Laboratory 3

Title of the Laboratory Exercise: Logical operations

1. Introduction and Purpose of Experiment

   Students will be able to perform all logical operations using assembly instructions.

2. Aim and Objectives

   Aim

   To develop assembly language program to perform all logical operations

   Objectives

   At the end of this lab, the student will be able to

   - Identify the appropriate assembly language instruction for the given logical operations
   - Perform all logical operations using assembly language instructions
   - Get familiar with assembly language program by developing simple programs

3. Experimental Procedure

   1. Write algorithm to solve the given problem
   2. Translate the algorithm to assembly language code
   3. Run the assembly code in GNU assembler
   4. Create a laboratory report documenting the work

4. Questions:

   1. Consider the following source code fragment

      Int a,b,c,d;

      a= (b AND c) XOR d;

      a=(b XOR c) OR d;

      Assume that *b, c, d* are in registers. Develop an assembly language program to perform this assignment statements. Assume that *b, c* are in registers and *d* in memory. Develop an assembly language program to perform this assignment statements.

2. Consider the following source code fragment

   Int a,b,c,d;

   *A = (b*c) / d;*

   Perform multiplication and division by shift operations

5. Calculations/Computations/Algorithms

```
# Logical Operations and Arithemetic Shifting
.section .data

.section .bss

.section .text

.globl _start

# function for system exit code
_ret:
    movq    $60, %rax               # sys_exit
    movq    $0, %rdi                # exit code
    syscall

# driver function
_start:
    # AND and XOR operations
    movl $10, %ebx   # b = 1010
    movl $2, %ecx    # c = 0010
    movl $3, %edx    # d = 0011
    andl %ebx, %ecx # b AND c
    movl %ecx, %eax # move the result to a, a = b AND c
    xorl %edx, %eax # a = (b AND c) XOR d, result is 1

    # XOR and OR operation
    movl $10, %ebx   # b = 1010
    movl $2, %ecx    # c = 0010
    movl $3, %edx    # d = 0011
    xorl %ebx, %ecx # b XOR c
    movl %ecx, %eax # move the result to a, a = b XOR c
    orl  %edx, %eax # a = (b XOR c) OR d, result is 11


    # Right and Left shifting operations
    # b = 64, c = 128, d = 4
```

```
# perform a = (b * c) / 4
movl $64, %ebx
sall $7, %ebx
sarl $2, %ebx    # the result is 2048

syscall
call _ret           # exit
```

6. Presentation of Results

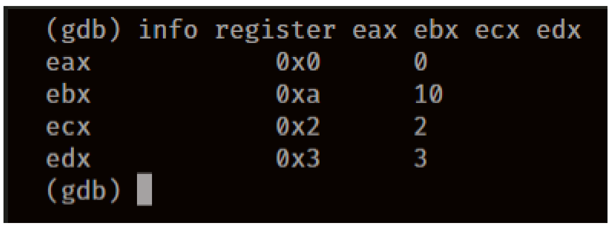

Figure 1 c = b AND c

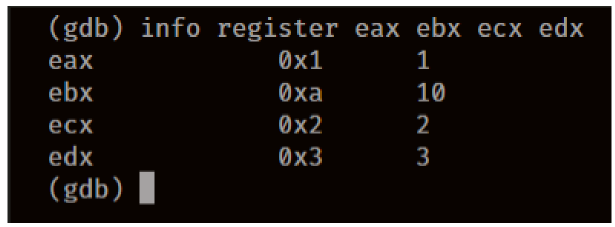

Figure 2 a = (b AND c) XOR d

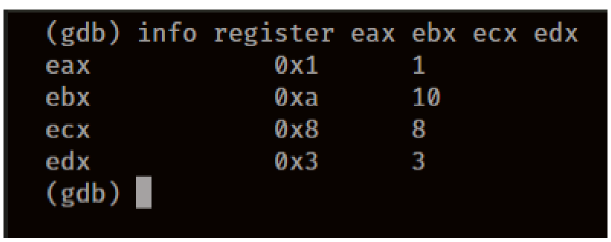

Figure 3 c = b XOR c

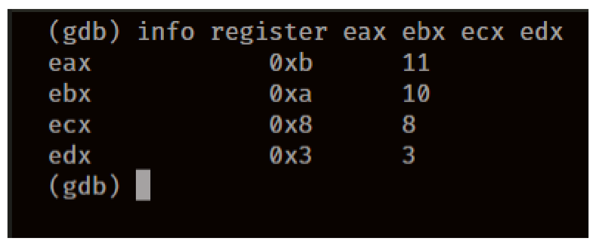

Figure 4 a = (b XOR c) OR d

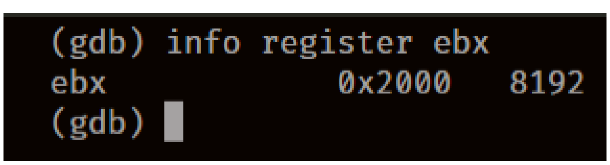

*Figure 5 b = 64 * 128*

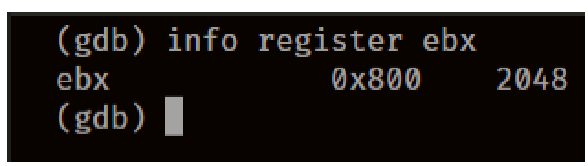

*Figure 6 b = (64 * 128) / 4*

7. Analysis and Discussions

Code	`and <source> <destination>`
Example	`andl $20, %ebx`

Explanation	Performs: `Destination = Destination AND Source` Description: Performs a bitwise AND operation on the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location.

Code	`or <source> <destination>`
Example	`orl $20, %ebx`
Explanation	Performs: `Destination = Destination OR Source` Description: Performs a bitwise inclusive OR operation between the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location. (However, two memory operands cannot be used in one instruction.) Each bit of the result of the OR instruction is set to 0 if both corresponding bits of the first and second operands are 0; otherwise, each bit is set to 1.

Code	`xor <source> <destination>`
Example	`xorl $20, %ebx`
Explanation	Performs:

	Destination = Destination XOR Source Description: Performs a bitwise exclusive OR (XOR) operation on the destination (first) and source (second) operands and stores the result in the destination operand location. The source operand can be an immediate, a register, or a memory location; the destination operand can be a register or a memory location. (However, two memory operands cannot be used in one instruction.) Each bit of the result is 1 if the corresponding bits of the operands are different; each bit is 0 if the corresponding bits are the same.

Code	sal <shift_amt> <destination> sar <shift_amt> <destination>
Example	sal $2, %ebx sar $7, %ebx
Explanation	Performs: Destination = bitwise shift destination shit_amt times, either to the left or to the right, depending upon usage of sal or sar respectively. Description: The shift arithmetic left (SAL) and shift logical left (SHL) instructions perform the same operation; they shift the bits in the destination operand to the left (toward more significant bit locations). For each shift count, the most significant bit of the destination operand is shifted into the CF flag, and the least significant bit is cleared.

	The shift arithmetic right (SAR) and shift logical right (SHR) instructions shift the bits of the destination operand to the right (toward less significant bit locations). For each shift count, the least significant bit of the destination operand is shifted into the CF flag, and the most significant bit is either set or cleared depending on the instruction type. The SHR instruction clears the most significant bit, the SAR instruction sets or clears the most significant bit to correspond to the sign (most significant bit) of the original value in the destination operand. In effect, the SAR instruction fills the empty bit position's shifted value with the sign of the unshifted value. The SAR and SHR instructions can be used to perform signed or unsigned division, respectively, of the destination operand by powers of 2. For example, using the SAR instruction to shift a signed integer 1 bit to the right divides the value by 2.

8. Conclusions

To perform logical operations, we have instructions such as `and`, `or` and `xor`, that perform logical AND, logical OR and logical XOR bitwise respectively. These instructions take in two arguments, which is the source and the destination, and the operation is done for source and destination and the result is then stored in the destination.

To perform multiplication and division we use instructions such as `sal`, and `sar`, which are shift arithmetic left and shift arithmetic right respectively, these basically shift the bits in the register. Shifting the bits to the left multiplies the number by 2 and shifting the bits to the right divides the number by 2.

9. Comments

1. Limitations of Experiments

The Experiment is limited to multiplying and dividing numbers using bitwise shifting operator by only powers of 2, such as 1, 2, 4, 8 . . , basically $2^k \ \forall \ k \geq 0$.

2. Limitations of Results

Shit Left and Shift Right instructions can only multiply the operand by a positive value, i.e. the operand can only be multiplied by a positive number or divided by a positive number.

3. Learning happened

We learnt how to use bitwise logical operators on values stored in registers and also learnt how to multiply and divide numbers by using bitwise shifting operations.

4. Recommendations

Since shifting operations take way less machine execution cycles, they are preferred over `div` and `mul` instructions.

Signature and date

Marks