TASM/TLNK/TLIB Relocatable Record and File Formats

Version 91.1
6/15/91

Contents

Library File Overhead Record Structure

Introduction

All non-text files used by the Toolset share a common record structure. Each file consists of a series of these records appended together with no intervening space or separators. Not all file formats use all records; each of the three types of files uses a different mixture. The three basic file types are:

TASM output file/TLNK input file. This file encodes the relocatable information that TASM generates when assembling a Transputer assembly language source file. TLNK in turn reads this file to translate the relocatable form into a downloadable "object" file. The standard filename extension for these files is ".TRL" (**T**ransputer **R**e**L**ocatable).

TLNK output file/loader input file. This file encodes the object file produced by TLNK in a format suitable for use with the various loaders provided with the Toolset. Files in this format have instantiated load and stack addresses and are completely resolved with regard to intermodule linkages. The standard filename extension for these files is ".TLD" (**T**ransputer down**L**oa**D**).

TLIB output file/TLNK library file. This file is used to gather a number of ".TRL" format files into a "library" for use with TLNK. The TLIB librarian has many facilities for manipulating files in this format. The standard filename extension for these files is ".TLL" (**T**ransputers **L**ike **L**ibraries).

In addition to these standard formats, the TASM and TLNK programs use variants of these formats for internal temporary files. This document makes no claim of documenting such internal usage, however, the individual record descriptions contain some mention of which records are used (if not HOW they are used...).

The remainder of this document covers the file formats in two stages. First the individual records which make up the various files are discussed. Following that, the rules for record order within the files are covered.

Individual Record Descriptions

General Characteristics

1. A file is considered to be a stream of 8 bit "bytes". If this is not the case with a particular computer system, either a "protocol conversion" library must be used to make it appear that this is the case, or the various Transputer Toolset utilities must be modified to accomodate the differences.

2. All fields within records are "little-endian"; The lowest order byte is found first in the file, followed by the second lowest order byte, etc. Within a byte each bit is labeled from right to left, lowest order to highest order, numbered from #0 to #7.

3. Fields within a record are either considered to be signed or unsigned as appropriate. Signed fields are assumed to be in two's complement format. The size in bytes and sign status of each field is indicated in the descriptions.

4. Each record consists of a 1 byte header field which gives the record "type". The "type" is right justified in the low order bits and the rest of the header byte is zero. Following this, the remainder of the record consists of a fixed number of fields (which may be of variable length depending on the record type).

5. Records which are generated by Transputer instructions to TASM and are completely "bound" (the instruction "prefix" is completely determined), turn into T_DATA records. Those which aren't turn into some form of T_XXX_OP record. Within these records the basic opcode byte is encoded in the last field (named "opcode"). The 4 high order bits in this field determine which of the basic 16 functions is desired ("ldc", "stl"...). The 4 low order bits are normally zero with the following exceptions:

It is sometimes possible to simulate the operation of a particular instruction with another combination of instructions. An example is a "ldc" which is used to load 0x80000000 on a 32 bit Transputer. Using the normal prefix technique this instruction is 8 bytes long. It may be functionally replaced by a "mint" instruction which is two bytes long and executes faster. TASM can be instructed to allow this sort of optimization by the use of the ".ldc" pseudo-opcode instead of the normal "ldc" instruction. The effect this has is that the lowest order bit within the 4 low order bits is set to "1" indicating that functional replacements for the basic instruction are allowed (currently only used with the "ldc" opcode).

In a similar fashion, TASM can be instructed to generate runtime relocatable code by the use of the ".rel" pseudo-op. This is encoded by a "2" in the low order 4 bits of the "ldc" opcode which indicates that ".ldc" instructions which form address expressions must be "bound" by TLNK into a "ldc"/"ldpi" instruction sequence instead of whatever form generates the best code (indicated by a "1" as described above). Since only address expressions are affected by this, the only record type which will use this is the T_ADDR_OP type with "ldc" as the opcode.

6. Records which are generated by TASM in response to the 16 basic opcodes contain immediate data information. As mentioned, if the immediate data is completely known at assembly time TASM can "bind" the instruction and turn the record type into T_DATA. Those instruction which aren't known at assembly time are passed on to TLNK with some implied assumptions about instruction length. These assumptions are reflected in the "minimum length" field. This works as follows:

- The overall instruction length (counting prefix bytes), may be no shorter than the specified "minimum length" (in bytes).

- Within the above constraint, the resulting instruction may be no longer than necessary. This is required since TASM may have already used the fact that it knew what the instruction length was (even if not the actual instruction prefix string), in the construction of other instruction prefix strings whose values are dependent on this instruction.

Detailed Record Descriptions

In the following descriptions the record "type" (taken from the record header field), is used to label the record. Following that the record fields are shown in a tabular fashion (with the field size in bytes and signed/unsigned status listed above):

Record: 0 **Name:** T_RESERVED

```
1U
[0]
```

This record is reserved for future use by Logical Systems. It is used internally in TASM to convey assembly error information. As used in TASM the record format is:

```
1U   2U         1U         1U
[0]  [line #]  [length]  [error code]
```

Where "line #" is the source line the error appeared on, "length" is the default instruction length to assume for the instruction which generated the error, and "error code" is the ASCII character to place on the listing output file.

Record: 1 **Name:** T_REL_FILE

```
1U   1U
[1]  [processor type]
```

This record appears as the first record in ".TRL" format relocatable files. The "processor type" field is used to encode the CPU type for which the code was assembled:

"processor type"	Meaning
0	The CPU type is unknown (32 bit CPU).
1	The code is for a T400/T414/T425 CPU.
2	The code is for a T800/T801/T805 CPU.
3	The code is for a T212/T222/T225 CPU.

This record should only appear once at the beginning of ".TRL" files. It is also appears within ".TLL" files since libraries will contain the entire contents of one or more ".TRL" files (plus overhead records).

Record: 2 **Name:** `T_LIB_FILE`

```
1U   1U
[2]  [processor type]
```

This record appears as the first record in ".TLL" format library files. The "processor type" field is used to encode the CPU type for which ALL the code in the library was originally assembled:

"processor type"	Meaning
0	The CPU type is unknown (32 bit CPU).
1	The code is for a T400/T414/T425 CPU.
2	The code is for a T800/T801/T805 CPU.
3	The code is for a T212/T222/T225 CPU.

This record should only appear once at the beginning of ".TLL" files (it is used nowhere else).

Record: 3 **Name:** `T_LD_FILE`

```
1U   1U
[3]  [processor type]
```

This record appears as the first record in ".TLD" format download files. The "processor type" field is used to encode the CPU type for which the code was assembled and linked:

"processor type"	Meaning
0	The CPU type is unknown (32 bit CPU).
1	The code is for a T400/T414/T425 CPU.
2	The code is for a T800/T801/T805 CPU.
3	The code is for a T212/T222/T225 CPU.

This record should only appear once at the beginning of ".TLD" files (it is used nowhere else).

Record: 4 Name: T_SIZE

```
1U   4S              4S
[4]  [date/time]     [library subfile size]
```

This record is used in ".TLL" format libraries to indicate the size in bytes of a library sub-module (one of the ".TRL" format files which makes up a library). This record also records the last modification date and time of the file before it was part of the library. This date and time is displayed when a library is listed using TLIB and is also used to set the modification date and time of any file which is created as a copy of the corresponding library subfile. The "date/time" modification field is encoded as the number of seconds since 00:00:00 GMT, January 1, 1970. This is the normal date/time format provided by the "time()" "C" library function. The "library subfile size" is simply the size of the subfile in bytes (the subfile immediately follows this record in the library).

Record: 5 Name: T_EOF

```
1U   2U
[5]  [line #]
```

This record is the last record in ".TRL", ".TLL" and ".TLD" files. It also appears at the end of each library subfile in ".TLL" files. Any data which trails this "final" record in one of these files is ignored. The "line #" field records the line number of the source level statement which generated the T_EOF record. If the source code is assembly language which has been run through TASM, this line number reflects the line number of the ".END" pseudo-opcode.

Record: 6 **Name:** T_SYMBOL

```
1U   1U                    1U               VAR
[6]  [public/external]     [symbol length] [symbol name]
```

This record is used in the ".TRL" and ".TLL" files. It is used to hold the name of an external symbol. In ".TRL" files, one or more of these records follows the initial T_REL_FILE and they hold the external symbol names which are defined or referenced in the file. In ".TLL" files these symbols appear both within library subfiles (as described for ".TRL" files), and copies of those symbols which are "defined" in the subfile appear in a overhead structure for each library subfile. See the section on ".TLL" record order for more information about this overhead structure. The "public/external" field is set to a binary "1" if the symbol is public (a definition), binary "0" if the symbol is external (a reference). The "symbol length" field determines the length of the following "symbol name" field in bytes and ranges from 1 to 255 bytes. There is NO zero termination byte in the "symbol name" field and most ASCII characters may be used in a symbol name (except zero, although many control characters and other strange values will cause difficulty with linker maps and may not be possible to specify as an "entry point" symbol to TLNK).

Within a ".TRL" file, symbol names are referenced via a "symbol number". The symbol numbers start at 0 within each file, with any symbols which are present in T_SYMBOL records representing the low ordered numbers. Within the sequence of T_SYMBOL records, the first record in the file is equivalent to #0, the second T_SYMBOL represents #1, etc. Any symbol numbers referenced which are greater than that represented by T_SYMBOL records are defined to be references to "local" symbols. These references are defined by corresponding T_DEF records which represent the value of the corresponding symbol by implicit position within the file (the symbol number is mentioned explicitly in the T_DEF record). Another symbol definition mechanism is the T_SET record which allows for "constant" definitions (the symbol value and number are explicitly present).

While exporting of "local" symbols is unusual in an assembler, the Transputer encourages this activity by allowing all immediate data references to be variable length. If you want variable length external references you pay a price in terms of additional complexity!

Record: 7 Name: T_FILENAME

```
     1U   2U                   2U
     [7]  [previous line #]    [new line #]

(continued with)    1U                  VAR
                    [filename size]     [filename]
```

This record is present in ".TRL" and ".TLD" files and is used to indicate that the code and data following was taken from a different original source code file. Since ".TRL" files are present within ".TLL" files, this record also appears there. As most records contain an explicit line number, this record is only present when a different source code file is being used, not just for a line number change. Fields within this record contain the line number of the last source line read from the "old" file, the first source line read from the "new" file, and the name of the new source file (in the conventional format of a name length within the range of 0 to 255 ASCII characters, followed by the actual name string WITHOUT a zero byte terminator). A name length of 0 represents a NULL filename.

As used by TASM, one of these records is present JUST after the last T_SYMBOL record in the ".TRL" file. This filename represents the actual input filename that TASM used when reading the assembly language file (or a NULL filename if TASM is compressing the output file and removing debug information). Assuming TASM is retaining debug information, subsequent T_FILENAME records represent the various "#line" input statements which TASM processes WHICH contain an explicit filename. "#line" statements with just a line number are represented by changes in the line number fields within subsequent non-T_FILENAME records.

Record: 8 Name: T_MODULE

```
     1U   2U         1U
     [8]  [line #]   [module #]
```

This record is present in ".TRL" files (and thus ".TLL" files). It is NOT present in ".TLD" files! This record is used to represent the results of a ".mod" pseudo-opcode in the input file to TASM. It represents the selection of a new "load region" within the source file. As used by TASM, an initial T_MODULE is output which specifies a module number of 0 (in the "module #" field). Any actual ".mod" pseudo-opcodes will generate additional T_MODULE records. As expected, the "line #" field indicates the input source line number the ".mod" was encountered on (as updated via "#line" statements).

Record: 9 **Name:** T_ALIGN

```
1U   2U
[9]  [line #]
```

This record is present in ".TRL" (and thus ".TLL" files). It tells TLNK that the following code or data should be "word" aligned. The "line #" field gives the line number where the ".align" occurred (although these records are also generated implicitly by ".mod" changes and for other TASM purposes).

Record: 10 **Name:** T_DATA

```
1U   2U         2U                    VAR
[10] [line #]  [# of data bytes]   [data]
```

This record is present in ".TRL", ".TLL" and ".TLD" files. It holds the basic "object" data generated by code or data statements. As the instruction and data binding progresses from assembly (TASM), through link and locate (TLNK), most other relocatable records become T_DATA records. This record contains a "line #" field, followed by field defining the length of the data field (1-65535 bytes), and finally the variable length data byte field proper.

Record: 11 **Name:** T_REL_DATA

```
1U   2U         4S
[11] [line #]  [address]
```

This record is present in ".TRL" (and thus ".TLL" files). It is used to represent a word of data which is to be initialized to the relative offset between where the first byte of data will be located and the indicated memory address. This record is generated by TASM in response to statements such as:

```
.dw  @0x10              ;Offset between data start and address 0x10
```

Record: 12 **Name:** T_RELSYM_DATA

```
1U   2U         2U              4S
[12] [line #]  [symbol #]     [offset]
```

This record is present in ".TRL" (and thus ".TLL" files). It is used to represent a word of data which is to be initialized to the relative offset between where the first byte of data will be located and the defined value of the "symbol #" (the defined value will be the symbol's address for a simple label). The "offset" field is used to hold a constant offset to the relative offset between the data and the symbol definition. Some example TASM input statements:

```
.dw  @hello          ;Offset between data start and value of "hello"
.dw  @hello+22       ;Like above but with a constant "offset" of 22
```

Record: 13 **Name:** T_RELREL_DATA

```
     1U   2U        2U                  2U
     [13] [line #]  [left symbol #]     [right symbol #]

(continued with)    4S
                    [offset]
```

This record is present in ".TRL" (and thus ".TLL" files). It is used to represent a word of data which is to be initialized to the difference between the defined values of the two symbols modified by a constant "offset". Some example TASM input statements:

```
     .dw  @hello-@goodbye      ;Constant offset of zero
     .dw  hello-goodbye        ;The same since both are "relative"
     .dw  hello-goodbye-1      ;A constant "offset" of -1 (bytes)
```

Record: 14 **Name:** T_ADDR_DATA

```
     1U   2U        2U             4S
     [14] [line #]  [symbol #]     [offset]
```

This record is present in ".TRL" (and thus ".TLL" files). It is used to represent a word of data which is to be initialized to the defined value of the symbol modified by a constant "offset". Some example TASM input statements:

```
     .dw  hello           ;Constant offset of zero
     .dw  hello+1         ;Constant offset of 1 byte
```

Record: 15 **Name:** T_STORAGE

```
     1U   2U        4S
     [15] [line #]  [# of data bytes]
```

This record is present in ".TRL", ".TLL" and ".TLD" files. It serves as a place-holder for the specified # of bytes of uninitialized data. The area reserved using this record is initialized to contain all zero's during the bootstrap process.

Record: 16 Name: T_DEF

```
1U   2U         2U
[16] [line #]  [symbol #]
```

This record is present in ".TRL" (and thus ".TLL"), files. It serves as a place-holder for the definition position of a symbol in the file. Exactly one of these records is present for every label in the input file during the execution of TASM. T_DEF records whose relative address can't be completely bound by TASM are passed along to TLNK for further processing.

Record: 17 Name: T_SET

```
1U   2U         2U              4S
[17] [line #]  [symbol #]      [value]
```

This record is present in ".TRL" (and thus ".TLL"), files. It is similar in function to T_DEF, except it arises out of symbols defined by ".set" pseudo-opcodes in TASM. There is only one T_SET record for a particular symbol in a file, but its position within the file is not significant (the "value" field is used instead). T_SET records are only externally written by TASM when the value is being exported (via a ".pub" pseudo-opcode), or the symbol value is needed in some other record which couldn't be completely bound at assembly time. An example of this would be if the symbol defined in this fashion was one of the two symbols present in a "symbol"-"symbol" expression within some other record, and where the other symbol couldn't be completely bound by TASM.

Record: 18 Name: T_REL_OP

```
1U   2U         1U                    4S
[18] [line #]  [minimum length]     [address]

(continued with)    1U
                    [opcode]
```

This record is present in ".TRL" (and thus ".TLL" files). It is used to represent an instruction whose immediate data is the relative offset between the beginning of the following instruction and the indicated memory address. This record is generated by TASM in response to statements such as:

```
j     @0x1000          ;Jump to absolute location 0x1000
```

Record: 19 **Name:** T_RELSYM_OP

```
1U   2U        1U                  2U
[19] [line #]  [minimum length]    [symbol #]
```

(continued with)

```
4S        1U
[offset]  [opcode]
```

This record is present in ".TRL" (and thus ".TLL" files). It is used to represent an instruction whose immediate data is the relative offset between the beginning of the following instruction and the indicated symbol #, modified by a constant offset. This record is generated by TASM in response to statements such as:

```
j     @hello          ;Jump to location defined by "hello", 0 offset
j     @hello-3        ;Jump to 3 bytes before symbol definition addr
```

Record: 20 **Name:** T_RELREL_OP

```
1U   2U        1U                  2U
[20] [line #]  [minimum length]    [left symbol #]
```

(continued with)

```
2U                  4S        1U
[right symbol #]    [offset]  [opcode]
```

This record is present in ".TRL" (and thus ".TLL" files). It is used to represent an instruction whose immediate data is the difference between the defined values (or addresses), of two symbols modified by a constant offset. This record is generated by TASM in response to statements such as:

```
ldc  @hello-@goodbye     ;Load difference between defs, 0 offset
ldc  hello-goodbye       ;Same as above since difference is relative
ldc  a1-a2+0x100         ;Difference plus offset of 0x100
```

Record: 21 **Name:** T_ADDR_OP

```
1U   2U        1U                  2U
[21] [line #]  [minimum length]    [symbol #]
```

(continued with)
```
4S        1U
[offset]  [opcode]
```

This record is present in ".TRL" (and thus ".TLL" files). It is used to represent an instruction whose immediate data is the defined value of the "symbol #", modified by a constant offset. This record is generated by TASM in response to statements such as (assuming "hello" is a label definition and "byte_count" is a symbol defined by a ".set" pseudo-opcode):

```
ldc  hello                ;Load a pointer to "hello", 0 offset
ldc  byte_count+1         ;Load symbolic value plus 1
```

Record: 22 **Name:** T_LOAD

```
1U   4S
[22] [load address]
```

This record is only present in ".TLD" files. It is used to specify a load address for the T_DATA records which follow. There will always be at least one of these records in a ".TLD" file but more than one may exist as the result of location directives given to TLNK. The "load address" is simply the next address to load code or data into.

Record: 23 **Name:** T_STACK

```
1U   4S
[23] [stack address]
```

This record is only present in ".TLD" files. It is used to specify an initial starting address for the workspace pointer (stack). There will be exactly one of these records in each ".TLD" file.

Record: 24 **Name:** T_ENTRY

```
1U   4S
[24] [entry point address]
```

This record is only present in ".TLD" files. It is used to specify an initial program starting address. There will be exactly one of these records in each ".TLD" file.

Record: 25 Name: T_DEBUG_DATA

```
1U   2U         4S         2U              VAR
[25] [line #]  [value]    [data length]   [data]
```

This record is optionally present in ".TRL", ".TLL" and ".TLD" files, and is used to hold debugging information. The record is generated by TASM when it encounters a ".SYM" pseudo opcode. For example:

```
.sym zip,27
```

This builds a record with a "value" field of 27, and the name "zip" in the variable length "data" field. Note that the "data" field has the following internal structure:

```
1U                    VAR             VAR
[symbol length]       [symbol name]   [additional data]
```

The "symbol length" field holds the length of the symbol (3 in the example of "zip" above), and the "symbol name" field holds the actual name string. The "symbol name" field may be from 0 to 255 bytes long depending on the value of "symbol length".

The "data length" field would contain 4 in this example (1 for the "symbol length" field and 3 for the actual "symbol name"). Note that it is possible for "additional data" to follow the name in the record. This consists of zero or more signed, four byte, values which hold more information about the symbol. In TASM this data is supplied to the ".SYM" pseudo opcode by appending extra comma separated expressions after the "value". For example:

```
.sym zip,27,1,2,3
```

This statement will cause a record which contains three "additional data" fields to be generated which will contain the values 1, 2 and 3 respectively. Note that you can determine the number of these fields present in the "data" section by subtracting the size of the "symbol length" field from the "data length" field and dividing the result by 4.

Obviously, the actual use of the T_DEBUG_DATA field is part of a convention between the tool which generates the records and the tool which later interprets them. For the case of records which are written by TASM at the request of the TCX "C" compiler, the following conventions hold:

1. The "symbol name" field holds the "real" source level name of a "C" variable or function.

2. The "value" field holds either the physical address of the variable/function or the local workspace offset of the variable (depending on the storage class).

3. Two "additional data" fields are generated; the first holds the "storage class" of the variable/function, the second holds a bit mask of the "top level type" information for the variable/function.

The "storage class" entry is defined by a set of macro definitions in the "tcx.h" TCX compiler include file. The following values are of interest:

Macro Name	Value	Description
SC_GLOBAL	2	A var/func external definition.
SC_ST_GLB	3	A "static" var/func external definition.
SC_STATIC	6	A "static" variable inside a function.
SC_AUTO	7	A "auto" variable inside a function.
SC_ARG	8	A "parameter" variable inside a function.
SC_REG	9	A "register" variable inside a function.

The "top level type" bit mask entry is also defined in "tcx.h" with the following values of interest:

Macro Name	Bit Position	Description
UNSIGNED_NODE	0x40000000	Variable is of type "unsigned".
SIGNED_NODE	0x20000000	Variable is of type "signed".
CONST_NODE	0x10000000	Variable is of type "const".
VOID_NODE	0x08000000	Variable is of type "void".
CHAR_NODE	0x04000000	Variable is of type "char".
SHORT_NODE	0x02000000	Variable is of type "short".
INT_NODE	0x01000000	Variable is of type "int".
LONG_NODE	0x00800000	Variable is of type "long".
FLOAT_NODE	0x00400000	Variable is of type "float".
DOUBLE_NODE	0x00200000	Variable is of type "double".
LDOUBLE_NODE	0x00100000	Variable is of type "long double".
PTR_NODE	0x00080000	Variable is of type "pointer".
STRUCT_NODE	0x00040000	Variable is of type "struct".
UNION_NODE	0x00020000	Variable is of type "union".
ENUM_NODE	0x00004000	Variable is of type "enum".
ARRAY_NODE	0x00002000	Variable is an array.
FUNC_NODE	0x00001000	Variable is a function.

Note that the above information provides only superficial type information, thus the field name: "top level type".

Record: 26 **Name:** T_DEBUGSYM_DATA

```
1U   2U         4S         2U              2U
[26] [line #]  [value]    [data length]   [symbol #]
```

(continued with)
```
VAR
[data]
```

This record is extremely similar to the T_DEBUG_DATA record, except it is used where the "value" field in the generating ".SYM" record is symbolic (instead of being a strictly constant expression). For example:

```
.sym zip,zap+27
```

In this case the "symbol #" field in the record is associated with "zap" and the "value" field would hold the offset of "27". The "data length" field holds the size of the "data" field in bytes.

This record is commonly used for getting the address of a function or variable from the linker. Note that the linker converts this type of record into the T_DEBUG_DATA form when the associated address has been determined (the address is added to the offset in the "value" field and used for the "value" field of the new T_DEBUG_DATA record).

This record is present in ".TRL" and ".TLL" files. See the description of the T_DEBUG_DATA record for information about the contents of the "data" field.

Record: 27 **Name:** T_WRELREL_OP

```
1U   2U         1U                    2U
[27] [line #]  [minimum length]      [left symbol #]
```

(continued with)
```
2U                    4S         1U
[right symbol #]      [offset]   [opcode]
```

This record is present in ".TRL" (and thus ".TLL" files). It is used to represent an instruction whose immediate data is the difference between the defined values (or addresses), of two symbols modified by a constant offset SCALED by the word length of the processor. This record must resolve into a value which can be divided by the word length with a remainder of zero. This record is generated by TASM in response to statements such as:

```
ldc  $@hello-@goodbye     ;Load diff between defs / wordlength
ldc  $hello-goodbye       ;Same as above since difference is relative
ldc  $a1-a2+0x100         ;(diff + 0x100) / wordlength
```

Record Order Within Files

TASM Output Files/TLNK Input Files

Introduction

As expected, TASM output files (also called relocatable files), use most of the record types. The records which don't appear in these files are:

`T_RESERVED`	These are for internal use only.
`T_LIB_FILE`	Used only for TLIB library files.
`T_LD_FILE`	Used only by TLNK output files.
`T_SIZE`	Used only for TLIB library files.
`T_LOAD`	Used only for TLNK output files.
`T_STACK`	Used only for TLNK output files.
`T_ENTRY`	Used only for TLNK output files.

The following description provides some information about individual record order within a relocatable file. For more information check the detailed record descriptions above.

General Relocatable Record Structure

First record: `T_REL_FILE`

The record defines the file to be a relocatable file and keeps track of the CPU for which the code in the file is intended (T414B, T800, ...).

External symbol definition/reference record #1: `T_SYMBOL`
External symbol definition/reference record #2: `T_SYMBOL`
...
External symbol definition/reference record #N: `T_SYMBOL`

One T_SYMBOL record is present for each external symbol used in TASM (either via ".ext" or ".pub"). These records start immediately following the T_REL_FILE record.

Last record: `T_EOF`

This record defines the end of the relocatable file.

General Load File Record Structure

Given the previously mentioned records, the middle of the relocatable file consists of a sequence of the remaining types of records. See the **Detailed Record Descriptions** section for more information about these records.

TLNK Output Files/Loader Input Files

Introduction

Load files produced by TLNK contain a fairly small subset of the records mentioned elsewhere in this document. In general, you need the file type; the load, stack and entry point addresses; the actual data and T_STORAGE place-holders, and the information about where the data in the load file originated from. Optionally, debug information records may be present unless the "-C" flag has been used with TCX/TASM/TLNK to remove them (which makes the file MUCH smaller).

General Load File Record Structure

First record: `T_LD_FILE`

The record defines the file to be a load file and keeps track of the CPU for which code in the file is intended.

Second record: `T_LOAD`

The record specifies the "default" load address for the file. This address is set by TLNK to the start of off-chip ram (based on the CPU type). If the user explicitly specifies a load address to TLNK an additional T_LOAD is placed later in the file.

Third record: `T_STACK`

Sets the initial workspace pointer (stack), address for the program. The default value used by TLNK is the top of the on-chip ram, however the user may explicitly request it to be somewhere else.

Fourth record: `T_ENTRY`

Sets the program entry point address. TLNK assigns this to be the value of the "_main" symbol by default, but it may be explicitly set by the user.

Last record: `T_EOF`

This record defines the end of the load file.

Given the previously mentioned records, the middle of the load file consists of a sequence of the following four (optionally five), records:

Initialized code and data: `T_DATA`

Uninitialized data: `T_STORAGE`

Source filename information: `T_FILENAME`

User specified load addresses: `T_LOAD`

Debugging information: T_DEBUG

TLIB Output Files (libraries)

Introduction

Libraries produced by TLIB may be thought of as a series of relocatable files (TASM output files), together with some library overhead information to delineate the beginning and end of files and the library proper. The following description covers the library file format at the symbolic record level and treats the embedded relocatable files as indivisible entities (see the **Detailed Record Descriptions** section for more information about relocatable records).

General Library Record Structure

First record: `T_LIB_FILE`

The record defines the file to be a library and keeps track of the CPU for which code in the library is intended.

Library file #1 records
Library file #2 records
...
Library file #N records

These library file records are the collection of library overhead and actual relocatable file records which make up an entry in the library. Note that if a library is "empty" it will contain only the T_LIB_FILE and T_EOF records and no library file records. The structure of the library file records is defined in the next section.

Last record: `T_EOF`

This record defines the end of the library.

Library File Overhead Record Structure

First record: `T_FILENAME`

This record holds the filename of the individual relocatable file in the library. This name is the one which was given when the file was added to the library and may be different than any name which is recorded in the relocatable file proper (since source filename and line number information appear in relocatable files). This is the only name which is used to refer to the file as far as TLIB is concerned.

External symbol definition record #1: `T_SYMBOL`
External symbol definition record #2: `T_SYMBOL`
...
External symbol definition record #N: `T_SYMBOL`

This series of records keeps track of which external symbols are defined in this library file. The records are sorted on the symbol names (ASCII order). Although it is not illegal to have a relocatable file with no external symbol definitions, it is fairly meaningless to TLNK (TLIB will generate a warning message if this is detected).

Size/date record: `T_SIZE`

This record holds two things:

1. The creation date of the relocatable file (not the library, just the individual relocatable "subfile" in the library).

2. The size of the actual relocatable file in bytes. This is the same as the number of bytes that the file takes up when it is not part of a library. Since the library copy of the relocatable file has the same data as the individual file, this record also defines the size of the relocatable file records which follow this record (see the **TASM Output Files/TLNK Input Files** section for more information). Another way to look at it is that if you add this size to the current file position you will be pointing at the first record of the NEXT library file.