

ELEC2041

Microprocessors and Interfacing

Lectures 25: Compiler, Assembler, Linker and Loader – II

<http://webct.edtec.unsw.edu.au/>

May 2006

Saeid Nooshabadi

saeid@unsw.edu.au

ELEC2041 lec25-linker.II.1

Saeid Nooshabadi

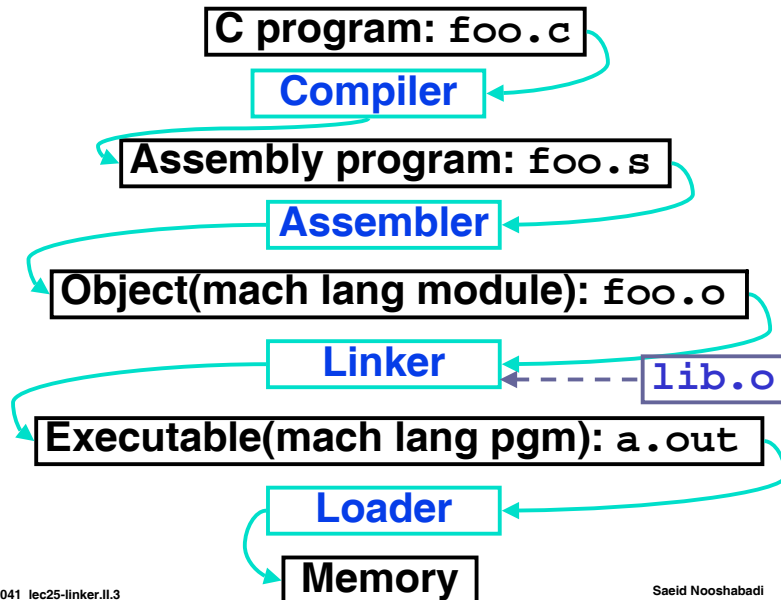
Overview

- Assembler
- Linker
- Loader
- Example

ELEC2041 lec25-linker.II.2

Saeid Nooshabadi

Review: Steps to Starting a Program



ELEC2041 lec25-linker.II.3

Saeid Nooshabadi

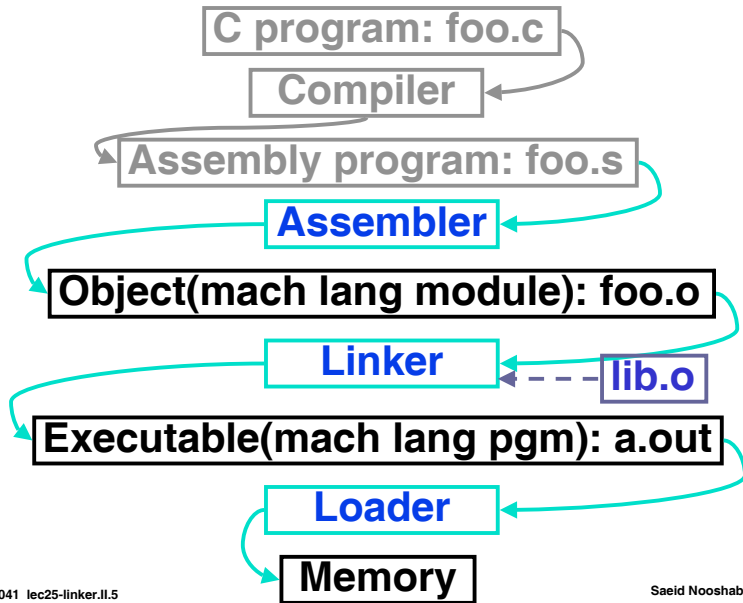
Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

```
extern int posmul(int mlier, int mcand);
int main (void)
{
    char *MMSG = "Multiplication";
    static int a=20,b=18, c;
    c = posmul(a, b);
    return c;
}
```

ELEC2041 lec25-linker.II.4

Saeid Nooshabadi

Review: Where Are We Now?



ELEC2041 lec25-linker.II.5

Saeid Nooshabadi

Example: C ⇒ **Asm** ⇒ Obj ⇒ Exe ⇒ Run (#1/2)

```

.data      ; assembler directive =>
           ; following define words
           ; in static data segment

a_b: .word 20, 18 ;two 32-bit ints a & b

c:  .space 4      ; 4 bytes space for c

.align 2    ; next word aligned 2^2
MSG:  .asciz "Multiplication"

           ; 13 ascii bytes
  
```

ELEC2041 lec25-linker.II.6

Saeid Nooshabadi

Example: C ⇒ **Asm** ⇒ Obj ⇒ Exe ⇒ Run (#2/2)

```

.text
    .align 2
    .global main

main:
    ldr a3, =MSG          ; Pseudo Inst. to
                        ; load address of
                        ; MSG string
    ldr a1, =(a_b)        ; instructions to
    ldr a1, [a1]          ; read a
    ldr a2, =(a_b+4)      ; instructions to
    ldr a2, [a2]          ; read b
    bl posmul             ; call to multiply
                        ; function
    ldr a2, =c            ; instructions to
    str a1, [a2]          ; save c
    mov pc, lr
  
```

ELEC2041 lec25-linker.II.7

Saeid Nooshabadi

Example: C ⇒ **Asm** ⇒ **Obj** ⇒ Exe ⇒ Run

```

Disassembly of section .text:
00000000 <main>:
0: e59f201c ldr a3, [pc, #28] ; 24 <main+0x24>
4: e59f001c ldr a1, [pc, #28] ; 28 <main+0x28>
8: e5900000 ldr a1, [a1]
c: e59f1018 ldr a2, [pc, #24] ; 2c <main+0x2c>
10: e5911000 ldr a2, [a2]
14: ebfffffe bl 0 <main>
18: e59f1010 ldr a2, [pc, #16] ; 30 <main+0x30>
1c: e5810000 str a1, [a2]
20: e1a0f00e mov pc, lr
24: 0000000c andeq r0, r0, ip; address of MSG
28: 00000000 andeq r0, r0, r0; address of a
2c: 00000004 andeq r0, r0, r4; address of b
30: 00000008 andeq r0, r0, r8; address of c
Disassembly of section .data:
00000000 <a_b>:
0: 00000014 andeq r0, r0, r4, lsl r0
4: 00000012 andeq r0, r0, r2, lsl r0
00000008 <c>:
8: 00000000 andeq r0, r0, r0
0000000c <MSG>:
c: 746c754d strvcbt r5, [ip], -#1357 ; "Multiplication"
10: 696c7069 stmvsdb ip!,{r0,r3,r5,r6,ip,sp,lr}^
14: 69746163 ldmvsdb r4!,{r0,r1,r5,r6,r8,sp,lr}^
18: 00006e6f andeq r6, r0, pc, ror #28
  
```

ELEC2041 lec25-linker.II.8

Saeid Nooshabadi

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run (#1/3)

Disassembly of section .text:

```
00000000 <main>:
0: e59f201c ldr a3, [pc, #28] ; 24 <main+0x24>
4: e59f001c ldr a1, [pc, #28] ; 28 <main+0x28>
8: e5900000 ldr a1, [a1]
c: e59f1018 ldr a2, [pc, #24] ; 2c <main+0x2c>
10: e5911000 ldr a2, [a2]
14: ebfffffe bl 0 <main>; Address unknown
18: e59f1010 ldr a2, [pc, #16] ; 30 <main+0x30>
1c: e5810000 str a1, [a2]
20: e1a0f00e mov pc, lr
```

ELEC2041 lec25-linker.II.9

Saeid Nooshabadi

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run (#2/3)

```
24: 0000000c andeq r0, r0, ip; address of MSG
28: 00000000 andeq r0, r0, r0; address of a
2c: 00000004 andeq r0, r0, r4; address of b
30: 00000008 andeq r0, r0, r8; address of c
```

Literal Pool

ELEC2041 lec25-linker.II.10

Saeid Nooshabadi

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run (#3/3)

Disassembly of section .data:

```
00000000 <a_b>:
0: 00000014 andeq r0, r0, r4, lsl r0; 0x14=20
4: 00000012 andeq r0, r0, r2, lsl r0; 0x12=18
00000008 <c>:
8: 00000000 andeq r0, r0, r0
0000000c <MSG>: "Multiplication"
c: 746c754d strvcbt r5, [ip], -#1357;
10: 696c7069 stmvldb ip!, {r0, r3, r5, r6, ip, sp, lr}^
14: 69746163 ldmvldb r4!, {r0, r1, r5, r6, r8, sp, lr}^
18: 00006e6f andeq r6, r0, pc, ror #28
```

ELEC2041 lec25-linker.II.11

Saeid Nooshabadi

Typical Two Pass Assembly

- ° Construct basic layout of data segment and code segment
- ° Fill in all the opcodes, register numbers, literals, and initial data values
- ° Record the address associated with each label
- ° Then go back and put the address into fields with symbolic labels

MSG:	on0
	cati
	ipli
	Mult
c:	0
	18
a_b:	20

	e5900000
	e59f001c
main:	e59f201c

ELEC2041 lec25-linker.II.12

Saeid Nooshabadi

Producing Machine Language (#1/3)

◦ Simple Case

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

Producing Machine Language (#2/3)

- What about data in the data segment.
 - Accessed via (`ldr rdest, =label`)
 - Addresses stored in literal pool
 - Literal pool Accessed via PC-Relative
 - We can easily compute by how many instructions the PC offset to literal pool is.
 - Same thing for `ldr rdeast, =imm32`. loaded from literal pool if **cannot** be done via `mov` and `movn` instructions

Producing Machine Language (#2/3)

◦ What about data and labels in the text segment

- Accessed via
 - `ldr/str rdest, label`
 - `adr rdest, label`
 - `ladr rdest, label`
- All are PC Relative Addressing
- So once pseudoinstructions are replaced by real ones, we know by how many instructions the PC offset to label is

Producing Machine Language (#3/3)

- What about jumps (b and bl)?
 - Branches to other labels (addresses) in the same module and other modules require **absolute addresses of the labels**, to generate pc relative addresses:
`address@label - (pc + 8)`
- These can't be determined yet, so we create two tables...
- Example

```
14: ebfffffe bl    0 <main>
; Branch to unknown Address unknown
```

Symbol Table

- List of “items” in this file that may be used by other files.
- What are they?
 - Labels: branching & function calling
 - Data: anything in the `.data` section; variables which may be accessed across files
- First Pass: record label-address pairs
- Second Pass: produce machine code
 - Result: can jump to a later label without first declaring it (**forward referencing**)

Example Symbol Table

```
SYMBOL TABLE:
00000000 1      .data 00000000 a_b
00000008 1      .data 00000000 c
0000000c 1      .data 00000000 MSG
00000000 g      .text 00000000 main
00000000      *UND* 00000000 posmul
```

l; scope is local (not visible outside this module)

g: scope is global (visible outside this module)

UND: Undefined in this module

Offsets in each section

Relocation Table

- List of “items” for which this file needs the address.
- What are they?
 - Any label jumped to: `b` or `b1`
 - internal
 - external (including lib files)
 - Any piece of data addressed via literal pool
 - such as the `ldr rdest, =label` instruction

Example Relocation Table

```
RELOCATION RECORDS FOR [.text]:
OFFSET  TYPE          VALUE
00000014 R_ARM_PC24      posmul
00000024 R_ARM_ABS32     .data
00000028 R_ARM_ABS32     .data
0000002c R_ARM_ABS32     .data
00000030 R_ARM_ABS32     .data
```

posmul in `.text` segment needs relocation

The labels in `.data` segment need relocation

Recall addresses in `.data` segment start from zero:

24: 0000000c `andeq r0, r0, ip; add of MSG`

28: 00000000 `andeq r0, r0, r0; add of a`

2c: 00000004 `andeq r0, r0, r4; add of b`

30: 00000008 `andeq r0, r0, r8; add of c`

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
- **relocation information**: identifies lines of code that need to be “handled”
- **symbol table**: list of this file’s labels and data that can be referenced
- **debugging information**

ELEC2041 lec25-linker.II.21

Saeid Nooshabadi

Example: Object file Header

```
mul.o:      file format elf32-littlearm
mul.o

architecture: arm, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x00000000

private flags = 0: [APCS-32] [FPA float format]
```

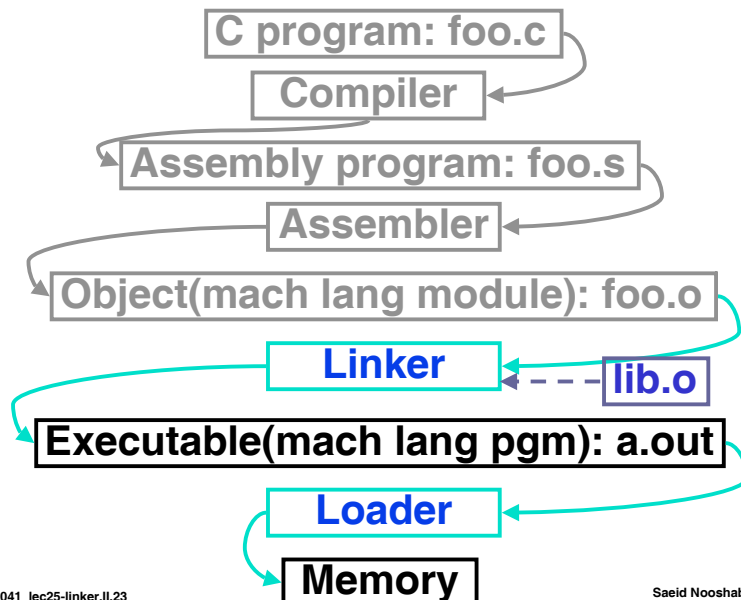
Sections:

Idx	Name	Size	VMA	LMA	File offset	Algn
0	.text	00000034	00000000	00000000	00000034	2**2
	CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE					
1	.data	0000001c	00000000	00000000	00000068	2**2
	CONTENTS, ALLOC, LOAD, DATA					

ELEC2041 lec25-linker.II.22

Saeid Nooshabadi

Where Are We Now?



ELEC2041 lec25-linker.II.23

Saeid Nooshabadi

Link Editor/Linker (#1/2)

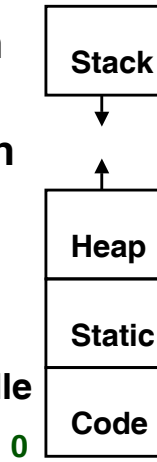
- What does it do?
- Combines several object (.o) files into a single executable (“**linking**”)
- Enable Separate Compilation of files
 - Changes to one file do not require recompilation of whole program
 - Windows XP source is >700 M lines of code! And Growing!
 - Called a **module (file)**
 - Link Editor name from editing the “links” in branch and link (b1) instructions

ELEC2041 lec25-linker.II.24

Saeid Nooshabadi

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.
- Step 3: Resolve References
 - Go through Relocation Table and handle each entry
 - That is, fill in all **absolute addresses**



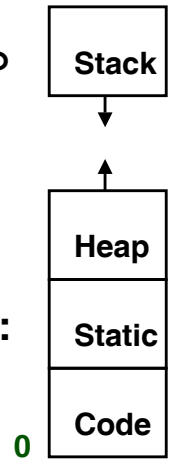
ELEC2041 lec25-linker.II.25

Saeid Nooshabadi

Example: Linker ELF output

- Combines 3 files: cstart.o mul.o and posmul.o
- Command Line:

```
arm-elf-ld -Ttext=0x0 -o mul.elf
  cstart.o mul.o posmul.o
```
- Produces the final executable code:



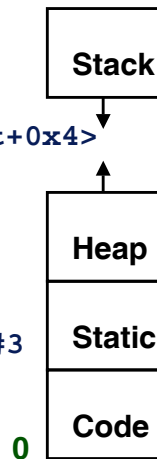
ELEC2041 lec25-linker.II.26

Saeid Nooshabadi

Example: C ⇒ Asm ⇒ _Obj ⇒ Exe ⇒ Run (#1/6)

Disassembly of section .text:

```
00000000 <_start>:
0: e59fd004 ldr sp, [pc, #4] ; c <exit+0x4>
4: eb000001 bl 10 <main>
00000008 <exit>:
8: eaaffffe b 8 <exit>
c: 000021a8 andeq r2, r0, r8, lsr #3
```



ELEC2041 lec25-linker.II.27

Saeid Nooshabadi

Example: C ⇒ Asm ⇒ _Obj ⇒ Exe ⇒ Run (#2/6)

```
00000010 <main>:
10: e59f201c ldr r2, [pc, #28] ; 34 <main+0x24>
14: e59f001c ldr r0, [pc, #28] ; 38 <main+0x28>
18: e5900000 ldr r0, [r0]
1c: e59f1018 ldr r1, [pc, #24] ; 3c <main+0x2c>
20: e5911000 ldr r1, [r1]
24: eb000006 bl 44 <posmul>
28: e59f1010 ldr r1, [pc, #16]; 40 <main+0x30>
2c: e5810000 str r0, [r1]
30: e1a0f00e mov pc, lr
```

posmul with its address resolved

ELEC2041 lec25-linker.II.28

Saeid Nooshabadi

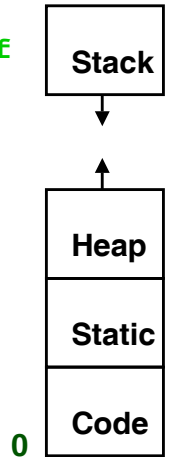
Example: C ⇒ Asm ⇒ _Obj ⇒ Exe ⇒ Run (#3/6)

```
34: 00000198 muleq r0, r8, r1 ; address of MESH
38: 0000018c andeq r0, r0, ip, lsl #3;address of a
3c: 00000190 muleq r0, r0, r1 ; address of b
40: 00000194 muleq r0, r4, r1; ; address of c
```

Literal pool with all addresses resolved

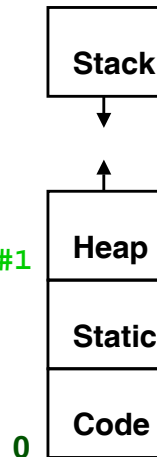
Example: C ⇒ Asm ⇒ _Obj ⇒ Exe ⇒ Run (#4/6)

```
00000044 <posmul>:
44: e35100ff  cmp     r1, #255  ; 0xff
48: da000001  ble    54 <continue1>
4c: e3e02000  mvn    r2, #0    ; 0x0
50: ea00000b  b      84 <finished>
00000054 <continue1>:
54: e3510000  cmp    r1, #0    ; 0x0
58: aa000001  bge    64 <continue2>
5c: e3e02000  mvn    r2, #0    ; 0x0
60: ea000007  b      84 <finished>
```



Example: C ⇒ Asm ⇒ _Obj ⇒ Exe ⇒ Run (#5/6)

```
00000064 <continue2>:
64: e1b010a1  movs   r1, r1, lsr #1
68: 21a02000  movcs  r2, r0
6c: 33a02000  movcc  r2, #0    ; 0x0
00000070 <shift_loop>:
70: e1b010a1  movs   r1, r1, lsr #1
74: 20822080  addcs  r2, r2, r0, lsl #1
78: 31a00080  movcc  r0, r0, lsl #1
7c: 0a000000  beq    84 <finished>
80: eaffffffa  b      70 <shift_loop>
00000084 <finished>:
84: e1a00002  mov    r0, r2
88: e1a0f00e  mov    pc, lr
```



Example: C ⇒ Asm ⇒ _Obj ⇒ Exe ⇒ Run (#6/6)

Disassembly of section .data:

```
0000018c <__data_start>:
18c: 00000014 andeq  r0, r0, r4, lsl r0
190: 00000012 andeq  r0, r0, r2, lsl r0
00000194 <c>:
194: 00000000 andeq  r0, r0, r0
00000198 <MESH>:
198: 746c554d strvcbt r5, [ip], -#1357
19c: 696c7069 stmvsdb
      ip!,{r0,r3,r5,r6,ip,sp,lr}^
1a0: 69746163 ldmvsdb
      r4!,{r0,r1,r5,r6,r8,sp,lr}^
1a4: 00006e6f andeq  r6, r0, pc, ror #28
```

Symbol Table Entries

◦ Symbol Table

- Label Address

`_start:`

`main:`

`posmul:`

`MESG:`

`exit:`



◦ Relocation Information

Symbol Table Entries

◦ Symbol Table

- Label Address

`_start:` 0x00000000

`main:` 0x00000010

`posmul:` 0x00000044

`MESG:` 0x10000198

`exit:` 0x00000008

◦ Relocation Information

- Every thing is resolved. No more Relocation Information

Five Types of Addresses

◦ PC-Relative Addressing : never relocate:

- `ldr/str rdest, label,`
- `adr rdest, label,`
- `ladr rdest, label`

◦ Branch via register (`mov pc, lr`) never relocate

◦ Absolute Address (`b, b1`): always relocate

◦ External Reference (usually `b1`): always relocate

◦ Data Reference via literal entries (often `ldr rdest =label`): always relocate

Resolving References (#1/2)

◦ Linker *assumes* first word of first text segment is at address 0x00000.

◦ 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

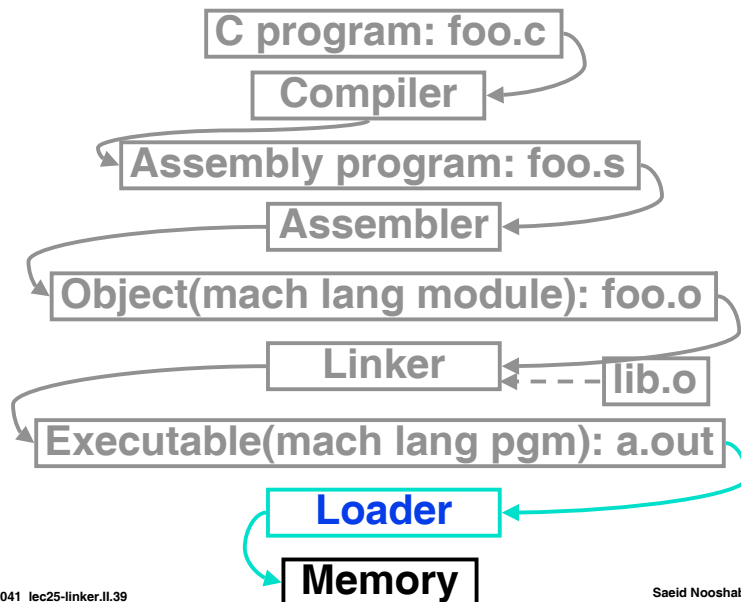
Resolving References (#2/2)

- 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)

Reading Material

- Reading assignment:
 - David Patterson and John Hennessy: "Computer Organisation & Design: The HW/SW Interface," 2nd Ed Morgan Kaufmann, 1998, ISBN: 1 - 55860 - 491 - X. (A.8, 8.1-8.4)

Where Are We Now?



Loader (#1/3)

- Executable files are stored on disk.
- When one is run, loader's job is to load it into memory and start it running.
- In reality, loader is the operating system (OS)
 - 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 (this may be anywhere in memory)

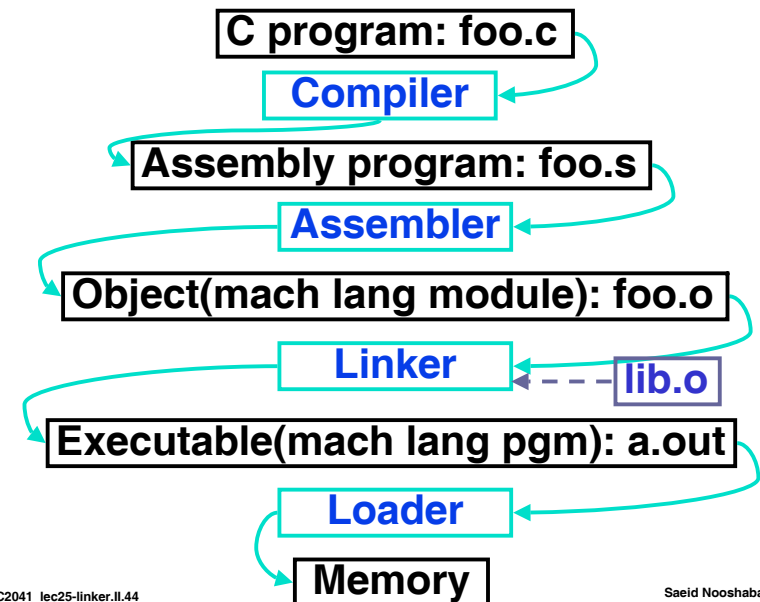
Loader (#3/3)

- 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

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

0: e59fd004	38: 0000018c	74: 20822080
4: eb000001	3c: 00000190	78: 31a00080
8: eaffffff	40: 00000194	7c: 0a000000
c: 000021a8	44: e35100ff	80: eaffffff
10: e59f201c	48: da000001	84: e1a00002
14: e59f001c	4c: e3e02000	88: e1a0f00e
18: e5900000	50: ea00000b	18c: 00000014
1c: e59f1018	54: e3510000	190: 00000012
20: e5911000	58: aa000001	194: 00000000
24: eb000006	5c: e3e02000	198: 746c554d
28: e59f1010	60: ea000007	19c: 696c7069
2c: e5810000	64: e1b010a1	1a0: 69746163
30: e1a0f00e	68: 21a02000	1a4: 00006e6f
34: 00000198	6c: 33a02000	
38: 0000019c	70: e1b010a1	

Things to Remember (#1/3)



Things to Remember (#2/3)

- **Compiler converts a single HLL file into a single assembly language file.**
- **Assembler removes pseudos, 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)

- **Stored Program concept mean 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 ⇒ Assembler ⇒ Linker (⇒ Loader)**
- **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**