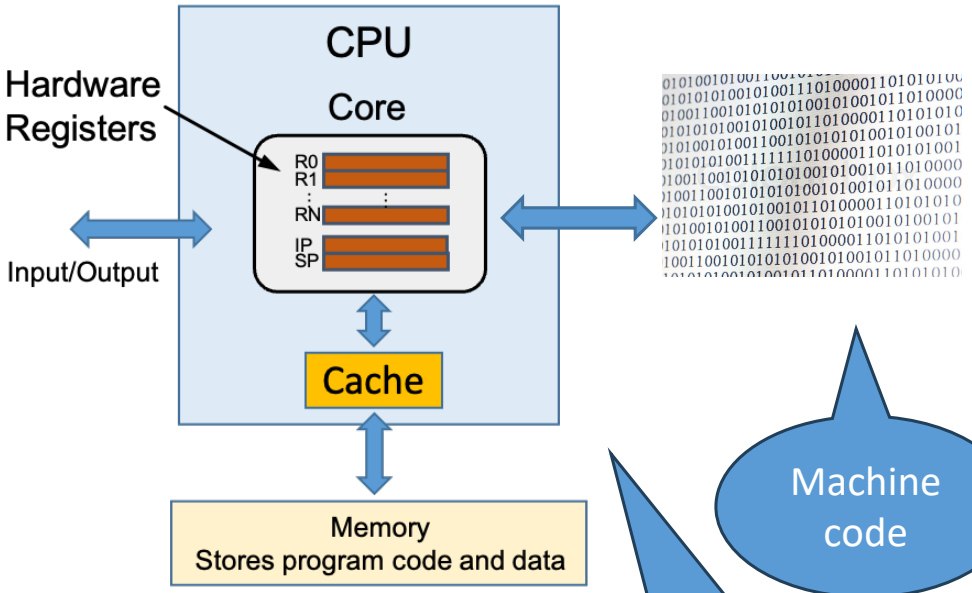
Lecture 02: Source Code to Machine Code

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Today's Class

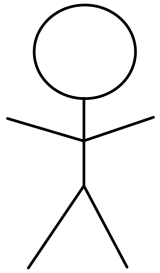
- Compilation steps
- Linking
- Introduction to Executable File Format (ELF)

Hardware and Software



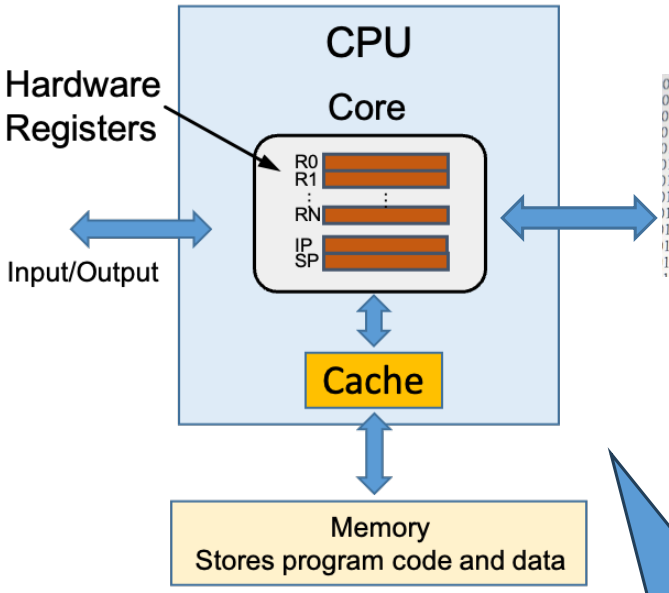
Machine code

I only understand 0s and 1s

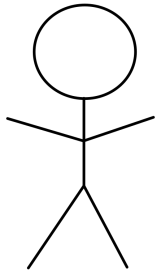


I can only talk to a processor in some human readable format. Afterall, I am human

Hardware and Software



```
movl    8(%ebp), %eax
subl    $1, %eax
subl    $12, %esp
pushl   %eax
call    fib
addl    $16, %esp
movl    %eax, -16(%ebp)
movl    8(%ebp), %eax
subl    $2, %eax
subl    $12, %esp
pushl   %eax
call    fib
addl    $16, %esp
movl    %eax, -12(%ebp)
movl    -16(%ebp), %edx
movl    -12(%ebp), %eax
addl    %edx, %eax
```



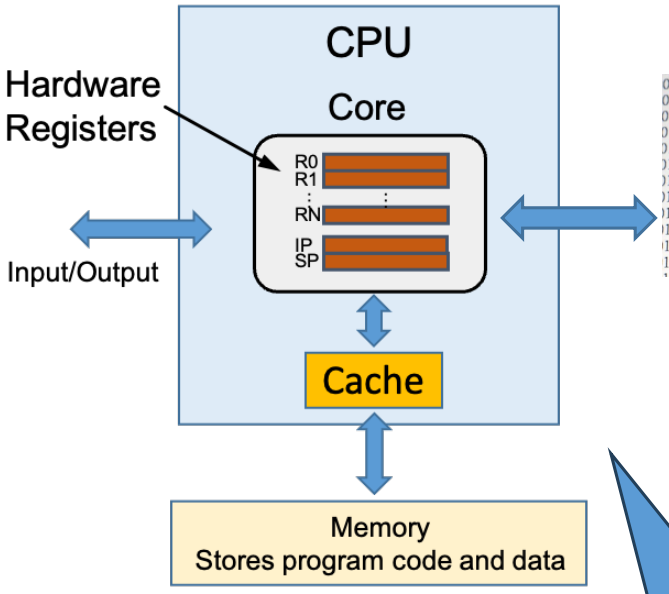
Machine code

I only understand 0s and 1s

Assembly code

It's better.. but Afterall, I am human

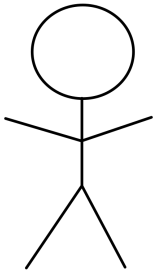
Hardware and Software



```
movl    8(%ebp), %eax
subl    $1, %eax
subl    $12, %esp
pushl   %eax
call    fib
addl    $16, %esp
movl    %eax, -16(%ebp)
movl    8(%ebp), %eax
subl    $2, %eax
subl    $12, %esp
pushl   %eax
call    fib
addl    $16, %esp
movl    %eax, -12(%ebp)
movl    -16(%ebp), %edx
movl    -12(%ebp), %eax
addl    %edx, %eax
```

C-program

```
uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x = fib(n-1);
    uint64_t y = fib(n-2);
    return (x + y);
  }
}
```



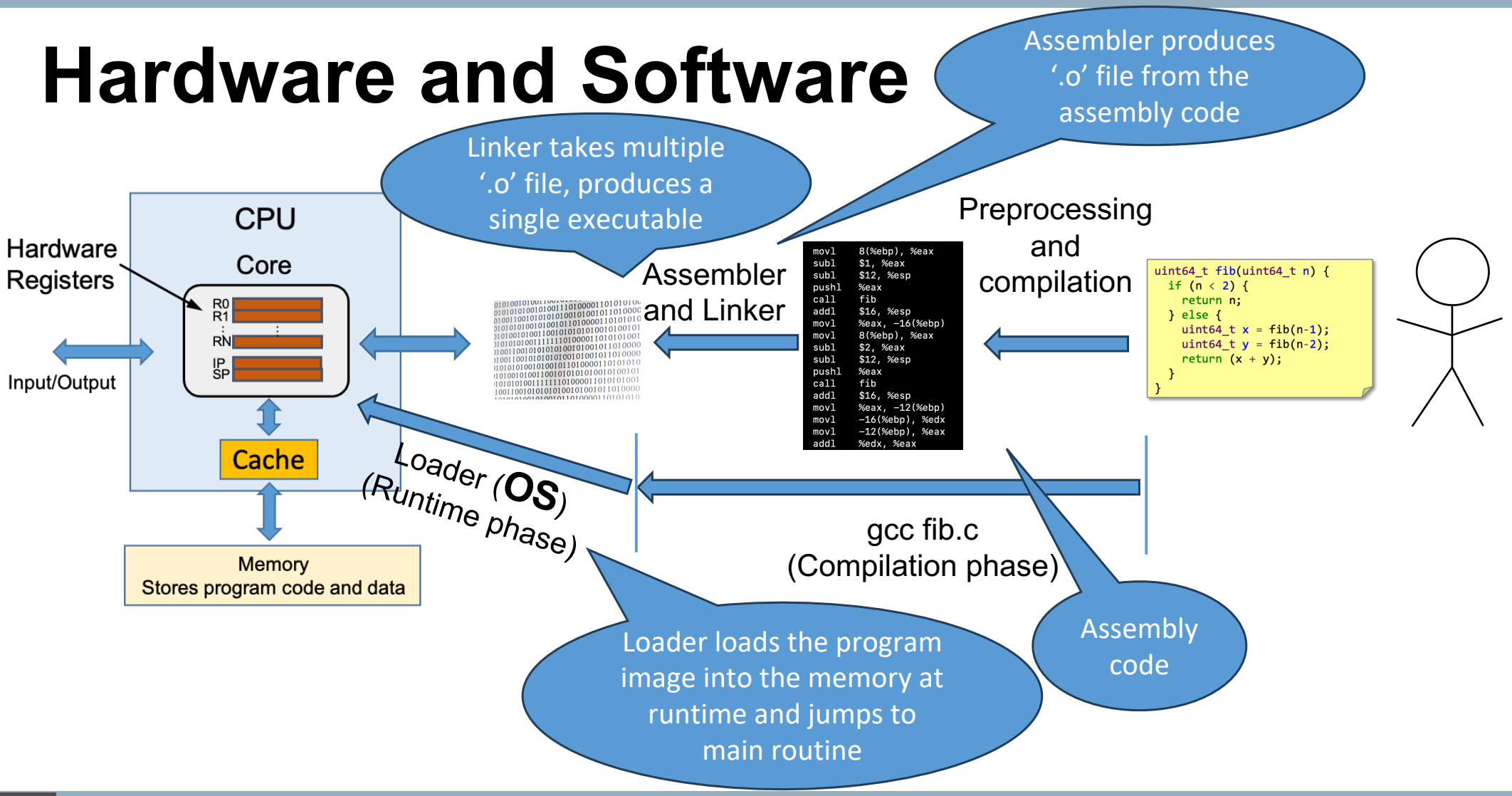
Machine code

I only understand 0s and 1s

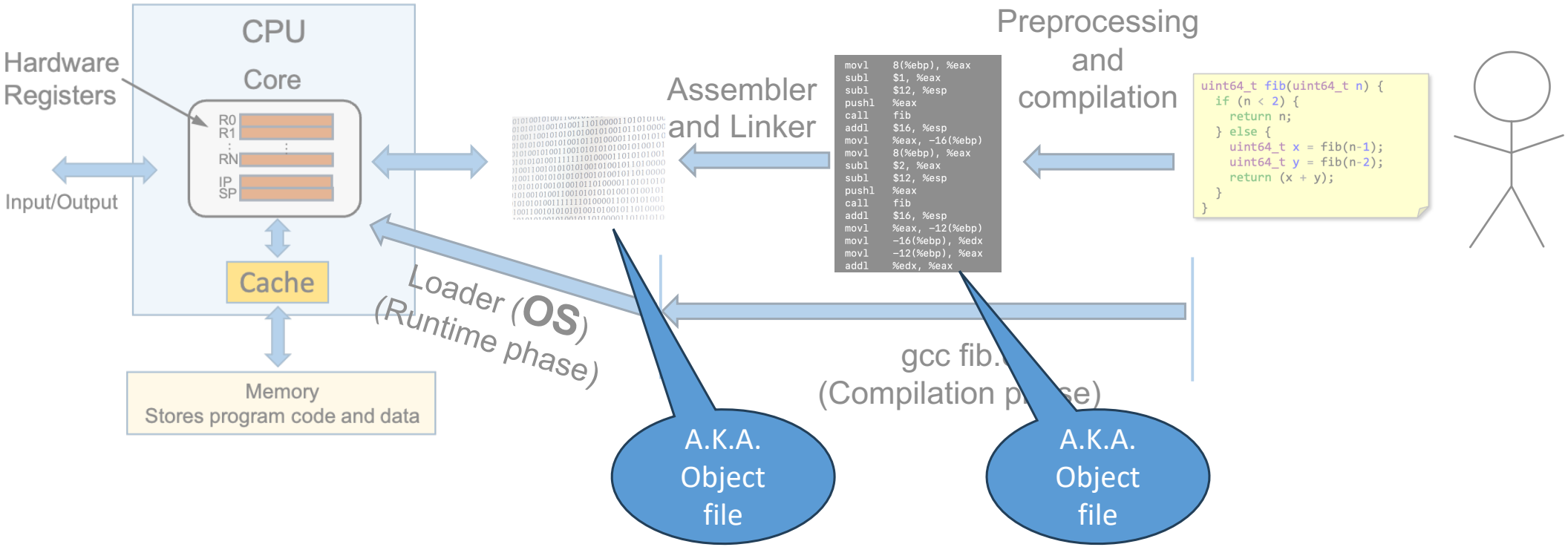
Assembly code

Wonderful!

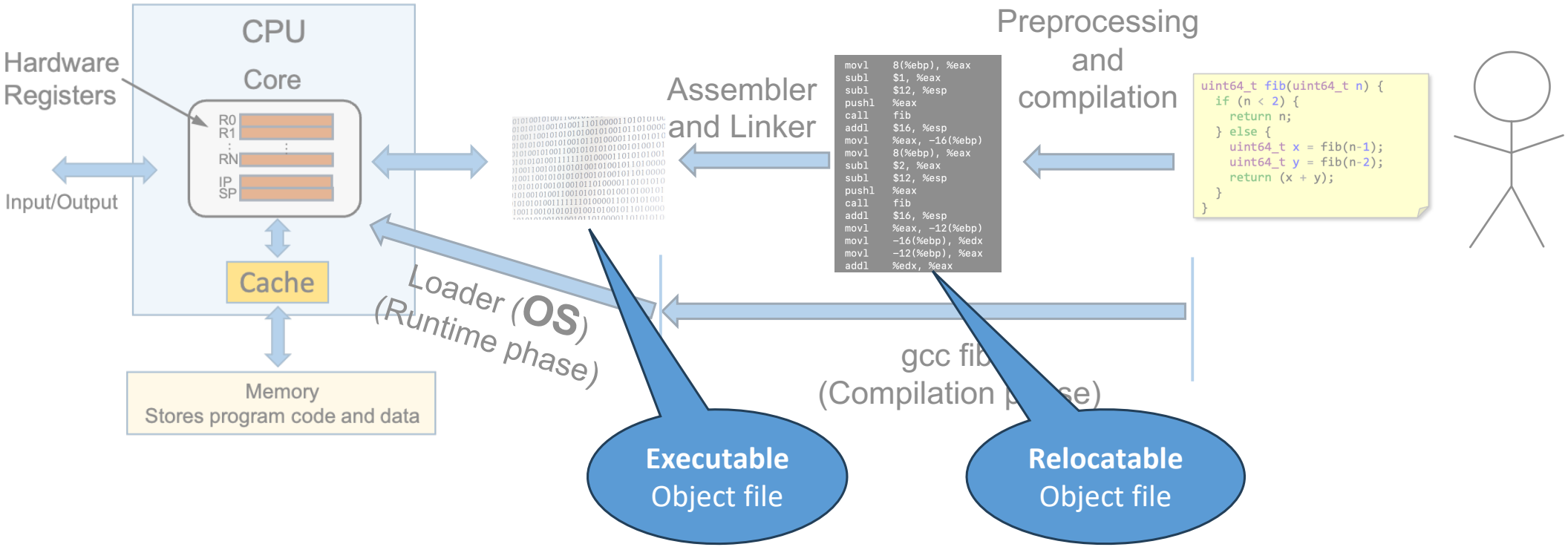
Hardware and Software



Hardware and Software



Hardware and Software



Relocatable Object Files

- Symbols (functions and variables) are not bound to any specific address
  - It will happens after linking
- Contains code and data suitable for linking with other object files to create the final executable or shared object

Executable Object Files

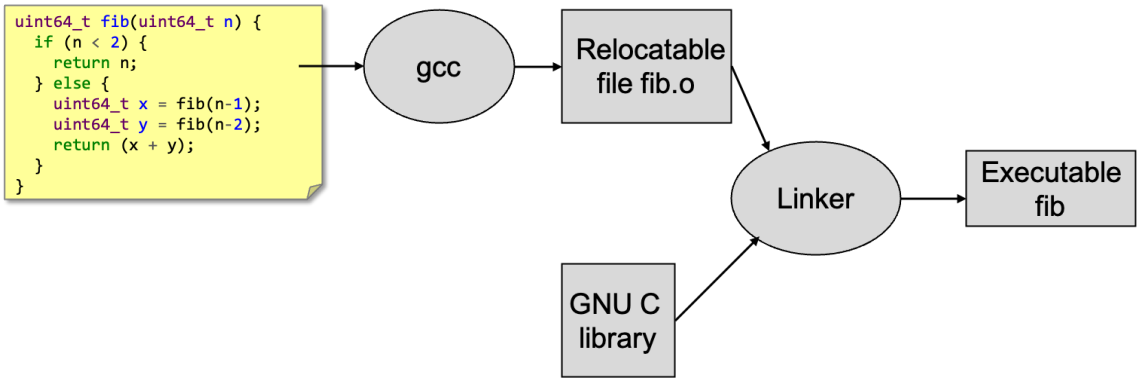
- It is similar to the relocatable file, but all symbols (functions and variables) have addresses resolved
- Contains all the information for the OS loader to run the program

Today's Class

- Compilation steps
- Linking
- Introduction to Executable File Format (ELF)

Linking

```
$ gcc -o fib fib.c     // compiling + linking
$ file fib
...ELF 64-bit shared object...dynamically linked
$ gcc -c fib.c         // compiling
$ file fib.o           // relocatable file
fib.o
$ gcc -o fib fib.o     // linking
```

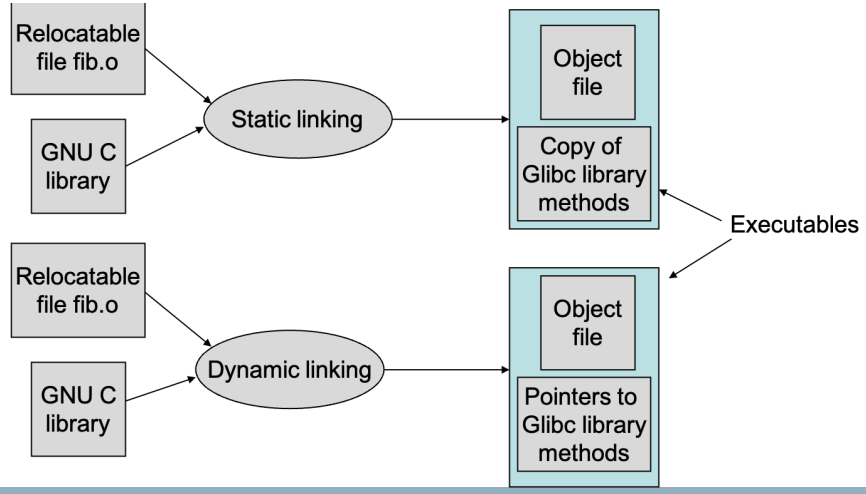


- Combines several relocatable files (.o) into a single executable
- Enable separate compilation of files
  - Changes in one file does not affect other files
    - Recompilation process is small

Static and Dynamic Linking

```
$ gcc -static -o fib fib.c
$ file fib
...ELF 64-bit executable...statically linked
```

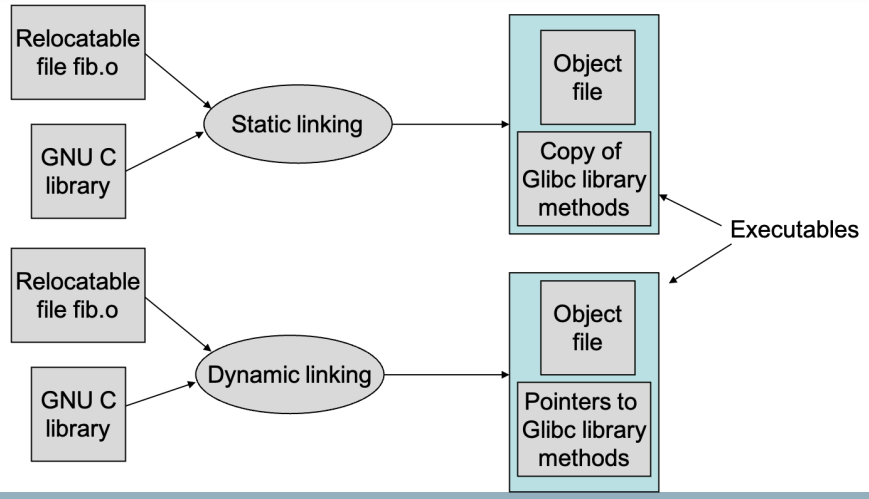
```
$ gcc -o fib fib.c
```



- Static linking
  - Each and every library modules referenced in the relocatable file is copied into the final executable
    - Static binding at compile time
- Dynamic linking
  - Final executable only contains references (pointers) to the library method instead of the copy of a library method
    - Binding with library done at runtime during execution

Static and Dynamic Linking

```
$ gcc -static -o fib fib.c
$ file fib
...ELF 64-bit executable...statically linked
$ ls -l fib
-rwxrwxr-x 1 vivek vivek 845304 Aug  6 10:26 fib
$ gcc -o fib fib.c
$ ls -l fib
-rwxrwxr-x 1 vivek vivek 8328 Aug  6 10:27 fib
```



- Static linking
  - Each and every library modules referenced in the relocatable file is copied into the final executable
    - Static binding at compile time
- Dynamic linking
  - Final executable only contains references (pointers) to the library method instead of the copy of a library method
    - Binding with library done at runtime during execution
- Which is better?

Dynamic Linking

- Advantages
  - Modular programming
    - If library requires changes/recompilation, the executables referring it does not need recompilation
  - Saves disk space
    - Executable just contain references
    - Multiple executables can access the same **.so** at runtime
- Disadvantage
  - What if the shared library changes or is missing at runtime?

Today's Class

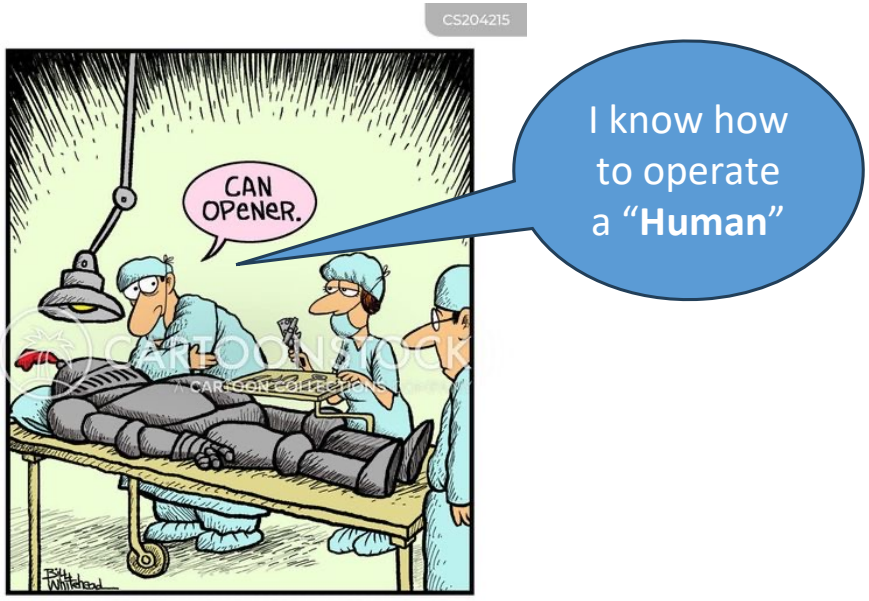
- Compilation steps
- Linking
- Introduction to Executable File Format (ELF)

Executable and Linkable Format (ELF)

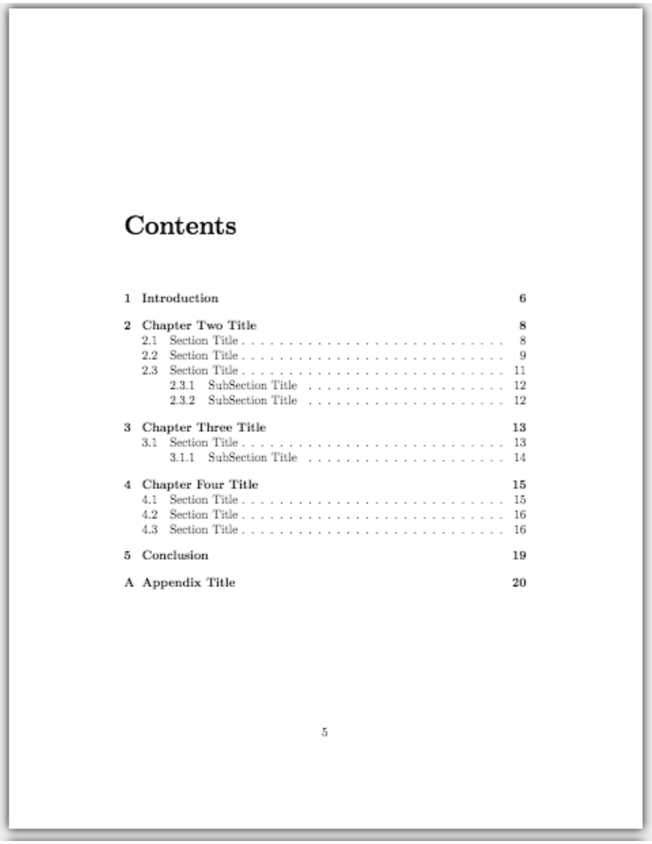
- Standard **format** for binary files (object files and core dumps) on Linux based OS
  - Contains all the information required by the OS to run the program
  - Platform independent format
  - A standard format eases the process of dynamic linking, loading and runtime control on a program
- Microsoft Windows OS uses Portable Executable (PE)
- IBM AIX OS uses Extended Common Object File Format (X-COFF)

Binary File Format

- Having a standard format for binary files helps linker and loaders in carrying out its operations easily

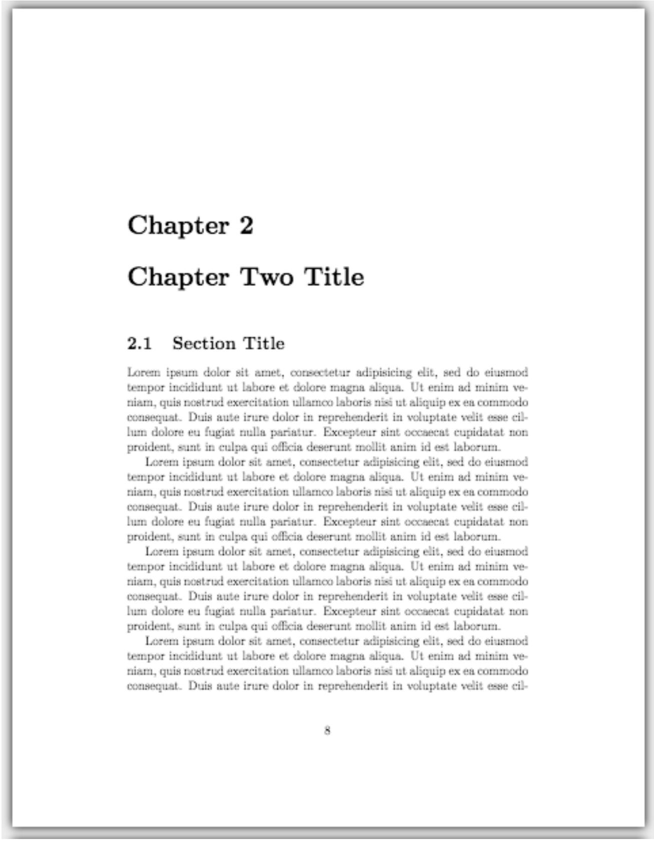


ELF Motivation (1/2)



Contents

5



Chapter 2

Chapter Two Title

2.1 Section Title

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

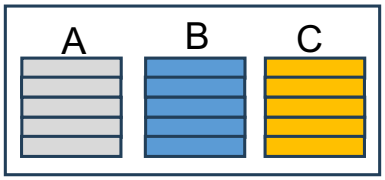
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cil-

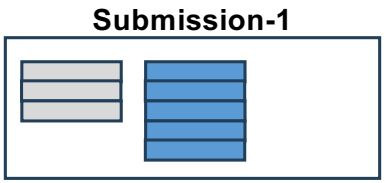
8

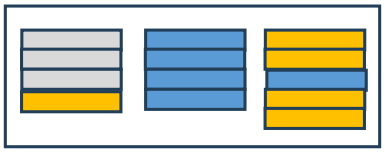
- Assume you are doing some research that you have to publish
- You have put a lot of effort, but can you just write as you like?
- There are certain rules/format for writing a research work

ELF Motivation (2/2)



Expectations: You would submit the document in this layout

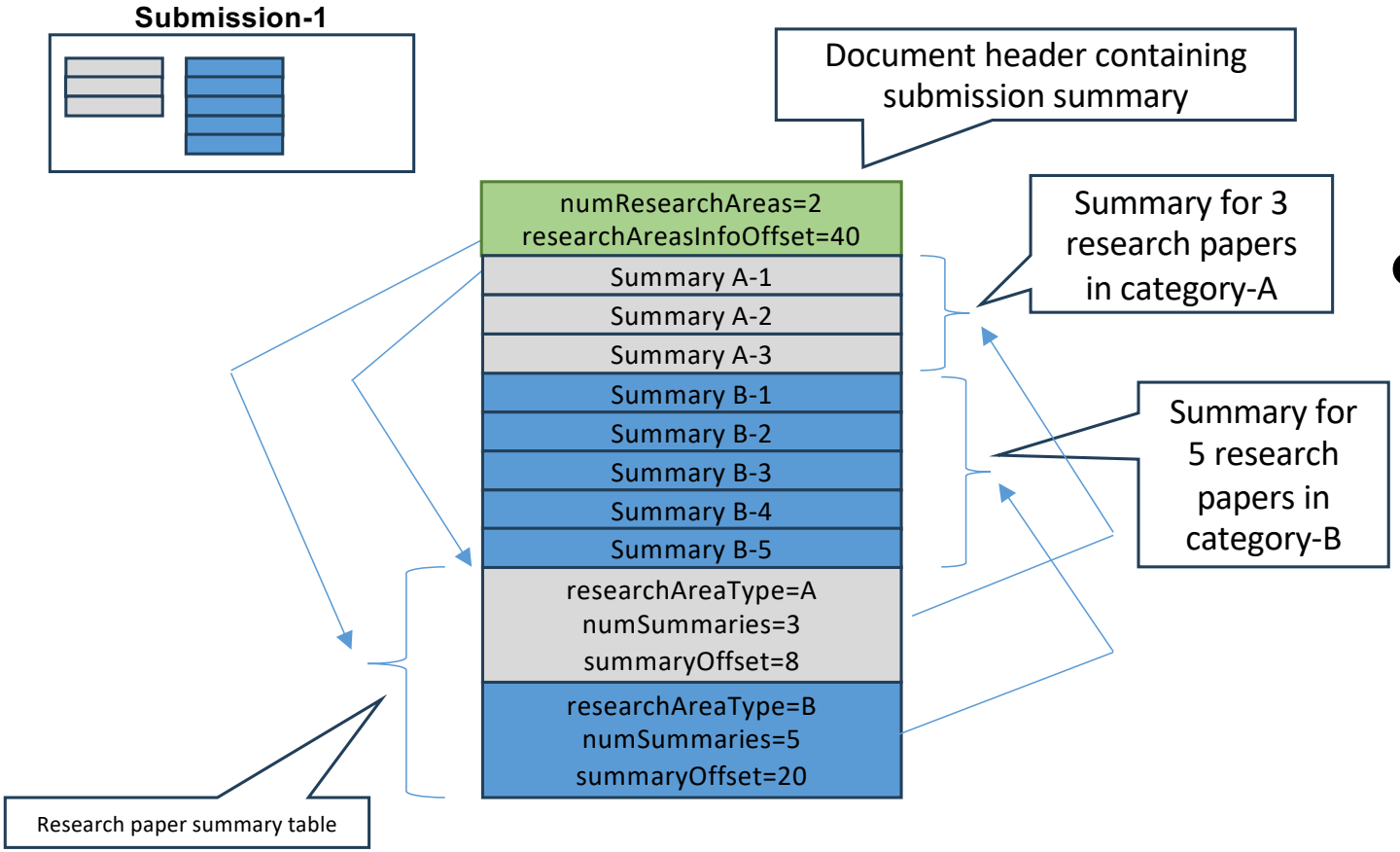




Reality: Different styles and combinations followed by students

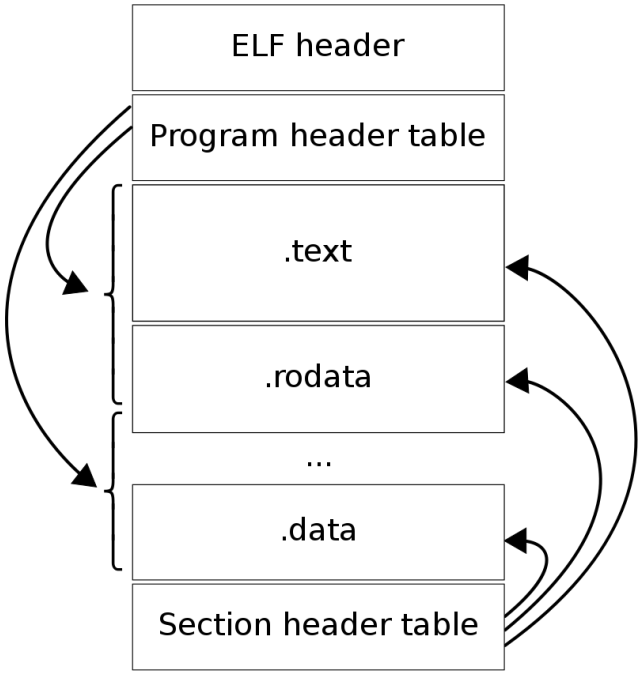
- Assume that you have an assignment to provide a single line summary of 5 articles from 3 different topics A, B and C
  - Total 15 articles

ELF Motivation (2/2)



- I propose a specific layout to overcome this challenge and to make it easy to understand your document
- I can easily "**load**" the student "**file**" from any number of students in my "**memory**" if they follow a standard "**format**" to store information, e.g., as the one shown below for the **Submission-1** shown here

Executable and Linkable Format (ELF)



Note: For simplicity, we will only discuss ELF format for 32-bit object files

Image source: https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

- ELF header
  - Provides a roadmap for the entire file organization
    - Always at offset zero of the object file
    - Provides entry point address for execution
- Two views of an ELF file
  - Linkable view (relocatable file)
  - Execution view (executable file, shared object)
- Linkable view
  - Section header table
    - One section header for each section
  - Sections
    - Contains data required for linking
    - Machine code, global variables, (initialized and un-initialized), symbol tables, line mapping between machine code and original C code, etc.
- Execution (or Loader) view
  - Program header table
    - One program header for each segment
  - Segments
    - Created by merging several sections
    - Contains information required for by the loader for execution
- Contiguous chunk of memory (ELF header, PHT, SHT, each section, each segment)

ELF Header

```
#define EI_NIDENT 16

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;
```

```
vivek@possum:~/os23$ readelf -h ./a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1160
  Start of program headers:          64 (bytes into file)
  Start of section headers:          14320 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

Image source: https://refspecs.linuxfoundation.org/elf/gabi4+/ch4.eheader.html

ELF Header

```
#define EI_NIDENT 16

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;
```

Identification for object file

Object file type

Entry point address for starting the executable

Program Header Table (PHT) file offset

Section Header Table (SHT) file offset

ELF header size in bytes (fixed)

Size in bytes of each entry in PHT

Total number of entries in PHT

Size in bytes of each entry in SHT

Total number of entries in SHT

Index of the section called as "String Table"

Image source: https://refspecs.linuxfoundation.org/elf/gabi4+/ch4.eheader.html

Entry Point?

```
vivek@possum:~/elf_os-12$ readelf -h ./a.out
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x3e0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          6060 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           40 (bytes)
  Number of section headers:         29
  Section header string table index: 28
```

Entry Point?

```
vivek@possum:~/elf_os-12$ readelf -h ./a.out
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x3e0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          6060 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           40 (bytes)
  Number of section headers:         29
  Section header string table index: 28
vivek@possum:~/elf_os-12$ objdump -d ./a.out | grep -A16 3e0
000003e0 <_start>:
 3e0:   31 ed                   xor    %ebp,%ebp
 3e2:   5e                      pop    %esi
 3e3:   89 e1                   mov    %esp,%ecx
 3e5:   83 e4 f0                and    $0xfffffff0,%esp
 3e8:   50                      push   %eax
 3e9:   54                      push   %esp
 3ea:   52                      push   %edx
 3eb:   e8 22 00 00 00          call   412 <_start+0x32>
 3f0:   81 c3 e8 1b 00 00       add    $0x1be8,%ebx
 3f6:   8d 83 f8 e5 ff ff       lea    -0x1a08(%ebx),%eax
 3fc:   50                      push   %eax
 3fd:   8d 83 98 e5 ff ff       lea    -0x1a68(%ebx),%eax
 403:   50                      push   %eax
 404:   51                      push   %ecx
 405:   56                      push   %esi
 406:   ff b3 20 00 00 00       pushl  0x20(%ebx)
 40c:   e8 af ff ff ff          call   3c0 <__libc_start_main@plt>
```

Entry point address

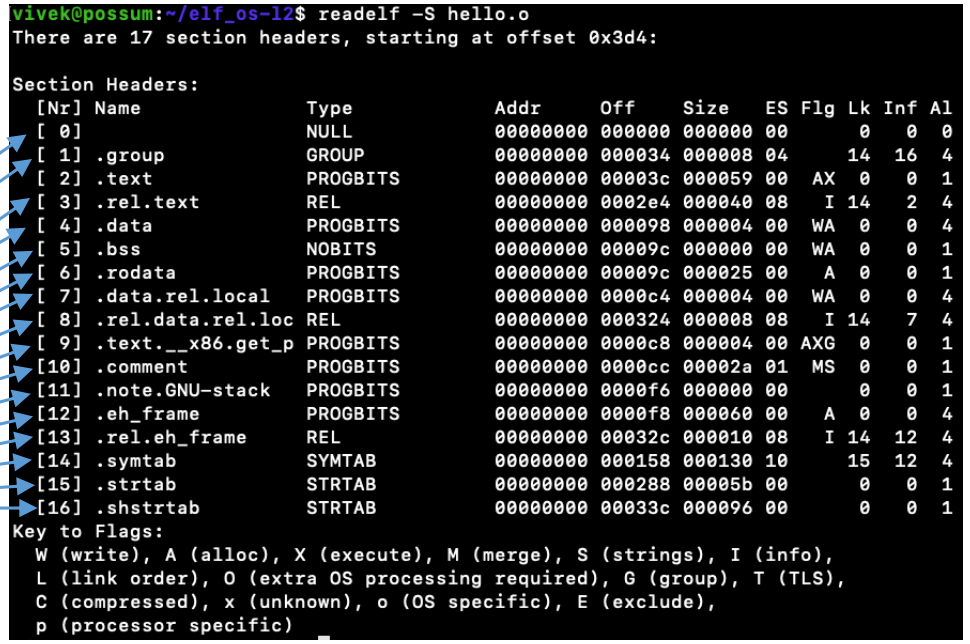
Entry point actual method

Entry point address

_start calls __libc_start_main with the address of user main as a parameter (invoked from here)

ELF Section Header

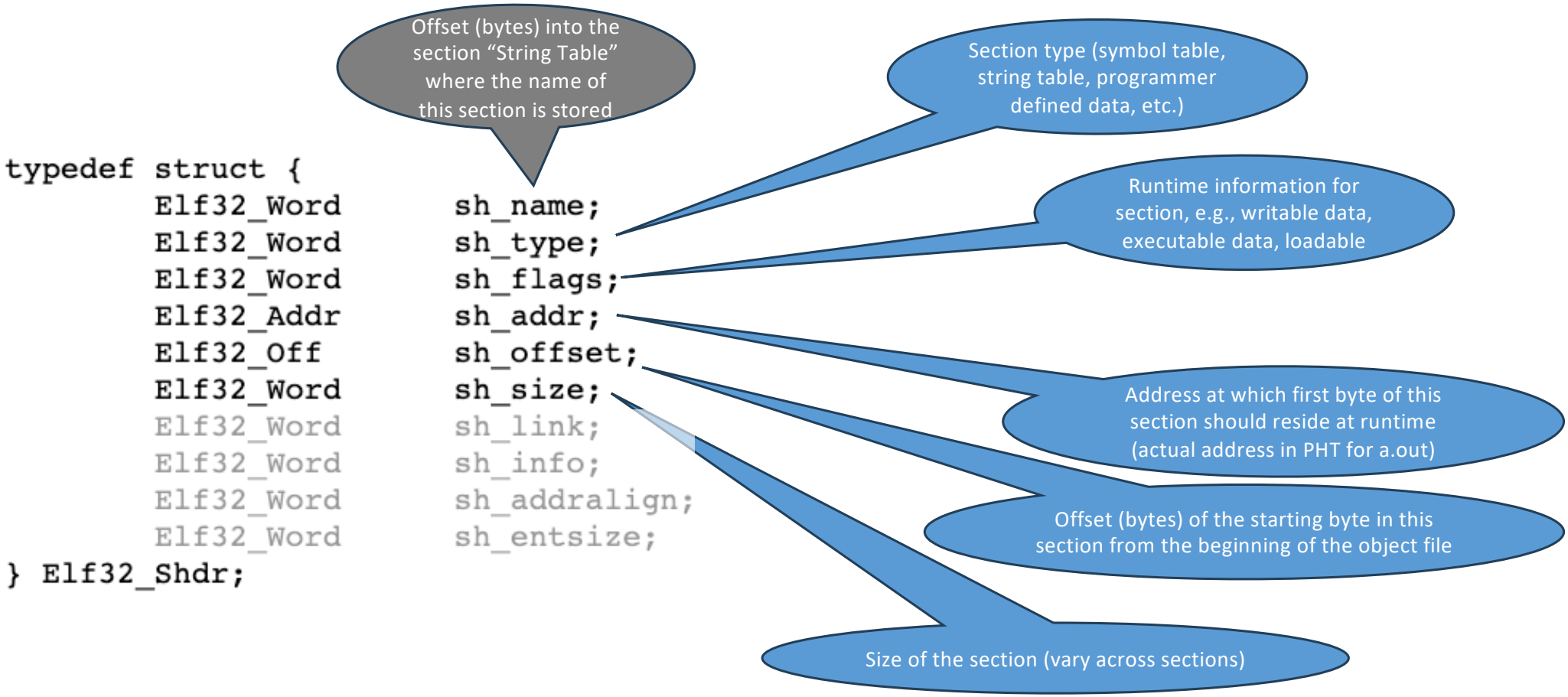
```
typedef struct {
        Elf32_Word      sh_name;
        Elf32_Word      sh_type;
        Elf32_Word      sh_flags;
        Elf32_Addr      sh_addr;
        Elf32_Off       sh_offset;
        Elf32_Word      sh_size;
        Elf32_Word      sh_link;
        Elf32_Word      sh_info;
        Elf32_Word      sh_addralign;
        Elf32_Word      sh_entsize;
} Elf32_Shdr;
```



```
vivek@possum:~/elf_os-12$ readelf -S hello.o
There are 17 section headers, starting at offset 0x3d4:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .group            GROUP           00000000 000034 000008 04     14  16  4
  [ 2] .text             PROGBITS        00000000 00003c 000059 00  AX  0   0  1
  [ 3] .rel.text         REL             00000000 0002e4 000040 08   I 14   2  4
  [ 4] .data             PROGBITS        00000000 000098 000004 00  WA  0   0  4
  [ 5] .bss              NOBITS          00000000 00009c 000000 00  WA  0   0  1
  [ 6] .rodata           PROGBITS        00000000 00009c 000025 00   A  0   0  1
  [ 7] .data.rel.local   PROGBITS        00000000 0000c4 000004 00  WA  0   0  4
  [ 8] .rel.data.rel.loc REL             00000000 000324 000008 08   I 14   7  4
  [ 9] .text.__x86.get_p PROGBITS        00000000 0000c8 000004 00 AXG  0   0  1
  [10] .comment          PROGBITS        00000000 0000cc 00002a 01  MS  0   0  1
  [11] .note.GNU-stack   PROGBITS        00000000 0000f6 000000 00      0   0  1
  [12] .eh_frame         PROGBITS        00000000 0000f8 000060 00   A  0   0  4
  [13] .rel.eh_frame     REL             00000000 00032c 000010 08   I 14  12  4
  [14] .symtab           SYMTAB          00000000 000158 000130 10     15  12  4
  [15] .strtab           STRTAB          00000000 000288 00005b 00      0   0  1
  [16] .shstrtab         STRTAB          00000000 00033c 000096 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

ELF Section Header



Sections at Linking and Executable

```
vivek@possum:~/elf_os-12$ readelf -S hello.o
There are 17 section headers, starting at offset 0x3d4:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .group            GROUP           00000000 000034 000008 04     14  16  4
  [ 2] .text             PROGBITS        00000000 00003c 000059 00  AX  0   0  1
  [ 3] .rel.text         REL             00000000 0002e4 000040 08   I 14   2  4
  [ 4] .data             PROGBITS        00000000 000098 000004 00  WA  0   0  4
  [ 5] .bss              NOBITS          00000000 00009c 000000 00  WA  0   0  1
  [ 6] .rodata           PROGBITS        00000000 00009c 000025 00   A  0   0  1
  [ 7] .data.rel.local   PROGBITS        00000000 0000c4 000004 00  WA  0   0  4
  [ 8] .rel.data.rel.loc REL             00000000 000324 000008 08   I 14   7  4
  [ 9] .text.__x86.get_p PROGBITS        00000000 0000c8 000004 00 AXG  0   0  1
  [10] .comment          PROGBITS        00000000 0000cc 00002a 01  MS  0   0  1
  [11] .note.GNU-stack   PROGBITS        00000000 0000f6 000000 00      0   0  1
  [12] .eh_frame         PROGBITS        00000000 0000f8 000060 00   A  0   0  4
  [13] .rel.eh_frame     REL             00000000 00032c 000010 08   I 14  12  4
  [14] .symtab           SYMTAB          00000000 000158 000130 10     15  12  4
  [15] .strtab           STRTAB          00000000 000288 00005b 00      0   0  1
  [16] .shstrtab         STRTAB          00000000 00033c 000096 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

Q: Why there are more number of sections in executable v/s relocatable file?

A: To assist the dynamic loader, the linker adds several more sections in the executable than in the relocatable

```
vivek@possum:~/elf_os-12$ readelf -S hello
There are 29 section headers, starting at offset 0x17fc:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        00000154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            00000168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            00000188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        000001ac 0001ac 000020 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          000001cc 0001cc 000080 10   A  6   1  4
  [ 6] .dynstr           STRTAB          0000024c 00024c 00009d 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          000002ea 0002ea 000010 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         000002fc 0002fc 000030 00   A  6   1  4
  [ 9] .rel.dyn          REL             0000032c 00032c 000048 08   A  5   0  4
  [10] .rel.plt          REL             00000374 000374 000010 08  AI  5  22  4
  [11] .init             PROGBITS        00000384 000384 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        000003b0 0003b0 000030 04  AX  0   0 16
  [13] .plt.got          PROGBITS        000003e0 0003e0 000010 08  AX  0   0  8
  [14] .text             PROGBITS        000003f0 0003f0 000202 00  AX  0   0 16
  [15] .fini             PROGBITS        000005f4 0005f4 000014 00  AX  0   0  4
  [16] .rodata           PROGBITS        00000608 000608 00002d 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        00000638 000638 00003c 00   A  0   0  4
  [18] .eh_frame         PROGBITS        00000674 000674 0000fc 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      00001ed8 000ed8 000004 04  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      00001edc 000edc 000004 04  WA  0   0  4
  [21] .dynamic          DYNAMIC         00001ee0 000ee0 0000f8 08  WA  6   0  4
  [22] .got              PROGBITS        00001fd8 000fd8 000028 04  WA  0   0  4
  [23] .data             PROGBITS        00002000 001000 000010 00  WA  0   0  4
  [24] .bss              NOBITS          00002020 001010 400020 00  WA  0   0 32
  [25] .comment          PROGBITS        00000000 001010 000029 01  MS  0   0  1
  [26] .symtab           SYMTAB          00000000 00103c 000460 10     27  43  4
  [27] .strtab           STRTAB          00000000 00149c 000264 00      0   0  1
  [28] .shstrtab         STRTAB          00000000 001700 0000fc 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

Few Interesting Sections

```
vivek@possum:~/elf_os-12$ readelf -S hello
There are 29 section headers, starting at offset 0x17fc:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        00000154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            00000168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            00000188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        000001ac 0001ac 000020 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          000001cc 0001cc 000080 10   A  6   1  4
  [ 6] .dynstr           STRTAB          0000024c 00024c 00009d 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          000002ea 0002ea 000010 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         000002fc 0002fc 000030 00   A  6   1  4
  [ 9] .rel.dyn          REL             0000032c 00032c 000048 08   A  5   0  4
  [10] .rel.plt          REL             00000374 000374 000010 08  AI  5  22  4
  [11] .init             PROGBITS        00000384 000384 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        000003b0 0003b0 000030 04  AX  0   0 16
  [13] .plt.got          PROGBITS        000003e0 0003e0 000010 08  AX  0   0  8
  [14] .text             PROGBITS        000003f0 0003f0 000202 00  AX  0   0 16
  [15] .fini             PROGBITS        000005f4 0005f4 000014 00  AX  0   0  4
  [16] .rodata           PROGBITS        00000608 000608 00002d 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        00000638 000638 00003c 00   A  0   0  4
  [18] .eh_frame         PROGBITS        00000674 000674 0000fc 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      00001ed8 000ed8 000004 04  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      00001edc 000edc 000004 04  WA  0   0  4
  [21] .dynamic          DYNAMIC         00001ee0 000ee0 0000f8 08  WA  6   0  4
  [22] .got              PROGBITS        00001fd8 000fd8 000028 04  WA  0   0  4
  [23] .data             PROGBITS        00002000 001000 000010 00  WA  0   0  4
  [24] .bss              NOBITS          00002020 001010 400020 00  WA  0   0 32
  [25] .comment          PROGBITS        00000000 001010 000029 01  MS  0   0  1
  [26] .symtab           SYMTAB          00000000 00103c 000460 10     27  43  4
  [27] .strtab           STRTAB          00000000 00149c 000264 00      0   0  1
  [28] .shstrtab         STRTAB          00000000 001700 0000fc 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

```
vivek@possum:~/elf_os-12$ cat hello.c
#include<stdio.h>

int my_global1 = 1, my_global2[1024*1024];
char* str = "Updated value of my_global1";

int main() {
  my_global1 += 1;
  printf("%s = %d\n", str, my_global1);
  return 0;
}
```

String Table Section (".shstrtab")

```
vivek@possum:~/elf_os-12$ readelf -p .shstrtab hello

String dump of section '.shstrtab':
  [     1]  .symtab
  [     9]  .strtab
  [    11]  .shstrtab
  [    1b]  .interp
  [    23]  .note.ABI-tag
  [    31]  .note.gnu.build-id
  [    44]  .gnu.hash
  [    4e]  .dynsym
  [    56]  .dynstr
  [    5e]  .gnu.version
  [    6b]  .gnu.version_r
  [    7a]  .rel.dyn
  [    83]  .rel.plt
  [    8c]  .init
  [    92]  .plt.got
  [    9b]  .text
  [    a1]  .fini
  [    a7]  .rodata
  [    af]  .eh_frame_hdr
  [    bd]  .eh_frame
  [    c7]  .init_array
  [    d3]  .fini_array
  [    df]  .dynamic
  [    e8]  .data
  [    ee]  .bss
  [    f3]  .comment
```

- As with other sections, this section is also a contiguous chunk of memory
- Stores the name of each sections
  - String (NULL terminated)

Executable Instructions (".text")

```
vivek@possum:~/elf_os-l2$ objdump -d hello.o | grep -A20 text
Disassembly of section .text:

00000000 <main>:
   0:   8d 4c 24 04             lea    0x4(%esp),%ecx
   4:   83 e4 f0                and    $0xfffffff0,%esp
   7:   ff 71 fc                pushl  -0x4(%ecx)
   a:   55                      push   %ebp
   b:   89 e5                   mov    %esp,%ebp
   d:   53                      push   %ebx
   e:   51                      push   %ecx
   f:   e8 fc ff ff ff          call   10 <main+0x10>
  14:   05 01 00 00 00          add    $0x1,%eax
  19:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
  1f:   83 c2 01                add    $0x1,%edx
  22:   89 90 00 00 00 00       mov    %edx,0x0(%eax)
  28:   8b 88 00 00 00 00       mov    0x0(%eax),%ecx
  2e:   8b 90 00 00 00 00       mov    0x0(%eax),%edx
  34:   83 ec 04                sub    $0x4,%esp
  37:   51                      push   %ecx
  38:   52                      push   %edx
  39:   8d 90 1c 00 00 00       lea    0x1c(%eax),%edx
```

```
vivek@possum:~/elf_os-l2$ objdump -d hello | grep -A20  text
Disassembly of section .text:

000003f0 <_start>:
 3f0:   31 ed                   xor    %ebp,%ebp
 3f2:   5e                      pop    %esi
 3f3:   89 e1                   mov    %esp,%ecx
 3f5:   83 e4 f0                and    $0xfffffff0,%esp
 3f8:   50                      push   %eax
 3f9:   54                      push   %esp
 3fa:   52                      push   %edx
 3fb:   e8 22 00 00 00          call   422 <_start+0x32>
 400:   81 c3 d8 1b 00 00       add    $0x1bd8,%ebx
 406:   8d 83 18 e6 ff ff       lea    -0x19e8(%ebx),%eax
 40c:   50                      push   %eax
 40d:   8d 83 b8 e5 ff ff       lea    -0x1a48(%ebx),%eax
 413:   50                      push   %eax
 414:   51                      push   %ecx
 415:   56                      push   %esi
 416:   ff b3 20 00 00 00       pushl  0x20(%ebx)
 41c:   e8 af ff ff ff          call   3d0 <__libc_start_main@
 421:   f4                      hlt
```

- Stores bytes corresponding to program execution (user code and the GNU C library supporting code)

Section ".rodata"

```
vivek@possum:~/elf_os-12$ cat hello.c
#include<stdio.h>

int my_global1 = 1, my_global2[1024*1024];
char* str = "Updated value of my_global1";

int main() {
  my_global1 += 1;
  printf("%s = %d\n", str, my_global1);
  return 0;
}
```

```
vivek@possum:~/elf_os-12$ readelf -p .rodata hello

String dump of section '.rodata':
  [     8]  Updated value of my_global1
  [    24]  %s = %d^J
```

- Stores all the string literals defined in the program
- Question
  - Would the content change across relocatable and executable?

Section ".data"

```
vivek@possum:~/elf_os-12$ cat hello.c
#include<stdio.h>

int my_global1 = 1, my_global2[1024*1024];
char* str = "Updated value of my_global1";

int main() {
  my_global1 += 1;
  printf("%s = %d\n", str, my_global1);
  return 0;
}
```

```
vivek@possum:~/elf_os-12$ readelf -S hello | grep '\.data \| Name'
  [Nr] Name              Type             Addr     Off    Size   ES Flg Lk Inf Al
  [23] .data             PROGBITS         00002000 001000 000010 00  WA  0   0  4
```

```
vivek@possum:~/elf_os-12$ readelf -S hello.o | grep '\.data \| Name'
  [Nr] Name              Type             Addr     Off    Size   ES Flg Lk Inf Al
  [ 4] .data             PROGBITS         00000000 000098 000004 00  WA  0   0  4
```

- Contains the **initialized value** of all the global and static variables from the user code (not the variable name)
  - In our running example it is the **value** of the variable "my_global1"
    - Why not the value of "str"?
- "WA" flag indicates that the section content is to be loaded at runtime in memory and is writable
- PROGBITS indicating it contains information defined by the user code
- Where is the name of the global variable stored?

Section ".bss"

```
vivek@possum:~/elf_os-12$ cat hello.c
#include<stdio.h>

int my_global1 = 1, my_global2[1024*1024];
char* str = "Updated value of my_global1";

int main() {
  my_global1 += 1;
  printf("%s = %d\n", str, my_global1);
  return 0;
}
```

Hex

```
vivek@possum:~/elf_os-12$ readelf -S hello | grep '\.bss\|Name'
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [24] .bss              NOBITS          00002020 001010 400020 00  WA  0   0 32
```

- .bss (Block Started by Symbol) section indicates the total memory to be allocated at runtime for uninitialized global/static variables
  - In our running example it is the **value** of the variable "my_global2"
- "WA" flag indicates that the section content is to be loaded at runtime in memory and is writable
- NOBITS indicating it relates to the user code, but it doesn't occupy any space in the object file
- Where is the name of the global variable stored?

Sections ".symtab" and ".strtab"

```
vivek@possum:~/elf_os-12$ readelf -p .symtab hello.o

String dump of section '.symtab':
  [     d4]
  [     da]  @
  [     f0]  #
  [     f8]  Y
  [    100]  (
  [    110]  >
  [    120]  T

vivek@possum:~/elf_os-12$ readelf -p .strtab hello.o

String dump of section '.strtab':
  [      1]  hello.c
  [      9]  my_global1
  [     14]  my_global2
  [     1f]  str
  [     23]  main
  [     28]  __x86.get_pc_thunk.ax
  [     3e]  _GLOBAL_OFFSET_TABLE_
  [     54]  printf
```

- ".symtab" is like a directory to map the name of the symbols (present inside .strtab) with the actual content (value) of that symbol
  - E.g., for the symbol "my_global1", symbol table will contain an index into the string table ".strtab" where the name "my_global1" is stored. It will also contain an index to the section header holding the value of "my_global1"
    - What will be that section name?

Segments in Executable File

```
vivek@possum:~/elf_os-12$ readelf -l hello

Elf file type is DYN (Shared object file)
Entry point 0x3f0
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00120 0x00120 R   0x4
  INTERP         0x000154 0x00000154 0x00000154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x00000000 0x00000000 0x00770 0x00770 R E 0x1000
  LOAD           0x000ed8 0x00001ed8 0x00001ed8 0x00138 0x400168 RW  0x1000
  DYNAMIC        0x000ee0 0x00001ee0 0x00001ee0 0x000f8 0x000f8 RW  0x4
  NOTE           0x000168 0x00000168 0x00000168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x000638 0x00000638 0x00000638 0x0003c 0x0003c R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000ed8 0x00001ed8 0x00001ed8 0x00128 0x00128 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .
gnu.version .gnu.version_r .rel.dyn .rel.plt .init .plt .plt.got .text .fini .
rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .dynamic .got .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
```

- Segments are created by merging several sections and each segment contains a specific type of information required for by the loader for execution (i.e., all the sections inside a segment are related)
- Not available in relocatable file

Reading Materials

- Executable and Linkable Format
  - https://www.cs.cmu.edu/afs/cs/academic/class/15213-f00/docs/elf.pdf
  - https://man7.org/linux/man-pages/man5/elf.5.html

Next Lecture

- Dissecting an ELF file
  - It will be the topic for your Assignment-1
    - Will be released on 23rd August