Assembler Output 3

This chapter is an overview of ELF *(Executable and Linking Format)* for the relocatable object files produced by the assembler. The fully detailed definition of ELF appears in the *System V Application Binary Interface* and the *Intel 386 Processor Supplement*.

This chapter is organized as follows:

Introduction	*page 151*
Object Files in Executable and Linking Format (ELF)	*page 152*

Introduction

The main output produced by assembling an input assembly language source file is the translation of that file into an object file in (ELF). ELF files produced by the assembler are relocatable files that hold code and/or data. They are input files for the linker. The linker combines these relocatable files with other ELF object files to create an executable file or a shared object file in the next stage of program building, after translation from source files into object files.

The three main kinds of ELF files are relocatable, executable and shared object files.

The assembler can also produce ancillary output incidental to the translation process. For example, if the assembler is invoked with the `-V` option, it can write information to standard output and to standard error.

3

The assembler also creates a default output file when standard input or multiple input files are used. Ancillary output has little direct connection to the translation process, so it is not properly a subject for this manual. Information about such output appears in `as`(1) manual page.

Certain assembly language statements are directives to the assembler regarding the organization or content of the object file to be generated. Therefore, they have a direct effect on the translation performed by the assembler. To understand these directives, described in Chapter 2, "Instruction-Set Mapping", it is helpful to have some working knowledge of ELF, at least for relocatable files.

Object Files in Executable and Linking Format (ELF)

Relocatable ELF files produced by the assembler consist of:

- An ELF header
- A section header table
- Sections

The ELF header is always the first part of an ELF file. It is a structure of fixed size and format. The fields, or members, of the structure describe the nature, organization and contents of the rest of the file. The ELF header has a field that specifies the location within the file where the section header table begins.

The section header table is an array of section headers that are structures of fixed size and format. The section headers are the elements of the array, or the entries in the table. The section header table has one entry for each section in the ELF file. However, the table can also have entries (section headers) that do not correspond to any section in the file. Such entries and their array indices are reserved. The members of each section header constitute information useful to the linker about the contents of the corresponding section, if any.

All of a relocatable file's information that does not lie within its ELF header or its section header table lies within its sections. Sections contain most of the information needed to combine relocatable files with other ELF files to produce shared object files or executable files. Sections also contain the material to be combined. For example, sections can hold:

- Relocation tables
- Symbol tables
- String tables

Each section in an ELF file fills a contiguous (possibly empty) sequence of that file's bytes. Sections never overlap. However, the (set theoretic) union of a relocatable ELF header, the section header table, and all the sections can omit some of the bytes. Bytes of a relocatable file that are not in the ELF header, or in the section header table, or in any of the sections constitute the inactive space. The contents of a file's inactive space, if any, are unspecified.

ELF *Header*

The ELF *header* is always located at the beginning of the ELF file. It describes the ELF file organization and contains the actual sizes of the object file control structures.

The ELF header consists of the following fields, or members, some have the value 0 for relocatable files:

`e_ident`

This is a byte array consisting of the `EI_NIDENT` initial bytes of the ELF header, where `EI_NIDENT` is a name for 16. The elements of this array mark the file as an ELF object file and provide machine-independent data that can be used to decode and interpret the file's contents.

`e_type`

Identifies the object file type. A value of 1, that has the name `ET_REL`, specifies a relocatable file. Table 3-1 describes all the object file types.

`e_machine`

Specifies the required architecture for an individual file. A value of 3, that has the name `EM_386`, specifies Intel 80386. `EM_486`, specifies Intel 80486.

`e_version`

Identifies the version of this object file's format. This field should have the current version number, named `EV_CURRENT`.

`e_entry`

Virtual address where the process is to start. A value of 0 indicates no associated entry point.

`e_phoff`

Program header table's file offset, in bytes. The value of 0 indicates no program header. (Relocatable files do not need a program header table.)

`e_shoff`

Section header table's file offset, in bytes. The value of 0 indicates no section header table. (Relocatable files must have a section header table.)

`e_flag`

Processor-specific flags associated with the file. For the Intel 80386, this field has value 0.

`e_ehsize`

ELF header's size, in bytes.

`e_phentsize`

Size, in bytes, of entries in the program header table. All entries are the same size. (Relocatable files do not need a program header table.)

`e_phnum`

Number of entries in program header table. A value of 0 indicates the file has no program header table. (Relocatable files do not need a program header table.)

`e_shentsize`

Size, in bytes, of the section header structure. A section header is one entry in the section header table; all entries are the same size.

`e_shnum`

Number of entries in section header table. A value of 0 indicates the file has no section header table. (Relocatable files must have a section header table.)

`e_shstrndx`

Section header table index of the entry associated with the section name string table. A value of `SHN_UNDEF` indicates the file does not have a section name string table.

Table 3-1 Object File Types

Type	Value	Description
none	0	No file type
rel	1	Relocatable file
exec	2	Executable file
dyn	3	Shared object file
core	4	Core file
loproc	0xff00	Processor-specific
hiproc	0xffff	Processor-specific

Section Header

The section header table has all of the information necessary to locate and isolate each of the file's sections. A section header entry in a section header table contains information characterizing the contents of the corresponding section, if the file has such a section.

Each entry in the section header table is a section header. A section header is a structure of fixed size and format, consisting of the following fields, or members:

`sh_name`

Specifies the section name. The value of this field is an index into the section header string table section, wherein it indicates the beginning of a null-terminated string that names the section.

`sh_type`

Categorizes the section's contents and semantics. Table 3-3 describes the section types.

`sh_flags`

One-bit descriptions of section attributes. Table 3-2 describes the section attribute flags.

`sh_addr`

Address where the first byte resides if the section appears in the memory image of a process; a value of 0 indicates the section does not appear in the memory image of a process.

`sh_offset`

Specifies the byte offset from the beginning of the file to the first byte in the section.

Note – If the section type is `SHT_NOBITS`, the corresponding section occupies no space in the file. In this case, `sh_offset` specifies the location at which the section would have begun if it did occupy space within the file.

`sh_size`

Specifies the size, in byte units, of the section.

Note – Even if the section type is `SHT_NOBITS`, `sh_size` can be nonzero; however, the corresponding section still occupies no space in the file.

`sh_link`

Section header table index link. The interpretation of this information depends on the section type, as described in Table 3-3.

`sh_info`

Extra information. The interpretation of this information depends on the section type, as described in Table 3-3.

`sh_addralign`

If a section has an address alignment constraint, the value in this field is the modulus, in byte units, by which the value of `sh_addr` must be congruent to 0; i.e., `sh_addr` = 0 (`mod sh_addralign`).

For example, if a section contains a long (32 bits), the entire section must be ensured long alignment, so `sh_addralign` has the value 4. Only 0 and positive integral powers of 2 are currently allowed as values for this field. A value of 0 or 1 indicates no address alignment constraints.

`sh_entsize`

Size, in byte units, for entries in a section that is a table of fixed-size entries, such as a symbol table. Has the value 0 if the section is not a table of fixed-size entries

Table 3-2 Section Attribute Flags

Flag	Default Value	Description
SHF_WRITE	0x1	Contains data that is writable during process execution.
SHF_ALLOC	0x2	Occupies memory during process execution. This attribute is *off* if a control section does not reside in the memory image of the object file.
SHF_EXECINSTR	0x4	Contains executable machine instructions.
SHF_MASKPROC	0xf0000000	Reserved for processor-specific semantics.

Table 3-3 Section Types

Name	Value	Description	Interpretation by	
			sh_info	sh_link
SHT_NULL	0	Marks section header as inactive; file has no corresponding section.	0	SHN_UNDEF
SHT_PROGBITS	1	Contains information defined by the program, and in a format and with a meaning determined solely by the program.	0	SHN_UNDEF
SHT_SYMTAB	2	Is a complete symbol table, usually for link editing. This table can also be used for dynamic linking; however, it can contain many unnecessary symbols. Note: Only one section of this type is allowed in a file	One greater than the symbol table index of the last local symbol.	The section header index of the associated string table.

Table 3-3 Section Types (Continued)

Name	Value	Description	Interpretation by	
			sh_info	**sh_link**
SHT_STRTAB	3	Is a string table. A file can have multiple string table sections.	0	SHN_UNDEF
SHT_RELA	4	Contains relocation entries with explicit addends. A file can have multiple relocation sections.	The section header index of the section to where the relocation applies.	The section header index of the associated symbol table.
SHT_HASH	5	Is a symbol rehash table. Note: Only one section of this type is allowed in a file	0	The section header index of the symbol table to which the hash table applies.
SHT_DYNAMIC	6	Contains dynamic linking information. Note: Only one section of this type is allowed in a file	0	The section header index of the string table used by entries in the section.
SHT_NOTE	7	Contains information that marks the file.	0	SHN_UNDEF
SHT_NOBITS	8	Contains information defined by the program, and in a format and with a meaning determined by the program. However, a section of this type occupies no space in the file, but the section header's *offset* field specifies the location at which the section would have begun if it did occupy space within the file.	0	SHN_UNDEF
SHT_REL	9	Contains relocation entries without explicit addends. A file can have multiple relocation sections.	The section header index of the section to where the relocation applies.	The section header index of the associated symbol table.
SHT_SHLIB	10	Reserved.	0	SHN_UNDEF

Table 3-3 Section Types (Continued)

Name	Value	Description	Interpretation by	
			sh_info	**sh_link**
SHT_DYNSYM	11	Is a symbol table with a minimal set of symbols for dynamic linking. Note: Only one section of this type is allowed in a file	One greater than the symbol table index of the last local symbol.	The section header index of the associated string table.
SHT_LOPROC SHT_HIPROC	0x700 00000 0x7fff fff	Lower and upper bounds of range of section types reserved for processor-specific semantics.	0	SHN_UNDEF
SHT_LOUSER SHT_HIUSER	0x800 00000 0xffff fff	Lower and upper bounds of range of section types reserved for application programs. Note: Section types in this range can be used by an application without conflicting with system-defined section types.	0	SHN_UNDEF

Note – Some section header table indices are reserved, and the object file does not contain sections for these special indices.

Sections

A section is the smallest unit of an object file that can be relocated. Sections containing the following material usually appear in relocatable ELF files:

- Executable text
- Read-only data
- Read-write data
- Read-write uninitialized data (only *section header* appears)

Sections do not need to occur in any particular order within the object file. The sections of a relocatable ELF file contain all of the file information that is not contained in the ELF header or in the section header table. The sections in any ELF file must satisfy several conditions:

1. Every section in the file must have one section header entry in the section header table to describe the section. However, the section header table can have section header entries that correspond to no section in the file.
2. Each section occupies one contiguous sequence of bytes within a file. The section can be empty (even so, its section header entry in the section header table can have a nonzero value for the field `sh_size`).
3. A byte in a file can reside in at most one section. Sections in a file cannot overlap.
4. An object file can have inactive space. Inactive space is the set of all bytes in the file that are not part of the ELF header, the section header table, the program header table (for executable files), or of any section in the file. The contents of the inactive space are unspecified.

Sections can be added for multiple text or data segments, shared data, user-defined sections, or information in the object file for debugging.

Note – Not all of the sections where there are entries in the file section header table need to be present.

Predefined Sections

Sections having certain names beginning with "." (dot) are predefined, with their types and attributes already assigned. These special sections are of two kinds: predefined user sections and predefined nonuser sections.

Predefined User Sections

Sections that an assembly language programmer can manipulate by issuing section control directives in the source file are *user sections*. The predefined user sections are those predefined sections that are also user sections.

Table 3-4 lists the names of the predefined user sections and briefly describes each.

Table 3-4 Predefined User Sections

Section Name	Description
".bss"	Uninitialized read-write data.
".comment"	Version control information.
".data" & ".data1"	Initialized read-write data.
".debug"	Debugging information.
".fini"	Runtime finalization instructions.
".init"	Runtime initialization instructions.
".rodata" & ".rodata1"	Read-only data.
".text"	Executable instructions.
".line"	Line # info for symbolic debugging.
".note"	Special information from vendors or system builders.

Predefined Non-User Sections

Table 3-5 shows the predefined sections that are not user sections, because assembly language programmers cannot manipulate them by issuing section control directives in the source file.

Table 3-5 Predefined Non-User Sections

Section Name	Description
".dynamic"	Dynamic linking information.
".dynstr"	Strings needed for dynamic linking.
".dynsym"	Dynamic linking symbol table.
".got"	Global offset table.

Table 3-5 Predefined Non-User Sections (Continued)

Section Name	Description
".hash"	A symbol hash table.
".interp"	The path name of a program interpreter.
".plt"	The procedure linking table.
"rel*name*" & ".rela*name*"	Relocation information. *name* is the section to which the relocations apply. e.g., ".rel.text", ".rela.text".
".shstrtab"	String table for the section header table names.
".strtab"	The string table.
".symtab"	The symbol table.

Relocation Tables

Locations represent *addresses in memory* if a section is allocatable; that is, its contents are to be placed in memory at program runtime. Symbolic references to these locations must be changed to addresses by the link editor.

The assembler produces a companion *relocation table* for each relocatable section. The table contains a list of relocations (that is, adjustments to locations in the section) to be performed by the link editor.

Symbol Tables

The *symbol table* contains information to locate and relocate symbolic definitions and references. The assembler creates the symbol table section for the object file. It makes an entry in the symbol table for each symbol that is defined or referenced in the input file and is needed during linking.

The symbol table is then used by the link editor during relocation. The symbol table's section header contains the symbol table index for the first non-local symbol.

The symbol table contains the following information:

`st_name`

Index into the object file symbol string table. A value of zero indicates the corresponding entry in the symbol table has no name; otherwise, the value represents the string table index that gives the symbol name.

`st_value`

Value of the associated symbol. This value is dependent on the context; for example, it can be an address, or it can be an absolute value.

`st_size`

Size of symbol. A value of 0 indicates that the symbol has either no size or an unknown size.

`st_info`

Specifies the symbol type and binding attributes. Table 3-6 andTable 3-7 describe the symbol types and binding attributes.

`st_other`

Undefined meaning. Current value is 0.

`st_shndx`

Contains the section header table index to another relevant section, if specified. As a section moves during relocation, references to the symbol continue to point to the same location because the value of the symbol changes as well.

Table 3-6 Symbol Types

Value	Type	Description
0	notype	Type not specified.
1	object	*Symbol* is associated with a data object; for example, a variable or an array.
2	func	*Symbol* is associated with a function or other executable code. When another object file references a function from a shared object, the link editor automatically creates a procedure linkage table entry for the referenced symbol.
3	section	*Symbol* is associated with a section. These types of symbols are primarily used for relocation.
4	file	Gives the name of the source file associated with the object file.
13 15	loproc hiproc	Values reserved for processor-specific semantics.

Table 3-7 Symbol Bindings

Value	Binding	Description
0	local	*Symbol* is defined in the object file and not accessible in other files. Local symbols of the same name can exist in multiple files.
1	global	*Symbol* is either defined externally or defined in the object file and accessible in other files.
2	weak	*Symbol* is either defined externally or defined in the object file and accessible in other files; however, these definitions have a lower precedence than globally defined symbols.
13 15	loproc hiproc	Values reserved for processor-specific semantics.

String Tables

A *string table* is a section which contains null-terminated variable-length character sequences, or strings. The object file uses these strings to represent symbol names and file names. The strings are referenced by indices into the string table section. The first and last bytes of a string table must be the null character.

- A string table index can refer to any byte in the section.

- Empty string table sections are permitted if zero is the value of `sh_size` in the section header entry for the string table in the section header table.

A string can appear multiple times and can also be referenced multiple times. References to substrings can exist, and unreferenced strings are allowed.

3