

**Namal University Mianwali**

**Computer Science Department**

**Coal lab 04**

**Submitted By:**

Name Anam Fatima

Roll-No NUM-BSCS-23-06

Subject Coal

**Submitted To:**

Mr. Muzamil Ahmed

**Date:**10/20/2024

**Task 1: Develop two assembly language programs based on the provided source codes, ensuring they display register values and match output values with corresponding comments, and document your observations.**

**Code 1:**

include \masm32\include\masm32rt.inc

.data

    val1 WORD 1000h

    val2 WORD 2000h

    arrayB BYTE 10h,20h,30h,40h,50h

    arrayW WORD 100h,200h,300h

    arrayD DWORD 10000h,20000h

    newline  db 13,10,0

    buffer dword 32 dup(0)

    outputmsg db"The result is:" ,0

.code

start:

    ; Demonstrating MOVZX instruction:

    mov bx,0A69Bh

    movzx eax,bx ; EAX = 0000A69Bh

      invoke dwtoa, eax, addr buffer

         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    movzx edx,bl ; EDX = 0000009Bh

    invoke dwtoa, edx, addr buffer

         invoke StdOut, addr newline

         invoke StdOut, addr outputmsg

         invoke StdOut, addr buffer

    movzx cx,bl ; CX = 009Bh

        movzx ecx,cx

         invoke dwtoa, ecx, addr buffer

         invoke StdOut, addr newline

         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    ; Demonstrating MOVSX instruction:

    mov bx,0A69Bh

    movsx eax,bx ; EAX = FFFFA69Bh

      invoke dwtoa, eax, addr buffer

         invoke StdOut, addr newline

         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    movsx edx,bl ; EDX = FFFFFF9Bh

      invoke dwtoa, edx, addr buffer

         invoke StdOut, addr newline

         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    mov bl,7Bh

    movsx cx,bl ; CX = 007Bh

      movzx ecx,cx

      invoke dwtoa, ecx, addr buffer

         invoke StdOut, addr newline

         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    ; Memory-to-memory exchange:

    mov ax,val1 ; AX = 1000h

    xchg ax,val2 ; AX=2000h, val2=1000h

    mov val1,ax ; val1 = 2000h

    movzx eax,val1

       invoke dwtoa, eax, addr buffer

         invoke StdOut, addr newline

         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    ; Direct-Offset Addressing (byte array):

    mov al,arrayB ; AL = 10h

    mov al,[arrayB+1] ; AL = 20h

    movzx ax,al

    movzx eax,ax

       invoke dwtoa, eax, addr buffer

         invoke StdOut, addr newline

         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    mov al,[arrayB+2] ; AL = 30h

     movzx ax,al

    movzx eax,ax

       invoke dwtoa, eax, addr buffer

         invoke StdOut, addr newline

         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    ; Direct-Offset Addressing (word array):

    mov ax,arrayW ; AX = 100h

    mov ax,[arrayW+2] ; AX = 200h

     movzx ax,al

    movzx eax,ax

       invoke dwtoa, eax, addr buffer

         invoke StdOut, addr newline

         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    ; Direct-Offset Addressing (doubleword array):

    mov eax,arrayD ; EAX = 10000h

    mov eax,[arrayD+4] ; EAX = 20000h

    mov eax,[arrayD+4] ; EAX = 20000h

       invoke dwtoa, eax, addr buffer

         invoke StdOut, addr newline

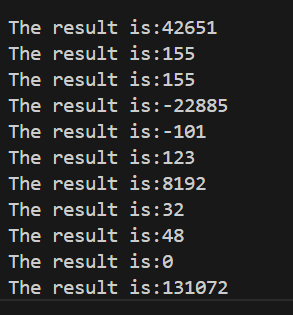
         invoke StdOut, addr outputmsg

             invoke StdOut, addr buffer

    invoke ExitProcess, 0

end start

**Output:**

****

**Observation:**

1. **Using MOVZX Instruction:**

* MOVZX transfers data from a smaller register to a larger one. It fills the leftover space with zeroes.  
  When the program shifts the 16-bit value inside BX into EAX, it fills up EAX with zeroes at the leftover space that makes it 0000A69B.
* Similarly, when the program copies the 8 bit inside BL to EDX, it fills up EDX with zeroes up to 0000009B.
* When these are stdout, the result is converted into the decimal number.

**2. Using MOVSX Instruction:**

* MOVSX also copies from a narrower register to a wider one but maintains the sign bit - in other words, its positive or negative value.  
  When a program moves a 16-bit value into EAX using MOVSX it fills in the additional space with its sign; so A69B is changed to FFFFA69B. As you can see, this is a negative value.

**3.Swapping Values Using XCHG:**

* The XCHG instruction swaps the values of two locations.  
  Before, the value of AX is 1000h, and the value of val2 is 2000h. After the swap, AX holds 2000h, and val2 holds 1000h.

**4. Accessing Elements in a Byte Array:**

* The above program shows how one can access individual elements from a byte array (arrayB).  
  It will read the first, second, and third bytes; convert them into 32-bit values and write them to the console.

**5. Accessing Elements in Word Array:**

* The same as above for the word array (arrayW).  
  The program accesses the first and second elements (100h and 200h) convert them into higher values and prints.

**6. Accessing Elements in Doubleword Array:**

The doubleword array (arrayD) is accessed the same as above.  
The program accesses the first and second elements (10000h and 20000h) and prints them as it is.

**Code 2:**

include \masm32\include\masm32rt.inc

.data

    Rval SDWORD ?

    Xval SDWORD 26

    Yval SDWORD 30

    Zval SDWORD 40

    temp SDWORD ?

    newline db 13,10,0

     buffer dword 32 dup(0)

      outputmsg db"The result is:" ,0

.code

start:

    ; INC and DEC

    mov ax,1000h

    inc ax ; 1001h

    movzx eax,ax

    mov ebx,eax

    invoke dwtoa, ebx, addr buffer

    invoke StdOut, addr outputmsg

    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    dec bx ; 1000h

    movzx ebx,bx

    invoke dwtoa, ebx, addr buffer

    invoke StdOut, addr outputmsg

    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    ; Expression: Rval = -Xval + (Yval - Zval)

    mov eax,Xval

    neg eax ; -26

    mov temp,eax

    invoke dwtoa, eax, addr buffer

    invoke StdOut, addr outputmsg

    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    mov ebx,Yval

    sub ebx,Zval ; -10

    invoke dwtoa, ebx, addr buffer

    invoke StdOut, addr outputmsg

    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    add ebx,temp

    mov Rval,ebx ; -36

    invoke dwtoa, Rval, addr buffer

    invoke StdOut, addr outputmsg

    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    ; Zero flag example:

    mov cx,1

    sub cx,1 ; ZF = 1

    lahf

    shr ah,6

    movzx eax,ah

    invoke dwtoa, eax, addr buffer

    invoke StdOut, addr outputmsg

    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    mov ax,0FFFFh

    inc ax ; ZF = 1

    lahf

    shr ah,6

    movzx eax,ah

    invoke dwtoa, eax, addr buffer

    invoke StdOut, addr outputmsg

    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    ; Sign flag example:

    mov cx,0

    sub cx,1 ; SF = 1

    lahf

    shr ah,7

    movzx eax,ah

    invoke dwtoa, eax, addr buffer

    invoke StdOut, addr outputmsg

    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    mov ax,7FFFh

    add ax,2 ; SF = 1

    lahf

    shr ah,7

    movzx eax,ah

    invoke dwtoa, eax, addr buffer

    invoke StdOut, addr outputmsg

    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    ; Carry flag example:

    mov al,0FFh

    add al,1 ; CF = 1, AL = 00

    lahf

    and ah,1

    movzx eax,ah

    invoke dwtoa, eax, addr buffer

    invoke StdOut, addr outputmsg

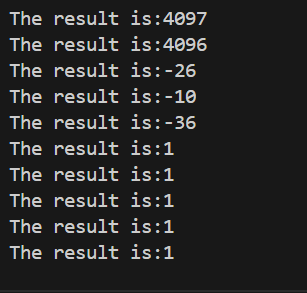
    invoke StdOut, addr buffer

    invoke StdOut, addr newline

    INVOKE ExitProcess,0

end start

**Output:**



**Observation:**

**1. Using INC (Increment) and DEC (Decrement):**

* Loads number 1000h or 4096 in decimal in AX  
  **INC :** Increments it by 1, so AX turns to be 1001h or 4097, transfers that number to EBX, and echoes it again,  
  **DEC :** It then reduces BX by 1, so BX ends to be 1000h or 4096, then echoes again.

**2. Evaluating an Expression:**

* The program calculates Rval = -Xval + (Yval - Zval) using only a few arithmetic operations.  
  Xval = 26, therefore, by the application of NEG, -Xval = -26  
  It proceeds to calculate Yval - Zval = 30 - 40 = -10  
  Next it adds -26 and -10 to get -36 and puts the result into Rval. The output values generated at each step are printed.

**3. Zero Flag (ZF) Example:**

* The Zero Flag (ZF) is used to tell if the result of an operation is zero.  
  Thus, the program first does a subtract 1 from 1, which means the result is 0; therefore, the Zero Flag gets set to 1. It extracts and prints this flag.  
  Then it adds 1 to 0FFFFh, which is 65535 in decimal; when it does so, it wraps around to 0000h. Since the result is zero, the Zero Flag gets set again and then printed.

**4. Sign Flag (SF) Example:**

* The Sign Flag (SF) is set if the result is negative.  
  It first subtracts 1 from 0, obtaining FFFFh (-1 in signed notation). So the Sign Flag gets set to 1. It reads and writes this flag.  
  Then it adds 2 to 7FFFh (32767), getting 8001h, a negative in signed notation, so the Sign Flag gets set and written once again.

**5. Carry Flag (CF) Example:**

* Carry Flag (CF) is set when the result of an operation exceeds the capacity that can be stored in a register.  
  The program increments 0FFh, which is equivalent to the base-10 value 255, but there are only 8 bits. The result wraps around to 00 because FFh + 1 = 100h, but only 8 bits are available. The carry flag is set, so it prints 1 because the result exceeded the register, and it prints the flag.

**Task 2:** **Using the XCHG instruction no more than three times, reorder the values in four 8-bit registers from the order A, B, C, D to B, C, D, A.**

**Code:**

include \masm32\include\masm32rt.inc

.data

    Aval byte "A",0

    Bval byte "B",0

    Cval byte "C",0

    Dval byte "D",0

    newline db 13,10,0

    outputmsg db"The values stored before xchang:" ,0

     outputmsg1 db"The values stored after xchang ins:" ,0

.code

start:

invoke StdOut, addr  outputmsg

invoke StdOut, addr newline

invoke StdOut, addr  Aval  ; Output the character

invoke StdOut, addr newline ; Output newline

invoke StdOut, addr Bval  ; Output the character

invoke StdOut, addr newline

invoke StdOut, addr Cval  ; Output the character

invoke StdOut, addr newline

invoke StdOut, addr Dval  ; Output the character

invoke StdOut, addr newline

mov al, Aval

XCHG al, Dval

XCHG al, Cval

XCHG al, Bval

mov Aval,al

invoke StdOut, addr  outputmsg1

invoke StdOut, addr newline

invoke StdOut, addr  Aval  ; Output the character

invoke StdOut, addr newline ; Output newline

invoke StdOut, addr Bval  ; Output the character

invoke StdOut, addr newline

invoke StdOut, addr Cval  ; Output the character

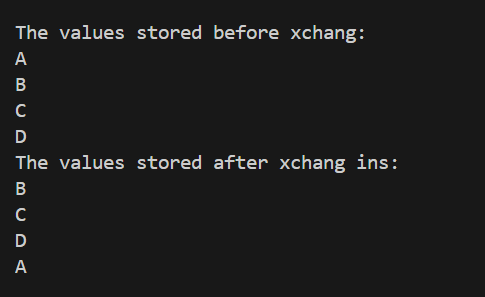
invoke StdOut, addr newline

invoke StdOut, addr Dval  ; Output the character

invoke StdOut, addr newline

    INVOKE ExitProcess,0

end start



**Output:**

**Observation:**

**1. Displaying Values Before XCHG:**

The program first prints the message "The values stored before xchang:" and outputs the characters stored in variables Aval, Bval, Cval, and Dval, each followed by a newline.

* **Initially:**
  + Aval contains "A".
  + Bval contains "B".
  + Cval contains "C".
  + Dval contains "D".

**2. Using XCHG to Swap Values:**

The program then swaps the values of the characters using the XCHG instruction, which exchanges values between registers and memory locations.

* The steps:
  + First, the program moves Aval ("A") into AL (the AL register).
  + **Swap 1:** It swaps the contents of AL (which has "A") with Dval (which has "D"). Now AL = "D" and Dval = "A".
  + **Swap 2:** Then it swaps AL (which now has "D") with Cval (which has "C"). Now AL = "C" and Cval = "D".
  + **Swap 3:** Finally, it swaps AL (which now has "C") with Bval (which has "B"). Now AL = "B" and Bval = "C".
  + After all swaps, it moves "B" from AL back into Aval.

**3. Displaying Values After XCHG:**

After the swaps, the program prints the message "The values stored after xchang ins:" and outputs the new values of Aval, Bval, Cval, and Dval.

* **After the swaps:**
  + Aval contains "B".
  + Bval contains "C".
  + Cval contains "D".
  + Dval contains "A".

**Task 3:** **Implement the following arithmetic expressions in assembly language:**

1. **(Assume that val1, val2, and val3 are 32-bit integer variables)**

**(Assume that val2 and val4 are 16-bit integer variables.)**

include \masm32\include\masm32rt.inc

.data

val1    Dword 56

val2    Dword 29

val3    Dword 21

val4    Dword 7

newline db 13,10,0

result1 db 11 dup(0)

result2 db 11 dup(0)a

result3 db 11 dup(0)

message db "Result after performing operation: ", 0

.code

start:

    mov eax, val2       ; Load val2 from memory into EAX

    neg eax

    mov ebx,eax           ; Negate the value in EAX

    invoke dwtoa, ebx, addr result1

    invoke StdOut, addr message  ; Convert EAX (negated value + 7) to string

    invoke StdOut, addr result1

    invoke StdOut, addr newline      ; Output the string

    add ebx,val4

    invoke dwtoa, ebx, addr result2

    invoke StdOut, addr message  ; Convert EAX (negated value + 7) to string

    invoke StdOut, addr result2     ; Output the string

    invoke StdOut, addr newline

    sub ebx , val3   ; eax = -83

     invoke dwtoa, ebx, addr result2

    invoke StdOut, addr message  ; Convert EAX (negated value + 7) to string

    invoke StdOut, addr result2     ; Output the string

    invoke StdOut, addr newline

    add ebx , val1

    invoke dwtoa, ebx, addr result3

    invoke StdOut, addr message  ; Convert EAX (negated value + 7) to string

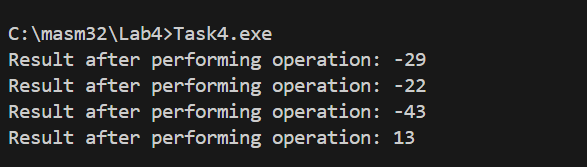
    invoke StdOut, addr result3       ; Output the string

    invoke StdOut, addr newline        ; Output a newline

    invoke ExitProcess, 0

end start

**Output:**



**Observation:**

**1. Load and Negate val2 :**

* The program loads val2 which is 29 into EAX register.  
  It now negates the value in EAX with NEG so EAX becomes -29.  
  The result is copied to EBX and the value -29 written out as a string.

**2. Add val4 (7) to the Negated Value:**

* The program adds val4, which is 7, to the negated value stored in EBX, which is -29.  
  The result is -29 + 7 = -22, which is converted to a string and printed.

**3.Subtract val3 (21):**

* Subtract val3, which is 21, from EBX, which is -22.  
  The result is -22 - 21 = -43, which is converted to a string and printed.

**4. Add val1 (56):**

* Finally, it's added val1 which is 56 to EBX that has -43 in.  
  The value that's stored is -43 + 56 = 13; this value is converted to a character string and printed.

**Code :**

**Part 2:**

include \masm32\include\masm32rt.inc

.data

val2    word 29

val4    word 7

newline db 13,10,0

result1 word ?

message db "Result after performing operation: ", 0

.code

start:

    mov bx, 0       ; Load val2 from memory into EAX

    add bx,val2

    mov ax,bx

    movzx eax,ax         ; Negate the value in EAX

    invoke dwtoa, eax, addr result1

    invoke StdOut, addr message  ; Convert EAX (negated value + 7) to string

    invoke StdOut, addr result1

    invoke StdOut, addr newline      ; Output the string

    sub bx,val4

     mov ax,bx

      movzx eax,ax          ; Negate the value in EAX

    invoke dwtoa, eax, addr result1

    invoke StdOut, addr message  ; Convert EAX (negated value + 7) to string

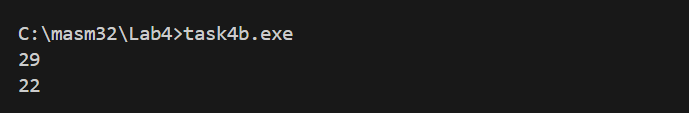
    invoke StdOut, addr result1

    invoke StdOut, addr newline

    invoke ExitProcess, 0

end start

**Output:**



**Observation:**

**1.Adding val2 to BX:**

* The program initialized the BX register to 0.  
  It added the value of val2 which was 29 to BX. So, now BX = 29.
* The value in BX is transferred to AX and then the MOVZX instruction is applied to the 16-bit value in AX to extend it to 32 bits in EAX. This enables the value to be printed.
* The value 29 is converted to a string and printed with the message "Result after performing operation:".

1. **Subtraction of val4 from BX:**

* The program now subtracts val4 whose value is 7 from BX. Thus, BX = 29 - 7 = 22.
* The new value in BX is moved again to AX, and the MOVZX instruction is used in order to extend it to EAX.
* The same message gets printed with the value 22 as a string.

**Task 4: Develop a program that uses INC, DEC, and NEG instructions to manipulate the contents of registers and shows the change in values.**

include \masm32\include\masm32rt.inc

.data

val1    Dword 56

val2    Dword 29

newline db 13,10,0

Result DWORD 0

message db "Result after performing operation: ", 0

.code

start:

    mov eax, val2       ; Load val2 from memory into EAX

    neg eax

    mov ebx,eax           ; Negate the value in EAX

    invoke dwtoa, ebx, addr Result

    invoke StdOut, addr message  ; Convert EAX (negated value + 7) to string

    invoke StdOut, addr Result

    invoke StdOut, addr newline      ; Output the string

    inc ebx

    invoke dwtoa, ebx, addr Result

    invoke StdOut, addr message  ; Convert EAX (negated value + 7) to string

    invoke StdOut, addr Result     ; Output the string

    invoke StdOut, addr newline

    dec ebx  ; eax = -83

     invoke dwtoa, ebx, addr Result

    invoke StdOut, addr message  ; Convert EAX (negated value + 7) to string

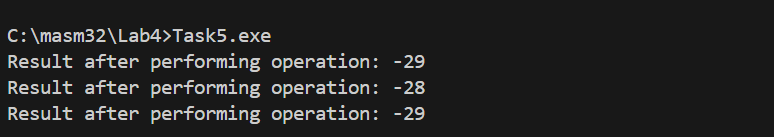
    invoke StdOut, addr Result     ; Output the string

    invoke StdOut, addr newline

    invoke ExitProcess, 0

end start

**Output:**



**Observation:**

**1. Load and Negate val2:**

* The program loads the value of val2 (which is 29) into the EAX register.
* Then it negates (NEG) the value in EAX, which makes EAX equal to -29.
* The negated value is moved to the EBX register.

**2. Print Negated Value:**

* The value in EBX (which is -29) is converted to a string using dwtoa and stored in the Result variable.
* The program then prints the message "Result after performing operation:" followed by the result -29, and then a newline.

**3. Increment and Print EBX:**

* The program increments the value in EBX by 1 (INC EBX), so EBX now becomes -28.
* The new value is converted to a string and printed in the same way as before, with the result -28 followed by a newline.

**4. Decrement and Print EBX:**

* The program decrements the value in EBX by 1 (DEC EBX), so EBX returns to -29.
* This value is converted to a string and printed again, with the result -29 followed by a newline.

------------------------------------------------------------------------------------------------------------

**( The End )**