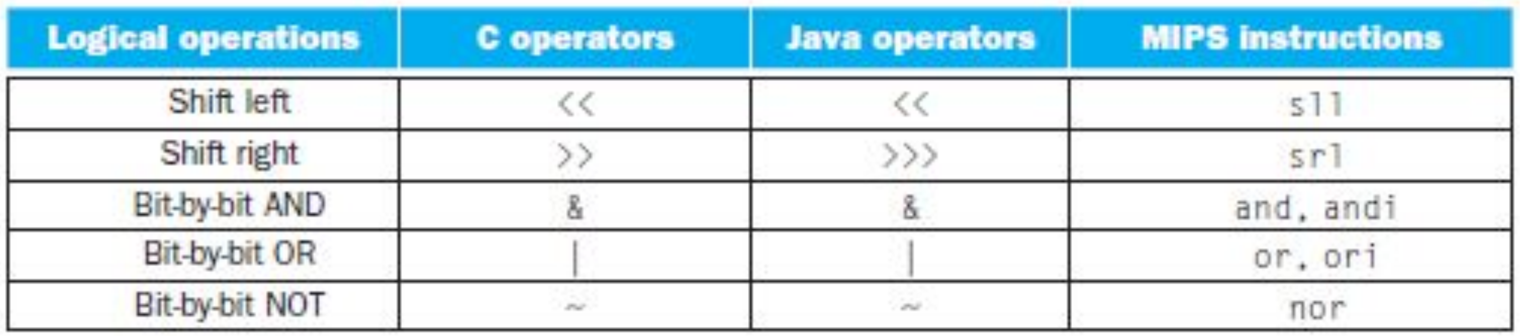
# CSE-413 Computer Architecture
Lecture 5

## Logical Operations

# Introduction



| Logical operations | C operators | Java operators | MIPS instructions |
|---|---|---|---|
| Shift left | << | << | sll |
| Shift right | >> | >>> | srl |
| Bit-by-bit AND | & | & | and, andi |
| Bit-by-bit OR | \| | \| | or, ori |
| Bit-by-bit NOT | ~ | ~ | nor |

## Shifts

The first class of such operations is called shifts. They move all the bits in a word to the left or right, filling the emptied bits with 0s.

For example, if register $s0 contained

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001_{two} = 9_{ten}$$

and the instruction to shift left by 4 was executed, the new value would be:

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_{two} = 144_{ten}$$

## Shifts-Continue

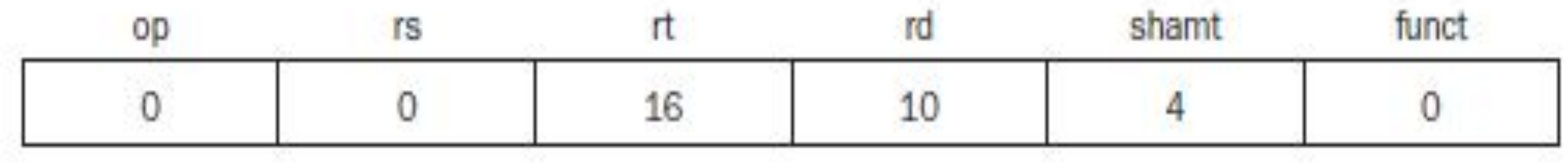
The dual of a shift left is a shift right. The actual name of the two MIPS shift instructions are called *shift left logical (sll)* *and shift right logical (srl).*

sll $t2,$s0,4

Example

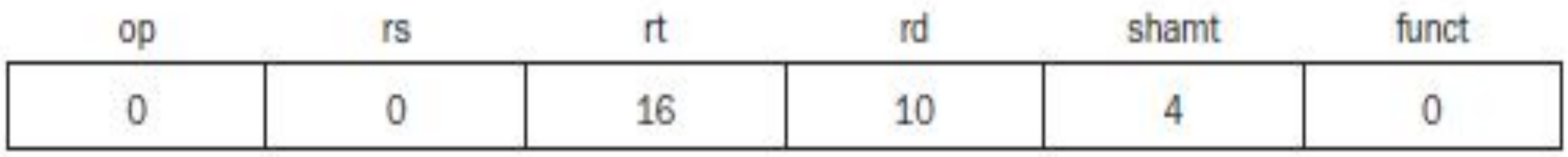
sll $t2,$s0,4

The machine language version of the instruction above is



| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 0 | 0 | 16 | 10 | 4 | 0 |

## Shifts-Continue



| op | rs | rt | rd | shamt | funct |
|---|---|---|---|---|---|
| 0 | 0 | 16 | 10 | 4 | 0 |

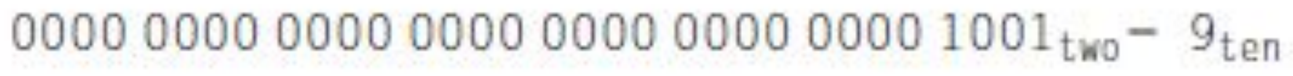
The encoding of sll is 0 in both the op and funct fields, rd contains 10 (register $t2), rt contains 16 (register $s0), and shamt contains 4. The rs field is unused and thus is set to 0.

# Shifts-Continue

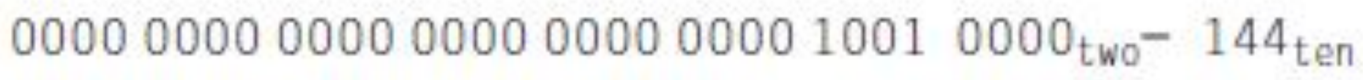
Shift left logical provides a bonus benefit. Shifting left by i bits gives the same result as multiplying by $2^i$, just as shifting a decimal number by i digits is equivalent to multiplying by $10^i$.

For example, the above sll shifts by 4, which gives the same result as multiplying by $2^4$ or 16.

The first bit pattern above represents 9, and 9 × 16 = 144, the value of the second bit pattern.


0000 0000 0000 0000 0000 0000 0000 1001$_{two}$ – 9$_{ten}$

sll $t2,$s0,4


0000 0000 0000 0000 0000 0000 1001 0000$_{two}$ – 144$_{ten}$

## AND operation

AND is a bit-by-bit operation that leaves a 1 in the result only if both bits of the operands are 1.

For example, if register $t2 contains

$$0000\ 0000\ 0000\ 0000\ 0000\ 1101\ 1100\ 0000_{\text{two}}$$

and register $t1 contains

$$0000\ 0000\ 0000\ 0000\ 0011\ 1100\ 0000\ 0000_{\text{two}}$$

then, after executing the MIPS instruction

```
and $t0,$t1,$t2
```

the value of register $t0 would be

$$0000\ 0000\ 0000\ 0000\ 0000\ 1100\ 0000\ 0000_{\text{two}}$$

## AND operation-Continue

- AND can apply a bit pattern to a set of bits to force 0s where there is a 0 in the bit pattern.

- Such a bit pattern in conjunction with AND is traditionally called a *mask, since the mask "conceals" some bits.*

## OR operation

It is a bit-by-bit operation that places a 1 in the result if *either operand bit is a 1.*

For example, if register \$t2 contains

$0000\ 0000\ 0000\ 0000\ 0000\ 1101\ 1100\ 0000_{two}$

and register \$t1 contains

$0000\ 0000\ 0000\ 0000\ 0011\ 1100\ 0000\ 0000_{two}$

the result of the MIPS instruction

```
or $t0,$t1,$t2
```

is this value in register \$t0:

$0000\ 0000\ 0000\ 0000\ 0011\ 1101\ 1100\ 0000_{two}$

## Not Operation

- **NOT takes one operand and** places a 1 in the result if one operand bit is a 0, and vice versa.
- In keeping with the three-operand format, the designers of MIPS decided to include the instruction **NOR (NOT OR) instead of NOT**.
- **If one operand is zero, then it is equivalent to** NOT: *A* NOR 0 = NOT (*A* OR 0) = NOT (*A*).

## Not Operation

If the register \$t1 is 0000 0000 0000 0000 0011 1100 0000 0000$_{two}$ and register \$t3 has the value 0, the result of the MIPS instruction

nor \$t0,\$t1,\$t3

is this value in register \$t0:

1111 1111 1111 1111 1100 0011 1111 1111two

## Instructions for Making Decisions

MIPS assembly language includes two decision-making instructions, similar to an *if* statement with a *go to*. The first instruction is

beq register1, register2, L1

This instruction means go to the statement labeled L1 if the value in register1 equals the value in register2.

The mnemonic beq stands for *branch if equal*.

## Instructions for Making Decisions-Cont.

```
bne register1, register2, L1
```

- It means go to the statement labeled L1 if the value in register1 does *not equal* the value in register2.

- The mnemonic *bne* stands for *branch if not equal.*
- *These* two instructions are traditionally called **conditional branches.**

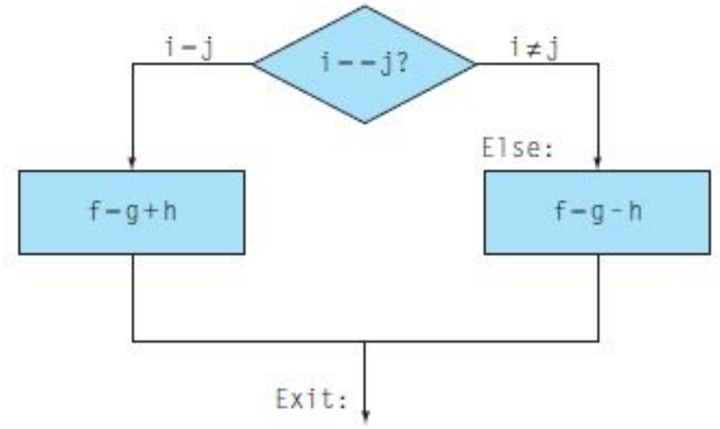
## Example

In the following code segment, f, g, h, i, and j are variables. If the five variables f through j correspond to the five registers $s0 through $s4, what is the compiled MIPS code for this C *if statement?*

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

```
bne $s3,$s4,Else    # go to Else if i ≠ j
add $s0,$s1,$s2     # f = g + h (skipped if i ≠ j)
j Exit              # go to Exit
Else:sub $s0,$s1,$s2 # f = g - h (skipped if i = j)
Exit:
```

# Example-Cont.



This example introduces another kind of branch, often called an unconditional branch.

This instruction says that the processor always follows the branch.

To distinguish between conditional and unconditional branches, the MIPS name for this type of instruction is jump, abbreviated as j.

## Loops

Decisions are important both for choosing between two alternatives—found in *if* statements—and for iterating a computation—found in loops. The same assembly instructions are the building blocks for both cases.

## Example

Here is a traditional loop in C:

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

Assume that i and k correspond to registers $s3 and $s5 and the base of the array save is in $s6. What is the MIPS assembly code corresponding to this C segment?

## Loops

Here is a traditional loop in C:

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

Assume that i and k correspond to registers $s3 and $s5 and the base of the array save is in $s6. What is the MIPS assembly code corresponding to this C segment?

```
Loop: sll $t1,$s3,2 # Temp reg $t1 = i * 4
add $t1,$t1,$s6     # $t1 = address of save[i]
lw $t0,0($t1)       # Temp reg $t0 = save[i]
bne $t0,$s5, Exit   # go to Exit if save[i] ≠ k
addi $s3,$s3,1      # i = i + 1
j Loop              # go to Loop
Exit:
```

## set on less than Instruction

```
slt $t0, $s3, $s4   # $t0 = 1 if $s3 < $s4
```

means that register $t0 is set to 1 if the value in register $s3 is less than the value in register $s4; otherwise, register $t0 is set to 0.

Constant operands are popular in comparisons, so there is an immediate version of the set on less than instruction. To test if register $s2 is less than the constant 10, we can just write

```
slti $t0,$s2,10      # $t0 = 1 if $s2 < 10
```

## Signed versus Unsigned Comparison

- Comparison instructions must deal with the difference between signed and unsigned numbers.

- Sometimes a bit pattern with a 1 in the most significant bit represents a negative number and, of course, is less than any positive number, which must have a 0 in the most significant bit.

- With unsigned integers, on the other hand, a 1 in the most significant bit represents a number that is *larger than* any that begins with a 0.

## Cont.

- MIPS offers two versions of the set on less than comparison to handle these alternatives.

- *Set on less than (slt) and set on less than immediate (slti) work with* signed integers.

- Unsigned integers are compared using *set on less than unsigned* (sltu) and *set on less than immediate unsigned (sltiu).*

## Example

Suppose register $s0 has the binary number

$$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111_{two}$$

and that register $s1 has the binary number

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two}$$

What are the values of registers $t0 and $t1 after these two instructions?

```
slt $t0, $s0, $s1 # signed comparison
sltu $t1, $s0, $s1 # unsigned comparison
```

## Solution

- The value in register \$s0 represents $-1_{ten}$ if it is an integer and $4{,}294{,}967{,}295_{ten}$ if it is an unsigned integer.

- The value in register \$s1 represents $1_{ten}$ in either case.

- Then register \$t0 has the value 1, since $-1_{ten} < 1_{ten}$, and register \$t1 has the value 0, since $4{,}294{,}967{,}295_{ten} > 1_{ten}$.