

CS2100

<http://www.comp.nus.edu.sg/~cs2100/>

COMPUTER ORGANISATION

Lecture #6

C for Hardware Programming



NUS
National University
of Singapore

School of
Computing

Lecture #6: C for Hardware Programming

1. Overview and Motivation
2. Hardware Programming: Concerns
3. C and Hardware Programming
4. Code Compilation Process
5. Debugging – GDB

1. Overview and Motivation

- C is a high-level language which is widely used to program microcontrollers and single board computers
- Capability to address hardware memory locations
- Capability to address hardware enhancements by the vendor
- Another extension of the C language to support certain exclusive features required in embedded systems is Embedded C
- Here, we briefly introduce C based hardware programming and not embedded C

2. Hardware Programming: Concerns

- Code speed
 - Timing constraints
 - Slow processor compared to desktop processors
- Code size
 - Limited memory
- Programming methods

Machine code (0,1)

- High difficulty
- Almost no readability

Assembly language (ADD, SLL, etc.)

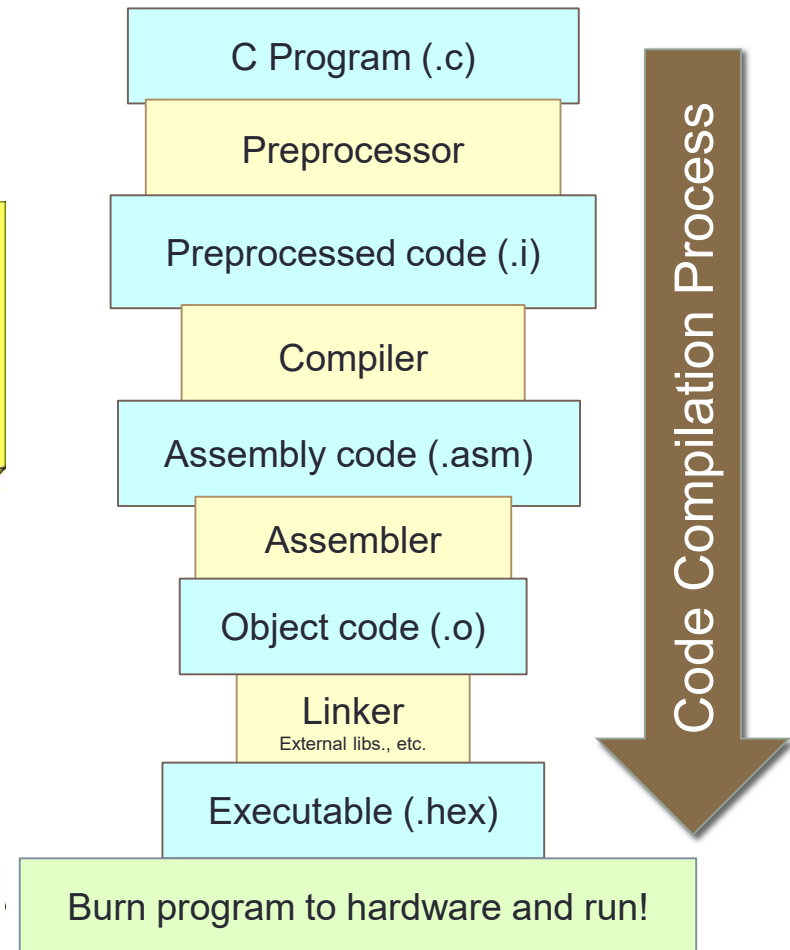
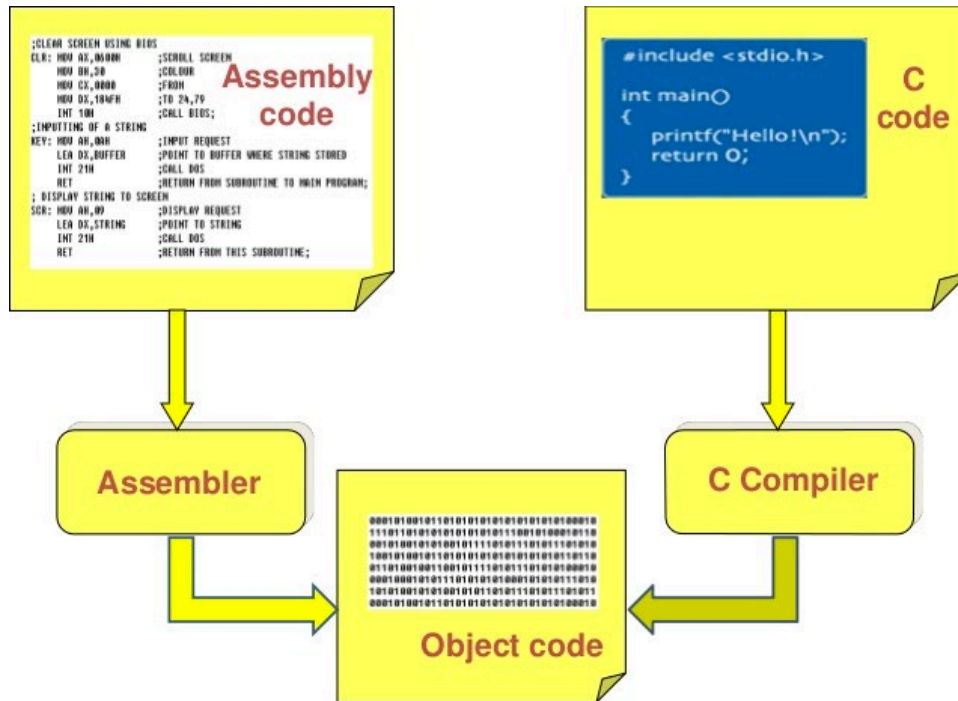
- Low readability
- Difficult in large projects
- High speed, low size

C

- Fairly efficient
- High readability
- Suitable for large projects

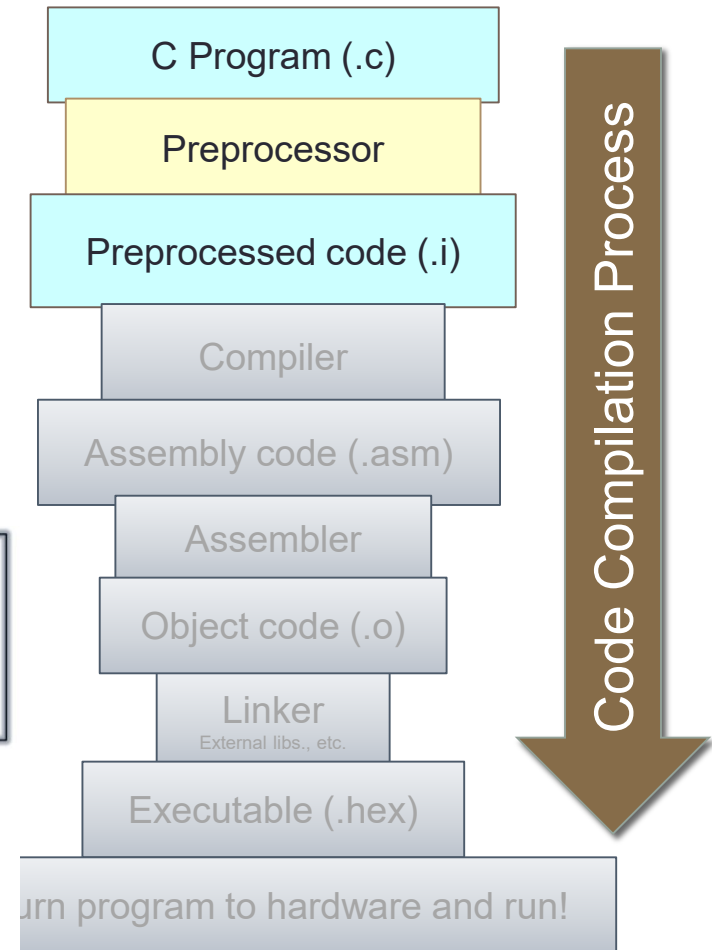
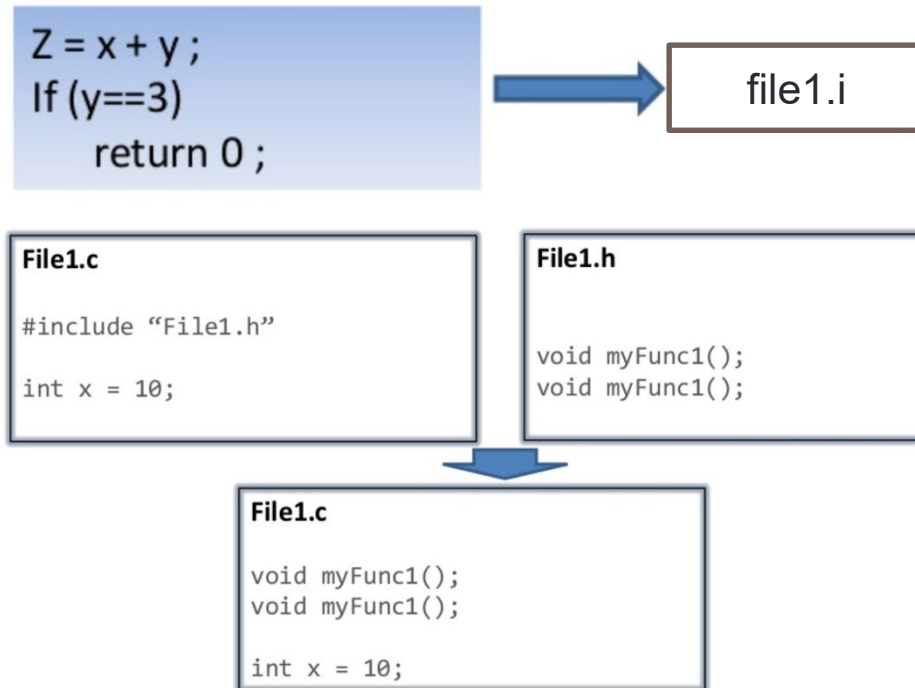
3. C and Hardware Programming

- Gives symbolic names to values
- Provides abstraction of underlying hardware



4. Code Compilation Process (1/8)

- **Preprocessing:** It is the first stage of compilation. It processes preprocessor directives like include-files, conditional compilation instructions and macros.



4. Code Compilation Process (2/8)

- **Preprocessing:** It is the first stage of compilation. It processes preprocessor directives like include-files, conditional compilation instructions and macros.



Object-like Macro:

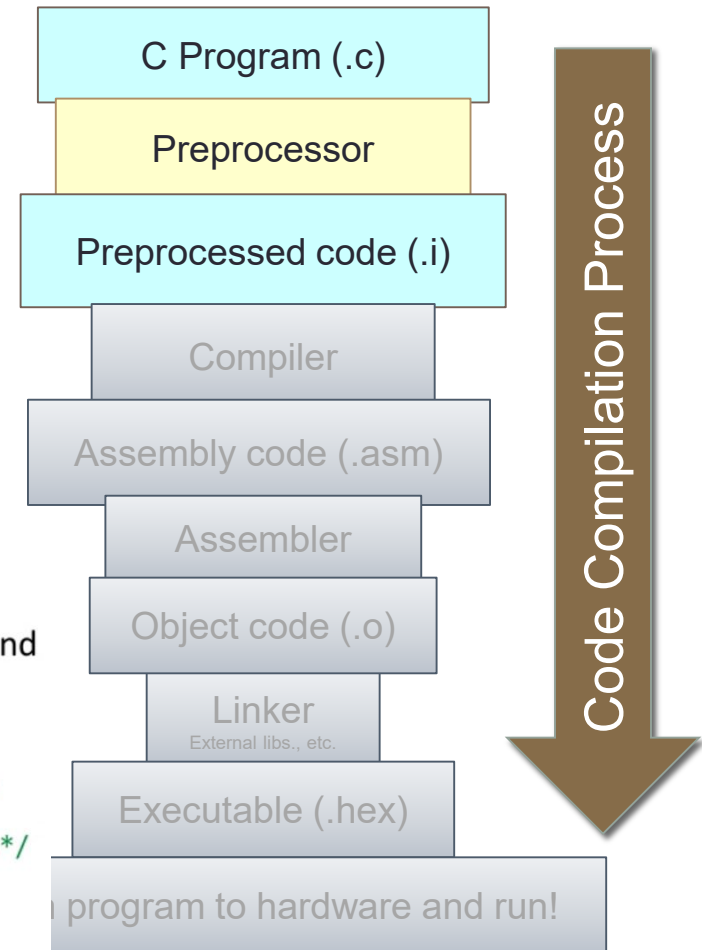
```
#define LED_PIN 10
```

Tells the preprocessor that whenever the symbol `LED_PIN` is found inside the code, replace it with `10`.

So we can type inside the code:

```
int x = LED_PIN;           /* x will have the value 10 */
ledInit(LED_PIN);          /*Initialize LED with value 10*/
#define MY_SECOND_NUMBER LED_PIN
```

Now `MY_SECOND_NUMBER` also has the value `10`.



4. Code Compilation Process (3/8)

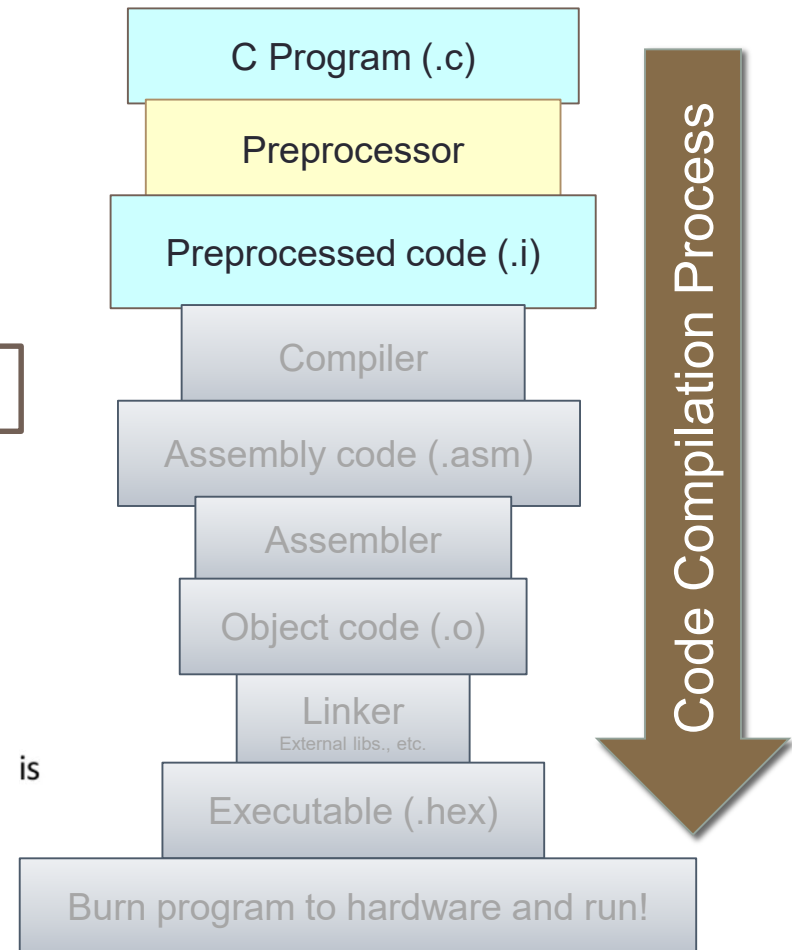
- **Preprocessing:** It is the first stage of compilation. It processes preprocessor directives like include-files, conditional compilation instructions and macros.



- **Conditional compilation:**

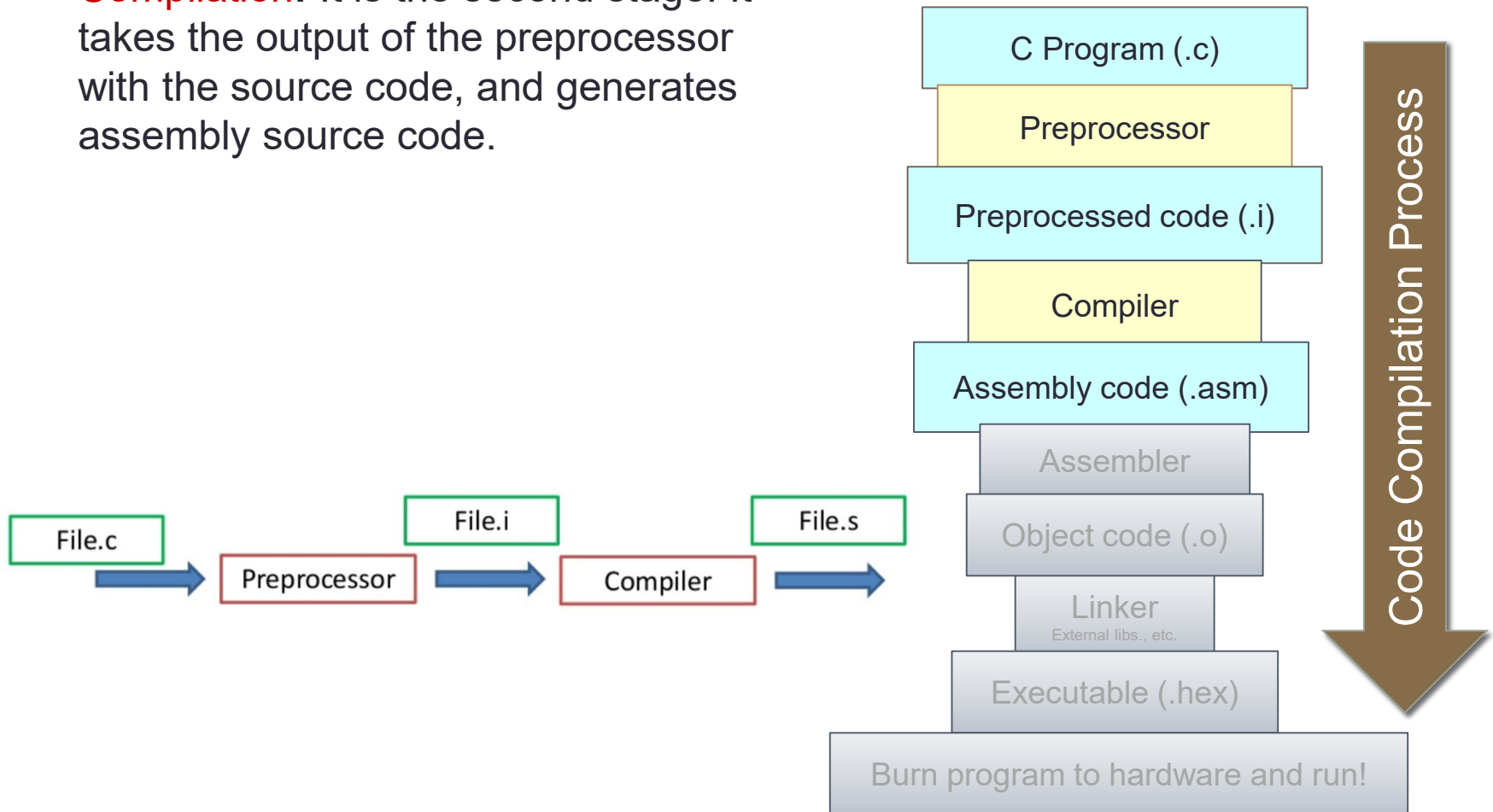
```
#if(LED_PIN==10)  
    printf("LED_PIN=10");  
#endif
```

The `printf` line will be compiled only if the macro `LED_PIN` is defined with value 10.



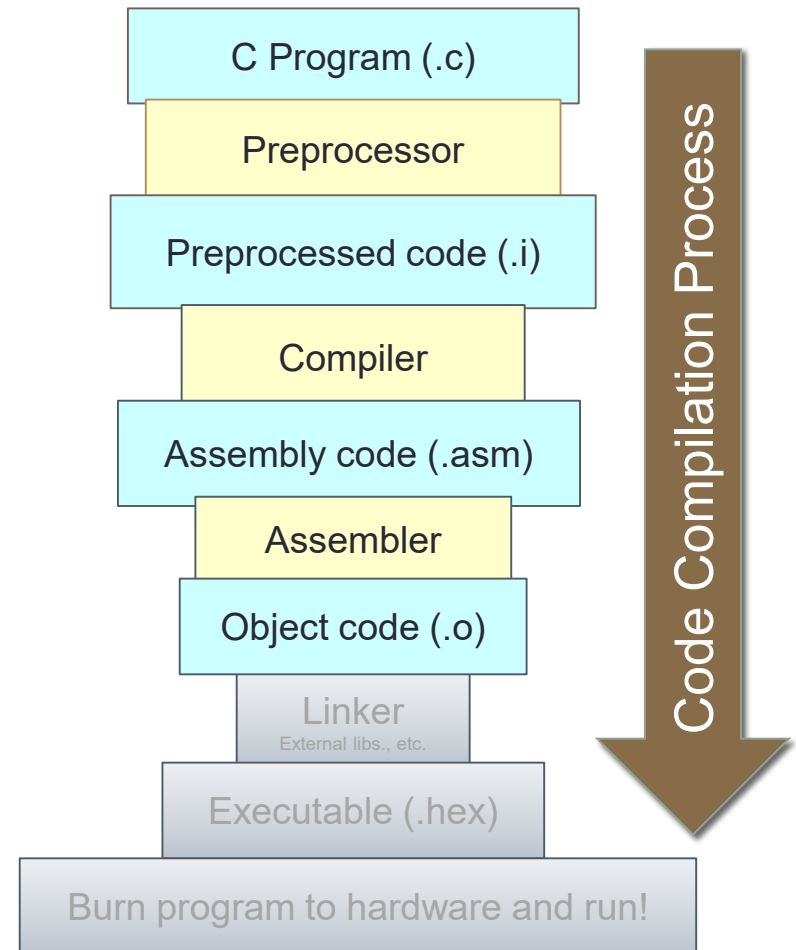
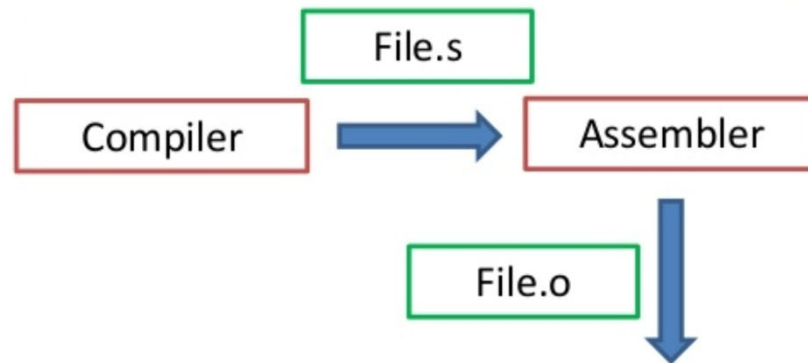
4. Code Compilation Process (4/8)

- **Compilation:** It is the second stage. It takes the output of the preprocessor with the source code, and generates assembly source code.



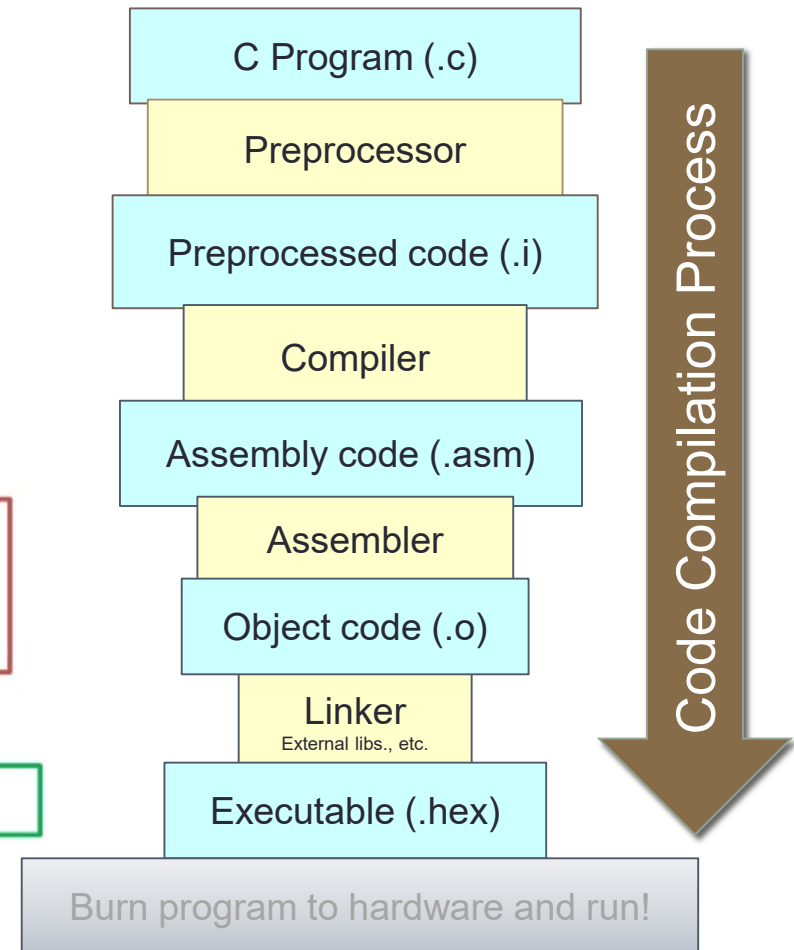
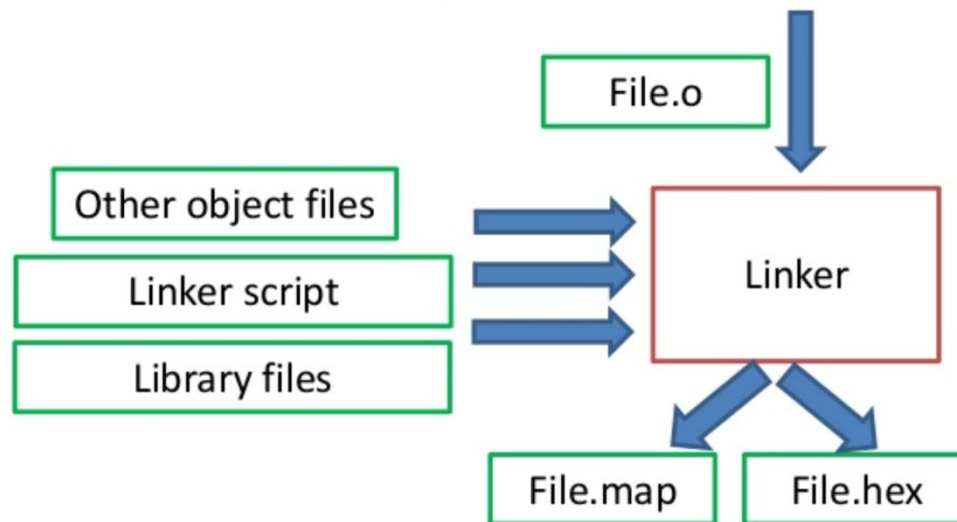
4. Code Compilation Process (5/8)

- **Assembler stage:** It is the third stage of compilation. It takes the assembly source code and produces the corresponding object code.



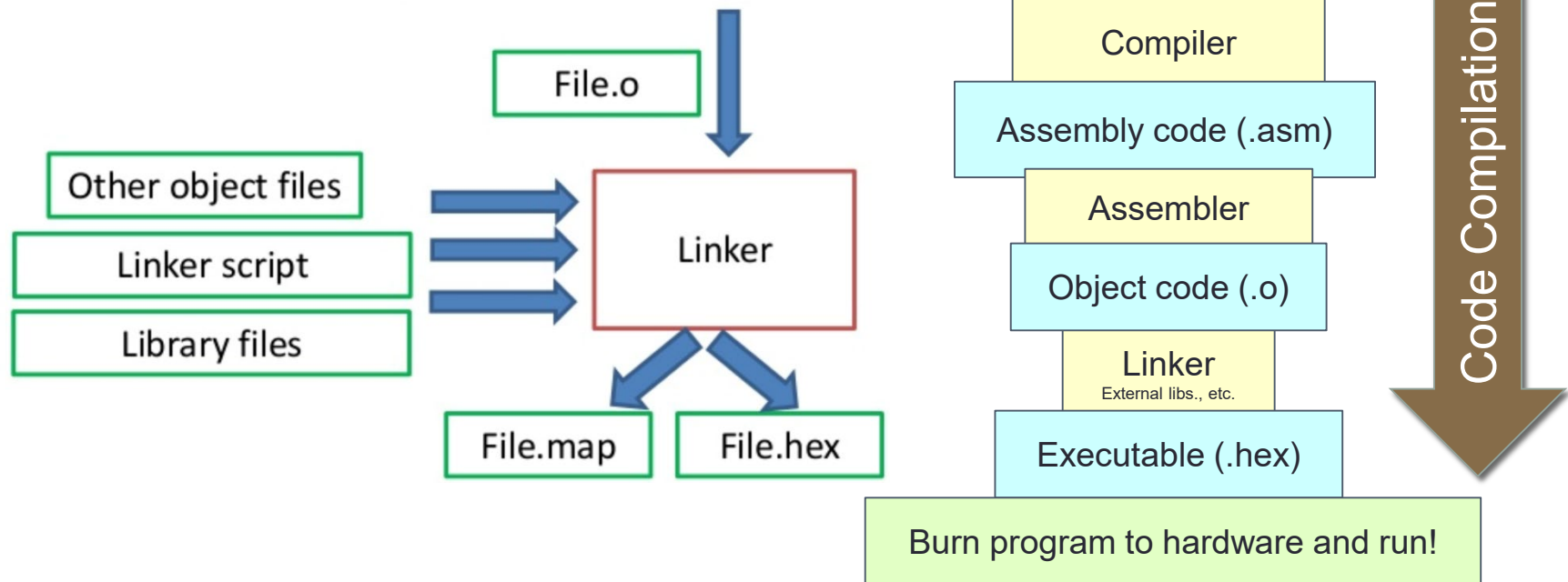
4. Code Compilation Process (6/8)

- **Linking:** It is the final stage of compilation. It takes one or more object files or libraries and linker script as input and combines them to produce a single executable file.

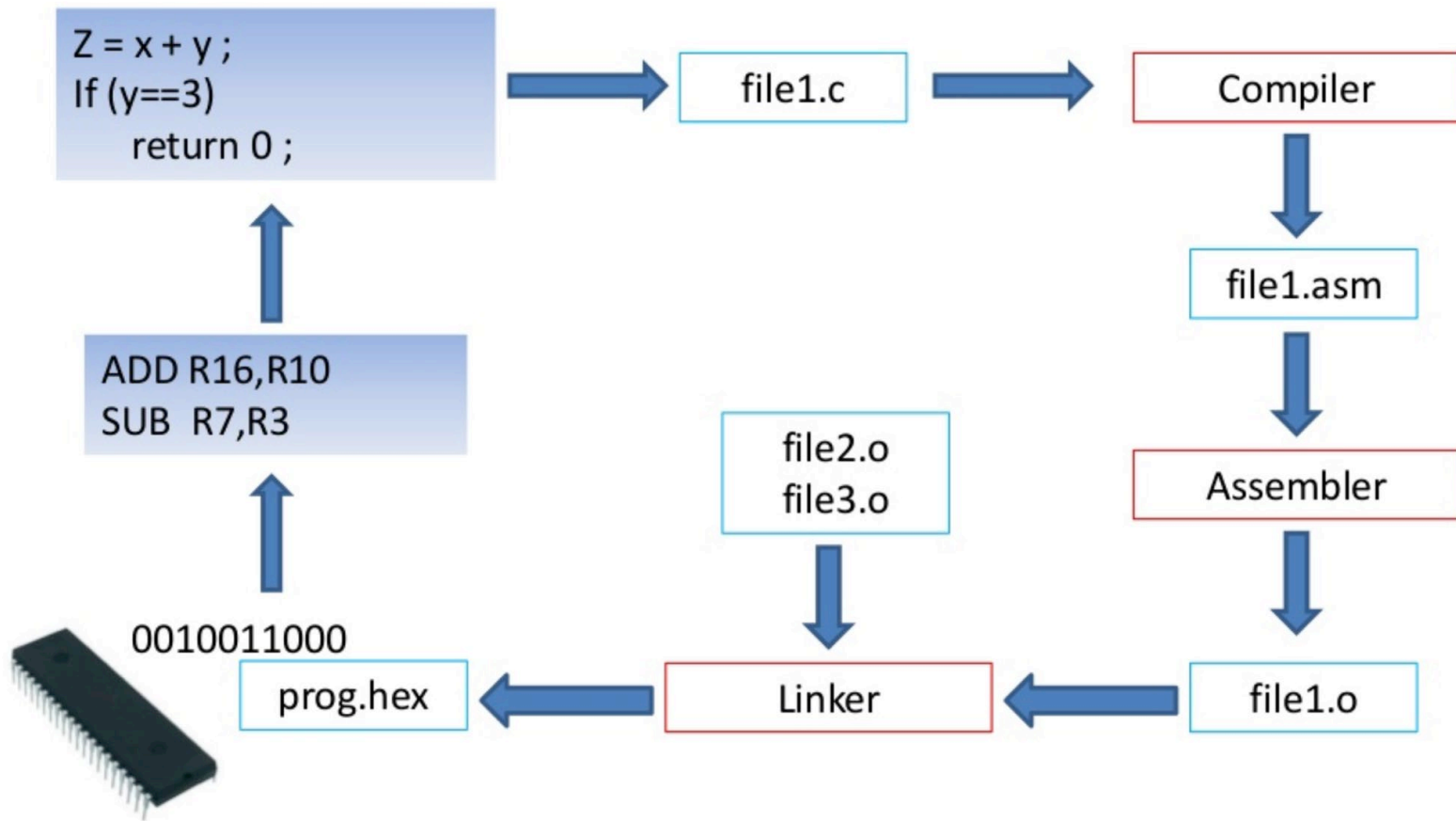


4. Code Compilation Process (7/8)

- In doing so, it resolves references to external symbols, assigns final addresses to procedures/functions and variables, and revises code and data to reflect new addresses (a process called **relocation**).



4. Code Compilation Process (8/8)



5. Debugging – GDB (1/6)

- GDB or GNU Debugger is used to debug C programs
- It is used to inspect, step by step, the execution of a program
- Commands:
 - Start
 - Step
 - explore
 - print <variable>
 - run
 - Continue
 - up
 - down
 - where

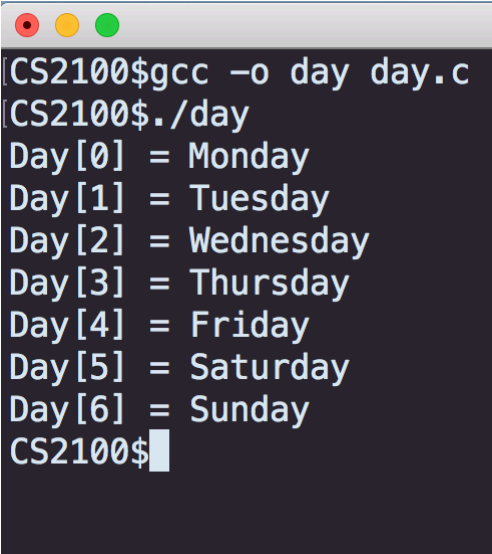
5. Debugging – GDB (2/6)

<p>Output/messages</p> <p>Expressions</p> <p>History</p> <p>Memory</p> <p>Stack</p> <p>[0] from 0x0000000100000eed in main+125 at factorial.c:16 (no arguments)</p> <p>Threads</p> <p>[3] id 5123 from 0x0000000100000eed in main+125 at factorial.c:16</p> <pre> 16 for(i=1; i<=n; ++i) >>> print factorial \$1 = 2 >>> </pre>	<p>Registers</p> <table border="0"> <tr> <td>rax 0x0000000000000002</td> <td>rbx 0x0000000000000000</td> <td>rcx 0x00007fffa5e47388</td> </tr> <tr> <td>rdx 0x0000000000000103</td> <td>rsi 0x0000000000000100</td> <td>rdi 0x0000000000012068</td> </tr> <tr> <td>rbp 0x00007ffefbfbfb10</td> <td>rsp 0x00007ffefbffa0</td> <td>r8 0x00007fffa5e26ea8</td> </tr> <tr> <td>r9 0x0000000000000040</td> <td>r10 0x00007fffa5e26ea0</td> <td>r11 0xffffffffffffff</td> </tr> <tr> <td>r12 0x0000000000000000</td> <td>r13 0x0000000000000000</td> <td>r14 0x0000000000000000</td> </tr> <tr> <td>r15 0x0000000000000000</td> <td>rip 0x0000000100000eed</td> <td>eflags [TF IF]</td> </tr> <tr> <td>cs 0x0000002b</td> <td>ss <unavailable></td> <td>ds <unavailable></td> </tr> <tr> <td>es <unavailable></td> <td>fs 0x00000000</td> <td>gs 0x00000000</td> </tr> </table>	rax 0x0000000000000002	rbx 0x0000000000000000	rcx 0x00007fffa5e47388	rdx 0x0000000000000103	rsi 0x0000000000000100	rdi 0x0000000000012068	rbp 0x00007ffefbfbfb10	rsp 0x00007ffefbffa0	r8 0x00007fffa5e26ea8	r9 0x0000000000000040	r10 0x00007fffa5e26ea0	r11 0xffffffffffffff	r12 0x0000000000000000	r13 0x0000000000000000	r14 0x0000000000000000	r15 0x0000000000000000	rip 0x0000000100000eed	eflags [TF IF]	cs 0x0000002b	ss <unavailable>	ds <unavailable>	es <unavailable>	fs 0x00000000	gs 0x00000000
rax 0x0000000000000002	rbx 0x0000000000000000	rcx 0x00007fffa5e47388																							
rdx 0x0000000000000103	rsi 0x0000000000000100	rdi 0x0000000000012068																							
rbp 0x00007ffefbfbfb10	rsp 0x00007ffefbffa0	r8 0x00007fffa5e26ea8																							
r9 0x0000000000000040	r10 0x00007fffa5e26ea0	r11 0xffffffffffffff																							
r12 0x0000000000000000	r13 0x0000000000000000	r14 0x0000000000000000																							
r15 0x0000000000000000	rip 0x0000000100000eed	eflags [TF IF]																							
cs 0x0000002b	ss <unavailable>	ds <unavailable>																							
es <unavailable>	fs 0x00000000	gs 0x00000000																							
<p>Assembly</p> <pre> 0x0000000100000ee0 main+112 movslq -0xc(%rbp),%rax 0x0000000100000ee4 main+116 imul -0x18(%rbp),%rax 0x0000000100000ee9 main+121 mov %rax,-0x18(%rbp) 0x0000000100000eed main+125 mov -0xc(%rbp),%eax 0x0000000100000ef0 main+128 add \$0x1,%eax 0x0000000100000ef3 main+131 mov %eax,-0xc(%rbp) 0x0000000100000ef6 main+134 jmpq 0x100000ed4 <main+100> </pre>	<p>Source</p> <pre> 11 if (n < 0) 12 printf("Error! Factorial of a negative number doesn't exist."); 13 14 else 15 { 16 for(i=1; i<=n; ++i) 17 { 18 factorial *= i; // factorial = factorial*i; 19 } 20 printf("Factorial of %d = %llu", n, factorial); 21 } </pre>																								

5. Debugging – Sample (3/6)

- Program to print the days of the week:

```
#include <stdio.h>
int main(void) {
    int i;
    char *day[7] = {
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday",
        "Sunday"
    };
    for (i = 0; i<=10; i++) {
        printf("Day[%d] = %s \n", i, day[i]);
    }
    return 0;
}
```



```
CS2100$gcc -o day day.c
CS2100$./day
Day[0] = Monday
Day[1] = Tuesday
Day[2] = Wednesday
Day[3] = Thursday
Day[4] = Friday
Day[5] = Saturday
Day[6] = Sunday
CS2100$
```


5. Debugging – Sample (4/6)

- GDB
- **Step**: helps iterating through the program step by step
- **Print**: gives you the current value of the variable
- **List**: lists the program

```
>>> print i
$1 = 7
>>> █
```

```
>>> help step
Step program until it reaches a different source line.
Usage: step [N]
Argument N means step N times (or till program stops for another reason).
>>> █
```

```
>>> list
1      #include<stdio.h>
2      int main(int argc, char *argv[]){
3          int i;
4          char *day[7] = {
5              "Monday",
6              "Tuesday",
7              "Wednesday",
8              "Thursday",
9              "Friday",
10             "Saturday",
>>> █
```



To change
parameters

5. Debugging – Sample (5/6)

■ Segmentation fault

```
code — gdb • sudo — 167x47

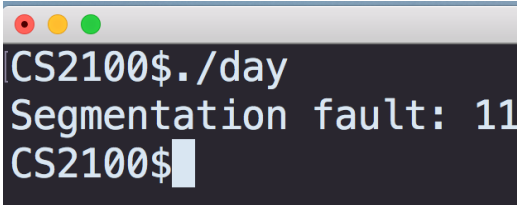
Output/messages
Thread 3 received signal SIGSEGV, Segmentation fault.
Assembly
0x00007fff6c847432 ? pcmpeqb (%rdi),%xmm0
0x00007fff6c847436 ? pmovmskb %xmm0,%esi
0x00007fff6c84743a ? and    $0xf,%rcx
0x00007fff6c84743e ? or     $0xffffffffffffffff,%rax
0x00007fff6c847442 ? shl    %cl,%rax
0x00007fff6c847445 ? and    %eax,%esi
0x00007fff6c847447 ? je     0x7fff6c847460
Expressions
History
$$0 = 7
Memory
Registers
rax 0x69189193df120061  rbx 0x0000000000000002  rcx 0x69189193df120061  rdx 0x69189193df120061  rsi 0xffffffffffffffff  rdi 0x69189193df120060
rbp 0x00007ffefbfff520  rsp 0x00007ffefbfff520  r8 0x00007ffa5e271a8    r9 0x000000000000000a  r10 0x00007ffa5e266b8  r11 0x0000000010000fb5
r12 0x0000000010000fb7  r13 0x0000000000000000  r14 0x0000000000000073  r15 0x00007fff6c88ea08  rip 0x00007fff6c847432  eflags [ PF IF RF ]
cs 0x00000002b         ss <unavailable>      ds <unavailable>      es <unavailable>      fs 0x00000000         gs 0x00000000
Source
Stack
[0] from 0x00007fff6c847432
(no arguments)
[1] from 0x00007ffefbfff8b0
(no arguments)
[+]
Threads
[3] id 5123 from 0x00007fff6c847432

0x00007fff6c847432 in ?? ()
>>> █
```

5. Debugging – Sample (6/6)

- Program to print the days of the week:

```
#include <stdio.h>
int main(void) {
    int i;
    char *day[7] = {
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday",
        "Sunday"
    };
    for (i = 0; i<=10; i++) {
        printf("Day[%d] = %s \n", i, day[i]);
    }
    return 0;
}
```



A terminal window with a dark background and light text. It shows the command `CS2100$./day` followed by the output `Segmentation fault: 11` and the prompt `CS2100$`.

← Line 11

End of File