CENG 3420
Computer Organization and Design

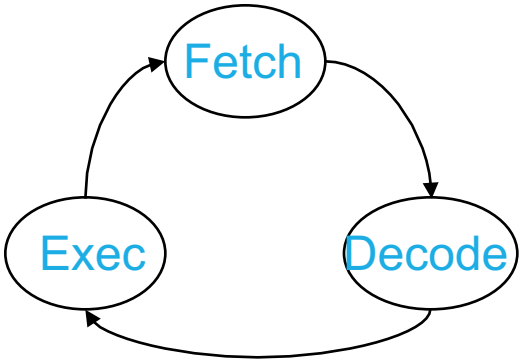
Lecture 03: Instruction Set Architecture Review

Bei Yu

香港中文大學
The Chinese University of Hong Kong

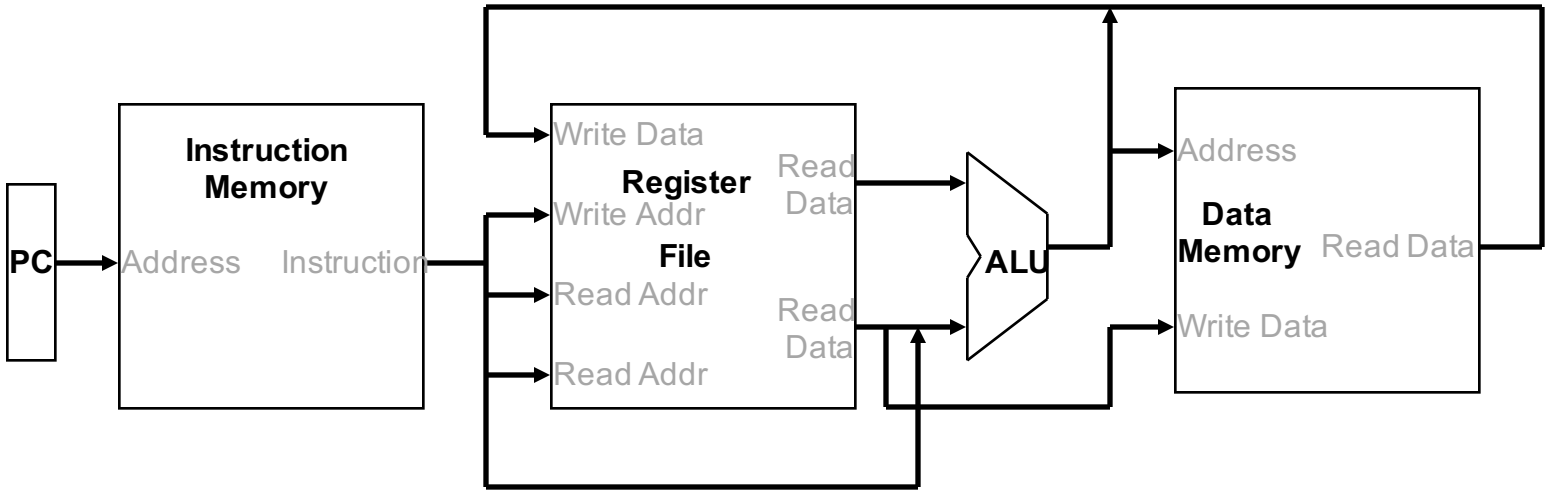
Review: Processor Organization

- Control needs to have circuitry to
  - Decide which is the next instruction and input it from memory
  - Decode the instruction
  - Issue signals that control the way information flows between datapath components
  - Control what operations the datapath's functional units perform



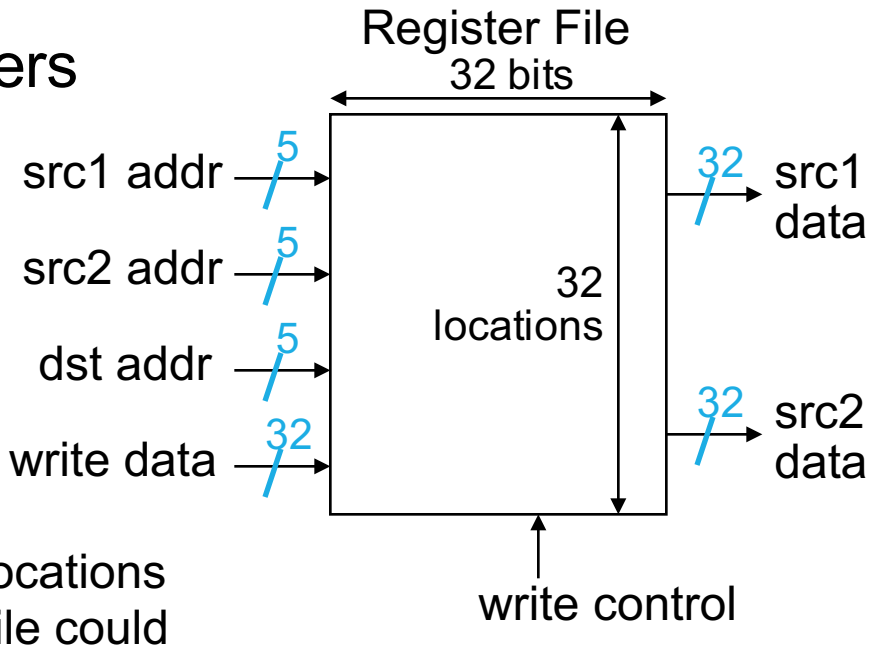
- Datapath needs to have circuitry to
  - Execute instructions - functional units (e.g., adder) and storage locations (e.g., register file)
  - Interconnect the functional units so that the instructions can be executed as required
  - Load data from and store data to memory

Abstract Implementation View



MIPS Register File

- Holds thirty-two 32-bit registers
  - Two read ports and
  - One write port



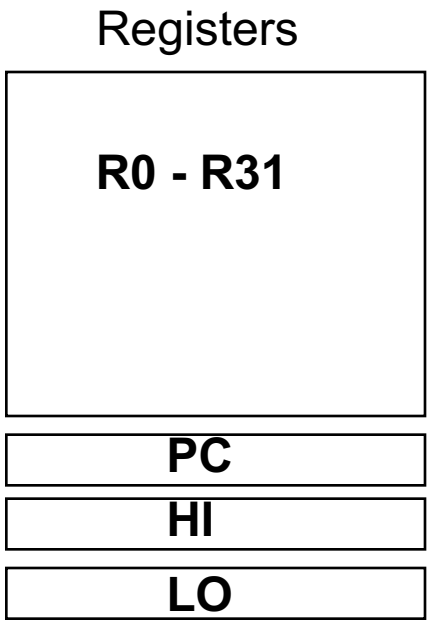
- Registers are
  - Faster than main memory
    - But register files with more locations are slower (e.g., a 64 word file could be as much as 50% slower than a 32 word file)
    - Read/write port increase impacts speed quadratically
  - Easier for a compiler to use
    - e.g., (A*B) – (C*D) – (E*F) can do multiplies in any order vs. stack
  - Can hold variables so that
    - code density improves (since register are named with fewer bits than a memory location)

RISC - Reduced Instruction Set Computer

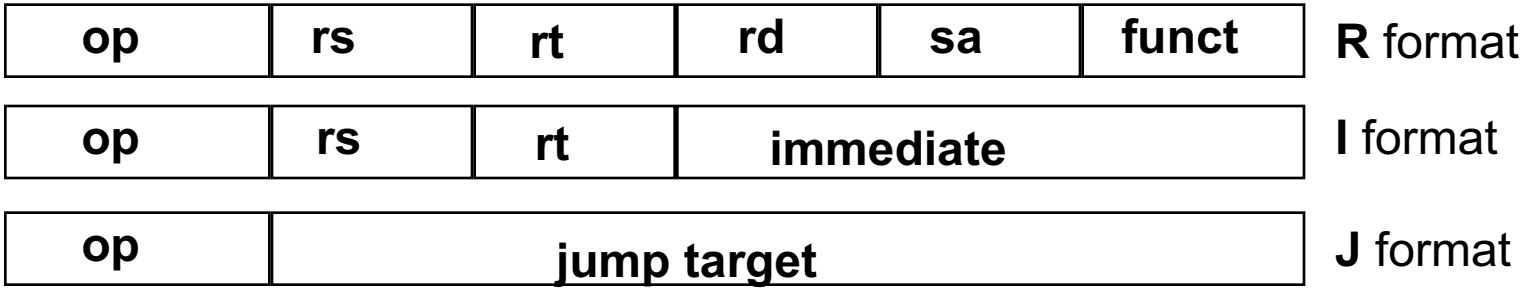
- RISC philosophy
  - fixed instruction lengths
  - load-store instruction sets
  - limited number of addressing modes
  - limited number of operations
- MIPS, Sun SPARC, HP PA-RISC, IBM PowerPC …
- Instruction sets are measured by how well compilers use them as opposed to how well assembly language programmers use them
- CISC (C for complex), e.g., Intel x86

MIPS-32 ISA

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

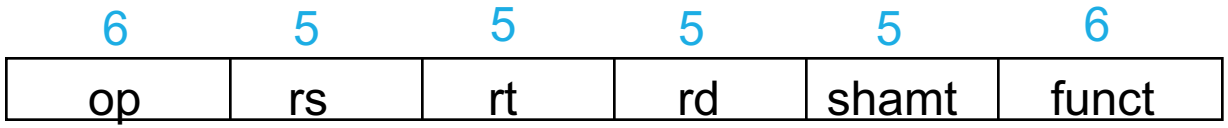


3 Instruction Formats: all 32 bits wide



MIPS Instruction Fields

- MIPS fields are given names to make them easier to refer to



op	6-bits	opcode that specifies the operation
rs	5-bits	register file address of the first source operand
rt	5-bits	register file address of the second source operand
rd	5-bits	register file address of the result's destination
shamt	5-bits	shift amount (for shift instructions)
funct	6-bits	function code augmenting the opcode

Aside: MIPS Register Convention

Name	Register Number	Usage	Preserve on call?
$zero	0	constant 0 (hardware)	n.a.
$at	1	reserved for assembler	n.a.
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	arguments	yes
$t0 - $t7	8-15	temporaries	no
$s0 - $s7	16-23	saved values	yes
$t8 - $t9	24-25	temporaries	no
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return addr (hardware)	yes

MIPS Arithmetic Instructions

- MIPS assembly language arithmetic statement

```
add   $t0, $s1, $s2
sub   $t0, $s1, $s2
```

- Each arithmetic instruction performs one operation
- Each specifies exactly three operands that are all contained in the datapath's register file (`$t0,$s1,$s2`)

destination = source1 op source2

- Instruction Format (R format)

MIPS Arithmetic Instructions

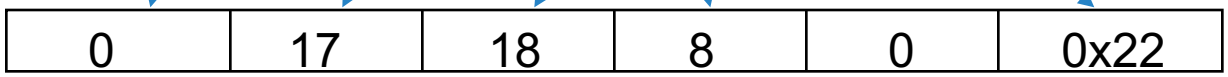
- MIPS assembly language arithmetic statement

```
add  $t0, $s1, $s2
sub  $t0, $s1, $s2
```

- Each arithmetic instruction performs one operation
- Each specifies exactly three operands that are all contained in the datapath's register file (`$t0,$s1,$s2`)

destination = source1 op source2

- Instruction Format (R format)



MIPS Memory Access Instructions

- MIPS has two basic data transfer instructions for accessing memory

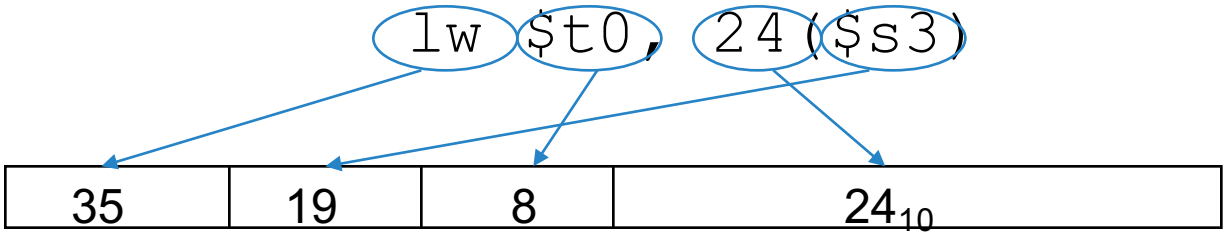
```
lw $t0, 4($s3)   #load word from memory

sw $t0, 8($s3)   #store word to memory
```

- The data is loaded into (lw) or stored from (sw) a register in the register file – a 5 bit address

- The memory address – a 32 bit address – is formed by adding the contents of the base address register to the offset value
  - A 16-bit field meaning access is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register

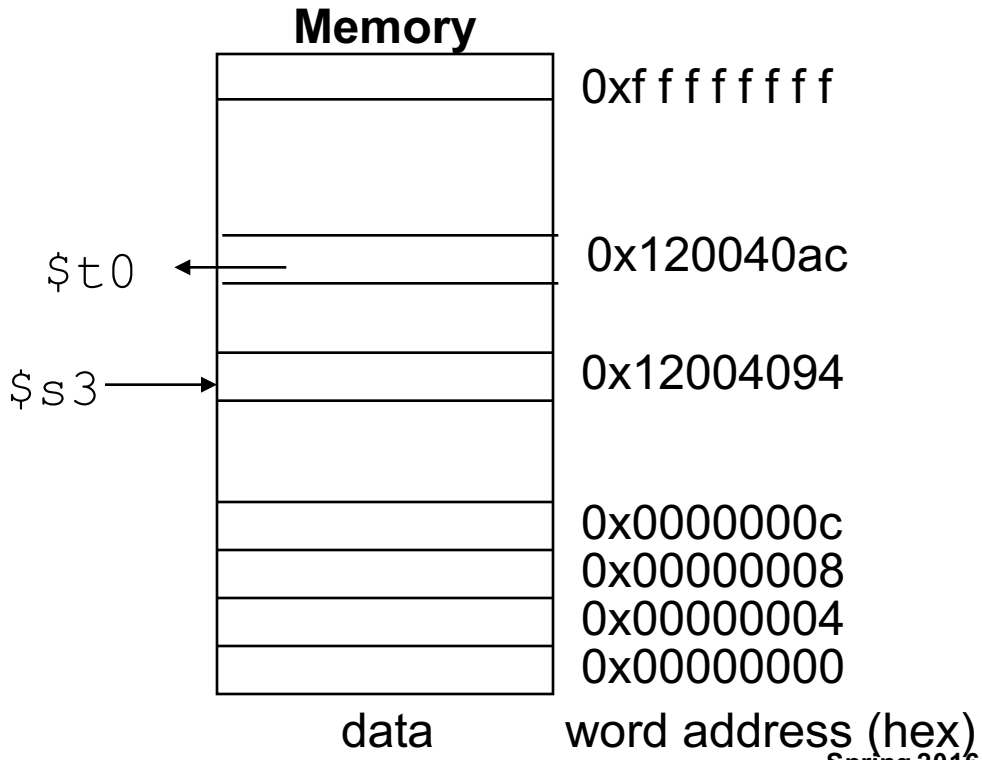
Machine Language - Load Instruction

- Load/Store Instruction Format (I format):



24_{10} + $s3 =

. . . 0001 1000
+ . . . 1001 0100
. . . 1010 1100 =
0x120040ac

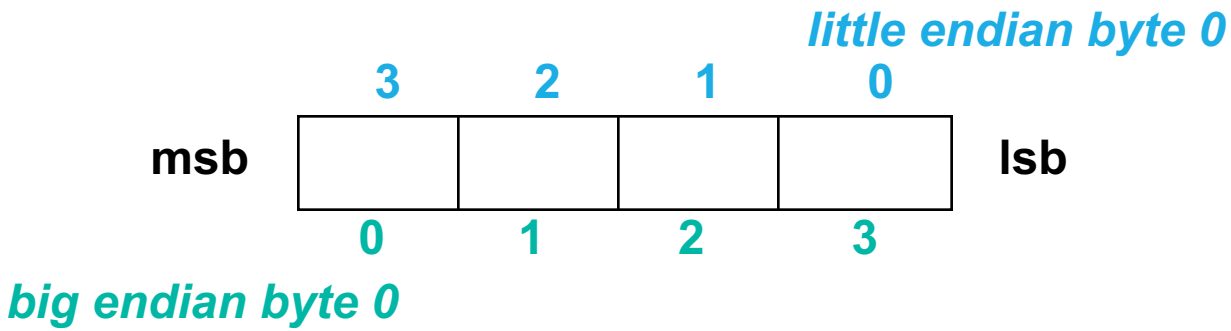


Byte Addresses

- Since 8-bit bytes are so useful, most architectures address individual bytes in memory
  - Alignment restriction - the memory address of a word must be on natural word boundaries (a multiple of 4 in MIPS-32)
- 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 (Windows NT)



Aside: 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 a byte in memory
    - what happens to the other bits in the memory word?

EX-1:

- 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)
```

- What value is left in $t0?

Memory	Word Address (Decimal)
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	

- What word is changed in Memory and to what?
- What if the machine was little Endian?

MIPS Immediate Instructions

- Small constants are used often in typical code
- Possible approaches?
  - put "typical constants" in memory and load them
  - create hard-wired registers (like $zero) for constants like 1
  - have special instructions that contain constants !

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

slti  $t0, $s2, 15    #$t0 = 1 if $s2<15
```

  - Machine format (I format):

- The constant is kept inside the instruction itself!
  - Immediate format limits values to the range $+2^{15}-1$ to -2^{15}

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):

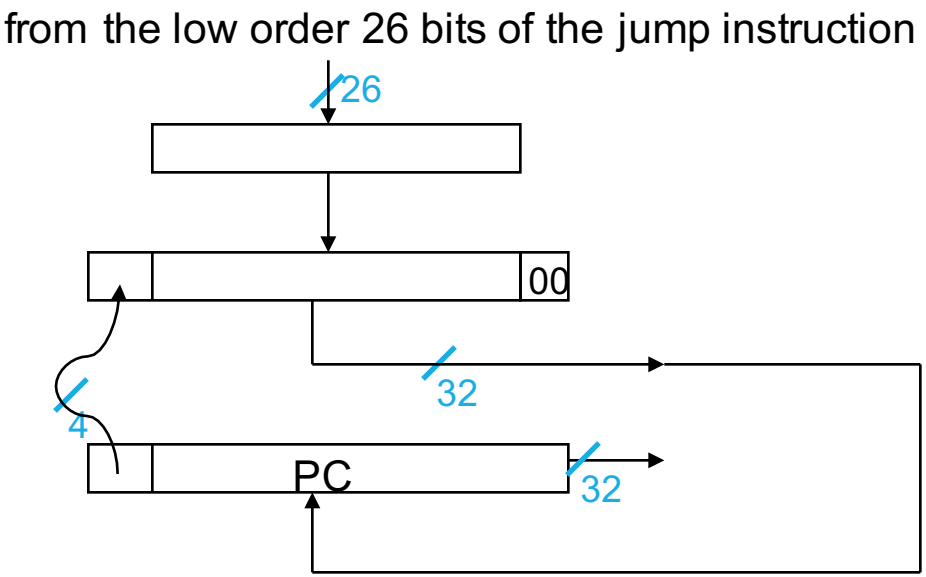
- How is the branch destination address specified?

Other Control Flow Instructions

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

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

- Instruction Format (J Format):



EX-2: Branching Far Away

- What if the branch destination is further away than can be captured in 16 bits?

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