

CS 203A
Advanced Computer Architecture

Lecture 4

MIPS64

Instructor: Jun Yang

Review

- Branches
- Instruction Format
- Role of Compiler
- RISC vs. CISC

Outline

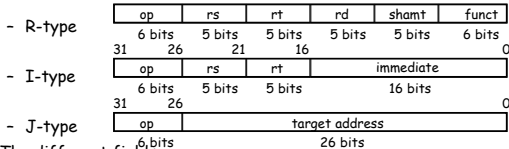
- MIPS64

MIPS64 Registers and Data Types

- 32 GPRs - R0, R1, ..., R31 (64-bit)
- 32 FPRs - F0, F1, ..., F31 (64-bit)
- R0 == 0
- Special registers:
 - PC, LO, HI, etc.
- Integer:
 - 8, 16, 32, 64-bit are supported
- Floating point:
 - single, double-precision are supported
- Narrow values are sign-extended or zero-extended.

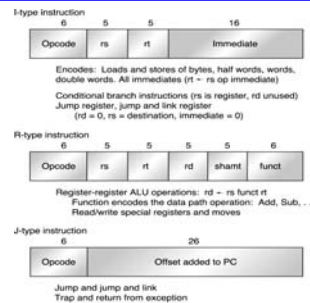
The MIPS64 Instruction Formats

- All MIPS instructions are 32 bits long. The three instruction formats:



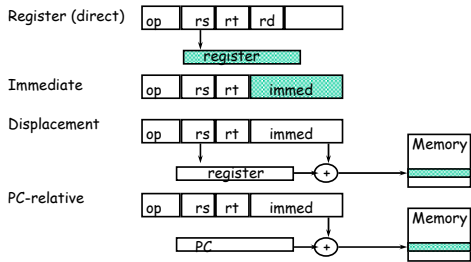
- The different fields are:
 - op: operation of the instruction
 - rs, rt, rd: the source and destination register specifiers
 - shamt: shift amount
 - funct: selects the variant of the operation in the "op" field
 - address / immediate: address offset or immediate value
 - target address: target address of the jump instruction

MIPS Instruction Layout



MIPS Addressing Modes/Instruction Formats

- All instructions 32 bits wide



Oct. 8, 2002

Lec. 4

7

MIPS I Operation Overview

Arithmetic Logical:

- Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU
- AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, LUI
- SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access:

- LB, LBU, LH, LHU, LW, LWL, LWR
- SB, SH, SW, SWL, SWR

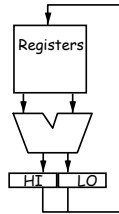
Oct. 8, 2002

Lec. 4

8

Multiply / Divide

- Start multiply, divide
 - MULT rs, rt
 - MULTU rs, rt
 - DIV rs, rt
 - DIVU rs, rt
- Move result from multiply, divide
 - MFHI rd
 - MFLO rd
- Move to HI or LO
 - MTHI rd
 - MTLO rd



- Why not Third field for destination? (Hint: how many clock cycles for multiply or divide vs. add?)

Oct. 8, 2002

Lec. 4

9

MIPS Arithmetic Instructions

Instruction	Example	Meaning	Comments
add	add \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; exception possible
subtract	sub \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; exception possible
add immediate	addi \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; exception possible
add unsigned	addu \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; no exceptions
sub unsigned	subu \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; no exceptions
add imm. unsign.	addiu \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; no exceptions
multiply	mult \$2,\$3	Hi, Lo = $\$2 \times \3	64-bit signed product
multiply unsigned	multu \$2,\$3	Hi, Lo = $\$2 \times \3	64-bit unsigned product
divide	div \$2,\$3	Lo = $\$2 \div \3 , Hi = remainder	
divide unsigned	divu \$2,\$3	Lo = $\$2 \div \3 , Hi = $\$2 \bmod \3	Unsigned quotient & remainder
Move from Hi	mfhi \$1	$\$1 = \text{Hi}$	Used to get copy of Hi
Move from Lo	mflo \$1	$\$1 = \text{Lo}$	Used to get copy of Lo

Which add for address arithmetic? Which add for integers?

Oct. 8, 2002

Lec. 4

10

MIPS Logical Instructions

Instruction	Example	Meaning	Comment
and	and \$1,\$2,\$3	$\$1 = \$2 \& \$3$	3 reg. operands; Logical AND
or	or \$1,\$2,\$3	$\$1 = \$2 \$3$	3 reg. operands; Logical OR
xor	xor \$1,\$2,\$3	$\$1 = \$2 \wedge \$3$	3 reg. operands; Logical XOR
nor	nor \$1,\$2,\$3	$\$1 = \sim(\$2 \$3)$	3 reg. operands; Logical NOR
and immediate	andi \$1,\$2,10	$\$1 = \$2 \& 10$	Logical AND reg, constant
or immediate	ori \$1,\$2,10	$\$1 = \$2 10$	Logical OR reg, constant
xor immediate	xori \$1,\$2,10	$\$1 = \$2 \wedge 10$	Logical XOR reg, constant
shift left logical	sll \$1,\$2,10	$\$1 = \$2 \ll 10$	Shift left by constant
shift right logical	srl \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right by constant
shift right arithm.	sra \$1,\$2,10	$\$1 = \$2 \gg 10$	Shift right (sign extend)
shift left logical	sllv \$1,\$2,\$3	$\$1 = \$2 \ll \$3$	Shift left by variable
shift right logical	srlv \$1,\$2,\$3	$\$1 = \$2 \gg \$3$	Shift right by variable
shift right arithm.	srav \$1,\$2,\$3	$\$1 = \$2 \gg \$3$	Shift right arith. by variable

Oct. 8, 2002

Lec. 4

11

MIPS Data Transfer Instructions

Instruction	Comment
SW 500(R4), R3	Store word
SH 502(R2), R3	Store half
SB 41(R3), R2	Store byte
LW R1, 30(R2)	Load word
LH R1, 40(R3)	Load halfword
LHU R1, 40(R3)	Load halfword unsigned
LB R1, 40(R3)	Load byte
LBU R1, 40(R3)	Load byte unsigned
LUI R1, 40	Load Upper Immediate (16 bits shifted left by 16)

Oct. 8, 2002

Lec. 4

12

Methods of Testing Condition

- **Condition Codes**
 - Processor status bits are set as a side-effect of arithmetic instructions or explicitly by compare or test instructions.
 - ex: add r1, r2, r3
bz label
- **Condition Register**
 - Ex: cmp r1, r2, r3
bgtz r1, label
- **Compare and Branch**
 - Ex: beq r1, r2, label

Oct. 8, 2002

Lec. 4

13

MIPS Compare and Branch

- **Compare and Branch**
 - BEQ rs, rt, offset if R[rs] == R[rt] then PC-relative branch
 - BNE rs, rt, offset \leftrightarrow
- **Compare to zero and Branch**
 - BLEZ rs, offset if R[rs] <= 0 then PC-relative branch
 - BGTZ rs, offset >
 - BLT <
 - BGEZ >=
 - BLTZAL rs, offset if R[rs] < 0 then branch and link (into R 31)
 - BGEZAL >=!
- Remaining set of compare and branch ops take two instructions
- Almost all comparisons are against zero!

Oct. 8, 2002

Lec. 4

14

MIPS Jump, Branch, Compare Instructions

Instruction	Example	Meaning
branch on equal	beq \$1,\$2,100	if (\$1 == \$2) go to PC+4+100 <i>Equal test; PC relative branch</i>
branch on not eq.	bne \$1,\$2,100	if (\$1 != \$2) go to PC+4+100 <i>Not equal test; PC relative</i>
set on less than	slt \$1,\$2,\$3	if (\$2 < \$3) \$1=1; else \$1=0 <i>Compare less than; 2's comp.</i>
set less than imm.	slti \$1,\$2,100	if (\$2 < 100) \$1=1; else \$1=0 <i>Compare < constant; 2's comp.</i>
set less than uns.	sltu \$1,\$2,\$3	if (\$2 < \$3) \$1=1; else \$1=0 <i>Compare less than; natural numbers</i>
set l. t. imm. uns.	sltiu \$1,\$2,100	if (\$2 < 100) \$1=1; else \$1=0 <i>Compare < constant; natural numbers</i>
jump	j 10000	go to 10000 <i>Jump to target address</i>
jump register	jr \$31	go to \$31 <i>For switch, procedure return</i>
jump and link	jal 10000	\$31 = PC + 4; go to 10000 <i>For procedure call</i>

Oct. 8, 2002

Lec. 4

15

When does MIPS Sign Extend?

- When value is sign extended, copy upper bit to full value:
 - Examples of sign extending 8 bits to 16 bits:


```
00001010 => 00000000 00001010
10001100 => 11111111 10001100
```
- When is an immediate value sign extended?
 - Arithmetic instructions (add, sub, etc.) sign extend immediates *even for the unsigned versions of the instructions!*
 - Logical instructions *do not sign extend*
- Load/Store half or byte *do sign extend*, but unsigned versions do not

Oct. 8, 2002

Lec. 4

16

Signed vs. Unsigned Comparison

R1= 0...00 0000 0000 0000 0001
R2= 0...00 0000 0000 0000 0010
R3= 1...11 1111 1111 1111 1111

- After executing these instructions:
 - slt r4,r2,r1 ; if (r2 < r1) r4=1; else r4=0
 - slt r5,r3,r1 ; if (r3 < r1) r5=1; else r5=0
 - sltu r6,r2,r1 ; if (r2 < r1) r6=1; else r6=0
 - sltu r7,r3,r1 ; if (r3 < r1) r7=1; else r7=0
 - What are values of registers r4 - r7? Why?
- r4 = ; r5 = ; r6 = ; r7 = ;

Oct. 8, 2002

Lec. 4

17

Details of the MIPS Instruction Set

- Register zero always has the value zero (even if you try to write it)
- Branch/jump and link put the return addr. PC+4 or 8 into the link register (R31) (depends on logical vs physical architecture)
- All instructions change all 32 bits of the destination register (including lui, lb, lh) and all read all 32 bits of sources (add, sub, and, or, ...)
- Immediate arithmetic and logical instructions are extended as follows:
 - Logical immediates ops are zero extended to 32 bits
 - Arithmetic immediates ops are sign extended to 32 bits (including addu)
- The data loaded by the instructions lb and lh are extended as follows:
 - lbu, lhu are zero extended
 - lb, lh are sign extended
- Overflow can occur in these arithmetic and logical instructions:
 - add, sub, addi
 - It cannot occur in addu, subu, addiu, and, or, xor, nor, shifts, mult, multu, div, divu

Oct. 8, 2002

Lec. 4

18

Summary: Instruction Set Design (MIPS)

- Use general purpose registers with a load-store architecture: [YES](#)
- Provide at least 16 general purpose registers plus separate floating-point registers: [31 GPR & 32 FPR](#)
- Support basic addressing modes: displacement (with an address offset size of 12 to 16 bits), immediate (size 8 to 16 bits), and register deferred: : [YES: 16 bits for immediate, displacement \(disp=0 => register deferred\)](#)
- All addressing modes apply to all data transfer instructions : [YES](#)
- Use fixed instruction encoding if interested in performance and use variable instruction encoding if interested in code size : [Fixed](#)
- Support these data sizes and types: 8-bit, 16-bit, 32-bit integers and 32-bit and 64-bit IEEE 754 floating point numbers: [YES](#)
- Support these simple instructions, since they will dominate the number of instructions executed: load, store, add, subtract, move register-register, and, shift, compare equal, compare not equal, branch (with a PC-relative address at least 8-bits long), jump, call, and return: [YES](#)
- Aim for a minimalist instruction set: [YES](#)