Shift Instructions and Logic Instructions

It often happens that a bit pattern must be shifted several positions left or right. The instructions that do this are the shift instruction.

sll

A **shift left logical** of one position moves each bit to the left by one. The low-order bit (the right-most bit) is replaced by a zero bit and the high-order bit (the left-most bit) is discarded.

Shifting by two positions is the same as performing a one-position shift two times. Shifting by zero positions leaves the pattern unchanged. Shifting an N-bit pattern left by N or more positions changes all of the bits to zero.

The picture shows the operation performed on eight bits. The original pattern is 1010 0111. The resulting pattern is 0100 1110.

The MIPS processor always performs the operation on a 32-bit register and puts the result in a 32-bit register.

```
sll  d,s,shft      # $d gets the bits in $s
                   # shifted left logical
                   # by shft positions,
                   # where  0 ≤ shft < 32
```

The ALU (arithmetic/logic unit) which does the operation pays no attention to what the bits mean. If the bits represent an unsigned integer, then a left shift is equivalent to multiplying the integer by two.

```
## Program to logical shift left a pattern

    .text
    .globl  main

main:
    ori     $8, $0, 0x6F      # put bit pattern into register $8
    sll     $9, $8, 2         # shift left logical by two

## End of file
```

srl

MIPS also has a **shift right logical** instruction. It moves bits to the right by a number of positions less than 32. The high-order bit gets zeros and the low-order bits are discarded.

If the bit pattern is regarded as an unsigned integer, or a positive two's comp. integer, then a right shift of one bit position performs an integer divide by two. A right shift by N positions performs an integer divide by 2^N.

```
srl  d,s,shft      # $d gets the logical  right shift of $s by shft positions. shft is a 5-bit
                      # integer, 0 ≤ shft < 32
```

OR

MIPS has many instructions that use two registers as operands and that put the result in a register. All the action takes place inside the processor. The data comes from registers, the ALU performs the operation, and the result is written to a register. All this can be done quickly.

The assembly language must specify four things: (1) the operation, (2) the first operand register, (3) the second operand register, and (4) the destination register. Of course, the machine language must encode the same four things in bit patterns.

Here is the OR instruction:

```
or  d,s,t        # $d gets the bitwise OR between $s with $t.
```

AND

MIPS has an AND instruction:

```
and  d,s,t    # $d gets the bitwise AND between $s with $t.
```

Recall that the result of AND is one only when both operand bits are one

XOR

```
   xor  d,s,t      # $d gets bitwise XOR between $s with
   $t.
```

Recall that the result of XOR is one when only one operand bit is one

NOR

There is a bitwise NOR instruction. There is no immediate operand NOR instruction. NOR is equivalent to performing the OR operation, then complementing the bits (change 0 to 1 and 1 to 0). Here is the assembly language for it:

```
    nor  d,s,t    # $d gets bitwise NOR between $s with $t.
```

Summary

AND	OR	XOR	NOR
`and d,s,t`	`or d,s,t`	`xor d,s,t`	`nor d,s,t`
`$d <— $s and $t`	`$d <— $s or $t`	`$d <— $s xor $t`	`$d <— $s nor $t`

NOT as NOR with `$0`

The NOT operation is done by using the NOR instruction with `$0` as one of the operands:

```
    nor  d,s,$0      # $d <— bitwise NOT of $s.
```

No-Op

Register `$0` always contains a 32-bit zero (have you heard this before?) so shifting it left by zero positions and attempting to put the result back in `$0` does nothing. Any instruction that attempts to alter `$0` does nothing, but this instruction is the preferred way of doing nothing.

A machine instruction that does nothing is called (in official computer science jargon) a no-op. The *no operation* instruction is surprisingly useful, especially for MIPS.

<u>Lab Exercises-3</u>

With the help of Shift Instructions in MIPS ISA, perform the following:

a. Using shift operation get 28 using with input 5 and 2. (Hint: sll)

b. Using arithmetic operation get 1 using 5 and 2. (Hint: srl)

c. Find the cost of 16 apples if once costs 8 rupees, without using any arithmetic operation?

d. Find the quotient when 128 is divided by 8, without using arithmetic operation?

5. Solve the linear equation using MIPS code without using mult and div. 2a+b = 4, a+4b = 9 (Note : No repeated addition or subtraction)

6. Write a MIPS assembly program, when given the value of x and y in $s0 and $ s1, it should calculate the expression, x^3+6x^2y+12xy^2+8y^3.

[Hint: Achieve this MIPS Program in minimum lines of code. And you may avoid the overflow conditions]