Silicon Graphics

subject: **MIPS Mdebug Debugging Information**

date: **March 7, 1996**

from: **David Anderson**

1. Disclaimer

This document describes the situation as known to the author(s). Omissions and errors are likely and changes to the "mdebug" format, while not frequent, do occur and readers need to be aware that this document can contain obsolete or inaccurate information. We have attempted to make it accurate, however.

2. Overview

The symbol table was once described in the "MIPS Assembly Language Programmers Guide" chapter named "The Symbol Table".

Unfortunately, that document is only in print now an edition which leaves out all the information on this old debugging-information format (using IRIS Insight, see the "MIPSpro Assembly Language Programmers Guide"). In IRIX4 days, this information was "the symbol table". However, now that ELF is the object format, this information, when present, is only one of multiple symbol tables. For example, the ELF symbol table has information needed by the linker, and the dynsym symbol table has information used by the dynamic linker (run-time linker, rld(1)). Consequently, the symbol table information we describe here is now termed the "mdebug" symbol table (or "mdebug" for short), after it's section name, ".mdebug".

Because mdebug was used for both debugging and linking in IRIX3, IRIX4 and IRIX 5.[123], it is closely tied to all things requiring linker information (like the addresses of global symbols).

The header C structs mentioned here are two views (generally two views) of the same data. For example, HDRR and CHDRR are representing the same data in different contexts. On disk, HDRR is relevant. In memory (after being read in by libmld) the CHDRR (pointer-based) version is relevant. I find it much easier to understand the disk-based version. In addition, since libmld routines leak memory and are poorly documented, I consider it good practice to work with the disk based version and avoid libmld if you must work with the details of the symbol table.

DeltaC++ introduces a whole host of new things. These are not discussed here.

The data types like 'long' in the structures here are all 32 bits at most. This debug information is not used for 64bit object files. The user's object may have 'long long' data (64-bit integer) though.

An important point is that the LOGICAL structure of the file is as follows:

```
HDRR
 FDR (file descriptor, referenced via an ifd).
   local strings
   local symbols
   local procedures
   local file indirect table
   local auxes
   local line entries

 FDR
 ...
```

external strings
externals

On the other hand, the physical structure is quite different.
It is
HDRR
all procedure
all local symbols
all auxes
all local strings
all external strings
all FDRs
all file indirects
all externals

all line tables somewhere.
(in IRIX5.3 the line table is off somewhere odd after linking,
and is not in the elf section. This is a bug in 5.3 ld).

A correctly structured program which wishes to process all symbol data will proceed in the following way:

```
For each FDR
    Process each Local Symbol
    Process each PDR in this FDR
end for
For each EXTR
    Process the EXTR
end for
```

Local symbols and PDRs can only be properly understood in the context of the relevant FDR.

Below I give one psuedo-code example of how to get to local strings for a given local symbol. To give this its complete context, I show a) a C sample program, b) the compilation command, c) the output of stdump from the compilation, and d) the calculation to get local strings.

Optimization symbols do not, in fact, exist, so stdump always prints an empty "Opts:" section.

Source For file t.c:

```
int main()
{
        char buff[100];

}
```

compiled with cc -g -c t.c the command `stdump t.o` gives, in IRIX6.2 (in earlier releases not all the fields are printed so less data is shown):

```
mdebug HDRR: magic 0x7009 vstamp 0x0602 (6.2)
  ilineMax 10 (0xa) cbLine 4 (0x4) cbLineOffset 424 (0x1a8)
  idnMax 8 (0x8) cbDnOffset 864 (0x360) (unused in object files)
  ipdMax 1 (0x1) cbPdOffset 428 (0x1ac) (PDRs)
  isymMax 7 (0x7) cbSymOffset 480 (0x1e0) (local symbols)
  ioptMax 0 (0x0) cbOptOffset 0 (0x0) (not used)
  iauxMax 23 (0x17) cbAuxOffset 564 (0x234) (AUX records)
```

```
  issMax 84 (0x54) cbSsOffset 656 (0x290) (local strings)
  issExtMax 20 (0x14) cbSsExtOffset 740 (0x2e4) (external strings)
  ifdMax 1 (0x1) cbFdOffset 760 (0x2f8) (FDRs)
  crfd 0 (0x0) cbRfdOffset 0 (0x0) (relative file descriptors)
  iextMax 2 (0x2) cbExtOffset 832 (0x340) (external symbols)


SYMBOLS TABLE:
FILE 0. unmergable, new, C (adr 0x0, csym 7):
        (rss 1 issBase 0 cbSs 81 isymBase 0 ilineBase 0
         cline 10 ipdFirst 0 cpd 1 iauxBase 0 caux 23
         rfdBase 0 crfd 0 ipdfM 0x0 cpdM 0x0 cbLineO 0 cbLine 2)

Binary of auxes:
  0. 0x02000000   0x03000000   0x04000000   0x05000000   0x06000000
  5. 0x07000000   0x08000000   0x09000000   0x20000000   0x21000000
 10. 0x0a000000   0x0b000000   0x0b000000   0x1a000000   0x00000000
 15. 0x00000006   0x06000000   0x03003000   0xfff00004   0x00000000
 20. 0x00000000   0x00000063   0x00000008


Local Symbols:
from file t.c           compiled -g2  Print aux if present
        compilation host:path quasar:/d2/cmplrs/mongoose/doc/mdebug/t.c
        compiled Thu Mar  7 15:07:53 1996
        file size was 41 bytes.
  0. ( 0)(   0) t.c        File       Text       symref 7
  1. ( 1)(   0) main       Proc       Text       [15] endref 6, int
  2. ( 2)(  16)            Block      Text       symref 5
  3. ( 3)(-100) buff       Local      Abs        [17] Array [(extended file 0, aux 4)0-99:8] of unsigned char
  4. ( 2)(  24)            End        Text       symref 2
  5. ( 1)(  40) main       End        Text       symref 1
  6. ( 0)(   0) t.c        End        Text       symref 0

File indirect table:


Opts:


Procs(1 entries, only those with a symbol print):
  0. (  1) main, csym 7, iline= 0,regmask=0x00000000, save-reg-offset 0
         fregmask 0x00000000, sav_flt_offset 0, low-line 2, hi-line 5
        framesize 104 addr 0 (0x0) frame-reg $r29 pcreg $r31 isym 1 LnOff 0


Lines:
      0.   2      1.   2      2.   2      3.   2
  4.   5      5.   5      6.   5      7.   5
  8.   5      9.   5


Externals table:
0. (file  0) (   0) main       Proc       Text       symref 1
1. (file  0) (   0) _gp_disp   Global     Undefined  indexNil
```

```
Dense number table:
2. file 0, symbol 0
3. external symbol 0
4. file 0, symbol 2
5. file 0, symbol 4
6. file 0, symbol 5
7. external symbol 1
```

The following illustrates getting the function name ("main"), which is local symbol 1 in FILE 0. The illustration is sketchy but will, I hope, help.

```
FDR is at HDRR.cbFdOffset + (sizeof struct FDR)*0

SYM is at HDRR.cbSymOffset    + FDR.isymbase + 1*(sizeof struct SYMR)

The string is at disk location:
HDRR.cbSsOffset +   FDR.issbase  + SYM.iss
```

3. The mdebug overall header: CHDRR pCHDRR

This begins the .mdebug section and provides access to all the parts of the .mdebug data. The C struct declarations are given in `Courier` type.

As stated above, this is the liblmd run-time version of the header, which we shall not explain.

```
typedef struct {
      long  fappend;    /* are we appending to this beast ? */
      pCFDR pcfd;       /* the compile time file descriptors */
            /* pcfd points to array of CFDR records, via index
               called an ifd   pcfd[ifd] for example
            */
      pFDR  pfd;        /* all of the file descriptors in this cu */
      long  cfd;        /* count of file descriptors */
      long  cfdMax;           /* max file descriptors */
      pSYMR psym;       /* the symbols for this cu */
      pEXTR pext;       /* externals for this cu */
            /* points to the array of external symbols
             they run in the external symbol table
            0 thru cext-1
            */
      long  cext;       /* number of externals */
            /* they run in the external symbol table
            0 thru cext-1
            cext is number actually used
            */
      long  cextMax;    /* max currently allowed */
      char  *pssext;    /* string space for externals */
            /* For external symbols. pssext indexed by iss of
               SYMR iss fieldrecord.
               This is strings for externals for all
                the files. indices into here via  iss
               are obviously global index, not per file or cu
```

```
               when read in, read hdr.issExtMax bytes.
            */
      long  cbssext;    /* # of bytes in ss */
      long  cbssextMax; /* # of bytes allowed in ss */
      pAUXU paux;       /* auxiliaries for this cu */
      char  *pss;       /* string space for this cu */
            /* When read into pss, read hdr.issMax bytes
            Not indexed into directly, Indexed thru CFDR->pss
            */
      pDNR  pdn;        /* dense number table for this cu */
      long  cdn;        /* number of dn's */
      long  cdnMax;           /* max currently allowed */
      pOPTR popt;       /* optimization table for this cu */
            popt Not accessed directly, indexed thru CFDR->popt
      pLINER      pline;            /* lines for this cu */
      pRFDT prfd;       /* file indirect for this cu */
            prfd Not accessed directly, indexed thru CFDR->prfd
      pPDR  ppd;        /* procedure descriptor tables */
            ppd Not accessed directly, indexed thru CFDR->ppd
      int   flags;            /* which has been read in already */
      int   fswap;            /* do tables need to be swapped */
      HDRR  hdr;        /* header from disk */
      } CHDRR, *pCHDRR;
#define cbCHDRR sizeof (CHDRR)
#define chdrNil ((pCHDRR) 0)
#define ichdrNil -1
#endif LANGUAGE_C
```

4. The mdebug file header: HDRR pHDRR

This is the mdebug file header as found on disk.

Note that the HDRR offsets are offsets from the beginning of the a.out, not from the beginning of a section. This is a "design mistake," but a very old one. It was, in fact, part of the design that was purchased from Third Eye Software.

```
typedef struct {
      short magic;            /* to verify validity of the table */
      short vstamp;           /* version stamp */
      long  ilineMax;   /* number of line number entries */
      long  cbLine;           /* number of bytes for line number entries */
      long  cbLineOffset;     /* offset to start of line number entries*/
      long  idnMax;           /* max index into dense number table */
      long  cbDnOffset; /* offset to start dense number table */
      long  ipdMax;           /* number of procedures */
      long  cbPdOffset; /* offset to procedure descriptor table */
      long  isymMax;    /* number of local symbols */
      long  cbSymOffset;      /* offset to start of local symbols*/
      long  ioptMax;    /* max index into optimization symbol entries */
      long  cbOptOffset;      /* offset to optimization symbol entries */
      long  iauxMax;    /* number of auxiliary symbol entries */
      long  cbAuxOffset;      /* offset to start of auxiliary symbol entries*/
      long  issMax;           /* max index into local strings */
      long  cbSsOffset; /* offset to start of local strings */
      long  issExtMax;  /* max index into external strings */
      long  cbSsExtOffset;    /* offset to start of external strings */
      long  ifdMax;           /* number of file descriptor entries */
            /* ifd's for the file run from 0 to ifdMax-1 */
            /* Each one is an index to a CFDR record
               indexed off of PCHDRR pointer
            */
      long  cbFdOffset; /* offset to file descriptor table */
      long  crfd;       /* number of relative file descriptor entries */
      long  cbRfdOffset;      /* offset to relative file descriptor table */
      long  iextMax;    /* max index into external symbols */
      long  cbExtOffset;      /* offset to start of external symbol entries*/
      /* If you add machine dependent fields, add them here */
      } HDRR, *pHDRR;
#define cbHDRR sizeof(HDRR)
#define hdrNil ((pHDRR)0)
```

There are 3 entries here relating to lines, ilineMax, cbLine, and cbLineOffset because the line table is in a compressed format. ilineMax gives the number of line numbers after expansion to "one line number per instruction", while cbLine gives the number of bytes in the compressed table representing the numbers.

5. The In Memory File Header: CFDR pCFDR

This is the libmld form, not the disk form.

```
#ifdef LANGUAGE_C
AUXU _auxtemp;
#define AUXINT(c) ((_auxtemp.isym = c), _auxtemp)

typedef struct {
      pFDR  pfd;        /* file descriptor for this file */
      pSYMR psym;       /* symbols for this file */
            /* an array of symbols, indexed by symbol numbers
               from 0 to pfd->csym.
            */
```

```
      long  csymMax;    /* max allocated */
            /*  Not all actually used. See psym on csym, prev line.
            */
      pAUXU paux;       /* auxiliaries for this file */
            /* local auxes indexed starting with SYMR->index
            where applicable
            */
      long  cauxMax;    /* max allocated */
      char  *pss;       /* strings space for this file */
            /*  local strings, not external strings. Indexed
                by iss of SYMR struct
            */
      long  cbssMax;    /* max bytes allowed in ss */
      pOPTR popt;       /* optimization table for this file */
            points to array of optimization stuff.
            indexed 0 to  pfd->copt
            stprint.c says "not done yet"
      long  coptMax;    /* max allocated */
      pLINER      pline;            /* lines for this file */
            /* an array of line numbers, one for each
            instruction location in the program. (ie
            one for each 4 bytes of code
            */
      long  clineMax;   /* max allocated */
      pRFDT prfd;       /* file indirect for this file */
            /* points to array of file indirects,
            indexed 0 to  pfd->crfd
            Each entry is simply a long.
            But what for???
            */
      long  crfdMax;    /* max allocated */
      pPDR  ppd;        /* procedure descriptor tables */
            points to array of procedures.,
            indexed 0 to pfd->cpd
      long  cpdMax;           /* max allocated */
      long  freadin;    /* set if read in */
      } CFDR, *pCFDR;
#define cbCFDR sizeof (CFDR)
#define cfdNil ((pCFDR) 0)
#define icfdNil -1
```

6. The on disk File Header: FDR pFDR

There is one of these for each file used in compilation. Each compile generates at least one. There may also be FDRs generated by compilers. For example, the Fortran compiler generates its own 'file' entries for each common.

```
typedef struct fdr {
      unsigned long     adr;  /* memory address of beginning of file */
            adr is incorrect if there are no PDR's for this FDR.

      long  rss;        /* file name (of source, if known) */
      long  issBase;    /* Offset with local strings
                        of the beginning of strings for
```

```
                  this FDR */
long  cbSs;       /* number of bytes in the ss */
long  isymBase;   /* beginning of symbols */
long  csym;       /* count file's of symbols */
long  ilineBase;  Index into virtual array of line
                  table entries (array is size of text
                  for this fdr).  Array does not exist
                  on disk. Array created by libmld
                  by expanding compressed line data
                  virtual table has a line number (integer)
                  per instruction in text.
                  virtual entry is 32-bit integer
long  cline;            Size of the the virtual array indexed
                  by ilineBase. Count of number of 32 bit
                  words

                  ilineBase + cline of this FDR are the
                  ilineBase of the next FDR
                  that has a line table (ie, has text).
long  ioptBase;   /* file's optimization entries */
long  copt;       /* count of file's optimization entries */
unsigned short    ipdFirst; /* start of procedures for this file */
unsigned short    cpd;    /* count of procedures for this file */
long  iauxBase;   /* file's auxiliary entries */
long  caux;       /* count of file's auxiliary entries */
long  rfdBase;    /* index into the file indirect table */
long  crfd;       /* count file indirect entries */
unsigned lang: 5; /* language for this file */
      /* language numbers assigned "permanently"
      */
unsigned fMerge : 1;    /* whether this file can be merged */
      /* fmerge shows as "mergeable" or "unmergeable"
         for the file via st_printfd() in stdump
      */
unsigned fReadin : 1;   /* true if it was read in (not just created) */
      /* Readin shows as "preexisting" if readin set, else
            " new" via st_printfd() in stdump
      */
unsigned fBigendian : 1;/* if set, was compiled on big endian machine */
                  /*    auxes will be in compile hosts sex */
unsigned glevel : 2;    /* level this file was compiled with */
  unsigned signedchar : 1; /* whether files was compiled with char being signed */

/* the following were added very late in IRIX 5.2 to allow
     greater than 64K procedures */
  unsigned ipdFirstMSBits: 4; /* upper bits to allow  ipdFirst to
                   exceed 64K entries
                   (These are the most significant bits of what is,
                   after concatenating the bits, a 20 bit number) */
  unsigned cpdMSBits: 4;  /* upper bits to allow cpd to exceed 64K
                   entries
                   (These are the most significant bits of what is,
                   after concatenating the bits, a 20 bit number) */
  unsigned reserved : 13;  /* reserved for future use */
```

```
      long  cbLineOffset;     /* byte offset from header for this file ln's */
                        This is offset of the compressed line table
                        for this fdr (within the physical
                        compressed line table).
                        Used by the debugger as a way to start at
                        the beginning of
                        information for a procedure.
                        The cbLineOffset of the next FDR
                        cbLineOffset can
                        be a line table stopping rule.
                          (thus, cbLineOffset is non-decreasing).

      long  cbLine;           /* size of lines for this file */
                          Size in bytes of the compressed line table
                        of this fdr.

                        cbLIneOffset + cbLine for this fdr
                        are the cbLineOffset of the next FDR
                        that has a line table (ie, has text).
      } FDR, *pFDR;
#define cbFDR sizeof(FDR)
#define fdNil ((pFDR)0)
#define ifdNil -1
#define ifdTemp 0
#define ilnNil -1
#define __sgi_mld_set_cpd(__fdr,__val)          ((__fdr)->cpd = __val ,  (__fdr)->cpdMSBits = (__val >>16))
#define __sgi_mld_get_cpd(__fdr)                 ((__fdr)->cpd | ((__fdr)->cpdMSBits<<16))
#define __sgi_mld_set_ipdFirst(__fdr,__val)          ((__fdr)->ipdFirst = __val ,  (__fdr)->ipdFirstMSBits = (__val >>16))
#define __sgi_mld_get_ipdFirst(__fdr)                 ((__fdr)->ipdFirst | ((__fdr)->ipdFirstMSBits<<16))
```

The __sgi_mld_set_cpd (etc) macros above are used to add extra bits onto the to-small ipdFirst and cpd fields of the structure. Before IRIX5.3, the extra bits were not reliably set to 0, so cautious programs will ignore the macros unless a human tells the program to pay attention to the extra bits. The cpd and ipdFirst fields were also, for releases before IRIX5.3, "signed short" so the maximum value was 32,767. For these releases, the extra bits are not set correctly so the macros must not be used.

The changes allowed, first, increasing the limit to 66635 for ipdFirst and cpd, and second (with the bit fields) increasing the limit to 1048575.

7. The procedure descriptor record: PDR pPDR

This structure exists only in this one form: there is no separate run-time form used by libmld.

```
typedef struct pdr {
      unsigned long     adr;  /* memory address of start of procedure */
            if isym is isymNil, adr is unused.
            if FDR->csym != 0
                adr is offset from beginning of
                the file stFile address.
                By 5.3 we normally set stFile 0, so adr is the
                absolute address.
            else
                adr is absolute address of the procedure.

      long  isym;       /* start of local symbol entries */
            if isym is isymNil, just there is no procedure entry here really
            else if FDR->csym != 0
                  this is offset in local symbol table of symbol
            else
                  is offset in external symbols table
      long  iline;            /* start of line number entries*/
            iline gives the instruction number (within the FDR,s
            instructions, if an FDR can be said to have such) of the
            first instruction for this procedure.  In effect, an index
            into an expanded line table with a line number per instruction.
      long  regmask;    /* save register mask */
      long  regoffset;  /* save register offset */
      long  iopt;       /* start of optimization symbol entries*/
      long  fregmask;   /* save floating point register mask */
      long  fregoffset; /* save floating point register offset */
      long  frameoffset;      /* frame size */
      short framereg;   /* frame pointer register */
      short pcreg;            /* offset or reg of return pc */
      long  lnLow;            /* lowest source line number in the procedure*/
      long  lnHigh;           /* highest source line in the procedure */
            For Fortran alternate entries, lnHigh is -1.
      long  cbLineOffset;     /* byte offset for this procedure
                  from the fd base */
            cbLineOffset gives a byte offset within the compressed
            line table for this FDR of the first byte of line info
            for this procedure.
            cbLineOffset is, within a given FILE, nondecreasing
            and, by definition, the 0th procedure in a given FILE
            has cbLineOffset 0.
      } PDR, *pPDR;
#define cbPDR sizeof(PDR)
#define pdNil ((pPDR) 0)
#define ipdNil    -1
```

Note carefully that depending on whether FDR->csym is 0 or not the meaning of adr and isym change.

8. The Line Table

This first set of declarations table represents the expanded-by-libmld in-memory form of the line table, not the on-disk form. The array as expanded is in a one-integer-per-instruction form, so it is the same size as the program text.

```
typedef      long LINER, *pLINER;
#define lineNil ((pLINER)0)
#define cbLINER sizeof(LINER)
#define ilineNil  -1
```

The above is the expanded form. Now we define the compressed form. It is a byte stream. There are two entry formats, each of which we attempt to describe with a pseudo-C declaration. The declarations are pseudo-C since there are no declarations used or available in headers. There is a short format and a long format. Begin with the short format, a 1 byte entry.

short format:
  signed   int delta:4;
  unsigned int count:4;

delta is a 4bit value in the range -7..7, defining the number of source lines between the current source line, and the previous line generating executable instructions.

count is a 4bit field with a value in the range 0..15 indicating the number (1..16) of executable instructions associated with a source line. If more than 16 instructions (15+1) are associated with a source line, new line numbers are generated with delta = 0.

The long format takes 3 bytes per entry.
long format:
  signed int constval :4; /* value 0x800 */
  unsigned int count   :4;
  unsigned int updelta :8;
  unsigned int lowdelta:8;

Since delta is a 16-bit integer (signed) it allows a maximum of 32676 source lines (of all kinds) between two source lines that produce executable instructions.

```
#include <stdio.h>
main()
{
  int c;

  printf("this program just prints its input0); /* line 6 */
  while(( c= getc(stdin)) != EOF) { /* line 7 */
    /*
    * 1
    * 2
    * 3
    * 4
    * 5
    * 6
    * 7
    */
    printf("%c",c); /* line 17 */
  } /* end while */ /* line 18 */
}  /* line 19 */
```

Now, as an example, we show output part of the object code that a compiler might produce.

```
func:line
        inst addr
main:6, 0x4001b0 addiu...
main:7, 0x4001b4 lw...
main:7, 0x400200 nop
main:17, 0x400204
main:17, 0x400208
main:17, 0x40020c
main:18, 0x400210
main:18, 0x40025c
main:19, 0x400260 ...
```

Now, as an example, we produce output that some version of the C compiler might produce.

```
Source      contents    meaning
Line    (hex)           (decimal)
(decimal)
3     02          delta 0, count 2
6     31          delta 3, count 1
7     1f          delta 1 count 15
7     03          delta 0 count 3
17    82 00 0a    extended format:count 2 delta 10
18    1f          delta 1 count 15
18    03          continuation: delta 0, count 3
19    15          delta 1 count 5
```

9. The symbol structure: SYMR pSYMR

This is the structure used for all local symbols. In addition, this structure is also a part of all external symbols (see the ’asym’ field of the EXTR structure).

We start with the actual structure fields, concisely shown.

```
typedef struct {
      long  iss;        /* index into String Space of name */
      long  value;            /* value of symbol */
      unsigned st : 6;  /* symbol type */
      unsigned sc  : 5; /* storage class - text, data, etc */
      unsigned reserved : 1;  /* reserved */
      unsigned index : 20;    /* index into sym/aux table */
      } SYMR, *pSYMR;
```

And follow with that same structure, but now with explanations expanded in the declaration.

```
typedef struct {
      long  iss;        /* index into String Space of name */
            If from external symbol table, index into
            external string table.
            If from local symbol table, index into
            local string table via
                  FDR->issBase + iss

      long  value;            /* value of symbol */
            See below for the interpretation of ’value’.
      unsigned st : 6;  /* symbol type */

            "stNil",          /*  stNil   0 */
                  index is index into aux table
                  sc Irrelevant.
                  Value  Irrelevant.
                  This is/was a symbol created by the
                  compiler that is a never-used declaration.
                  ld -r removes it. The symbol should be ignored.
            "Global",         /*  stGlobal      1 */
                  sc is large/small data/bss (scData scBss)
                  index is index into aux table
                  value is the address of the item
            "Static",         /*  stStatic      2 */
                  sc is large/small data/bss (scData scBss)
                  index is index into aux table
                  value is the address of the item
            "Param",          /*  stParam 3 */
                  index is index into aux table
                  scAbs:
                    value is Frame Offset
                    (offset from virtual frame pointer)
                  scRegister: value is register number
                  scVar: value is Frame Offset
                    (offset from virtual frame pointer)
                  scVarRegister: value is register number
```

```
"Local",          /*  stLocal 4 */
      index is index into aux table
      scAbs: value is Frame Offset
        (offset from virtual frame pointer)
      scRegister: value is register number
"Label",          /*  stLabel 5 */
      scText is the storage class.
      printed as "symref",
      index  meaning:
        Index of symbol of block label refers to ?
        According to doc, index is indexNil
        (ie not used)
      value is the address of the symbol.
"Proc",                 /*  stProc  6 */
      Is a block
      if external-symbol and sc != scUndefined
            index prints as symref
            (symref: local symbol # in file indicated
            by ifd in EXTR)
      else
            if index == 0xfffff prints indexNil, done.
            else
            index is index into aux table
      In AUX:  first record is isym (symbol table index
            of symbol after the Proc end)
            Next is function type
      scText:
         value is address
      scNil:
         value is address
      scUndefined:
         value is address
"Struct"                  /* stStruct 26, sgi only  */
"Union"                   /* stUnion 27, sgi only  */
"Enum"                    /* stEnum 28, sgi only  */
      The above 3 are just like stBlock, but positively
      identify the type of block.
"Block",          /*  stBlock       7 */
      Only occurs in local sym table, not Externals
            table.
      This is any one of a number of things.
      printed as "symref",
      index  meaning: index of symbol after end this block
      scInfo:
            If enum: value is max enum value
            If struct/union, value is size of
              the struct/union.
            If variant arm, value is the iaux of
             the ranges of the variant
             arm. One knows it is variant arm
             because it is a stBlock 'owned'
             by an stVariant entry. That is,
             an stBlock and its matching
             stEnd owns the top-level entries
             between them.
```

```
        scText:a text block.
              value is relative address
              from the beginning of the procedure.
        scComon: Common block.
              Value is the size of the block
        scVariant: Variant record (Pascal).
              Value is the symbol number of the symbol
              that is the tag for the variant
  "End",                  /*  stEnd   8 */
        stEnd represents end of block index
        printed as "symref",
        index  meaning: index of symbol beginning this block
        scInfo:
              if Enum end, value is 0
              if struct end, value is 0
              if union end, value is 0
              if variant arm end, value is 0
        scText:
              Whether file, procedure, or text block,
              is being ended, value
              is the relative displacement from the
              beginning of the procedure.
        scCommon:
              value is 0
        scVariant:
              value is 0

  "Member",         /*  stMember      9 */
        Can be enumeration, struct, or union member.
        scInfo is storage class.
        If struct/union, value is bit offset
                from beginning of the struct/union
          index is index into aux table entry
          describing the member type and size (a /btBlocknormal
          aux entry).
        If enum, value is ordinal value of enum
          index is indexNil
  "Typdef",         /*  stTypedef     10 */
        index is index into aux table
        scInfo is class.
        value is 0.
  "File",                 /*  stFile  11 */
        treated as a block.
        printed as "symref",
        Storage class is scText.
        index  meaning: index of symbol after end this block
        value: address of lowest-address procedure in the
          file. This is unreliable with -O3 and with corded
          files.  And perhaps with entire releases.
          Do not use this value.
  (char *)0,
  (char *)0,
  "StaticProc",           /*  stStaticProc 14 */
        Storage Class is scText.
        Is a kind of block...
```

```
            if external-symbol and sc != scUndefined
                  index prints as symref
                  (symref: local symbol # in file indicated
                  by ifd in EXTR)
            else
                  if index == 0xfffff prints indexNil, done.
                  else
                  index is index into aux table
            In AUX:  first record is isym (symbol table index
                  of symbol after the Proc end)
                  Next is function type
            value is address.
      "Constant",       /*  stConstant   15 */
            index is index into aux table
            Storage Class is ?????
            value is const value?????
        "compiler typedef"     /* stIndirect     33  */
            Storage Class is scInfo
            index is index into aux.
            This came as a result of bug #148436.
            Installed in symconst Nov, 1993.

              The stIndirect node is a placeholder. created
            when the compiler sees, for example,
                  struct opaque;
            or
                  ..
                  struct opaque *x;
            as the first mention of the struct tag.

            If the struct content is later declared, the
            stIndirect will point to the declaration stStruct/
            stUnion by having a type like
            struct(extended file 0, index 13) (btStruct/btUnion
            /btBlock/btEnum).

            If the struct content is never declared, the
            stIndirect will have type ’void’ (btVoid).
            Essentially, stTypedef is now reserved for use
            with the typedef keyword and stIndirect is
            used for struct tags mentioned before declared
            (or mentioned but never declared).


unsigned sc  : 5; /* storage class - text, data, etc */
      "scNil",          /* scNil    0 */
      "Text",                 /* scText   1 */
      "Data",                 /* scData   2 */
      "Bss",                  /* scBss    3 */
      "Register",       /* scRegister     4 */
      "Abs",                  /* scAbs    5 */
            For Abs, the ‘‘value’’ is an address.
      "Undefined",            /* scUndefined    6 */
      "CdbLocal",       /* scCdbLocal     7 */
      "Bits",                 /* scBits   8 */
```

```
          "CdbSystem",            /* scCdbSystem    9
                            scCdbSystem is unused.
          "Dbx"             /* scDbx        9
                Used by dbx to flag that a struct has already
                been seen since multiple symbols refer to
                one structure. Prevents duplicate copies of
                struct desription. Purely an in-memory thing.
                Not used on the disk copy.
                And as of IRIX5.3 dbx, not used at all.*/
          "RegImage",       /* scRegImage     10 */
          "Info",                 /* scInfo   11 */
          "UserStruct",           /* scUserStruct   12 */
          "SData",          /* scSdata  13 */
          "SBss",                 /* scSBss   14 */
          "RData",          /* scRdata  15 */
          "Var",                  /* scVar    16 */
          "Common",         /* scCommon 17 */
          "SCommon",        /* scSCommon      18 */
          "VarRegister",          /* scVarRegister19 */
          "Variant",        /* scVariant      20 */
          "SUndefined",           /* scSUndefined   21 */
          "Init"                  /* scInit   22 */

          "BasedVar"              /*scBasedVar    23
                            Fortran or PL/1 ptr based var */
          "XData"                 /*scXData       24
                            exception handling data */
          "PData"           /* scPData"     25
                            Procedure section */
          "Fini"                  /* scFini       26
                            .fini section */
          "NonGP"                 /* scNonGP      27
                            don't put this symbol in GP region */
          "Virtual"         /* scVirtual   scRegister
                            C++ virtual */
          "Pure"                  /* scPure      scRegImage
                             C++ pure */
          "Friend"          /* scFriend    scVarRegister
                            C++ friend */
          "Class"                 /* scClass     scPData
                            class member */
          "Static"                /* scStatic    scCommon
                            static function or base */
          "VirStatic"             /* scVirStatic scSCommon
                            virtual static base */

          "Max"             /* scMax        32 */
                            Only 5 bits in the field.


      unsigned reserved : 1;  /* reserved */
      unsigned index : 20;    /* index into sym/aux table */
            meaning depends on st etc, above
      } SYMR, *pSYMR;
#define symNil ((pSYMR)0)
```

```
#define cbSYMR sizeof(SYMR)
#define isymNil -1
#define indexNil 0xfffff
#define issNil -1
#define issNull 0
```

Before IRIX 4.0.5, Structs, Unions, and Enums were all simply stBlock and one could tell which by guessing (or by knowing which aux entry pointed to the local symbol (because the aux entry would be btStruct or btUnion or btEnum)). And all terminated with stEnd. As of IRIX4.0.5 this was changed and stStruct stUnion stEnum were added with the obvious meanings. They are defined just like stBlock but are more specific.

Nested structs are in the symbol table nested.

With old compilers (pre IRIX4.0.5) Structs can be differentiated from unions only in that the second and subsequent Member has a non-zero offset - and of course unions don't have bit fields. So a single-member struct cannot be distinguished from a single-member union. The aux information gives the data type and, if a bit field, the width. The ‘‘value’’ gives the offset.

With old compilers (pre IRIX4.0.5) Enums are distinguished in that the basic type is btNil. (It would seem more sensible to be btInt). The aux record seems to be unused (ie all bits zero), though present. The ‘‘value’’ gives the enum value.

Up through IRIX5.3 compilers, the index field continues to point to an aux record with all 0 bits.

With recent compilers, the stMember index may be indexNil in which case there is no aux record used.

10. Local String Table

As explained above, the local string table is physically one block of strings. Logically it is a separate table per FDR.

Within the local strings of an FDR there is a little bit of (perhaps surprising) structure.

First, the 0th byte of the local strings should be 0. Thus an iss or rss of 0 gives a 0-length string by indexing this 0 byte. Byte 1 of the local string table is generally the source file name indexed by the rss of the FDR and the iss of the first (stFile) local symbol.

Following this file name, there are 2 optional string sequences, each of which is optional. These are not indexed by anything. Debuggers simply know to look for them.

The first optional sequence begins with `</4Debug/>` followed by a set of strings (each null-terminated) (each string is documented on one line here)

hostname:full_path_name_of_source_file
compilation-date
file-length

hostname:full_path_name_of_source_file is the host name at compilation (perhaps 0-length) followed by a colon followed by the full-path-name of the source file.

compilation-date is a decimal representation of the 32-bit unix time returned by time(2) (as produced by printf %d).

file-length is a decimal representation of the number of bytes in the source file at the time of compilation (as produced by printf %d).

The second optional sequence begins with `</Command_Line/>` followed by a set of strings (each null-terminated) (each string is documented on one line here)

command
working_directory

command is the command-line given to the compiler at the time this source was compiled.

working_directory is the working-directory of the compilation.

11. External symbol: EXTR pEXTR

These exist in just one form: there is no separate struct for the in-memory version.

```
typedef struct {
      unsigned jmptbl:1;      /* symbol is a jump table entry for shlibs */
      unsigned cobol_main:1;  /* symbol is a cobol main procedure */
        unsigned weakext:1;     /* symbol is weak external */
        unsigned deltacplus:1;  /* symbol is delta C++ symbol */
        unsigned multiext:1;    /* symbol may be defined multiple times
                           delta c++ issue */

        unsigned reserved:11;   /* reserved for future use */
      short ifd;        /* where the iss and index fields point into */
            if ifd > 0, we set the current fd to this ifd
                  And use that as pcfd.
            else
                  use 0 as the index into pcfd table
                  Does this make sense?

                ifd Changed post-IRIX5.3 to:
            unsigned short ifd;
            if ifd != ifdNil,
                  ifd is the index of the FDR applicable
            else
                  means there is no FDR, though in the
                  source to the assembler, 0 is taken
                  to mean use index 0 (which might make
                  sense for the assembler somehow, but
                  not, I think in general).
      SYMR  asym;       /* symbol for the external */
      } EXTR, *pEXTR;
#define extNil ((pEXTR)0)
#define cbEXTR sizeof(EXTR)
```

For IRIX5.3 and earlier, `ifdNil` was -1. For IRIX5.3 and later, `ifdNil` is `0xffff`.

If the ifd which is referred to is an ifd with csym 0, then that IFD and that global symbol is the result of the operation of an ld -x command. In that case, there are no local strings for the IFD and no local symbols. And no aux information. So the SYMR iss refers to the global string table.

Note that ifd is limited because it is only a 16-bit field.

12. Type Information: TIR pTIR

This is one of the forms a 32-bit AUX record can take. See the AUXU description below for an overview.

```
typedef struct {
      unsigned fBitfield : 1; /* set if bit width is specified */
      unsigned continued : 1; /* indicates additional TQ info
                        in next AUX */
      unsigned bt  : 6; /* basic type */
                  basic types are listed below.
      unsigned tq4 : 4;
            "tqNil",          /* tqNil    0 */
            "Pointer to",           /* tqPtr    1 */
            "Function returning",   /* tqProc   2 */
            "Array",          /* tqArray 3 */
            "NON-mips",       /* tqFar    4 */
            "Volatile",       /* tqVol    5 */
      unsigned tq5 : 4;
      /* ---- 16 bit boundary ---- */
      unsigned tq0 : 4;
      unsigned tq1 : 4; /* 6 type qualifiers - tqPtr, etc. */
      unsigned tq2 : 4;
      unsigned tq3 : 4;
      } TIR, *pTIR;
#define cbTIR sizeof(TIR)
#define tiNil ((pTIR)0)
#define itqMax 6
```

The ’continued’ flag is supported in IRIX5 cc at least. The ’bt’ field is set to the type in each of the continued records.

13. Relative Symbol Record: RNDXR pRNDXR

This version of an AUX record is used to ’point to’ other records.

If rfd is ST_RFDESCAPE (0xfff) then the following AUX record is the true file (FDR) number (as a 32-bit integer).

If there is a File Indirect Table (and there normally is with files output by the linker), then this is an index into that table and that table gives the true FDR number. If there is no File Indirect Table (normal for files not processed by the linker), then the file number is the true FDR number.

The ’index’ field is an index to one of a number of tables depending on context. See the AUX description.

```
typedef struct {
      unsigned    rfd : 12;    /* index into the file indirect table */
      unsigned    index : 20; /* index int sym/aux/iss tables */
      } RNDXR, *pRNDXR;
#define cbRNDXR sizeof(RNDXR)
#define rndxNil ((pRNDXR)0)
```

14. Auxiliary Information: AUXU pAUXU

The aux records are a series of 32-bit records defined by the following union. The meaning of the various records is determined from context. A sketch of the processing logic is given below to try to explain the context-dependent meaning and contents.

```
/* a union of all possible info in the AUX universe */
typedef union {
      TIR   ti;         /* type information record */
      RNDXR rndx;       /* relative index into symbol table */
      long  dnLow;            /* low dimension */
      long  dnHigh;           /* high dimension */
      long  isym;       /* symbol table index (end of proc) */
      long  iss;        /* index into string space (not used) */
      long  width;            /* width for non-default sized struc fields */
      long  count;            /* count of ranges for variant arm */
      } AUXU, *pAUXU;
#define cbAUXU sizeof(AUXU)
#define auxNil ((pAUXU)0)
#define iauxNil -1
```

For C, the first aux entries in a file’s collection are one each of the simple basic types, though the list varies by language and release:

- btChar
- btUChar
- btShort
- btUShort
- btInt
- btUint
- btLong
- btULong

The following is the layout of aux record sequences. The layout is slightly complicated by the presence of ST_RFDESCAPEs (rfd of 0xfff) in the RNDX records, which simply mean that the rfd field was not big enough so the next aux record is the number that would have been in rfd of RNDXR had there been enough bits. The layout is also complicated by the fact that some items’ presence is dependent on the st type from the symbol and some on the TIR type.

14.1 stProc stStaticProc from local symbols

This is an exceptional case. When an stProc or stStaticProc is the "st" of a local symbol, this case applies.

```
isym, whose value is the index of the next local symbol record
after the stEnd of the stProc or stStaticProc.

RNDX
giving the rfd and local symbol index of the pointer itself.

Standard Aux Processing (giving the function type).
```

14.2 Standard Aux Processing

The following subsections define the standard AUX sequence. The first entry in each AUX sequence is a specified. An entry specified as an RNDX is either a single 32-bit entry, with rfd and an index (whose meaning is context dependent and specified below) or is ST_RFDESCAPE (0xfff) as the rfd with the index (whose meaning is context dependent and specified below) followed by an "isym" which is the rfd (in other words, 2 32-bit words).

14.2.1 btNil

```
btNil       0     /* undefined */
```

14.2.2 numeric types

```
btAdr32         1     /* 32-bit address */
btAdr       btAdr32     /* address - integer same size as pointer */
btChar          2     /* character */
btUChar         3     /* unsigned character */
btShort         4     /* short */
btUShort    5     /* unsigned short */
btInt32         6     /* 32-bit int */
btInt       btInt32     /* int */
btUInt32    7     /* 32-bit unsigned int */
btUInt          btUInt32 /* unsigned int */
btLong32    8     /* 32-bit long */
btLong          btLong32 /* long */
btULong32   9     /* 32-bit unsigned long */
btULong         btULong32 /* unsigned long */
btFloat         10    /* float (real) */
btDouble    11    /* Double (real) */
btComplex   18    /* fortran complex */
btDComplex  19    /* fortran double complex */
btVoid          26    /* void */
btLong64    30    /* 64-bit long */
btULong64   31    /* 64-bit unsigned long */
btLongLong64      32    /* 64-bit long long */
btLongLong  btLongLong64 /* long long */
btULongLong64     33    /* 64-bit unsigned long long */
btULongLong btULongLong64     /* unsigned long long */
btAdr64         34    /* 64-bit address */
btLongDouble      36    /* Long double */

TIR

[if fBitField
      width]

[tq processing]

[if continued  ... ]
```

14.2.3 User Defined Types, possibly with names

```
btStruct    12    /* Structure (Record) */
btUnion           13    /* Union (variant) */
btEnum            14    /* Enumerated */
btSet       17    /* pascal sets */
btTypedef       15      /* defined via a typedef */
btIndirect      20      /* forward or unnamed typedef */

TIR

RNDX gives FDR and symbol index of the local symbol
with the tag name and following that, up to stEnd, the
contents. For a btTypedef (and probably
btIndirect), the name in the local symbol
is the user-type-name.


[tq processing]

[if continued   ... ]
```

14.2.4 btRange

```
btRange           16    /* subrange of int */

TIR

[if fBitField
      width]

RNDX rfd is FDR index and index is aux index of the basic
type this range is based on. Note that I have observed
stdump print the incorrect index for a btRange.

dnLow

dnHigh

[tq processing]

[if continued ... ]
```

14.2.5 specialized types

```
btFixedDec  21    /* Fixed Decimal */
btFloatDec  22    /* Float Decimal */
btString    23    /* Varying Length Character String */
btBit       24    /* Aligned Bit String */
btPicture   25    /* Picture */
```

14.2.6 specialized types The following are not used in SGI C++.

```
btPtrMem    27    /* DEC C++:  Pointer to member */
btVptr            28    /* DEC C++:  Virtual Pointer */
btClass           29    /* DEC C++:  Class */
btPlainChar 35    /* Plain char for C++ */
```

14.2.7 btPtrBasedVar

```
btPtrBasedVar     37    /* Pointer Based Variable for Fortran */

We get to this aux from a symbol. Call it Symbol1.
This symbol is the user variable, not
the pointer-variable (it is the pointer variable that
the user must assign an address to).


TIR

RNDX rfd is a FDR index and index is a local symbol number.
Call this Symbol2. Symbol2 is special(see below).

Followed by Standard Aux Processing starting with
the TIR encountered after RNDX
This aux sequence will have at least one tqPtr by definition.
This is the user variable itself described here, but with
an extra tqPtr because it is really a pointer-to, not the user data
itself. This gives the type of Symbol1 with an extra 'pointer' on it.

By the way, the Fortran pointer name is not mentioned in this
mdebug aux record:
the Fortran pointer name is a symbol with type
btAdr which is assigned the
same location as the user name.

IF Symbol2 is stProc or stStaticProc
      Address of Symbol1 is Symbol1.value as
      a frame pointer offset. Add Symbol1.value
      to the current virtual frame pointer.
      This address points to a 32-bit pointer.
else IF Symbol2 is stStatic or stGlobal or stStaParam
      get the symbol1 name through the Symbol1 iss (and FDR)
      and look up the name in the externals table.
      This address points to a 32-bit pointer.
else
      Symbol1.value+Symbol2.value is the address.
      This address points to a 32-bit pointer.
```

```
The address of the user’s data is the contents of the 32-bits
that the ’this address’ points to.
These contents are only meaningful at runtime (after the
user has assigned something to the Fortran Pointer).
The type of the user data is the type of Symbol1 with
one ’tqPtr’ stripped off.
```

14.3 tq processing

This is the building of of the pointer/array information on types. For Functions, the TIR gives the returned type of the function.

If the information is representing "array of pointer to int" then tq5,4,3,2 are tqNil, tq1 is tqArray, tq0 is tqPtr.

```
For each of tq5, tq4,tq3,tq2,tq1,tq0
  case tqARRAY
     RNDX  of the type it is an array of. rfd is the local file,
         index is an aux index for that FDR’s aux records.
         So with a C 2-dimension array, the index and rfd will
         be the same (the symbol table is thus
         pretending that C actually has
         2 dimensional arrays from the point of view of these
         RNDX values (but not from the point of view of the
         high dimension below, where the dnLow, dnHigh,
         width is unique for each dimension as you would
         expect)).

     dnLow  low dimension (0 for C)

     dnHigh highest valid subscript

     width  width of an entry in bits
     continue
  case tqNil
        continue;
  case tqPtr
        Add * (pointer)
        continue
  case tqProc
        Function Type
        continue
  case tqVar
  case tqVol
  case TqRef
  case tqPtrMem
  case tqPtrMemFunc
        Needs to be documented.
        continue
end for each
```

If array and pointer designations are arbitrarily mixed, (as if char ****[3]****cp were legal C) then where would the continued aux be in relation to the array auxes? I’m not sure, but possibly the continued aux would follow the array describing auxes.

15. Optimization Symbols: OPTR pOPTR

These do not exist. Ignore these completely.

16. File Indirect entries: RFDT pRFDT

This table is used to make type definitions in a different file from the type use accessible in a way which does not force ld to update the aux information itself. Instead this table can be updated. Note that in many cases, the File Indirect Table refers one back to the same file one is currently looking at!

If there is a File Indirect table, the value in the table gives the file-table number of the real file.

Example: in aux, if the "extended file" from RNDXR is 12 and the array of RFDT entries is the arry "indirects" then true file number (used for pushfile to get data about the referenced symbol) is

```
indirects[12].
```

If there is no such table of indirects, then the index (12 in this example) is taken to mean real file (FDR) 12

```
typedef long RFDT, *pRFDT;
#define cbRFDT sizeof(RFDT)
#define rfdNil    -1
```

17. Handling exceptions: RPDR pRPDR

The following table has some of the same fields as the PDR, but this table is constructed by the linker on demand and is not in the mdebug section at all. See `/usr/include/exception.h`.

```
typedef struct runtime_pdr {
      unsigned long     adr;  /* memory address of start of procedure */
      long  regmask;    /* save register mask */
      long  regoffset;  /* save register offset */
      long  fregmask;   /* save floating point register mask */
      long  fregoffset; /* save floating point register offset */
      long  frameoffset;      /* frame size */
      short framereg;   /* frame pointer register */
      short pcreg;            /* offset or reg of return pc */
      long  irpss;            /* index into the runtime string table */
      long  reserved;
      struct exception_info *exception_info;/* pointer to exception array */
} RPDR, *pRPDR;
#define cbRPDR sizeof(RPDR)
#define rpdNil ((pRPDR) 0)
```

18. FIT pFIT (Really the same as RFDT)

The following table is not really a separate table and you should ignore it. It was used by ld internally.

```
typedef long FIT, *pFIT;
#define cbFIT     sizeof(FIT)
#define ifiNil    -1
#define fiNil     ((pFIT) 0)
```

19. DNR pDNR

Dense numbers are simply a method of assigning a single integer to a file-number and local-symbol pair. These are used in the ucode intermediate representation and are not meaningful in an object file. Consequently they can be ignored by programs interested in the mdebug information.

```
typedef struct {
      unsigned long     rfd;    /* index into the file table */
      unsigned long     index;      /* index int sym/aux/iss tables */
      } DNR, *pDNR;
#define cbDNR sizeof(DNR)
#define dnNil ((pDNR)0)
```