



inst.eecs.berkeley.edu/~cs61c
UCB CS61C : Machine Structures

Lecture 19 – Running a Program II (Compiling, Assembling, Linking, Loading)

Lecturer SOE
Dan Garcia

2008-03-06

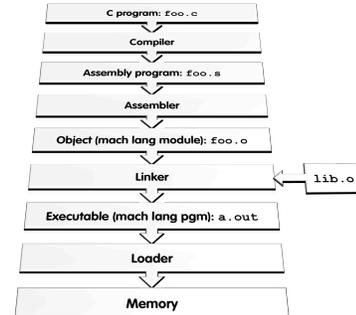
CS DEGREES BOTTOMING OUT?

I'm sure you've known that the number of people graduating with CS degrees has been dipping since its peak in 2000 due to (false?) perceptions of outsourcing, that CS is "just programming", and other reasons. We may be on our way to turning that around; for the first time in 8 years the numbers of new CS majors has actually increased! It's certainly a start!

developers.slashdot.org/developers/08/03/05/2113234.shtml



Where Are We Now?



CS61C L19: Running a Program II (8)

Garcia, Spring 2008 © UCB

Linker (1/3)

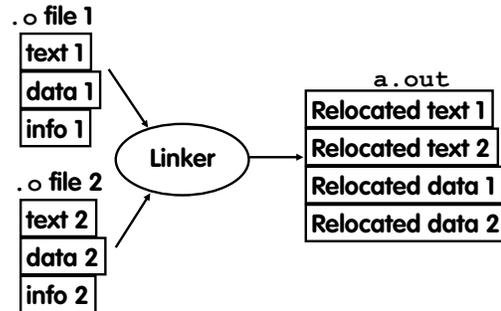
- **Input: Object Code files, information tables (e.g., foo.o, libc.o for MIPS)**
- **Output: Executable Code (e.g., a.out for MIPS)**
- **Combines several object (.o) files into a single executable ("linking")**
- **Enable Separate Compilation of files**
 - Changes to one file do not require recompilation of whole program
 - Windows NT source was > 40 M lines of code!
 - Old name "Link Editor" from editing the "links" in jump and link instructions



CS61C L19: Running a Program II (7)

Garcia, Spring 2008 © UCB

Linker (2/3)



CS61C L19: Running a Program II (8)

Garcia, Spring 2008 © UCB

Linker (3/3)

- **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; handle each entry
 - That is, fill in all absolute addresses



CS61C L19: Running a Program II (9)

Garcia, Spring 2008 © UCB

Four Types of Addresses we'll discuss

- **PC-Relative Addressing (beq, bne)**
 - never relocate
- **Absolute Address (j, jal)**
 - always relocate
- **External Reference (usually jal)**
 - always relocate
- **Data Reference (often lui and ori)**
 - always relocate



CS61C L19: Running a Program II (10)

Garcia, Spring 2008 © UCB

Absolute Addresses in MIPS

- Which instructions need relocation editing?

- J-format: **jump, jump and link**

j/jal	xxxxxx
-------	--------

- Loads and stores to variables in static area, relative to global pointer

lw/sw	\$gp	\$x	address
-------	------	-----	---------

- What about conditional branches?

beq/bne	\$rs	\$rt	address
---------	------	------	---------

- PC-relative addressing preserved even if code moves



Resolving References (1/2)

- Linker assumes first word of first text segment is at address 0x00000000.

- (More later when we study “virtual memory”)

- 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 “user” 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)



Static vs Dynamically linked libraries

- What we’ve described is the traditional way: statically-linked approach

- The library is now part of the executable, so if the library updates, we don’t get the fix (have to recompile if we have source)
- It includes the entire library even if not all of it will be used.
- Executable is self-contained.

- An alternative is dynamically linked libraries (DLL), common on Windows & UNIX platforms



en.wikipedia.org/wiki/Dynamic_linking Dynamically linked libraries

- Space/time issues

- + Storing a program requires less disk space
- + Sending a program requires less time
- + Executing two programs requires less memory (if they share a library)
- – At runtime, there’s time overhead to do link

- Upgrades

- + Replacing one file (`libXYZ.so`) upgrades every program that uses library “XYZ”
- – Having the executable isn’t enough anymore

Overall, dynamic linking adds quite a bit of complexity to the compiler, linker, and operating system. However, it provides many benefits that often outweigh these.



Dynamically linked libraries

- The prevailing approach to dynamic linking uses machine code as the “lowest common denominator”

- The linker does not use information about how the program or library was compiled (i.e., what compiler or language)
- This can be described as “linking at the machine code level”
- This isn’t the only way to do it...



Administrivia...Midterm on Sunday!

- Review Sat @ 2-5pm in 2050 VLSB
- Exam Sun @ 6-9pm in 2050 VLSB
- Covers labs,hw,proj,lec,book through today
- Bring...
 - NO backpacks, cells, calculators, pagers, PDAs
 - 2 writing implements (we'll provide write-in exam booklets) – pencils ok!
 - One handwritten (both sides) 8.5"x11" paper
 - One green sheet (make sure to correct green sheet bugs)



CS6C L1P Running a Program I (7)

©ards, Spring 2008 © UCS

Upcoming Calendar

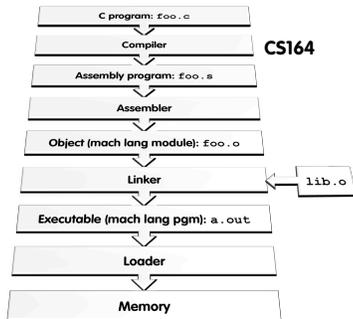
Week #	Mon	Wed	Thu Lab	Fri
#7 This week	MIPS Inst Format III	Running Program I	Running Program	Running Program II
#8 Midterm week Midterm Sunday 6-9pm 2050 VLSB	Go over and hand back Midterms	SDS I (Scott Beamer)	SDS	SDS II (Scott Beamer)



CS6C L1P Running a Program I (8)

©ards, Spring 2008 © UCS

Where Are We Now?



CS6C L1P Running a Program I (9)

©ards, Spring 2008 © UCS

Loader (1/3)

- Input: Executable Code (e.g., a.out for MIPS)
- Output: (program is run)
- 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



CS6C L1P Running a Program I (10)

©ards, Spring 2008 © UCS

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



CS6C L1P Running a Program I (11)

©ards, Spring 2008 © UCS

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



CS6C L1P Running a Program I (12)

©ards, Spring 2008 © UCS

Peer Instruction

Which of the following instr. may need to be edited during link phase?

```

Loop: lui $at, 0xABCD
      ori $a0,$at, 0xFEDC }# A
      jal add_link        # B
      bne $a0,$v0, Loop  # C
    
```

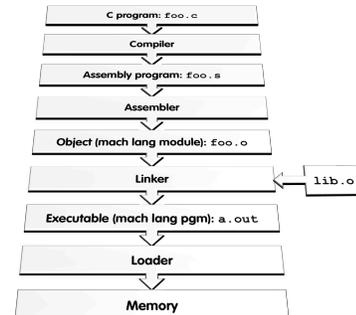
ABC
0: FFF
1: FFT
2: FTF
3: FTT
4: TFF
5: TFT
6: TTF
7: TTT

Cal

CMSC 137 Reading a Program I (2)

Geards, Spring 2008 © UCS

Things to Remember (1/3)



Cal

CMSC 137 Reading a Program I (2)

Geards, Spring 2008 © UCS

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). A .s file becomes a .o file.
 - Does 2 passes to resolve addresses, handling internal forward references
- Linker combines several .o files and resolves absolute addresses.
 - Enables separate compilation, libraries that need not be compiled, and resolves remaining addresses
- Loader loads executable into memory and begins execution.

Cal

CMSC 137 Reading a Program I (2)

Geards, Spring 2008 © UCS

Things to Remember 3/3

- Stored Program concept is very powerful. It means that instructions sometimes act just like data. Therefore we can use programs to manipulate other programs!
 - Compiler ⇒ Assembler ⇒ Linker (⇒ Loader)

Cal

CMSC 137 Reading a Program I (2)

Geards, Spring 2008 © UCS

Bonus slides

- These are extra slides that used to be included in lecture notes, but have been moved to this, the "bonus" area to serve as a supplement.
- The slides will appear in the order they would have in the normal presentation

Bonus

Cal

CMSC 137 Reading a Program I (2)

Geards, Spring 2008 © UCS

Big Endian vs. Little Endian

Big-endian and little-endian derive from Jonathan Swift's *Gulliver's Travels* in which the Big Endians were a political faction that broke their eggs at the large end ("the primitive way") and rebelled against the Lilliputian King who required his subjects (the Little Endians) to break their eggs at the small end.

- The order in which BYTES are stored in memory
- Bits always stored as usual. (E.g., 0xC2=0b 1100 0010)

Consider the number 1025 as we normally write it:

BYTE3 BYTE2 BYTE1 BYTE0
00000000 00000000 00000100 00000001

Big Endian

Little Endian

• ADDR3 ADDR2 ADDR1 ADDR0	• ADDR3 ADDR2 ADDR1 ADDR0
BYTE0 BYTE1 BYTE2 BYTE3	BYTE3 BYTE2 BYTE1 BYTE0
00000001 00000100 00000000 00000000	00000000 00000000 00000100 00000001
• ADDR0 ADDR1 ADDR2 ADDR3	• ADDR0 ADDR1 ADDR2 ADDR3
BYTE3 BYTE2 BYTE1 BYTE0	BYTE0 BYTE1 BYTE2 BYTE3
00000000 00000000 00000100 00000001	00000001 00000100 00000000 00000000

www.webopedia.com/TERM/b/big_endian.html
searchnetworking.techtarget.com/sDefinition/0,,sid7_gci211659_00.html
www.noveltheory.com/TechPapers/Endian.asp
en.wikipedia.org/wiki/Big_endian

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

C Program Source Code: *prog.c*

```
#include <stdio.h>
int main (int argc, char *argv[]) {
    int i, sum = 0;
    for (i = 0; i <= 100; i++)
        sum = sum + i * i;
    printf ("The sum of sq from 0 .. 100 is
    %d\n",      sum);
}
"printf" lives in "libc"
```



Compilation: MAL

```
__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
    jr $ra
.data
    Where are
    .align 0 7 pseudo-
str:
    instructions?
.asciiz "The sum
of sq from 0 ..
100 is %d\n"
```

Compilation: MAL

```
__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
    jr $ra
    .data
    7 pseudo-
    .align 0 instructions
str:
    underlined
    .asciiz "The sum
of sq from 0 ..
100 is %d\n"
```

Assembly step 1:

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, 24($29)
14 sw $0, 28($29)
18 lw $14, 28($29)
1c multu $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
40 lui $4, l.str
44 ori $4,$4,r.str
48 lw $5,24($29)
4c jal printf
50 add $2, $0, $0
54 lw $31,20($29)
58 addiu $29,$29,32
5c jr $31
```

Assembly step 2

Create relocation table and symbol table

Symbol Table

Label	address (in module)	type
main:	0x00000000	global text
loop:	0x00000018	local text
str:	0x00000000	local data

Relocation Information

Address	Instr. type	Dependency
0x00000040	lui	l.str
0x00000044	ori	r.str
0x0000004c	jal	printf



Assembly step 3

Resolve local PC-relative labels

```
00 addiu $29,$29,-32
04 sw $31,20($29)
08 sw $4, 32($29)
0c sw $5, 36($29)
10 sw $0, 24($29)
14 sw $0, 28($29)
18 lw $14, 28($29)
1c multu $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, -10
40 lui $4, l.str
44 ori $4,$4,r.str
48 lw $5,24($29)
4c jal printf
50 add $2, $0, $0
54 lw $31,20($29)
58 addiu $29,$29,32
5c jr $31
```

Assembly step 4

- **Generate object (.o) file:**
 - **Output binary representation for**
 - **ext segment (instructions),**
 - **data segment (data),**
 - **symbol and relocation tables.**
 - **Using dummy “placeholders” for unresolved absolute and external references.**

Cal

CMSC 137 Reading a Program I (8)

Gerds, Spring 2008 © UCS

Text segment in object file

```

0x000000  00100111101111011111111111111100000
0x000004  101011111011111110000000000010100
0x000008  101011111010010000000000000100000
0x00000c  101011111010010100000000000100100
0x000010  101011111010000000000000000011000
0x000014  101011111010000000000000000011100
0x000018  100011111010111000000000000011100
0x00001c  100011111011100000000000000011000
0x000020  0000001110011100000000000011001
0x000024  00100101110010000000000000000001
0x000028  0010100100000001000000000100101
0x00002c  1010111110101000000000000011100
0x000030  000000000000000000111100000010010
0x000034  0000001000011111100100000100001
0x000038  000101000010000011111111110111
0x00003c  1010111110110010000000000011000
0x000040  0011100000001000000000000000000
0x000044  1000111110100101000000000000000
0x000048  0000110000010000000000001101100
0x00004c  0010010000000000000000000000000
0x000050  1000111110111110000000000010100
0x000054  0010011110111101000000000100000
0x000058  000000111110000000000000001000
0x00005c  0000000000000000000000100000100001
  
```

Cal

CMSC 137 Reading a Program I (9)

Gerds, Spring 2008 © UCS

Link step 1: combine prog.o, libc.o

- **Merge text/data segments**
- **Create absolute memory addresses**
- **Modify & merge symbol and relocation tables**
- **Symbol Table**
 - **Label** **Address**
 - main: 0x00000000
 - loop: 0x00000018
 - str: 0x10000430
 - printf: 0x000003b0 ...
- **Relocation Information**
 - **Address** **Instr. Type** **Dependency**
 - 0x00000040 lui l.str
 - 0x00000044 ori r.str
 - 0x0000004c jal printf ...

Cal

CMSC 137 Reading a Program I (10)

Gerds, Spring 2008 © UCS

Link step 2:

- **Edit Addresses in relocation table**
 - *(shown in TAL for clarity, but done in binary)*

```

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, -10
10 sw      $0, 24($29)   | 40 lui   $4, 4096
14 sw      $0, 28($29)   | 44 ori   $4,$4,1072
18 lw      $14, 28($29)  | 48 lw    $5,24($29)
1c multu $14, $14        | 4c jal   812
20 mflo $15              | 50 add   $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
  
```

Link step 3:

- **Output executable of merged modules.**
 - **Single text (instruction) segment**
 - **Single data segment**
 - **Header detailing size of each segment**
- **NOTE:**
 - **The preceding example was a much simplified version of how ELF and other standard formats work, meant only to demonstrate the basic principles.**

Cal

CMSC 137 Reading a Program I (11)

Gerds, Spring 2008 © UCS