
COMP-273

Starting a Program

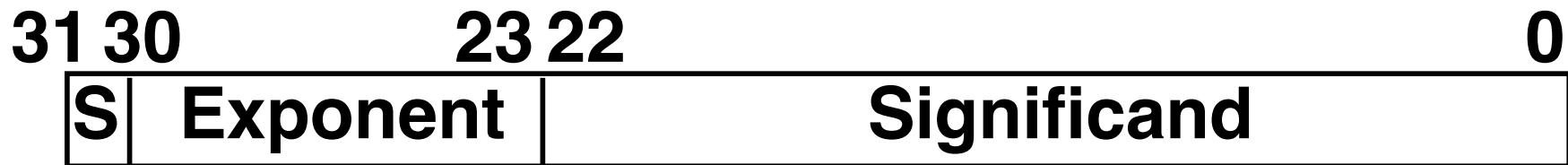
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

IEEE 754 Floating Point Review

° Summary (single precision):



1 bit 8 bits 23 bits
 ° $(-1)^S \times (1 + \frac{\text{Significand}}{2^{23}})^{(\text{Exponent}-127)}$

• Double precision: $(-1)^S \times (1 + \frac{\text{Significand}}{2^{52}})^{(\text{Exponent}-1023)}$
 except with exponent bias of 1023

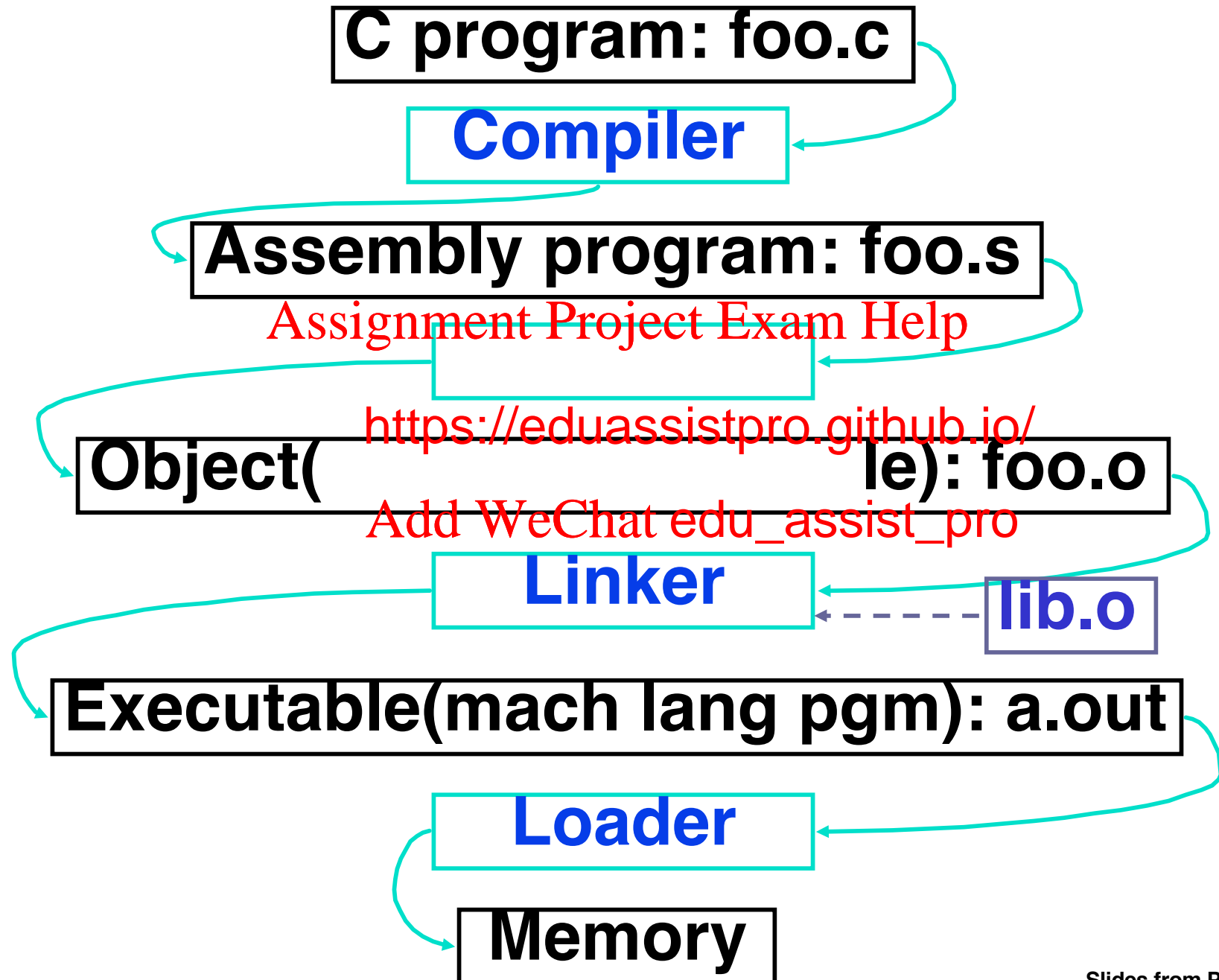
° Special reserved exponents for 0, infinity, NotANumber (NaN), and denorms (small numbers not in normalized)

° Multiply/Divide on MIPS use hi, lo registers

Outline

- **Compiler**
- **Assembler**
- **Linker** Assignment Project Exam Help
- **Loader** <https://eduassistpro.github.io/>
- **Example** Add WeChat edu_assist_pro

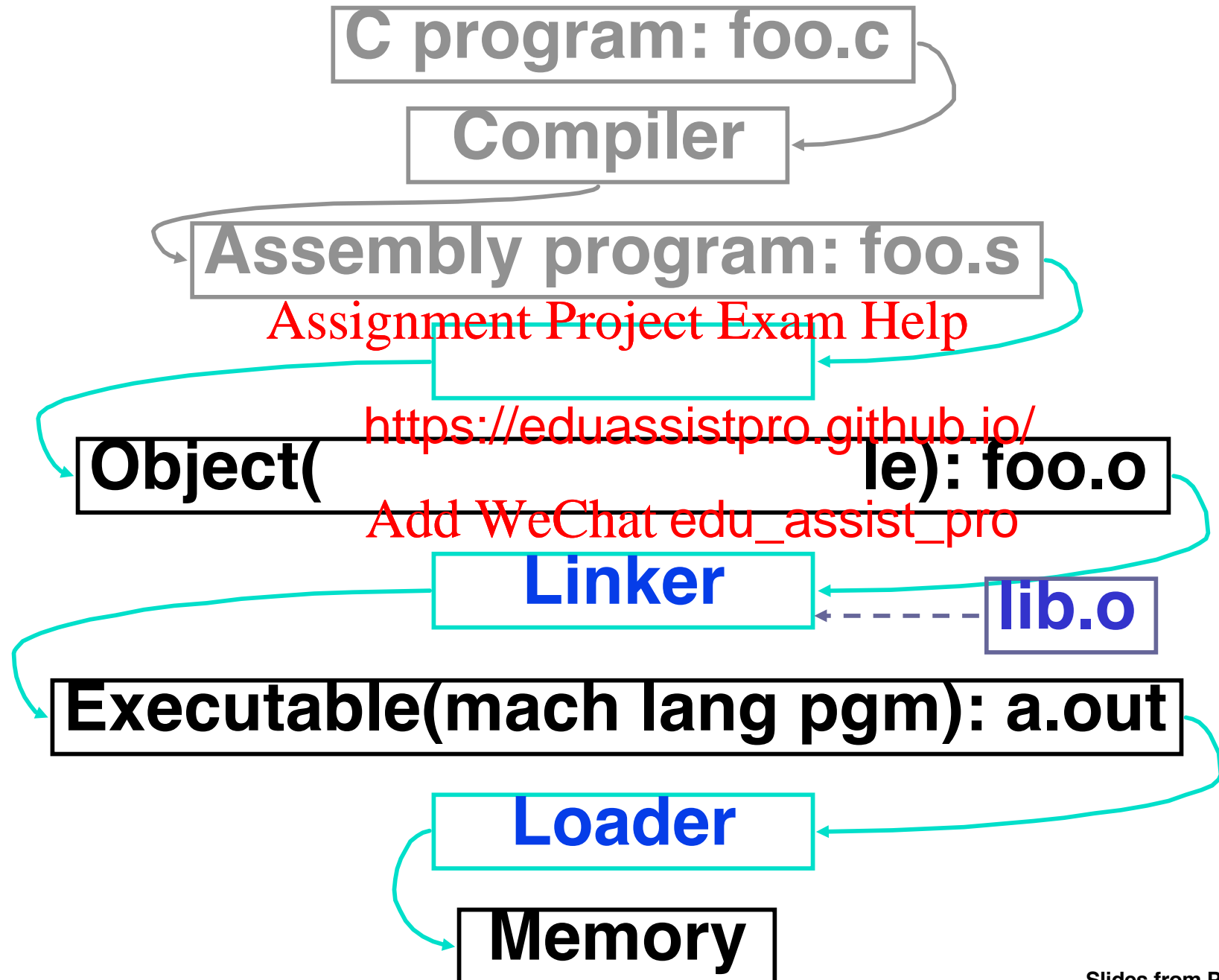
Steps to Starting a Program




Compiler

- **Input: High-Level Language Code**
(e.g., C, Java)
- **Output: Assembly Language Code**
(e.g., MIPS) [Assignment Project Exam Help](https://eduassistpro.github.io/)
- **Note: Out pseudoinstructions** <https://eduassistpro.github.io/>
[Add WeChat edu_assist_pro](#)
 - **Pseudoinstructions**: instructions that assembler understands but not in machine

Where Are We Now?



Assembler

- Reads and Uses **Directives**
- Replaces Pseudoinstructions
- Produces Machine Language
- Creates  <https://eduassistpro.github.io/>
Add WeChat edu_assist_pro

Assembler Directives (p. A-51 to A-53)

- Give directions to assembler, but does not produce machine instructions

.text: Subsequent items put in user text (instructions) segment

.data: Subsequent items put in user data segment

.globl sym: declares `sym` and can be referenced from other files

.ascii str: Store the string `str` in memory and null-terminate it

.word w1...wn: Store the n 32-bit quantities in successive memory words

Pseudoinstruction Replacement

- Asm. treats convenient variations of machine language instructions as if real instructions

Pseudo (MAL): Real (TAL):

subu \$sp, \$sp, 32 addiu \$sp, \$sp, -32

sd \$a0, 32(\$sp) sd \$a0, (\$sp)

addu \$t0, \$t6, 1 addi \$t0, \$t6, 1

ble \$t0, 100, loop slti \$at, \$t0, 101
bne \$at, \$0, loop

la \$a0, str lui \$at, left(str)
ori \$a0, \$at, right(str)

mul \$t7, \$t6, \$t6 mult \$14, \$14
mflo \$15

Producing Machine Language (1/2)

- **Simple instructions for Assembler**

- **Arithmetic, Logical, Shifts, and so on.**
- **All necessary info is within the instruction already.**

Assignment Project Exam Help

- **What about** <https://eduassistpro.github.io/>

- **PC-Relative** [Add WeChat edu_assist_pro](#)
- **So once pseudoinstructions are replaced by real ones, we know by how many instructions to branch.**

- **So these 2 cases are handled easily.**

Producing Machine Language (2/2)

- What about jumps (j and jal)?
 - Jumps require **absolute address**.
- What about references to data?
 - la gets broken up into lui and ori
 - These will be 2-bit address of the data.
- These can't be determined yet, must wait to see where this code will appear in final program.
- Two tables are used to help assembly and later resolution of addresses

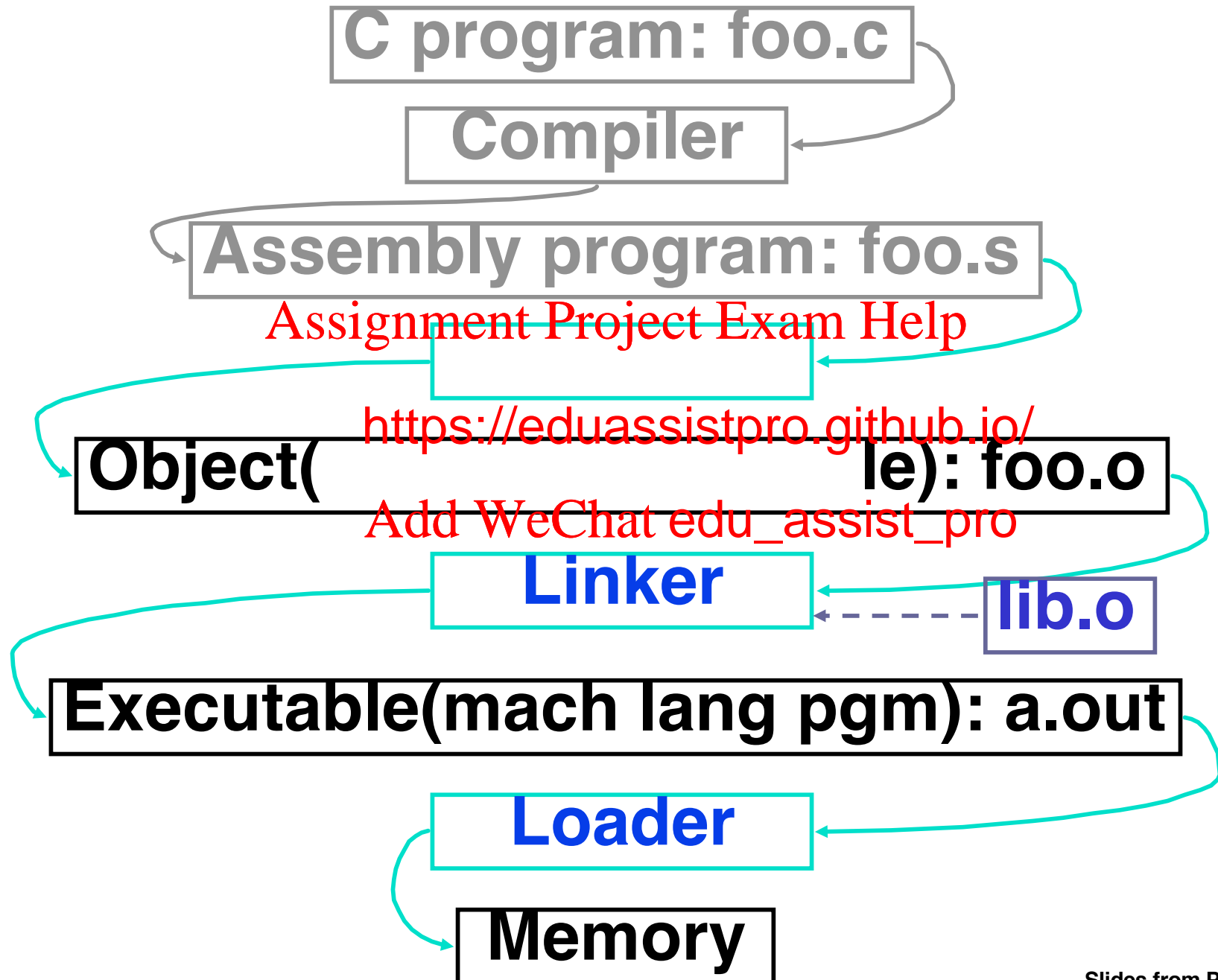
1st Table: Symbol Table

- **Symbol table**: List of “items” in this file that may be used by this and other files.
- What are they?
 - **Labels**: function calling
 - **Data**: anyt variables files
- First Pass: record label-address pairs
- Second Pass: produce machine code
 - Result: can jump to a label later in code without first declaring it

2nd Table: Relocation Table

- **Relocation Table**: line numbers of “items” for this file which need the address filled in (or fixed up) later.
- **What are they?**
 - Any label
 - Internal (i.e. label in file)
 - external (including lib files)
 - Any absolute address of piece of data
 - such as used by the `la` pseudo-instruction:
`la $destination, label`

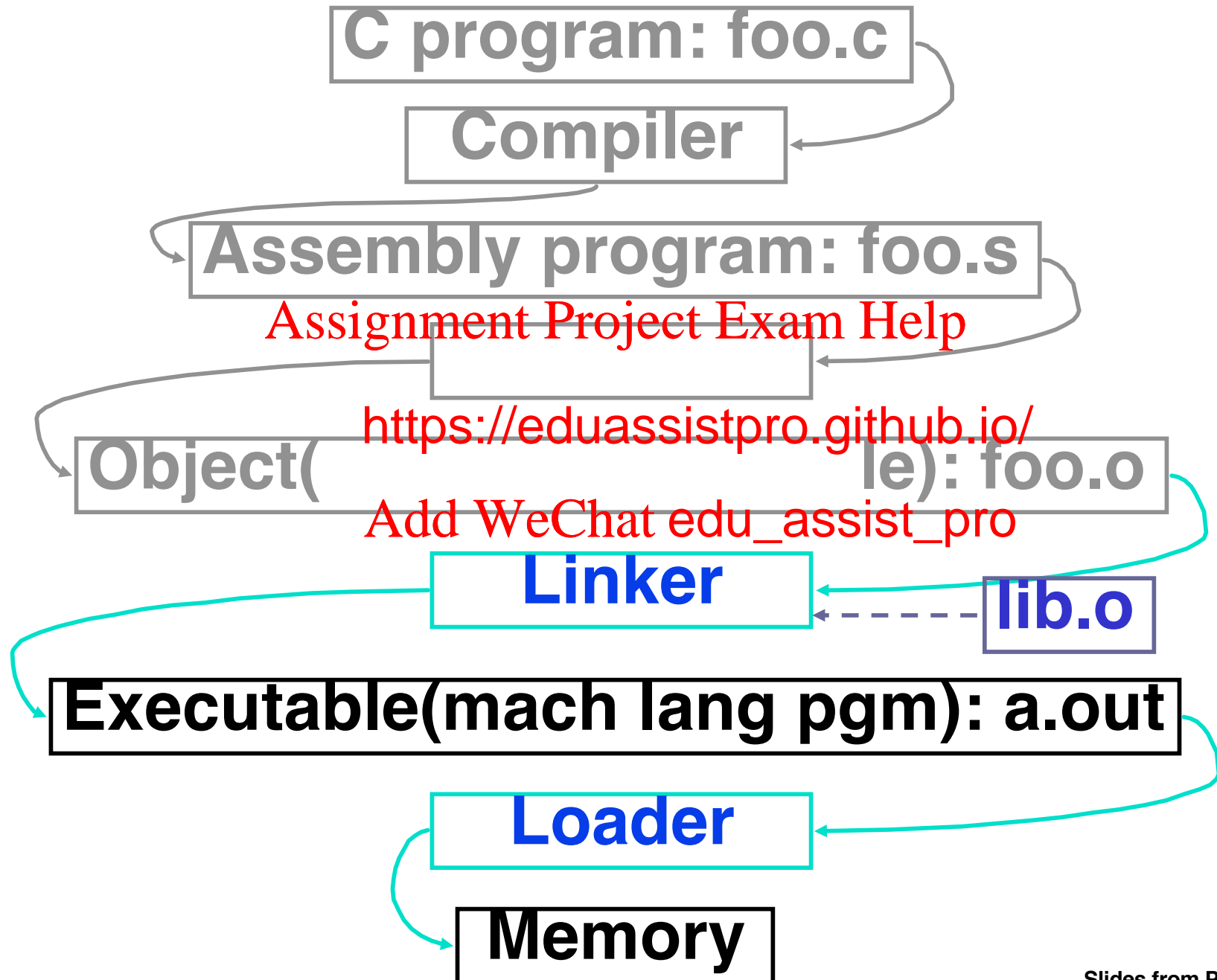
Where Are We Now?



Object File Format

- object file header: size and position of the other pieces of the object file
- text segment: the machine code
- data segment: binary representation of the data in <https://eduassistpro.github.io/>
- relocation table: list of addresses of code that need to be “handled”
- symbol table: list of this file’s labels and data that can be referenced
- debugging information

Where Are We Now?



Link Editor/Linker (1/2)

- What does Link Editor do?
- Combines several object (.o) files into a single executable (“linking”)
- Enables **S** **tion of files**
 - Changes **require**
recompilation of wh **rain**
 - Windows source is >50 M lines of code! And Growing!
 - Code in file called a module
 - Link Editor name from editing the “links” in jump and link instructions

Link Editor/Linker (2/2)

- **Step 1: Take text segment from each .o file and put them together.**
- **Step 2: Take data segment from each .o file, put them together, and concatenate segments of text.**
Assignment Project Exam Help
<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro
- **Step 3: Resolve Ref**
 - **Go through Relocation Table and handle each entry using the Symbol Table**
 - **That is, fill in all absolute addresses**

Four Types of Addresses

- **PC-Relative Addressing (beq, bne):**
never fix up (never “relocate”)
- **Absolute Address (j, jal):**
always relocate
- **External Reference (jalr):**
always relocate
- **Symbolic Data Reference (often lui and ori, for la):** always relocate

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Resolving References (1/2)

- Linker **assumes** first word of first text segment is at address 0x00000000.
- Linker knows:
 - length of a segment
 - ordering <https://eduassistpro.github.io/> segments
- Linker calculates:
 - absolute address of each label to be jumped to (internal or external) and each piece of data being referenced

Resolving References (2/2)

◦ To resolve references:

- search for reference (data or label) in all symbol tables
- if not found, search in library files (for example)
- once absolute address is determined, fill in the machine code

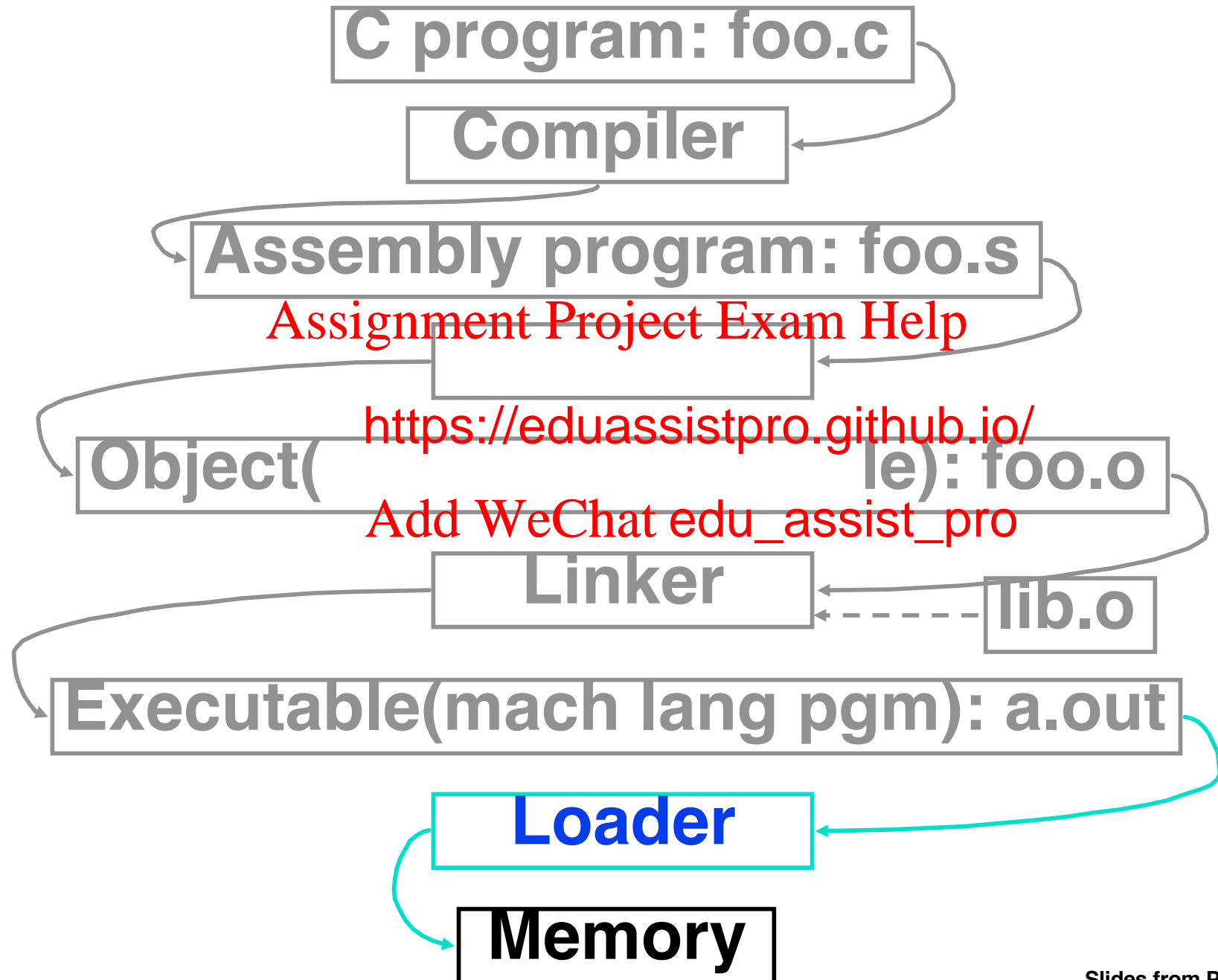
Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- ## ◦ Output of linker: executable file containing text and data (plus header)
- ## ◦ May not have library object files resolved if dynamically loaded

Where Are We Now?



Loader (1/3)

- Executable files are stored on disk.
- When one is to be run, loader's job is to load it into memory and start it running.
- In reality, loading is one of the OS tasks

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Loader (2/3)

- So what does a loader do?
- Reads executable file's header to determine size of text and data segments
- Creates new program I/O data segments, along with a stack segment
- Copies instructions and data from executable file into the new address space

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

e for
old text and
a stack

Loader (3/3)

- Copies arguments passed to the program onto the stack
- Initializes machine registers
 - Most registers assigned location
- Jumps to start-up routine that copies program's arguments from stack to registers and sets the PC
 - If main routine returns, start-up routine terminates program with the exit system call

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

stack pointer
e stack

Dynamic Linking

- Some operating systems allow “dynamic linking”
- Both the loader *and* the linker are part of the operating system so modules can be linked at runtime
<https://eduassistpro.github.io/>
- If a module is needed already loaded, it need not be loaded again
Add WeChat edu_assist_pro
- Called DLLs

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

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

```
int main (int argc, char *argv[]) {
```

```
    int i;
```

```
    int prod = 0;
```

<https://eduassistpro.github.io/>

```
    for (i = 0; i <= 100; i++)  
        prod = prod + i * i;
```

```
    printf ("The product from 0 .. 100 is  
    %d\n", prod);
```

```
}
```

Example: C \Rightarrow Asm \Rightarrow Obj \Rightarrow Exe \Rightarrow Run

```
.text
- .align 2
  .globl main
main:
  subu $sp, $sp, 32
  sw $ra, 20($sp)
  sd $a0, 32
  sw $0, 24($sp)
  sw $0, 28($sp)
loop:
  lw $t6, 28($sp)
  mul $t7, $t6, $t6
  lw $t8, 24($sp)
  addu $t9, $t8, $t7
  sw $t9, 24($sp)
```

```
addu $t0, $t6, 1
sw $t0, 28($sp)
ble $t0, 100, loop
la $a0, str
lw $a1, 24($sp)
jal printf
sv0, $0
sw $0, 20($sp)
sw $0, $sp, $sp, 32
j $ra
.data
.align 0
str:
  .asciiz "The
- product from 0
  .. 100 is %d\n"
```

Example: C \Rightarrow Asm \Rightarrow Obj \Rightarrow Exe \Rightarrow Run

- Remove pseudoinstructions, assign addresses

00 **addiu** \$29,\$29,-32

04 **sw** \$31,20(\$29)

08 **sw** \$4,32(\$29)

0c **sw** \$5,36(\$29)

10 **sw** \$0,

14 **sw** \$0,

18 **lw** \$14,28(\$29)

1c **mult** \$14,\$14

20 **mflo** \$15

24 **lw** \$24,24(\$29)

28 **addu** \$25,\$24,\$15

2c **sw** \$25,24(\$29)

30 **addiu** \$8,\$14,1

34 **sw** \$8,28(\$29)

38 **slti** \$1,\$8,101

3c **bne** \$1,\$0,loop

\$4,1.str

\$4,\$4,r.str

\$5,24(\$29)

4c **jal** printf

50 **addu** \$2,\$0,\$0

54 **lw** \$31,20(\$29)

58 **addiu** \$29,\$29,32

5c **jr** \$31

Symbol Table Entries

◦ Symbol Table

- Label Address

main: 0x00000000

loop: 0x

str: 0x

printf: 0x00000000

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

◦ Relocation Table

- Address

• 0x0000004c

- Instr. Type Dependency

jal

printf

Example: C \Rightarrow Asm \Rightarrow Obj \Rightarrow Exe \Rightarrow Run

•Edit Local Addresses

00	addiu	\$29,\$29,-32	30	addiu	\$8,\$14, 1
04	sw	\$31,20 (\$29)	34	sw	\$8,28 (\$29)
08	sw	\$4, 32 (\$29)	38	slti	\$1,\$8, 101
0c	sw	\$5,36 (\$29)	3c	lne	\$1,\$0, <u>-10</u>
10	sw	\$0,		\$4,	<u>0x1000</u>
14	sw	\$0,		\$4,	<u>0x0430</u>
18	lw	\$14, 28 (\$29)		\$5,	24 (\$29)
1c	multu	\$14, \$14	4c	jal	<u>0</u>
20	mflo	\$15	50	addu	\$2, \$0, \$0
24	lw	\$24, 24 (\$29)	54	lw	\$31,20 (\$29)
28	addu	\$25,\$24,\$15	58	addiu	\$29,\$29,32
2c	sw	\$25, 24 (\$29)	5c	jr	\$31

•Next Generate object file

Example: C \Rightarrow Asm \Rightarrow Obj \Rightarrow Exe \Rightarrow Run

0x000000	00100111101111011111111111111111000000
0x000004	1010111111011111110000000000000010100
0x000008	10101111110100100000000000000001000000
0x00000c	1010111111010010100000000000000100100
0x000010	1010111111010000000000000000000011000
0x000014	1010111111010000000000000000000011100
0x000018	1000111111010111000000000000000011100
0x00001c	000000011100111000000000000000011001
0x000020	000000000000000000000000000000010010
0000000000000000	10
1000000000000000	00
0000000000000000	10
0x000030	00100101111000000000000000000001
0x000034	101011111101111100000000000000011000
0x000038	00101001000000000000000000000001100101
0x00003c	00010100000010000000111111111111110111
0x000040	00111100000000001000000100000000000000
0x000044	00110100100000100000000010000001100000
0x000048	1000111111010010100000000000000011000
0x00004c	0000110000001000000000000000011101100
0x000050	00000000000000000000000000000001000001
0x000054	1000111111011111110000000000000010100
0x000058	0010011111011111010000000000000100000
0x00005c	00000001111110000000000000000000000001000

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Example: C \Rightarrow Asm \Rightarrow Obj \Rightarrow Exe \Rightarrow Run

- Combine with object file containing “printf”.
- Edit absolute addresses: in this case edit `jal printf` to contain actual address of
<https://eduassistpro.github.io/>
- Output single binar
Add WeChat edu_assist_pro

Things to Remember 1/3

- **Stored Program concept means instructions just like data, so can take data from storage, and keep transforming it until load registers and jump to routine to begin execution**
- **Compiler (⇒ Loader)** **Linker**
<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro
- **Assembler does 2 things to resolve addresses, handling internal forward references**
- **Linker enables separate compilation, libraries that need not be compiled, and resolves remaining addresses**

Things to Remember (2/3)

- Compiler converts a single HLL file into a single assembly language file.
- Assembler removes pseudoinstructions, converts what it can to machine language, linker (relocates) each .s file into a .o file.
<https://eduassistpro.github.io/>
Add WeChat edu_assist_pro
- Linker combines several .o files and resolves absolute addresses.
- Loader loads executable into memory and begins execution.

Things to Remember (3/3)

