Computer Organization & Assembly Language (Lecture 05)

Omar Bin Samin

Lecturer

Institute of Management Sciences, Peshawar.

db Issue

- As discussed in previous lecture, using db with registers of 16-bits produces ambiguity in the results. Program discussed in previous lecture is as follows:

```
[org 0x0100]
mov ax,[Tag1]
mov bx,4
add ax,bx

mov ax,0x4c00      ; exit
int 0x21

Tag1: db 2, 1
```

- The issue arises due to use of two consecutive bytes of separate data .i.e., first byte for ‘2’ (the data becomes ‘02’) and second byte for ‘1’ (the data becomes ‘01’) set by the following the command:

  ```
  Tag1: db 2, 1
  ```

- While the command,

  ```
  mov ax,[Tag1]
  ```

  moves the 16-bit (2 bytes) data from Tag1 to ‘ax’ in the form 0102, as the architecture is following little endian

- The use of ‘ax’ register forced to move 2 bytes from the memory

- This issue can be resolved by replacing ‘ax’ with ‘al’ and ‘bx’ with ‘bl’, as ‘al’ and ‘bl’ refer to one byte each.
- The program is given as follows:

```
[org 0x0100]
mov al,[Tag1]
mov bl,4
add al,bl

mov ax,0x4c00   ; exit
int 0x21

Tag1: db 2, 1
```

Listing File

```
1                      [org 0x0100]
2   00000000  A0[0C00]    mov al,[Tag1]
3   00000003  B304       mov bl,4
4   00000005  00D8       add al,bl
5
6   00000007  B8004C     mov ax,0x4c00       ; exit
7   0000000A  CD21       int 0x21
8
9   0000000C  0201       Tag1: db 2, 1
```

Task 01

- Show the values (data) of ax, bx and Tag1 for all executable instructions:

```
[org 0x0100]
mov ah,[Tag1]
mov bh,4
add ah,bh

mov ax,0x4c00   ; exit
int 0x21

Tag1: db 2, 1
```

Task 02

- Show the values (data) of ax, bx and Tag1 for all executable instructions:

```
[org 0x0100]
mov al,[Tag1]
mov bh,4
add al,bh

mov ax,0x4c00   ; exit
int 0x21

Tag1: db 2, 1
```

Tip

- While programming in assembly language, people assume that the registers they are going to use are empty
- This assuption may lead to undesirable results
- To be on the safe side, either use “mov ax, 0” or “xor ax, ax” to clear the required register
- Typically “xor ax, ax” is used to clear the register

Addressing Techniques

- The address field(s) in a typical instruction format is/ are relatively small

- We would like to be able to reference a large range of locations in main memory or for some systems virtual memory

- To achieve this objective, a variety of addressing techniques can be employed

- They all involve some trade-off between address range and/or addressing flexibility on one hand and on other hand, the number of memory references in the instruction and/or the complexity of address calculation

- Most commonly used addressing techniques are:
  - **Immediate Addressing:** the instruction itself contains the data to be loaded into the destination

    ```
    mov ax, 2
    ```

  - **Direct Addressing:** the address field specifies which memory word or register contains the operand

    ```
    mov ax, [Tag1]
    ```

- **Indirect Addressing:** the address field specifies which memory word or register contains the address of an operand

  mov dx, num1

  mov ax, [dx]

- **Indexed Addressing:** the address field consists of 2 parts:

1. Index Register Contents
2. Constant

The address field is the sum of index register's contents and a constant. It specifies which memory word or register contains the operand

mov ax, [Tag1+bx]

Immediate Addressing

- Consider the following piece of code:

```
[org 0x0100]
xor ax,ax
xor bx,bx
mov ax,2  ⎫
          ⎬ Immediate Addressing
mov bx,4  ⎭
add ax,bx

mov ax,0x4c00    ; exit
int 0x21
```

Direct Addressing

- Consider the following piece of code:

```
[org 0x0100]
xor ax,ax
xor bx,bx
mov ax,[Tag1]  } Direct Addressing
mov bx,4
add ax,bx

mov ax,0x4c00      ; exit
int 0x21

Tag1: dw 2, 1
```

Indirect Addressing

- Consider the following piece of code:

```
[org 0x0100]
xor ax,ax
xor bx,bx
mov bx,Tag1  }  Indirect Addressing
mov ax,[bx]  }
mov dx,4
add ax,dx

mov ax,0x4c00     ; exit
int 0x21

Tag1: dw 2, 1
```

Indexed Addressing

- Consider the following piece of code:

```
[org 0x0100]
xor ax,ax
xor bx,bx
mov bx,0
mov ax,[Tag1+bx]    } Indexed Addressing
mov dx,4
add ax,dx

mov ax,0x4c00       ; exit
int 0x21

Tag1: dw 2, 1
```