**Computer Organization and Assembly Language**

|  |  |
| --- | --- |
| ***Lab 3*** | |
| **Topic** | 1. Addressing Modes with variations. 2. Flags (CF,ZF,SF) |

**PART 1**

**Types of Registers:-**

The registers are grouped into three categories:-

1. **General Purpose registers**
   1. *Data registers*
      1. ***AX*** is the primary accumulator.
      2. ***BX*** is known as the base register.
      3. ***CX*** is known as the count register.
      4. ***DX*** is known as the data register.
   2. *Pointer registers*
      1. Instruction Pointer ***IP***
      2. Stack Pointer ***SP***
      3. Base Pointer ***BP***
   3. *Index registers*
      1. Source Index ***SI***
      2. Destination Index ***DI***
2. **Control registers**
   1. Instruction Pointer and Flag register
3. **Segment registers**
   1. Code Segment ***CS***
   2. Data Segment ***DS***
   3. Stack Segment ***SS***
   4. Extra Segment ***ES***

**Types of variables**

|  |  |  |
| --- | --- | --- |
| **Type** | **No. of bits** | **Example declaration:** |
| Byte | 8 | Num1: db 43 |
| Word=> 2 bytes | 16 | Num2: dw 0xABFF |
| double word=> 2 words | 32 | Num3: dd 0xABCDEF56 |

***Note: size of both operands must be same for any type of instruction.***

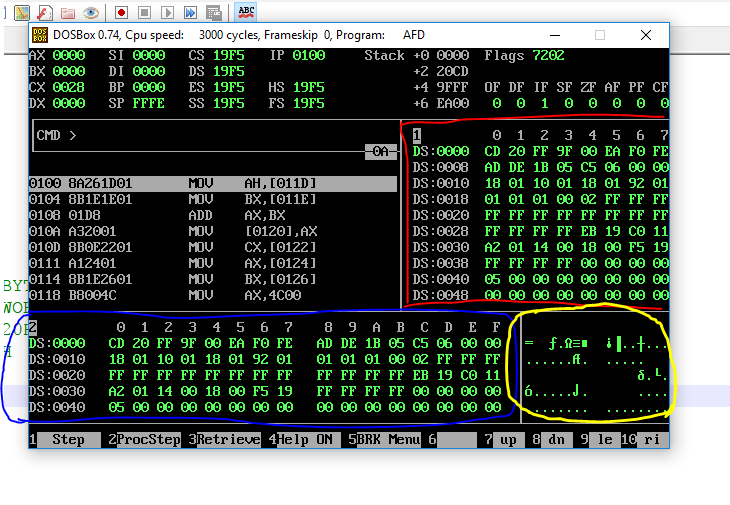
For example:

Mov ax,dh ;is wrong because destination is 2 bytes and source is 1 byte.

***Viewing memory in DOSBOX***

Areas highlighted in red( memory 1) “m1” and blue (memory 2) “m2” are showing the memory contents. *Note:* Two copies of the same memory is displayed in the given windows.

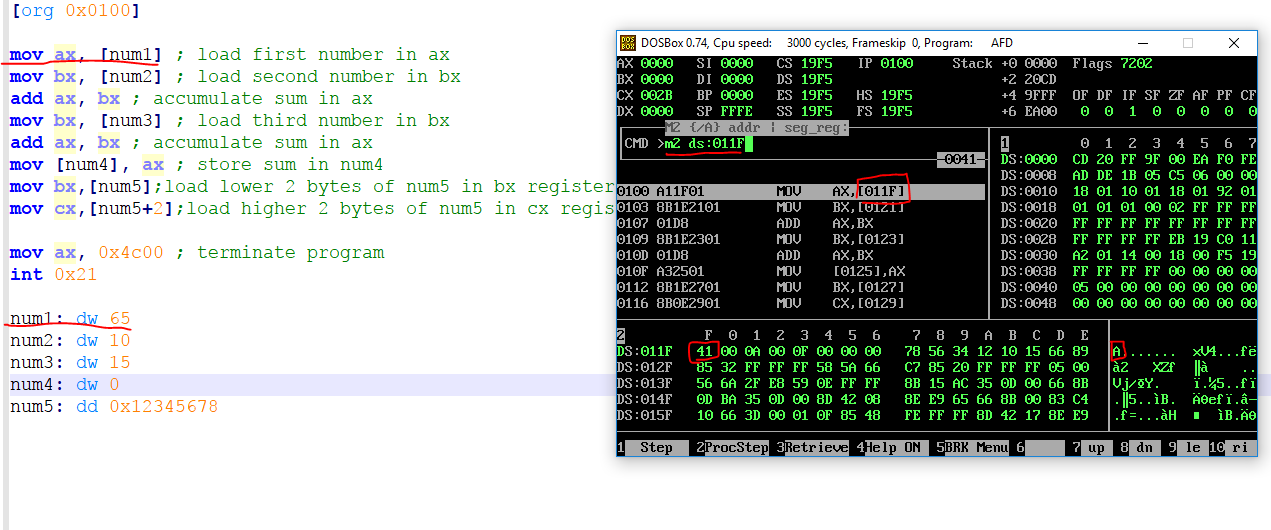
Area highlighted with yellow is showing the ascii values of the contents displayed in the memory m2.

******

***Viewing sample variable in memory.***

* To view memory from window m2 run the command “m2 ds:***Addressofvariable***” example: m2 ds:011F
* A variable with name “num1” is initialized at memory location 11F with value 65 decimal.

41 hex = 65 decimal is the ascii of “A”.



Types of Addressing Modes

|  |  |
| --- | --- |
| **Direct**  A fixed offset is given in brackets and the memory at that offset is accessed. For example “mov [1234], ax” stores the contents of the AX registers in two bytes starting at address 1234 in the current data segment. The instruction “mov [1234], al” stores the contents of the AL register in the byte at offset 1234. | * Mov ax,[num1] ;reading * Mov [num2],ax ;writing |
| **Based Register Indirect**  A base register is used in brackets and the actual address accessed depends on the value contained in that register. For example “mov [bx], ax” moves the two byte contents of the AX register to the address contained in the BX register in the current data segment. The instruction “mov [bp], al” moves the one byte content of the AL register to the address contained in the BP register in the current stack segment. | * Mov bx,var * Mov cx,[bx] * Mov [bx],ax |
| **Indexed Register Indirect**  An index register is used in brackets and the actual address accessed depends on the value contained in that register. For example “mov [si], ax” moves the contents of the AX register to the word starting at address contained in SI in the current data segment. The instruction “mov [di], ax” moves the word contained in AX to the offset stored in DI in the current data  segment. | * Mov si,var1 * Mov di,var2 * Mov [si], ax * Mov [di],bx * Mov cx,[si] * Mov dx,[di] |
| **Based Register Indirect + Offset**  A base register is used with a constant offset in this addressing mode. The value contained in the base register is added with the constant offset to get the effective address. For example “mov [bx+300], ax” stores the word contained in AX at the offset attained by adding 300 to BX in the current data segment. The instruction “mov [bp+300], ax” stores the word in AX to the offset attained by adding 300 to BP in the current stack segment. | * mov [bx+3], ax * mov cl,[bp+5] |
| **Indexed Register Indirect + Offset**  An index register is used with a constant offset in this addressing mode. The value contained in the index register is added with the constant offset to get the effective address. For example “mov [si+300], ax” moves the word contained in AX to the offset attained by adding 300 to SI in the current data segment and the instruction “mov [di+300], al” moves the byte contained in AL to the offset attained by adding 300 to DI in the current data segment. | * Mov [si+2],al * Mov bl,[di+4] |
| **Base + Index**  One base and one index register is used in this addressing mode. The value of the base register and the index register are added together to get the effective address. For example “mov [bx+si], ax” moves the word contained in the AX register to offset attained by adding BX and SI in the current data segment. The instruction “mov [bp+di], al” moves the byte contained in AL to the offset attained by adding BP and DI in the current stack segment. Observe that the default segment is based on the base register and not on the index register. This is why base registers and index registers are named separately. Other examples are “mov [bx+di], ax” and “mov [bp+si], ax.” This method can be used to access a two dimensional array such that one dimension is in a base register and the other is in an index register. | * mov [bx+si], ax * mov al,[bp+di] |
| **Base + Index + Offset**  This is the most complex addressing method and is relatively infrequently used. A base register, an index register, and a constant offset are all used in this addressing mode. The values of the base register, the index register, and the constant offset are all added together to get the effective address. For example “mov [bx+si+300], ax” moves the word contents of the AX register to the word in memory starting at offset attained by adding BX, SI, and 300 in the current data segment. Default segment association is again based on the base register. It might be used with the array base of a two dimensional array as the constant offset, one dimension in the base register and the other in the index register. This way all calculation of location of the desired element has been delegated to the processor. | * mov [bx+si+100], ax |

Execute every part of Question 1 in ***Nasm with Dosbox*** and observe the memory variables and register values.

Q1.

1. Create simple variables of type byte, word and double word.

[org 0x0100]

mov ah,[var1]

mov bx,[var2]

add ax,bx

mov [var3],ax

mov cx,[var4]

mov ax,[var5]

mov bx,[var5+2]

mov ax,0x4c00

int 21h

var1: dw 10

var2: dw 20

var3: dw 20H

var4: dw 40H

var5: dd 0xABCDEF56

Answer: 4C00

1. Direct addressing of variables by using the address of only one variable

[org 0x0100]

mov bl,[v]

add bl,[v-1]

add bl,[v+1]

mov ax,0x4c00

int 21h

u: db 34h

v: db 35h

w: db 36h

Answer : 9F

1. Indirect accessing of memory of byte size

[org 0x100]

xor ax,ax

xor bx,bx

xor cx,cx

xor dx,dx

mov bx, var1

mov al,[bx]

mov cl,[bx+1]

mov ch,[bx+2]

sub ch,cl

mov ax,0x4c00

int 21h

var1: db 10

var2: db 55h

var3: db 13

1. Copy this code and observe what’s wrong with that and correct it.

[org 0x100]

MOV ax,[var1]

MOV bx,[var2]

add ax,bx

MOV [var3],ax

MOV cx,[var4]

mov ax,0x4c00

int 21h

var1: db 5

var2: dw 6

var3: dw 7

var4: dw 8

1. Indirect accessing of Word size

[org 0x100]

xor ax,ax

xor bx,bx

xor cx,cx

xor dx,dx

mov bx, var1

mov ax,[bx]

mov cx,[bx+2]

mov dx,[bx+4]

add ax,dx

mov [bx+2],ax

mov ax,0x4c00

int 21h

var1: dw 10

var2: dw 20h

var3: dw 13

1. Reading and writing in memory through indirect memory address.

[org 0x100]

xor ax,ax

xor bx,bx

xor cx,cx

xor dx,dx

mov bx, var1

mov ax,[bx]

mov cx,[bx+2]

add ax,cx

mov [bx],ax

mov [bx+2], 0

mov dx,[bx+3]

add dx,[bx+3]

mov [bx+3],dx

mov ax,0x4c00

int 21h

var1: dw 60

var2: db 5

var3: dw 100

**Q2. Write down the values of Carry Flag, Sign Flag and Zero Flags. Justify your values in flags with explanations.**

*Run these codes one by one.*

1. mov ax,10

mov bx,10

|  |  |
| --- | --- |
| **CF** | **0** |
| **SF** | **0** |
| **ZF** | **1** |

sub bx,ax

*Explanation: Because Answer Stored in bx will be zero after Subtraction.*

1. mov ax,200

mov bx,100

add ax,bx

|  |  |
| --- | --- |
| **CF** | **0** |
| **SF** | **0** |
| **ZF** | **0** |

*Explanation: Because adding 100 to 200 Create un-signed Number so no carry /borrow needed.*

1. mov ax,-50

|  |  |
| --- | --- |
| **CF** | **0** |
| **SF** | **1** |
| **ZF** | **0** |

*Explanation: Because -50 is signed number so answer is also signed*

d) mov ax,100

mov bx,200

sub ax,bx

|  |  |
| --- | --- |
| **CF** | **1** |
| **SF** | **1** |
| **ZF** | **0** |

*Explanation: Because subtracting 100 from 200 will be a negative number and carry flag will be needed*