Chapter 3

8086 Microprocessor

Assembly Language Programming

Bharati Ingale (bharati_ingale@yahoo.com)

8086 - Assembler Directives

Assembly language consists of 2 type of statements:

i. **Executable Statements**

ii. **Assembler Directives**

8086 - Assembler Directives

i. Executable Statements

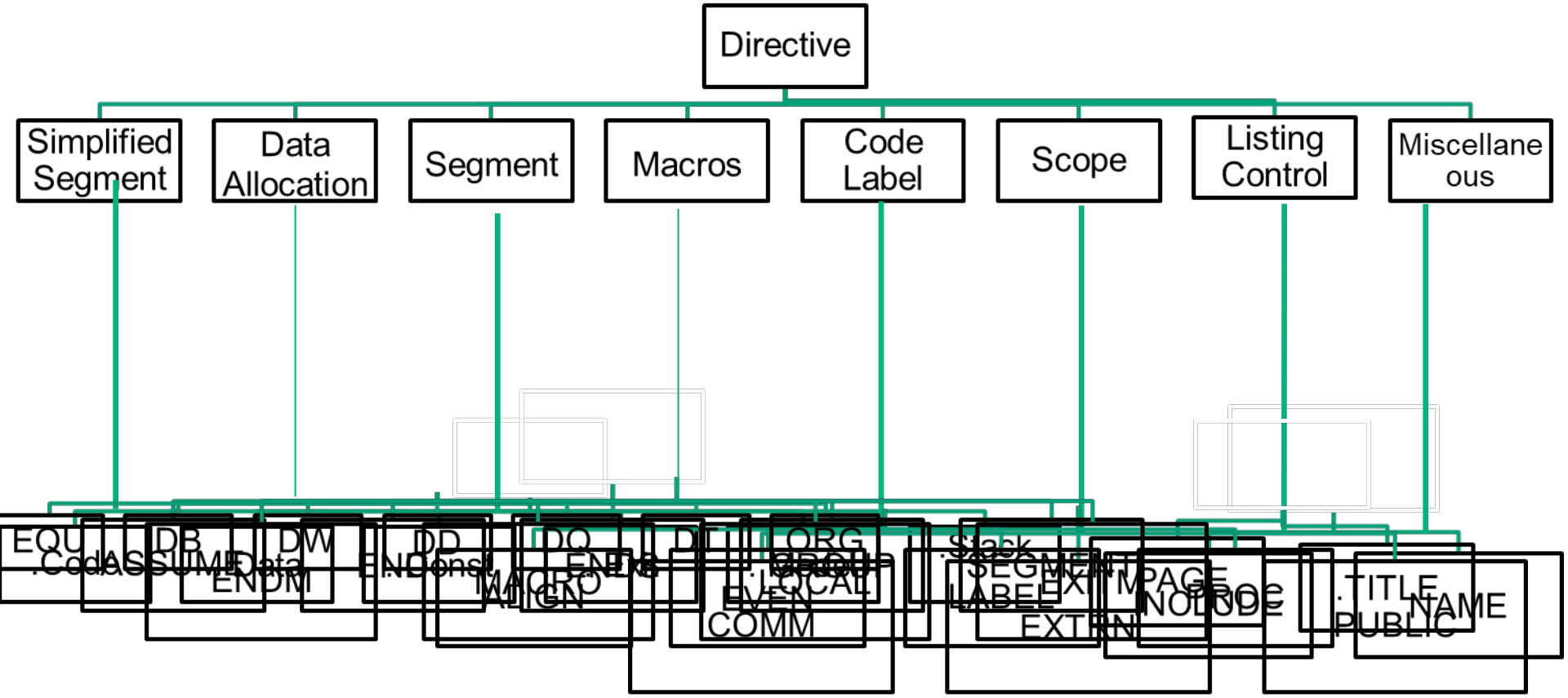
- These are statements to be executed by the processor.
- It consist of the entire instruction set of 8086
- Instructions that are translated into machine code by assembler

8086 - Assembler Directives

ii. Assembler Directives

- The statements that direct the assembler to do some special task
- They are effective only during the assembly of program.
- They do not generate code or No M/C language code is produced for these statements.
- Their main task is to inform the assembler about the start/endof a segment, procedure or program, to reserve appropriate space for data storage etc.

8086


Directive
Simplified Segment
Data Allocation
Segment
Macros
Code Label
Scope
Listing Control
Miscellaneous
EQU
DB
DW
DD
DQ
DT
ORG
.Code
ENDM
ENDS
MACRO
ALIGN
EVEN
LOCAL
COMM
.Stack
SEGMENT
LABEL
EXTRN
PAGE
NOLIST
.TITLE
NAME
PUBLIC

Model Definition

MODEL directive –selects the size of the memory model

MODEL MEDIUM

- Data must fit into 64KB
- Code can exceed 64KB

- **MODEL COMPACT**
  - Data can exceed 64KB
  - Code cannot exceed 64KB
- **MODEL LARGE**
  - Data can exceed 64KB (but no single set of data should exceed 64KB)
  - Code can exceed 64KB

Model Definition

MODEL directive –selects the size of the memory model

- **MODEL HUGE**
  - Data can exceed 64KB (data items i.e. arrays can exceed 64KB)
  - Code can exceed 64KB
- **MODEL TINY**
  - Data must fit into 64KB
  - Code must fit into 64KB
  - Used with COM files

Segments

Segment definition:

The 80x86 CPU has four segment registers: CS, DS, SS, ES

Segments of a program:

.STACK ; marks the beginning of the stack segment

E*xample:* .STACK 64; reserves 64B of memory for the stack

.DATA ; marks the beginning of the data segment

Example: .DATA1 DB 52H

;DB directive allocates memory in byte-size chunks

Segments

.CODE ; marks the beginning of the code segment

- starts with PROC (procedures) directive
- the PROC directive may have the option FAR or NEAR
- ends by ENDP directives

PAGE and TITLE directives

PAGE [lines],[columns]

- To tell the printer how the list should be printed
- Default mode is 66 lines per page with 80 characters per line
- The range for number of lines is 10 to 255 and for columns is 60 to 132

TITLE

- Print the title of the program
- The text after the TITLE pseudo-instruction cannot be more than 60 ASCII characters

Bharati Ingale (bharati_ingale@yahoo.com)

8086 - Assembler Directives

1	.CODE	Indicates the beginning of the Code seg .CODE [NAME]
2	.DATA	Indicates the beginning of the Data seg
3	.MODEL	Used for selecting a standard memory model: small, medium, compact, large, huge .MODEL [memory model]
4	.STACK	Used for defining the stack .STACK [size]
5	EQU	Used to give name to some value or symbol in the program

8086 - Assembler Directives

6	DB	Defines a byte type variable: SUM DB 11
7	DW	Defines a word type variable (2 byte)
8	DD	Defines a double word type variable (4 byte)
9	DQ	Defines a quad word type variable (8 byte)
10	DT	Defines a 10 bytes to a variable (10 byte)

8086 - Assembler Directives

11	ORG	Allows us to set the location counter to any desired value at any point in the program
12	DUP	Copies the contents of the bracket followed by this keyword into the memory location specified before it LIST DB 10 DUP(0): stores LIST as a series of 10 bytes initialized to 0

8086 - Assembler Directives

13	ASSUME	Used for telling the assembler the name of the logical segment which should be used ASSUME CS: Code, DS: Data, SS: Stack
14	END	Placed at the end of a source. Acts as a last statement. Terminates the entire program
15	SEGMENT	Used to indicate the start of a logical segment
16	ENDS	Indicates the end of a segment

8086 - Assembler Directives

17	PROC	Used to indicate the start of a procedure
18	ENDP	Indicates the end of a procedure
19	LABEL	Assigns name to the current value of the location counter

Assembly Language Programming

Bharati Ingale (bharati_ingale@yahoo.com)

Assembly Programming

- Assembly Language instruction consist of four fields

[label:] mnemonic [operands] [;comment]

- Labels
  - See rules
- mnemonic, operands
  - MOV AX, 6764
- comment
  - ; this is a sample program

Bharati Ingale (bharati_ingale@yahoo.com)

Assemble, Link, and Run Program

STEP	INPUT	PROGRAM	OUTPUT
1. Edit the program	keyboard	editor	myfile.asm
2. Assemble the program	myfile.asm	MASM or TASM	myfile.obj myfile.lst myfile.crf
3. Link the program	myfile.obj	LINK or TLINK	myfile.exe myfile.map

Bharati Ingale (bharati_ingale@yahoo.com)

Assemble, Link, Run Files

.asm – source file

.obj – machine language file

.lst – list file

- it lists all the Opcodes, Offset addresses, and errors that MASM detected

.crf – cross-reference file

- an alphabetical list of all symbols and labels used in the program as well as the program line numbers in which they are referenced

.map – map file

- to see the location and number of bytes used when there are many segments for code or data

8086

Data_Here	SEGMENT	Data_Description
	LIST DB 10 DUP(0)	Stores LIST as a seriesof 10 Bytes initialized to zero
	--------	
Data_Here	ENDS	
Code_Here	SEGMENT	Body of theprogram
	ASSUME CS: Code_Here, DS: Data_Here	Makes Code_Here as code segment & Data_Here as data segment
	----	

Code_Here	ENDS	
	END	

8086 - DOS CALLS

Function no

DOS INTERRUPT

mov ah, **Function No**

int 21h

8086 - DOS CALLS

1	mov ah,01h int 21h	Input a character from the keyboard. Takes the user input character from the user and returns the ascii value of character in AL register
2	mov ah,02h int 21h	To display a character on the screen, DL should contain the offset address of the output screen
3	mov dx, offset msg mov ah,09h int 21h	To display a string on the screen, it displays the string whose offset address is in DX

4	mov ah,4ch int 21h	Terminate the program

ADD Program

```
data segment
    num1  db 23h
    num2  db 12h
    msg1 db 0dh,0ah,"the result of the addition is $"
data ends

code segment
assume cs: code, ds: data
start:  mov ax,data
         mov ds,ax

        mov dx, offset msg1;
        mov ah,09h;
        int 21h; to display a string on the     screen, it displays the string whose offset address is
in DX



    mov bl,num1
    mov cl,num2
    add bl,cl
    mov cl,bl ; result in cl
```

ADD Program

```
data segment
    num1  db 23h
    num2  db 12h
    msg1 db 0dh,0ah,"the result of the addition is $"
data ends

code segment
assume cs: code, ds: data
start:  mov ax,data
        mov ds,ax

     mov dx, offset msg1;
     mov ah,09h;
     int 21h; to display a string on the screen, it displays the string whose offset address is in DX

    mov bl,num1
    add bl,num2
    mov cl,bl ; result in cl

    mov bl,cl ; to display Higher    Byte i.e 3
     and bl,0F0h
     ror bl,04h

     call convert; convert HEX into ASCI

    mov dl,bl
```

```
.model small
.data
      num1  db 23h
      num2  db 12h
.code
      mov ax,data
      mov ds,ax

      mov al, num1
      mov bl, num2
      add al,bl

      mov ah,4ch
      int 21h
end
```

```
data segment
num1  db 23h
num2  db 12h
data ends

code segment
assume cs: code, ds: data
start:     mov ax,data
           mov ds,ax

           mov al, num1
      mov bl, num2
      add al,bl

mov ah,4ch
int 21h

code ends
end start
```

```
data segment
    num1  db 23h
    num2  db 12h
    msg1 db 0dh,0ah,"the result of the addition is $"
data ends

code segment
assume cs: code, ds: data
start:   mov ax,data
     mov ds,ax

     mov dx, offset msg;
     mov ah,09h; to display a string on the screen, it displays
      int 21h;    the string whose offset address is in DX
```

```
mov bl,num1
add bl,num2

mov cl,bl

mov bl,cl
and bl,F0h
ror bl,04h

call convert ; convert decimal into ASCI

mov dl,bl
mov ah,02h; to display a character on the screen, DL should
int 21h;      contain the offset address of the out put screen
```

```
mov bl,cl
and bl,0Fh
call convert

mov dl,bl
mov ah,02h
int 21h

mov ah,4ch;    Terminate the program
int 21h
```

8086

```
convert proc
      cmp bl,0Ah
      jc l1
      add bl,37h
      jmp l2
            l1: add bl, 30h
            l2: ret
endp
code ends
end start
```

Thank You