# Computer Organization

**605.204**

## Module Four

Part Three

## Object File

# Module Four

- Part Three
- In this presentation, we are going to talk about :
- Assembler
  - Object File - the Assembler's output file

# Previously

- Previously we talked about:
- Assembler
  - Functions
  - Organization
  - Algorithm
- Now: Assembler
  - Object File - the Assembler's output file

# Object files

- Created by the assembler, object files are binary representations of programs intended to execute directly on a processor.
- There are three main types of object files.
- A **relocatable** file holds code and data suitable for linking with other object files to create an executable or a shared object file.
- An **executable** file holds a program suitable for execution; the file specifies how the dynamic linker creates a program's process image.
- A **shared object** file holds code and data suitable for linking in two contexts. First, the link editor may process it with other relocatable and shared object files to create another object file. Second, the dynamic linker combines it with an executable file and other shared objects to create a process image.

# Object file

- An early Unix example
- The a.out file format

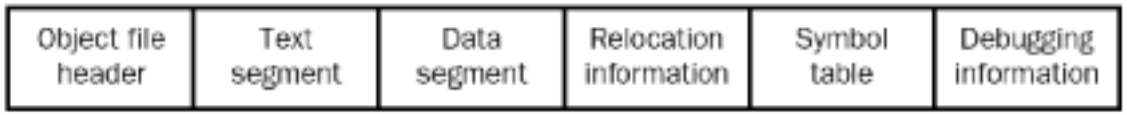



**FIGURE A.7 Object file.** A Unix assembler produces an object file with six distinct sections.

# a.out Details

An *a.out* file consists of up to seven sections, in the following order:

- **exec header**
  Contains parameters used by the kernel to load a binary file into memory and execute it, and by the link editor to combine a binary file with other binary files. This section is the only mandatory one.
- **text segment**
  Contains machine code and related data that are loaded into memory when a program executes. May be loaded read-only.
- **data segment**
  Contains initialized data; always loaded into writable memory.
- **text relocations**
  Contains records used by the link editor to update pointers in the text segment when combining binary files.

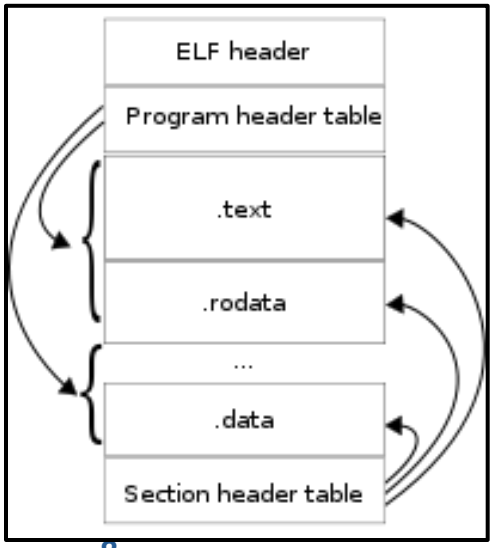
# more a.out Details

- **data relocations**
  Like the text relocation section, but for data segment pointers.
- **symbol table**
  Contains records used by the link editor to cross-reference the addresses of named variables and functions (*symbols*) between binary files.
- **string table**
  Contains the character strings corresponding to the symbol names.

The *a.out* support for **debug** information is done through the use of special entries in the symbol table called stabs.

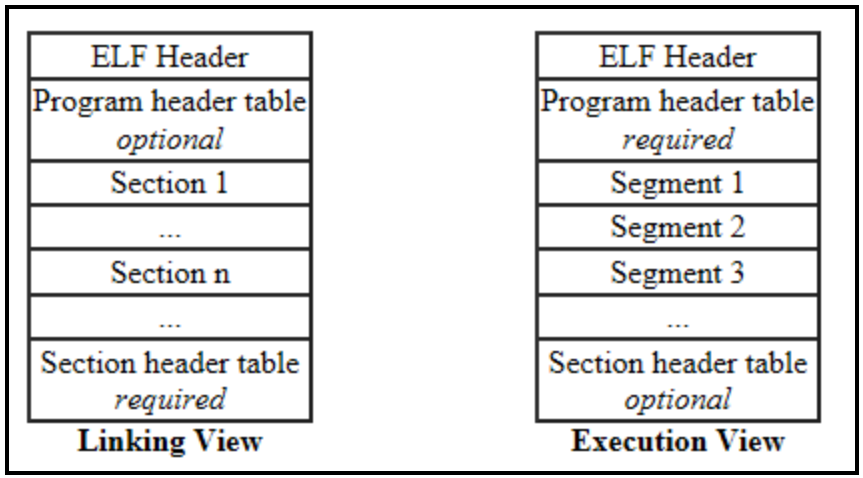
# ELF: Executable and Linking Format

- The a.out file format was replaced by ELF in the late 1990's.
- The Executable and Linking Format is a portable object file format that works on 32-bit Intel Architecture environments for a variety of operating systems.
- The ELF standard format:



# ELF format

- An **ELF header** resides at the beginning and holds a ‘‘road map’’ describing the file’s organization.
- **Sections** hold the bulk of object file information for the linking view: instructions, data, symbol table, relocation information, and so on.



# Section definitions

- **.text** - This section holds the ‘‘text,’’ or executable instructions, of a program.
- **.data** and **.data1** - These sections hold initialized data that contribute to the program’s memory image.
- **.rodata** and **.rodata1** - These sections hold read-only data that typically contribute to a non-writable segment in the process image.
- **.bss** - This section holds uninitialized data that contribute to the program’s memory image. By definition, the system initializes the data with zeros when the program begins to run

# Symbol Table

- SYMTAB and DYNSYM
- These sections hold a symbol table.
- SYMTAB provides symbols for link editing, though it may also be used for dynamic linking. As a complete symbol table, it may contain many symbols unnecessary for dynamic linking.
- DYNSYM section, which holds a minimal set of dynamic linking symbols, to save space.

# Symbol Table

- An object file's symbol table holds information needed to locate and relocate a program's symbolic definitions and references.
- **name** - holds an index into the object file's symbol string table, which holds the character representations of the symbol names.
- **value** - gives the value of the associated symbol. Depending on the context, this may be an absolute value, an address, etc.; details in the info object.
- **size** - many symbols have associated sizes.
- **info** - specifies the symbol's type and binding attributes.

# String Table

- STRTAB
- The section holds a string table. An object file may have multiple string table sections.
- String table sections hold null-terminated character sequences, commonly called strings. The object file uses these strings to represent symbol and section names. One references a string as an index into the string table section.
- The first byte, which is index zero, is defined to hold a null character. Likewise, a string table's last byte is defined to hold a null character, ensuring null termination for all strings.

# 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.

- RELA
- The section holds relocation entries with explicit addends.
- REL
- The section holds relocation entries without explicit addends.

- An object file may have multiple relocation sections.

# Relocation

- 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.
- **offset** - This member gives the location at which to apply the relocation action. The value is the byte offset from the beginning of the section to the storage unit affected by the relocation.
- **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.

# Summary

- Object file - standard formats
- a.out
- ELF
- Header section
- Multiple sections

- Next: Additional Assembler Features