C335
Computer Structures

MIPS Instructions (Part #5)

Dr. Liqiang Zhang

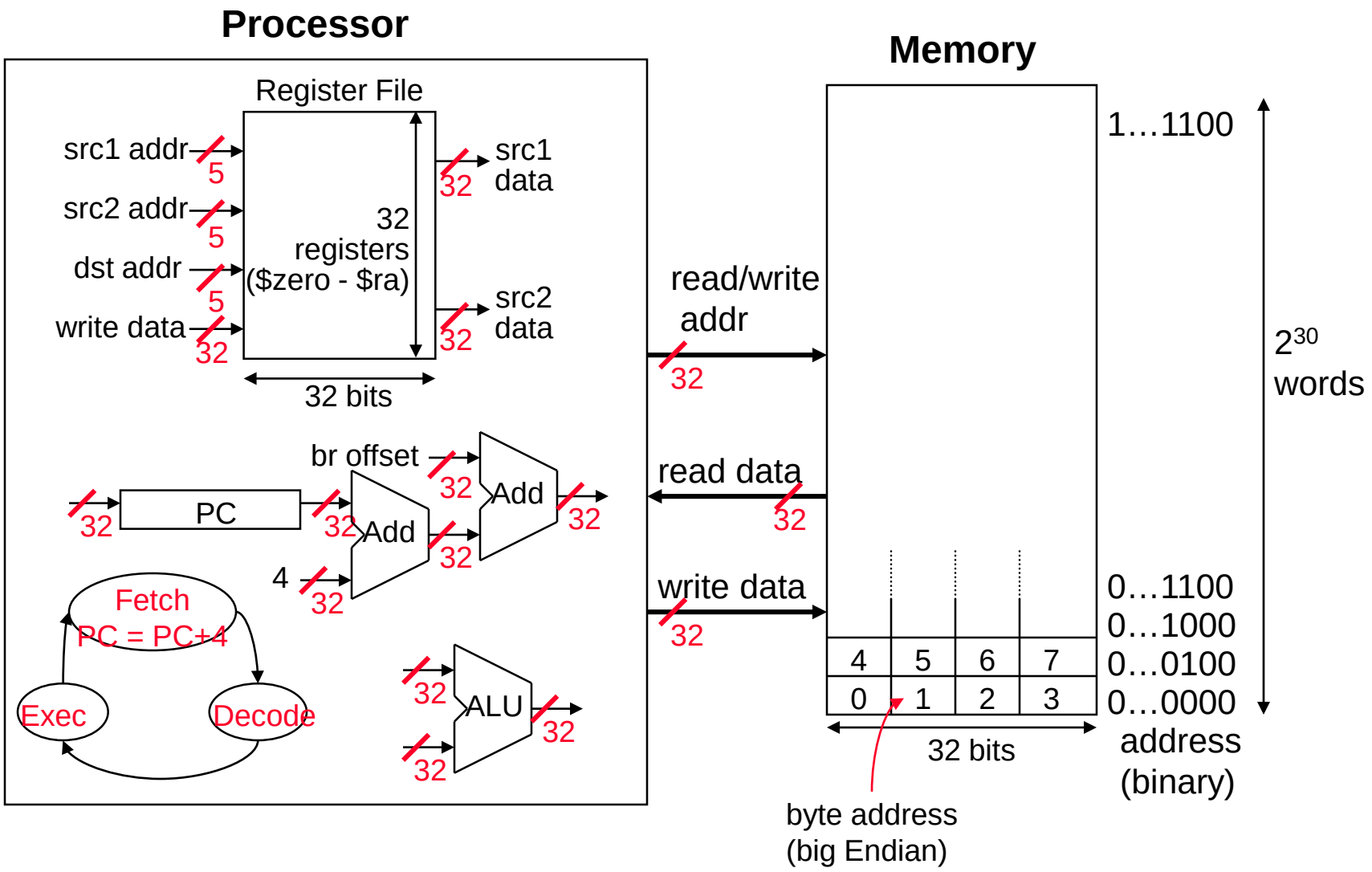
Department of Computer and Information Sciences

Adapted from Morgan Kaufmann and others

Review: MIPS Instructions, so far

Category	Instr	OpCd	Example	Meaning
Arithmetic (R & I format)	add	0 & 20	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	0 & 22	sub $s1, $s2, $s3	$s1 = $s2 - $s3
	add immediate	8	addi $s1, $s2, 4	$s1 = $s2 + 4
Data transfer (I format)	load word	23	lw $s1, 100($s2)	$s1 = Memory($s2+100)
	store word	2b	sw $s1, 100($s2)	Memory($s2+100) = $s1
Cond. branch (I 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 immediate	a	slti $s1, $s2, 100	if ($s2<100) $s1=1; else $s1=0
(R format)	set on less than	0 & 2a	slt $s1, $s2, $s3	if ($s2<$s3) $s1=1; else $s1=0
Uncond. jump	jump	2	j 2500	go to 10000
	jump register	0 & 08	jr $t1	go to $t1

MIPS Organization



Review: Bits can represent anything!!

- Characters
  - 26 letters $\Rightarrow$ 5 bits ($2^5 = 32$)
  - upper/lower case + punctuation $\Rightarrow$ 7 bits (in 8) ("ASCII")
  - standard code to cover all the world's languages $\Rightarrow$ 8,16,32 bits ("Unicode") `www.unicode.com`
- Logical values
  - 0 $\Rightarrow$ False, 1 $\Rightarrow$ True
- colors
- locations / addresses / commands

Byte Addresses

- Since bytes (8 bits) are so useful, most ISAs support addressing individual bytes in memory
- **Endianness**: refer to how bytes of a data word are ordered within memory.

- Little Endian: increasing numeric significance with increasing memory addresses

  Intel 80x86, x86-64, DEC Vax, DEC Alpha, Z80, 8051

- Big Endian: on the contrary to Little Endian

  IBM 360/370, Motorola 68k, SPARC (before v9)

- Bi-Endian: can switch between the two endianness

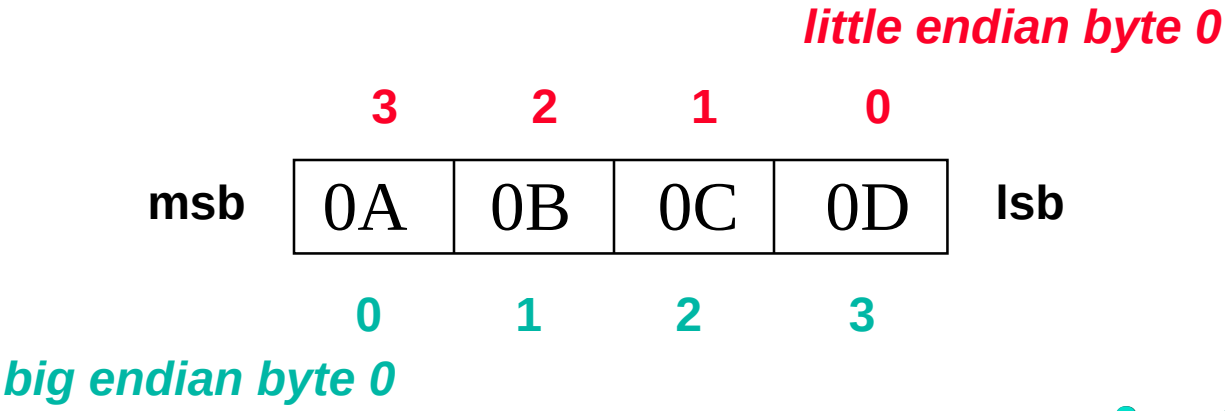
  MIPS, Power PC, SPARC (v9), ARM (after v3), IA-64

Byte Addresses

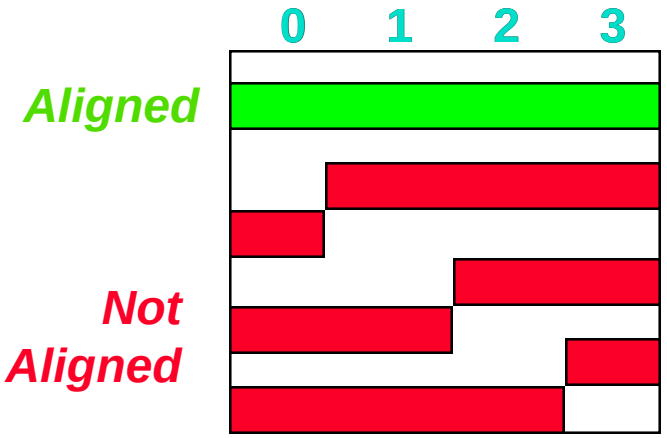
- For example: to store 0x0A0B0C0D in memory:
- If Little Endian:
  - Base Address+0: 0D
  - Base Address+1: 0C
  - Base Address+2: 0B
  - Base Address+3: 0A
- If Big Endian:
  - Base Address+0: 0A
  - Base Address+1: 0B
  - Base Address+2: 0C
  - Base Address+3: 0D

Addressing Objects: Endianess and Alignment

- Big Endian: leftmost byte is the address of word
- Little Endian: rightmost byte is the address of word



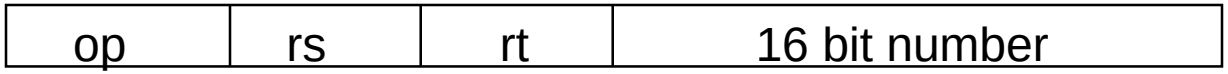
Alignment restriction: requires that objects fall on address that is multiple of their size



Loading and Storing Bytes

- MIPS provides special instructions to move bytes

```
lb   $t0, 1($s3)  #load byte from memory
sb   $t0, 6($s3)  #store byte to  memory
```



- What 8 bits get loaded and stored?
  - load byte places the byte from memory in the rightmost 8 bits of the destination register
    - what happens to the other bits in the register?
  - store byte takes the byte from the rightmost 8 bits of a register and writes it to the byte in memory
    - leaving the other bytes in the memory word unchanged

Example of Loading and Storing Bytes

- Given following code sequence and memory state what is the state of the memory after executing the code?

```
add   $s3, $zero, $zero
lb    $t0, 1($s3)
sb    $t0, 6($s3)
```

Memory	
0x 0 0 0 0 0 0 0 0	24
0x 0 0 0 0 0 0 0 0	20
0x 0 0 0 0 0 0 0 0	16
0x 1 0 0 0 0 0 1 0	12
0x 0 1 0 0 0 4 0 2	8
0x F F F F F F F F	4
0x 0 0 9 0 1 2 A 0	0
Data	Address (Decimal)

- What value is left in $t0?

  $t0 = 0xFFFFFF90

- What word is changed in Memory and to what?

  mem(4) = 0xFFFF90FF

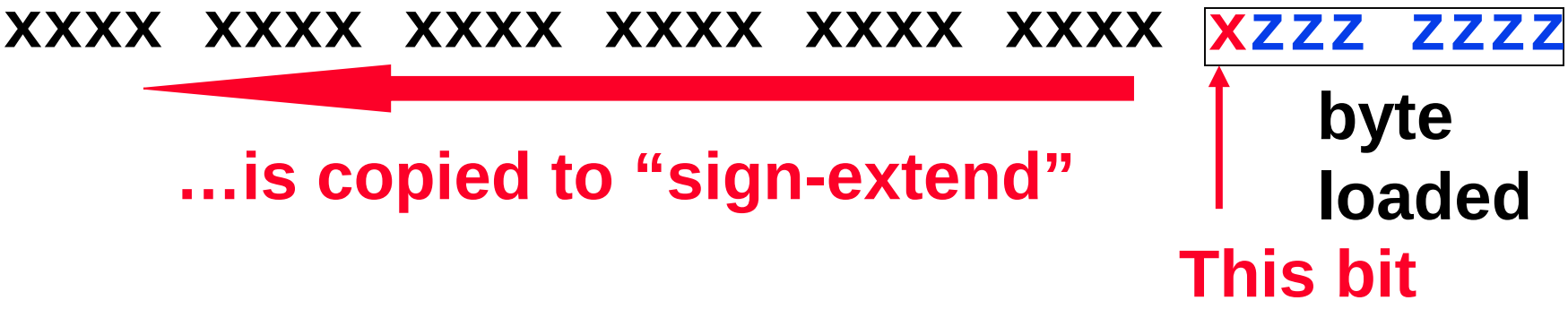
- What if the machine was little Endian?

  $t0 = 0x00000012

  mem(4) = 0xFF12FFFF

Loading and Storing Bytes

- What do with other 24 bits in the 32 bit register?
  - `lb`: sign extends to fill upper 24 bits



- **Normally don’t want to sign extend chars**
- **MIPS instruction that doesn’t sign extend when loading bytes:**

load byte unsigned: `lbu`

Loading and Storing Half Words

- MIPS also provides special instructions to move half words

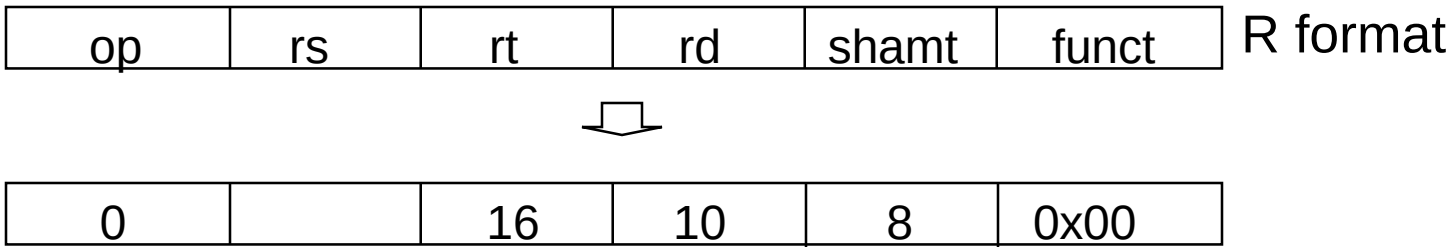
```
lh   $t0, 2($s3)  #load half word from memory
sh   $t0, 6($s3)  #store half word to  memory
```

- What 16 bits get loaded and stored?
  - load half word places the half word from memory in the rightmost 16 bits of the destination register
    - what happens to the other bits in the register?
  - store half word takes the half word from the rightmost 16 bits of the register and writes it to the half word in memory
    - leaving the other half word in the memory word unchanged

Shift Operations

- Need operations to pack and unpack 8-bit characters into 32-bit words
- Shifts move all the bits in a word left or right

```
sll $t2, $s0, 8    #$t2 = $s0 << 8 bits
srl $t2, $s0, 8    #$t2 = $s0 >> 8 bits
```

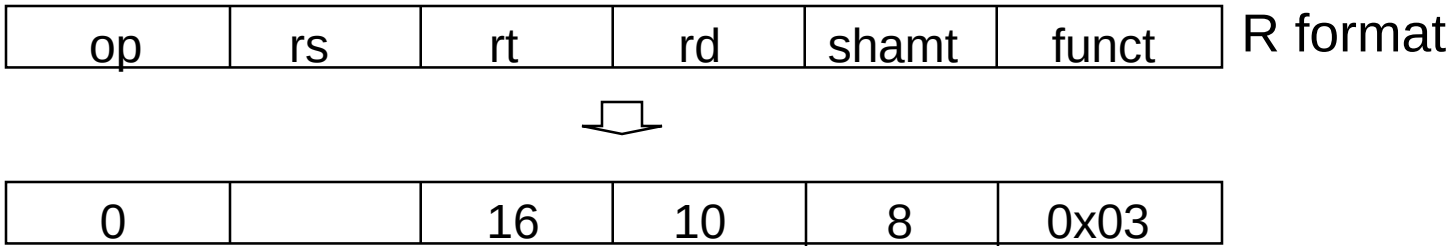


- Such shifts are called logical because they fill with zeros
  - Notice that a 5-bit shamt field is enough to shift a 32-bit value $2^5 - 1$ or 31 bit positions

More Shift Operations

- An arithmetic shift (`sra`) maintain the arithmetic correctness of the shifted value (i.e., a number shifted right one bit should be ½ of its original value; a number shifted left should be 2 times its original value)
  - `sra` uses the most significant bit (sign bit) as the bit shifted in
  - `sll` works for arithmetic left shifts for 2's compl. (so there is no need for a `sla`)

```
sra $t2, $s0, 8   #$t2 = $s0 >> 8 bits
```



Compiling Another While Loop

- Compile the assembly code for the C `while` loop where i is in `$s3`, k is in `$s5`, and the base address of the array `save` is in `$s6`

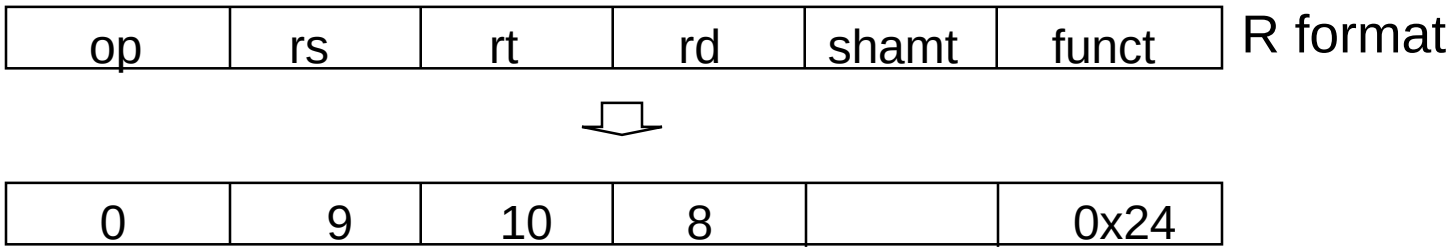
```
while (save[i] == k)
     i += 1;
```

```
Loop:   sll  $t1, $s3, 2
        add  $t1, $t1, $s6
        lw   $t0, 0($t1)
        bne  $t0, $s5, Exit
        addi $s3, $s3, 1
        j    Loop
Exit:   . . .
```

Logical Operations

- There are a number of bit-wise logical operations in the MIPS ISA

```
and $t0, $t1, $t2  #$t0 = $t1 & $t2
or  $t0, $t1, $t2  #$t0 = $t1 | $t2
nor $t0, $t1, $t2  #$t0 = not($t1 | $t2)
```




```
andi $t0, $t1, 0xff00   #$t0 = $t1 & ff00
ori  $t0, $t1, 0xff00   #$t0 = $t1 | ff00
```

Logic Operations

- Logic operations operate on individual bits of the operand.

```
                          $t2 = 0…0 0000 1101 0000
                          $t1 = 0…0 0011 1100 0000
and   $t0, $t1, $t2 $t0 = 0…0 0000 1100 0000

or    $t0, $t1, $t2 $t0 = 0…0 0011 1101 0000

nor   $t0, $t1, $t2 $t0 = 1…1 1100 0010 1111
```

- How about “not”?
  - No “not” Instruction in MIPS! Why?
  - A NOR 0 = NOT (A OR 0) = NOT (A)

How About Larger Constants?

- We'd also like to be able to load a 32-bit constant into a register, for example: 0xaaaabbbb
- Must use two instructions, new "load upper immediate" instruction

```
lui $t0, 0xaaaa
```

f	0	8	1010101010101010

- Then must get the lower order bits right, i.e.,

```
ori $t0, $t0, 0xbbbb
```

1010101010101010	0000000000000000
0000000000000000	1011101110111011

1010101010101010	1011101110111011

Review: MIPS Instructions, so far

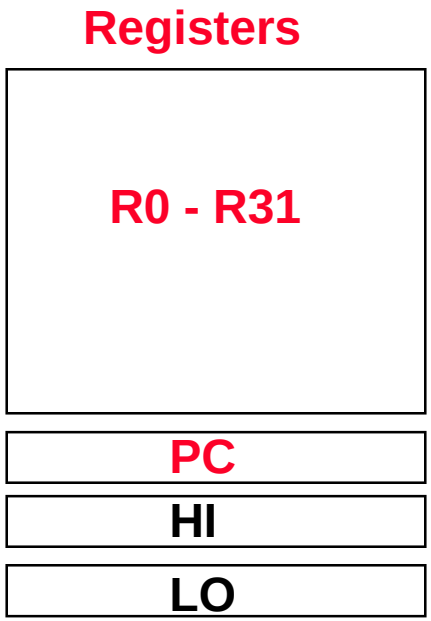
Category	Instr	OpC	Example	Meaning
Arithmetic (R & I format)	add	0 & 20	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	0 & 22	sub $s1, $s2, $s3	$s1 = $s2 - $s3
	add immediate	8	addi $s1, $s2, 4	$s1 = $s2 + 4
	shift left logical	0 & 00	sll $s1, $s2, 4	$s1 = $s2 << 4
	shift right logical	0 & 02	srl $s1, $s2, 4	$s1 = $s2 >> 4 (fill with zeros)
	shift right arithmetic	0 & 03	sra $s1, $s2, 4	$s1 = $s2 >> 4 (fill with sign bit)
	and	0 & 24	and $s1, $s2, $s3	$s1 = $s2 & $s3
	or	0 & 25	or $s1, $s2, $s3	$s1 = $s2 | $s3
	nor	0 & 27	nor $s1, $s2, $s3	$s1 = not ($s2 | $s3)
	and immediate	c	and $s1, $s2, ff00	$s1 = $s2 & 0xff00
	or immediate	d	or $s1, $s2, ff00	$s1 = $s2 | 0xff00
	load upper immediate	f	lui $s1, 0xffff	$s1 = 0xffff0000

Review: MIPS Instructions, so far

Category	Instr	OpC	Example	Meaning
Data transfer (I format)	load word	23	lw $s1, 100($s2)	$s1 = Memory($s2+100)
	store word	2b	sw $s1, 100($s2)	Memory($s2+100) = $s1
	load byte	20	lb $s1, 101($s2)	$s1 = Memory($s2+101)
	store byte	28	sb $s1, 101($s2)	Memory($s2+101) = $s1
	load half	21	lh $s1, 101($s2)	$s1 = Memory($s2+102)
	store half	29	sh $s1, 101($s2)	Memory($s2+102) = $s1
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 immediate	a	slti $s1, $s2, 100	if ($s2<100) $s1=1; else $s1=0
	set on less than	0 & 2a	slt $s1, $s2, $s3	if ($s2<$s3) $s1=1; else $s1=0
Uncond. jump	jump	2	j 2500	go to 10000
	jump register	0 & 08	jr $t1	go to $t1

Review: MIPS R3000 ISA

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special



- 3 Instruction Formats: all 32 bits wide