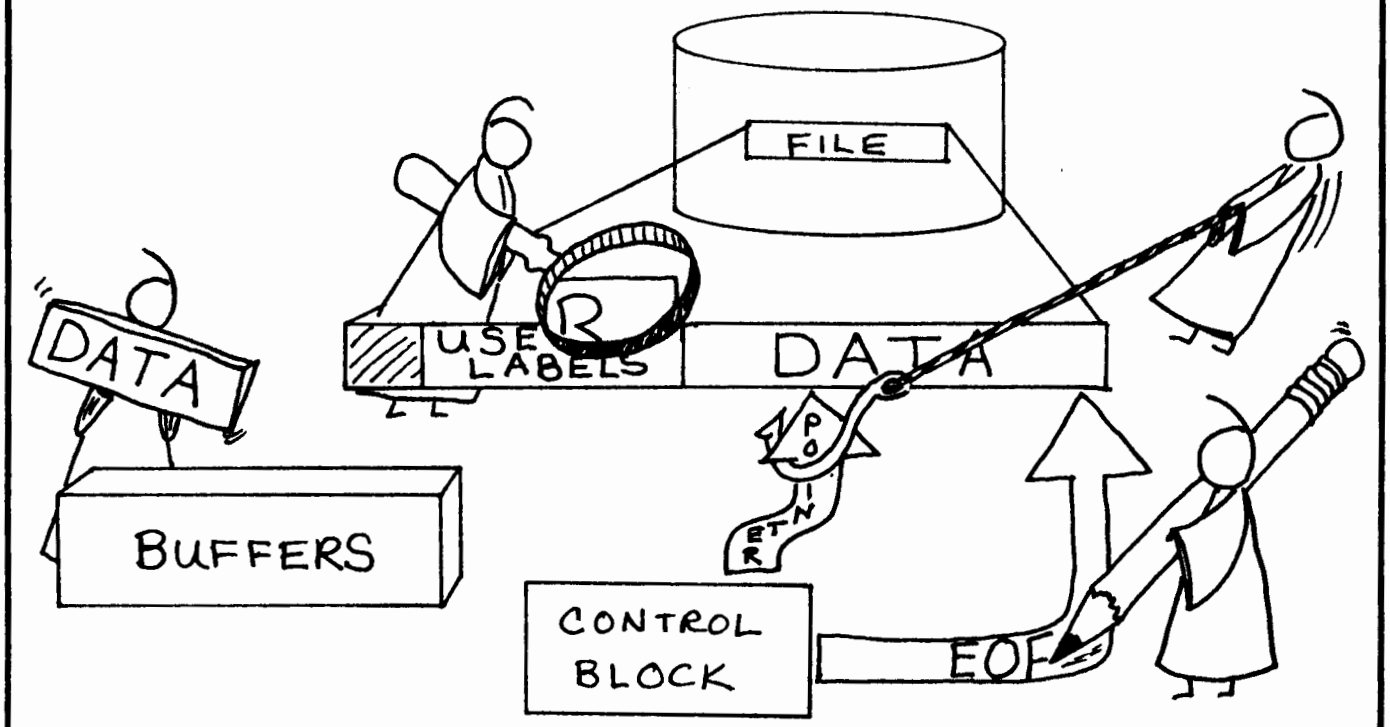
EVERY FOPEN:

CALLER REQUESTS **MULTI**

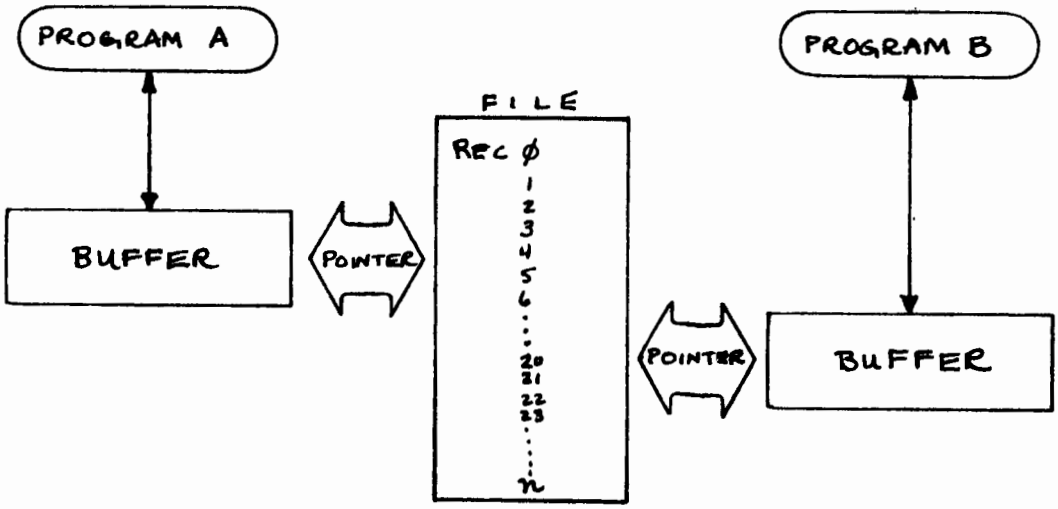
	ACCESS GRANTED BY FOPEN		
	SHR	EAR	EXC
INPUT	MULTI	MULTI	NO MULTI
OUTPUT	MULTI	NO MULTI	NO MULTI
INPUT/OUTPUT	MULTI	MULTI	NO MULTI

- SHARED BUFFERS/POINTERS ("MULTI") LIMITED TO RELATED PROCESSES [IMPLIES PROCESS HANDLING]
- INDEPENDENT BUFFERS/POINTERS ("NO MULTI") ARE DEFAULT

SHARING THE FILE

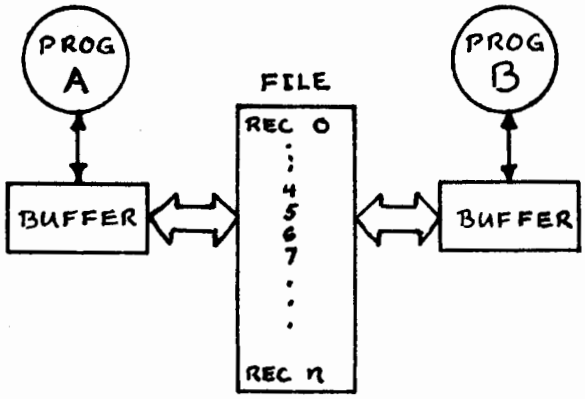


SHARING DATA IN A FILE (NO MULTI)

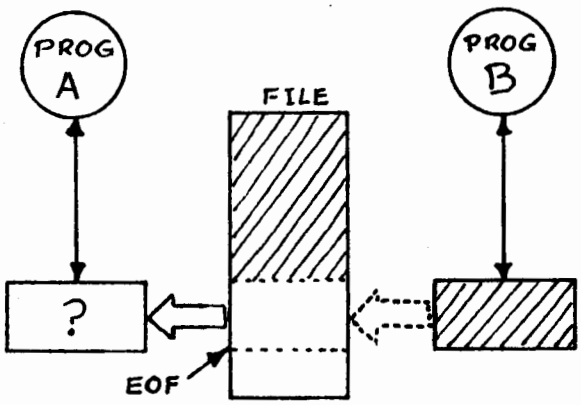


- TWO (OR MORE) PROGRAMS CAN SHARE THE DATA IN A FILE
- BUFFERS AND POINTERS ARE INDEPENDENT
- EITHER ACCESS CAN BE NOBUF

BUFFER CONTENTION



- POINTERS MAY COINCIDE
- DIFFERENT BUFFERS CONTAIN THE SAME BLOCK
- IF ONE BUFFER CHANGED, OTHER RETAINS OLD DATA
- IF BOTH BUFFERS CHANGED, LAST POSTED OVERLAYS FIRST

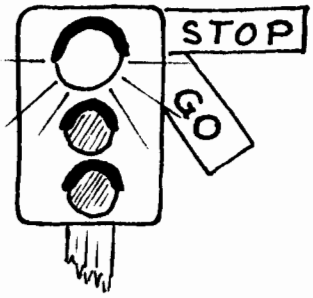


- DATA WRITTEN TO BUFFER BY PROG B
- EOF UPDATED (CONTROL BLOCK)
- BUFFERS NOT YET POSTED DATA NOT ON FILE
- PROG ATTEMPTS TO READ TO EOF - RESULT GARBAGE DATA

AVOIDING BUFFER CONTENTION

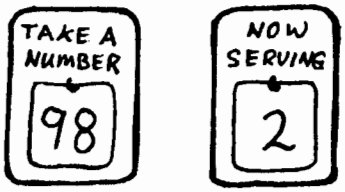
BUFFER MAINTENANCE

EMPTY THE BUFFERS
SAVE CHANGED DATA



BUSY FLAG

SIGNAL TEMPORARY
EXCLUSIVE ACCESS
TO THE FILE



WAITING QUEUE

SUSPEND (OPTIONAL)
UNTIL FILE IS
AVAILABLE

REQUIREMENTS

FLOCK

IV LV
FLOCK(*filenum*,*lockcond*);

- FOPEN WITH DYNAMIC LOCKING AOPTION
- USE ON ANY SHARED FILE
  [F, V, U ; BUF/NOBUF]
- "LOCK" CONDITIONALLY (I.E., ONLY IF AVAILABLE) OR
  LOCK UNCONDITIONALLY (I.E., QUEUE UNTIL AVAILABLE)
- **DOES NOT PREVENT SIMULTANEOUS USE**
- SUCCESSFUL "LOCK" OF BUFFERED FILES
  - POSTS CHANGED BUFFERS
  - MARKS ALL BUFFERS EMPTY

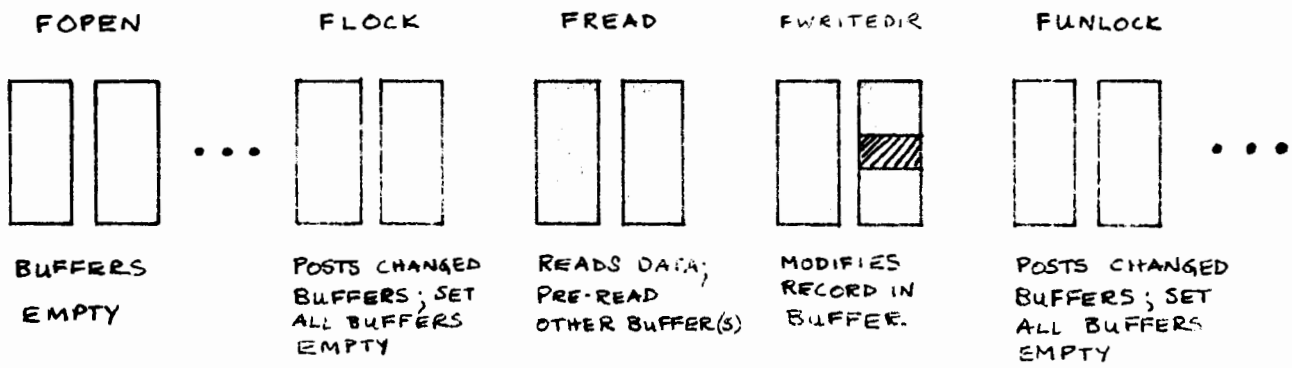
FUNLOCK

IV
FUNLOCK(*filename*);

- POSTS DATA BUFFERS IF CHANGED
- MARKS **ALL** BUFFERS EMPTY
- DOES NOT UPDATE EOF IN FILE LABEL
- NEXT PROGRAM QUEUED 'LOCKS' THE FILE
- FILE 'UNLOCKED' AUTOMATICALLY AT FCLOSE

HEWLETT hp PACKARD

SOLVING BUFFER CONTENTION



FLOCK : STARTS WITH EMPTY BUFF

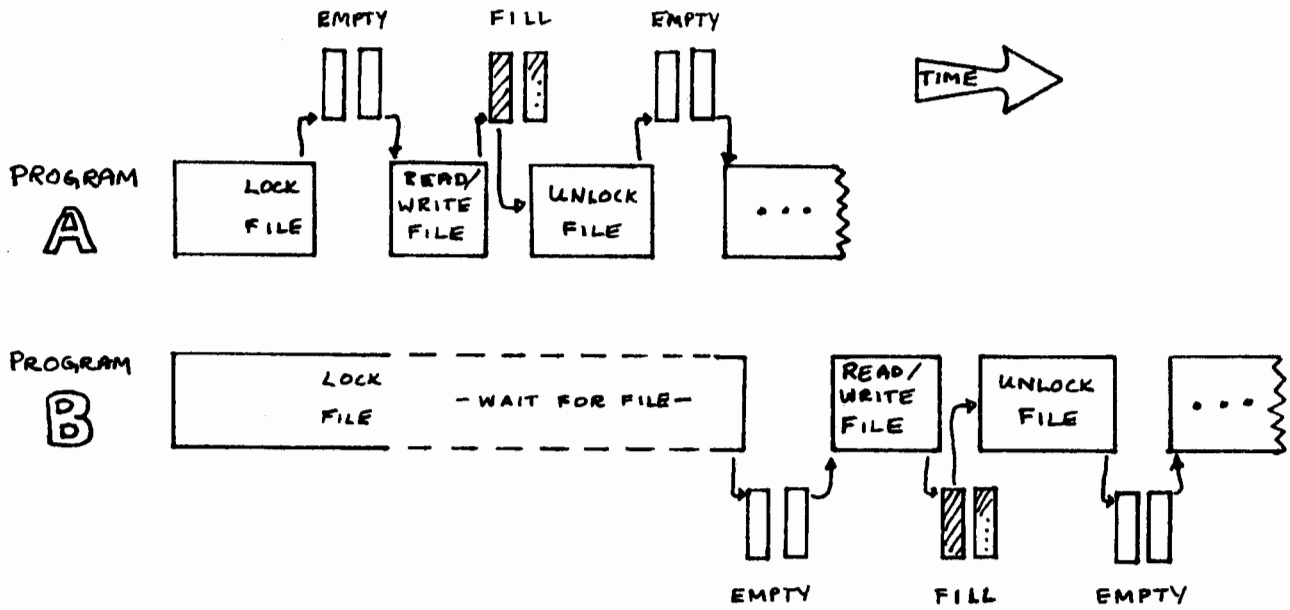
FUNLOCK: ENDS WITH EMPTY BUFFERS

TRANSFERS SURROUNDED BY FLOCK & FUNLOCK ARE PROTECTED ONLY IF ALL OTHER USERS SURROUND THEIR TRANSFERS IN THE SAME WAY.

[NOBUF FILES: USER SHOULD MANAGE 'BUFFER' SIMILARLY]

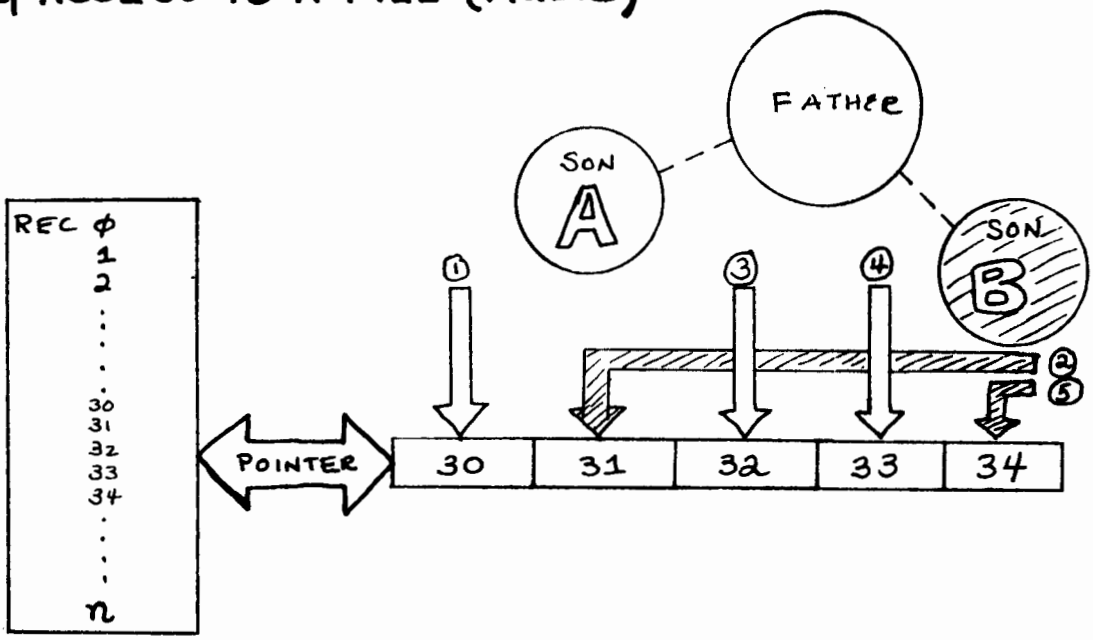
FLOCK ALLOCATES A RIN

LOCKING -- NO MULTI FILES



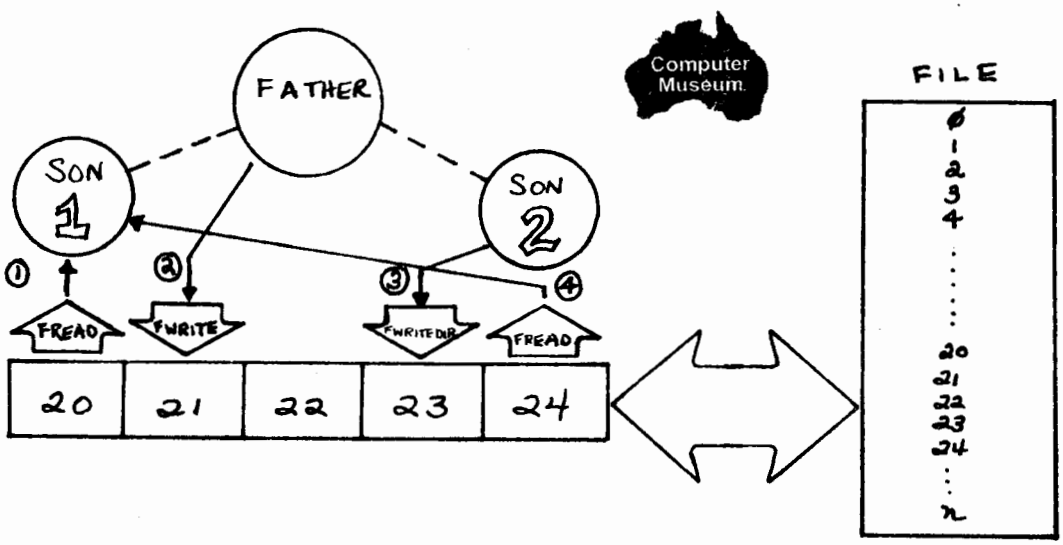
BY USING FLOCK/FUNLOCK, COOPERATING PROGRAMS CAN INSURE THAT DATA IS CONTAINED IN ONLY ONE SET OF BUFFERS AT ANY GIVEN TIME.

SHARING ACCESS TO A FILE (MULTI)



- USED IN SPOOLING OR LOGGING APPLICATIONS
- TWO (OR MORE) PROCESSES (SAME PROCESS TREE) CAN SHARE ACCESS TO A FILE
- FILE MUST BE BUFFERED
- SHARED BUFFERS : NO CONTENTION (DUPLICATE DATA) PROBLEMS
- SHARED POINTER : PRESERVE TEMPORAL SEQUENCE (READ ONLY / WRITE ONLY)

POINTER CONSIDERATIONS - MULTI



- OPERATIONS WHICH CHANGE THE POINTER CHANGE IT FOR **ALL** ACCESSORS
- PROBLEMS CAN RESULT IF ONE ACCESSOR SETS THE POINTER TO A LOCATION UNEXPECTED BY SUBSEQUENT ACCESSORS

GOOD REASON FOR GETTING FATHER PROCESS DO ALL FILE HANDLING

ESTABLISHING POINTER CONVENTIONS

INTRINSIC CALLS:

SEQUENTIAL (INPUT ONLY)
SEQUENTIAL (OUTPUT ONLY)
} POINTER MOVES IN TEMPORAL SEQUENCE

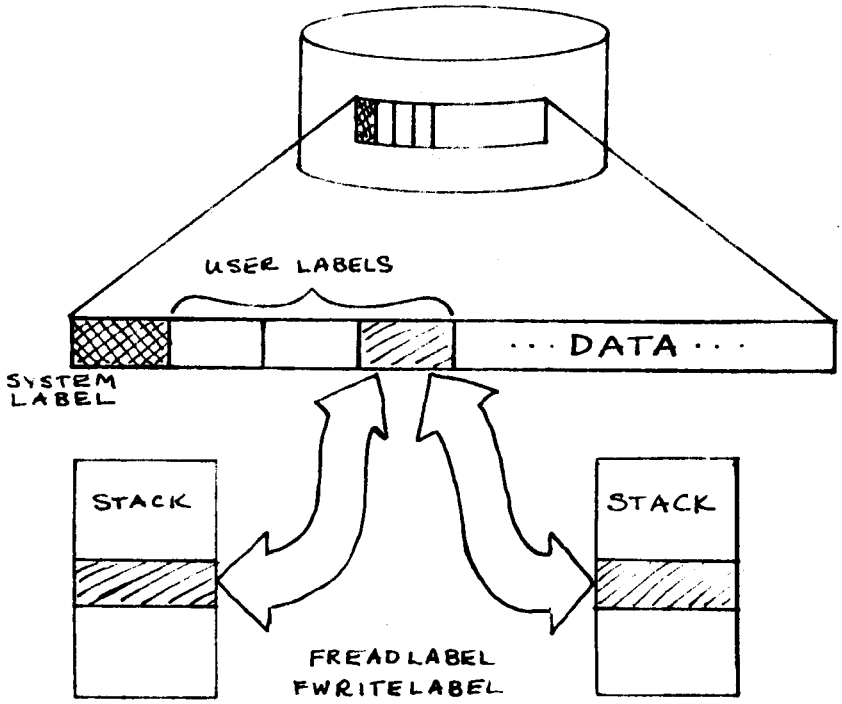
DIRECT ACCESS ONLY (INPUT / OUTPUT)
} POINTER DETERMINED BY EACH ACCESS

SIGNAL SYSTEM:

MPE 'RIN'
- BUSY FLAG & WAITING QUEUE
- DOES NO BUFFER MANAGEMENT

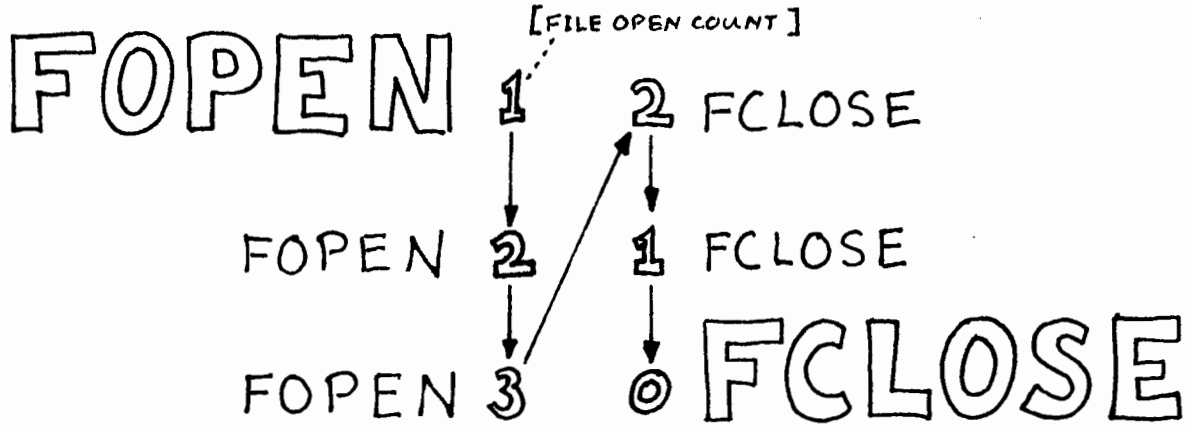
} ALLOWS TEMPORARY EXCLUSIVE ACCESS AND CONTROL OF THE POINTER

SHARING USER LABELS



- TWO PROGRAMS ATTEMPT TO CHANGE A USER LABEL AT THE SAME TIME
- THE LAST FWRITELABEL OVERLAYS THE FIRST
- REQUIRES LOCKING MECHANISM WITHOUT BUFFER MANAGEMENT: RIN

FINAL DISPOSITION

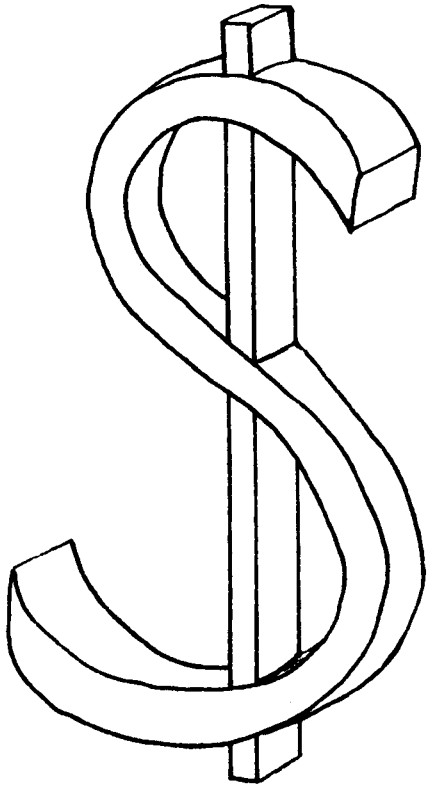


COUNT OF CURRENT OPENS { INCREMENTED BY FOPEN / DECREMENTED BY FCLOSE }

- FCLOSE : COUNT <> 0
  - SAVE SMALLEST NON-ZERO DISPOSITION
  - TERMINATE CALLER'S ACCESS
  - LEAVE FILE DISPOSITION UNCHANGED

- FCLOSE : COUNT = 0
  - TERMINATE CALLER'S ACCESS
  - USE SAVED DISPOSITION (SMALLEST NUMBER) TO CLOSE THE FILE

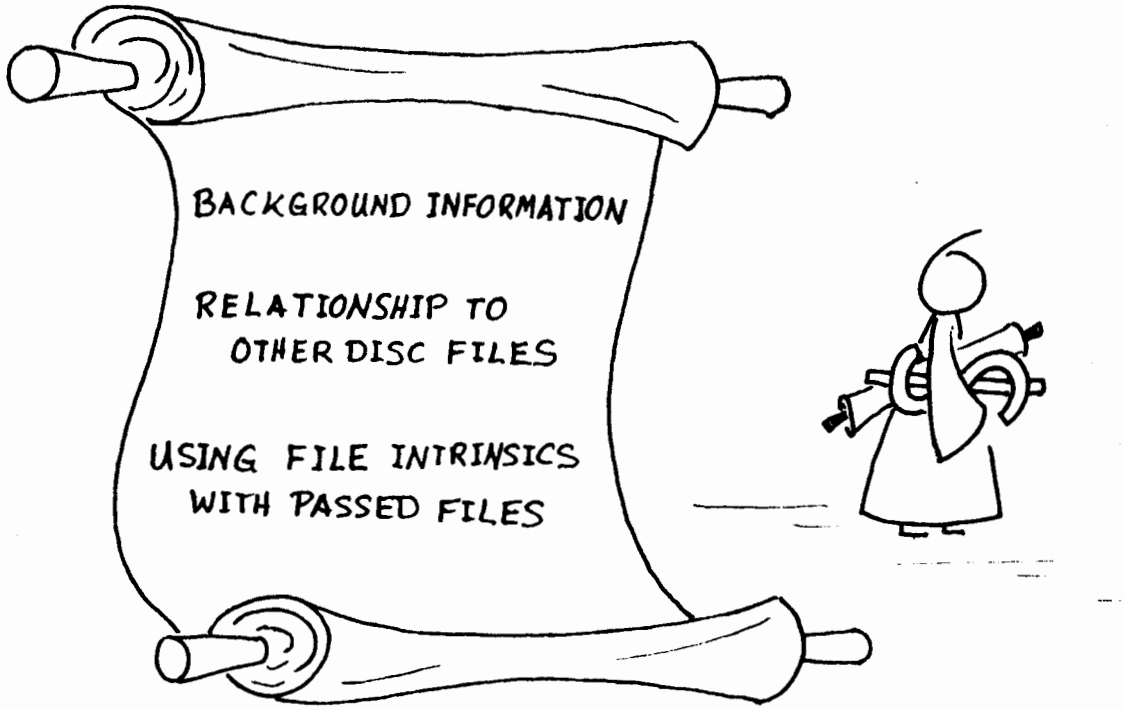
PASSED FILES



$PASSED FILES

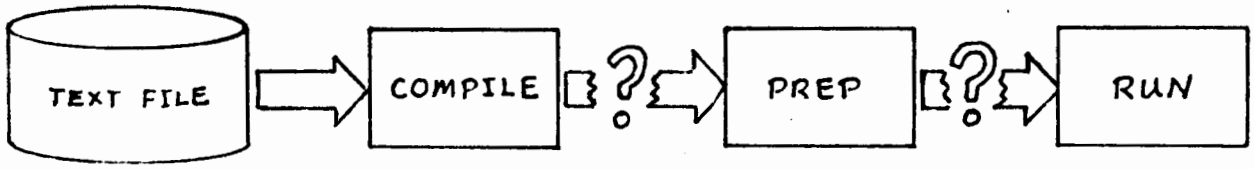
[$NEWPASS , $OLDPASS]

PASSED FILE CHARACTERISTICS



BACKGROUND ON PASSED FILES

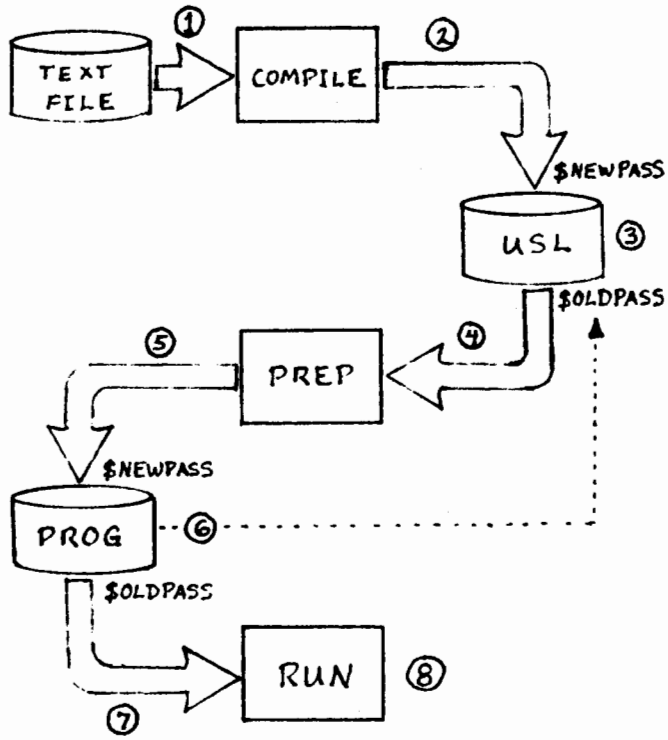
:SPLGO textfile



- "PASSED" FILES FACILITATE MULTIPLE JOB STEP COMMANDS
- DEFAULT FILES : PASS INFORMATION BETWEEN STEPS
- SPECIAL FILE HANDLING
  - NO DIRECTORY SEARCH REQUIRED

SAMPLE USAGE - MPE SUBSYSTEMS

:SPLGO text file



1) TEXT READ; COMPILED

2) OBJECT WRITTEN TO $NEWPASS

3) $NEWPASS CLOSED: NAME CHANGED TO $OLDPASS

4) OBJECT PREP'D FROM $OLDPASS

5) PROGRAM WRITTEN TO NEW $NEWPASS

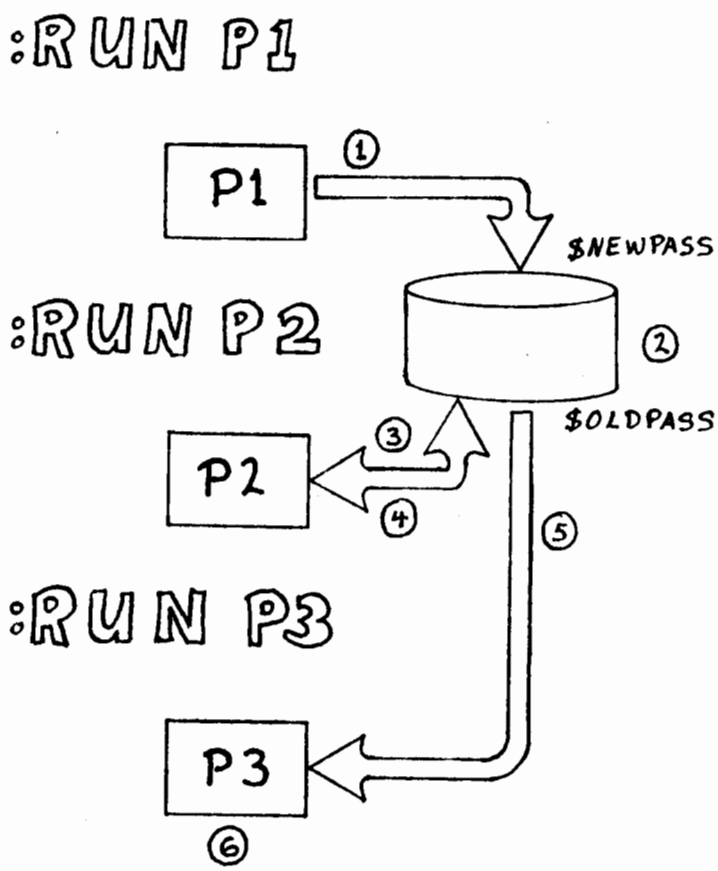
6) $NEWPASS CLOSED: NAME CHANGED TO $OLDPASS; OLD $OLDPASS DELETED

7) RUN $OLDPASS

8) $OLDPASS (PROGRAM FILE) NOT CHANGED OR DELETED AFTER RUN

SAMPLE USAGE- USER PROGRAMS

HEWLETT hp PACKARD



1) USER PROGRAM P1 WRITES TO $NEWPASS

2) $NEWPASS CLOSED : NAME CHANGED TO $OLDPASS

3) PROGRAM P2 READS FROM $OLDPASS AND WRITES TO $OLDPASS

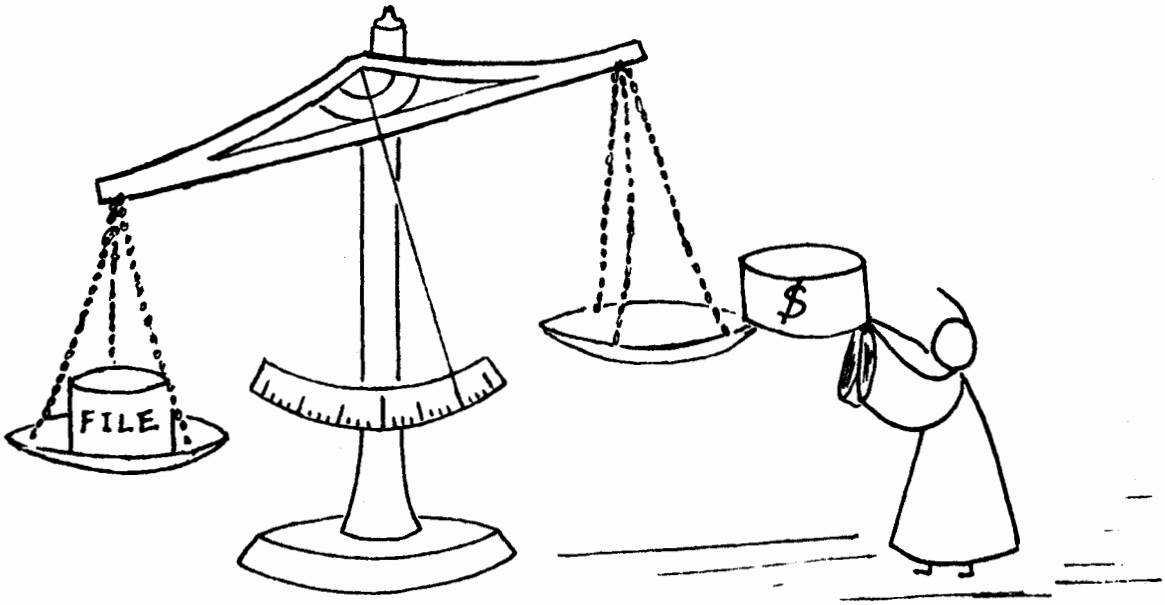
4) $OLDPASS CLOSED; REMAINS $OLDPASS

5) PROGRAM P3 READS FROM $OLDPASS

6) $OLDPASS WILL REMAIN UNTIL REPLACED, DELETED, OR SAVED (RENAMED)

IF DON'T USE $CONTROLINIT
SUBSEQUENT COMPILES APPEND TO $OLDPASS
SO CAN COMBINE PROGRAMS IN ONE PROGRAM FILE

RELATIONSHIP TO OTHER DISC FILES



HOW DO PASSED FILES COMPARE WITH REGULAR FILES?

HOW DO THE INTRINSIC CALLS DIFFER?

NEW FILE/$NEWPASS COMPARISON

NEW	$NEWPASS
• DISC SPACE ALLOCATED	• DISC SPACE ALLOCATED
• DISC ADDRESS PUT INTO CONTROL BLOCK	• DISC ADDRESS PUT INTO CONTROL BLOCK
• DEFAULT CLOSE DISPOSITION - DEALLOCATE SPACE - DELETE CONTROL BLOCK	• DEFAULT CLOSE DISPOSITION - RENAME TO $OLDPASS - SAVE DISC ADDRESS IN JOB/SESSION TABLE (JIT) - DELETE CONTROL BLOCK
• DISC ADDRESS NOT SAVED (E.G. NOT IN ANY DIRECTORY)	• DISC ADDRESS SAVED FOR FUTURE USE IN THE JOB/SESSION

OLD FILE/$OLDPASS COMPARISON

OLD

- DIRECTORY (JOB TEMP. OR SYSTEM) SEARCHED FOR DISC ADDRESS
- DISC ADDRESS PUT INTO CONTROL BLOCK
- DEFAULT CLOSE DISPOSITION DELETE CONTROL BLOCK
- DISC ADDRESS STILL IN DIRECTORY FOR FUTURE USE

$OLDPASS

- DISC ADDRESS OBTAINED FROM JOB INFORMATION TABLE (JIT)
- DISC ADDRESS PUT INTO CONTROL BLOCK
- DEFAULT CLOSE DISPOSITION DELETE CONTROL BLOCK
- DISC ADDRESS STILL IN JIT FOR FUTURE USE IN JOB/SESSION

USING FOPEN/FCLOSE – $NEWPASS

FOPEN

<< FORMAL DESIGNATOR IS $NEWPASS
AND/OR
FOPTIONS.(10:3) = 2 >>

- SETS : DOMAIN TO NEW
  DEVICE CLASS TO "DISC"
  NAME TO "$NEWPASS"
  DEFAULT RECORD SIZE FOR DEVICE

FCLOSE

<< ANY DISPOSITION >>

- SETS: DISC ADDRESS IN JIT
  NAME TO "$OLDPASS"
- DELETES PREVIOUS $OLDPASS

USING FOPEN/FCLOSE – $OLDPASS

FOPEN « OPTIONAL FORMAL DESIGNATOR, FOPTIONS.(10:3)=3 »

- SETS DOMAIN TO NEW (NO DIRECTORY SEARCH)
- CCL RETURNED IF NO $OLDPASS EXISTS

FCLOSE « DISPOSITION »

- DEFAULT DISPOSITION : NO CHANGE
  [ADDRESS LEFT IN JIT]
- SAVE PERMANENT/TEMPORARY DISPOSITION
  FORMAL DESIGNATOR VALID: MAKE DIRECTORY ENTRY
  ZERO ADDRESS IN JIT
  FORMAL DESIGNATOR INVALID: CCL RETURNED
- DELETE DISPOSITION : RELEASE FILE SPACE
  ZERO ADDRESS IN JIT

FASTER BECAUSE NO DIRECTORY SEARCH

OTHER FILE INTRINSIC USAGE

STATUS-
-TRANSFER
CONTROL-

EXCEPT FOR FRENAME,

ALL OTHER FILE INTRINSICS APPLY TO
$NEWPASS/$OLDPASS AS DISCUSSED PREVIOUSLY

USING FRENAME

FRENAME

$NEWPASS

- CHANGES : NAME
  FOPTIONS.(10:3)=0
- NO LONGER "PASSED FILE" (SAME AS FOPEN NEW WITH CURRENT NAME)
- CAN BE SAVED (PERM/TEMP) OR DELETED

$OLDPASS

- CHANGES: NAME
  DOMAIN TO OLDTEMP
  ADDRESS IN JIT → Ø
- NO LONGER A "PASSED FILE" (SAME AS AN OLD TEMP FILE)
- CAN BE SAVED (PERM/TEMP) OR DELETED

FILE SYSTEM LAB #3

WRITE AN SPL PROGRAM TO:

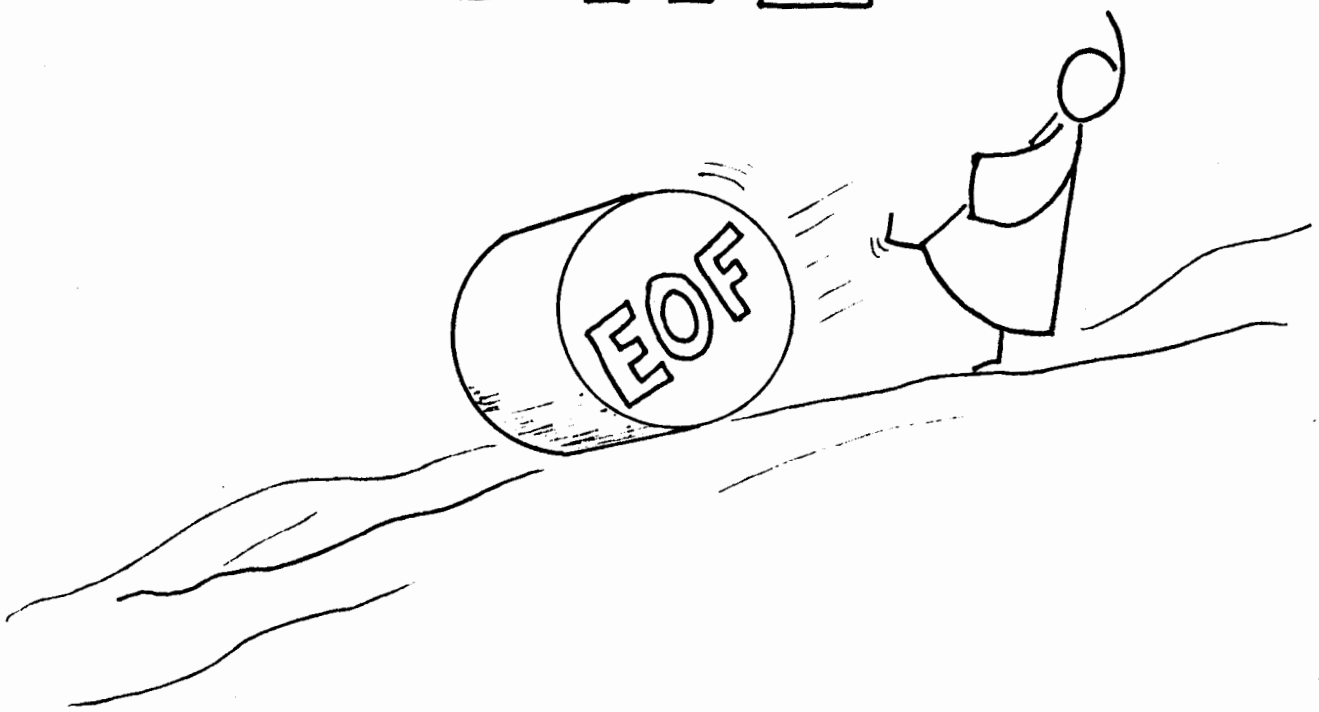
1. OPEN FILE "LAB1F" (CLOSED PERMANENT IN FILE SYSTEM LAB #2) WITH READ/WRITE ACCESS.
2. SET THE RECORD POINTER TO RECORD 12.
3. CALL PRINT'FILE'INFO TO VERIFY.
4. SPACE BACK 7 RECORDS.
5. CALL PRINT'FILE'INFO TO VERIFY.
6. CLOSE THE FILE (DEFAULT DISPOSITION).

FILE SYSTEM LAB #1 : SOLUTION

```
PAGE 0001   HEWLETT-PACKARD 32100A.06.3  SPL[4W]  THU, FEB  3, 1977


00001000  00000 0   $CONTROL USLINIT
00002000  00000 0   BEGIN
00003000  00000 1   BYTE ARRAY NAME(0:5):="LAB1F;";
00004000  00004 1   LOGICAL FOPTIONS:=4,  <<ASCII>>
00005000  00004 1           AOPTIONS:=4;  <<R/W>>
00006000  00004 1
00007000  00004 1   INTEGER FILE;
00008000  00004 1
00009000  00004 1   INTRINSIC FOPEN,FCLOSE,PRINT'FILE'INFO;
00010000  00004 1
00011000  00004 1   FILE:=FOPEN(NAME,FOPTIONS,AOPTIONS,
00012000  00004 1                <<RECSIZE>>   40,,,
00013000  00005 1                <<USER LBLS>> 2,
00014000  00007 1                <<BLK FCTR>>  16,,
00015000  00010 1                <<FILE SIZE>> 16D,,,
00016000  00016 1                <<FILE CODE>> 200);
00017000  00023 1
00018000  00023 1   PRINT'FILE'INFO(FILE);
00019000  00025 1
00020000  00025 1   FCLOSE(FILE,0,0);
00021000  00030 1
00022000  00030 1   PRINT'FILE'INFO(FILE);
00023000  00032 1
00024000  00032 1   END.
 PRIMARY DB STORAGE=%004;   SECONDARY DB STORAGE=%00003
 NO. ERRORS=0000;           NO. WARNINGS=0000
 PROCESSOR TIME=0:00:01;    ELAPSED TIME=0:00:18
```

HEWLETT hp PACKARD

THE



Computer Museum

213