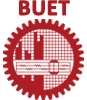
**BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY**



**Department of Electrical and Electronic Engineering**

**Course No. :** EEE 416

**Course Title:** Microprocessor and Interfacing Laboratory

**Rotate, Shift and Loops in Assembly Language**

**Name:** Mir Sayeed Mohammad

**ID:** 1606003

**Level:** 4

**Term:** 1

**Section:** A

**Submission Deadline:** 21 - 3 -2021

**Lab work 1**

Find the LCM of 3 given numbers (0Fh, 4Bh, 20Dh)

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

; \*\*\*\*\*\*\*\*\*\* LCM between num1 and num2 \*\*\*\*\*\*\*\*\*\* ;

MOV AX, num1

MOV BX, num2

MUL BX ; AX = AX\*BX

MOV tmp, AX ; tmp = num1\*num2

MOV AX, num1

GCD1: XOR DX, DX ; Set dividend to zero in start of each cycle

DIV BX ; Divide AX by BX, quotient saved to AX, dividend to DX

MOV AX, BX ; AX = BX

MOV BX, DX ; BX = dividend

CMP DX, 0H ; check if dividend was 0

JNZ GCD1 ; keep jumping until dividend is zero

; result will be in AX register

MOV BX, AX ; GCD stored in BX

MOV AX, tmp ; AX = GCD \* LCM

DIV BX ; AX will store LCM

MOV lcm, AX ; store current lcm in AX

; \*\*\*\*\*\*\*\*\*\* LCM between LCM1 and num3 \*\*\*\*\*\*\*\*\*\* ;

MOV BX, num3

MUL BX ; AX = lcm1 \* num3

MOV tmp, AX ; tmp = lcm1 \* num3

MOV AX, lcm

GCD2: XOR DX, DX ; Set dividend to zero in start of each cycle

DIV BX ; Divide AX by BX, quotient saved to AX, dividend to DX

MOV AX, BX ; AX = BX

MOV BX, DX ; BX = dividend

CMP DX, 0H ; check if dividend was 0

JNZ GCD2 ; keep jumping until dividend is zero

; result will be in AX register

MOV BX, AX ; GCD stored in BX

MOV AX, tmp ; AX = GCD \* LCM

DIV BX ; AX will store LCM

MOV lcm, AX ; store LCM in variable

HLT

; \*\*\*\*\*\*\*\*\*\* Variables initialization \*\*\*\*\*\*\*\*\*\* ;

num1 DW 0FH

num2 DW 4BH

num3 DW 20DH

tmp DW 0H ; temporary variable

lcm DW 0H ; result storage

CODE ENDS

END

**Explanation:**

The code uses the methodology of counting the GCD of two numbers, and finding their LCM from their multiplied result and GCD. In this method, num1 and num2 are used to find LCM1, and then final LCM is the LCM of LCM1 and num3.

**Result:**

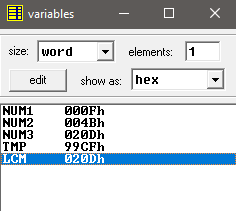


Fig: LCM of 3 variables

**Lab work 2**

Consider a number 32h. Reverse its bit pattern and count the number of 1s.

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

MOV AX, 32H ; value under consideration

MOV CX, 16D ; each register has 16 bits in total

; we have to iterate through 16 bits

FLIP:

CMP CX, 0H ; quit loop if loop counter CX is 0

JZ LAST

SHR AX, 01H ; logical shifting AX register to the right

; produces carry of 1 if the value that

; popped out of register was 1

DEC CX

JC APPEND1

JNC APPEND0

APPEND0: ; left shift DX and append 0 on right

SHL DX, 01H

JMP FLIP

APPEND1: ; left shift DX and append 1 on right

SHL DX, 01H

INC DX

INC BX ; count the 1 in question

JMP FLIP

LAST:

HLT

CODE ENDS

END

**Explanation:**

Logical right shift generates carry 1 if a 1 pops out of the register. If the carry is 1, a 1 is appended on the right most bit of result DX (and counting the 1 by incrementing a register BX). If the carry is 0, a 0 is appended on the right most bit of result DX.

**Result:**

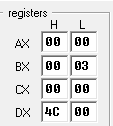


Fig: Execution of Reversing and Counting operation

Input AX = 32H (0000 0000 0011 0010)

Number of 1’s, BX = 3

AX reversed, DX = 4C00H (0100 1100 0000 0000)

**Report 1**

Suppose x=20 and y=28. Add y with x for 30 times

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

; \*\*\*\*\*\*\*\*\*\* Main Sequence \*\*\*\*\*\*\*\*\*\* ;

MOV CX, n

MOV AX, x

adder:

ADD AX, y

LOOP adder

MOV ans, AX

HLT

; \*\*\*\*\*\*\*\*\*\* Variables initialization \*\*\*\*\*\*\*\*\*\* ;

x DW 20D

y DW 28D

n DW 30D

ans DW 0H

CODE ENDS

END

**Result:**

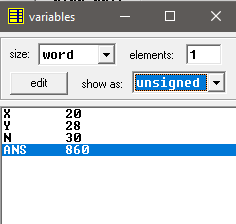
****

Fig: ans = 20 + 28\*30 = 860

**Report 2**

Multiply 12 by 6 as long as result is below 3000H. If result is greater than this, divide the result by 2 for 3 times.

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

; \*\*\*\*\*\*\*\*\*\* Main Sequence \*\*\*\*\*\*\*\*\*\* ;

MOV AX, 12D

MOV BX, 6D

multiplier:

MUL BX

CMP AX, 03000H

JS multiplier ; jump to multiplier as long as AX < 3000H

MOV CX, 3D ; loop counter init to 3

MOV BX, 2D

divider:

DIV BX

LOOP divider

HLT

CODE ENDS

END

**Result:**

|  |  |  |
| --- | --- | --- |
| AX = 00CH (12D) | AX < 3000H, multiply by BX = 06H |  |
| AX = 0048H | AX < 3000H, multiply by BX = 06H |
| AX = 01B0H | AX < 3000H, multiply by BX = 06H |
| AX = 0A20H | AX < 3000H, multiply by BX = 06H |
| AX = 3CC0H | AX > 3000H, divide by CX = 02H (1) |
| AX = 1E60H | divide by CX = 02H (2) |
| AX = 0F30H | divide by CX = 02H (3) |
| AX = 0798H | Halt |

**Report 3**

Take an input from the keyboard until b is pressed

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

; \*\*\*\*\*\*\*\*\*\* Main Sequence \*\*\*\*\*\*\*\*\*\* ;

input: MOV AH, 1H ; keyboard input subprogram

INT 21H

CMP AX, character\_b

JNZ input ; take input if AX != character\_b

HLT

character\_b DW 0162H ; hardcoded value

CODE ENDS

END

**Explanation:**

Input subprogram takes keyboard input from console window

MOV AH, 1H

INT 21H

Each input is saved into the AX register. After each input, AX value is compared to the ‘b’ character value with has the data\_word value of 0162H (checked by inputting b in the console). If the value is equal to ‘b’, program halts.

**Result:**

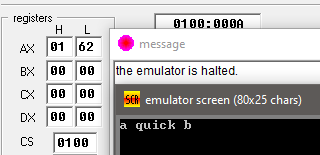
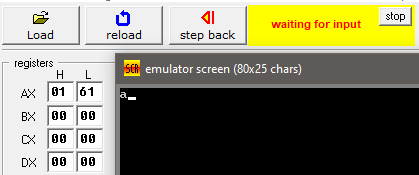


Fig: (a) console waiting for input (b) emulator halted when b is input

**Report 4**

Write an assembly code to compute the product of the integers in AL & BL by add-and-shift-loop method. Put result in AX. You cannot use MUL command.

**Assembly Code:**

CODE SEGMENT

ASSUME CS:CODE, DS:CODE

; Loading Data

MOV AL, num1

MOV BL, num2

MOV DL, AL ; creating backup as AX will store result

MOV CL, 8D ; loop counter for multiplying 8 bits

MOV AX, 0H ; initialize result to 0

multiplier:

SHL AX, 01H ; adding a zero to right of result

SHL BL, 01H ; shifting BL by one for MSB

JNC skip ; if MSB was 0, no operation

ADD AX, DX ; if MSB was 1, add num1 bits to result

skip: NOP

loop multiplier ; loop the multiplier 16 times

last: HLT

num1 DB 1BH

num2 DB 1CH

CODE ENDS

END

**Explanation:**

The code first creates a backup of AL into DX. AX is then set to 0 (sum). A loop iterates 8 times (for 8 bits). In each loop, result AX is first left shifted. Then BL is left shifted. If the MSB of BL was 1, DX is added to AX, otherwise nothing happens and the loop continues. In this way, the multiplication occurs from the MSB of BX to LSB.

**Output:**

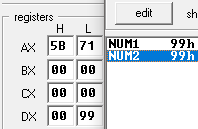
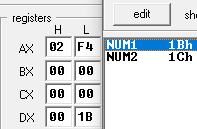
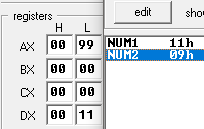


Fig: Several multiplication results