Relocation

Relocation is the process of connecting symbolic references with symbolic definitions. For example, when a program calls a function, the associated call instruction must transfer control to the proper destination address at execution. In other words, relocatable files must have information that describes how to modify their section contents, thus allowing executable and shared object files to hold the right information for a process's program image. *Relocation entries* are these data.

Figure 1-20: Relocation Entries

```
typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
} Elf32_Rel;

typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
        Elf32_Sword     r_addend;
} Elf32_Rela;
```

`r_offset` This member gives the location at which to apply the relocation action. For a relocatable file, the value is the byte offset from the beginning of the section to the storage unit affected by the relocation. For an executable file or a shared object, the value is the virtual address of the storage unit affected by the relocation.

`r_info` This member gives both the symbol table index with respect to which the relocation must be made, and the type of relocation to apply. For example, a call instruction's relocation entry would hold the symbol table index of the function being called. If the index is `STN_UNDEF`, the undefined symbol index, the relocation uses 0 as the ''symbol value.'' Relocation types are processor-specific. When the text refers to a relocation entry's relocation type or symbol table index, it means the result of applying `ELF32_R_TYPE` or `ELF32_R_SYM`, respectively, to the entry's `r_info` member.

```
#define ELF32_R_SYM(i)    ((i)>>8)
#define ELF32_R_TYPE(i)   ((unsigned char)(i))
#define ELF32_R_INFO(s,t) (((s)<<8)+(unsigned char)(t))
```

`r_addend` This member specifies a constant addend used to compute the value to be stored into the relocatable field.

As shown above, only `Elf32_Rela` entries contain an explicit addend. Entries of type `Elf32_Rel` store an implicit addend in the location to be modified. Depending on the processor architecture, one form or the other might be necessary or more convenient. Consequently, an implementation for a particular machine may use one form exclusively or either form depending on context.

A relocation section references two other sections: a symbol table and a section to modify. The section header's `sh_info` and `sh_link` members, described in ''Sections'' above, specify these relationships. Relocation entries for different object files have slightly different interpretations for the `r_offset` member.

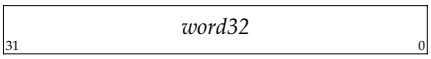
- In relocatable files, `r_offset` holds a section offset. That is, the relocation section itself describes how to modify another section in the file; relocation offsets designate a storage unit within the second section.
- In executable and shared object files, `r_offset` holds a virtual address. To make these files' relocation entries more useful for the dynamic linker, the section offset (file interpretation) gives way to a virtual address (memory interpretation).

Although the interpretation of `r_offset` changes for different object files to allow efficient access by the relevant programs, the relocation types' meanings stay the same.

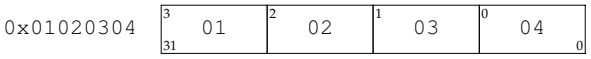
Relocation Types

Relocation entries describe how to alter the following instruction and data fields (bit numbers appear in the lower box corners).

Figure 1-21: Relocatable Fields



word32 This specifies a 32-bit field occupying 4 bytes with arbitrary byte alignment. These values use the same byte order as other word values in the 32-bit Intel Architecture.



Calculations below assume the actions are transforming a relocatable file into either an executable or a shared object file. Conceptually, the link editor merges one or more relocatable files to form the output. It first decides how to combine and locate the input files, then updates the symbol values, and finally performs the relocation. Relocations applied to executable or shared object files are similar and accomplish the same result. Descriptions below use the following notation.

`A` This means the addend used to compute the value of the relocatable field.

`B` This means the base address at which a shared object has been loaded into memory during execution. Generally, a shared object file is built with a 0 base virtual address, but the execution address will be different.

G — This means the offset into the global offset table at which the address of the relocation entry's symbol will reside during execution. See ''Global Offset Table'' in Part 2 for more information.

GOT — This means the address of the global offset table. See ''Global Offset Table'' in Part 2 for more information.

L — This means the place (section offset or address) of the procedure linkage table entry for a symbol. A procedure linkage table entry redirects a function call to the proper destination. The link editor builds the initial procedure linkage table, and the dynamic linker modifies the entries during execution. See ''Procedure Linkage Table'' in Part 2 for more information.

P — This means the place (section offset or address) of the storage unit being relocated (computed using `r_offset`).

S — This means the value of the symbol whose index resides in the relocation entry.

A relocation entry's `r_offset` value designates the offset or virtual address of the first byte of the affected storage unit. The relocation type specifies which bits to change and how to calculate their values. The SYSTEM V architecture uses only `Elf32_Rel` relocation entries, the field to be relocated holds the addend. In all cases, the addend and the computed result use the same byte order.

Figure 1-22: Relocation Types

Name	Value	Field	Calculation
R_386_NONE	0	none	none
R_386_32	1	*word32*	S + A
R_386_PC32	2	*word32*	S + A – P
R_386_GOT32	3	*word32*	G + A – P
R_386_PLT32	4	*word32*	L + A – P
R_386_COPY	5	none	none
R_386_GLOB_DAT	6	*word32*	S
R_386_JMP_SLOT	7	*word32*	S
R_386_RELATIVE	8	*word32*	B + A
R_386_GOTOFF	9	*word32*	S + A – GOT
R_386_GOTPC	10	*word32*	GOT + A – P

Some relocation types have semantics beyond simple calculation.

R_386_GOT32 — This relocation type computes the distance from the base of the global offset table to the symbol's global offset table entry. It additionally instructs the link editor to build a global offset table.

R_386_PLT32 — This relocation type computes the address of the symbol's procedure linkage table entry and additionally instructs the link editor to build a procedure linkage table.

R_386_COPY — The link editor creates this relocation type for dynamic linking. Its offset member refers to a location in a writable segment. The symbol table index specifies a symbol that should exist both in the current object file and in a shared object. During execution, the dynamic linker copies data associated with the shared object's symbol to the location specified by the offset.

R_386_GLOB_DAT	This relocation type is used to set a global offset table entry to the address of the specified symbol. The special relocation type allows one to determine the correspondence between symbols and global offset table entries.
R_3862_JMP_SLOT	The link editor creates this relocation type for dynamic linking. Its offset member gives the location of a procedure linkage table entry. The dynamic linker modifies the procedure linkage table entry to transfer control to the designated symbol's address [see ''Procedure Linkage Table'' in Part 2].
R_386_RELATIVE	The link editor creates this relocation type for dynamic linking. Its offset member gives a location within a shared object that contains a value representing a relative address. The dynamic linker computes the corresponding virtual address by adding the virtual address at which the shared object was loaded to the relative address. Relocation entries for this type must specify 0 for the symbol table index.
R_386_GOTOFF	This relocation type computes the difference between a symbol's value and the address of the global offset table. It additionally instructs the link editor to build the global offset table.
R_386_GOTPC	This relocation type resembles R_386_PC32, except it uses the address of the global offset table in its calculation. The symbol referenced in this relocation normally is _GLOBAL_OFFSET_TABLE_, which additionally instructs the link editor to build the global offset table.