Lecture 12
Computer Technology

First Grade

2018-2019

Assistant Prof. Dr. Emad I Abdul Kareem

College of Education

Computer Science Department

الجامعة المستنصرية| Mustansiriyah University

Lecture 12

Introduction to 8086 Programming

The 8086 microprocessor is one of the family of 8086, 80286, 80386, 80486, Pentium, PentiumI, II, III …. Also referred to as the X86 family. Learning any imperative programming language involves mastering a number of common concepts :

Variables: declaration/definition

Assignment: assigning values to variables

Input/Output: Displaying messages

Displaying variable values Control flow: if-then

Loops Subprograms: Definition and Usage

Programming in assembly language involves mastering the same concepts and a few other issues .

12.1 **Variables**

For the moment we will skip details of variable declaration and simply use the 8086 registers as the variables in our programs. Registers have predefined names and do not need to be declared.

The 8086 has 14 registers. Each of these is a 16-bit register. Initially, we will use four of them – the so called the general purpose registers :

ax, bx, cx, dx

These four 16-bit registers can also be treated as eight 8-bit registers:

ah, al, bh, bl, ch, cl, dh, dl

12.2 Assignment

In Java, assignment takes the form :

x = 42 ; y = 24; z = x + y;

In assembly language we carry out the same operation but we use an instruction to denote the assignment operator ("=" in Java). The above assignments would be carried out in 8086 assembly langauge as follows

mov x, 42 mov y, 24 add z, x add z, y

The mov instruction carries out assignment. It which allows us place a number in a register or in a memory location (a variable) i.e. it assigns a value to a register or variable .

Example: Store the ASCII code for the letter A in register bx .

mov bx, 'A '

The mov instruction also allows you to copy the contents of one register into another register .

Example : mov bx, 2 mov cx, bx

The first instruction loads the value 2 into bx where it is stored as a binary number. [a number such as 2 is called an integer constant]. The Mov instruction takes two operands, representing the destination where data is to be placed and the source of that data . The General Form of Mov Instruction is:

mov destination, source

Where destination must be either a register or memory location and source may be a constant, another register or a memory location .

Note: The comma is essential. It is used to separate the two operands. A missing comma is a common syntax error Comments. Anything that follows semi-colon (;) is ignored by the assembler. It is called a comment. Comments are used to make your programs readable. You use them to explain what you are doing in English .

12.3 More 8086 Instructions

add, inc, dec and sub instructions

The 8086 provides a variety of arithmetic instructions. For the moment, we only consider a few of them. To carry out arithmetic such as addition or subtraction, you use the appropriate instruction . In assembly language you can only carry out a single arithmetic operation at a time. This means that if you wish to evaluate an expression such as :

$$z = x + y + w - v$$

You will have to use 3 assembly language instructions – one for each arithmetic operation. These instructions combine assignment with the arithmetic operation .

Example

```
mov  ax, 5 ; load 5 into ax

add  ax, 3 ;  add 3 to the contents of ax,      ;  ax now contains 8

inc  ax ;     add 1 to ax     ; ax now contains 9

dec  ax ;     subtract 1 from ax     ; ax now contains 8

sub  ax, 6 ;  subtract 4 from ax     ; ax now contains 2
```

The ***add*** instruction adds the source operand to the destination operand, leaving the result in the destination operand . The destination operand is always the first operand in 8086 assembly language.

The ***inc*** instruction takes one operand and adds 1 to it. It is provided because of the frequency of adding 1 to an operand in programming .

The ***dec*** instruction like ***inc*** takes one operand and subtracts 1 from it. This is also a frequent operation in programming .

The ***sub*** instruction subtracts the source operand from the destination operand leaving the result in the destination operand.

12.4 Implementing a loop: The jmp instruction

```
Label_X:  add ax, 2
          add bx, 3
          jmp Label_X
```

The ***jmp*** instruction causes the program to start executing from the position in the program indicated by the label ***Label_X***. This is an example of an endless loop . We could implement a while loop using a conditional jump instruction such as ***JL*** which means jump-if-less-than. It is used in combination with a comparison instruction – ***cmp*** .

```
          mov ax, 0
Label_X:  add ax, 2
          add bx, 3
          cmp ax, 10
          jl Label_X
```

The above loop continues while the value of ***ax*** is less than 10. The ***cmp*** instruction compares ***ax*** to 0 and records the result. The ***jl*** instruction uses this result to determine whether to jump to the point indicated by ***Label_X*** .

12.3 Input / Output

Each microprocessor provides instructions for I/O with the devices that are attached to it, e.g. the keyboard and screen .The 8086 provides the instructions in for input and out for output. These instructions are quite complicated to use, so we usually use the operating system to do I/O for us instead .In assembly language we must have a mechanism to call the operating system to carry out I/O . In addition we must be able to tell the operating system what kind of I/O operation we wish to carry out, e.g. to read a character from the keyboard, to display a character or string on the screen or to do disk I/O . In 8086 assembly language, we do not call operating system subprograms by name, instead, we use a software interrupt mechanism .An interrupt signals the processor to suspend its current activity (i.e. running your program) and to pass control to an interrupt service program (i.e. part of the operating system).

A software interrupt is one generated by a program (as opposed to one generated by hardware) . The 8086 int instruction generates a software interrupt .It uses a single operand which is a number indicating which MS-DOS subprogram is to be invoked . For I/O and some other operations, the number used is 21h .Thus, the instruction int 21h transfers control to the operating system, to a subprogram that handles I/O operations. This subprogram handles a variety of I/O operations by calling appropriate subprograms .

This means that you must also specify which I/O operation (e.g. read a character, display a character) you wish to carry out. This is done by placing a specific number in a register . The ***ah*** register is used to pass this information . For example, the subprogram to display a character is subprogram number 2h. This number must be stored in the ***ah*** register. We are now in a position to describe character output .

When the I/O operation is finished, the interrupt service program terminates and our program will be resumed at the instruction following int.

12.3.1 Character Output

The task here is to display a single character on the screen. There are three elements involved in carrying out this operation using the ***int*** instruction :

1. We specify the character to be displayed. This is done by storing the character's **ASCII** code in a specific 8086 register. In this case we use the ***dl*** register, i.e. we use dl to pass a parameter to the output subprogram .

2. We specify which of **MS-DOS's I/O** subprograms we wish to use. The subprogram to display a character is subprogram number **2h**. This number is stored in the ***ah*** register .

3. We request **MS-DOS** to carry out the I/O operation using the ***int*** instruction. This means that we interrupt our program and transfer control to the **MS-DOS** subprogram that we have specified using the ***ah*** register .

Example 1: Write a code fragment to display the character 'a' on the screen :

```
mov  dl, 'a' ;  dl = 'a'

mov  ah, 2h ; character output subprogram

int  21h  ; call ms-dos output character
```

As you can see, this simple task is quite complicated in assembly language.

12.3.1 Character Input

The task here is to read a single character from the keyboard. There are also three elements involved in performing character input:

1. As for character output, we specify which of **MS-DOS's I/O** subprograms we wish to use, i.e. the character input from the keyboard subprogram. This is **MS-DOS** subprogram number 1h. This number must be stored in the ***ah*** register.

2. We call **MS-DOS** to carry out the I/O operation using the ***int*** instruction as for character output.

3. The **MS-DOS** subprogram uses the al register to store the character it reads from the keyboard.

Example 2: Write a code fragment to read a character from the keyboard:

```
mov ah, 1h ; keyboard input subprogram
int 21h ; character input ;
                    character is stored in al
```

The following example combines the two previous ones, by reading a character from the keyboard and displaying it.

Example 3: Reading and displaying a character:

```
mov ah, 1h ; keyboard input subprogram
int 21h ;    read character into al
mov dl, al ; copy character to dl
mov ah, 2h ; character output subprogram
int 21h ;    display character in dl
```