CONFIDENTIAL

PASCAL DEVELOPMENT SYSTEM INTERNAL DOCUMENTATION

Bill Schottstaedt
Ext 2379
26-Jan-82

CONTENTS:

ICODE DEFINITION

The first pass of the compiler generates a .I file. Its contents are:

```
    00
```

Variable references:

```
    01 +offset                  Global variable reference
    02 +offset                  Local variable reference
    03 lev +offset              Intermediate level variable reference
    04 com +offset              Common variable reference
    05 reg size expr            Register reference
    05 reg 0
    06 ???????                  String temp
    07 ???????                  Set temp
 08-0B nnnnnnn                  1/2/4/8 byte temp
```

Addressing operators:

```
    0C addr...                  '^' - Dereference operator
    0D addr...                  '^' - File dereference operator
    0E addr...                  '^' - Text file dereference operator
    0F +offset addr...          '.' - Record field offset
 10-13 nnnnnnn addr... expr...  '[]' - 1/2/4/8 byte array index
    14 nnnnnnn nnnnnnn addr...
          expr...               '[]' - Long array index
    15 +nn nnnnnnn addr...
          expr...               '[]' - Packed array access
    16 addr...                  '@' - Address of operator
```

Constants:

```
    17                          nil
 18-1B #######                  1/2/4/8 byte constant
    1C nnn 'ABC...'             String constant
    1D nnn 'ABC...'             PAOC Constant
    1E nnn [1,5..7,21]          Set constant
    1F                          [] - Null set
```

Assignment operators:

```
 20-23 flippable addr... expr...':=' - 1/2/4/8 byte assignment

    (* flippable is true if the assignment left hand side can be computed
       after the right hand side.  In this case, we have expr...addr... *)

    24 nnnnnnn addr... expr...  ':=' - Multiple byte assignment
    25 nnn addr... expr...      ':=' - Set assignment
    26 (15/3E/3F ... ) expr...  ':=' - Packed assignment
    27 nnn addr... expr...      ':=' - String assignment
    28 nnn nnn addr... expr...  ':=' - PAOC Assignment
    29 nnn addr... expr...      ':=+' - Add to
    2A nnn addr... expr...      ':=-' - Subtract from
```

```
   2B nnn                       WITH field reference, level nnn
   2C lev isptr addr...         Begin WITH statement, level nnn
   2D lev                       End WITH statement, level nnn
   2E lo----- hi----- expr...   2 Byte Range Check
   2F hi- expr...               String Range Check - assignment, not index
```

Data Conversion:

```
 30-32 expr...                  1->2,2->4,1->4 integer
 33-35 expr...                  2->1,4->2,4->1 integer
 36-37 expr...                  4->8,8->4 real conversion
 38-39 expr...                  4->4,4->8 Float
 3A-3B expr...                  4->4,8->4 Trunc
 3C-3D expr...                  4->4,8->4 Round
   3E fff expr...               Extract unsigned field
   3F fff expr...               Extract signed field
```

Scalar operators:

```
 40-41 expr... expr...          2/4 Scalar Addition
 42-43 expr... expr...          2/4 Scalar Subtraction
 44-45 expr... expr...          2/4 Scalar Multplication
 46-47 expr... expr...          2/4 Scalar Division
 48-49 expr... expr...          2/4 Scalar Modulus
 4A-4B expr...                  2/4 Scalar Negate
 4C-4D expr...                  2/4 Scalar Absolute Value
 4E-4F expr...                  2/4 Scalar Square
 50-52 expr... expr...          1/2/4 Scalar AND
 53-55 expr... expr...          1/2/4 Scalar OR
 56-58 expr... expr...          1/2/4 Scalar XOR
 59-5B expr...                  1/2/4 Scalar NOT
 5C-5E expr... expr...          1/2/4 Scalar <
 5F-61 expr... expr...          1/2/4 Scalar >
 62-64 expr... expr...          1/2/4 Scalar <=
 65-67 expr... expr...          1/2/4 Scalar >=
 68-6A expr... expr...          1/2/4 Scalar =
 6B-6D expr... expr...          1/2/4 Scalar <>
   6E expr...                   Boolean NOT
   6F expr...                   ODD
 70-71 expr... expr...          4/8 Real Addition
 72-73 expr... expr...          4/8 Real Subtraction
 74-75 expr... expr...          4/8 Real Multiplication
 76-77 expr... expr...          4/8 Real Division
 78-79 expr... expr...          4/8 Real Modulus
 7A-7B expr... expr...          4/8 Real <
 7C-7D expr... expr...          4/8 Real >
 7E-7F expr... expr...          4/8 Real <=
 80-81 expr... expr...          4/8 Real >=
 82-83 expr... expr...          4/8 Real =
 84-85 expr... expr...          4/8 Real <>
 86-87 expr...                  4/8 Real Negation
 88-89 expr...                  4/8 Real Absolute Value
 8A-8B expr...                  4/8 Real Square
   8C
```

```
    8D
    8E
    8F
```

String Operators:

```
    90 expr... expr...          String <
    91 expr... expr...          String >
    92 expr... expr...          String <=
    93 expr... expr...          String >=
    94 expr... expr...          String =
    95 expr... expr...          String <>
    96 nnn nnn expr... expr...  PAOC <
    97 nnn nnn expr... expr...  PAOC >
    98 nnn nnn expr... expr...  PAOC <=
    99 nnn nnn expr... expr...  PAOC >=
    9A nnn nnn expr... expr...  PAOC =
    9B nnn nnn expr... expr...  PAOC <>
    9C
    9D
    9E
    9F
```

Set Operators:

```
    A0 nnn expr... expr...      Set +
    A1 nnn expr... expr...      Set -
    A2 nnn expr... expr...      Set *
    A3 nnn expr... expr...      IN
    A4 nnn expr... expr...      Set <=
    A5 nnn expr... expr...      Set >=
    A6 nnn expr... expr...      Set =
    A7 nnn expr... expr...      Set <>
    A8 nnn expr...              Singleton Set
    A9 nnn expr... expr...      Set Range
    AA nnn nnn expr...          Adjust Set
    AB
    AC
    AD
    AE
    AF
```

Procedure/Function Calls:

```
    B0 nnnnnnn                  User Function Call
    B1 nnnnnnn                  User Procedure Call
    B2 nnn                      Standard Function Call
    B3 nnn                      Standard Procedure Call
    B4 addr...                  Parametric Function Call
    B5 addr...                  Parametric Procedure Call
    B6 nnn                      Make Room for Function Result
    B7 addr...                  Reference Parameter
 B8-BB expr...                  1/2/4/8 Byte Value Parameter
    BC nnnnnnn expr...          Large Value Parameter
```

```
    BD nnn expr...              Set Value Parameter
    BE                          Begin Parameter List
    BF nnnnnnn                  User Function/Procedure Parameter
```

Control:

```
    C0 nnnnnnn                  Define Internal Label
    C1 nnnnnnn                  Jump
    C2 nnnnnnn expr...          Jump False
    C3 nnnnnnn expr...          Jump True
    C4 user-no nnnnnnn          Define Local User Label
    C5 user-no nnnnnnn link-no  Define Global User Label
    C6 user-no nnnnnnn          Jump to Local User Label
    C7 lev link-no              Jump to Global User Label
    C8 expr...                  Case Jump
    C9 0 lobound hibound elselab endlab
       lo--lab ... hi--lab      Case Table - must follow case jump
    C9 1 lobound hibound elselab count
       [value, label]           If expr list - must follow case jump
    CA nnn addr... expr...
       expr... expr...          FOR statement, nnn=1,2,4=size
    CB                          FOR end
    CC                          CASE end
    CD nnnn                     Line number
    CD -1 length filename       To open an include (or uses) file
    CE
    CF
    D0--DF
    E0--EF
    F0 (ln) (un) (sn) lev
       varsize prmbyts glb      Begin Module
                                (ln)     - 8-byte Linker name
                                (un)     - 8-byte User name
                                (sn)     - 8-byte Segment name
                                lev      - level (1=global)
                                varsize - Number of bytes of local variables
                                prmbyts - Bytes of parameters
                                glb      - Global Label Flag is Bit 0
                                           Stack Expan. Flag is Bit 1
                                regmask - register mask for MOVEM (D0..SP)
    F1 (ln) (un) nnnnnn lev     External Reference Definition
    F2 (cn) nnn                 Common Reference Definition
    F3 (cn) nnnnnnn             Common Area Definition
    F4 (un) textaddr4 textsize4
            globsiz2            Unit File Header
    F5
    F6--FD
    FE                          End of module
    FF                          End of file
```

PACKING INFORMATION

Packed records are very expensive in terms of the number of bytes of code generated by the compiler to reference a field of a packed record. In general, you should avoid packing records unless there are many more instances of a particular record than there are references to it.

Packed arrays are also code-expensive, with one exception. Packed arrays of char are treated as a special case, and the code associated with them is compact.

To paraphrase von Neumann, anyone who needs to know the details of the packing algorithms is in a state of sin, but the following is provided for the sake of completeness.

Elements of packed arrays are stored with multiple values per byte whenever more than one value can be fit into a byte. This only happens when the values require 4 bits or less. Values requiring 3 bits are stored into 4 bits.

The first value in a packed array is stored in the lowest numbered bit position of the lowest addressed (most significant) byte. Subsequent values are stored in the next available higher numbered bit positions within that byte. When the first byte is full, the same positions are used in the next higher addressed byte. Consider the following examples:

```
a:  PACKED ARRAY[1..12] OF BOOLEAN

    byte 1:                                 bit 0
    +----+----+----+----+----+----+----+----+
    | a8 | a7 | a6 | a5 | a4 | a3 | a2 | a1 |
    +----+----+----+----+----+----+----+----+

    byte 2:
    +----+----+----+----+----+----+----+----+
    |  --- Unused ---   | a12| a11| a10| a9 |
    +----+----+----+----+----+----+----+----+

b:  PACKED ARRAY[3..8] OF 0..3

    byte 1:
    +----+----+----+----+----+----+----+----+
    |  a[6]   |  a[5]   |  a[4]   |  a[3]   |
    +----+----+----+----+----+----+----+----+

    byte 2:
    +----+----+----+----+----+----+----+----+
    |  --- Unused ---   |  a[8]   |  a[7]   |
    +----+----+----+----+----+----+----+----+
```

```
c:  PACKED ARRAY[0..2] OF 0..7
          or
    PACKED ARRAY[0..2] OF 0..15

    byte 1:
    +----+----+----+----+----+----+----+----+
    |        a[1]       |        a[0]       |
    +----+----+----+----+----+----+----+----+

    byte 2:
    +----+----+----+----+----+----+----+----+
    |  --- Unused ---   |     a[2]          |
    +----+----+----+----+----+----+----+----+
```

You can use the @ operator to poke around inside any packed value and thereby discover what the packing algorithm (probably) is. For example, to get the data given above, you can use a program like the following:

```
Program Test;
Var i:integer;
    p:^integer;
    boolArr:packed array [1..12] of boolean;
Begin
boolArr[1]:=true;       (* find out where 1st bit is put *)
for i:=2 to 12 do boolArr[i]:=false;
p:=@boolArr;
WriteLn('equiv word is ',p^);
                        (* write the packed array as an integer *)
End.
```

TRANSLATION FROM APPLE PASCAL TO LISA PASCAL

Translation of Apple Pascal programs is usually not very difficult. The following hints may be of use to you if you find yourself saddled with the translation task. Thanks to Ken Friedenbach for the hints!

MOVELEFT(Source_Buf[i],Dest_Buf[k],n) can be translated into:

```
FOR LocalI:=0 TO n-1 DO Dest_Buf[LocalI+k]:=Source_Buf[LocalI+i];
```

It may be necessary to declare the local integer used as the FOR loop control variable.

MOVERIGHT(Source_Buf[i],Dest_Buf[k],n) becomes:

```
FOR LocalI:=n-1 DOWNTO 0 DO Dest_Buf[k+LocalI]:=Source_Buf[i+LocalI];
```

FILLCHAR(Buf[i],n,Ch) becomes:

```
FOR LocalI:=0 TO n-1 DO Buf[i+LocalI]:=ch;
```

i:=SCAN(n,<>ch,Buf[k]) becomes:

```
LocalI:=0;
IF n>0 THEN
  WHILE (LocalI<n) AND (Buf[k+LocalI]=ch) DO LocalI:=LocalI+1
ELSE
  WHILE (LocalI>n) AND (Buf[k+LocalI]=ch) DO LocalI:=LocalI-1;
i:=LocalI;
```

If SCAN is looking for =ch, just substitute <>ch in the loops above.

READ(KEYBOARD,ch) becomes:

```
UNITREAD(2,ChArr,1);
ch:=ChArr[0];
```

where chArr=packed array [0..1] of char.

EOLN(KEYBOARD)

can check the character read above. If ch=CHR(13) then EOLN is true.

KEYPRESS

is NOT UNITBUSY(2).

Strings must be given a length, non-local EXITs must be replaced with GOTOs. Sets with negative numbers can be shifted upward to fall within 0..MAXINT.

ClearScreen and other such functions can be handled by Jim Merritt's CUSTOMIO unit. ClearScreen on the Lisa is presently WRITE(CHR(27),CHR(42));

If underbars are used in the Apple Pascal program, they must be used consistently (they are ignored by the Apple Pascal Compiler!).

If the Apple Pascal units have code in the intialization block, put it in a procedure called at the beginning of the program.

To force segments to be resident, build a chain of dummy procedure calls that forces the loader to keep them all in core. The main program then becomes a procedure called by the top of the chain. Say we have 3 segments called SEG1, SEG2, and SEG3, and have put our main program into a procedure named MAIN_PROGRAM. We can now force everything to be memory resident by adding the following procedures:

```
(*$S SEG1*)
Procedure Kludge3;
BEGIN
Main_Program;
END;

(*$S SEG2*)
Procedure Kludge2;
BEGIN
Kludge3;
END;

(*$S SEG3*)
Procedure Kludge1;
BEGIN
Kludge2;
END;

(*$S            *)
BEGIN
Kludge1;
END.            (* end of main program *)
```

MPASLIB

Various Pascal procedures and functions call the run-time library (MPASLIB.OBJ). MPASLIB puts the parameters on the stack, followed by an index indicating which routine is to be called, and then traps to the monitor. For low level I/O, the monitor sends this information to the Apple II, which actually executes the I/O request.

This section gives a complete list of the indices and parameters handled in this manner, with both their assembler mnemonic name and the name of the Pascal procedure which invokes them. If a parameter or returned address is 32 bits long, it is preceded in the drawing of the parameter list given below by the word LONG.

Pascal-Name	Assembler-Name	Index	
WRITE(f,Ch)	FWRTCHAR	$8	8

```
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | Return Address |
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | File Pointer   |
       +----------------+
       |   Ch           |
       +----------------+
       | Index = $8     |
SP-->  +----------------+
```

Note: WRITE(f,Ch,i,j) is implemented as three calls on FWRTCHAR, one for each entity. Conversion (integer to char, for example) is done in the run-time library routines.

Pascal-Name	Assembler-Name	Index	
WRITELN(f)	FWRITELN	$C	12

```
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | Return Address |
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | File Pointer   |
       +----------------+
       | Index = $C     |
SP-->  +----------------+
```

READ(f,Ch) FREADCHR $10 16

```
        +---------------+
        |  (LONG)       |
        + - - - - - - - +
        | Func Result   |
        +---------------+
        |  (LONG)       |
        + - - - - - - - +
        | Return Address|
        +---------------+
        |  (LONG)       |
        + - - - - - - - +
        | File Pointer  |
        +---------------+
        | Index = $10   |
SP-->   +---------------+
```

READLN(f) FREADLN $14 20

```
        +---------------+
        |  (LONG)       |
        + - - - - - - - +
        | Return Address|
        +---------------+
        |  (LONG)       |
        + - - - - - - - +
        | File Pointer  |
        +---------------+
        | Index = $14   |
SP-->   +---------------+
```

RESET and REWRITE FINIT $18 24

```
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | File Pointer   |
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | Window Pointer |
       +----------------+
       | Record Size    | (-2 = text, -1 = file, >0 = #words per item)
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | Return Address |
       +----------------+
       | Index = $18    |
SP-->  +----------------+
```

FOPEN $1C 28

```
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | File Pointer   |
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | File Title     |
       +----------------+
       | Open Old (Bool)|
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | Zero           |
       +----------------+
       |  (LONG)        |
       + - - - - - - - -+
       | Return Address |
       +----------------+
       | Index = $1C    |
SP-->  +----------------+
```

Note: FINIT initializes the file buffer. FOPEN opens a new file (REWRITE) if Open Old is false. It opens an old file (RESET) if Open Old is true.

BLOCKREAD and BLOCKWRITE   BLKIO $20   32

```
      +----------------+
      | (LONG)         |
      + - - - - - - - -+
      | Function result|
      +----------------+
      | (LONG)         |
      + - - - - - - - -+
      | File Pointer   |
      +----------------+
      | (LONG)         |
      + - - - - - - - -+
      | Buffer Address |
      +----------------+
      | Number of Blocks|
      +----------------+
      | Block Number   | (-1 = sequential)
      +----------------+
      | Read(1)/Write(0)|
      +----------------+
      |  (LONG)        |
      + - - - - - - - -+
      | Return Address |
      +----------------+
      | Index = $20    |
SP--> +----------------+
```

NEW(p) MNEW $24 36

```
         +---------------+
         |  (LONG)       |
         + - - - - - - - +
         | Return Address|
         +---------------+
         |  (LONG)       |
         + - - - - - - - +
         |Pointer Address|
         +---------------+
         |Number of Words|
         +---------------+
         | Index = $24   |
  SP-->  +---------------+
```

MARK(p) MMRK $28 40

```
         +---------------+
         |  (LONG)       |
         + - - - - - - - +
         | Return Address|
         +---------------+
         |  (LONG)       |
         + - - - - - - - +
         |Pointer Address|
         +---------------+
         | Index = $28   |
  SP-->  +---------------+
```

RELEASE(p) MRLS $2C 44

```
         +---------------+
         |  (LONG)       |
         + - - - - - - - +
         | Return Address|
         +---------------+
         |  (LONG)       |
         + - - - - - - - +
         |Pointer Address|
         +---------------+
         | Index = $2C   |
  SP-->  +---------------+
```

```
MEMAVAIL                      MEMA                        $30    48

        +----------------+
        |  (LONG)        |
        + - - - - - - - -+
        | Return Address |
        | and func result|
        +----------------+
        | Index = $30    |
  SP--> +----------------+
```

```
UNITCLEAR(u)                  UCLR                        $34    52

        +----------------+
        |  (LONG)        |
        + - - - - - - - -+
        | Return Address |
        +----------------+
        | Unit Number    |
        +----------------+
        | Index = $34    |
  SP--> +----------------+
```

```
UNITREAD                      UREAD                       $38    56
UNITWRITE                     UWRITE                      $3C    60

        +----------------+
        |  (LONG)        |
        + - - - - - - - -+
        | Return Address |
        +----------------+
        | Unit Number    |
        +----------------+
        | (LONG)         |
        + - - - - - - - -+
        | Buffer Address |
        +----------------+
        | Number of Bytes|
        +----------------+
        | Block Number   |
        +----------------+
        | Mode           |
        +----------------+
        | Index=$38 or $3C|
  SP--> +----------------+
```

```
   UNITBUSY(u)                 UBUSY                  $40    64

      +---------------+
      |  (LONG)       |
      + - - - - - - - +
      | Return Address |
      | and Func Result |
      +---------------+
      | Unit Number   |
      +---------------+
      | Index = $40   |
SP--> +---------------+
```

```
   IORESULT                    $$IORES                $44    68

      +---------------+
      |  (LONG)       |
      + - - - - - - - +
      | Return Address |
      | and Func Result |
      +---------------+
      | Index = $44   |
SP--> +---------------+
```

```
   CLOSE(f)                    FCLOSE                 $4C    76

      +---------------+
      |  (LONG)       |
      + - - - - - - - +
      | File Pointer  |
      +---------------+
      | Mode          | (0=normal, 1=LOCK, 2=PURGE, 3=CRUNCH)
      +---------------+
      |  (LONG)       |
      + - - - - - - - +
      | Pointer Address |
      +---------------+
      | Index = $4C   |
SP--> +---------------+
```

MHALT $50 80 HALT

```
        +-----------------+
        | (LONG)          |
        + - - - - - - - - +
        | Return Address  |
        +-----------------+
        | Index = $50     |
SP-->   +-----------------+
```

MIOERR $54 84

```
        +-----------------+
        | (LONG)          |
        + - - - - - - - - +
        | Return Address  |
        +-----------------+
        | Index = $54     |
SP-->   +-----------------+
```

MGOTOXY $58 88 GOTOXY

```
        +-----------------+
        | (LONG)          |
        + - - - - - - - - +
        | Return Address  |
        +-----------------+
        | X coordinate    |
        +-----------------+
        | Y coordinate    |
        +-----------------+
        | Index = $58     |
SP-->   +-----------------+
```

```
RCERR          $5C    92    (Range value error)

       +-----------------+
       | (LONG)          |
       + - - - - - - - - +
       | Return Address  |
       +-----------------+
       | Index = $5C     |
SP-->  +-----------------+
```

```
SCERR          $60    96    (String index error)

       +-----------------+
       | (LONG)          |
       + - - - - - - - - +
       | Return Address  |
       +-----------------+
       | Index = $60     |
SP-->  +-----------------+
```

There are three indices to handle segment swapping:

$$LOADIT $0 (loads a segment)

```
        +---------------+
        | (LONG)        |
        + - - - - - - - +
        | Fake Return Addr|
        +---------------+
        | (LONG)        |
        + - - - - - - - +
        | Return Address |
        +---------------+
        | Index = $0    |
 SP-->  +---------------+
```

$$UNLOAD $4 (unload a segment)

```
        +---------------+
        | Index = $4    |
 SP-->  +---------------+
```

REMOVE1 $48 (72) (used by GOTO to unload a segment)

```
        +---------------+
        | (LONG)        |
        + - - - - - - - +
        | Return Address |
        +---------------+
        | Index = $48   |
 SP-->  +---------------+
```

MONITOR--APPLE II INTERFACE

1) PROCEDURE UNITCLEAR(UNITNUMBER);

2) PROCEDURE UNITREAD(UNITNUMBER, ARRAY, LENGTH, BLOCKNUMBER, ASYNC);

3) PROCEDURE UNITWRITE(UNITNUMBER, ARRAY, LENGTH, BLOCKNUMBER, ASYNC);

4) FUNCTION UNITBUSY(UNITNUMBER):BOOLEAN;

The ASYNC parameter is ignored, and the UNITWAIT procedure is a no-op.

The IO sequence is:

1) The 68000 sends an IO command byte followed by a one byte unit number.

2) If the command byte is a UNITREAD or UNITWRITE, the 68000 also sends:

   a) 4 byte address
   b) 2 byte length
   c) 2 byte block number

3) The Apple II interprets the IO command as follows:

   UNITCLEAR -- A UNITCLEAR is issued for the unit specified. IORESULT is sent to the 68000.

   UNITREAD -- A UNITREAD is issued for the unit specified. PUTSTREAM is used to write data into the 68000. IORESULT is sent to the 68000.

   UNITWRITE -- GETSTREAM is used to read data from the 68000. A UNITWRITE is issued for the unit specified. IORESULT is sent to the 68000.

   UNITBUSY -- If the unit number is 1 or 2, NOT KEYPRESS is returned, otherwise false is returned. IORESULT is sent to the 68000.

4) The Apple II returns the IORESULT, then waits for the next I/O command byte. The 68000 continues execution.

IO Command Summary:

UNITCLEAR	1 UNITNUM IORSLT
UNITREAD	2 UNITNUM ADDR COUNT BLKNUM RDDATA IORSLT
UNITWRITE	3 UNITNUM ADDR COUNT BLKNUM WRDATA IORSLT
UNITBUSY	4 UNITNUM BUSYFLAG IORLST

where:

UNITNUM	1 byte unit number
ADDR	4 byte address
COUNT	2 byte count
BLKNUM	2 byte block number
RDDATA	byte stream to the Apple II
WRDATA	byte strea from the Apple II
BUSYFLAG	2 byte function result
IORSLT	2 byte IORESULT

SUMMARY OF INDICES

Name	Index	Parameters							
$$LOADIT	$0	(Rtn adr)	Rtn adr	Index					
$$UNLOAD	$4	Index							
FWRTCHAR	$8	Rtn adr	File Ptr	Char	Index				
FWRITELN	$C	Rtn adr	File Ptr	Index					
FREADCHR	$10	Rtn adr	Result	File Ptr	Index				
FREADLN	$14	Rtn adr	File Ptr	Index					
FINIT	$18	File Ptr	Window	RecSize	Rtn adr	Index			
FOPEN	$1C	File Ptr	Title	Old	Zero	Rtn adr	Index		
BLKIO	$20	Result	File Ptr	Buffer	Blocks	Block#	Read/Write	Rtn adr	Index
MNEW	$24	Rtn adr	Ptr adr	Words	Index				
MMRK	$28	Rtn adr	Ptr adr	Index					
MRLS	$2C	Rtn adr	Ptr adr	Index					
MEMA	$30	Result	Rtn adr	Index					
UCLR	$34	Rtn adr	Unit#	Index					
UREAD	$38	Rtn adr	Unit#	Buffer	#Bytes	Block#	Mode	Index	
UWRITE	$3C	Rtn adr	Unit#	Buffer	#Bytes	Block#	Mode	Index	
UBUSY	$40	Rtn adr	Unit#	Index					
$$IORES	$44	Rtn adr	Index						
REMOVE1	$48	Rtn adr	Index						
FCLOSE	$4C	File Ptr	Mode	Rtn adr	Index				
MHALT	$50	Rtn adr	Index						
MIOERR	$54	Rtn adr	Index						
MGOTOXY	$58	Rtn adr	X Coord	Y Coord	Index				
RCERR	$5C	Rtn adr	Index						
SCERR	$60	Rtn adr	Index						

MPASLIB ROUTINES

%%%TERM
%_TERM

%%%MATH
%I_MUL4
%I_DIV4
%I_MOD4

%%%MOVE
%_MOVEL
%_MOVER
%_FILLC
%_SCANE
%_SCANN

%%%TRING
%_CAT
%_POS
%_COPY
%_DEL
%_INS

%%%SCOMP
%S_NE
%S_EQ
%S_GT
%S_LE
%S_LT
%S_GE

%%%SET
%_INTER
%_SING
%_UNION
%_DIFF
%_RDIFF
%_RANGE
%_ADJ
%_SETNE
%_SETEQ
%_SETGE
%_SETLE

%%%TEXT
%W_LN
%W_C
%W_STR
%W_PAOC
%W_I
%W_B
%_PAGE
%R_C

```
        %R_LN
        %R_PAOC
        %R_STR
        %R_I

        %%%MISC
        %_HALT
        %_IOERR
        %_GOTOXY
        %_LSTSG
        %_GOTO

        %%%IO
        %_REWRT
        %_RESET
        %_CLOSE
        %_EOF
        %_EOLN
        %_BLKRD
        %_BLKWR
        %_UREAD
        %_UWRIT
        %_IORES
        %_UCLR
        %_UBUSY
        %_GET
        %_PUT
        %_UPARR
        %_SEEK

    %%%MEM
        %_NEW
        %_MARK
        %_RELSE
        %_MEMAV

    FP%DEFAU     0010     000000
        %%%BASE
        $DECX
        %W_F
        %W_E
        %XPOT
        %XMUL
        %XINT
        %XDIV32
        %XDIV
        %XDEC
        %XCOMP
        X%TOS
        X%TODEC
        X%STO
        X%POT
        X%MUL
        X%MINUS
```

```
       X%KIND
       X%INT
       X%DIV
       X%DEC
       X%COMP
       STRINGTO
       STRING%R
       REALTOST
       REAL%STR
       FPDEFAUL

  $$$INIT
       %_BEGIN
       %_END
       %_INIT

  %%%RANGE
       %_RCHCK
       %_SRCHK

       SWAPMODE

       SWAPTRAP

       SWAPEXCE

  %%%REAL
       UNPACKRB
       UNPACKRA
       PACKR
       %_TRUNC
       %_ROUND
       %_PWR10
       %I32F32
       %F_SUB
       %F_NEG
       %F_NE
       %F_MUL
       %F_LT
       %F_LE
       %F_GT
       %F_GE
       %F_EQ
       %F_DIV
       %F_ADD
       %F_ABS
       %F32SUB
       %F32NE
       %F32MUL
       %F32LT
       %F32LE
       %F32I32
       %F32GT
       %F32GE
```

%F32EQ
%F32DIV
%F32ADD
%I_FLT

LINKER FILE LAYOUT

The linker tries to handle object files created by several different versions of the compiler and previous versions of the linker, created for several different versions of the hardware. Not surprisingly, object file formats are a mess. There are three basic types of object file:

OldExecutable	--	release 1.0 to 5.0 compilers, either machine
PhysicalExec	--	release 6.0 compiler or later, release 1.0 to 5.3 linkers, either machine
Executable	--	release 6.0 compiler or later, release 6.0 linker or later, new hardware

The release numbers refer to Monitor releases. The new linker (release 6.0 or later) can handle intrinsic units, and produces a version control record. There are two distinctly different "old" linkers: the OldLinker and the HackedLinker. Some attempt is made below to distinguish object files created by these linkers.

An additional source of woe is the group of blocks created by the symbolic debugger. Because these undergo constant change, the documentation is always wrong. The information given here was correct at some point in September, 1981.

As used below, * means 0 or more, + means 1 or more.

```
<LinkFile>        ::=  <ModuleFile>      (* main program output from compiler *)
                                         (* or Assembler *)
                  ::=  <LibraryFile>     (* output of LIBRARY program -- no *)
                                         (* longer fully supported *)
                  ::=  <UnitFile>        (* unit output from compiler *)
                  ::=  <IntrinLibFile>   (* *INTRINSIC.LIB itself -- only *)
                                         (* one per boot disk *)
                  ::=  <ExecuteFile>     (* output of Linker *)

<ModuleFile>      ::=  <Module>*
                       EOFMark

<LibraryFile>     ::=  LibModule+
                       LibEntry+
                       <Module>+
                       TextBlock*
                       EOFMark

<UnitFile>        ::=  UnitBlock
                       <Module>+
                       TextBlock
                       EOFMark

<IntrinLibFile>   ::=  VersionCtrl
                       UnitLocation
                       SegLocation
                       FilesBlock
                       EOFMark

<ExecuteFile>     ::=  <ExecutableHeader>
                       <Module>*
                       <OtherBlock>*
                       EOFMark

<ExecutableHeader>
                  ::=  OldExecutable  (* old linker *)
                  ::=  PhysicalExec   (* HackedLinker *)
                  ::=  VersionCtrl
                       Executable     (* new linker without intrinsic units *)
                  ::=  VersionCtrl
                       UnitTable
                       Executable
                       SegmentTable   (* new linker with intrinsic units *)
                       UnitBlock*     (* one per unit linked into this file *)

<Module>          ::=  ModuleName
                       <OtherModBlock>+
                       EndBlock

<OtherModBlock>   ::=  EntryPoint
                  ::=  External
                  ::=  StartAddress
```

```
                ::=  CodeBlock
                ::=  Relocation
                ::=  CommonReloc
                ::=  CommonDef
                ::=  ShortExternal

<OtherBlock>    ::=  DebugSymbols
                ::=  DebugEntry
                ::=  DebugCommon
```

There may also be an ExecIUUnit ("Executable Intrinsic Unit Unit"?):

```
                ::=  VersionCtrl
                     UnitTable
                     SegmentTable
```

You can sometimes tell which linker produced the file you are dealing with by looking at the first blocks. The newest linker always puts the VersionCtrl block first, the HackedLinker started with either the PhysicalExec or the Executable block, and the old linker started with the Executable block. The Pascal definitions of all the blocks given below can be found in OBJIO.TEXT.

The Segment and UnitTable blocks are found in object files that have linked intrinsic unit code segments. The Loader then uses the SegLocation, UnitLocation, and FilesBlock blocks in INTRINSIC.LIB to locate these intrinsic units.

ModuleName:

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 80 |  size  |  module name  | segment name  | csize | comment ...  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
 1   2     4 5            12 13           20 21    24               size
```

```
80           - Hexadecimal 80
size         - Number of bytes in this block
module name  - Blank padded ASCII name of this module
segment name- ASCII name of segment in which this module will reside
csize        - Number of bytes in the code block for this module
comment      - Arbitrary information.  Ignored by the Linker.
```

EndBlock:

```
+----+---+---+---+-+-+-+-+
| 81 |  size   | csize  |
+----+---+---+---+-+-+-+-+
 1    2     4  5     8
```

```
81        - Hexadecimal 81
size      - Number of bytes in this block (always 000008)
csize     - Numer of bytes in the code block for this module
```

EntryPoint:

```
| 82 |  size  |     link name    |    user name    |  loc  | comment ...  |
  1    2     4 5             12 13             20 21    24 25           size
```

```
82         - Hexadecimal 82
size       - Number of bytes in this block
link name  - Blank padded ASCII linker name of entry point
user name  - Blank padded ASCII user name of entry point
loc        - Location of entry point relative to this module
             (a zero based byte address within a segment)
comment    - Arbitrary information.  Ignored by Linker
```

External:

```
| 83 |  size  |     link name    |    user name    | ref 1 | ... | ref n |
  1    2     4 5             12 13             20 21    24              size
```

```
83         - Hexadecimal 83
size       - Number of bytes in this block
link name  - Blank padded ASCII linker name of external reference
user name  - Blank padded ASCII user name of external reference
ref 1      - Location of first reference relative to this block
...        - Other references
ref n      - Location of last reference
```

StartAddress:

```
+----+--+--+--+-+-+-+-+-+-+-+-+-+-+--------+
| 84 |  size  | start | gsize | comment ...  |
+----+--+--+--+-+-+-+-+-+-+-+-+-+-+--------+
 1     2     4  5     8 9    12 13         size
```

```
84       - Hexadecimal 84
size     - Number of bytes in this block
start    - Starting address relative to this module
gsize    - Number of bytes in the global data area
comment  - Arbitrary information.  Ignored by the Linker.
```

CodeBlock:

```
+----+--+--+--+-+-+-+-+-+-+-+-+-------+
| 85 |  size  | addr  | object code ...|
+----+--+--+--+-+-+-+-+-+-+-+-+-------+
 1     2     4  5     8 9            size
```

```
85          - Hexadecimal 85
size        - Number of bytes in this block
addr        - Module relative address of first byte of code
object code- The object code.  Always an even number of bytes.
```

Relocation:

```
+----+--+--+--+-+-+-+-+-------+-+-+-+
| 86 | size   | ref 1 | ... | ref n |
+----+--+--+--+-+-+-+-+-------+-+-+-+
 1     2     4  5     8           size
```

```
86       - Hexadecimal 86
size     - Number of bytes in this block
ref 1    - Location of first address to relocate
...      - Other addresses
ref n    - Location of last address to relocate
```

CommonReloc:

```
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+------+---+---+---+
| 87 |   size    |    common name     | ref 1  |  ...  | ref n  |
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+------+---+---+---+
 1    2        4 5               12 13     16              size
```

```
87          - Hexadecimal 87
size        - Number of bytes in this block
common name - Blank padded ASCII name of common block
ref 1       - Location of first reference relative to this module
...         - Other references
ref n       - Location of last reference
```

CommonDef:

```
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+--------------+
| 88 |   size    |    common name     | dsize  | comments ... |
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+--------------+
 1    2        4 5               12 13     16 17         size
```

```
88          - Hexadecimal 88
size        - Number of bytes in this block
common name - Blank padded ASCII name of common area
dsize       - Number of bytes in this common data area
comments    - Arbitrary information.  Ignored by the Linker.
```

ShortExternal:

```
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 89 |   size    |     link name      |     user name     |  ref  |
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
  1   2        4 5               12 13               20 21 22
```

```
89          - Hexadecimal 89
size        - Number of bytes in this block (always 000016)
link name   - Blank padded ASCII linker name of external reference
user name   - Blank padded ASCII user name of external reference
ref         - Location of reference
```

OldExecutable:

```
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+------+
| 8F |  size   | JT laddr     | JT size      | data size | jump table ...     |
+----+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+------+
 1     2     4  5           8  9          12 13       16                   size
```

```
8F          - Hexadecimal 8F
size        - Number of bytes in this block
JT laddr    - Absolute load address of jump table
JT size     - Number of bytes in jump table
data size   - Total number of bytes in global common data areas
jump table  - The jump table itself, including the executable
              code for the loader.  This table is in the old
              jump table format (see below).
```

LibModule:

```
+----+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+----
| 90 | size     |  module name   | msize | caddr | taddr | tsize | ...
+----+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+----
 1      2     4  5            12 13   16 17   20 21   24 25    28

               ---+----+----+----+-----+------+----+----+
               ...| #mods | mod 1 | ...  | mod n |
               ---+----+----+----+-----+------+----+----+
                   29   30   31   32             size
```

```
90           - Hexadecimal 90
size         - Number of bytes in this block
module name  - name of this module
msize        - Size of code for this module in bytes
caddr        - Disk address of module
taddr        - Disk address of text block, if any (0 otherwise)
tsize        - Size of text block
#mods        - Number of other modules referenced by this module
mod 1        - Number of first module referenced
...          - Other module numbers
mod n        - Number of last module referenced
```

LibEntry:

```
+----+---+---+---+-+-+-+-+-+-+-+----+----+---+---+---+
| 91 |  size   |    link name     | module  | address     |
+----+---+---+---+-+-+-+-+-+-+-+----+----+---+---+---+
 1     2     4  5              12 13      14 15        18
```

```
91          - Hexadecimal 91
size        - Number of bytes in this block (always 000012)
link name   - Blank padded ASCII link name of entry point
module      - Module in which entry point resides
address     - Relative address of entry point to that module
```

UnitBlock:

```
+----+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---+
| 92 | size    |    unit name     | caddr | taddr | tsize | gsize |type |
+----+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---+
 1     2     4 5               12 13    16 17    20 21   24 25    28     30
```

```
92          - Hexadecimal 92
size        - Number of bytes in this block (always 00001E)
unit name   - Name of this unit
caddr       - Disk address of module
taddr       - Disk address of text block
tsize       - Size of text block
gsize       - Number of bytes of globals in this unit
type        - unit type: 0=regular, 1=intrinsic, 2=shared
```

PhysicalExec:

```
+----+---+---+---+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---+---+---+---
| 97 |  size   | JT laddr   |JTsize | dsize | msize |JTkSegDelta| ...
+----+---+---+---+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+---+---+---+---
 1     2     4  5        8  9      13        17      21              25

    --+---+---+---+------------------+
   ...|StkSegDelta| Jump Table ...   |
    --+---+---+---+------------------+
      25          29             size
```

```
97          - Hexadecimal 97
size        - Number of bytes in this block
JT laddr    - Absolute load address of jump table
JTsize      - Number of bytes in jump table
dsize       - Total number of bytes in regular units global data areas
msize       - Size of main program global data area
JTSegDelta  - Distance from base of segment to beginning of data pointers
StkSegDelta- see below
jump table  - The jump table itself, including the executable
              code for the loader.  This table is in the old
              jump table format (see below).
```

Executable:

```
+----+---+---+---+---+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---
| 98 | size    | JT Laddr  |JTSize | dsize | msize | JTSegDelta|StkSegDelta|...
+----+---+---+---+---+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+---+---+---+---+---+---+---
1    2     4 5         8 9      13        17       21           25

   ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+------+
   ...| Dyn Stack | Max Stack | Min Heap  | Max Heap  | Jump Table ...   |
   ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+------+
      29          33          37          41          45               size
```

```
98          - Hexadecimal 98
size        - Number of bytes in this block
JT laddr    - Absolute load address of jump table
JTsize      - Number of bytes in jump table
dsize       - Total number of bytes in regular units global data areas
msize       - Size of main program global data area
JTSegDelta  - Distance from base of segment to beginning of data pointers
StkSegDelta- Distance from JTSegDelta to A5 at runtime
Dyn Stack   - Initial dynamic stack size
Max Stack   - Maximum total stack size
Min Heap    - Initial heap size
Max Heap    - Maximum total heap size
jump table  - The jump table itself, including the executable
              code for the loader.  This table is in the new
              jump table format (see below).
```

VersionCtrl:

```
+----+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| 99 |   Size   | SrcMin| SrcMax| CmpMin| CmpMax| LnkMin| LnkMax|
+----+---+---+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
1    2          5       9       13      17      21      25
```

```
99       - Hexadecimal 99
Size     - Always 00001C
SrcMin   - Minimum source version number
SrcMax   - Maximum source version number
CmpMin   - Minimum compiler version number
CmpMax   - Maximum compiler version number
LnkMin   - Minimum linker version number
LnkMax   - Maximum linker version number
```

SegmentTable:

```
+----+---+---+---+----+----+----------+-    -+----------+
| 9A | size    | Nsegs | segInfo1 | ... | segInfoN |
+----+---+---+---+----+----+----------+-    -+----------+
 1     2       4 5       7           25           size
```

9A - Hexadecimal 9A
size - Number of bytes in segment table block
NSegs - Number of segment descriptors in table. Each Seginfo record is of the form:

```
+-+-+-+-+-+-+-+-+----+----+---+---+---+---+---+---+
|  Seg Name      |SegNumber| Version1  | Version2  |
+-+-+-+-+-+-+-+-+----+----+---+---+---+---+---+---+
 1               9         11          15        18
```

Seg Name - Segment Name
SegNumber - MMU number (currently)
Version1 - Version control info
Version2 - Version control info

UnitTable:

```
+----+---+---+---+----+----+-----+-----+----------+-    -+----------+
| 9B | size    | NUnits| maxunit |UnitInfo1 | ... |UnitInfoN |
+----+---+---+---+----+----+-----+-----+----------+-    -+----------+
 1     2       4 5       7         9             21            size
```

9B - Hexadecimal 9B
size - Number of bytes in unit table block
NUnits - Number of unit descriptors in table.
maxunit - maximum unit number. If, for example, units number 1, 7, and 11 are present, nunits=3 and maxunit=11.

Each Unitinfo record is of the form:

```
+-+-+-+-+-+-+-+-+----+----+----+----+
|  UnitName      | UnitNum |Unit type|
+-+-+-+-+-+-+-+-+----+----+----+----+
 1               9   10   11   12
```

UnitName - Unit Name
UnitNum - Index into data pointer table
Unit type - 0=Regular Unit (an error)
1=Regular Intrinsic Unit
2=Shared Intrinsic Unit

SegLocation:

```
+----+--+--+--+---+---+----------+-  -+----------+
| 9C | size    | Nsegs | segInfo1 | ... | segInfoN |
+----+--+--+--+---+---+----------+-  -+----------+
 1    2      4 5      7            35          size
```

9C - Hexadecimal 9C
size - Number of bytes in segLocation block
NSegs - Number of segment descriptors in table. Each Seginfo record is of the form:

```
+-+-+-+-+-+-+-+-+----+----+---+---+---+---+---+---+---
|  Seg Name      |SegNumber| Version1  | Version2  |....
+-+-+-+-+-+-+-+-+----+----+---+---+---+---+---+---+---
 1               9        11         15       18

       ---+---+---+-+-+-+-+-+-+-+
       ... | FileNo|FileLoc| CSize |
       ---+---+---+-+-+-+-+-+-+-+
           19      21      25
```

Seg Name - Segment Name
SegNumber - MMU number (currently)
Version1 - Version control info
Version2 - Version control info
FileNo - Index into the FilesBlock file table
FileLoc - Byte location within file of CodeBlock
CSize - code size?

UnitLocation:

```
+----+--+--+--+---+---+----------+-  -+----------+
| 9D | size    | NUnits|UnitInfo1 | ... |UnitInfoN |
+----+--+--+--+---+---+----------+-  -+----------+
 1    2      4 5      7            23           size
```

9D - Hexadecimal 9D
size - Number of bytes in unitLocation block
NUnits - Number of unit descriptors in table. Each Unitinfo record is of the form:

```
+-+-+-+-+-+-+-+-+----+----+----+----+---+---+---+
|  UnitName      | UnitNum |Unit type| Data Size |
+-+-+-+-+-+-+-+-+----+----+----+----+---+---+---+
 1               9    10   11   12   13       16
```

UnitName - Unit Name
UnitNum - Index into data pointer table
Unit type - See UnitTable above
Data Size - Size in bytes of global data area for unit

FilesBlock:

```
+----+--+--+--+----+----+-+-+-+-+-+-+--  --+-+-+-+-+-+-+---------------+
| 9E | Size    | nFiles  | FileInfo1 | ...  | FileInfoN | StringTable ... |
+----+--+--+--+----+----+-+-+-+-+-+-+--  --+-+-+-+-+-+-+---------------+
 1     2      4  5     6   7            13                            size
```

9E - Hexadecimal 9E
nFiles - number of file descriptors in block. Each Fileinfo record has the form:

```
+----+----+--+--+--+--+
| FileNo  |  NameAddr |
+----+----+--+--+--+--+
```

FileNo - Index into FilesBlock table
NameAddr - Byte address within this file (*INTRINSIC.LIB) of a Pascal string

StringTable - Table of file names

DebugSymbols:

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
| C0 | size   |    proc name    |    seg name    | proc base      |...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
 1    2      4 5              12 13             20 21             29

        ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
        ...| proc syms | proc stmt | proc node | uses size |...
        ---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---
            30       33 34       37 38       41 42       45
```

```
    C0          - hexadecimal $C0
    size        - Size in bytes of this record (including C0 and size)
    proc name   - Name of the module these symbols are for
    seg name    - Name of the segment this module is in
    proc base   - Low core address for this procedure
    proc syms   - Core address of root of symbol table for this proc
    proc stmt   - Core address of root of stmt tree for this proc
    proc node   - Core address of procedure definition node record
    uses size   - File size of Uses definition section in bytes (not including
                    hole top and hole base)
if uses size > 0 then
```

```
---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-----------
...| hole base | hole top  | map base  | map top   | map name       | proc heap ...
---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+-----------
    46       49 50       53 54       57 58       61 62             69
```

```
    hole base   - Lowest core address of used units' symbols
    hole top    - Highest core address of used units' symbols
                  (UsesSize bytes of these records one per used unit)
    map base    - Core address of base of this used units symbols
    map top     - Core address of top of this used units symobls
    map name    - Name of this used unit
    proc heap   - heap for this proc (starting at proc base)
```

DebugEntry:

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+------------
| C1 | size   |    proc name    | entry seg | entry loc | comment...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+------------
 1    2      4 5              12 13       16 17       20 21
```

```
    C1          - hexadecimal $C1
    size        - Size in bytes of this record (including C1 and size)
    proc name   - Name of the module this is entry point for
    entry seg   - Segment number of this proc's entry point
    entry loc   - Offset within the segment of this proc's entry point
    comment     - Arbitrary information ignored by the Linker
```

DebugCommon:

```
+----+--+--+--+--+-+-+-+-+-+-+-+--+--+--+--+----------
| C2 | size    |    unit name    |common base| comment...
+----+--+--+--+--+-+-+-+-+-+-+-+--+--+--+--+----------
 1     2      4 5             12 13        16
```

```
C2          - hexadecimal $C2
Size        - Size in bytes of this record (including C1 and size)
unit name   - Name of the unit this is common area definition of
common base - Core address of base of common area of this unit
comment     - Arbitrary information ignored by the Linker
```

TextBlock:

```
+---+---+---+---+---+--------+
| Textual data ...           |
+---+---+---+---+---+--------+
```

The operating system determines the format of the text block. The current version uses the UCSD format without the two initial header blocks. The text block is always stored on disk block boundaries.

EOFMark:

```
+---+---+
| 00 00 |
+---+---+
```

Intrinsic Unit Trap Handler

(Uses the Line 1010 exception handler)

Instruction Definitions:

```
IUJSR   xxxxxx    1010 0000 SSSS SSSO  0000 0000 0000 0000 {word address}
IUJMP   xxxxxx    1010 01XX SSSS SSSO  0000 0000 0000 0000 {word address}
IULEA   xxxxxx,Ai 1010 10AA ASSS SSSS  0000 0000 0000 0000 {0 implied}
IUPEA   xxxxxx    1010 11XX SSSS SSSO  0000 0000 0000 0000 {byte address}
```

Note: xxxxxx represents a 24 bit logical address

These instructions preserve all registers except CCR and Ai.
The instruction IULEA xxxxxx,A7 is undefined.
A's give a three bit a register descriptor.
S's give a seven bit segment number.
O's give a 17 bit offset (16 bits for IULEA with bit 0 an implied 0).
X's are don't care bits in the current decoding scheme.

By the way, all four instructions use RTS to continue execution.

The total cycles for the emulated JSR instruction equals 264 cycles.

Note: 110 of the 264 cycles are consumed in preservation of registers.

```
SDSEG   .EQU    $C00                     ; 1st LONG common to all domains
SDSEG2  .EQU    $C04                     ; 2nd LONG for saving A0 register

;
; Note: SDSEG must be either common to all domains or at least common to the
;       two domains of interest (ie. domain zero and the active user domain).
;       This means that either the OS or the OS drivers can not reference any
;       intrinsic units (since this code is not re-entrant due to the obvious
;       hardware constraints).  One way to allow both the OS and the OS
;       drivers to use intrinsic units is to have the exception handlers push
;       and pop SDSEG & SDSEG2 as though they were part of the machine state.
;       Another way to solve this problem would be to recode the LINE 1010
;       Trap handler so that it did not use absolute memory locations.  If we
;       assume that the user stack is always available to the LINE 1010 Trap
;       handler then the trap handler could use the user stack at the expense
;       of the extra pushes and pops.
;

;       Regular procedure call/return overhead equals 258 cycles (ie. 30us).
;
;               MOVE.L  A3,-(A7)            ( 16)
;               MOVE.W  D4,-(A7)            ( 12)
;               MOVE.W  e(A6),-(A7)         ( 20)
;               JSR     e(A5)               ( 20)
;               JMP     $xxxxxx             ( 12)
```

```
;               TST.W    e(A7)                   ( 12)
;               LINK     A6,#e                   ( 20)
;               MOVEM.L  A3/A4/D4-D7,-(A7)       ( 58)
;               ...
;               MOVEM.L  (A7)+,A3/A4/D4-D7       ( 52)
;               UNLK     A6                      ( 12)
;               MOVE.L   (A7)+,A0                ( 12)
;               ADDQ.W   #8,A7                   (  4)
;               JMP      (A0)                    (  8)
;
;       Intrinisc procedure call/return overhead equals 490 cycles (ie. 76us).
;
;               MOVE.L   A3,-(A7)                ( 16)
;               MOVE.W   D4,-(A7)                ( 12)
;               MOVE.W   e(A6),-(A7)             ( 20)
;               IUJSR    e(A5)                   (264)
;               TST.W    e(A7)                   ( 12)
;               LINK     A6,#e                   ( 20)
;               MOVEM.L  A3/A4/D4-D7,-(A7)       ( 58)
;               ...
;               MOVEM.L  (A7)+,A3/A4/D4-D7       ( 52)
;               UNLK     A6                      ( 12)
;               MOVE.L   (A7)+,A0                ( 12)
;               ADDQ.W   #8,A7                   (  4)
;               JMP      (A0)                    (  8)
;

        .PROC   %%IUTRAP,0
;
;       Starting with a Line 1010 emulation exception                    (40)
;


;
; ***** DOMAIN ZERO *****
;
; Note: The following three lines will be in LISABUG (Might be copied into OS).
;
L1010:  MOVE.L  2(A7),SDSEG              ; Copy PC into shared data seg  (32)
        MOVE.L  #$F80000,2(A7)           ; Set dest address to seg #124  (20)
        RTE                              ; Goto F80000 in user domain    (20)


;
; ***** USER DOMAIN *****
;
; Note: The following lines will be in the jump table segment (ie. 124).
;
F80000: MOVE.L  A0,SDSEG2                ; Save A0                       (20)
        MOVE.L  SDSEG,A0                 ; Get PC from shared data seg   (16)
        MOVE.W  (A0),-(A7)               ; Get high word of opcode       (16)
        AND.W   #$0F00,(A7)+             ; Test for JSR                  (16)
        BNE.S   NOTJSR                   ; Branch Not taken for IUJSR    ( 8)

IUJSR:  PEA     4(A0)                    ; Push Return Address           (20)
        MOVE.L  (A0)+,-(A7)              ; Push address of procedure     (24)
```

```
IUJMP:   MOVE.L  SDSEG2,A0          ; Restore A0                  (16)
         RTS                        ; Goto the procedure          (16)

NOTJSR:  MOVE.L  (A0)+,-(A7)        ; Push the opcode and offset
         BTST    #3,(A7)            ; Test for JMP
         BEQ.S   IUJMP              ; Branch taken for IUJMP
         BTST    #2,(A7)            ; Test for LEA
         BEQ.S   IULEA              ; Branch taken for IULEA

IUPEA:   CLR.B   (A7)               ; Zap high byte
         MOVE.L  A0,-(A7)           ; Push address to continue execution
         MOVE.L  SDSEG2,A0          ; Restore A0
         RTS                        ; Continue execution

IULEA:   TST.L   (A7)+              ; Discard the opcode and offset, Ugh!
         MOVE.L  A0,-(A7)           ; Push address to continue execution
         MOVE.L  -(A0),A0           ; Get the effective address in A0
         ADD.L   A0,A0              ; Shift left to make offset right
         MOVE.L  A0,-(A7)           ; Push the effective address
         CLR.B   (A7)               ; Zap high byte
         EXG     A0,D0              ; Save D0 in A0, Move eff addr into D0
         SWAP    D0                 ; Get high word of opcode
         ADD.W   D0,D0              ; shift left to get Ai in bits 11..9
         AND.W   #$0E00,D0          ; extract Ai in bits 11..9
         BNE.S   NOTA0              ; Branch taken for cases 1..6 only
         EXG     A0,D0              ; Restore D0 for case 0
         MOVE.L  (A7)+,A0           ; LEA xxxxxx,A0
         RTS                        ; Continue execution
NOTA0:   MOVE.W  #$4E75,-(A7)       ; RTS
         OR.W    #$2040,D0          ; MOVE.L D0,Ai
         MOVE.W  D0,-(A7)           ; Can't execute it from D0 so push it
         MOVE.L  4(A7),D0           ; Get the effective address in D0
         PEA     RETURN             ; Setup address for pushed RTS
         JMP     4(A7)              ; Go execute LEA xxxxxx,Ai
RETURN:  ADD.W   #8,A7              ; Deletes the subr & eff addr
         EXG     A0,D0              ; Restore D0 from A0 for cases 1..6 only
         MOVE.L  SDSEG2,A0          ; Restore A0
         RTS                        ; Continue execution

         .END
```

THE OLD JUMP TABLE

The format of the Old Jump Table was:

```
          +--------------------+
    $$TOP | Number of segments |  2 bytes
          +--------------------+
          | Main Segment Table | 32 bytes
          +--------------------+
          | Segment Table #2   | 32 bytes
          +--------------------+
          |       . . .        |
          +--------------------+
          | Segment Table #n   | 32 bytes
          +--------------------+
          | Dummy Table #n+1   |  4 bytes
          +--------------------+
          | $_START Descriptor | 10 bytes
          +--------------------+
          | S#1 P#2 Descriptor |
          +--------------------+
          |      . . .         |
          +--------------------+
          | S#1 P#n Descriptor |
          +--------------------+
          | S#2 P#1 Descriptor |
          +--------------------+
          |      . . .         |
          | S#2 P#n Descriptor |
          +--------------------+
          | S#3 P#1 Descriptor |
          +--------------------+
          |      . . .         |
          +--------------------+
          | S#m P#n Descriptor | 10 bytes
          +--------------------+
      -20 | Addr. of REMOVE1   |  4 bytes
          +--------------------+
      -16 | Addr. of buffer    |  4 bytes
          +--------------------+
      -12 | Addr. of code file |  4 bytes
          +--------------------+
       -8 | Active segment lst |  4 bytes
          +--------------------+
       -4 | Addr. of $$TOP     |  4 bytes
          +--------------------+
 $$LOADIT |    Object code     |
          | necessary to load  |
          |   and execute a    |
          |      segment       |
          +--------------------+
```

A segment table consisted of eight 32-bit values. They were:

```
+------+------+------+------+
| Address of 1st descriptor |  0 -- 3
+------+------+------+------+
|  File Address of Segment  |  4 -- 7
+------+------+------+------+
|  Size of code in bytes    |  8 -- 11
+------+------+------+------+
| Actual address in memory  | 12 -- 15
+------+------+------+------+
| Scratch return address    | 16 -- 19
+------+------+------+------+
| Segment reference count   | 20 -- 23
+------+------+------+------+
| Active segment-list link  | 24 -- 27
+------+------+------+------+
|     --- Reserved ---      | 28 -- 31
+------+------+------+------+
```

A descriptor was in one of two states, depending on the whether the segment was present in memory. These states were:

```
   Segment Not in Memory            Segment In Memory

+------------+------------+     +------------+------------+
| Relative offset of this |     | Relative offset of this |
+---                   ---+     +---                   ---+
|  entry in its segment   |     |  entry in its segment   |
+------------+------------+     +------------+------------+
|        JSR xxx.L        |     |        JMP xxx.L        |
+------------+------------+     +------------+------------+
|   Absolute address of   |     |  Absolute address of    |
+---                   ---+     +---                   ---+
|        $$LOADIT         |     |  procedure as loaded    |
+------------+------------+     +------------+------------+
```

A segment was loaded into memory when the first call to one of its procedures was executed. Such a call was always by way of a descriptor in the jump table. The JSR to $$LOADIT executed the loader from its entry point "$$LOADIT". The loader could tell which segment to load by comparing the place that it was called from with the limits on the segment entry tables found at the top of the jump table. The loader then loaded that segment, fixed up all JSR's to jump directly to the procedure instead of calling the loader, saved the calling routine's return address in the segment entry, patched the return address on the stack to return through the un-loader entry point "$$UNLOAD", and jumped to the procedure desired in the first place.