LAB 03: Integer Arithmetic

Saleh AlSaleh
salehs@kfupm.edu.sa

King Fahd University of Petroleum and Minerals
College of Computing and Mathematics
Computer Engineering Department

COE301: Computer Architecture
Term 222

Agenda

1. Overflow
2. Logical Bitwise Instructions
3. Shift Instructions
4. Pseudo Instructions
5. Live Examples
6. Tasks

Overflow

- Max positive integer number represented in 4-bit:

Overflow

- Max positive integer number represented in 4-bit: $(+7)_{10} = (0111)_2$

Overflow

- Max positive integer number represented in 4-bit: $(+7)_{10} = (0111)_2$
- Min negative integer number represented in 4-bit:

Overflow

- Max positive integer number represented in 4-bit: $(+7)_{10} = (0111)_2$
- Min negative integer number represented in 4-bit: $(-8)_{10} = (1000)_2$

Overflow

- Max positive integer number represented in 4-bit: $(+7)_{10} = (0111)_2$
- Min negative integer number represented in 4-bit: $(-8)_{10} = (1000)_2$
- Max positive integer number represented in 32-bit:

Overflow

- Max positive integer number represented in 4-bit: $(+7)_{10} = (0111)_2$
- Min negative integer number represented in 4-bit: $(-8)_{10} = (1000)_2$
- Max positive integer number represented in 32-bit: $(0x7FFFFFFF)_{16}$

Overflow

- Max positive integer number represented in 4-bit: $(+7)_{10} = (0111)_2$
- Min negative integer number represented in 4-bit: $(-8)_{10} = (1000)_2$
- Max positive integer number represented in 32-bit: $(0x7FFFFFFF)_{16}$
- Min negative integer number represented in 32-bit:

Overflow

- Max positive integer number represented in 4-bit: $(+7)_{10} = (0111)_2$
- Min negative integer number represented in 4-bit: $(-8)_{10} = (1000)_2$
- Max positive integer number represented in 32-bit: $(0x7FFFFFFF)_{16}$
- Min negative integer number represented in 32-bit: $(0x80000000)_{16}$

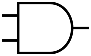
Overflow

- Max positive integer number represented in 4-bit: $(+7)_{10} = (0111)_2$
- Min negative integer number represented in 4-bit: $(-8)_{10} = (1000)_2$
- Max positive integer number represented in 32-bit: $(0x7FFFFFFF)_{16}$
- Min negative integer number represented in 32-bit: $(0x80000000)_{16}$
- add/sub causes/raises arithmetic exception in the case of overflow and result is not written.

Overflow

- Max positive integer number represented in 4-bit: $(+7)_{10} = (0111)_2$
- Min negative integer number represented in 4-bit: $(-8)_{10} = (1000)_2$
- Max positive integer number represented in 32-bit: $(0x7FFFFFFF)_{16}$
- Min negative integer number represented in 32-bit: $(0x80000000)_{16}$
- add/sub causes/raises arithmetic exception in the case of overflow and result is not written.
- addu/subu ignores overflow and writes result to destination register

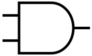
Logical Bitwise Instructions



AND

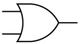
	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$A \& B$	0	1	0	0

Logical Bitwise Instructions



AND

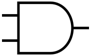
	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$A \& B$	0	1	0	0



OR

	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$A \vert B$	1	1	0	1

Logical Bitwise Instructions

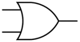


AND

	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$A\&B$	0	1	0	0

XOR

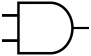
	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$A \wedge B$	1	0	0	1



OR

	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$A \vert B$	1	1	0	1

Logical Bitwise Instructions

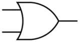


AND

	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$A\&B$	0	1	0	0

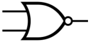
XOR

	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$A \wedge B$	1	0	0	1



OR

	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$A\|B$	1	1	0	1



NOR

	b_3	b_2	b_1	b_0
A	0	1	0	1
B	1	1	0	0
$\overline{(A\|B)}$	0	0	1	0

Shift Instructions (Left Shift)

$(0010)_2$

Shift Instructions (Left Shift)

$(0010)_2$ $\xrightarrow[\text{and append 0 in the LSB}]{\text{Shift every bit to the left by 1}}$

2

Shift Instructions (Left Shift)

$$\underset{2}{(0010)_2} \xrightarrow[\text{and append 0 in the LSB}]{\text{Shift every bit to the left by 1}} \underset{4}{(010\textcolor{red}{0})_2}$$

Shift Instructions (Left Shift)

$(0010)_2$ 2 $\xrightarrow[\text{and append 0 in the LSB}]{\text{Shift every bit to the left by 1}}$ $(0100)_2$ 4

$(0100)_2$ 4 $\xrightarrow[\text{and append 0 in the LSB}]{\text{Shift every bit to the left by 1}}$

Shift Instructions (Left Shift)

$$\underset{2}{(0010)_2} \xrightarrow[\text{and append 0 in the LSB}]{\text{Shift every bit to the left by 1}} \underset{4}{(010\textcolor{red}{0})_2}$$

$$\underset{4}{(0100)_2} \xrightarrow[\text{and append 0 in the LSB}]{\text{Shift every bit to the left by 1}} \underset{8}{(100\textcolor{red}{0})_2}$$

Shift Instructions (Left Shift)

$(0010)_2$ $\xrightarrow[\text{and append 0 in the LSB}]{\text{Shift every bit to the left by 1}}$ $(0100)_2$
2 → 4

$(0100)_2$ $\xrightarrow[\text{and append 0 in the LSB}]{\text{Shift every bit to the left by 1}}$ $(1000)_2$
4 → 8

- This is called Shift Left Logical (sll).
- Every single shift left logical is equivalent to multiplying by 2.
- MIPS instruction: sll $dst, $src, shift_amount.
  e.g. sll $t0, $t1, 3
  equivalent to multiplying $t1 by $2^3 = 8$

Shift Instructions (Logical Right Shift)

$(1010)_2$

Shift Instructions (Logical Right Shift)

$(1010)_2$
10

$\xrightarrow[\text{and append 0 in the MSB}]{\text{Shift every bit to the right by 1}}$

Shift Instructions (Logical Right Shift)

$$\begin{array}{c} (1010)_2 \\ 10 \end{array} \xrightarrow[\text{and append 0 in the MSB}]{\text{Shift every bit to the right by 1}} \begin{array}{c} ({\color{red}0}101)_2 \\ 5 \end{array}$$

Shift Instructions (Logical Right Shift)

$(1010)_2$ $\xrightarrow[\text{and append 0 in the MSB}]{\text{Shift every bit to the right by 1}}$ $(0101)_2$

10 5

$(0101)_2$ $\xrightarrow[\text{and append 0 in the MSB}]{\text{Shift every bit to the right by 1}}$

5

Shift Instructions (Logical Right Shift)

$$\underset{10}{(1010)_2} \xrightarrow[\text{and append 0 in the MSB}]{\text{Shift every bit to the right by 1}} \underset{5}{({\color{red}0}101)_2}$$

$$\underset{5}{(0101)_2} \xrightarrow[\text{and append 0 in the MSB}]{\text{Shift every bit to the right by 1}} \underset{2}{({\color{red}0}010)_2}$$

Shift Instructions (Logical Right Shift)

$(1010)_2$ $\xrightarrow[\text{and append 0 in the MSB}]{\text{Shift every bit to the right by 1}}$ $(0101)_2$

10 → 5

$(0101)_2$ $\xrightarrow[\text{and append 0 in the MSB}]{\text{Shift every bit to the right by 1}}$ $(0010)_2$

5 → 2

- This is called Shift Right Logical (srl).
- Every single shift right logical is equivalent to dividing by 2 (with floor) .
- MIPS instruction: srl $dst, $src, shift_amount.
  e.g. srl $t0, $t1, 3
  equivalent to dividing (with floor) $t1 by $2^3 = 8$

Shift Instructions (Arithmetic Right Shift)

$(1010)_2$

Shift Instructions (Arithmetic Right Shift)

$(1010)_2$ $\xrightarrow[\text{and duplicate the sign bit}]{\text{Shift every bit to the right by 1}}$

-6

Shift Instructions (Arithmetic Right Shift)

$$\underset{-6}{(1010)_2} \xrightarrow[\text{and duplicate the sign bit}]{\text{Shift every bit to the right by 1}} \underset{-3}{(\textcolor{red}{1}101)_2}$$

Shift Instructions (Arithmetic Right Shift)

$(1010)_2$ -6 $\xrightarrow[\text{and duplicate the sign bit}]{\text{Shift every bit to the right by 1}}$ $(1101)_2$ -3

$(1101)_2$ -3 $\xrightarrow[\text{and duplicate the sign bit}]{\text{Shift every bit to the right by 1}}$

Shift Instructions (Arithmetic Right Shift)

$(1010)_2$ $\xrightarrow[\text{and duplicate the sign bit}]{\text{Shift every bit to the right by 1}}$ $(1101)_2$

-6 → -3

$(1101)_2$ $\xrightarrow[\text{and duplicate the sign bit}]{\text{Shift every bit to the right by 1}}$ $(1110)_2$

-3 → -2

Shift Instructions (Arithmetic Right Shift)

$(1010)_2$ $\xrightarrow[\text{and duplicate the sign bit}]{\text{Shift every bit to the right by 1}}$ $(1101)_2$
-6 → -3

$(1101)_2$ $\xrightarrow[\text{and duplicate the sign bit}]{\text{Shift every bit to the right by 1}}$ $(1110)_2$
-3 → -2

- This is called Shift Right Arithmetic (sra).
- Every single shift right arithmetic is equivalent to dividing by 2 (with floor) for **signed numbers**.
- MIPS instruction: sra $dst, $src, shift_amount.
  e.g. sra $t0, $t1, 3
  equivalent to dividing (with floor) $t1 as a signed number by $2^3 = 8$

Pseudo Instructions

- Maps to one or more basic simple assembly instruction(s).

Pseudo Instructions

- Maps to one or more basic simple assembly instruction(s).
- They ease the programmer's tasks in writing applications.

Pseudo Instructions

- Maps to one or more basic simple assembly instruction(s).
- They ease the programmer's tasks in writing applications.
- Common pseudo instructions: li, la, abs.

Pseudo Instructions

- Maps to one or more basic simple assembly instruction(s).
- They ease the programmer's tasks in writing applications.
- Common pseudo instructions: li, la, abs.
  - li \$t0, 0xABCD $\Rightarrow$ addi \$t0, \$0, 0xABCD

Pseudo Instructions

- Maps to one or more basic simple assembly instruction(s).
- They ease the programmer's tasks in writing applications.
- Common pseudo instructions: li, la, abs.
  - li $t0, 0xABCD ⇒ addi $t0, $0, 0xABCD
  - li $t0, 0x89ABCDEF ⇒ lui $at, 0x89AB
    ori $t0, $at, 0xCDEF

Pseudo Instructions

- Maps to one or more basic simple assembly instruction(s).
- They ease the programmer's tasks in writing applications.
- Common pseudo instructions: li, la, abs.
  - li $t0, 0xABCD ⇒ addi $t0, $0, 0xABCD
  - li $t0, 0x89ABCDEF ⇒ lui $at, 0x89AB
    ori $t0, $at, 0xCDEF

Load Upper 16 bit	Clear Lower 16 bit	
0x89AB	0x0000	$at
0x89AB	0xCDEF	$t0
Keep Upper 16 bit	OR Lower 16 bit with immediate value	

Live Examples

Task #1

Write a MIPS program where you ask the user to enter a **signed integer x**. Then, calculate and print the value of **y** based on the following equation.

$$y = 53.125x$$

Sample Run 1

```
Enter x: 8
y = 425
```

Sample Run 2

```
Enter x: -16
y = -850
```

Task #2

Write a MIPS program where you prompt the user for an integer **a**. Then, set bit 11 and 17. Finally, display the value of that integer after modification.

Sample Run 1

```
Enter a: 465
Result = 133585
```

Sample Run 2

```
Enter a: 1023
Result = 134143
```