Linking and Loading

Γ.Κορνάρος – Κ. Χαρτερός

Περίληψη

- Linking
  - ✓ Τι συμβαίνει κατά το linking?
  - ✓ Διαφορετικά είδη linking
- Loading
  - ✓ ELF file format

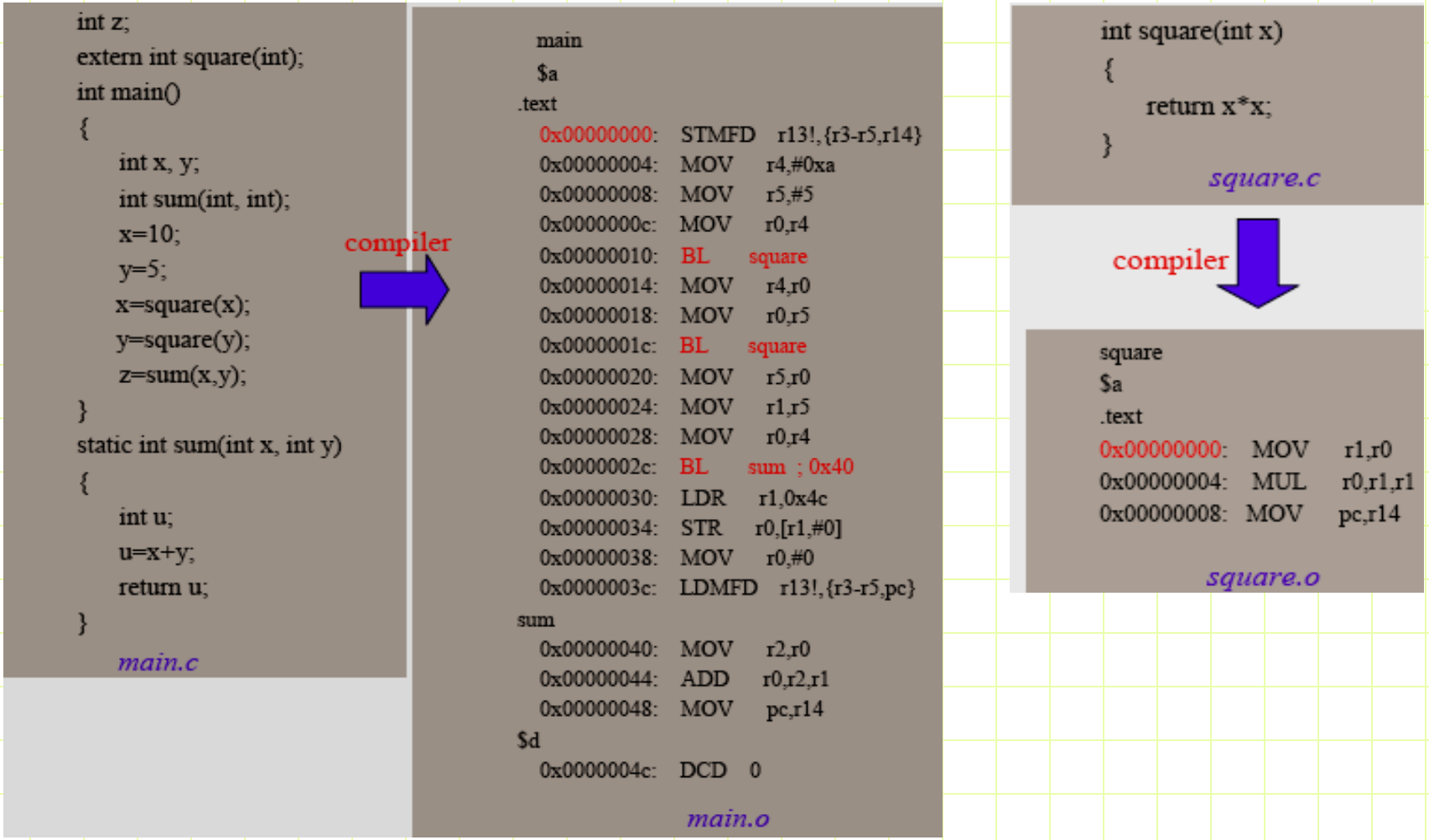
Linker

- Οι compilers και οι assemblers γεννούν re-locatable object files
  - ✓ References to external symbols are not resolved
  - ✓ Compilers generate object files in which code starts at address 0
  - ✓ Cannot execute a compiler produced object file
- Τα εκτελέσιμα αρχεία δημιουργούνται από το καθένα object file και τις βιβλιοθήκες με την διαδικασία του linking

Linker (2)

- Ο Linker εκτελεί δύο εργασίες :
  - ✓ Symbol resolution: Object files define and reference symbols, linker tries to resolve each symbol reference with one symbol definition
  - ✓ Relocation: Linker tries to relocate code and data from different object files so that different sections start at different addresses and all the references are updated

Παράδειγμα: Compiling main.c and square.c



Παράδειγμα: After Linking main.o and square.o

```
main
 $a
.text
      0x000080a8:   STMFD   r13!,{r3-r5,r14}
      0x000080ac:   MOV     r4,#0xa
      0x000080b0:   MOV     r5,#5
      0x000080b4:   MOV     r0,r4
      0x000080b8:   BL      square  ; 0x80f8
      0x000080bc:   MOV     r4,r0
      0x000080c0:   MOV     r0,r5
      0x000080c4:   BL      square  ; 0x80f8
      0x000080c8:   MOV     r5,r0
      0x000080cc:   MOV     r1,r5
      0x000080d0:   MOV     r0,r4
      0x000080d4:   BL      sum  ; 0x80e8
      0x000080d8:   LDR     r1,0x80f4
      0x000080dc:   STR     r0,[r1,#0]
      0x000080e0:   MOV     r0,#0
      0x000080e4:   LDMFD   r13!,{r3-r5,pc}
```

```
sum
      0x000080e8:   MOV     r2,r0
      0x000080ec:   ADD     r0,r2,r1
      0x000080f0:   MOV     pc,r14

 square
 $a
 .text
      0x000080f8:   MOV     r1,r0
      0x000080fc:   MUL     r0,r1,r1
      0x00008100:   MOV     pc,r14
```

linker adds the actual address of symbol *square*

linker merges the code from separate object files into a single executable file

linker relocates the code to a different memory location

5

Συναρτήσεις Βιβλιοθήκης

- Τι συμβαίνει όταν τα αρχεία source χρησιμοποιούν συναρτήσεις βιβλιοθήκης όπως printf, scanf, etc.?
- Compiler produces a symbol (in the same way as the square function in the previous example) in the object file
- Linker
  - ✓ Attempts to resolve these references by matching them to definitions found in other object files
  - ✓ If the symbol is not resolved, the linker searches for the symbol definition in library files
- What are library files?
  - ✓ Collection of object files that provide related functionality
  - ✓ Example: The standard C library libc.a is a collection of object files printf.o,
- scanf.o, fprintf.o, fscanf.o…

Συναρτήσεις Βιβλιοθήκης (2)

- Πως γνωρίζει ο linker που να βρει την βιβλιοθήκη?

  - ✓ User defined libraries can be specified as a command line argument, or in the CodeWarrior IDE (when you “Add Files” to a project)

  - ✓ The environment variable ARMLIB holds the path (Example– C:\Program Files\ARM\ADSv1_2\Lib) for the default ARM standard library

Συναρτήσεις Βιβλιοθήκης (3)

- Linker does a search to see whether the symbol is defined in the specified libraries
- The order in which this search is performed is determined by the order in which the libraries are specified
  - ✓ If the symbol is defined in more than 1 library, the first library in the path is selected
- Linker then extracts the specific .o file that defines the symbol in the library and processes this .o file with all the other object files
  - ✓ If the symbol is not defined in any of the library, linker throws an error

Είδη Αρχείων Object

- Three main types of object files
  - ✓ **Re-locatable file**: Code and data suitable for linking with other object files to create an executable or a shared object file
  - ✓ **Executable file**: Program suitable for execution
  - ✓ **Shared object file (also called "Dynamically linked library"**: Special type of relocatable object file that can be loaded into memory and linked dynamically

- First, the linker may process it with other re-locatable 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

- Compilers and assemblers generate re-locatable object files

- Linkers generate executable object files

Είδη Μοντέλων Linking

- Different kinds of linking models
  - ✓ ***Static:*** Set of object files, system libraries and library archives are statically bound, references are resolved, and a *self-contained executable file* is created
- Problem: If multiple programs are running on the processor simultaneously, and they require some common library module (say, printf.o), multiple copies of this common module are included in the executable file and loaded into memory (waste of memory!)
  - ✓ ***Dynamic:*** Set of object files, libraries, system shared resources and other shared libraries are linked together to create an executable file
- When this executable is loaded, *other shared resources and dynamic libraries must be made available* in the system for the program to run successfully
- If multiple programs running on a processor need the same object module, only one copy of the module needs to be loaded in the memory

Δυναμικό Linking

- Dynamically linked executable or shared object undergoes final linking when
  - ✓ Loaded into memory by a program loader
- An executable or shared object to be linked dynamically might
  - ✓ List one or more shared objects (shared libraries) with which it should be linked
- Other advantages of dynamic linking
  - ✓ Updating of libraries
- The size on disk of an executable that uses dynamically linked modules may be less than its size in memory (during run-time)
  - ✓ Why?

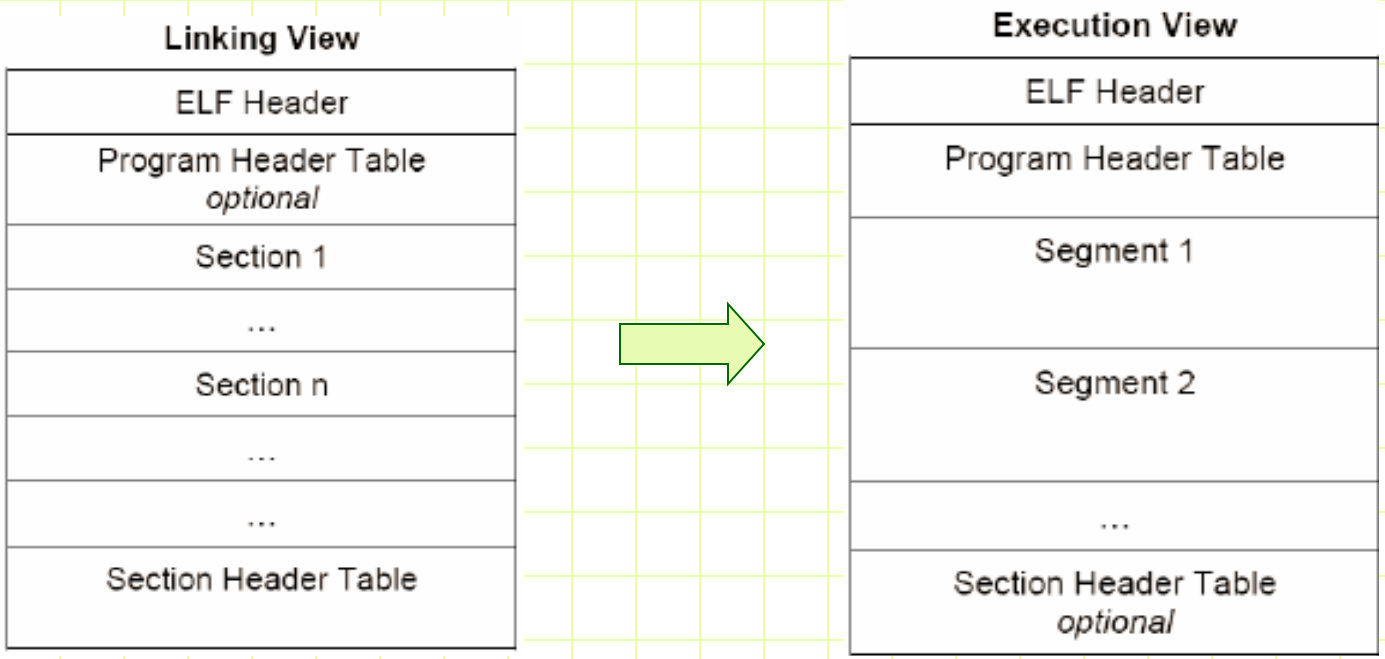
Executable and Linking Format (ELF)

- Τα Object files πρέπει να είναι σε ένα συγκεκριμένο φορματ για να διευκολυνθεί το linking και το loading
- Το Executable and Linking Format (ELF) είναι ένα δημοφιλές φορμάτ για ένα object file
- Supported by many vendors and tools
  - ✓ Diverse processors, multiple data encodings and multiple classes of machines
- ELF specifies the layout of the object files and not the contents of code or data

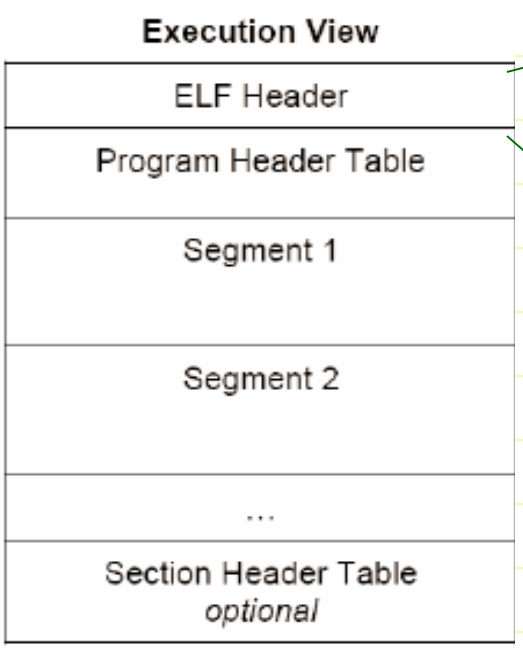
Executable and Linking Format (ELF)

- Ένα ELF object αρχείο αποτελείται από :
- ELF Header
  - ✓ Beginning of ELF file
  - ✓ Holds a road map of file's organization
  - ✓ How to interpret the file, independent of the processor
- Program header table
  - ✓ Tells the system how to create a process image
  - ✓ Files used to build a process image (execute a program) must have a program header table
  - ✓ Re-locatable files do not need one
- Sections
  - ✓ Object file information for the linking view
  - ✓ Instructions, data, symbol table, relocation information, etc.

Linking & Execution Views



ELF Execution View




```
typedef struct {
    unsigned char e_ident[EI_NIDENT];
    Elf32_Half e_type;
    Elf32_Half e_machine;
    Elf32_Word e_version;
    Elf32_Addr e_entry;
    Elf32_Off  e_phoff;
    Elf32_Off  e_shoff;
    Elf32_Word e_flags;
    Elf32_Half e_ehsize;
    Elf32_Half e_phentsize;
    Elf32_Half e_phnum;
    Elf32_Half e_shentsize;
    Elf32_Half e_shnum;
    Elf32_Half e_shstrndx;
} Elf32_Ehdr;
```

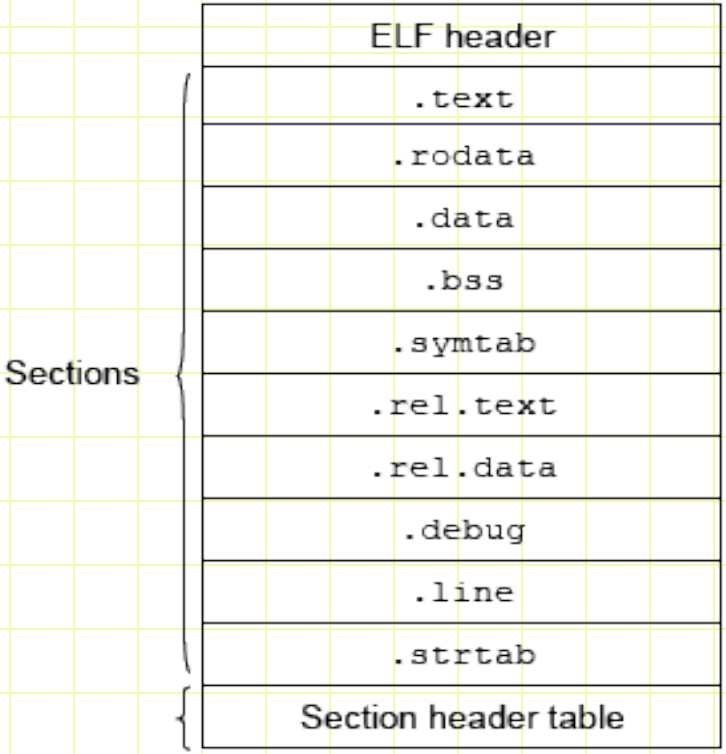
ELF Header

- Όλα τα ELF αρχεία περιέχουν μία επικεφαλίδα στην αρχή του αρχείου
  - ✓ Determines whether the file is an ELF file, whether it is in big/little endian format, the target processor, offsets to the program header table and/or section header table…
- Format of the ELF header

```
#define EI_NIDENT 16
typedeμf struct {
unsigned char e_ident[EI_NIDENT]; // file info (object file or not)
        Elf32_Half e_type; // type of file (relocatable, executable, etc.)
        Elf32_Half e_machine; // target processor (Intel x86, ARM, SPARC etc.)
        Elf32_Word e_version; / version # (to allow for future versions of ELF)
        Elf32_Addr e_entry; // program entry point (0 if no entry point)
        Elf32_Off e_phoff; // offset of program header (in bytes)
        Elf32_Off e_shoff; // offset of section header table
        Elf32_Word e_flags; // processor-specific flags
        Elf32_Half e_ehsize; // ELF header's size
        Elf32_Half e_phentsize; // entry size in pgm header tbl
        Elf32_Half e_phnum; // # of entries in pgm header
        Elf32_Half e_shentsize; // entry size in sec header tbl
        Elf32_Half e_shnum; // # of entries in sec header tbl
        Elf32_Half e_shstrndx; // sec header tbl index of str tbl
} Elf32_Ehdr;
```

ELF Sections

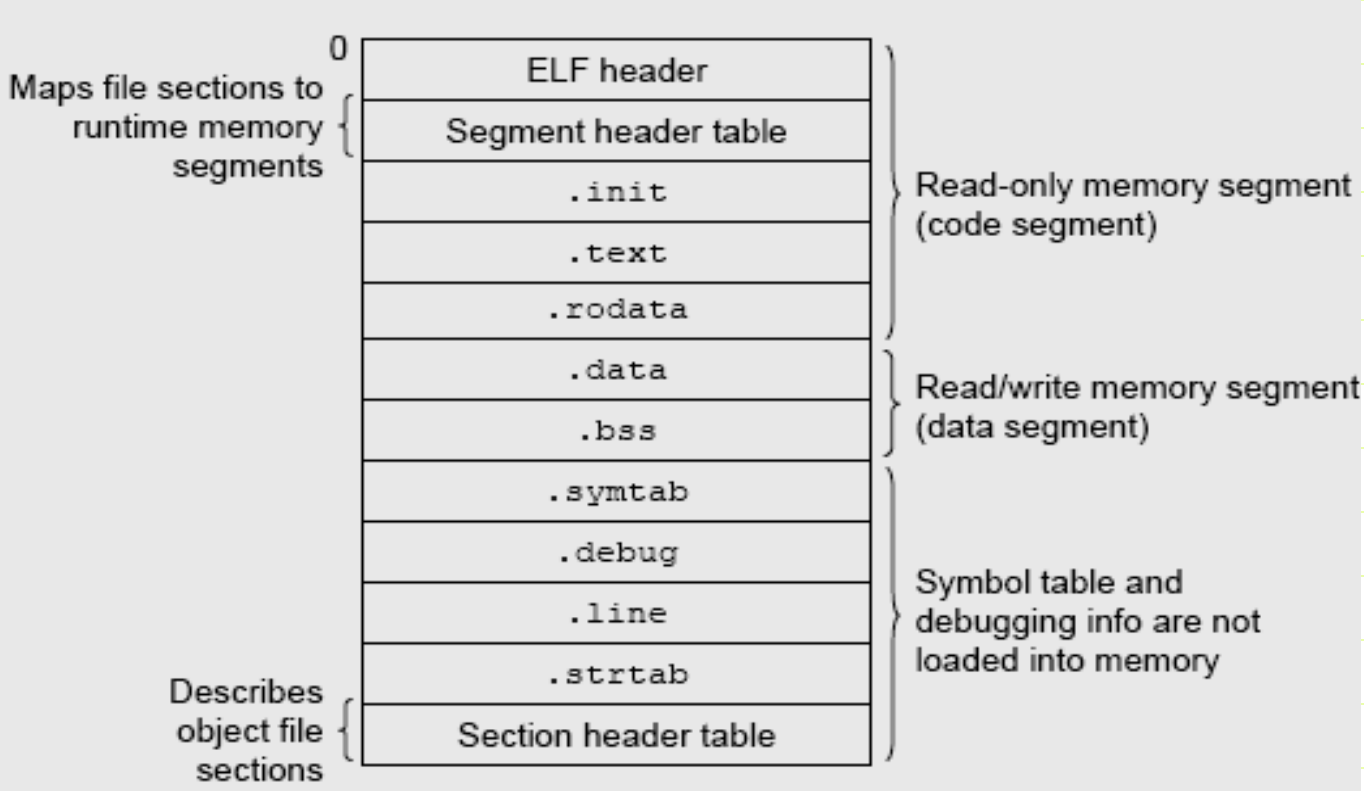
- Relocatable files must have a section header table
- Locations and size of sections are described by the section header table



Περιγραφή των διαφόρων Τμημάτων

- .text: program instructions and literal data
- .rodata: Read-only data such as the format strings in printf statements
- .data: initialized global data
- .bss: un-initialized global data (set to zero when program image is created)
  - ✓ This section does not occupy any space in the object file
- .symtab: this section holds the symbol table information
  - ✓ All global variables and functions that are defined and referenced in the program
- .rel.text: list of locations in the .text section that will need to be modified when linker combines this object files with others
- .rel.data: relocation information for any global variables that are referenced or defined in a module
- .debug: debugging information (present only if code is compiled to produce debug information)
- .line: mapping between line numbers in C program and machine code instructions (present only if code is compiled to produce debug information)
- .strtab: string table for symbols defined in .symtab and .debug sections

Executable Object Files



ELF Program Header

- Executable ELF files must have a program header table
  - ✓ The program header table is used to load the program (called “creating program image”)
  - ✓ Each segment has its own entry in the program header table
    - o e_*phnum* in ELF Header holds the number of program header entries
  - ✓ All program header entries have the same size (*e_phentsize* in ELF header)
- Program header entry for each segment

- What happens if p_memsz > p_filesz?
  - ✓ The remaining bytes (p_memsz-p_filesz) are initialized with 0

```
typedef struct {
        Elf32_Word p_type;      // type of segment – loadable, dll,...
        Elf32_Off p_offset;     // offset in bytes from the start of file
        Elf32_Addr p_vaddr;     // virtual address in memory of segment
        Elf32_Addr p_paddr;     // physical address in memory of segment
        Elf32_Word p_filesz;    // number of bytes in the file of the segment
        Elf32_Word p_memsz;     // number of bytes in memory of the process
                                // image of the segment
        Elf32_Word p_flags;     // indicates whether segment is executable
        Elf32_Word p_align;     // alignment information
} Elf32_Phdr;
```