

CSE 141 – Computer Architecture Spring 2005

Lectures 10 Mid-term Test Review

Pramod V. Argade
April 27, 2005

CSE141: Introduction to Computer Architecture

Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
Office Hour:
Mon. 5:00 - 6:00 PM (AP&M 5218)

TAs:

Baris Arslan: barslan@cs.ucsd.edu
Chris Roedel: croedel@cs.ucsd.edu
Raid Ayoub: rayoub@cs.ucsd.edu
Leo Porter: leporter@cs.ucsd.edu

Textbook: Computer Organization & Design
The Hardware Software Interface, 3rd Edition.
Authors: Patterson and Hennessy

Web-page: <http://www.cse.ucsd.edu/classes/sp05/cse141>

Announcements

- **Homework: None!**
- **Extra Office Hour Monday**
- **Midterm**

When: Mon., May 2nd

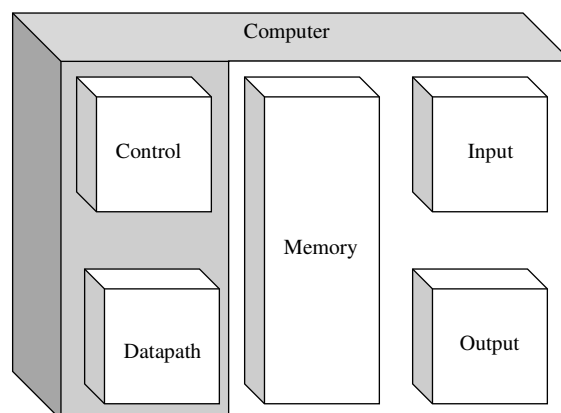
Where: **Center Room 101**

Need: Paper, pen, calculator (only for calculations, not for notes!)

Topics:

- Chapters 1 - 5
- All the material covered in lectures
- All topics covered in homework's

The five classic components of computers



Performance

- Performance

- Execution Time = (Instruction Count) * CPI * (Cycle Time)
- Clock rate is in cycles per second
 - > MHz (Millions of cycles per second, 10^6 Hz)
 - > GHz (Billions of cycles per second, 10^9 Hz)
- Cycle time = $1/(\text{Clock Rate})$
- Speedup = (exe time without change / exe time with change)

$$\text{Relative Performance} = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n$$

- Amdahl's Law

$$\text{Execution time after improvement} = \frac{\text{Execution Time Affected}}{\text{Amount of Improvement}} + \text{Execution Time Unaffected}$$

ISA

- Instruction Length

- Variable
- Fixed
- Hybrid
- MIPS has fixed 32 bit length

- Basic ISA Types

- Load-store
- Reg-mem
- Stack
- Accumulator

Overview of MIPS ISA

- 3-operand, load-store architecture
- 32 general-purpose registers (integer, floating point)
 - R0 always equals 0
- 2 special-purpose integer registers, HI and LO, because multiply and divide produce more than 32 bits
- Fixed 32-bit instructions
- 3 instruction formats
- Registers are 32-bits wide (word)
- Register, immediate, base+displacement, PC-relative and pseudo-direct addressing modes

MIPS Addressing Modes

1. Immediate addressing e.g. `addi $t0, $t1, 4`



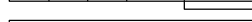
2. Register addressing e.g. `sub $t0, $t1, $t2`



Registers

Register

3. Base addressing e.g. `lw $t0, 4($t2)`



Memory

Byte | Halfword | Word

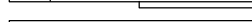
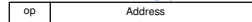
4. PC-relative addressing e.g. `beq $t1, $t2, 32`



Memory

Word

5. Pseudodirect addressing e.g. `j 0x1000`

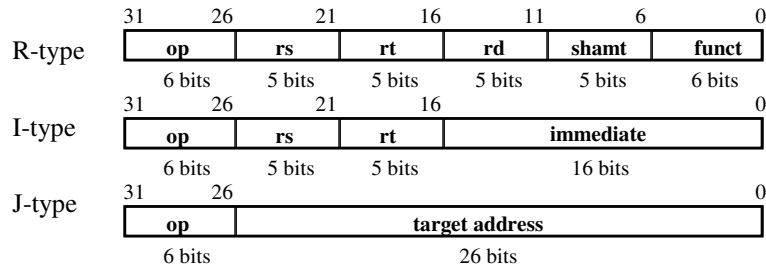


Memory

Word

The MIPS Instruction Formats

- All MIPS instructions are 32 bits long.



To summarize:

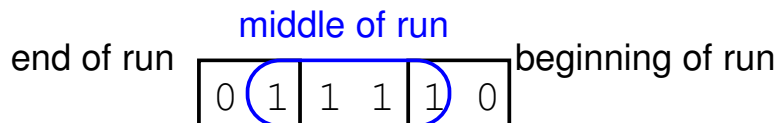
MIPS operands		
Name	Example	Comments
32 registers	\$s0-\$s7, \$t0-\$t9, \$zero, \$a0-\$a3, \$v0-\$v1, \$gp, \$fp, \$sp, \$ra, \$at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register \$zero always equals 0. Register \$at is reserved for the assembler to handle large constants.
2 ³⁰ memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language				
Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add \$s1, \$s2, \$s3	\$s1 = \$s2 + \$s3	Three operands; data in registers
	subtract	sub \$s1, \$s2, \$s3	\$s1 = \$s2 - \$s3	Three operands; data in registers
	add immediate	addi \$s1, \$s2, 100	\$s1 = \$s2 + 100	Used to add constants
Data transfer	load word	lw \$s1, 100(\$s2)	\$s1 = Memory[\$s2 + 100]	Word from memory to register
	store word	sw \$s1, 100(\$s2)	Memory[\$s2 + 100] = \$s1	Word from register to memory
	load byte	lb \$s1, 100(\$s2)	\$s1 = Memory[\$s2 + 100]	Byte from memory to register
	store byte	sb \$s1, 100(\$s2)	Memory[\$s2 + 100] = \$s1	Byte from register to memory
	load upper immediate	lui \$s1, 100	\$s1 = 100 * 2 ¹⁶	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq \$s1, \$s2, 25	if (\$s1 == \$s2) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne \$s1, \$s2, 25	if (\$s1 != \$s2) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt \$s1, \$s2, \$s3	if (\$s2 < \$s3) \$s1 = 1; else \$s1 = 0	Compare less than; for beq, bne
	set less than immediate	slli \$s1, \$s2, 100	if (\$s2 < 100) \$s1 = 1; else \$s1 = 0	Compare less than constant
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr \$ra	go to \$ra	For switch, procedure return
	jump and link	jal 2500	\$ra = PC + 4; go to 10000	For procedure call

Arithmetic

- Decimal, binary and hex representation
- Two's Complement
 - 2's complement representation of negative numbers
 - > Take the bitwise inverse and add 1
- Ripple carry adder
 - Worst case delay for a N-bit adder: 2N-gate delay
- Carry Lookahead adder
 - Generate Carry at Bit i $g_i = A_i \& B_i$
 - Propagate Carry via Bit i $p_i = A_i \text{ or } B_i$
 - 2 gate delay to calculate the carry in bits
 - > $C_{in1} = g_0 \mid (p_0 \& C_{in0})$
 - > $C_{in2} = g_1 \mid (p_1 \& g_0) \mid (p_1 \& p_0 \& C_{in0})$
 - > $C_{in3} = g_2 \mid (p_2 \& g_1) \mid (p_2 \& p_1 \& g_0) \mid (p_2 \& p_1 \& p_0 \& C_{in0})$
 - Worst case 5 gate delays
- Overflow flag = $C_O^{MSB} \wedge C_I^{MSB}$

Booth's algorithm: Signed multiplication

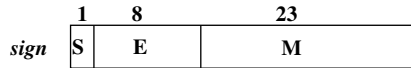


Current Bit	Bit to the Right	Explanation	Example	Op
1	0	Begins run of 1s	000111 <u>1</u> 000	sub
1	1	Middle of run of 1s	00011 <u>1</u> 000	none
0	1	End of run of 1s	000 <u>1</u> 111000	add
0	0	Middle of run of 0s	000 <u>1</u> 111000	none

Originally for Speed (when shift was faster than add)

- Replace a string of 1s in multiplier with an initial subtract when we first see a one and then later add for the bit after the last one
- Potential speed up recognizing that string of 0's and 1's requires no operation!

IEEE Floating Point Standard

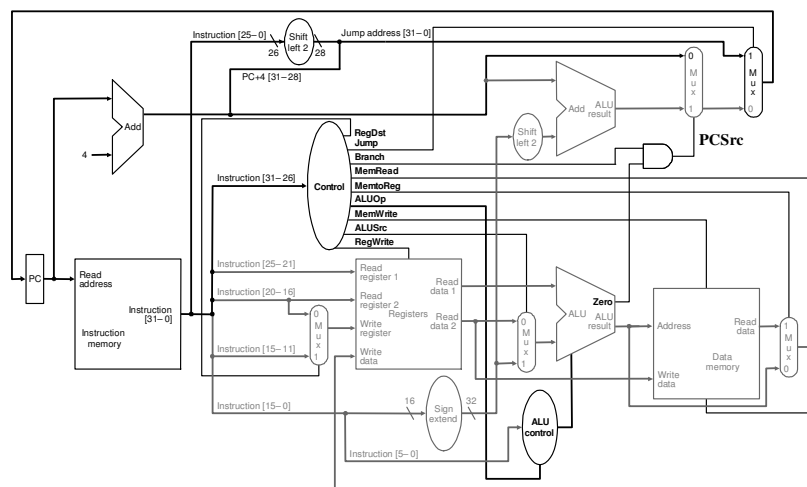


exponent:
excess 127
binary integer

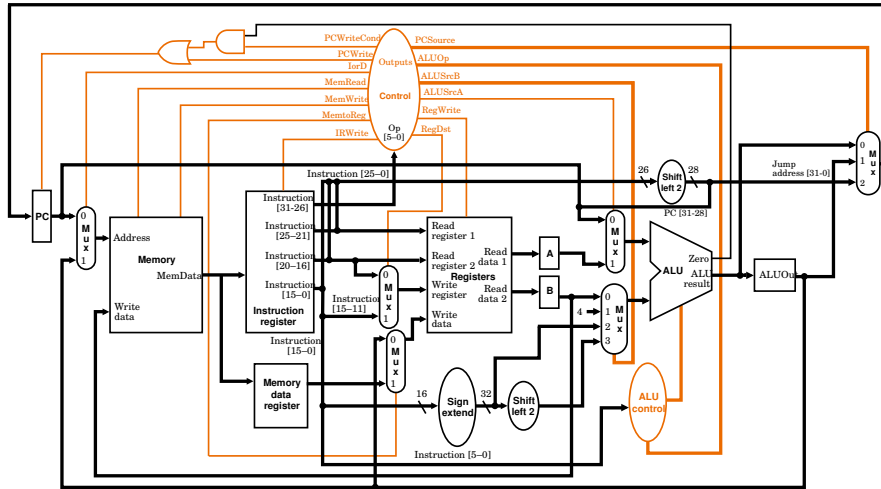
mantissa:
sign + magnitude, normalized
binary significand w/ hidden
integer bit: 1.M

- **Example:**
 - Convert decimal: - 0.75 to IEEE Single Precision Floating Point

Single Cycle CPU



Multi-cycle CPU

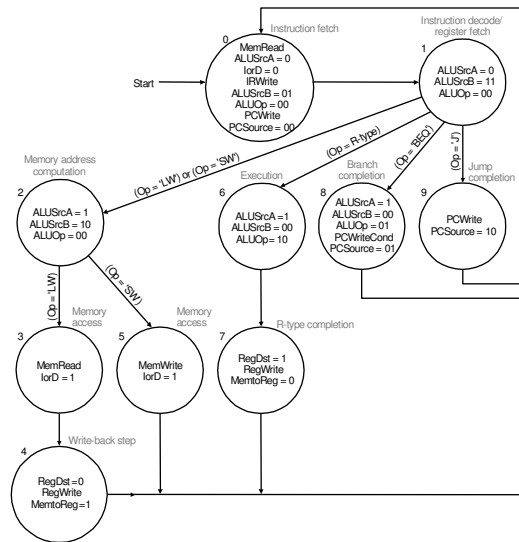


Pramod Argade

CSE 141, Spring 2005

10-15

Complete FSM



Pramod Argade

CSE 141, Spring 2005

10-16

Problem 2: Multi-cycle CPU Implementation of MemIndAdd Instruction

Modify multi-cycle datapath figure posted on the web so that it can also execute the instruction **MemIndAdd r1, offset(r2)**

This instruction calculates the address $M[\text{offset} + r2]$, which is a pointer back into memory. It then loads the value stored at address $M[\text{offset} + r2]$, and adds it to $r1$, storing the result back into $r1$.

The instruction uses the Immediate MIPS instruction format. This instruction has the same effect as sequentially executing the following code:

tmp = Memory[offset + r2]

tmp = Memory[tmp]

r1 = r1 + tmp

For this question, do not add any new ALUs, do not modify the instruction and data memory, and do not modify the register file. In addition, do not add any registers to the register file or temporary registers to the data path. You can only modify/add data paths, control lines, and MUXs. To answer this question give

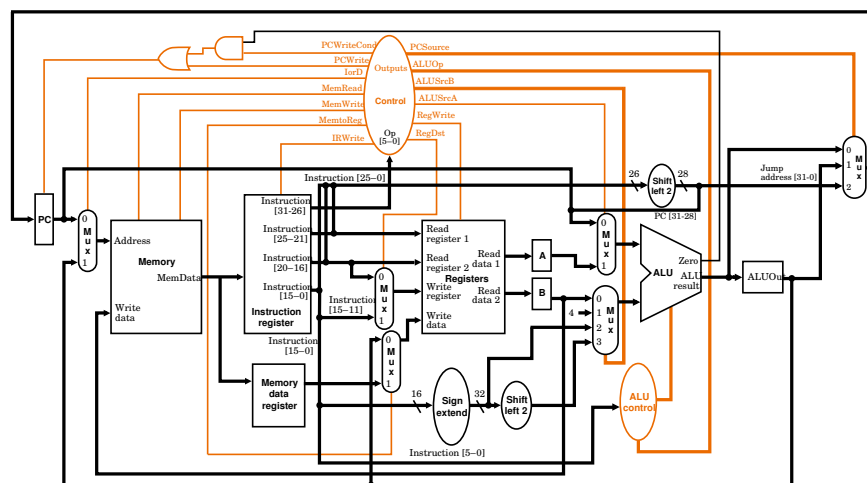
- How would you encode the instruction in I-format?
- Modify the multiple cycle processor to execute the new instruction, and
- Provide the finite state machine (FSM) for the control for this datapath. For the finite state machine show the values for the control lines and all MUXs in the figure. For the ALU control line, just give the ALU operation for each cycle. Show the important control lines and MUX values for every cycle (starting with the fetch cycle). In addition, show how many cycles it takes to execute the MemIndAdd instruction with your FSM.

Pramod Argade

CSE 141, Spring 2005

10-19

Multi-cycle CPU



Pramod Argade

CSE 141, Spring 2005

10-20

Problem 3: MIPS Assembly Code

- Comment the following MIPS assembly code and describe what is going on in the code and then convert it to the matching C code. \$s0 contains a pointer to a data memory location. Describe what is stored in memory. (A picture might help)

```
sw $zero, 0($s0)
addi $t0, $zero, 1
addi $s0, $s0, 4
sw $t0, 0($s0)
loop: addi $t0, $t0, 1
      sli $t1, $t0, 10
      be $t1, $zero, end
      addi $s0, $s0, 4
      lw $t2, -4($s0)
      lw $t3, -8($s0)
      add $t4, $t2, $t3
      sw $t4, 0($s0)
      j loop
end:   ...
```

Problem 4: Booth's Algorithm

- What is the result of -7×-5 using Booth's algorithm? Assume 4 bit 2's complement representation for the multiplicand and multiplier. Clearly show all the steps.