Lab 4

Arithmetic and Logical Operations

Goals

- Know how to use arithmetic, logical and shift instructions.
- Understand what overflow is and how to dectect overflow in arithmetic operations.

Preparation

Before doing this lab, you should review your lectures and textbooks in computer architecture.

Inlab Assignments

Sample Program 1

Overflow occurs when the result of an arithmetic operation cannot be represented within a particular number of bits (e.g., a 32 bits).

- Overflow in addition occurs when adding two positive numbers results in a negative sum, or vice versa.
- Overflow in substraction occurs when subtracting a negative number from a positive number results in a negative difference, or when subtracting a positive number from a negative number results in a positive difference.

The following assembly program detects overflow in addition based on a rule that is "*overflow in addition occurs when the sum of two non-negative (or negative) numbers is less (or greater) than eitheir operand"*.

```
#Laboratory Exercise 4, Sample Code 1
.text
start:
      li    $t0,0             # No overflow is set as default status
      addu  $s3,$s1,$s2       # s3 = s1 + s2
      xor   $t1,$s1,$s2       # Check if $s1 and $s2 have the same
                              # sign?
      bltz  $t1,EXIT          # If not, exit
      slt   $t2,$s3,$s1
      bltz  $s1,NEGATIVE      # Check if $s1 and $s2 is negative?
      beq   $t2,$zero,EXIT    # s1 and $s2 are positive
                              # If $s3 > $s1 then the result does not
                              # overflow
      j     OVERFLOW
NEGATIVE:
      bne   $t2,$zero,EXIT    # s1 and $s2 are negative
                              # If $s3 < $s1 then the result
                              # does not overflow
OVERFLOW:
      li    $t0,1             # The result overflows
EXIT:
```

- Create a new project to execute the Sample Program 1 on MARS.
- The two operands of addtion are stored in $s1 and $s2, the sum is stored in $s3. If overflow occurs, $t0 = 1; otherwise, $t0 = 0.
- Initialize the two operands of addition ($s1, $s2).
- Execute the program and observe any change in memory and registers.

Sample Program 2

The following assembly program shows how logical instructions can be used to extract information from a register. Depending on the mask, more than one bit can be extracted.

```
#Laboratory Exercise 4, Sample Code 2
.text
      li    $s0, 0x0563       # Load the test value for these function
      andi  $t0, $s0, 0xff    # Extract the LSB of $s0
      andi  $t1, $s0, 0x0400  # Extract bit 10 of $s0
```

- Create a new project to execute the Sample Program 2 on MARS.
- Execute the program and observe any change in memory and registers.

Sample Program 3

The following assembly program shows how a shift instruction is used to caculate a power of 2.

```
#Laboratory Exercise 4, Sample Code 3
.text
      li    $s0,1             # s0=1
      sll   $s1,$s0,2         # s1=s0*4
```

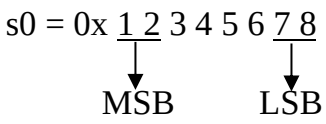
- Create a new project to execute the Sample Program 3 on MARS.
- Execute the program and observe any change in memory and registers.

Assignment 1

Write an assembly program to do the following tasks:

- Assign a 32-bit number to $s0
- Extract the MSB of $s0
- Clear the LSB of $s0
- Set the LSB of $s0 (all bits are set to 1)
- Clear register $s0 ($s0=0, must use logical instructions)
- Exchange the MSB of $s0 with the LSB

MSB: *Most Significant Byte*
LSB: *Least Significant Byte*



Assignment 2

Convert the following pseudo instructions into the real instructions. Explain your answer.

a. `abs    $s0, s1`
`s0 <= |$s1|`

b.
```
move    $s0, s1
      s0 <= $s1
```
c.
```
not     $s0
      s0 <= bit invert (s0)
```
d.
```
ble    $s1, s2, L
      if (s1 <= $s2)
          j L
```

Assignment 3

Write an assembly program to dectect overflow in addition based on the following rule:
"*When we add two numbers that have the same sign, if the sum doesn't have the same sign as the two numbers, overflow will occur".*

Assignment 4

A number of $ABCDE_{10}$ is represented 17 bits. Assume that the decimal number is stored in $s0 from the n^{th} bit to the m^{th} bit ($n > 0$, $m < 31$) and the other bits in $s0 are 0s. Write an assembly program to compute the product of $ABCDE_{10}$ and 32_{10}. The product is stored in $s0.

Example: