EE 3613: Computer Organization
Chapter 2: Instruction Set Architecture – 2/3

Avinash Karanth
Department of Electrical Engineering & Computer Science
Ohio University, Athens, Ohio 45701
E-mail: karanth@ohio.edu
Website:
http://oucsace.cs.ohiou.edu/~avinashk/classes/ee461a/ee461a.htm

1

MIPS Memory Instructions

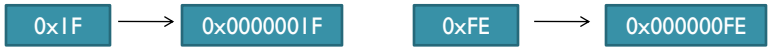
- Supports Base + Displacement mode only
- Format: 2 registers (dest, base) and 16-bit immediate (offset)
  - Example: `lw $s3, 1000($s4)`
  - Retrieves a word (32 bits) from address ($s4 + 1000)
  - Example: `lh $s3, 1000($s4)`
  - Retrieves a halfword (16 bits) from address ($s4+1000)
  - Example: `lb $s3, 1000($s4)`
  - Retrieves a byte (8 bits) from address ($s4+1000)

2

Sign/Zero Extension

- Registers in MIPS are all 32-bits !
- So what happens when you load 8 or 16 bits?
  - Sign extend if the load is signed

  - Zero extend if the load is unsigned



3

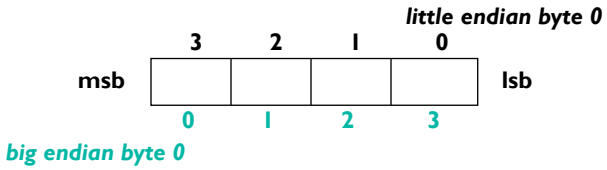
MIPS Memory Instruction - Recap

- Load Instruction
  - lb \\ load byte signed
  - lbu \\ load byte unsigned (zero extend)
  - lh \\ load halfword
  - lhu \\ load halfword unsigned (zero extend)
  - lw \\ load word
- Store Instruction
  - sb $s3, 1000($s4) \\ store 8 LSBs of $s3 to M[$s4 + 1000]
  - sh $s3, 1000($s4) \\ store 16 LSBs of $s3 to M[$s4 + 1000]
  - sw $s3, 1000($s4) \\ store all 32 bits of $s3 to M[$s4+1000]

4

Byte Addresses

- Since 8-bit addresses are so useful, most architectures address individual bytes in memory
  - The memory address of a word must be multiple of 4 (alignment restriction)
- Big Endian : leftmost byte is word address
  - IBM 360/370, Motorola 68K, MIPS, Sparc, HP PA
- Little Endian: rightmost byte is word address
  - Intel 80x86, DEC Vax, DEC Alpha

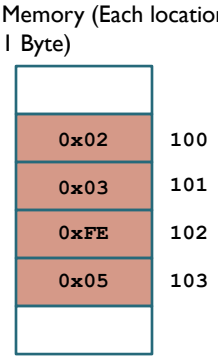


5

Example Code Sequence

- What is the final state of memory once you execute the following instruction sequence ($s0 = 0)?

```
lw   $s4, 100($s0)
lb   $s3, 102($s0)
sw   $s3, 100($s0)
sb   $s4, 102($s0)
```

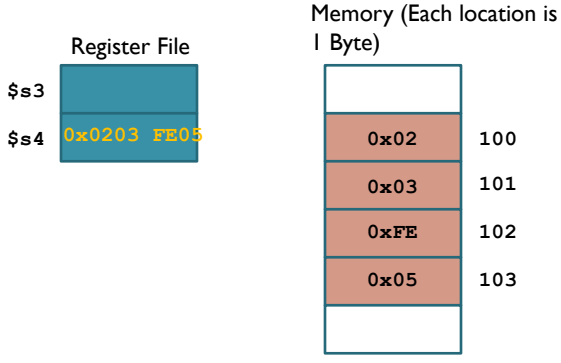


6

Example Code Sequence - 1

- What is the final state of memory once you execute the following instruction sequence?

```
lw   $s4, 100($s0)
lb   $s3, 102($s0)
sw   $s3, 100($s0)
sb   $s4, 102($s0)
```

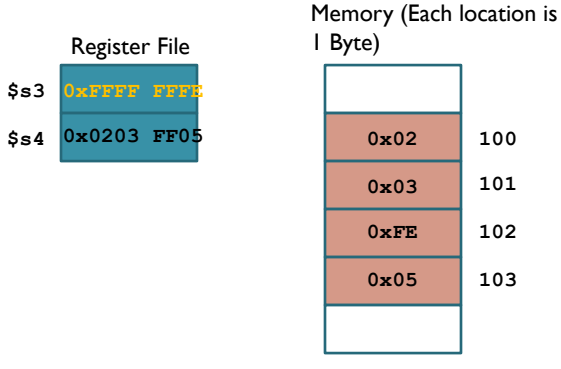


7

Example Code Sequence - 2

- What is the final state of memory once you execute the following instruction sequence?

```
lw   $s4, 100($s0)
lb   $s3, 102($s0)
sw   $s3, 100($s0)
sb   $s4, 102($s0)
```

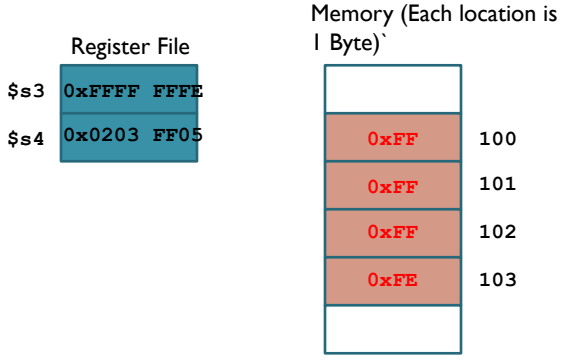


8

Example Code Sequence - 3

- What is the final state of memory once you execute the following instruction sequence?

```
lw   $s4, 100($s0)
lb   $s3, 102($s0)
sw   $s3, 100($s0)
sb   $s4, 102($s0)
```

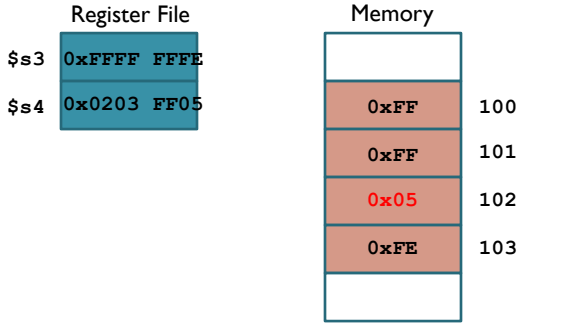


9

Example Code Sequence - 4

- What is the final state of memory once you execute the following instruction sequence?

```
lw   $s4, 100($s0)
lb   $s3, 102($s0)
sw   $s3, 100($s0)
sb   $s4, 102($s0)
```



10

Convert C to Assembly

Example 1

a = b + names[i]

Assume that a is in $s1, b is in $s2, i is in $s3 and the array names starts at address 1000 and holds 32-bit integers

```
lw  $t0, 1000($s3)    # load word
add $s1, $s2, $t0     # addition
```

Example 2

A[12] = b + A[8]

Assume that b is in $s2, base address of A in $s3

```
lw  $t0, 32($s3)    # load word
add $t0, $s2, $t0
sw  $t0, 48($s3)    # store word
```

11

What about Immediate Operands?

- Small constants often used, keep the constant inside the instruction itself

```
addi   $sp, $sp, 4   #$sp = $sp + 4
```

- Machine I-Format

- What about large operands?

12

Aside: How About Larger Constants?

- We'd also like to be able to load a 32 bit constant into a register, for this we must use two instructions
- a new "load upper immediate" instruction

```
lui $t0, 255
```

16	0	8	0000 0000 1111 1111

- Then must get the lower order bits right, use

```
ori $t0, $t0, 43690
```

0000 0000 1111 1111	0000000000000000
0000 0000 1111 1111	1010101010101010
0000 0000 1111 1111	1010101010101010

13

MIPS Control Flow Instructions

- MIPS conditional branch instructions:

```
bne $s0, $s1, Lbl     #go to Lbl if $s0≠$s1
beq $s0, $s1, Lbl     #go to Lbl if $s0=$s1
```

  - Ex:

```
        if (i==j) h = i + j;
        bne $s0, $s1, Lbl1
        add $s3, $s0, $s1
Lbl1:   ...
```

- Instruction Format (I format):

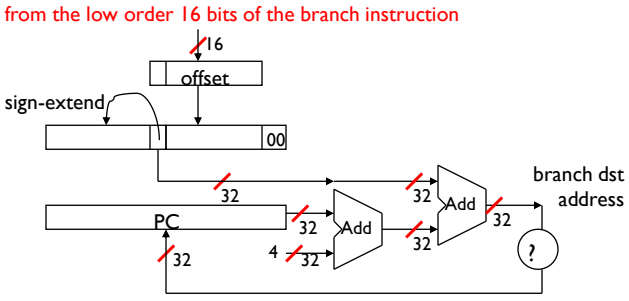
op	rs	rt	16-bit Offset

- How is the branch destination address specified?

14

Specifying Branch Destinations

- Use a register (like in lw and sw) added to the 16-bit offset
  - which register? Instruction Address Register (the Program Counter)
    - its use is automatically implied by instruction
    - PC gets updated (PC+4) during the fetch cycle so that it holds the address of the next instruction
  - limits the branch distance to -2^{15} to $+2^{15}-1$ instructions from the (instruction after the) branch instruction, but most branches are local anyway



15

More Branch Instructions

- We have `beq`, `bne`, but what about other kinds of branches (e.g., branch-if-less-than)? For this, we need yet another instruction, `slt`
- Set on less than instruction:

```
slt $t0, $s0, $s1     # if $s0 < $s1     then
                      # $t0 = 1          else
                      # $t0 = 0
```

- Instruction format (R format):

op	rs	rt	rd	shamt	funct

16

More Branch Instructions

- Can use `slt`, `beq`, `bne`, and the fixed value of 0 in register `$zero` to create other conditions
  - less than `blt $s1, $s2, Label`

```
slt  $at, $s1, $s2      #$at set to 1 if
bne  $at, $zero, Label  #  $s1 < $s2
```

  - less than or equal to `ble $s1, $s2, Label`
  - greater than `bgt $s1, $s2, Label`
  - great than or equal to `bge $s1, $s2, Label`

- Such branches are included in the instruction set as pseudo instructions - recognized (and expanded) by the assembler
  - Its why the assembler needs a reserved register (`$at`)

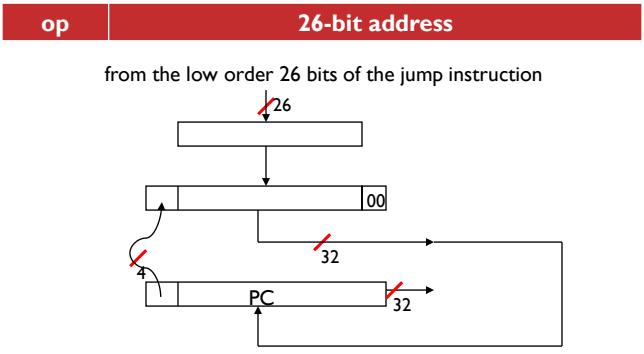
17

Other Control Flow Instructions

- MIPS also has an unconditional branch instruction or jump instruction:

```
j   label       #go to label
```

- Instruction Format (J Format):



18

Aside: Branching Far Away

- What if the branch destination is further away than can be captured in 16 bits?,
- The assembler comes to the rescue – it inserts an unconditional jump to the branch target and inverts the condition

```
        beq   $s0, $s1, L1
```

becomes

```
        bne   $s0, $s1, L2
        j     L1
L2:
```

19

MIPS ISA So Far

Category	Instr	Op Code	Example	Meaning
Arithmetic (R & I format)	add	0 and 32	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	0 and 34	sub $s1, $s2, $s3	$s1 = $s2 - $s3
	add immediate	8	addi $s1, $s2, 6	$s1 = $s2 + 6
	or immediate	13	ori $s1, $s2, 6	$s1 = $s2 v 6
Data Transfer (I format)	load word	35	lw $s1, 24($s2)	$s1 = Memory($s2+24)
	store word	43	sw $s1, 24($s2)	Memory($s2+24) = $s1
	load byte	32	lb $s1, 25($s2)	$s1 = Memory($s2+25)
	store byte	40	sb $s1, 25($s2)	Memory($s2+25) = $s1
	load upper imm	15	lui $s1, 6	$s1 = 6 * 2^{16}
Cond. Branch (I & R format)	br on equal	4	beq $s1, $s2, L	if ($s1==$s2) go to L
	br on not equal	5	bne $s1, $s2, L	if ($s1 !=$s2) go to L
	set on less than	0 and 42	slt $s1, $s2, $s3	if ($s2<$s3) $s1=1 else $s1=0
	set on less than immediate	10	slti $s1, $s2, 6	if ($s2<6) $s1=1 else $s1=0
Uncond. Jump (J & R format)	jump	2	j 2500	go to 10000
	jump register	0 and 8	jr $t1	go to $t1
	jump and link	3	jal 2500	go to 10000; $ra=PC+4

20

Register Organization

Name	Register Number	Usage	Preserved on Call
$zero	0	Constant Value of 0	n.a.
$v0-$v1	2-3	Values for results and expressions	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved	Yes
$t8-$t9	24-25	More Temporaries	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

- What about Register 1?

21