

Chapter 4: Assembly Language Programming

- The basic element of an assembly program is the statement.
 - Two types of statements:
 - **Instructions:** executable statements that actually do something.
 - **Directive:** provide information to assist the assembler in producing executable code. For example, create storage for a variable and initialize it.
-

Assembly Programming (cont'd)

- Assembly language instructions equate one-to-one to machine-level instructions, except they use a mnemonic to assist the memory.
 - Program control: Control the flow of the program and what instructions to execute next (jump, goto).
 - Data transfer: Transfer data to a register or to main memory.
 - Arithmetic: add, subtract, multiply, divide, etc.
-

Assembly Programming (cont'd)

- Logical: > < = etc.
 - Input-output: read, print etc.
-

Statements

- A statement is composed of a name, a mnemonic, operands and an optional comment. Their general format is as follows:
[name] [mnemonic [operand(s)]] [;comments]
-

Names, labels

- **Name:** Identifies a label for a statement/instruction or for a variable or constant.
 - Can contain one or more of several characters
 - Only first 31 characters are recognized
 - Case insensitive.
 - First character may not be a digit.
-

Names, labels

- The period “.” may only be used as the first character.
- Cannot use reserved names.
- Can be used to name variables. Such when put in front of a memory allocation directive. Can also be used to define a constant. For example:

```
count1 db 50 ; a variable (memory allocation  
directive db)
```

```
count2 equ 100 ; a constant
```

Names, labels

- Can be used as labels for statements to act as place markers to indicate where to jump to. Can identify a blank line. For example:

```
label1: mov    ax, 10
        mov    bx, 0
        .
        .
        .
label2:
        jmp    label1    ; jump to label1
```

Mnemonics

- **Mnemonic:** identifies an instruction or directive. These were described above.
 - The mnemonics are standard keywords of the assembly language for a particular processor.
 - You will become familiar with them as time goes on this semester.
-

Operands

- **Operands** are various pieces of information that tells the CPU what to take the action on. Operands may be a register, a variable, a memory location or an immediate value.

10 xx immediate value

count xx variable

AX xx register

[0200] xx memory location

- **Comments:** Any text can be written after the statement as long as it is preceded by the “;”.
-

Elements of Assembly Language for the 8086 Processor

- **Assembler Character Set** These are used to form the names, mnemonics, operands, variables, constants, numbers etc. which are legal in 8086 assembly.
 - **Constant:** A value that is either known or calculated at assembly time. May be a number or a string of characters. Cannot be changed at run time.
-

Elements of Assembly Language...

(cont.)

- **Variable:** A storage location that is referenced by name. A directive must be executed identifying the variable name with the location in memory.
 - **Integers:** Numeric digits with no decimal point, followed by the radix mentioned before (e.g., d, h, o, or b). Can be signed or unsigned.
-

Elements of Assembly Language...

(cont.)

- **Real numbers:** floating point number made up of digits, a decimal point, an optional exponent, and an optional leading sign.
(+ or -) digits.digits [exponential (+ or -) digits]
-

Elements of Assembly Language...

(cont.)

- **Characters and strings:**

- A character is one byte long.
 - Can be mapped into the binary code equivalent through the ASCII table, and vice-versa.
 - May be enclosed within single or double quotation marks.
 - Length of string determined by number of characters in string, each of which is 1 byte. For example:
-

Elements of Assembly Language...

(cont.)

“a” – 1 byte long

‘b’ – 1 byte long

“**stack overflow**” – 14 bytes long

‘**abc#?%%A**’ – 8 bytes long

Example of Simple Assembly Program

- The following simple program will be demonstrated and explained:

```
mov    ax,5h      ; move 5h into the AX register
add    ax,10h      ; add 10h to the AX register
add    ax,20h      ; add 20h to the AX register
mov    sum,ax      ; store value of AX in variable
; ending the program
mov ax, 4C00H
int 21h
```

Example (cont.)

- The result is that at the end of the program, the variable `sum`, which exists somewhere in memory (declaration not shown), now accepts the value accumulated in `AX`, namely, `35h`.
 - Explain program.
-

More Complex Example (cont.)

```
1:title Hello World Program (hello.asm)
2:
3:; this program displays "Hello, World"
4:
5:dosseg
6:.model small
7:.stack 100h
8:
9:.data
10:  hello_message db 'Hello, World!',0dh,0ah,'$'
11:
```

More Complex Example (cont.)

```
12: .code
13:  main  proc
14:         mov ax,@data
15:         mov ds,ax
16:
17:         mov ah,9
18:         mov dx,offset hello_message
19:         int 21h
20:
21:         mov ax,4C00h
22:         int 21h
23:  main      endp
24:  end      main
```

More Complex Example (cont.)

The program is explained as follows:

- **Line 1:** Contains the title directive. All characters located after the title directive are considered as comments, even though the ; symbol is not used.
 - **Line 3:** Comment line with the ; symbol
 - **Line 5:** The dosseg directive specifies a standard segment order for the code, data and stack segments. The code segment is where the program instructions are stored. The data segment is where the data (variables) are stored. The stack segment is where the stack is maintained.
-

More Complex Example (cont.)

- **Line 6:** The `.model` directive indicates the memory architecture to be used. In this case, it uses the Microsoft small memory architecture. It indicates this by the word `small` after `.model`.
 - **Line 7:** This directive sets aside `100h` of memory for the stack. This is equivalent to 256 bytes of memory ($16^2 = 256$).
 - **Line 9:** The `.data` directive marks the beginning of the data segment, where the variables are defined and memory allocated to them.
-

More Complex Example (cont.)

- **Line 10:** The `db` directive stands for “define byte”, which tells the assembler to allocate a sequence of memory bytes to the data that follow. `0dh` is a carriage return and `0ah` is the linefeed symbol. The `$` is the required terminator character. The number of memory bytes is determined by the data themselves. `Hello_message` is the name of the variable to be stored in memory, and the `db` allocates memory to it in the size defined by the data following it.
 - **Line 12:** The directive `.code` is the indication of the beginning of the code segment. The next few lines represent instructions.
-

More Complex Example (cont.)

- **Line 13:** The proc directive is used to declare the main procedure called “main”. Any name could have been used, but this is in keeping with the C/C++ programming requirement that the main procedure be called the main function. The first executable instruction following this directive is called *the program entry point* - the point at which the program begins to execute.
-

More Complex Example (cont.)

- **Line 14:** The instruction `mov` is used to copy the contents of one address into the other one. The first operand is called the destination address, while the second one is called the source address. In this particular use, we tell the assembler to copy the address of the data segment (`@data`) into the `AX` register.
 - **Line 15:** Copies the content of `AX` into `DS`, which is a register used to put the data segment, the default base location for variables.
-

More Complex Example (cont.)

- **Line 17:** This instruction places the value 9 in the AH register. Remember that from the page.com program, this is the register used to store the name of the DOS subroutine to be called with the int 21 instruction.
 - **Line 18:** This instruction places the address of the string to be identified in the DX register. Remember that this is the offset, where the default base segment is already identified in the DS register as the base segment for the data segment. Since the address of the variable hello_message begins at the beginning of the data segment, identified by DS, we only need to supply the offset for the variable.
-

More Complex Example (cont.)

- **Line 19:** The instruction `int 21`, as we saw before, takes the name of the function from the `DX` register, which in this case is 9. DOS function 9, incidentally, sends the contents of `DX` register to the VRT output device. The `DX` register contains the address of the string to be sent.
 - **Line 21 and 22:** These instructions represent the equivalent of an end or stop statement. This is different from that done for `page.com` because this will be an executable program (`.exe`), rather than a `.com` program. More on this later.
 - **Line 23:** Indicates the end of the main procedure.
-

More Complex Example (cont.)

- **Line 24:** The END directive – the last line to be assembled. The main next to it indicates the program entry point.
-

More Complex Example (cont.)

- The program may seem overly complicated for such a simple program.
 - But remember that assembly language corresponds one-to-one with machine language instructions.
 - Note that it takes only 562 bytes of memory when assembled and compiled.
-

More Complex Example (cont.)

- The same program written in a high level language will require several more machine level instructions to carry out the same thing.
 - Written in Turbo C++, the executable program would take 8772 bytes of memory to store.
-