

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.

```
c06-01.asm
1  [org 0x100]
2  jmp start
3  data: dw 60, 55, 45, 50
4  swap: db 0
5  bubblesort:
6      dec cx
7      shl cx, 1          ; we will be jumping by 2 every time. So, *2
8      mainloop:
9          mov si, 0      ; use as array index
10         mov byte[swap], 0 ; reset swap flag for this iteration
11
12         innerloop:
13             mov ax, [bx + si]
14             cmp ax, [bx + si + 2]
15             jbe noswap
16             mov dx, [bx + si + 2]
17             mov [bx + si], dx
18             mov [bx + si + 2], ax
19             mov byte[swap], 1
20         noswap:
21             add si, 2
22             cmp si, cx
23             jne innerloop
24         cmp byte[swap], 1
25         je mainloop
26     ret ; notice this!!
27 start:
28     mov bx, data
29     mov cx, 4
30     ; make a function call
31     call bubblesort
32     ; data is now sorted!
33     mov ax, 0x4c00
34     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
1  [org 0x100]
2  jmp start
3  data: dw 60, 55, 45, 50
4  swapflag: db 0
5  swap:
6      mov ax, [bx + si] ; this changes ax
7      xchg ax, [bx + si + 2]
8      mov [bx + si], ax
9      ret
10 bubblesort:
11     dec cx
12     shl cx, 1 ; This changes cx
13     mainloop:
14         mov si, 0 ; This changes si
15         mov byte[swapflag], 0
16         innerloop:
17             mov ax, [bx + si] ; This changes ax
18             cmp ax, [bx + si + 2]
19             jbe noswap
20             call swap ; another call here
21             mov byte[swapflag], 1
22         noswap:
23             add si, 2
24             cmp si, cx
25             jne innerloop
26         cmp byte[swap], 1
27         je mainloop
28     ret ; notice this!!
29 start:
30     mov bx, data
31     mov cx, 4
32     ; make a function call
33     call bubblesort
34     ; data is now sorted!
35     mov ax, 0x4c00
36     int 0x21

```

```

c06-03.asm
1  [org 0x100]
2  jmp start
3  data: dw 60, 55, 45, 50
4  swapflag: db 0
5  swap:
6      push ax ; -----;
7      ; push cx ; -----;
8
9      mov ax, [bx + si]
10     xchg ax, [bx + si + 2]
11     mov [bx + si], ax
12
13     dec cx
14     ; do some storage here
15     ; pop cx ; -----;
16     pop ax ; -----;
17     ret
18 bubblesort:
19     push ax ; three new pushes
20     push cx
21     push si
22     dec cx
23     shl cx, 1
24     mainloop:
25         mov si, 0 ; use as array index
26         mov byte[swapflag], 0 ; reset swap flag for this iteration
27         innerloop:
28             mov ax, [bx + si]
29             cmp ax, [bx + si + 2]
30             jbe noswap
31             call swap ; another call here
32             mov byte[swapflag], 1
33         noswap:
34             add si, 2
35             cmp si, cx
36             jne innerloop
37         cmp byte[swap], 1
38         je mainloop
39     ; pops in reverse order
40     pop si
41     pop cx
42     pop ax
43     ret ; notice this!!
44 start:
45     mov bx, data
46     mov cx, 4
47     ; make a function call
48     call bubblesort
49     ; data is now sorted!
50     mov ax, 0x4c00
51     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 popped 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
