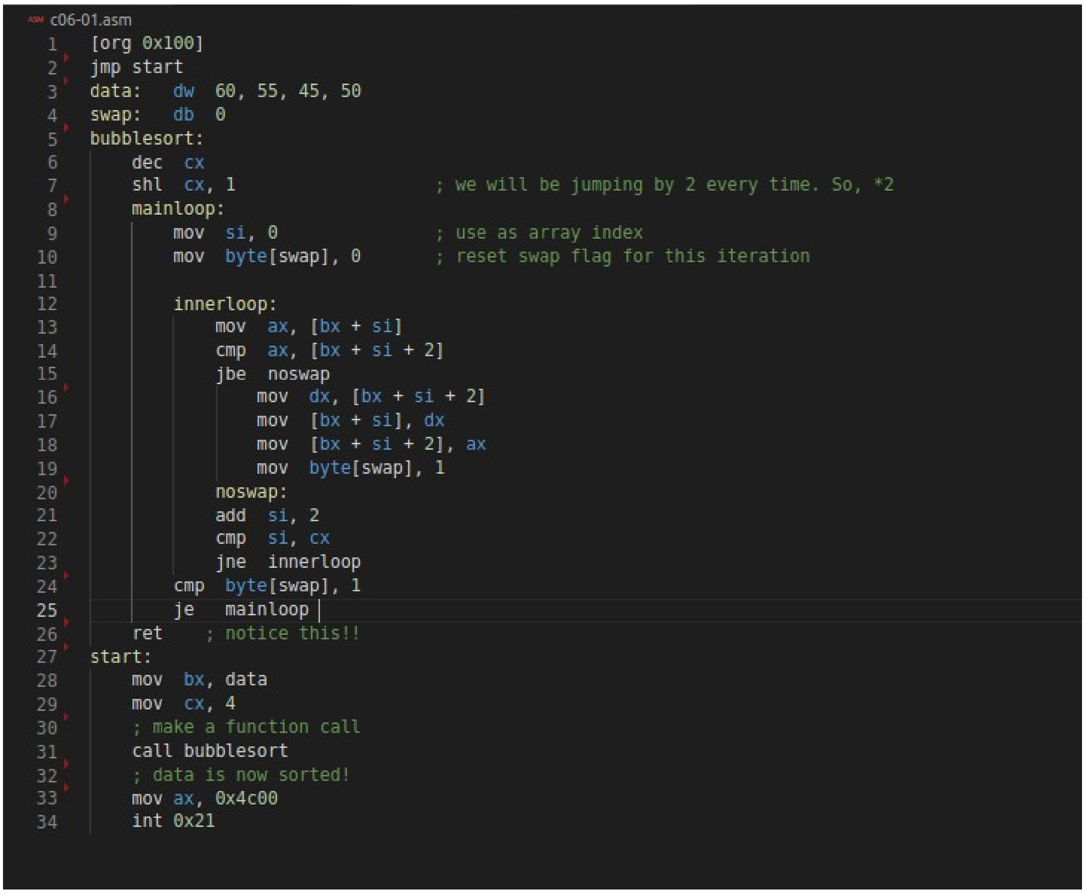
COAL_A_p200165_R8

- ## Introduction:

In Lab No 8 we have studied about subroutines. Let's suppose we want to add two numbers if we have no subroutine or functions we are going to write every time the complete code of addition every time when we want to add two numbers.
With the help of subroutines we write the code we needed more times to run in the subroutine and whenever we want to do task we simply call that subroutine and our that subroutine will do our work and that make code easy to understand and we as a programmer make our life easy.

- ## Practice Code:

The first code we practiced is the code of bubble sort, bubble sort is the common operation that may done many times in real world projects. So it does not make sense we will write every time the complete bubble sort code every time when we need. So what we going to do is that we are going to put the bubble sort code into a subroutine and we call that function when we want to sort our array data.



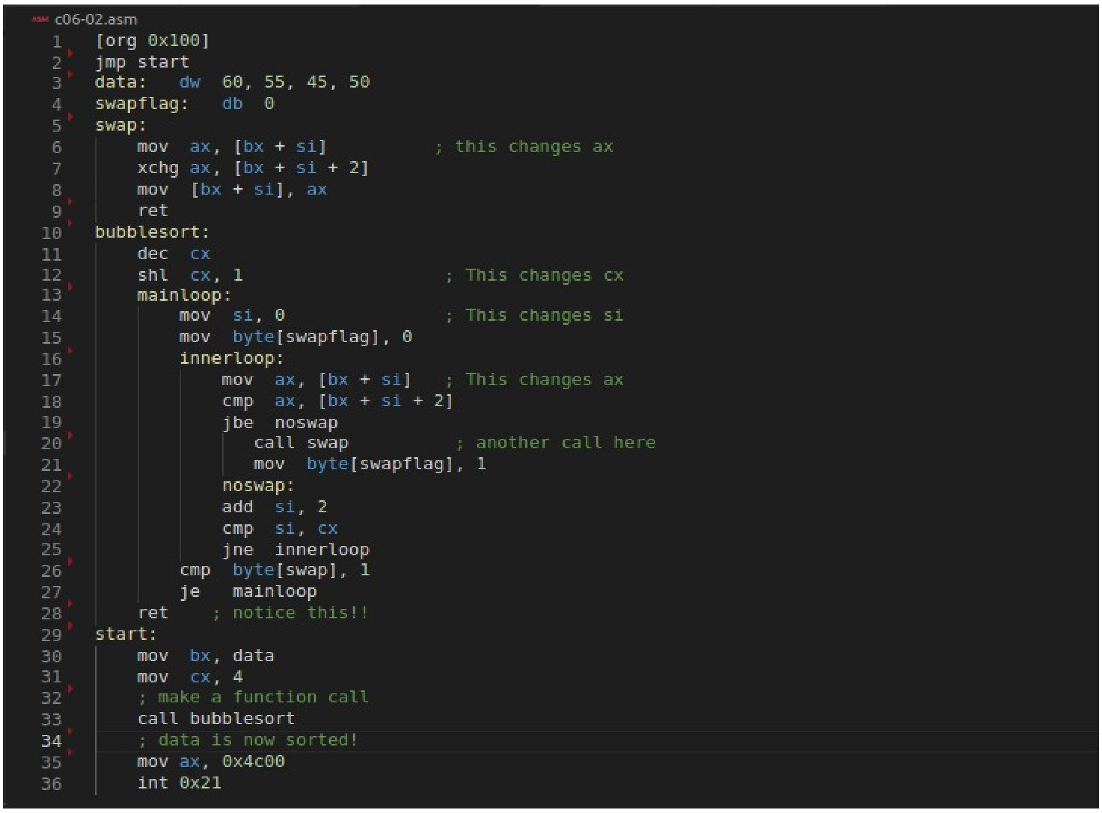
```
[org 0x100]
jmp start
data:   dw  60, 55, 45, 50
swap:   db  0
bubblesort:
    dec  cx
    shl  cx, 1                  ; we will be jumping by 2 every time. So, *2
    mainloop:
        mov  si, 0              ; use as array index
        mov  byte[swap], 0      ; reset swap flag for this iteration

        innerloop:
            mov  ax, [bx + si]
            cmp  ax, [bx + si + 2]
            jbe  noswap
                mov  dx, [bx + si + 2]
                mov  [bx + si], dx
                mov  [bx + si + 2], ax
                mov  byte[swap], 1
            noswap:
            add  si, 2
            cmp  si, cx
            jne  innerloop
        cmp  byte[swap], 1
        je   mainloop
    ret    ; notice this!!
start:
    mov  bx, data
    mov  cx, 4
    ; make a function call
    call bubblesort
    ; data is now sorted!
    mov ax, 0x4c00
    int 0x21
```

- **Call Instruction:**

  call instruction will the next line address value the instruction needed to be executed after calling subroutine into the stack.

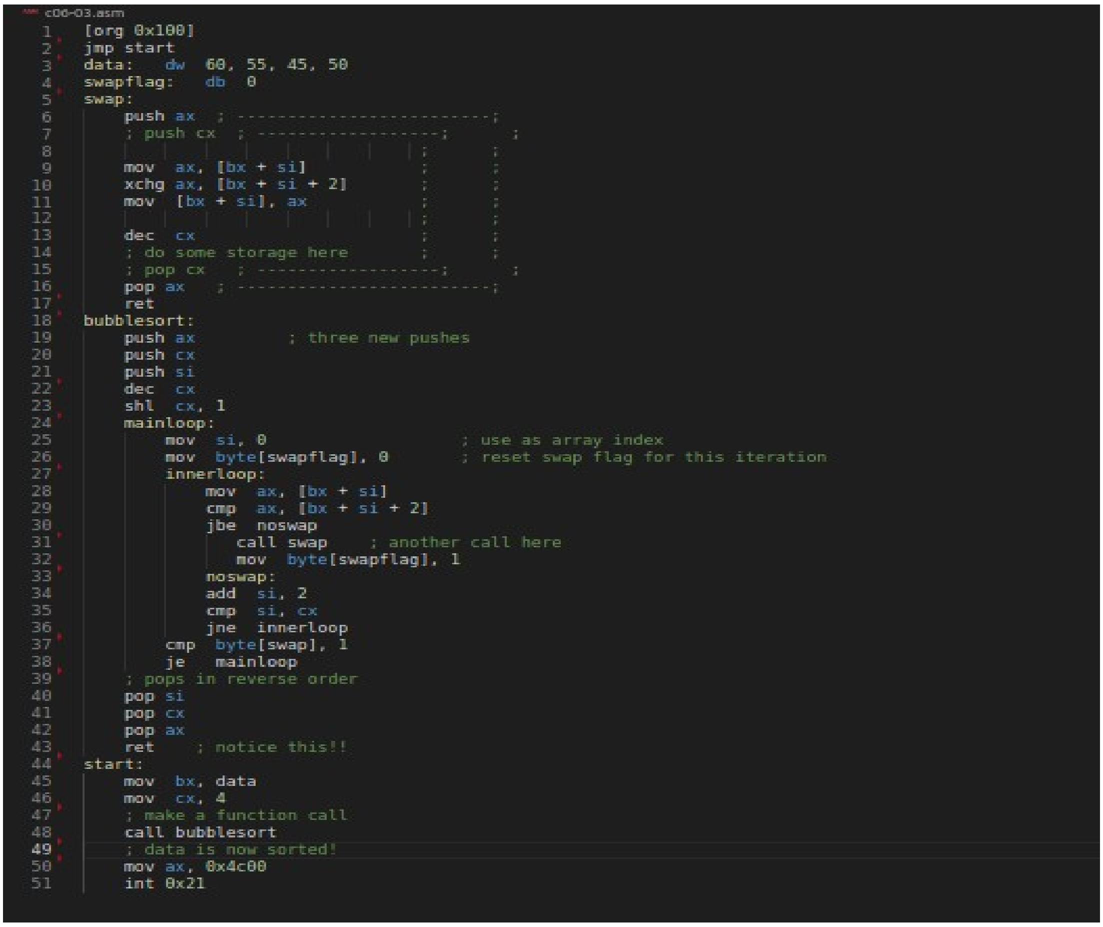
- **Ret Instruction**:

  the ret instruction mov pop 1 value from stack and move that value into the IP(Instruction Pointer).



c06-02.asm

```
[org 0x100]
jmp start
data:   dw  60, 55, 45, 50
swapflag:   db  0
swap:
    mov  ax, [bx + si]          ; this changes ax
    xchg ax, [bx + si + 2]
    mov  [bx + si], ax
    ret
bubblesort:
    dec  cx
    shl  cx, 1                   ; This changes cx
    mainloop:
        mov  si, 0               ; This changes si
        mov  byte[swapflag], 0
        innerloop:
            mov  ax, [bx + si]   ; This changes ax
            cmp  ax, [bx + si + 2]
            jbe  noswap
               call swap          ; another call here
               mov  byte[swapflag], 1
            noswap:
            add  si, 2
            cmp  si, cx
            jne  innerloop
        cmp  byte[swap], 1
        je   mainloop
    ret    ; notice this!!
start:
    mov  bx, data
    mov  cx, 4
    ; make a function call
    call bubblesort
    ; data is now sorted!
    mov ax, 0x4c00
    int 0x21
```



c06-03.asm

```
[org 0x100]
jmp start
data:   dw  60, 55, 45, 50
swapflag:   db  0
swap:
    push ax  ; -------------------------;
    ; push cx  ; ------------------;      ;
                                 ;      ;
    mov  ax, [bx + si]           ;      ;
    xchg ax, [bx + si + 2]       ;      ;
    mov  [bx + si], ax           ;      ;
                                 ;      ;
    dec  cx                      ;      ;
    ; do some storage here       ;      ;
    ; pop cx   ; ------------------;      ;
    pop ax   ; -------------------------;
    ret
bubblesort:
    push ax          ; three new pushes
    push cx
    push si
    dec  cx
    shl  cx, 1
    mainloop:
        mov  si, 0                  ; use as array index
        mov  byte[swapflag], 0      ; reset swap flag for this iteration
        innerloop:
            mov  ax, [bx + si]
            cmp  ax, [bx + si + 2]
            jbe  noswap
               call swap    ; another call here
               mov  byte[swapflag], 1
            noswap:
            add  si, 2
            cmp  si, cx
            jne  innerloop
        cmp  byte[swap], 1
        je   mainloop
    ; pops in reverse order
    pop si
    pop cx
    pop ax
    ret    ; notice this!!
start:
    mov  bx, data
    mov  cx, 4
    ; make a function call
    call bubblesort
    ; data is now sorted!
    mov ax, 0x4c00
    int 0x21
```

PUSH INSTRUCTION:

The push instruction will push the value to the stack, and it push a word on stack. The idea behind the pushing values on stack is that sometimes we need some registers to work within the function and we do not need that our function causes the value of registers to change when subroutine end. The basic idea is that we want to hold the concept of abstraction so that the abstraction concept not break for this we used stack and at end we poped all the pushed value to complete the concept of abstraction.

POP INSTRUCTION:

The pop instruction pop or get one value from stack and move that value to given operand and move the stack pointer +2 .

Local Variables:

By Using Push Instruction we have implemented the local variables.

Name : Jawad Ahmed
Roll NO: 20P-0165
Section : 3A