Computer Organization and Assembly Language

Lab Manual (Lab 12)



**Topic:** Logical operation, Shift operations, Rotate operation

**Course Instructor:**

**Lab Instructor:**

Session: Fall 2015

School of Systems and Technology

UMT Lahore Pakistan

**Objectives:** usage of Logical operations, BIT MASKING, Shift Operations in assembly language.

**Some concepts and Logical instruction assembly language.**

Logical operations are used for bitwise manipulation of data. These operations evaluate the binary values of operands and perform specific bitwise logical functions. The results are stored in the destination operand. Logical operations are fundamental in controlling the behavior of programs at a low level, particularly for data masking and flag manipulations.

#### **1. AND Instruction**

* **Purpose:** Performs a bitwise logical AND operation.
* **Syntax:** AND destination, source
* **Operation:** Each bit in the destination is ANDed with the corresponding bit in the source:

Destination = Destination & Source

* **Applications:** Commonly used for bit masking to clear specific bits.  
  1010 1100 AND 1111 0000 = 1010 0000

#### **2. OR Instruction**

* **Purpose:** Performs a bitwise logical OR operation.
* **Syntax:** OR destination, source
* **Operation:** Each bit in the destination is ORed with the corresponding bit in the source:

Destination = Destination | Source

* **Applications:** Commonly used to set specific bits.  
  **Example:**

1010 1100 OR 0000 1111 = 1010 1111

#### **3. XOR Instruction**

* **Purpose:** Performs a bitwise logical XOR operation.
* **Syntax:** XOR destination, source
* **Operation:** Each bit in the destination is XORed with the corresponding bit in the source. If the bits are different, the result is 1; otherwise, it is 0:

Destination = Destination ^ Source

* **Applications:** Used to toggle specific bits.  
  **Example:**

1010 1100 XOR 1111 0000 = 0101 1100

#### **4. NOT Instruction**

* **Purpose:** Performs a bitwise NOT operation, inverting all bits in the operand.
* **Syntax:** NOT destination
* **Operation:**

Destination = ~Destination

* **Applications:** Used for bitwise inversion.  
  **Example:**

NOT 1010 1100 = 0101 0011

#### **5. TEST Instruction**

* **Purpose:** Performs a bitwise AND operation between the destination and the source without modifying the destination. It is used to test specific bits and set relevant flags (ZF, SF, PF).
* **Syntax:** TEST destination, source
* **Applications:** Checking the state of specific bits (set or clear).

### **Shift**

Shift instructions shift the bits of the operand to the left or right, filling the empty positions with zeros or the sign bit, depending on the operation. These instructions are used for bit manipulation, arithmetic operations, and logical transformations.

#### **1. SHL (Shift Left)**

* **Purpose:** Shifts bits to the left, filling the rightmost bit with 0.
* **Syntax:**

SHL destination, count

* **Applications:** Used for multiplication by powers of 2.

#### **2. SHR (Shift Right)**

* **Purpose:** Shifts bits to the right, filling the leftmost bit with 0.
* **Syntax:**

SHR destination, count

* **Applications:** Used for division by powers of 2.

#### **3. SAL (Shift Arithmetic Left)**

* Same functionality as SHL, used explicitly for arithmetic operations.

#### **4. SAR (Shift Arithmetic Right)**

* **Purpose:** Similar to SHR, but preserves the sign bit (MSB).
* **Applications:** Used for signed division by powers of 2.

*****Rotate*****

Rotate instructions shift bits in a circular fashion. The bits shifted out from one end re-enter from the other end. Rotate instructions are useful in cryptographic algorithms and data manipulations.

#### **1. ROL (Rotate Left)**

* **Purpose:** Rotates bits to the left.
* **Syntax:**

ROL destination, count

* **Operation:** The most significant bit (MSB) is shifted into the least significant bit (LSB).

#### **2. ROR (Rotate Right)**

* **Purpose:** Rotates bits to the right.
* **Syntax:**

ROR destination, count

* **Operation:** The least significant bit (LSB) is shifted into the most significant bit (MSB).

#### **3. RCL (Rotate Through Carry Left)**

* Similar to ROL but involves the carry flag (CF).

#### **4. RCR (Rotate Through Carry Right)**

* Similar to ROR but involves the carry flag (CF).

### **Lab Tasks**

#### **Task 1: Logical Operations and Bit Examination**

1. Prompt the user to enter the first character input.
2. Prompt the user to enter the second character input.
3. Perform the following logical operations and display their results:
   * AND, OR, XOR, NOT.
4. Using the TEST instruction and switch statements, determine which specific bits in the result are set or cleared.
5. Display the following example on the console:

Enter 1st character: A

Enter 2nd character: B

Result of AND operation: 01000000

7th bit is set

6th bit is clear

...

****Code for Task 1:****

.model small

.stack 100h

.data

prompt1 db 'Enter 1st character (A-Z): $'

prompt2 db 0Dh, 0Ah, 'Enter 2nd character (A-Z): $'

result\_msg db 0Dh, 0Ah, 'Logical operation result: $'

bit\_set\_msg db 'th bit is set', 0Dh, 0Ah, '$'

bit\_clear\_msg db 'th bit is clear', 0Dh, 0Ah, '$'

newline db 0Dh, 0Ah, '$'

char1 db ?

char2 db ?

result db ?

bit\_pos db '0', '$' ; To display the bit position

.code

main proc

; Initialize data segment

mov ax, @data

mov ds, ax

; Prompt for 1st character

lea dx, prompt1

mov ah, 09h

int 21h

mov ah, 01h ; Read character

int 21h

mov char1, al ; Store in char1

; Prompt for 2nd character

lea dx, prompt2

mov ah, 09h

int 21h

mov ah, 01h ; Read character

int 21h

mov char2, al ; Store in char2

; Perform AND operation

mov al, char1

and al, char2

mov result, al

; Display the result message

lea dx, result\_msg

mov ah, 09h

int 21h

; Check each bit using TEST and display status

mov cx, 8 ; Loop counter for 8 bits

mov bl, 7 ; Start with the highest bit (bit 7)

mov si, offset bit\_pos ; Point to bit position storage

check\_bit:

; Load the bit position into bit\_pos

mov al, bl

add al, '0' ; Convert bit position to ASCII

mov [si], al ; Store in bit\_pos

; Test the bit in result

mov al, result

mov cl, bl

shl al, cl ; Shift the bit we want to check to the leftmost position

test al, 80h ; Test if the highest bit is set (equivalent to checking the bl-th bit)

jz bit\_clear ; Jump if the bit is clear

bit\_set:

; Display the bit is set message

lea dx, bit\_pos

mov ah, 09h

int 21h

lea dx, bit\_set\_msg

mov ah, 09h

int 21h

jmp next\_bit

bit\_clear:

; Display the bit is clear message

lea dx, bit\_pos

mov ah, 09h

int 21h

lea dx, bit\_clear\_msg

mov ah, 09h

int 21h

next\_bit:

dec bl ; Move to the next lower bit

loop check\_bit ; Repeat for all bits

; Exit program

mov ah, 4Ch

int 21h

main endp

end main

#### **Task 2: Bit Masking and Hexadecimal Digit Separation**

1. Move a 4-digit hexadecimal number into a register.
2. Separate the individual digits using bit masking (AND) and shifting (SHR, SHL).
3. Display each digit individually on the console.  
   **Example:**  
   For 2345h, the output should be:

Copy code

2

3

4

5

****Code for Task 2:****

.model small

.stack 100h

.data

hex\_num dw 2345h ; 4-digit hexadecimal number

newline db 0Dh, 0Ah, '$' ; Newline character (carriage return and line feed)

.code

main proc

; Initialize data segment

mov ax, @data

mov ds, ax

; Load the hexadecimal number into AX

mov ax, hex\_num

; Process and display the 1st digit (thousands place)

mov cx, 12 ; Shift right by 12 bits to get the first digit

shr ax, cl

and ax, 000Fh ; Mask to isolate the first digit

add al, '0' ; Convert to ASCII

mov dl, al ; Move digit to DL for output

mov ah, 02h ; Function to display character

int 21h

; Print a newline

lea dx, newline

mov ah, 09h

int 21h

; Prepare AX for the next digit by restoring the original number

mov ax, hex\_num

; Process and display the 2nd digit (hundreds place)

mov cx, 8 ; Shift right by 8 bits to get the second digit

shr ax, cl

and ax, 000Fh ; Mask to isolate the second digit

add al, '0' ; Convert to ASCII

mov dl, al ; Move digit to DL for output

mov ah, 02h ; Function to display character

int 21h

; Print a newline

lea dx, newline

mov ah, 09h

int 21h

; Prepare AX for the next digit by restoring the original number

mov ax, hex\_num

; Process and display the 3rd digit (tens place)

mov cx, 4 ; Shift right by 4 bits to get the third digit

shr ax, cl

and ax, 000Fh ; Mask to isolate the third digit

add al, '0' ; Convert to ASCII

mov dl, al ; Move digit to DL for output

mov ah, 02h ; Function to display character

int 21h

; Print a newline

lea dx, newline

mov ah, 09h

int 21h

; Prepare AX for the next digit by restoring the original number

mov ax, hex\_num

; Process and display the 4th digit (ones place)

and ax, 000Fh ; Mask to isolate the fourth digit (no shift needed)

add al, '0' ; Convert to ASCII

mov dl, al ; Move digit to DL for output

mov ah, 02h ; Function to display character

int 21h

; Print a newline

lea dx, newline

mov ah, 09h

int 21h

; Exit program

mov ah, 4Ch

int 21h

main endp

end main

### **Task 3: Binary Conversion and Summation**

This task prompts the user to input a 3-digit number, converts each digit to binary, counts the total number of 1s in the binary representation, and displays the sum.

#### **Code:**

.model small

.stack 100h

.data

prompt db 'Enter a 3-digit number: $'

binary\_sum\_msg db 0Dh, 0Ah, 'The Binary Sum is: $'

num db 3 dup('$') ; Array to store the 3-digit number

binary\_arr db 3 dup(0) ; Array to store binary representation counts

sum db 0 ; To store the total number of 1s in binary

.code

main proc

; Initialize data segment

mov ax, @data

mov ds, ax

; Prompt user for input

lea dx, prompt

mov ah, 09h

int 21h

; Read the 3-digit number

lea si, num

mov cx, 3 ; 3 digits to read

input\_loop:

mov ah, 01h ; Read character

int 21h

sub al, '0' ; Convert ASCII to integer

mov [si], al ; Store in num array

inc si

loop input\_loop

; Initialize sum and convert digits to binary

lea si, num

lea di, binary\_arr

mov cx, 3 ; Process 3 digits

convert\_loop:

mov al, [si] ; Get the current digit

xor bl, bl ; Clear counter for binary 1s

mov bh, 8 ; 8 bits to process

count\_ones:

shr al, 1 ; Shift right by 1 bit

jc increment\_count ; If carry flag is set, increment count

jmp continue\_count

increment\_count:

inc bl

continue\_count:

dec bh ; Decrement bit counter

jnz count\_ones ; Repeat for all 8 bits

mov [di], bl ; Store count of 1s in binary\_arr

add sum, bl ; Add count to total sum

inc si

inc di

loop convert\_loop

; Display the binary sum

lea dx, binary\_sum\_msg

mov ah, 09h

int 21h

mov al, sum ; Load sum into AL

add al, '0' ; Convert to ASCII

mov dl, al

mov ah, 02h ; Print character

int 21h

; Exit program

mov ah, 4Ch

int 21h

main endp

end main

### **Task 4: ASCII Code Display**

This task prompts the user to enter a character, calculates its ASCII code in hexadecimal, and displays the result.

#### **Code:**

.model small

.stack 100h

.data

prompt db 'Enter any Character: $'

ascii\_msg db 0Dh, 0Ah, 'Character ASCII Code is: $'

hex\_digits db '0123456789ABCDEF$' ; Hexadecimal digits

char\_input db ? ; To store user input character

.code

main proc

; Initialize data segment

mov ax, @data

mov ds, ax

; Prompt user for input

lea dx, prompt

mov ah, 09h

int 21h

; Read character input

mov ah, 01h

int 21h

mov char\_input, al ; Store input character

; Display ASCII message

lea dx, ascii\_msg

mov ah, 09h

int 21h

; Convert ASCII value to hexadecimal

mov al, char\_input ; Load character into AL

; Display high nibble

mov ah, al

shr ah, 4 ; Shift right by 4 bits to get the high nibble

and ah, 0Fh ; Mask to isolate high nibble

add ah, '0' ; Convert to ASCII

cmp ah, '9'

jbe display\_high

add ah, 7 ; Adjust for A-F

display\_high:

mov dl, ah

mov ah, 02h

int 21h

; Display low nibble

mov al, char\_input

and al, 0Fh ; Mask to isolate low nibble

add al, '0' ; Convert to ASCII

cmp al, '9'

jbe display\_low

add al, 7 ; Adjust for A-F

display\_low:

mov dl, al

mov ah, 02h

int 21h

; Display 'h' to indicate hexadecimal

mov dl, 'h'

mov ah, 02h

int 21h

; Exit program

mov ah, 4Ch

int 21h

main endp

end main