C335
Computer Structures

MIPS Instructions (Part #4)

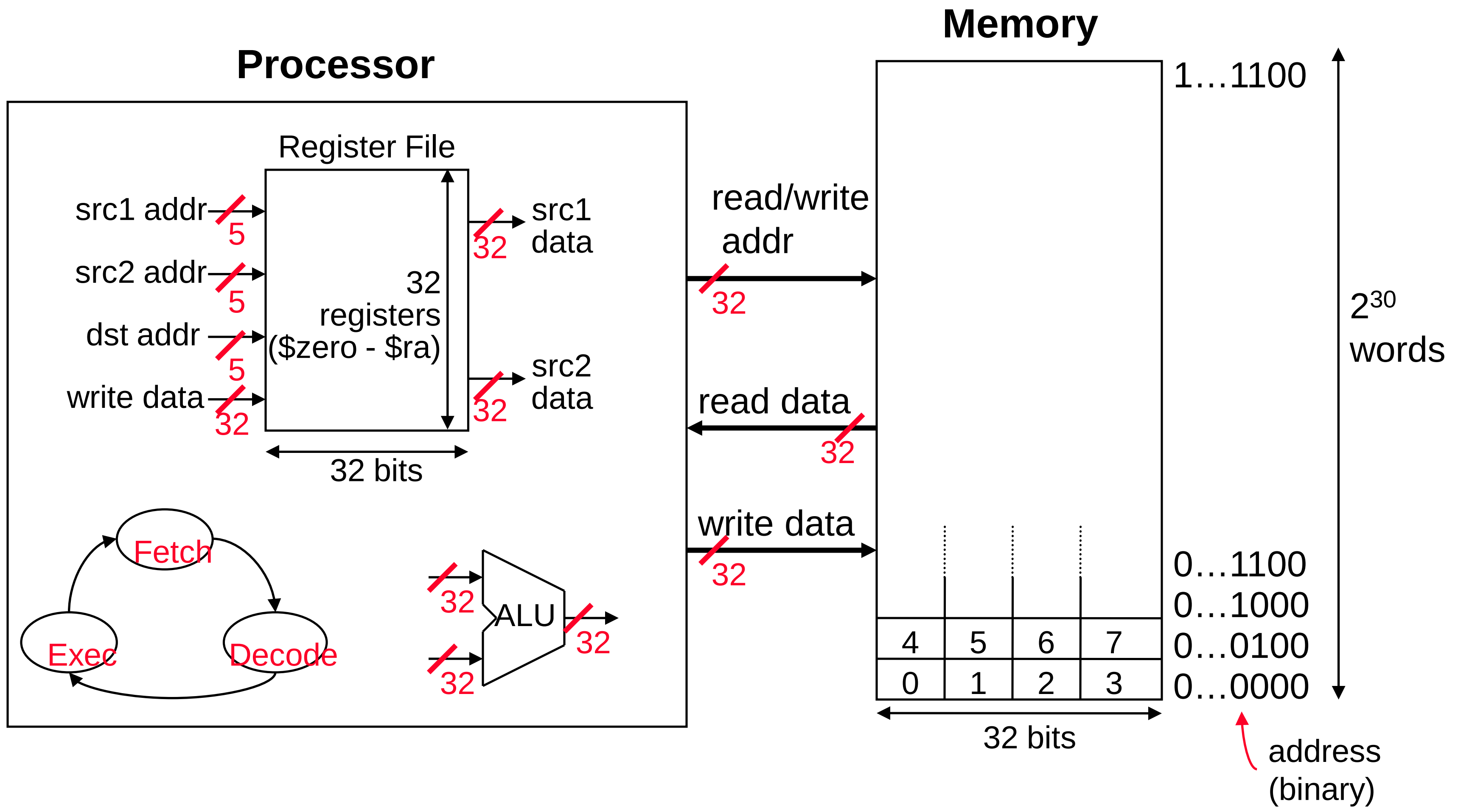
Dr. Liqiang Zhang

Department of Computer and Information Sciences

Adapted from Morgan Kaufmann and others

Review: MIPS Organization

- Arithmetic instructions – to/from the register file
- Load/store instructions – from/to memory



Review: MIPS Instructions, so far

Category	Instr	OpCode (hex)	Example	Meaning
Arithmetic (R format)	add	0 & 20	add $s1, $s2, $s3	$s1 = $s2 + $s3
	subtract	0 & 22	sub $s1, $s2, $s3	$s1 = $s2 - $s3
Arithmetic (I format)	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

Instructions for Making Decisions

- Decision making instructions
  - alter the control flow
  - i.e., change the "next" instruction to be executed

- MIPS conditional branch instructions:

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

- Example: `if (i==j) h = i + j;`

```
        bne $s0, $s1, Lbl1
        add $s3, $s0, $s1
Lbl1:   ...
```

Assembling Branches

- Instructions:

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

- Machine Formats:

op	rs	rt	16 bit number

I format

⇩

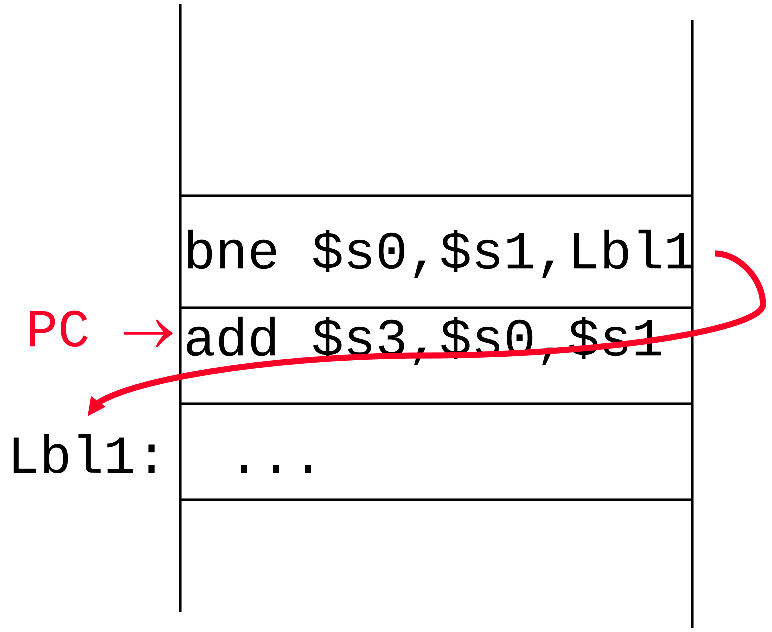
5	16	17	????

4	16	17	????

- How is the branch destination address specified?

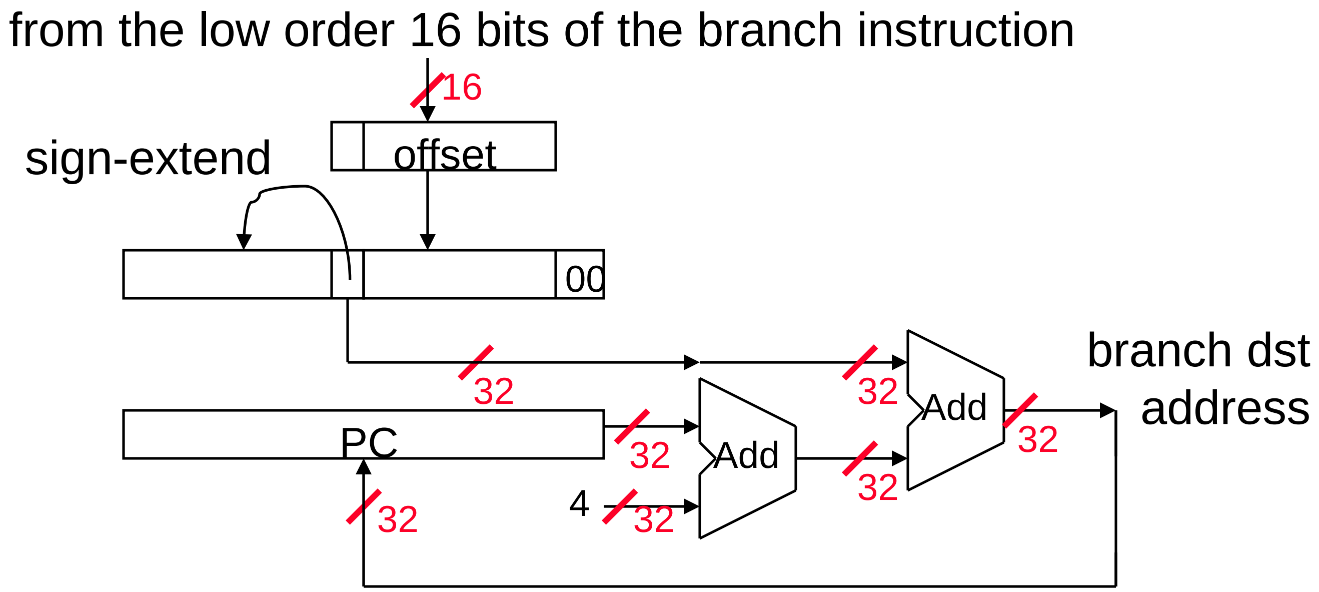
Specifying Branch Destinations

- Could specify the memory address - but that would require a 32 bit field
- Could use a "base" register and add to it the 16-bit offset
  - which register?
    - Instruction Address Register (PC = program counter) - its use is automatically implied by branch
    - 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 $+2^{15}-1$ instr's from the (instruction after the) branch
    - but most branches are local anyway



Disassembling Branch Destinations

- The contents of the updated PC (PC+4) is added to the 16 bit branch offset which is converted into a 32 bit value by
  - concatenating two low-order zeros to make it a word address and then sign-extending those 18 bits
- The result is written into the PC if the branch condition is true - before the next Fetch cycle



Offset Tradeoffs

- Why not just store the byte offset in the low order 16 bits? Then the two low order zeros wouldn't have to be concatenated, it would be less confusing, …

- But that would limit the branch distance to -2^{13} to $+2^{13}-1$ instr's from the (instruction after the) branch
- And concatenating the two zero bits costs us very little in additional hardware and has no impact on the clock cycle time

Assembling Branches Example

- Assembly code

```
          bne $s0, $s1, Lbl1
          add $s3, $s0, $s1
Lbl1:     ...
```

- Machine Format of bne:

op	rs	rt	16 bit offset
5	16	17	0x0001

I format

- Remember
  - After the bne instruction is fetched, the PC is updated so that it is addressing the add instruction (PC = PC + 4).
  - The offset (plus 2 low-order zeros) is sign-extended and added to the (updated) PC

Another Instruction for Changing Flow

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

```
    j   Lbl         #go to Lbl
```

- Example:

```
if (i!=j)
     h=i+j;
else
     h=i-j;
```

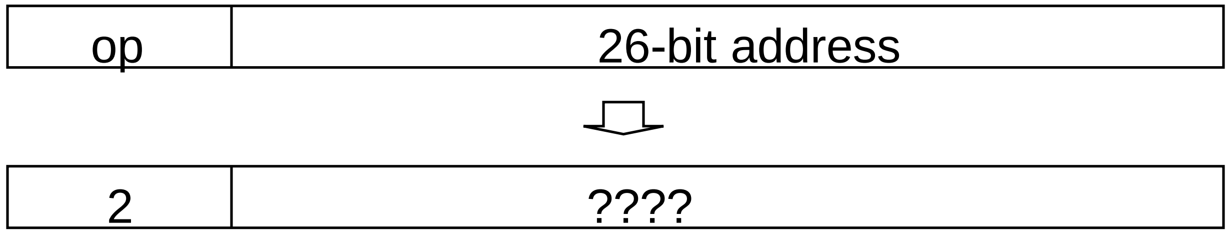
```
        beq  $s0, $s1, Else
        add  $s3, $s0, $s1
        j    Exit
Else:   sub  $s3, $s0, $s1
Exit:   ...
```

Assembling Jumps

- Instruction:

```
    j  Lbl          #go to Lbl
```

- Machine Format:



J format

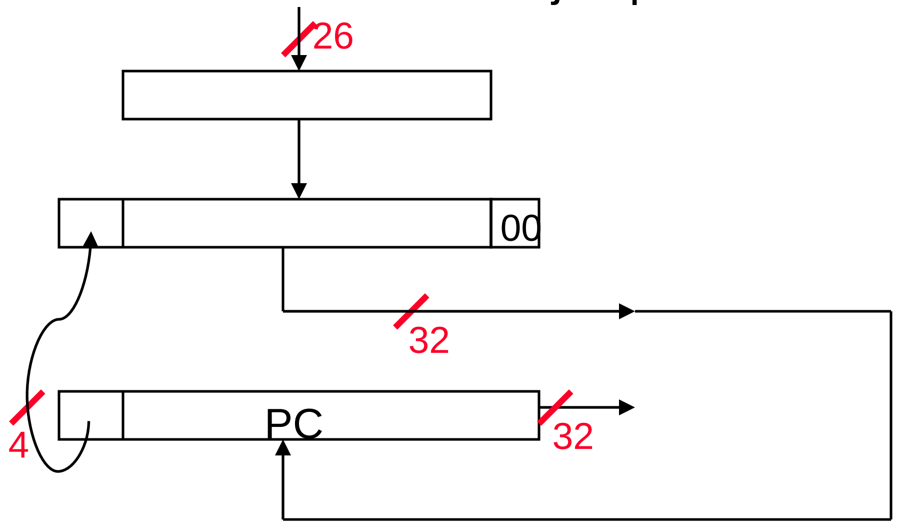
- How is the jump destination address specified?
  - As an absolute address formed by
    - concatenating 00 as the 2 low-order bits to make it a word address
    - concatenating the upper 4 bits of the current PC (now PC+4)

Disassembling Jump Destinations

- The low order 26 bits of the jump instr converted into a 32 bit jump destination address by
  - concatenating two low-order zeros to create an 28 bit (word) address and then concatenating the upper 4 bits of the current PC (now PC+4) to create a 32 bit (word) address

  that is put into the PC prior to the next Fetch cycle

from the low order 26 bits of the jump instruction



Assembling Branches and Jumps

- Assemble the MIPS machine code (in decimal is fine) for the following code sequence. Assume that the addr of the beq instr is $0x00400020_{hex}$

```
         beq  $s0, $s1, Else
         add  $s3, $s0, $s1
         j    Exit
Else:    sub  $s3, $s0, $s1
Exit:    ...
```

```
0x00400020         4     16    17          2
0x00400024         0     16    17    19    0   0x20
0x00400028         2     0000 0100 0 ... 0 0011 00_2
               jmp dst = (0x0) 0x040003 00_2(00_2)
                                 = 0x00400030
0x0040002c         0     16    17    19    0   0x22
0x00400030         ...
```

Branching Far Away

- What if the conditional 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:
```

Compiling While Loops

- Compile the assembly code for the C `while` loop where i is in `$s0`, j is in `$s1`, and k is in `$s2`

```
while (i!=k)
     i=i+j;
```

```
Loop:   beq  $s0, $s2, Exit
        add  $s0, $s0, $s1
        j    Loop
Exit:   . . .
```

A better version:

```
        beq  $s0, $s2, Exit
Loop:   add  $s0, $s0, $s1
        bne  $s0, $s2, Loop
Exit:   . . .
```

More Instructions for Making Decisions

- We have beq, bne, but what about branch-if-less-than?
- New instruction:

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

- Machine format:

op	rs	rt	rd		funct
0	16	17	8	0	42 = 0x2a

R format

Yet More Instructions for Making Decisions

- Since constant operands are popular in comparisons, also have `slti`
- New instruction:

```
slti $t0, $s0, 10        # if $s0 < 10
                         #    then
                         # $t0 = 1
                         #    else
                         # $t0 = 0
```

- Machine format:

op	rs	rt	16 bit number

I format

⇩

a	16	8	0x000a

Other Branch Instructions

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

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

  - less than or equal to `ble $s1, $s2, Lbl`
  - greater than `bgt $s1, $s2, Lbl`
  - great than or equal to `bge $s1, $s2, Lbl`
- As pseudo instructions they are recognized (and expanded) by the assembler
- The assembler needs a reserved register (`$at`)

Another Instruction for Changing Flow

- Most higher level languages have `case` or `switch` statements allowing the code to select one of many alternatives depending on a single value
- Instruction:

```
jr  $t1          #go to address in $t1
```

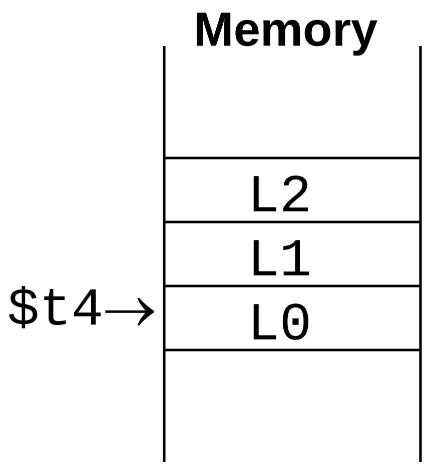
- Machine format:

R format

op	rs				funct
0	9	0	0	0	8 = 0x08

Compiling a Case (Switch) Statement

```
switch (k) {
   case 0:  h=i+j;  break; /*k=0*/
   case 1:  h=i+h;  break; /*k=1*/
   case 2:  h=i-j;  break; /*k=2*/
```



- Assuming three sequential words in memory starting at the address in `$t4` have the addresses of the labels L0, L1, and L2 and k is in `$s2`

```
        add     $t1, $s2, $s2        #$t1 = 2*k
        add     $t1, $t1, $t1        #$t1 = 4*k
        add     $t1, $t1, $t4        #$t1 = addr of JumpT[k]
        lw      $t0, 0($t1)          #$t0 = JumpT[k]
        jr      $t0                  #jump based on $t0
L0:     add     $s3, $s0, $s1        #k=0 so h=i+j
        j       Exit
L1:     add     $s3, $s0, $s3        #k=1 so h=i+h
        j       Exit
L2:     sub     $s3, $s0, $s1        #k=2 so h=i-j
Exit:   . . .
```