

Introduction to 8086 Assembly

Lecture 3

Object files, compiling, assembling and linking

Compiling a C file



K. N. Toosi
University of Technology

```
#include <stdio.h>
```

test.c

```
int main() {
```

```
    printf("Salaaaaam!!!\n");
```

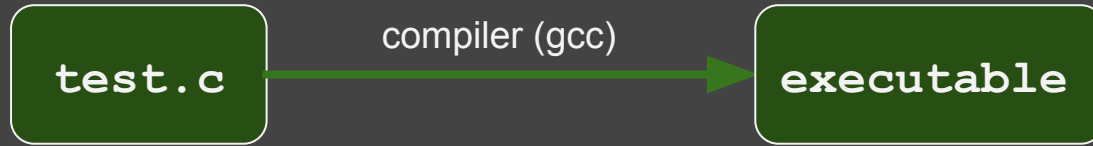
```
    return 0;
```

```
}
```

Compiling a C file



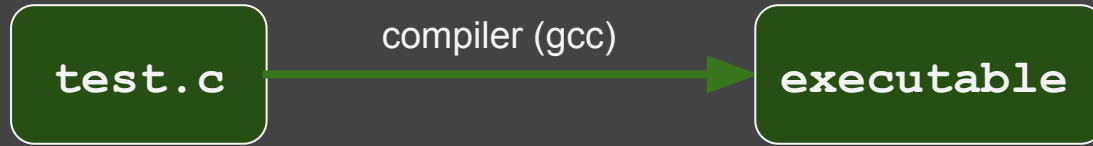
K. N. Toosi
University of Technology



Compiling a C file



K. N. Toosi
University of Technology



```
b.nasihatkon@kntu:lecture3$ gcc test.c  
b.nasihatkon@kntu:lecture3$ ./a.out  
Salaaaaam!!!
```

Compiling a C file



K. N. Toosi
University of Technology

```
#include <stdio.h>
```

test.c

```
int main() {
```

```
    printf("Salaaaaam!!!\n");
```

```
    return 0;
```

```
}
```

```
b.nasihatkon@kntu:lecture3$ gcc test.c
b.nasihatkon@kntu:lecture3$ ./a.out
Salaaaaam!!!
```

Compiling a C file



K. N. Toosi
University of Technology

```
#include <stdio.h>
```

test.c

```
int main() {
```

```
    printf("Salaaaaam!!!\n");
```

```
    return 0;
```

```
}
```

Who wrote **printf**?

Where is the code for **printf**?

```
b.nasihatkon@kntu:lecture3$ gcc test.c
b.nasihatkon@kntu:lecture3$ ./a.out
Salaaaaam!!!
```

Compiling a C file



K. N. Toosi
University of Technology

test.c

```
extern int printf (const char *__restrict __format, ...);

int main() {

    printf("Salaaaaam!!!\n");

    return 0;
}
```

Compiling a C file



K. N. Toosi
University of Technology

test.c

```
extern int printf (const char *__restrict __format, ...);  
  
int main() {  
  
    printf("Salaaaaam!!!\n");  
  
    return 0;  
}
```

```
b.nasihatkon@kntu:lecture3$ gcc test.c  
b.nasihatkon@kntu:lecture3$ ./a.out  
Salaaaaam!!!  
b.nasihatkon@kntu:lecture3$
```


Compiling a C file, object files



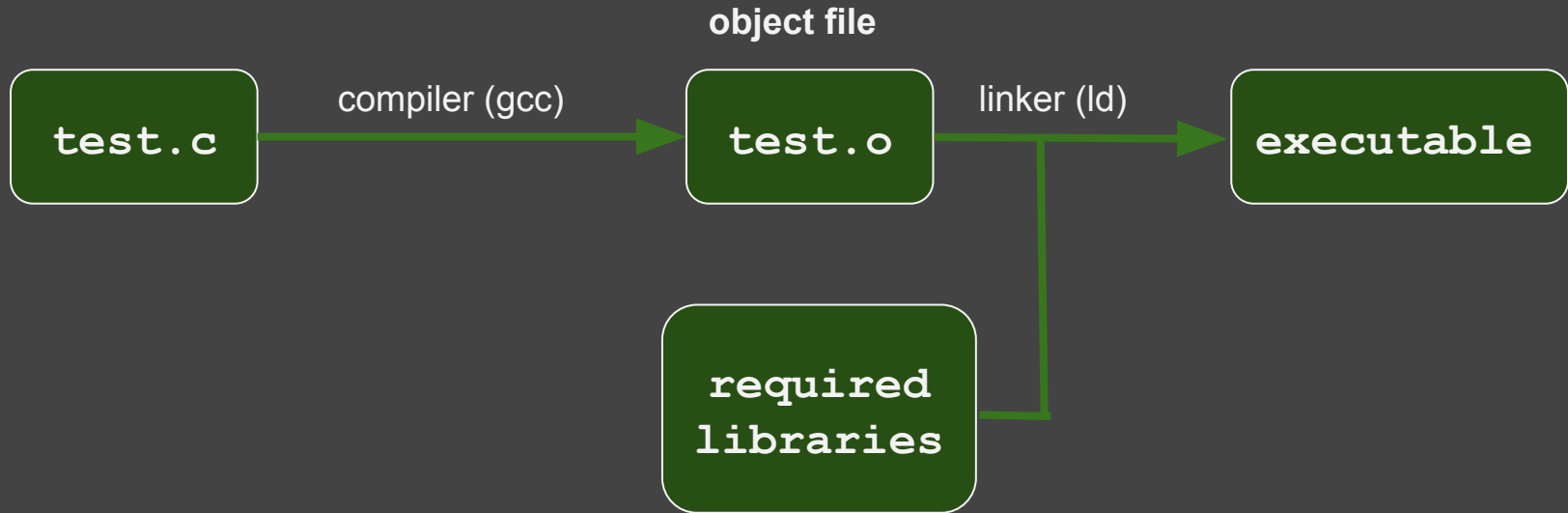
K. N. Toosi
University of Technology



Object files, libraries and linking



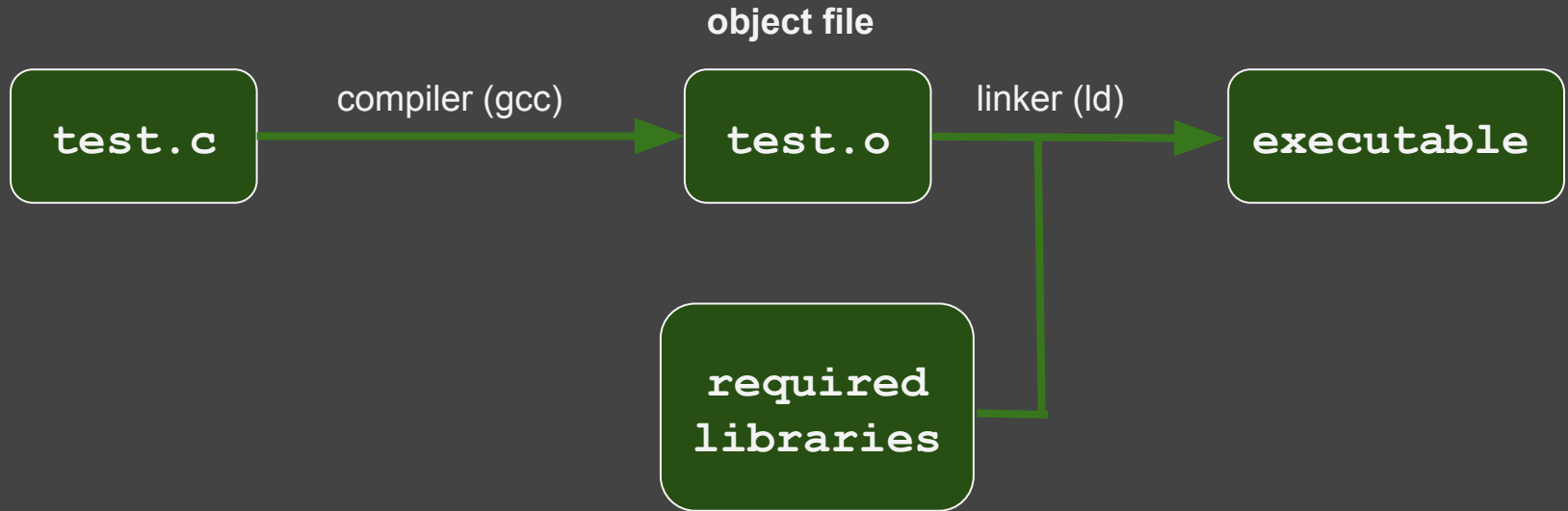
K. N. Toosi
University of Technology



Libraries



K. N. Toosi
University of Technology



Libraries



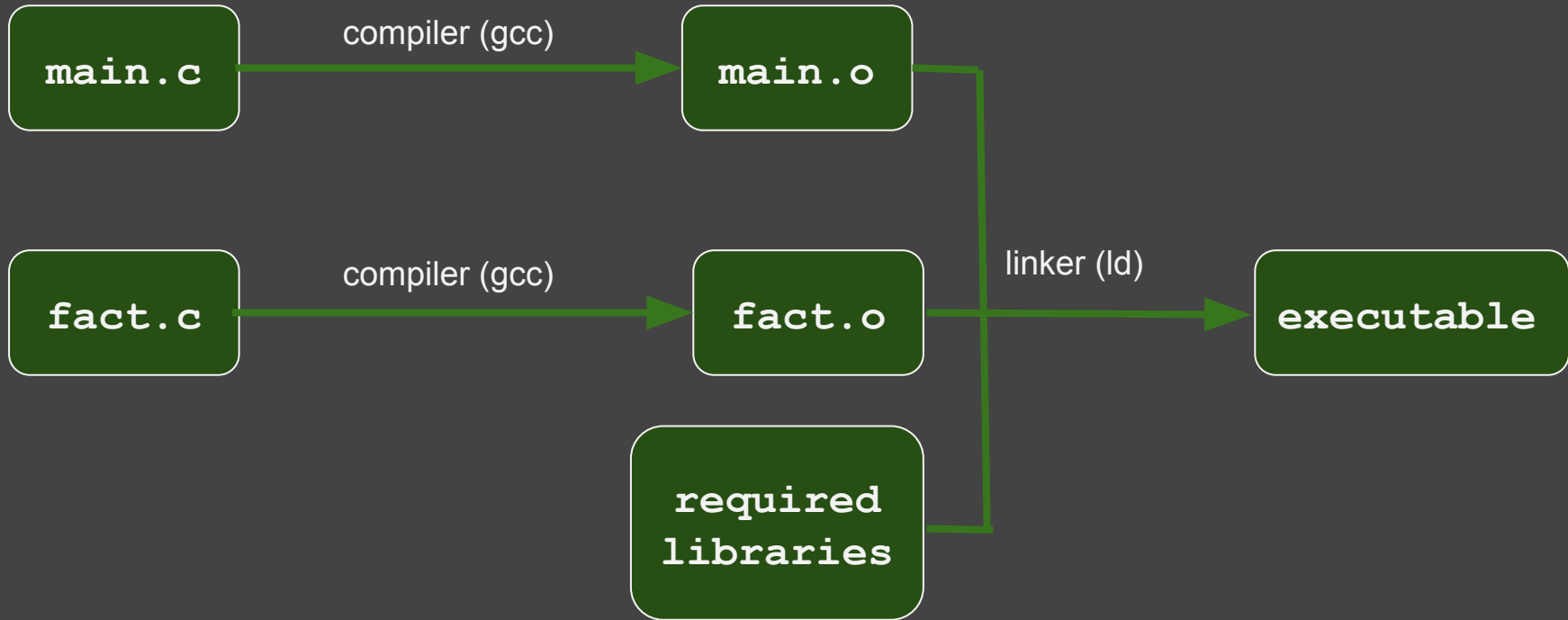
K. N. Toosi
University of Technology

- Collection of object files
- Static vs. Dynamic Linking
- Shared Objects (SO), Dynamic-Link Libraries (DLL)

Modular programming in C



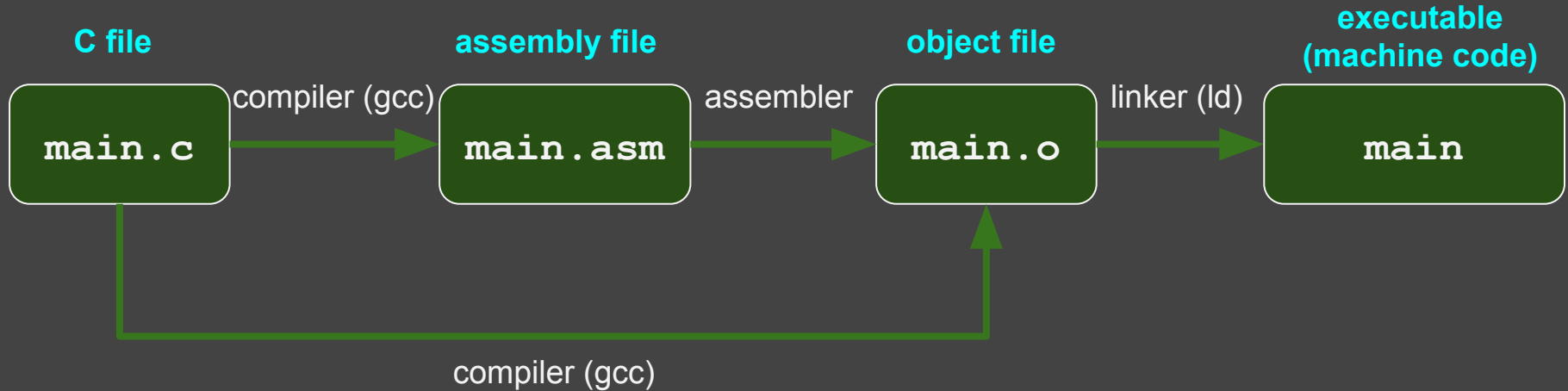
K. N. Toosi
University of Technology



high-level to low-level hierarchy



K. N. Toosi
University of Technology





Object files

- Machine code + metadata (unresolved symbols, etc.)
 - for linking, debugging, etc.
 - https://en.wikipedia.org/wiki/Object_file
 - https://en.wikipedia.org/wiki/Object_code
- Object file formats
 - Common Object File Format (COFF)
 - Relocatable Object Module Format (OMF)
 - Executable and Linkable Format (ELF)

Assembling assembly files



K. N. Toosi
University of Technology



Our first assembly program (64 bit)



```
segment .data
msg: db "Salaaaaaaam!!", 10 ; message + newline character

global _start

segment .text
_start: ; entry point (linker needs this)

    mov rax, 1 ; system call number (sys_write=1)
    mov rdi, 1 ; file descriptor (stdout=1)
    mov rsi, msg ; message to write
    mov rdx, 14 ; number of bytes to write

    syscall ; invoke system call (sys_write)

    mov rax, 60 ; system call number (sys_exit=60)
    mov rdi, 0 ; exit code 0

    syscall ; invoke system call (sys_exit)
```

Our first assembly program (64 bit)



K. N. Toosi
University of Technology

```
segment .data
msg: db "Salaaaaaaam!!", 10 ; message + newline character

global _start

segment .text
_start: ; entry point (linker needs this)

    mov rax, 1 ; system call number (sys_write=1)
    mov rdi, 1 ; file descriptor (stdout=1)
    mov rsi, msg ; message to write
    mov rdx, 14 ; number of bytes to write

    syscall ; invoke system call (sys_write)

    mov rax, 60 ; system call number (sys_exit=60)
    mov rdi, 0 ; exit status (0)
    syscall ; invoke system call (sys_exit)
```

```
b.nasihatkon@kntu:lecture3$ nasm -f elf64 first.asm
b.nasihatkon@kntu:lecture3$ ld first.o
b.nasihatkon@kntu:lecture3$ ./a.out
Salaaaaaaam!!
```

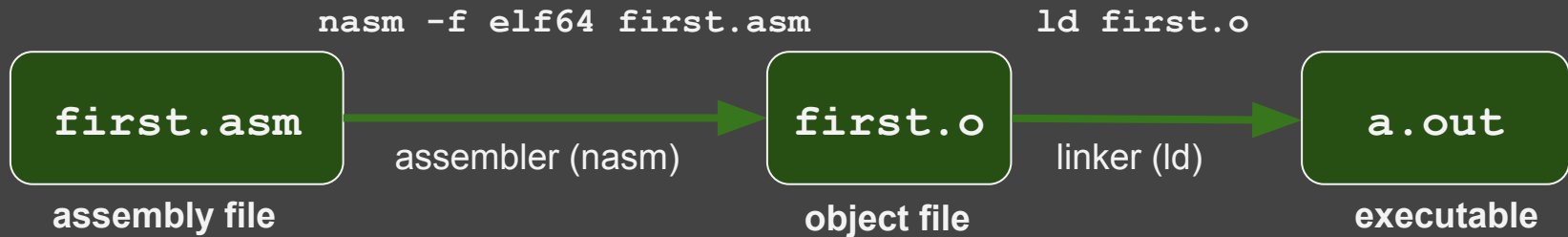
look at: <http://cs.lmu.edu/~ray/notes/nasmtutorial/>

Assembling, linking and running (64 bit)



K. N. Toosi
University of Technology

- **Assemble:** `nasm -f elf64 first.asm`
- **Link:** `ld first.o`
- **Execute:** `./a.out`



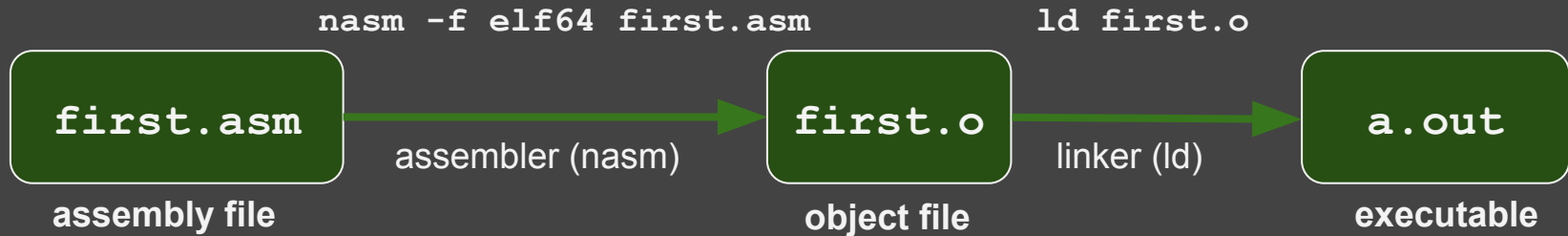
Assembling, linking and running (64 bit)



K. N. Toosi
University of Technology

assembler create 64 bit ELF object file format input assembly file

```
nasm -f elf64 first.asm
```



Our first assembly program (32 bit)



```
segment .data
msg: db "Salaaaaaaam!!", 10 ; message + newline character

segment .text
    global _start ; make entry point visible

_start: ; entry point (linker needs this)
    mov     edx, 14 ; message length
    mov     ecx, msg ; address of message to write
    mov     ebx, 1 ; file descriptor (stdout=1)
    mov     eax, 4 ; (32 bit) system call number (sys_write=4)
    int     0x80 ; interrupt no. x80 = system call

    mov     eax, 1 ; (32 bit) system call number (sys_exit=1)
    int     0x80 ; interrupt no. x80 = system call
```

Our first assembly program (32 bit)



```
segment .data
msg:  db          "Salaaaaaaam!!", 10  ; message + newline character

segment .text
    global _start      ; make entry point visible

_start:                ; entry point (linker needs this)
    mov  edx, 14        ; message length
    mov  ecx, msg       ; address of message to write
    mov  ebx, 1         ; file descriptor (stdout=1)
    mov  eax, 4         ; (32 bit) system call number (sys_write=4)
    int  0x80          ; interrupt no. x80 = system call

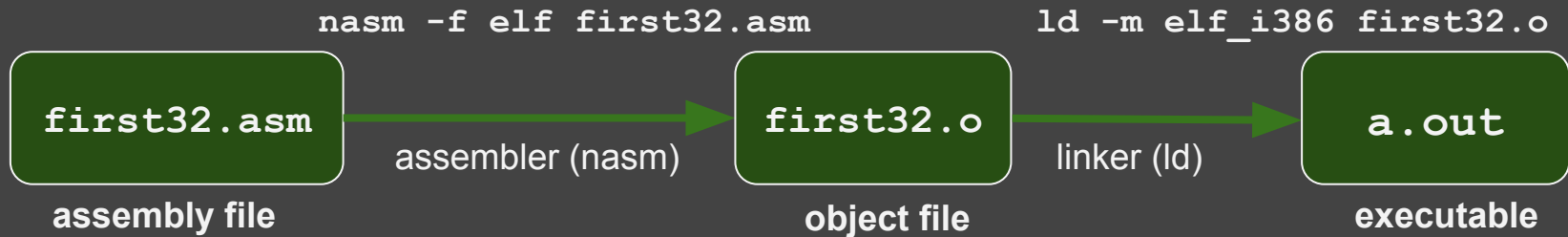
    mov  eax, 1        ; (32 bit) system call number (sys_exit=1)
    int  0x80          ; interrupt no. x80 = system call
```

Assembling, linking and running (32 bit)



K. N. Toosi
University of Technology

- **Assemble:** `nasm -f elf first32.asm`
- **Link:** `ld -m elf_i386 first32.o`
- **Execute:** `./a.out`



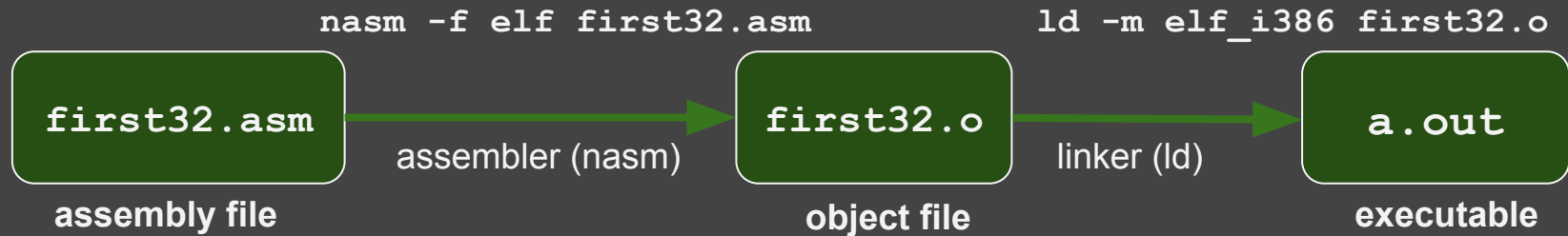
Assembling, linking and running (32 bit)



K. N. Toosi
University of Technology

assembler create 32 bit ELF object file format input assembly file

```
nasm -f elf first32.asm
```



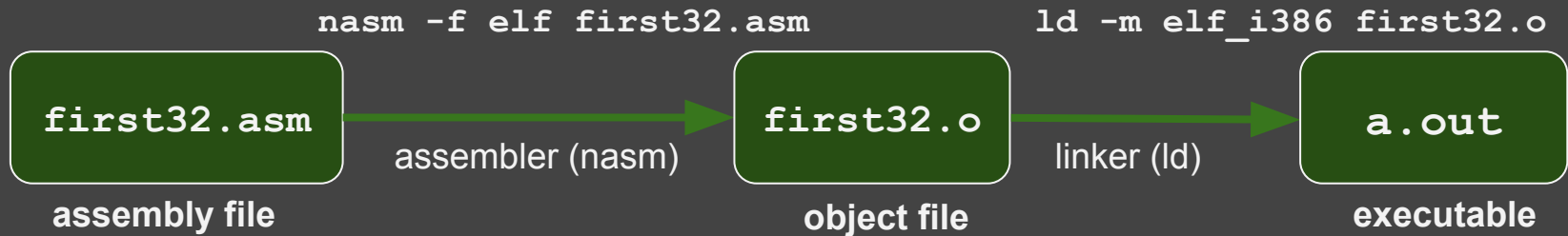
Assembling, linking and running (32 bit)



K. N. Toosi
University of Technology

linker create 32 bit executable input object file (32 bit ELF format)

```
ld -m elf_i386 first32.o
```



Compiling and linking C files



K. N. Toosi
University of Technology

test.c

```
#include <stdio.h>

int fact(int);

int main() {
    int x = 8;
    printf("x!=%d\n", fact(x));

    return 0;
}
```

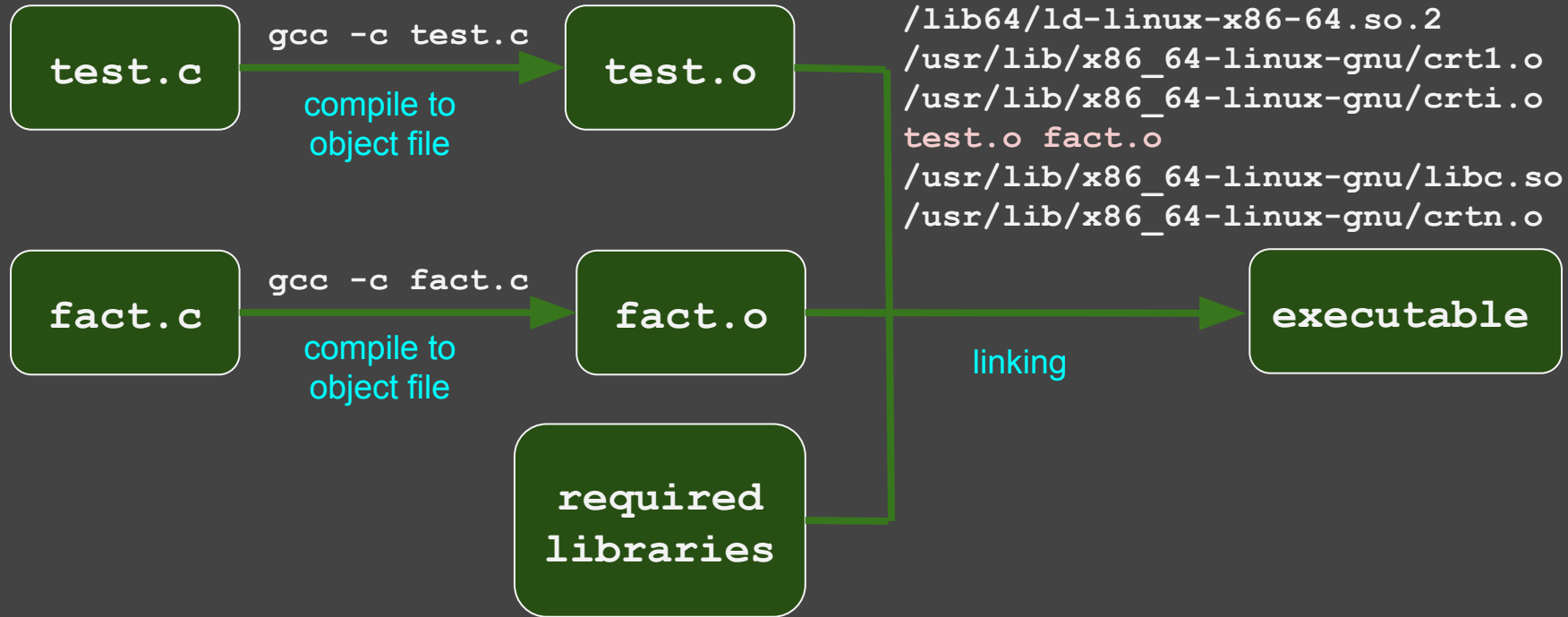
fact.c

```
int fact(int n) {
    return n==0 ? 1 : n*fact(n-1);
}
```

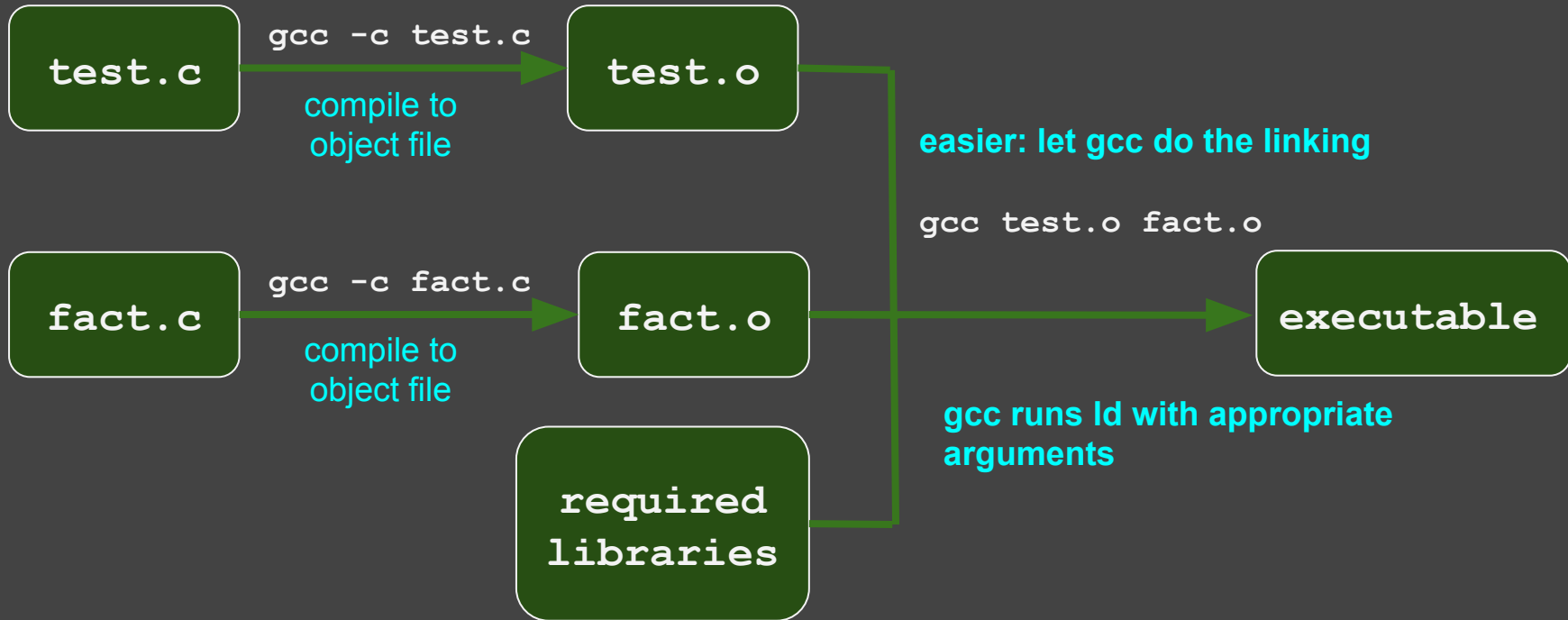
Compiling and linking C files



K. N. Toosi
University of Technology



Compiling and linking C files



Compiling C files to 32 bit apps



K. N. Toosi
University of Technology

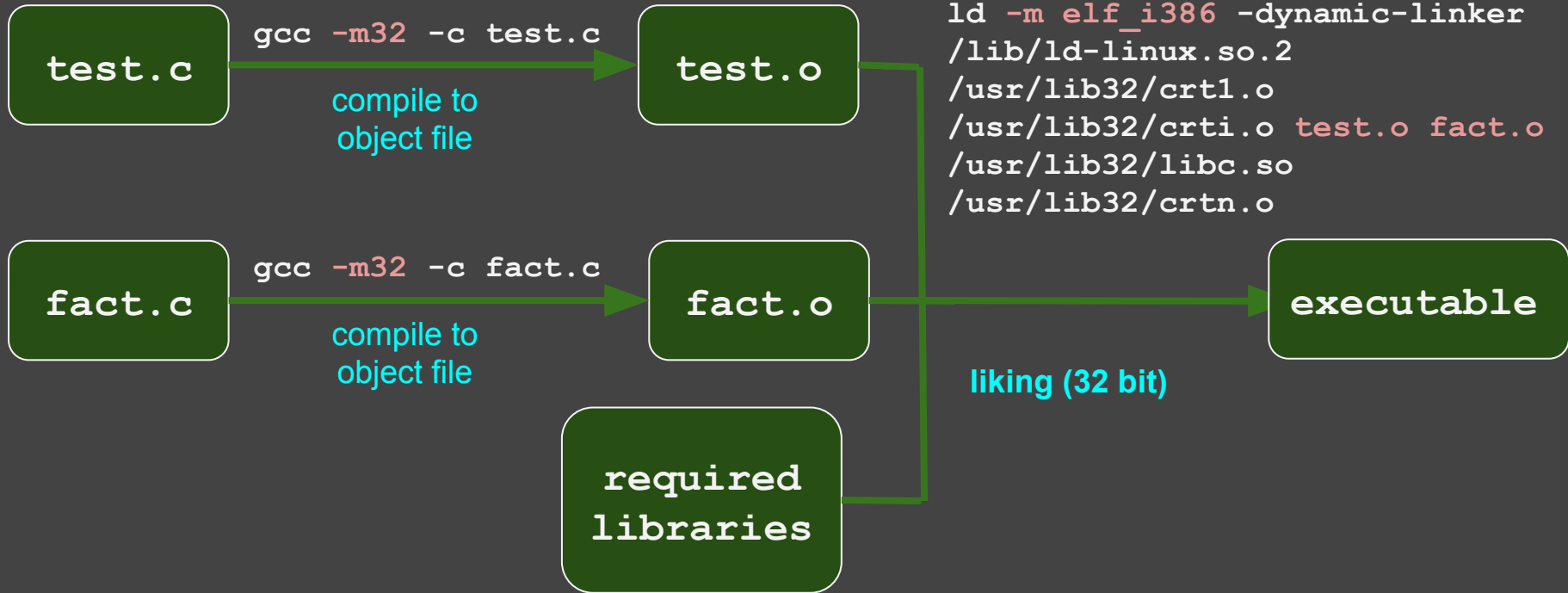
Compile to 32 bit executable on a 64 bit system:

- First, install the 32 bit libraries:
 - `sudo apt-get install libc6-dev-i386`
 - `sudo apt-get install libx32gcc-4.8-dev`
 - `sudo apt-get install gcc-multilib`
- You might need to install:
 - `sudo apt-get install gcc-6-multilib`

32 bit Compiling and linking C files



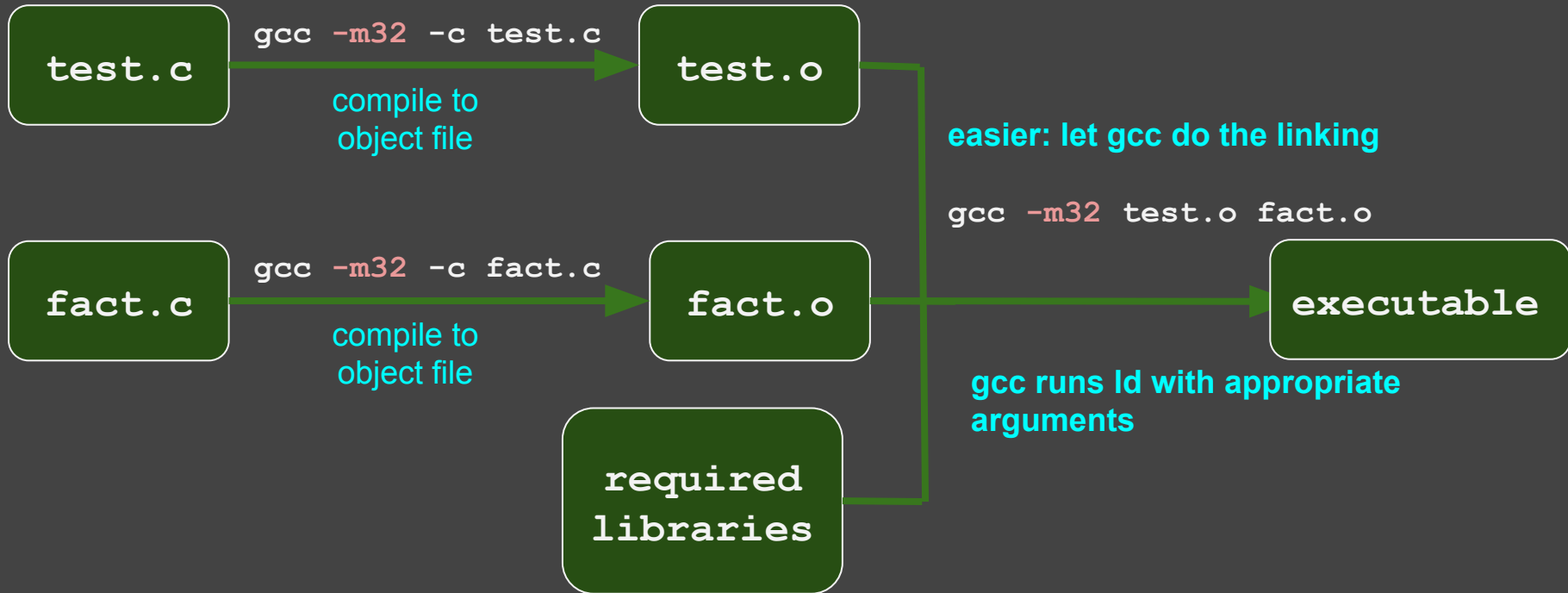
K. N. Toosi
University of Technology



32 bit Compiling and linking C files



K. N. Toosi
University of Technology



Our second assembly program!



K. N. Toosi
University of Technology

- We do (mostly) 32 bit assembly
- We use the functions/macros from the book (Carter, *PC Assembly Language*, 2007) for IO

I/O functions and macros from the book



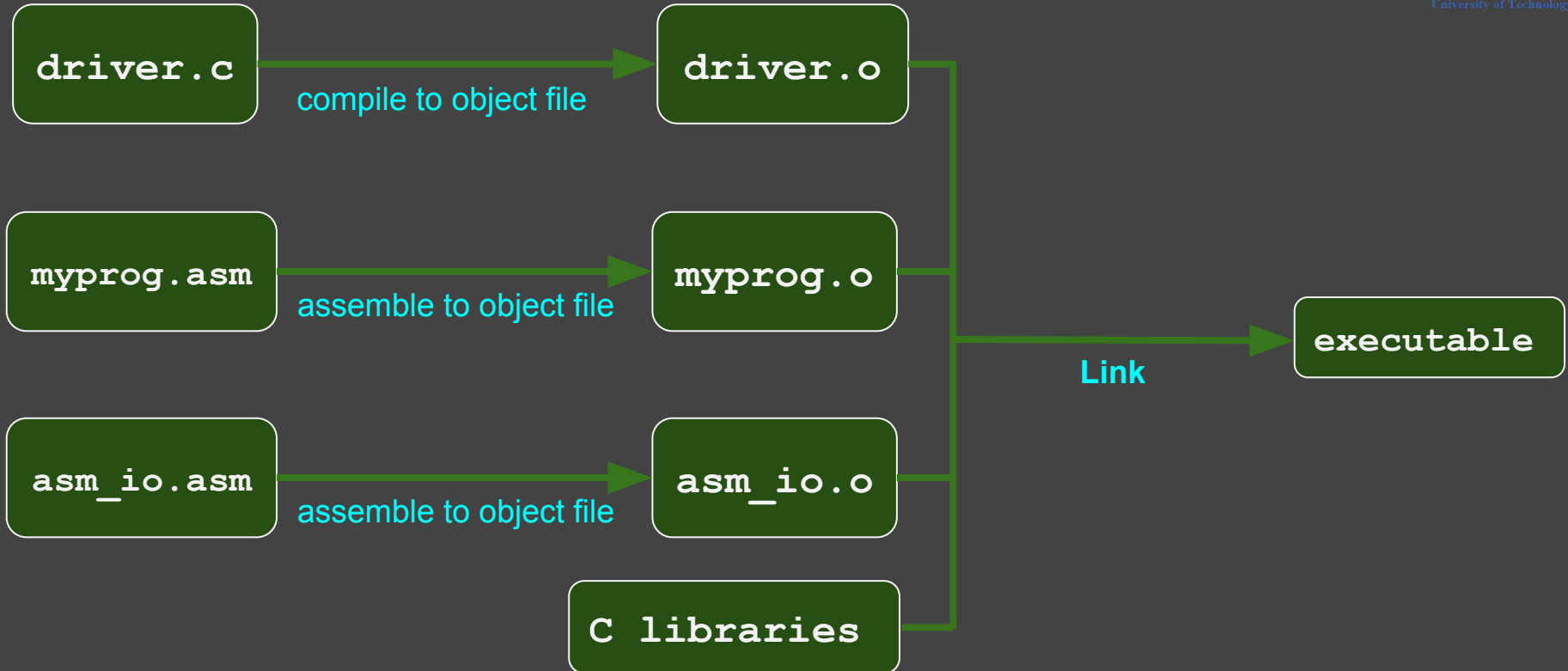
K. N. Toosi
University of Technology

<code>call print_int</code>	prints EAX as signed integer
<code>call print_char</code>	prints the character whose ascii code is stored in AL
<code>call print_string</code>	prints the string whose starting address is stored in EAX, string must be null-terminated (C-string)
<code>call print_nl</code>	prints a newline character
<code>call read_int</code>	reads an integer from standard input and stores it into EAX
<code>call read_char</code>	read a character from standard input and stores its ascii code in EAX
<code>dump_regs <num></code>	(MACRO) prints out values of registers and flags (<num> is some number like 12 making debugging easier)

Our second assembly program!



K. N. Toosi
University of Technology





steps to run our program

0. Install the Netwide Assembler
 - `sudo apt install nasm`
1. Download the example files (including IO files) from the books website:
<http://pacman128.github.io/pcasm/>
 - for linux click on the link [linux examples](#) to download the files
 - there are links for other platforms as well
2. Copy the files `asm_io.inc`, `asm_io.asm` and `cdecl.h` to your current working directory.
3. Compile the file `asm_io.asm` to object file (creating `asm_io.o`)
 - `nasm -f elf -d ELF_TYPE asm_io.asm`
4. Create a file named `driver.c`

steps to run our program



K. N. Toosi
University of Technology

4. Create a file named `driver.c` simply calling an assembly function:

```
void asm_main();

int main() {

    asm_main();

    return 0;

}
```

(alternatively, copy the files `driver.c` and `cdecl.h` to your current directory.)

steps to run our program



5. Install the 32 bit C libraries (if not installed)

- `sudo apt-get install libc6-dev-i386`
- `sudo apt-get install libx32gcc-4.8-dev`
- `sudo apt-get install gcc-multilib (if needed)`
- `sudo apt-get install gcc-6-multilib (if needed)`

6. Compile driver.c to 32 bit object file (creating driver.o)

- `gcc -m32 -c driver.c`

steps to run our program



7. Create your main assembly file containing the `asm_main` label
 - here the file named `myprog.asm` and looks like this:

```
%include "asm_io.inc"

segment .text

global asm_main

asm_main:
    enter 0,0
    pusha

    mov eax, 100
    mov ebx, 20
    sub eax, ebx

    call print_int    ; print EAX
    call print_nl    ; print a new line

    dump_regs 1111    ; printing the system state (registers, etc.)

    popa
    leave
    ret
```

steps to run our program



7. Create your main assembly file containing the `asm_main` label
 - here the file named `myprog.asm` and looks like this:

```
%include "asm_io.inc"

segment .text

global asm_main

asm_main:
    enter 0,0
    pusha

    mov eax, 100
    mov ebx, 20
    sub eax, ebx

    call print_int    ; print EAX
    call print_nl    ; print a new line

    dump_regs 1111   ; print registers, etc

    popa
    leave
    ret
```

`myprog.asm`

```
void asm_main();

int main() {

    asm_main();

    return 0;

}
```

`driver.c`



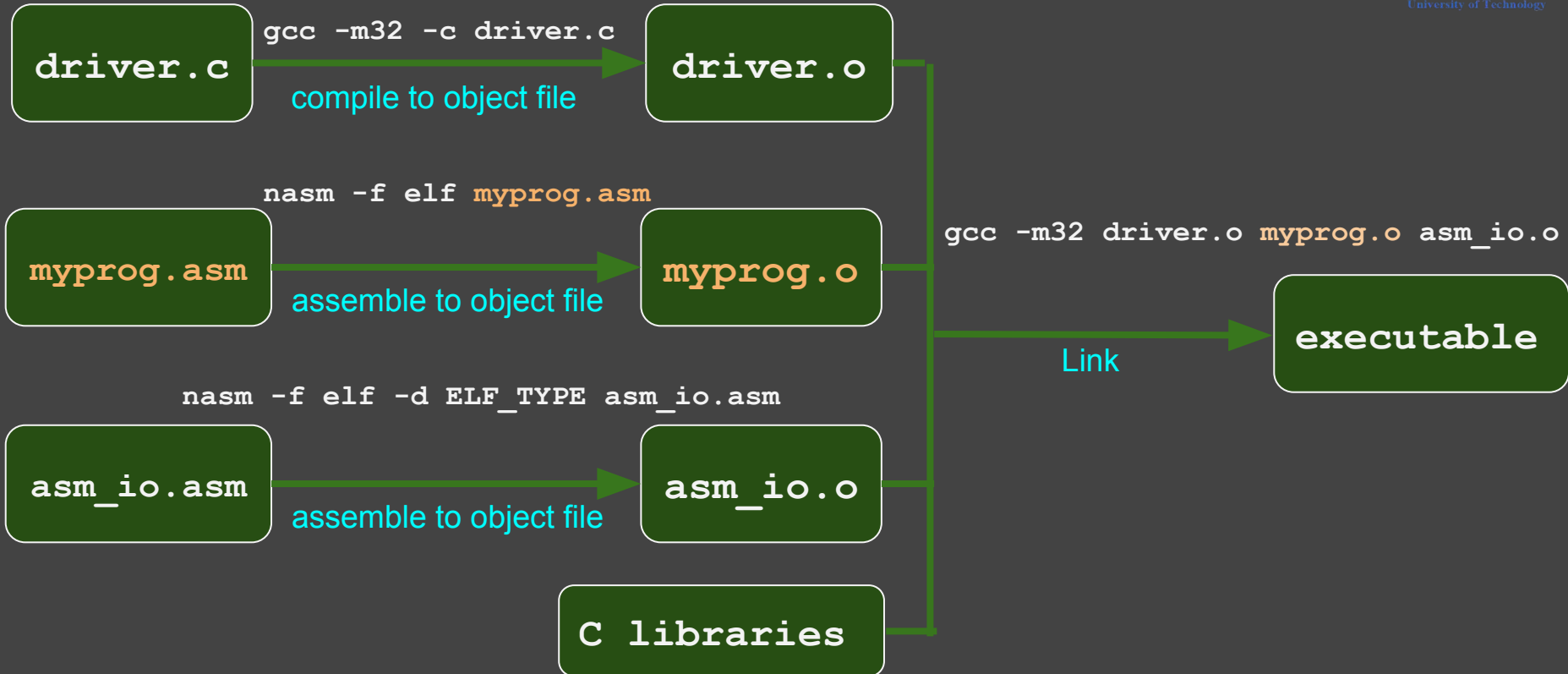
steps to run our program

8. Compile your assembly code to 32 bit object file (creating `myprog.o`)
 - `nasm -f elf myprog.asm`
9. link the object files `myprog.o`, `asm_io.o`, `driver.o` (and the C libraries) to create the executable
 - `gcc -m32 driver.o myprog.o asm_io.o`
10. run the executable and see the output
 - `./a.out`

steps to run our program



K. N. Toosi
University of Technology



steps to run our program



K. N. Toosi
University of Technology

myprog.asm

```
%include "asm_io.inc"

segment .text

global asm_main

asm_main:
    enter 0,0
    pusha

    mov eax, 100
    mov ebx, 20
    sub eax, ebx

    call print_int ; print E
    call print_nl ; print a
    dump_regs 1111 ; print

    popa
    leave
    ret
```

driver.c

```
void asm_main();

int main() {

    asm_main();

    return 0;

}
```

```
nasihatkon@kntu:lecture3$ nasm -f elf -d ELF_TYPE asm_io.asm
nasihatkon@kntu:lecture3$ gcc -m32 -c driver.c
nasihatkon@kntu:lecture3$ nasm -f elf myprog.asm
nasihatkon@kntu:lecture3$ gcc -m32 driver.o myprog.o asm_io.o
nasihatkon@kntu:lecture3$ ./a.out
```

```
80
Register Dump # 1111
EAX = 00000050 EBX = 00000014 ECX = FFDBFCB0 EDX = FFDBFCD4
ESI = F76B7000 EDI = F76B7000 EBP = FFDBFC88 ESP = FFDBFC68
EIP = 080484EB FLAGS = 0206 PF
```

All commands



K. N. Toosi
University of Technology

```
nasm -f elf -d ELF_TYPE asm_io.asm
```

```
gcc -m32 -c driver.c
```

```
nasm -f elf myprog.asm
```

```
gcc -m32 driver.o myprog.o asm_io.o
```

```
./a.out
```

Give output file a name (-o option)



K. N. Toosi
University of Technology

```
nasm -f elf -d ELF_TYPE asm_io.asm
```

```
gcc -m32 -c driver.c
```

```
nasm -f elf myprog.asm
```

```
gcc -m32 driver.o myprog.o asm_io.o -o myprog
```

```
./myprog
```

Writing your own program



K. N. Toosi
University of Technology

- Take the same steps as above
- Your source file should be like this (or use `ske1.asm` from the book file)

```
%include "asm_io.inc"

segment .text

global asm_main

asm_main:
    enter 0,0
    pusha

    ; write your assembly code here!

    popa
    leave
    ret
```