CS61C
Starting a Program

Lecture 10

February 19, 1999

Dave Patterson
(http.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs61c/schedule.html

Review: New MIPS arithmetic instructions

Example	Meaning	Comments
mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
multu$2,$3	Hi, Lo = $2 x $3	64-bit unsigned product
div $2,$3	Lo = $2 ÷ $3,	Lo = quotient, Hi = rem
divu $2,$3	Lo = $2 ÷ $3,	Unsigned quotient, rem.
mfhi $1	$1 = Hi	Used to get copy of Hi
mflo $1	$1 = Lo	Used to get copy of Lo
add.s $0,$1,$2	$f0=$f1+$f2	Fl. Pt. Add (single)
add.d $0,$2,$4	$f0=$f2+$f4	Fl. Pt. Add (double)
sub.s $0,$1,$2	$f0=$f1-$f2	Fl. Pt. Subtract (single)
sub.d $0,$2,$4	$f0=$f2-$f4	Fl. Pt. Subtract (double)
mul.s $0,$1,$2	$f0=$f1x$f2	Fl. Pt. Multiply (single)
mul.d $0,$2,$4	$f0=$f2x$f4	Fl. Pt. Multiply (double)
div.s $0,$1,$2	$f0=$f1÷$f2	Fl. Pt. Divide (single)
div.d $0,$2,$4	$f0=$f2÷$f4	Fl. Pt. Divide (double)
c.X.s $0,$1	flag1= $f0 X $f1	Fl. Pt.Compare (single)
c.X.d $0,$2	flag1= $f0 X $f2	Fl. Pt.Compare (double)

X is eq, lt, le; bc1t, bc1f tests flag

Review 2/2

- IEEE 754 Floating Point standard: accuracy first class citizen
- Multiply product 2n bits; Divide produces both n-bit quotient and n-bit remainder
- Computer numbers have limited size => limited precision
  - underflow: too small for Fl. Pt. (bigger negative exponent than can represent)
  - overflow: too big for Fl. Pt. or integer (bigger positive exponent than can represent, or bigger integer than fits in word)
  - Programmers beware!

Outline

- Floating Point Questions and Answers
- Compiling, Assembling a Program
- Administrivia, “Computers in the News”
- Linking, Loading a Program
- An Example
- Conclusion

Floating Point Questions: Negative 0?

- What is result of this program?

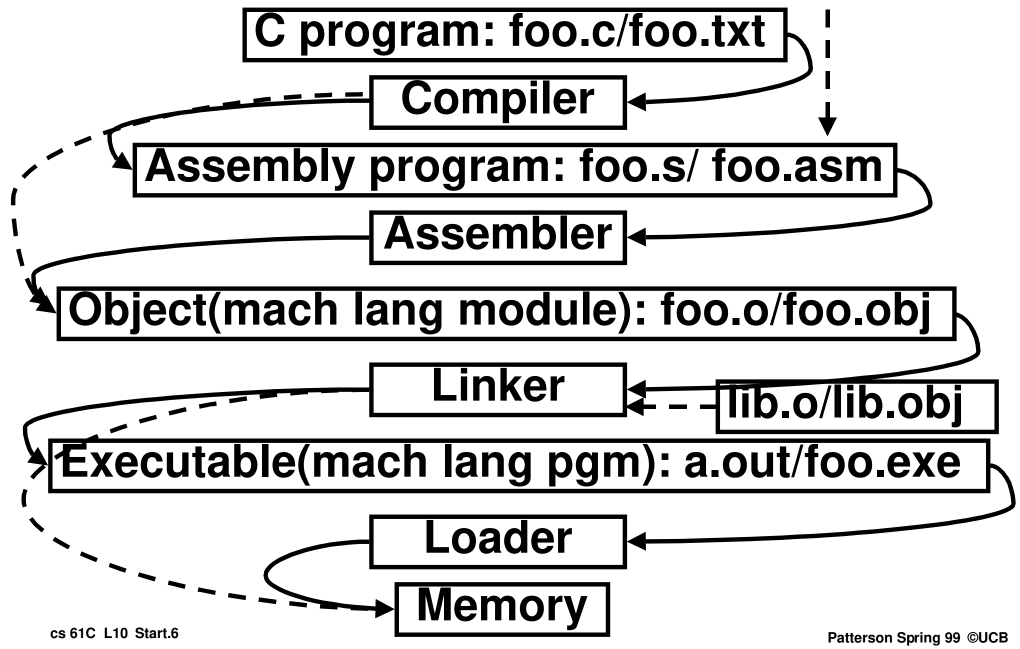
```
main() {
 float i,j;
 i = 1.0; j = 0.0;
 printf("%3.1f\n",i/j);
 printf("%3.1f\n",-i/j);
 printf("%3.1f\n",j/(-i/j));
 if (j/(-i/j) == 0.0)
     printf("0.0 == -0.0\n");
 else printf("0.0 != -0.0\n");
 printf("%3.1f\n",i/j-i/j);
 printf("%3.1f\n",(i/j-i/j)+i);

}
```

- Output:

```
Inf
-Inf
-0.0
0.0 == -0.0
NaN
NaN
```


Big Idea: Levels of Abstaction (Unix/DOS)



Assembler

- Follow Directions
- Replace Pseudoinstructions
- Find list of labels, addresses
- Produce machine language
- Creates Object File

Example Assembly Program: Directives

```
 .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 sum from 0 .. 100 is %d\n"
```

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

- Tell assembler how to translate program but do not produce machine instructions
  - `.text (addr)`: Subsequent items put in user text segment (starting at `addr`)
  - `.align n`: Align the next data on a 2^n byte boundary; `align 2` ⇒ next word boundary
  - `.globl sym`: declares `sym` global and can be referenced from other files
  - `.data (addr)`: Subsequent items put in user data segment (starting at `addr`)
  - `.asciiz str`: Store the string `str` in memory and null-terminate it

Example Assembly Program: PseudoInstr

```
 .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 sum
from 0 .. 100 is
%d\n"
```


Pseudoinstruction replacements

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

```
•subu $sp,$sp,32     addiu $sp,$sp,-32
•sd $a0, 32($sp)     sw $a0, 32($sp)
                     sw $a1, 36($sp)
•mul $t7,$t6,$t6     mul $t6,$t6
                     mflo $t7
•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)
•move $v0, $0        add $v0,$0,$0
```


Example Assembly Program: Labels

```
 .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 sum
from 0 .. 100 is
%d\n"
```

Addresses

- Main job of Assembler:

1) Find list of labels and their addresses

2) Produce machine language

- Puts label and memory address of that instruction in <u>Symbol Table</u>
- Can use a name before it is defined; <u>forward reference</u> e.g., `str` in example
  - Note: Must declare before use in C
- Uses Symbol Table in 2nd pass to produce machine code, including addresses

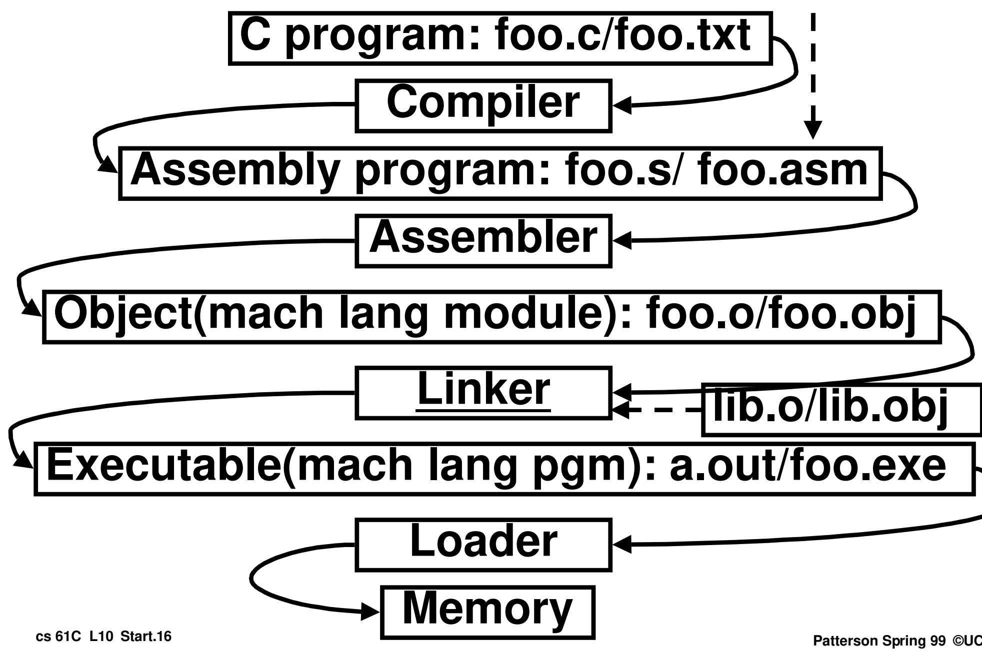
Administrivia

- Readings: (3.9) A.2, A.3, A.4, 3.10, 3.11
- 5th homework: Due 2/24 7PM
  - Exercises 4.21, 4.25, 4.28
- 3rd Project/5th Lab: MIPS Simulator Due Wed. 3/3 7PM; deadline Thurs 8AM
  - Twice / semester 24-hour extension
  - Everything (but midterm) may be done in pairs now
- Solutions : ~cs61c/solutions directory. Access it using cs61c account. <u>Not</u> www!
- Midterm conflict time: Mon 3/15 6-9PM

"Computers in the News"

- "Microsoft Denies Rumors That It Is Crafting a Language Like Java", NY Times, 2/15/99
  - "Microsoft Corp. is playing down a report in 'PC Week' that it has been briefing software developers on a new programming language (code-named 'Cool') that would be similar to Sun's Java software." [Java SW not binaries]
  - ".... Cross platform means that a developer can write one version of a program in Java, which can then run on Windows, Macintosh, Unix and other software operating systems."
  - "... A compiler is a programming tool that converts high-level languages, like C++ or Java, into the machine code of ones and zeros that computers understand."

Big Idea: Levels of Abstaction (Unix/DOS)



Object File Format (output of Assembler)

- 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 instructions and data words that depend on absolute addresses
- symbol table: remaining labels not defined, such as external references
- debugging information: help debugger map source file to instructions, data

Link Editor or Linker

- Enable Separate Compilation of files
  - Changes to one file do not require recompilation of whole program
    - Windows NT source is >30 M lines of code! And Growing!
  - Called a module
  - Link Editor name from editing the "links" in jump and link instructions
- Job of Linker: resolve external labels in object code and place modules and data into memory

Steps of Linker

1) Resolve references across files

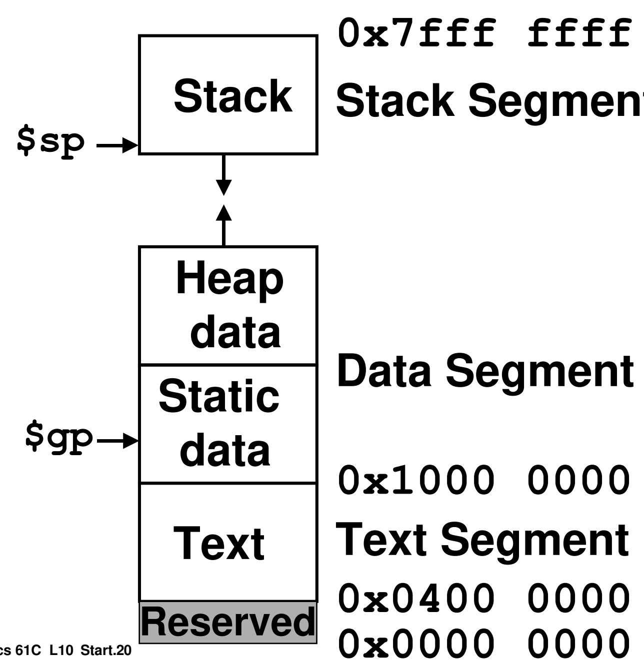
2) If some labels unresolved, search for library routine references (e.g., printf in example)

- If finds reference, it extracts routine's code and inserts into program text

3) Determine memory address of code modules and adjust absolute addresses (using relocation information)

- Produces file in same format, but no relocation information or unresolved references

C memory allocation seen by the Program



Absolute Addresses in MIPS

- Which instructions need relocation editing?
- J-format: jump, jump and link

`j/jal`	`xxxxx`

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

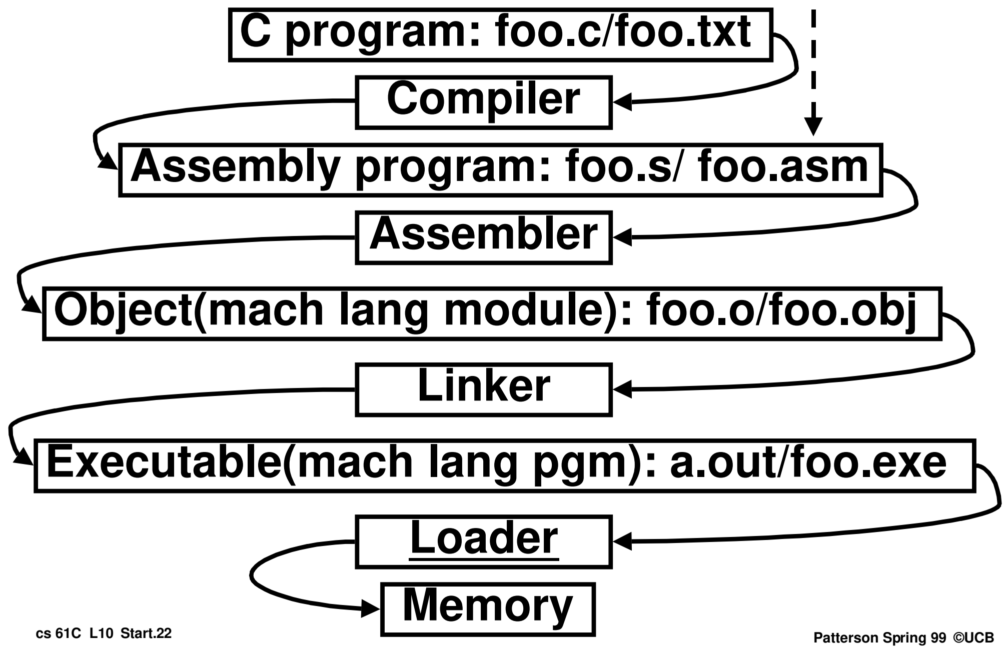
`lw/sw`	`$gp`	`$x`	`address`

- What about conditional branches?

`beq/bne`	`$rs`	`$rt`	`address`

- PC-relative addressing preserved even if code moves

Big Idea: Levels of Abstaction (Unix/DOS)



Loading

- If links without error can be run
- Before run, executable program stored on disk
- UNIX systems, Operating System Kernel loads the program from disks

Steps 1-5 (out of 6) for loading program

- 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
- 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

Steps 6 (out of 6) for loading program

- Jumps to start-up routine that copies program's arguments from stack to registers and calls program's main routine
  - If main routine returns, start-up routine terminates program with the exit system call

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

```
#include <stdio.h>

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

 int i;

 int sum = 0;


 for (i = 0; i <= 100; i = i + 1)
     sum = sum + i * i;

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

}
```


Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ 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 sum from 0 .. 100 is %d\n"
```


Symbol Table Entries

- Label   Address
- main:
- loop:
- str:

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ 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, l.str
14 sw     $0, 28($29)     44 addiu $4,$4,r.str
18 lw    $14, 28($29)     48 lw    $5,24($29)
1c multu $14, $14         4c jal   printf
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
```


Symbol Table Entries

- Symbol Table
  - Label — Address
  - `main:` — `0x00000000`
  - `loop:` — `0x00000018`
  - `str:` — `0x10000430`
  - `printf:` — `0x000003b0`
- Relocation Information
  - Address — Instr. Type — Dependency
  - `0x0000004c` — `jal` — `printf`

Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

- Edit Addresses: start at 0x0040000

```
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 addiu $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
```


Example: C ⇒ Asm ⇒ Obj ⇒ Exe ⇒ Run

```
0x004000  00100111101111011111111111100000
0x004004  10101111101111110000000000010100
0x004008  10101111101001000000000000100000
0x00400c  10101111101001010000000000100100
0x004010  10101111101000000000000000011000
0x004014  10101111101000000000000000011100
0x004018  10001111101011100000000000011100
0x00401c  10001111101110000000000000011000
0x004020  00000001110011100000000000011001
0x004024  00100101110010000000000000000001
0x004028  00101001000000010000000001100101
0x00402c  10101111101010000000000000011100
0x004030  00000000000000000111100000010010
0x004034  00000011000011111100100000100001
0x004038  00010100001000001111111111110111
0x00403c  10101111101110010000000000011000
0x004040  00111100000001000001000000000000
0x004044  10001111101001010000000000011000
0x004048  00001100000100000000000011101100
0x00404c  00100100100001000000010000110000
0x004050  10001111101111110000000000010100
0x004054  00100111101111010000000000100000
0x004058  00000011111000000000000000001000
0x00405c  00000000000000000001000000100001
```

“And in Conclusion..” 1/1

- 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
- Next: Reviewing C, Pointers, Procedures