CS 61C Spring 2010

MIPS Encoding

Scott Beamer (cs61c-ta)

Week 6 (2/23)

Bit Masking

- *Way #1* - use AND with 1's where you want to mask, followed by a shift to move it right (if necessary)
- *Way #2* - shift left until start of field, then shift right to align
- Given an input in $a0, mask out the following fields into the given registers.

$t0 (bits 31-16) $t1 (bits 15-8) $t2 (bits 23-20)

Way #1	Way #2

R-Format Instructions (Register)

- *opcode* of 0 indicates it is R-Format
- *funct* field specifies actual operation
- Examples: add, sll, slt, …

opcode (0)	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

I-Format Instructions (Immediate)

- *opcode* field specifies operation
- *immediate* is typically sign-extended for arithmetic and zero-extended for logic
- Examples: addi, andi, lw, …

opcode	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

J-Format Instructions (Jump)

- *opcode* field specifies operation
- *address* is word addressed and is an absolute address (not relative)
- Examples: j, jal, …

opcode	address
6 bits	26 bits

Things to Watch Out For

- R-Format instructions generally write to *rd* while I-Format generally write to *rt*
- *Unsigned* can mean many things in the way it affects an instruction
- jr is an R-Format instruction (jump address comes from register, not instruction)

Addressing in MIPS

- The *Program Counter (PC)* holds the address of the currently executing instruction
- Branch Addressing - uses *Relative Addressing* - beq, bne, bgez, bltz, …
  nextPC = signExtend(immediate<<2) + PC + 4
- Memory Addressing - uses *Base Displacement Addressing* - sw, lw, sb, lb, …
  memAddr = signExtend(immediate) + R[rs]
- Jump Addressing - uses *Pseudodirect Addressing (Absolute)* - j, jal, …
  nextPC = PC[31:28] | zeroExtend(address<<2)
- Jump Register Addressing - uses *Register Addressing* - jr
  nextPC = $ra

Example MIPS Assembling

Fill in the following table with the correct fields from the MIPS routine below. Then fill in the second table with each instruction translated to the raw 32-bit hex number.

Addr	opcode	rs	rt	rd	shamt	funct
0x00						
0x04						
0x08						
0x0c						
0x10						
0x14						
0x18						
0x1c						
0x20						

Addr	MIPS	Raw Bits (in hex)
0x00	add $s0, $0, $0	
0x04	add $s1, $0, $0	
0x08	addi $s2, $0, 16	
0x0c	beq $s0, $s2, L2	
0x10	L1: lw $t0, 0($s0)	
0x14	add $s1, $0, $t0	
0x18	addi $s0, $s0, 4	
0x1c	j L1	
0x20	L2: sll $0, $0, 0	