**Practical Workbook**

**CEL-221**

**Computer Organization and Assembly Language**

**Bachelor of Science (Computer Science)**



Student Name: Nimra Mushtaq

Registration Number: 02-134202-018

Class & Section: BSCS-3A

Lab Day & Timing: Wednesday 11:30 – 2:30

Course Teacher: Aisha Danish

Lab Engineer: Sir Muslim

Department of Computer Sciences

**Bahria University Karachi Campus**

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| Lab no. | Title |  |
| 1 | Exploring Instruction Set Architecture (ISA) of x86 Machine |  |
| 2 | Introduction to Assembly Language of x86 Machines and learn Basic Elements of Assembly Language |  |
| 3 | Introduction to Assembly Language Tools and Familiarization with Emu8086 environment |  |
| 4 | To learn how Data is represented in Assembly Language |  |
| 5 | Using x86 Data Transfer Instructions |  |
| 6 | Using x86 Arithmetic |  |
| 7 | Using x86 Logical Instructions |  |
| 8 | Using x86 Stack Instructions |  |
| 9 | Implementing Branching in x86 Assembly Language |  |
| 10 | Development of Procedures and Macros in x86 Assembly Language |  |
| 11 | Understanding Interrupts |  |
| 12 | Using x86 I/O Operations |  |
| 13 | Calling a Subroutine from another Assembly File as a near Procedure using MASM tool |  |

**Lab Session 01**

**Procedure:**

1. Use Emu8086 to make the following calculations:

10100101b = ? d

1234h = ? d

39d = ? h

1. Start *Emu8086* by selecting its icon from the start menu, or by running Emu8086.exe

3. Choose “Math” and specify “Base Convertor” in emu8086.

4. Enter one of the numbers like in the Figure below

5. Do the calculations manually and compare with the results produced by *base converter.*

1. Find out the largest positive 8-bit value in binary, hexadecimal, and decimal?

Binary= 11111111

Hexadecimal=FF

Decimal=255

1. Find out the smallest negative 8-bit value in binary, hexadecimal, and decimal?

Binary= 10000000

Hexadecimal=80

Decimal=128

1. Find out the largest positive 16-bit value in binary, hexadecimal, and decimal?

Binary= 11111111 11111111

Hexadecimal=FFFF

Decimal=65535

1. Find out the smallest negative 16-bit value in binary, hexadecimal, and decimal?

Binary= 10000000000000000

Hexadecimal=8000

Decimal= 32768

**Exercise:**

1. Name four software tools used for assembly language programming.

MASM (Macro Assembler from Microsoft), TASM(Turbo Assembler from Borland), NASM(Netwide Assembler for both Windows and Linux),GNU.

1. What is an Emulator?

A device emulator is a program or device that enables a computer system to behave like another device. An emulator essentially allows one computer system (aka “the host”) to imitate the functions of another (aka “the guest”). Emulation addresses the original hardware and software environment of the digital object and recreates it on a current machine. The emulator allows the user to have access to any kind of application or operating system on a current platform, while the software runs as it did in its original environment.

1. Convert the following numbers using Base Converter:
   1. 10110011b (binary to hexadecimal) =B3
   2. 455d (decimal to binary)=0000000111000111
   3. 2AFh (hexadecimal to binary)=0000001010101111
   4. 0A1h (hexadecimal to binary)=0000000010100001

Do all the calculations manually and compare with the results obtained using base converter.

A piece of paper with writing on it

Description automatically generated

Text, letter

Description automatically generated

4. Write each of the following Integers in 8-bit 2’s Complement Notation:

|  |  |
| --- | --- |
| a) -1 = 00000001 | d) -62 = 00111110 |
| b) -17 =00010001 | e) +127 =01000001 |
| c) -19 =00010011 | f) -128 =10000000 |

5**.** Write each of the following 8-bit Signed Binary Integers in Decimal:

|  |  |
| --- | --- |
| a) 01011100 = 92 | d) 01111110 =126 |
| b) 11011100 =-36 | e) 10010001 = -111 |
| c) 10001111 =-113 | f) 10000000 =-128 |

6. Indicate the sign for each of the following 16-bit signed hex integers:

a) 7FB9

Binary: 0111 1111 1011 1001

b) D000

Binary: 1101 0000 0000 0000

c) 8123

Binary: 1000 0001 0010 0011

d) 6FFF1

Binary: 0110 1111 1111 1111

**Lab Session 02**

# Procedure:

Start *Emu8086* by selecting its icon. Write the following code in the text editor Program 01:

org 100h

mov al, 5 ; bin=00000101b

mov bl, 10 ;bin=00001010b

; 5 + 10 = 15 (decimal) or hex=0Fh or bin=00001111b

add al, bl

ret

Press the *emulate* button and single step the code. Observe the values in the registers.

Note the final values of registers in the following table

|  |  |
| --- | --- |
| Register | Value |
| AX | 0F |
| BX | 0A |
| CS | 06 |
| IP | 0106 |

Program 02:

### org 100h

mov al, 5 ; al = 5 add al, -3 ; al = 2 ret

Observe the values in the registers.

Note the final values of registers in the following table

|  |  |
| --- | --- |
| Register | Value |
| AX | 02 |
| BX | 00 |
| CS | 0700 |
| IP | 0104 |

Program 03:

### Org 100h

mov bl, 5 ; bl = 5 add bl, -3 ; bl = 2 ret

Observe the values in the registers.

Note the final values of registers in the following table

|  |  |
| --- | --- |
| Register | Value |
| AX | 00 |
| BX | 02 |
| CS | 0700 |
| IP | 0105 |

Program 04:

### Org 100h mov al, 5

sub al, 1 ; al = 4 ret

Observe the values in the registers.

Note the final values of registers in the following table

|  |  |
| --- | --- |
| Register | Value |
| AX | 04 |
| BX | 00 |
| CS | 0700 |
| IP | 0104 |

Program 05:

### Org 100h mov al, 7

mov bl, 4 sub al,bl ret

Observe the values in the registers.

Note the final values of registers in the following table

|  |  |
| --- | --- |
| Register | Value |
| AX | 03 |
| BX | 04 |
| CS | 0700 |
| IP | 0106 |
|  |  |

Where (in which register) is the result of addition stored?

Al

Why is the answer stored in the register you mentioned above?

Because Al is the destination and Bl is the source the result of the operation is stored in the destination register.

Exercise:

1. Write a program to subtract two integer constants using SUB command.

Org 100h

mov al, 10

mov bl, 5

sub al,bl

ret

1. Differentiate between high-level and low-level language.

|  |  |
| --- | --- |
| **High Level Language** | **Low Level Language** |
| It is programmer friendly | It is machine friendly |
| High level language is less memory efficient | Low level language is high memory efficient. |
| It is simple to debug. | It is complex to debug comparatively. |
| It can run on any platform | It is machine dependent |
|  |  |
| It needs compiler or interpreter for translation. | It needs assembler for translation. |

1. What is meant by a *one-to-many relationship* when comparing a high-level language to machine language?

One to many relationships means that a single high level program statement is translated or expanded into multiple assembly language or machine language instructions.

1. What is the advantage of using Hexadecimal notation while addressing memory?

The main advantage of using Hexadecimal numbers is that it uses less memory to store more numbers, for example it stores 256 numbers in two digits whereas decimal number stores 100 numbers in two digits. This number system is also used to represent Computer memory addresses.

1. Explain the concept of *portability* as it applies to programming languages.

Portability is a characteristic attributed to a computer program if it can be used in an operating system other than the one in which it was created without requiring major rework. Porting is the task of doing any work necessary to make the computer program run in the new environment.

1. Is the assembly language for x86 processors the same as those for computer systems such as the Vax or Motorola 68x00?

No, each assembly language is based on the processor family or the specific computer.

**Lab Session 3**

**Procedure:**

Start *Emu8086* by selecting its icon.

Write the following codes in the text editor

Modify the codes according to given instructions

Program 1:

org 100h

bytea db 15d

mov al,bytea

ret

Modify Program 1 to include comments, labels and identifiers wherever necessary.

org 100h

.DATA

bytea db 15d ; declaring 1byte variable;. DATA label

.CODE

mov al,bytea ; moving data;. CODE label

ret

Fill in the following table for Program 1

Constants 15d

Variables bytea

Directive(s) . CODE .DATA

Mnemonic MOV

Program 2**:**

org 100h

bytea db 15d

byteb db 06d

mov al,bytea

mov ah,byteb

ret

Modify Program 2 to include comments, labels and identifiers wherever necessary.

org 100h

.DATA

bytea db 15d ; declaring 1byte variable; .DATA label

byteb db 06d ; declaring 1byte variable ;.DATA label

.CODE

mov al,bytea ; moving variable into register, .CODE label

mov ah,byteb; moving variable to register, .CODE label

ret

Fill in the following table for Program 2

Table

Description automatically generated

Program 3**:**

org 100h

worda dw 12d

wordb dw 13d

mov ax,worda

mov bx,wordb

ret

Modify Program 3 to include comments, labels and identifiers wherever necessary.

org 100h

.DATA

worda dw 12d ; declaring 2byte variable , .DATA label

wordb dw 13d ; declaring 2byte variable , .DATA label

.CODE

mov ax,worda ; moving variable to register, .CODE label

mov bx,wordb; moving variable to register, .CODE label

ret

Fill in the following table for Program 3

Constants 12d , 13d

Variables worda , wordb

Directive(s) .DATA , .CODE

Mnemonic MOV

Program 4**:**

org 100h

worda dw 120d

wordb dw 121d

mov ax,worda

mov bx,wordb

ret

Modify Program 4 to include comments, labels and identifiers wherever necessary.

org 100h

.DATA

worda dw 120d; declaring 2byte variable , .DATA label

wordb dw 121d; declaring 2byte variable , .DATA label

.CODE

mov ax,worda; moving variable to register, .CODE label

mov bx,wordb; moving variable to register, .CODE label

ret

Fill in the following table for Program 4

Constants 120d , 121d

Variables worda,wordb

Directive(s) .CODE , .DATA

Mnemonic MOV

**Exercise:**

1. Calculate (3010+1510)\*(57510-22510)+21010

org 100h

; add your code here

.data

val1 dw 30

val2 dw 15

val3 dw 575

val4 dw 225

val5 dw 210

.code

mov ax,val1

add ax,val2

mov bx,val3

sub bx,val4

mul bx

add ax,val5

ret

Output:

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Using a calculator, calculate the answer for the above arithmetic operations. Is it the same as the final answer in the AX register?

Yes, the value with calculator is same as the emulator programme

The AX register = 3E 58 which in decimal value is equal to 15960

2. Identify valid suffix characters used in integer constants.

An Integer constant is made up of an optional leading sign, one or more digits, and an optional radix character that indicates the number’s base.

Following are some valid suffix characters:

* + Hexadecimal (h)
  + Decimal (d/t)
  + Binary (b/y)
  + Encoded real (r)
  + Octal (q/o)

1. Does the multiplication operator (\*) have a higher precedence than the division operator?

(/) in integer expressions?

No, the division operator has higher precedence than a multiplication operator although both have a higher precedence than addition or subtraction but when written in an expression the division operation takes place before the multiplication operation.

4. Must string constants be enclosed in single quotes?

A string constant can be enclosed in both single or double quotes.

5. What is the maximum length of an identifier?

It may contain 1 to 247 characters including digits

6. Do Assembler directives execute at runtime?

Directives do not execute at run time, but they let you define the variables and procedure

whereas instructions do.

7. Name the four basic parts of an assembly language instruction.

The four basic parts of an instructions are

* Labels
* Mnemonics
* Operands
* Comments

8. Write an example of a block comment.

Block comments, beginning with the COMMENT directive and a user-specified symbol. All subsequent lines of text are ignored by the assembler until the same user-specified symbol appears.

Example:

COMMENT &

This is a comment

It can contain any symbol as long as it does not

Appear within the comment lines

&

**Lab Session 4**

**Procedure:**

Start *Emu8086* by selecting its icon.

Write the following codes in the text editor

Observe values in the registers and fill the observation tables.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Program 1: | | |  |  |  |
|  | org 100h | |  |  |  |  |
|  | ; variables: | | |  |  |  |
|  | bytea db 15d | | |  |  |  |
|  | mov al,bytea | | |  |  |  |
|  | ret | | |  |  |  |
|  |  |  | |  |  |  |
|  | Register | | | Value |  | Value |
|  |  |  | AX | AH= | 00 | AL=0F |
|  | Program 2**:** | | |  |  |  |
|  | org 100h |  | |  |  |  |
|  | ; variables: | | |  |  |  |
|  | bytea db 15d | | |  |  |  |
|  | byteb db 06d | | |  |  |  |
|  | mov al,bytea | | |  |  |  |
|  | mov ah,byteb | | |  |  |  |
|  | ret | | |  |  |  |
|  |  | | |  |  |  |
|  | Register | | | Value |  | Value |
|  |  |  | AX | AH= | 06 | AL=0F |
|  | Program 3**:** | | |  |  |  |
|  | org 100h |  | |  |  |  |
|  | ; variables: | | |  |  |  |
|  | worda dw 12d | | |  |  |  |
|  | wordb dw 13d | | |  |  |  |

mov ax,worda

mov bx,wordb

ret

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Register | Value |
|  |  | AX | A1 |
|  |  | BX | 0D |
|  | Program 4**:** | |  |
|  | org 100h |  |  |
|  | ; variables: | |  |
|  | worda dw 120d ; | |  |
|  | wordb dw 121d | |  |
|  | mov ax,worda | |  |
|  | mov bx,wordb | |  |
|  | ret | |  |
|  |  |  |  |
|  |  | Register | Value |
|  |  | AX | 78 |
|  |  | BX | 79 |

**Exercise:**

1. What is the major advantage of floating-point numbers over fixed-point and integer representations?

The advantage of floating-point numbers over fixed-point and integer representations is that they can support a much wider range of values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. Modify Program 1 and 2 for DX and fill the observation table: | | | |  |
|  |  |  |  |  |
| Register | Value | |  | Value |
| DX | DH=00 | |  | DL=0F |
| DX | DH=06 | |  | DL=0F |
| Program 1:  org 100h  ; variables:  bytea db 15d  byteb db 06d  mov dl,bytea  ret  Program 2 :  org 100h  ; variables:  bytea db 15d  byteb db 06d  mov dl,bytea  mov dh,byteb  ret  Modify program 3 and 4 for AX and DX and fill the observation table: | | | |  |
|  |  |  |  |  |
| Register |  |  | Value |  |
| AX |  |  | A1 |  |
| DX |  |  | 0D |  |
| Register |  |  | Value |  |
| AX |  |  | 78 |  |
| DX |  |  | 79 |  |

Program 3:

org 100h

; variables:

worda dw 12d

wordb dw 13d

mov ax,worda

mov dx,wordb

ret

Program 4:

org 100h

; variables:

worda dw 120d ;

wordb dw 121d

mov ax,worda

mov dx,wordb

ret

1. Write a program to add two Byte-size integers. Name the register in which the result is stored.

Al stores the result 15.

Program:

org 100h

; variables:

bytea db 15d

byteb db 06d

mov ah,bytea

mov al,byteb

add al,ah

ret

1. Write a program to subtract two Byte- size integers. Name the register in which the result is stored.

Al register stores the result 8.

Program:

org 100h

bytea db 10d

byteb db 2d

mov al, bytea

mov bl, byteb

sub al, bl

ret

1. What is the range of values for an unsigned byte integer?

Their values range from 0 to 255 for unsigned byte integers

7. What do DB, DW, DD, DQ, and DT stand for?

* DB (Define Byte)
* DW (Define Word – 2Bytes)
* DD (Define doubleword-4Bytes)
* DQ (Define quadword-8Bytes)
* DT (Define tenbytes-10Bytes)

8. What is the most efficient data storage size in the 32-bit programming environment?

One bit in the register can reference an individual byte in memory, so a 32-bit system can address a maximum of **4 GB (4,294,967,296 bytes)** of RAM.

1. What must you do when transferring an 8-bit integer value from a byte-size register to a word-size register?

If the number is positive, then fill the higher-order bits with zeros and if the number is negative, fill the higher-order bits with ones.

**Lab Session 05**

**Procedure:**

**Represent the decimal number 23.5 in 32-bit floating point binary format:**

|  |  |  |
| --- | --- | --- |
| Sign | Excess-127 Exponent | Normalized Fraction |
| 0 | 1 0 0 0 0 0 1 1 | 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |
|  |  |  |

**Represent the decimal number 46.875 in 32-bit floating point binary format:**

|  |  |  |
| --- | --- | --- |
| Sign | Excess-127 Exponent | Normalized Fraction |
| 0 | 10000100 | 01110111000000000000000 |

Express the following numbers in IEEE 32-bit floating-point format:

|  |  |
| --- | --- |
| Decimal | Floating-point Format |
| -5 | 1 10000001 01000000000000000000000 |
| -6 | 1 10000001 10000000000000000000000 |
| 384 | 0 10000111 10000000000000000000000 |
| 64.5 | 0 10000101 00000010000000000000000 |
| 1/8 | 0 01111100 00000000000000000000000 |
| 1/16 | 0 01111011 00000000000000000000000 |
| -1/32 | 0 01111010 00000000000000000000000 |

The following numbers use the IEEE 32-bit floating-point format. What is the equivalent decimal value?

1. 1 10000011 11000000000000000000000 **(-28) 10**
2. 0 01111110 10100000000000000000000 **(0.8125) 10**
3. 0 10000000 00000000000000000000000 **(2)10**

**Exercise:**

1. What are the four essential elements of a number in floating-point notation?

* Sign
* Significand
* Exponent
* Base

1. What is the benefit of using biased representation for the exponent portion of a

floating-point number?

The advantage of biased representation is that non-negative floating-point numbers can be treated as integers for comparison purposes.

1. What are the differences among positive overflow, exponent overflow, and significand overflow?

**Positive overflow** refers to integer representations and refers to a number that is larger than can be represented in a given number of bits.

E**xponent overflow**refers to floating point representations and refers to a positive exponent that exceeds the maximum possible exponent value.

**Significand overflow**occurs when the addition of two significands of the same sign may result in a carry out of the most significant bit

**LAB SESSION 6**

Procedure:

Start *Emu8086* by selecting its icon.

Write the following codes in the text editor and observe the operation of Arithmetic instructions.

Observe the values of Flags and fill the observation tables.

Program 01:



Algorithm:

operand = operand + 1

org 100h

MOV AL, 4

INC AL ; AL = 5

RET

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| 0 | 0 | 0 | 0 | 1 | 0 |

Program 02:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| 0 | 0 | 1 | 0 | 0 | 0 |



Algorithm:

operand = operand - 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| org 100h |  |  |  |  |
| MOV AL,255 | ; AL = 0FFh | (255 | or | -1) |
| DEC AL | ; AL = 0FEh | (254 | or | -2) |
| RET |  |  |  |  |
| Program 03: |  |  |  |  |



Algorithm: Invert all bits of the operand Add 1 to inverted operand

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Org 100h |  |  |  |  |
| MOV AL, 5 | ; AL = 05h | | |  |
| NEG AL | ; | AL = | 0FBh (-5) | |
| NEG AL | ; | AL = | 05h | (5) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| 1 | 0 | 0 | 0 | 1 | 1 |

program 04:



Algorithm:

operand1 = operand1 + operand2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Org 100h | |  |  |  |  |  |  |
| MOV AL, 5 | | ; AL = 5 | |  |  |  |  |
| ADD AL, -3 | | ; AL = 2 | |  |  |  |  |
| RET | |  |  |  |  |  |  |
| Flags | |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| C |  | Z |  | S | O | P | A |
| 1 | 0 |  | 0 |  | 0 | 0 | 1 |

Program 05:



Algorithm:

operand1 = operand1 - operand2

org 100h

MOV AL, 5

SUB AL, 1 ; AL = 4

RET

Flags

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| 0 | 0 | 0 | 0 | 0 | 0 |

Program 06:

Algorithm:

when operand is a **byte**:

when operand is a **word**:

AX = AL \* operand.

(DX AX) = AX \* operand.

* Unsigned multiply Org 100h

MOV AL, 200 ; AL = 0C8h

MOV BL, 4

MUL BL ; AX = 0320h (800)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RET  Flags | |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| C |  | Z | S | O |  | P | A |  |
| 1 | 0 |  | 0 | 1 |  | 0 | 0 |  |

Program 07:

Algorithm:

when operand is a **byte**:

when operand is a **word**:

AX = AL \* operand.

(DX AX) = AX \* operand.

1. Signed multiply Org 100h

MOV AL, -2 MOV BL, -4 IMUL BL ; AX = 8

RET

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| 0 | 0 | 0 | 0 | 0 | 0 |

Program 08:Background pattern

Description automatically generated with low confidence

Algorithm:

when operand is a **byte**:

when operand is a **word**:

AL = AX / operand

AH = remainder (modulus)

AX = (DX AX) / operand DX = remainder (modulus)

* Unsigned division Org 100h

MOV AX, 203 ; AX = 00CBh MOV BL, 4

DIV BL ; AL = 50 (32h), AH = 3

RET

Flags

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| 0 | 0 | 0 | 0 | 0 | 0 |

Program 09:

Algorithm:

when operand is a **byte**:

when operand is a **word**:

AL = AX / operand

AH = remainder (modulus)

AX = (DX AX) / operand DX = remainder (modulus)

* Signed division Org 100h

MOV AX, -203 ; AX = 0FF35h MOV BL, 4

IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)

RET

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| 0 | 0 | 0 | 0 | 0 | 0 |

* + Define two 8-bit variables and write the instructions to add them.

MOV AL,254

MOV BL,1

add al,bl

RET

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* + Define two 8-bit variables and write the instructions to sub them.

MOV AL,11

MOV BL,1

sub al,bl

RET

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* + Define two 8-bit variables and write the instructions to mul them.

MOV AL,21

MOV BL,7

mul bl

RET

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* + Define two 8-bit variables and write the instructions to divide them.

MOV AL,31

MOV BL,5

div bl

RET

Exercise:

1. What will be the contents of AX after the following operation? mov ax,63h

mov bl,10h div bl

|  |  |
| --- | --- |
| AH | AL |
| 03 | 06 |

1. Write instructions that multiply -5 by 3 and store the result in a 16-bit variable **var1.**

var1 dw 0

mov al,-5

mov bl,3

Imul bl

mov var1,aX

ret

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Write instructions that divide -276 by 10 and store the result in a 16-bit variable **var1.**

var1 dw 0

mov ax,-276

mov bx,10

Idiv bx

mov var1,ax

ret

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**LAB SESSION 7**

Procedure:

Start *Emu8086* by selecting its icon.

Write the following codes in the text editor

Fill the observation tables accordingly

Program 01:

org 100h

MOV AL, 'a' ; AL = 01100001b

AND AL, 11011111b ; AL = 01000001b ('A')

RET

FLAGS

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| C | | | | Z | | | S | | | | O | | | P | | | A | |
| 0 | | | | 0 | | | 0 | | | | 0 | | | 1 | | | 0 | |
|  | |
|  | | | | | |  | | | | | | |
|  | |  |  | | | | | |  |  | | | | | |  | | | | |
|  | | Register | | | | | | |  | Value | | | | | | Value | | | | |
|  | |  | AX | | | | | |  | AH=00 | | | | | | AL=41 | | | | |
|  | | Program 02: |  | | | | | | |  | | | | | |  | | | | |
|  | | org 100h | | | | | | | |  | | | | | |  | | | | |
|  | | MOV AL, 'A' ; AL = 01000001b | | | | | | | |  | | | | | |  | | | | |
|  | | OR AL, 00100000b ; AL = 01100001b ('a') | | | | | | | | | | | | | |  | | | | |
|  | | RET  FLAGS | | | | | | | |  | | | | | |  | | | | |
| C | | | | | Z | | | S | | | | O | | | P | | | A | |
| 0 | | | | | 0 | | | 0 | | | | 0 | | | 0 | | | 0 | |
|  | |  | | | | | | | |  | | | | | |  | | | | |
|  | | Register | | | | | | |  | Value | | | | | | Value | | | | |
|  | |  | AX | | | | | |  | AH=00 | | | | | | AL=61 | | | | |
|  | | Program 03: |  | | | | | | |  | | | | | |  | | | | |
|  | | org 100h | | | | | | | |  | | | | | |  | | | | |
|  | | MOV AL, 00011011b | | | | | | | |  | | | | | |  | | | | |
|  | | NOT AL ; AL = 11100100b | | | | | | | |  | | | | | |  | | | | |
|  | | RET  Flags | | | | | | | |  | | | | | |  | | | | |
| C | | | | | Z | | | S | | | | O | | | P | | | A | |
| 0 | | | | | 0 | | | 0 | | | | 0 | | | 0 | | | 0 | |
|  | |  | | | | | | | |  | | | | | |  | | | | |
|  | | Register | | | | | | |  | Value | | | | | | Value | | | | |
|  | |  | AX | | | | | |  | AH=00 | | | | | | AL=E4 | | | | |

Program 04:

org 100h

MOV AL, 00000111b

XOR AL, 00000010b ; AL = 00000101b

RET

Flags

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C | Z | S | O | P | A |
| 0 | 0 | 0 | 0 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
| Register | Value | Value |
| AX | AH=00 | AL=05 |

Exercise:

1. In the following instruction sequence, show the resulting value of AL where indicated, in binary:

|  |  |  |
| --- | --- | --- |
| MOV AL,01101111b |  |  |
| AND AL,00101101b | ; a. | AL= 00101101 |
| MOV AL,6Dh |  |  |
| AND AL,4Ah | ; b. | AL= 01001000 |
| MOV AL,00001111b |  |  |
| OR AL,61h | ; c. | AL= 01101111 |
| MOV AL,94h |  |  |
| XOR AL,37h | ; d. | AL= 10100011 |

1. Write a single instruction using 16-bit operands that clears the high 8 bits of AX and does not change the low 8 bits.

org 100h

mov ax,0010010000101010b;242A

and ax,0000000001101111b;002A

RET

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Write a single instruction using 16-bit operands that sets the high 8 bits of AX and does not change the low 8 bits.

org 100h

mov ax,0010010000101010b ;242A

or ax,1111111100000000b ;FF2A

RET

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Modify Program 01 to store two 16-bit values in AX and BX register and perform AND operation.

org 100h

MOV AX,01100001b ;61h

MOV BX,11011111b; DF

AND AX, BX ;41h

RET

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Modify Program 02 to store two 16-bit values in AX and BX register and perform OR operation.

org 100h

MOV Ax,41h ;00000000 01000001b

MOV Bx,20h ;00000000 00100000b

OR Ax, Bx ;61 ;00000000 01100001b

RET

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Modify Program 04 to store two 16-bit values in AX and BX register and perform XOR operation.

org 100h

MOV Ax, 00000000 00000111b ;07

MOV Bx, 00000000 00000010b ;02

XOR Ax,Bx ;0000000 00000101b;04

RET

**Lab Session 08**

Procedure:

Start *Emu8086* by selecting its icon.

Write the following codes in the text editor

Fill the observation tables accordingly

Program 01:

Shift operand1 Left. The number of shifts is set by operand2.

Algorithm:

Shift all bits left, the bit that goes off is set to CF.

Zero bit is inserted to the right-most position.

org 100h

MOV AL, 11100000b

SHL AL, 1

RET

AL 11000000

CF 1

Program 02:

Shift operand1 Right. The number of shifts is set by operand2.

Algorithm:

Shift all bits right, the bit that goes off is set to CF.

Zero bit is inserted to the left-most position.

Org 100h

MOV AL, 00000111b

SHR AL, 1

RET

AL 00000011

CF 1

Program 03:

Shift Arithmetic operand1 Left. The number of shifts is set by operand2.

Algorithm:

Shift all bits left, the bit that goes off is set to CF.

Zero bit is inserted to the right-most position.

Org 100h

MOV AL, 0E0h

SAL AL, 1

RET

|  |  |
| --- | --- |
| AL | 11000000 |
| CF | 1 |

Program 04:

Shift Arithmetic operand1 Right. The number of shifts is set by operand2.

Algorithm:

Shift all bits right, the bit that goes off is set to CF.

The sign bit that is inserted to the left-most position has the same value as before shift.

Org 100h

MOV AL, 0E0h

SAR AL, 1

MOV BL, 4Ch

SAR BL, 1

RET

|  |  |
| --- | --- |
| AL | 11110000 |
| CF | 0 |

Program 05:

Rotate operand1 left. The number of rotates is set by operand2.

Algorithm:

shift all bits left, the bit that goes

off is set to CF and the same bit is

inserted to the right-most position.

Org 100h

MOV AL, 1Ch

ROL AL, 1

RET

AL 00111000

CF 0

Program 06:

Rotate operand1 right. The number of rotates is

set by operand2.

Algorithm:

shift all bits right, the bit that

goes off is set to CF and the same bit

is inserted to the left-most position.

Org 100h

MOV AL, 1Ch

ROR AL, 1

RET

AL 00001110

CF 0

Exercise:

1. In the following code sequence, show the value of AL after each shift or rotate instruction has executed.

mov al,0d4h ; AL= 11010100

shr al,1 ; AL=01101010

mov al,0d4h

sar al,1 ; AL= 11101010

mov al,0d4h

sar al,4 ; AL= 11111101

mov al,0d4h

shl al,1 ; AL= 10101000

mov al,0d4h

sal al,1 ; AL= 10101000

mov al,0d4h

sal al,4 ; AL= 01000000

mov al,0d4h

ror al,1 ; AL= 01101010

mov al,0d4h

ror al,1 ; AL= 01101010

mov al,0d4h

ror al,3 ; AL= 10011010

mov al,0d4h

rol al,7 ; AL= 01101010

mov al,0d4h

rol al,1 ; AL= 10101001

1. Which instruction shifts each bit in an operand to the left and copies the highest bit into both the Carry flag and the lowest bit position?

The ROL (rotate left) instruction shifts each bit to the left. The highest bit is copied into the Carry flag and the lowest bit position.

1. Write a program that shifts multiple bytes using variables.

byte1 db 3Bh

byte2 db 46h

byte3 db 0FFh

Then after the execution of program the values should be:

byte1 = 03h

byte2 = 90h

byte3 = 6Fh

**Program:**

org 100h

.data

byte1 db 3Bh

byte2 db 46H

byte3 db 0FFh

.code

mov cx,4

mov al,byte1

mov bl,byte2

mov dl,byte3

L1:

shr al,1

rcr bl,1

rcr dl,1

loop L1

ret

1. Which instruction shifts each bit to the right, copies the lowest bit into the Carry flag, and copies the Carry flag into the highest bit position?

The ROR instruction shifts each bit to the right, with the lowest bit copied in the Carry flag and into the highest bit.

1. Write a single instruction that multiplies the operand by 2n. (n= no. of shifts)

Shifting left n bits multiplies the operand by 2n

mov dl,5

shl dl,2 ; DL = 20

1. Write a single instruction that divides the operand by 2n. (n= no. of shifts)

Shifting right n bits divides the operand by 2n

mov dl,80

shr dl,1 ; DL = 40

shr dl,2 ; DL = 10

**Lab Session 09**

Procedure:

**Example 1:**

ORG 100h

MOV AX, 5 ; set AX to 5.

MOV BX, 2 ; set BX to 2.

JMP calc ; go to 'calc'.

back: JMP stop ; go to 'stop'.

calc:

ADD AX, BX ; add BX to AX.

JMP back ; go 'back'.

stop:

RET ; return to operating system.

END ; directive to stop the compiler.

**Example 2:**

include emu8086.inc

ORG 100h

MOV AL, 25 ; set AL to 25.

MOV BL, 10 ; set BL to 10.

CMP AL, BL ; compare AL - BL.

JE equal ; jump if AL = BL (ZF = 1).

PUTC 'N' ; if it gets here, then AL <> BL,

JMP stop ; so print 'N', and jump to stop.

equal: ; if gets here,

PUTC 'Y' ; then AL = BL, so print 'Y'.

stop:

RET ; gets here no matter what.

END

**Exercise**

1. What is the function of PUTC instruction?

Macro with 1 parameter, prints out an ASCII char at current cursor position.

1. Try Example 2 with different values for **AL** and **BL**, open flags by clicking on [**FLAGS**] button, use [**Single Step**] and see what happens, don't forget to recompile and reload after every change.

**include emu8086.inc**

ORG 100h

MOV AL, 10 ; set AL to 25.

MOV BL, 10 ; set BL to 10.

CMP AL, BL ; compare AL - BL.

JE equal ; jump if AL = BL (ZF = 1).

PUTC 'N' ; if it gets here, then AL <> BL,

JMP stop ; so print 'N', and jump to stop.

equal: ; if gets here,

PUTC 'Y' ; then AL = BL, so print 'Y'.

stop:

RET ; gets here no matter what.

END

**include emu8086.inc**

ORG 100h

MOV AL, 9 ; set AL to 25.

MOV BL, 10 ; set BL to 10.

CMP AL, BL ; compare AL - BL.

JE equal ; jump if AL = BL (ZF = 1).

PUTC 'N' ; if it gets here, then AL <> BL,

JMP stop ; so print 'N', and jump to stop.

equal: ; if gets here,

PUTC 'Y' ; then AL = BL, so print 'Y'.

stop:

RET ; gets here no matter what.

END

1. Repeat example 2 for JNE, JZ, JNZ.

**JNE:**

include emu8086.inc

ORG 100h

MOV AL, 25 ; set AL to 25.

MOV BL, 10 ; set BL to 10.

CMP AL, BL ; compare AL - BL.

JNE notEqual ; jump if AL = BL (ZF = 1).

PUTC 'Y' ; if it gets here, then AL <> BL,

JMP stop ; so print 'N', and jump to stop.

notEqual: ; if gets here,

PUTC 'N' ; then AL = BL, so print 'Y'.

stop:

RET ; gets here no matter what.

END

**JZ:**

include emu8086.inc

ORG 100h

MOV AL, 10 ; set AL to 25.

MOV BL, 10 ; set BL to 10.

CMP AL, BL ; compare AL - BL.

JZ equal ; jump if AL = BL (ZF = 1).

PUTC 'N' ; if it gets here, then AL <> BL,

JMP stop ; so print 'N', and jump to stop.

equal: ; if gets here,

PUTC 'Y' ; then AL = BL, so print 'Y'.

stop:

RET ; gets here no matter what.

END

**JNZ**

include emu8086.inc

ORG 100h

MOV AL, 10 ; set AL to 25.

MOV BL, 10 ; set BL to 10.

CMP AL, BL ; compare AL - BL.

JNZ notEqual ; jump if AL = BL (ZF = 1).

PUTC 'Y' ; if it gets here, then AL <> BL,

JMP stop ; so print 'N', and jump to stop.

notEqual: ; if gets here,

PUTC 'N' ; then AL = BL, so print 'Y'.

stop:

RET ; gets here no matter what.

END

**Lab Session 10**

**Exercise**

1. Write a program to display the message ‘Ding! Ding! Ding!’ and output ASCII code 7 three times. (ASCII code 7 is the Beep character. It causes your machine to beep! )

org 100h

.DATA

message DB 'Ding!Ding!Ding' ,7,7,7,'$'

.CODE

mov ax,@data

mov ds,ax

mov ah,9

lea dx,message

int 21h

ret

**LAB SESSION 11**

**Exercise**

1. Differentiate between procedures and macros?

|  |  |
| --- | --- |
| Procedure | Macros |
| When you want to use a procedure you should use **CALL** instruction | When you want to use a macro, you can just type its name. |
| Procedure is located at some specific address in memory | Macro is expanded directly in program's code. |
| We use **stack** or any general purpose registers to pass parameters to procedure. | To pass parameters to macro, you can just type them after the macro name. |

1. Write a program to add 2 digits using procedure.

org 100h

.code

mov ax,3h

mov bx,6h

call myproc

myproc Proc

add ax,bx

ret

ENDP

end

1. Write a program to add 2 digits using a Macro.

org 100h

MyMacro MACRO p1, p2

MOV AX, p1

MOV BX, p2

ADD AX,BX

ENDM

MyMacro 6,3

RET

end