Lab# 4 BIT MANIPULATION INSTRUCTIONS

Instructor: I Putu Danu Raharja.

Objectives:
Learn to use MIPS bit manipulation instructions in assembly language programs.

Method:

Translate an algorithm from pseudo-code into assembly language.

Preparation:

Read the chapter 2 of lecture textbook.

4.1 INTRODUCTION

Every computer's architecture needs some bit manipulation instructions. At a minimum, it could provide a NAND operation, since all other logical functions can be derived from NAND operation.

These logical operations are semantically different to what is known as in most of high level programming language. The difference lies down at the fact that bitwise logical operations are performed at bit-by-bit basis.

4.2 BITWISE LOGICAL INSTRUCTIONS

Instructions	Description
`and` rd, rs, rt	rd = rs & rt
`andi` rt, rs, immediate	rt = rs & immediate
`or` rd, rs, rt	rd = rs | rt
`ori` rt, rs, immediate	rd = rs | immediate
`nor` rd, rs, rt	rd = ! (rs | rt)
`xor` rd, rs, rt	To do a bitwise logical Exclusive OR.
`xori` rt, rs, immediate	

The main usage of bitwise logical instructions are: ***to set, to clear, to invert***, and to ***isolate*** some selected bits in the destination operand. To do this, a source bit pattern known as a mask is constructed. The Mask bits are chosen based on the following properties of AND, OR, and XOR with Z represents a bit (either 0 or 1):

AND	OR	XOR
Z AND 0 = 0	Z OR 0 = Z	Z XOR 0 = Z
Z AND 1 = Z	Z OR 1 = 1	Z XOR 1 = ~Z

The AND instruction can be used to CLEAR specific destination bits while preserving the others. A zero mask bit clears the corresponding destination bit; a one mask bit preserves the corresponding destination bit.

The OR instruction can be used to SET specific destination bits while preserving the others. A one mask bit sets the corresponding destination bit; a zero mask bit preserves the corresponding destination bit.

The XOR instruction can be used to INVERT specific destination bits while preserving the others. A one mask bit inverts the corresponding destination bit; a zero mask bit preserves the corresponding destination bit.

A. Example 1:

The following code fragment will clear bit 2, 4, 6, and 7 of the register $t2:

```
addi  $t0, $zero, 0xFF2B
andi  $t2, $t2, $t0
```

B. Example 2:

The following code fragment will set bit 7, 6, 5, 3 and 0 of the register $t2 using OR operation:

```
ori   $t2, $t2, 0x00E9
```

C. Example 3:

The following code fragment will toggle bit 7, 2, and 0 of the register $t2 using XOR operation:

```
xori  $t2, $t2, 0x85
```

4.3 SHIFT INSTRUCTIONS

Instructions	Description
`sll` rd, rs, sa	rd = rs << sa (Shift Left Logical)
`sllv` rd, rt, rs	rd = rt << rs (To left-shift a word by a variable number of bits)
`sra` rd, rs, sa	The contents of the low-order 32-bit word of ***rt*** are shifted right, duplicating the sign-bit (bit 31) in the emptied bits; the word result is placed in ***rd***. The bit-shift amount is specified by ***sa***.
`srav` rd, rt, rs	rd = rt >> rs (Arithmetic)

Instructions	Description
`srl` rd, rs, sa	rd = rs >> sa (Shift Right Logical)
`srlv` rd, rt, rs	rd = rt >> rs

Logical Shift instructions are useful mainly in these situations:

1. To manipulate bits;

2. To multiply and divide unsigned numbers by a power of 2.

A. Example 1

The following code fragment will multiply the content of register $t0 with 80:

```
sll   $t1, $t0, 4        # *16
sll   $t0, $t0, 6        # *64
addu  $t0, $t0, $t1
```

B. Example 2

Explain what the following code fragment will do?

```
addu  $t1,$t0, $zero
sll   $t0,$t0,4
srl   $t1,$t1,4
or    $t1,$t1,$t0
```

4.4 EXERCISES

1. Write a MIPS assembly language program that converts all lowercase letters of a string to uppercase ones.

2. Write a MIPS assembly language program that displays the binary string of the content of register $t0.