
COMP-273

Starting a Program

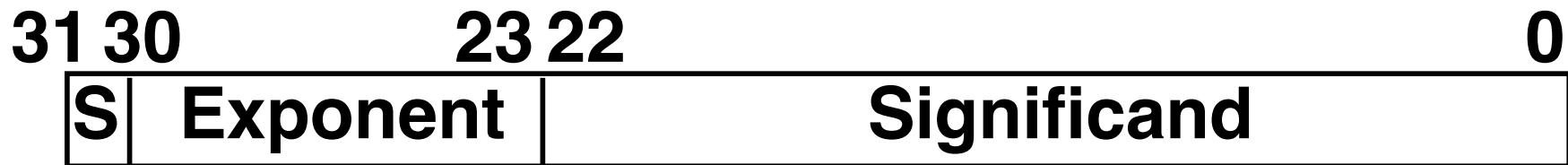
Assignment Project Exam Help

Kaleem Siddiqi
<https://powcoder.com>

Add WeChat powcoder

IEEE 754 Floating Point Review

° Summary (single precision):



1 bit 8 bits 23 bits

° $(-1)^S \times (1 + \text{Significand}) \times 2^{(\text{Exponent}-127)}$

- Double precision identical, except with exponent bias of 1023

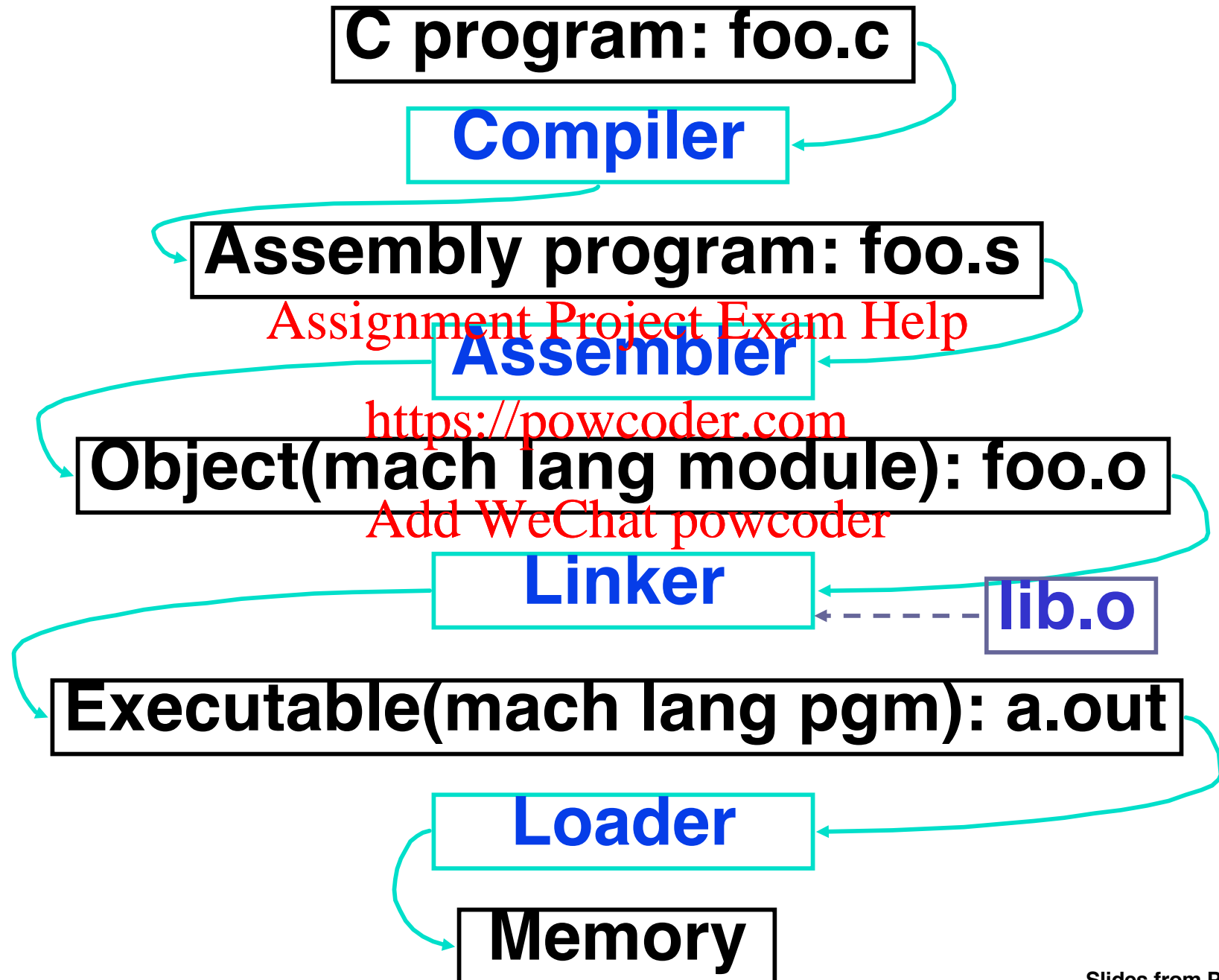
° Special reserved exponents for 0, infinity, NotANumber (NaN), and denorms (small numbers not in normalized)

° Multiply/Divide on MIPS use hi, lo registers

Outline

- **Compiler**
- **Assembler**
- **Linker** Assignment Project Exam Help
- **Loader** <https://powcoder.com>
- **Example** Add WeChat powcoder

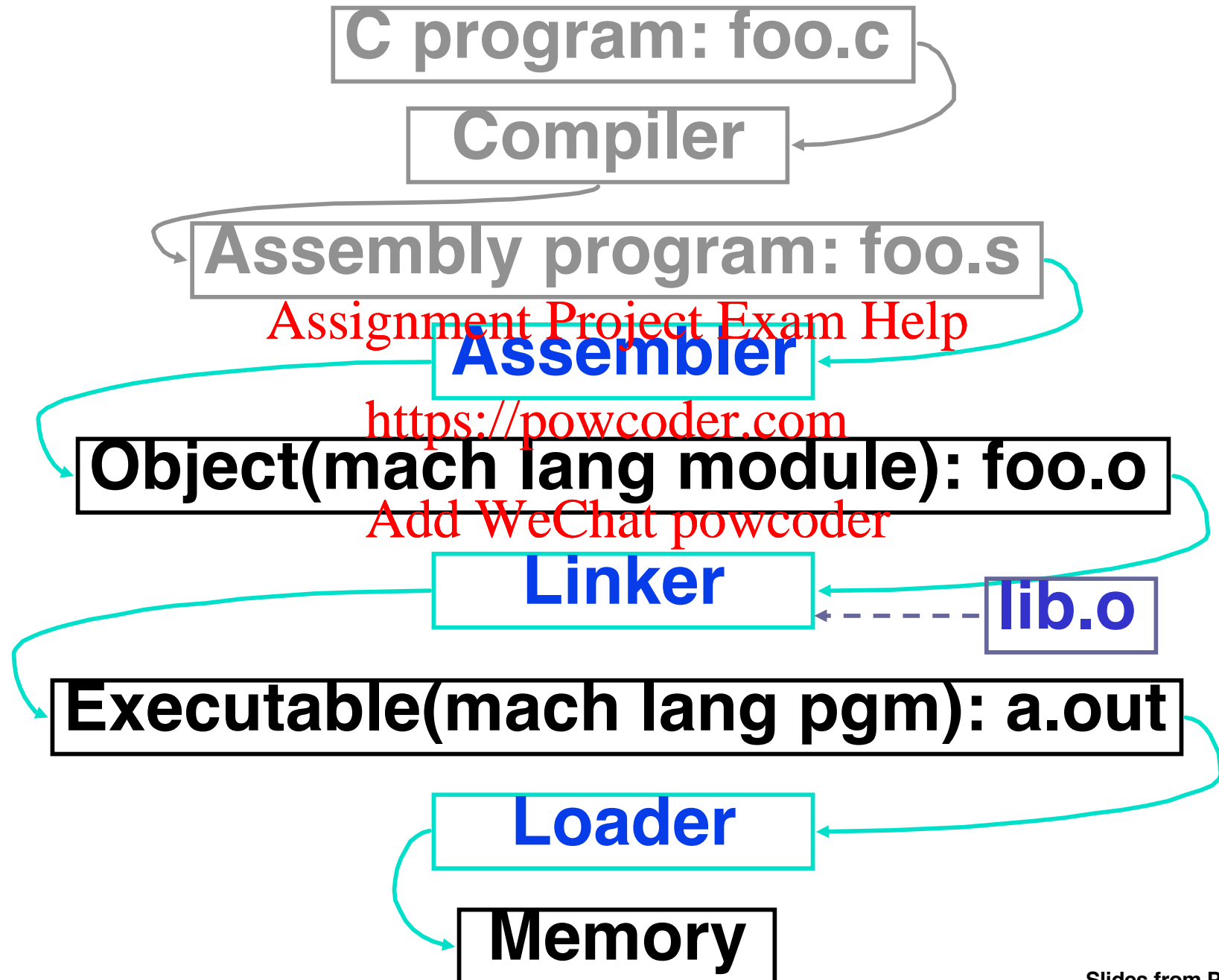
Steps to Starting a Program



Compiler

- Input: High-Level Language Code (e.g., C, Java)
- Output: Assembly Language Code (e.g., MIPS)
Assignment Project Exam Help
- Note: Output may contain pseudoinstructions
<https://powcoder.com>
Add WeChat powcoder
 - Pseudoinstructions: instructions that assembler understands but not in machine

Where Are We Now?



Assembler

- Reads and Uses **Directives**
 - Replaces Pseudoinstructions
 - Produces Machine Language
 - Creates **Object File**
- <https://powcoder.com>
Add WeChat powcoder

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** global 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) sw \$a0, 32(\$sp)
 sw \$a1, 36(\$sp)

addu \$t0, \$t6, 1 addiu \$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 Branches?**

<https://powcoder.com>

- **PC-Relative**
- **So once pseudoinstructions are replaced by real ones, we know by how many instructions to branch.**

Add WeChat powcoder

- **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 require the full 32-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

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

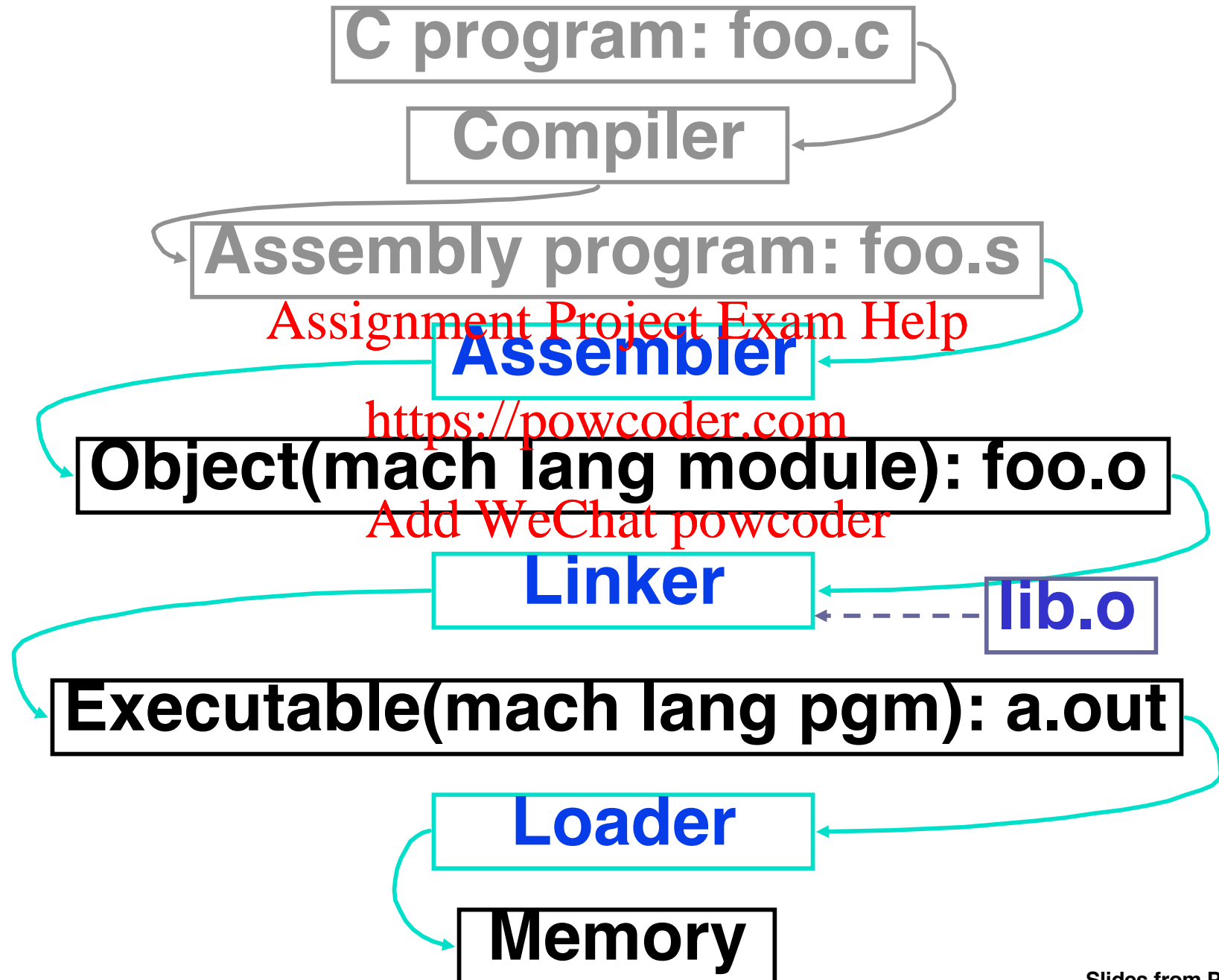
1st Table: Symbol Table

- **Symbol table**: List of “items” in this file that may be used by this and other files.
- What are they?
 - **Labels**: **function calling** Assignment Project Exam Help
 - **Data**: anything in the **data section**; variables which may be accessed across files <https://powcoder.com>
Add WeChat powcoder
- 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 jumped to: `j or jal`
 - Internal (i.e. label inside this file)
 - external (including lib files)
 - Any absolute address of piece of data
 - such as used by the `la` pseudo-instruction:
`la $destination, label`

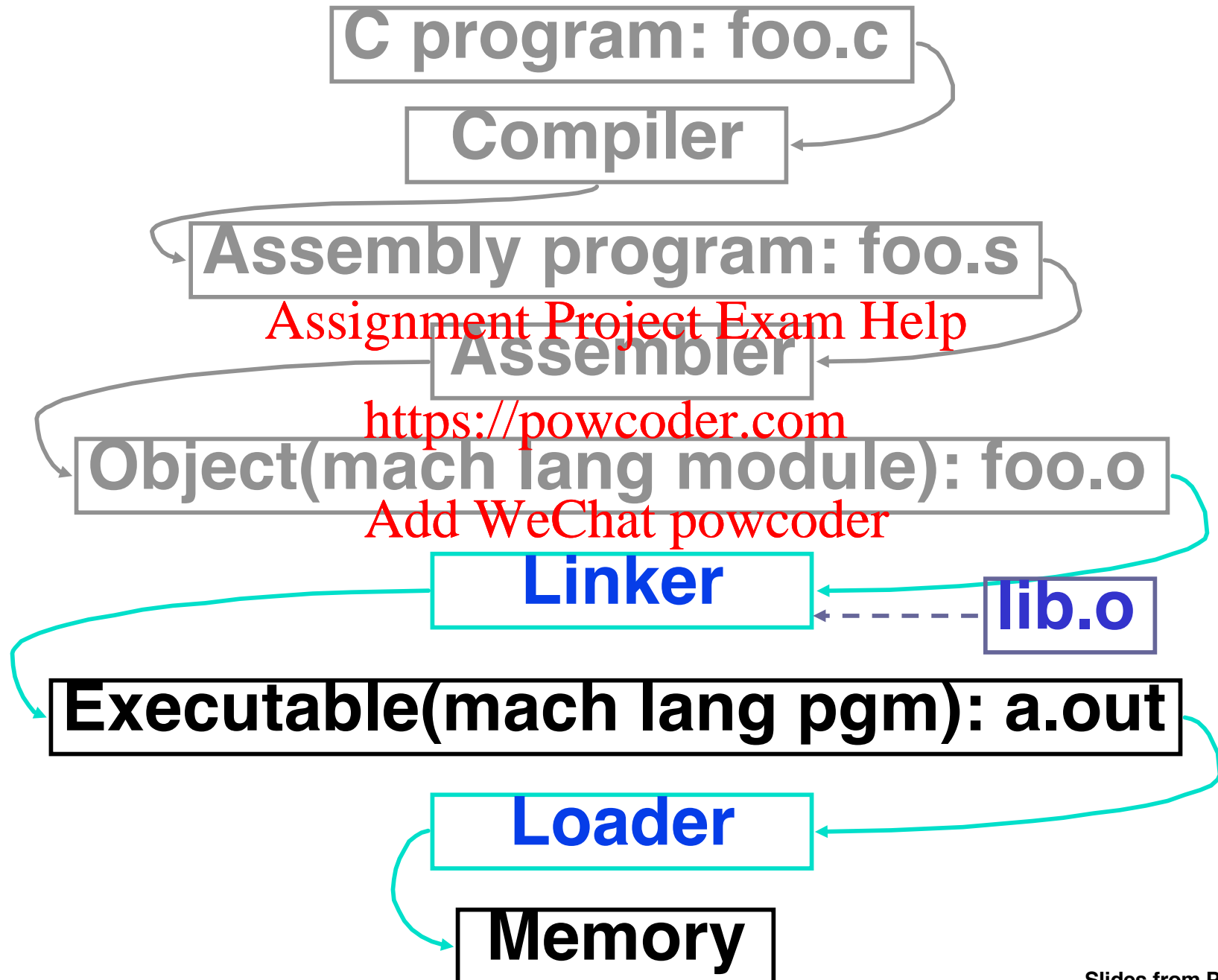
Where Are We Now?



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 the source file
Assignment Project Exam Help
<https://powcoder.com>
- relocation table: identifies lines of code that need to be “handled”
Add WeChat powcoder
- 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”)
Assignment Project Exam Help
- Enables Separate Compilation of files
<https://powcoder.com>
Add WeChat powcoder
 - Changes to one file do not require recompilation of whole program
 - 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 this onto end of text segments.**
Assignment Project Exam Help
<https://powcoder.com>
- **Step 3: Resolve References**
Add WeChat powcoder
 - **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 (usually jal):**
always relocate
- **Symbolic Data Reference (often lui and ori, for la):** always relocate

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Resolving References (1/2)

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

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Resolving References (2/2)

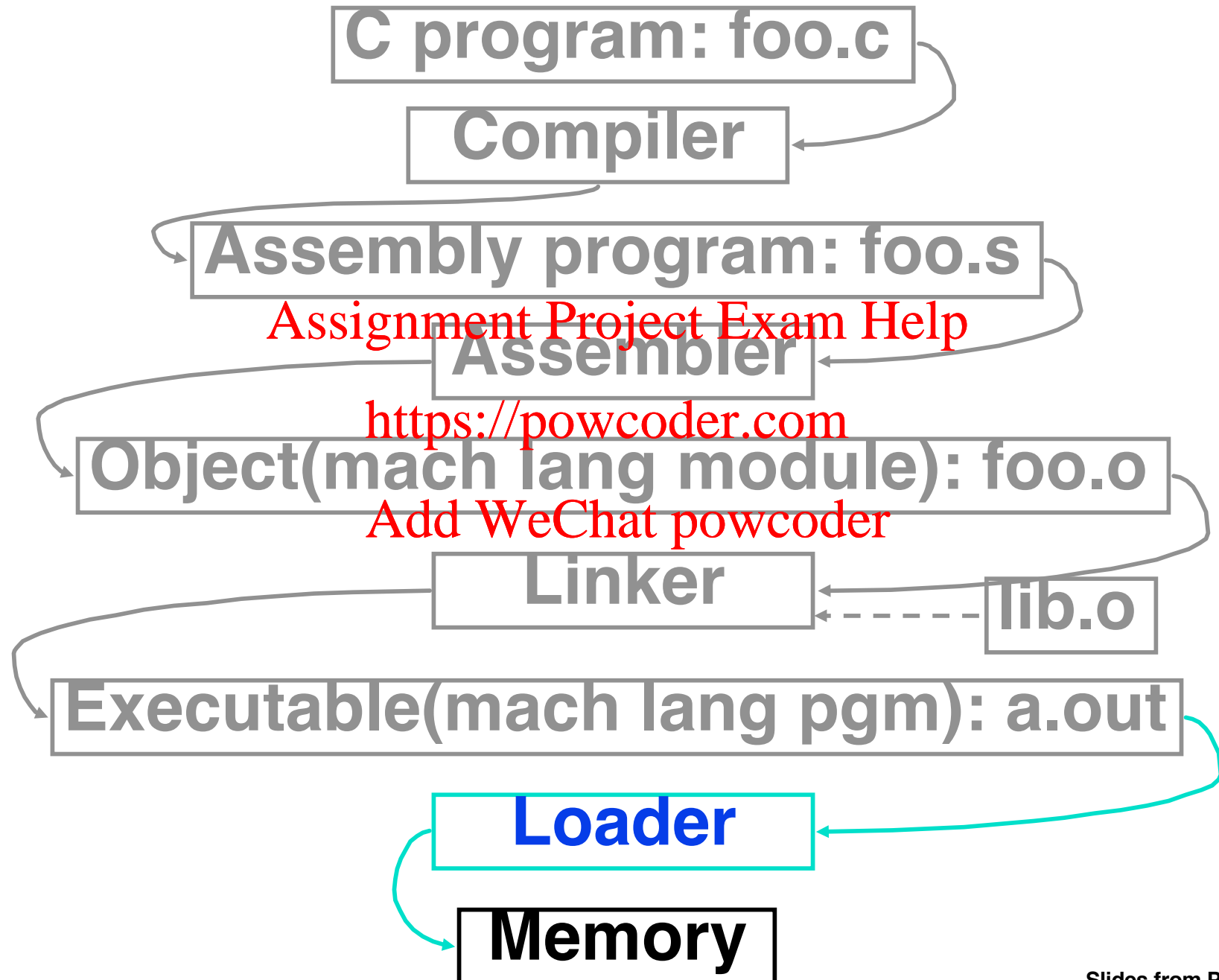
◦ To resolve references:

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

◦ 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. [Assignment Project Exam Help](https://powcoder.com)
- In reality, loader is the operating system (OS) <https://powcoder.com>
Add WeChat powcoder
 - loading is one of the OS tasks

Loader (2/3)

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

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Loader (3/3)

- Copies arguments passed to the program onto the stack
- Initializes machine registers
 - Most registers cleared, but stack pointer assigned address of 1st free stack 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://powcoder.com>

Add WeChat powcoder

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 and loaded at runtime
<https://powcoder.com>
- If a module is needed and already loaded, it need not be loaded again
Add WeChat powcoder
- Called DLLs

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

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

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

```
    int i;
```

```
    int prod = 0;
```

<https://powcoder.com>

```
    for (i = 0; i <= 100; i = i + 1)
        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 $a0, 32($sp)
  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
move $v0, $0
lw $ra, 20($sp)
addiu $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	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	bne	\$1, \$0, loop
10	sw	\$0, 24 (\$29)	40	lui	\$4, 1.str
14	sw	\$0, 28 (\$29)	44	ori	\$4, \$4, r.str
18	lw	\$14, 28 (\$29)	48	lw	\$5, 24 (\$29)
1c	mult	\$14, \$14	4c	jal	printf
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

Symbol Table Entries

◦ Symbol Table

- Label Address

main: 0x00000000

loop: 0x00000018

str: 0x10000430

printf: 0x00000000

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

◦ 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, 24 (\$29)	40	lui	\$4, <u>0x1000</u>
14	sw	\$0, 28 (\$29)	44	ori	\$4,\$4, <u>0x0430</u>
18	lw	\$14, 28 (\$29)	48	lw	\$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	001001111011110111111111111111000000
0x000004	10101111110111111000000000000010100
0x000008	10101111110100100000000000000100000
0x00000c	10101111110100101000000000000100100
0x000010	10101111110100000000000000000011000
0x000014	10101111110100000000000000000011100
0x000018	10001111110101110000000000000011100
0x00001c	0000000111100111000000000000011001
0x000020	0000000000000000000000000000010010
0x000024	10001111110111000000000000000011000
0x000028	00000001100000111111001000000100001
0x00002c	1010111111010100000000000000011100
0x000030	00100101111001000000000000000000001
0x000034	10101111110111001000000000000011000
0x000038	001010010000000010000000000001100101
0x00003c	00010100000100000011111111111110111
0x000040	0011110000000010000001000000000000
0x000044	00110100100000100000000100000110000
0x000048	10001111110100101000000000000011000
0x00004c	000011000000100000000000000011101100
0x000050	00000000000000000000000010000000100001
0x000054	10001111110111111000000000000010100
0x000058	00100111110111101000000000000100000
0x00005c	00000001111110000000000000000000001000

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

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 printf.
<https://powcoder.com>
- Output single binary file.
Add WeChat powcoder

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**
Assignment Project Exam Help
- **Compiler \Rightarrow Assembler \Rightarrow Linker (\Rightarrow Loader)**
<https://powcoder.com>
Add WeChat powcoder
- **Assembler does 2 passes 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, and creates a checklist for the linker (relocation table). This changes each .s file into a .o file.
- Linker combines several .o files and resolves absolute addresses.
- Loader loads executable into memory and begins execution.

Things to Remember (3/3)

