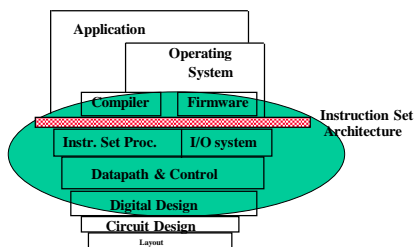
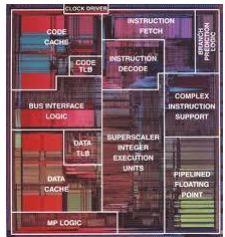




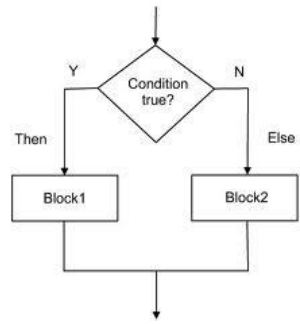
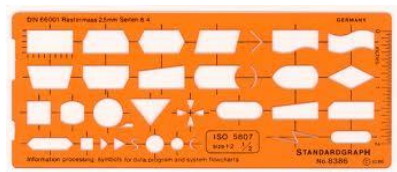
CS/SE 3340

Computer Architecture



Jump & Branch Instructions

Based on slides from Prof. Aviral Shrivastava of ASU



Control Flow

- In the Von Neumann computer architecture the *control flow* of a program is determined by the sequence of executed instructions
- Typically instructions are executed *sequentially*
- Sequential execution can be viewed from a memory level in which instructions are executed in sequential order of address
- What happens to the PC (program counter) of a MIPS processor when an instruction is executed in sequential order?

3

Jump and Branch Instructions

- The modern processors *jump* and *branch* instructions are used to change the default (sequential) flow of control
- Thus, during the execution of a branch instruction the PC is potentially changed to something different than $PC + 4$
- These hardware level branch instructions are needed to support constructs such as **if then else** statements and loops (e.g. **while**, **do-while**, **for**) in high level languages

4

Compiling If Statements

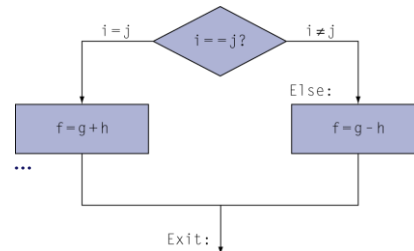
- C code:

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

f, g, ... in $s0, $s1, ...
```

- Compiled MIPS code:

```
    bne $s3, $s4, Else
    add $s0, $s1, $s2
    j    Exit
Else: sub $s0, $s1, $s2
Exit: ...
```



Assembler calculates label addresses

5

MIPS Instructions for Making Decisions

- Decision making instructions

- Change the course of the control flow based on certain condition
- i.e., change the "next" instruction to be executed when a condition is met

- MIPS provides *conditional branch instructions* for this purpose

```
bne $s0, $s1, Label #go to Label if $s0 != $s1
beq $s0, $s1, Label #go to Label if $s0 == $s1
```

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

```
    bne $s0, $s1, Lab1
    add $s3, $s0, $s1
Lab1: ...
```

6

Assembling Branch Instructions

- Instructions:

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

- Machine code format

op	rs	rt	16 bit number	I format
5	16	17	????	
4	16	17	????	

How to calculate the destination address (Label) ?

7

Calculating Branch Destinations

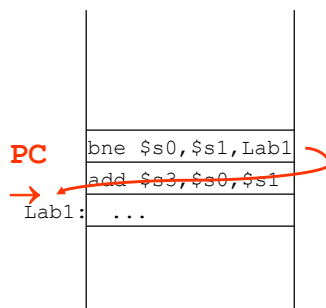
- Could use a register and add to it the 16-bit offset

- which register?

- PC = 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 (principle of locality)

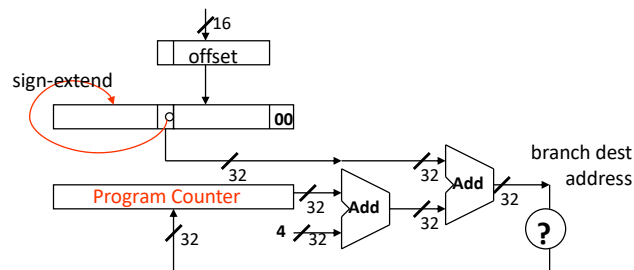


8

Calculating Branch Destinations – cont'd

- The contents of the updated PC (PC+4) is added to the low order 16 bits of the branch instruction which is converted into a 32 bit value by
 - concatenated two low-order zeros to create an 18 bit number
 - sign-extending those 18 bits
- The result is written into the PC if the branch condition is true prior to the next Fetch cycle

from the low order 16 bits of the branch instruction



9

Assembling Branch Example

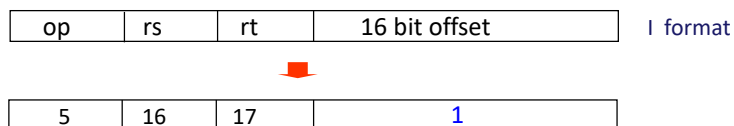
- Assembly code


```

bne $s0, $s1, Lab1
add $s3, $s0, $s1
            
```

Lab1: ...

- Machine code format of **bne**:



- Remember
 - After the **bne** instruction is fetched, the PC is updated to address the add instruction (PC = PC + 4).
 - Two low-order zeros are concatenated to the offset number and that value sign-extended is added to the (updated) PC

10

Another Instruction for Changing Control Flow

- MIPS provides also an *unconditional branch (or jump)* instruction

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

- Example:


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

```
        beq    $s0, $s1, Lab1
        add    $s3, $s0, $s1
        j      Lab2
Lab1:    sub    $s3, $s0, $s1
Lab2:    ...
```

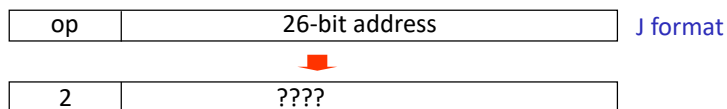
11

Assembling Jump Instructions

- Assembly instruction

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

- Machine code format:



12

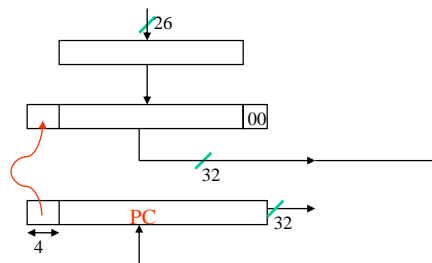
Jump Destination Address

How does the processor calculate the jump destination address ?

As an *absolute* 32-bit address formed by

1. concatenating the upper 4 bits of the current PC (now PC+4) to the 26-bit address and
2. concatenating 00 as the 2 low-order bits

from the low order 26 bits of the jump instruction



13

Assembling Branch and Jump

- Assemble the MIPS machine code for the following code sequence. Assume that the address of the `beq` instruction is `0x00400020` (hex address)

```

                beq    $s0, $s1, Lab1
                add    $s3, $s0, $s1
                j       Lab2
Lab1:          sub    $s3, $s0, $s1
Lab2:          ...

0x00400020      4      16      17      2
0x00400024      0      16      17      19      0      32
0x00400028      2      0000 0100 0000 0000 0000 0011 002

```

`jmp dst = (00002) 0000 0100 0000 0000 0011 002(002)`
`= 0x00400030`

```

0x0040002c      0      16      17      19      0      34
0x00400030      ...

```

14

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:    . . .
```

15

More Instructions for Decision Making

- 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 code format

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

16

More Conditional Operations

- Set result to 1 if a condition is true
 - Otherwise, set to 0
- **slt rd, rs, rt**
 - if (rs < rt) rd = 1; else rd = 0;
- **slti rt, rs, constant**
 - if (rs < constant) rt = 1; else rt = 0;
- Use in combination with **beq, bne**

```
slt $t0, $s1, $s2 # if ($s1 < $s2)
bne $t0, $zero, L  # branch to L
```

17

Other Branch Instructions

- **slt, beq, bne**, and the fixed value of 0 in register **\$zero** can be used to create all relative conditions
 - less than **blt \$s1, \$s2, Label**
 - less than or equal to **ble \$s1, \$s2, Label**
 - greater than **bgt \$s1, \$s2, Label**
 - great than or equal to **bge \$s1, \$s2, Label**

For example:

```
slt $t0, $s1, $s2      # $t0 set to 1 if
bne $t0, $zero, Label  # $s1 < $s2
```

18

Branch Instruction Design

- Why not `blt`, `bge`, etc.. In hardware?
- Hardware for `<`, `≥`, ... slower than `=`, `≠`
 - Combining with branch involves more work per instruction, requiring a slower clock
 - All instructions penalized!
- `beq` and `bne` are the common case
- This is a good design compromise

19

Signed vs. Unsigned

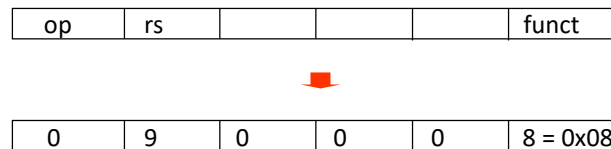
- Signed comparison: `slt`, `slti`
- Unsigned comparison: `sltu`, `sltui`
- Example
 - `$s0 = 1111 1111 1111 1111 1111 1111 1111 1111`
 - `$s1 = 0000 0000 0000 0000 0000 0000 0000 0001`
 - `slt $t0, $s0, $s1 # signed`
 - $-1 < +1 \Rightarrow \$t0 = 1$
 - `sltu $t0, $s0, $s1 # unsigned`
 - $+4,294,967,295 > +1 \Rightarrow \$t0 = 0$

20

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
 - Destination address is determined at run-time!
 - MIPS provides a special instruction to facilitate this
- Instruction:


```
jr $t1    #go to address in $t1 (9)
```
- Machine code format



21

Compiling a 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 JT[k]
	lw	\$t0, 0(\$t1)	#\$t0 = JT[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:	. . .		

Memory

L0

L1

L2

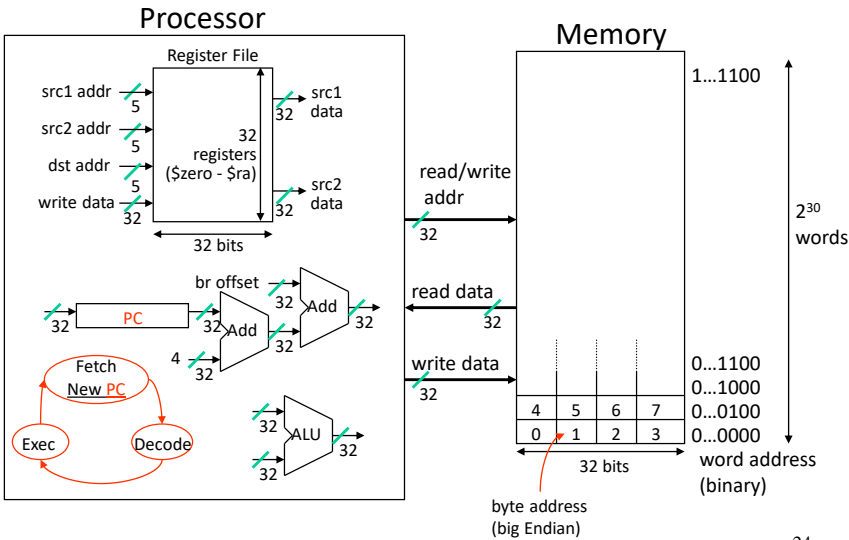
22

MIPS Instructions

Category	Instruction	Op Code	Example	Meaning
Arithmetic (R format)	add	0 and 32	add \$s1, \$s2, \$s3	$\$s1 = \$s2 + \$s3$
	subtract	0 and 34	sub \$s1, \$s2, \$s3	$\$s1 = \$s2 - \$s3$
Data transfer (I format)	load word	35	lw \$s1, 100(\$s2)	$\$s1 = \text{Memory}(\$s2+100)$
	store word	43	sw \$s1, 100(\$s2)	$\text{Memory}(\$s2+100) = \$s1$
	load byte	32	lb \$s1, 101(\$s2)	$\$s1 = \text{Memory}(\$s2+101)$
	store byte	40	sb \$s1, 101(\$s2)	$\text{Memory}(\$s2+101) = \$s1$
Branch (conditional jump)	br on equal	4	beq \$s1, \$s2, L	if ($\$s1 == \$s2$) go to L
	br on not equal	5	bne \$s1, \$s2, L	if ($\$s1 \neq \$s2$) go to L
	set on less than	0 and 42	slt \$s1, \$s2, \$s3	if ($\$s2 < \$s3$) $\$s1=1$ else $\$s1=0$
Unconditional jump	jump	2	j 2500	go to 10000
	jump register	0 and 8	jr \$t1	go to \$t1

23

MIPS ISA Functional View



24