Topic-IV

80X86 Instruction Sets and ALP

T1. Barry B Brey, The Intel Microprocessors .Pearson, Eight Ed. 2009. Chapter 4-6, 8

Feb 9th 2021

MASM Directives
Microsoft Assembler Directives

Assembler Directives

- **Directive**:- Instructions to the Assembler
- Helps the assembler to convert the ALP to machine language Program

- MASM makes use of directive to convert ALP to machine level program

Assembler Directives

- Indicate how an operand or section of a program is to be processed by the assembler.
  - some generate and store information in the memory; others do not
- The DB (define byte) directive stores bytes of data in the memory.
- The DW (define word) directive stores 1 word of data in the memory.
- The DD (define double word) directive stores double word (4 Bytes) of data in the memory.

- BYTE PTR indicates the size of the data referenced by a pointer or index register.

Storing Data in a Memory Segment

- DB (**define byte**), DW (**define word**), and DD (**define doubleword**) are most often used with MASM to define and store memory data.

- These directives label a memory location with a symbolic name and indicate its size.

⌘ Ex: STORAGE DW 100 DUP(0)

Reserve 100 words of storage in memory and give it the name STORAGE, and initialize all 100 words with 0000.

Assembler Directives

- **Data Declaration**

DB, DW, DD

- **DATA1 DB 45H, 35H, 74H**
- **DATA2 DW 2000H, 37H, 2222H**
- **DATA3 DD 234567ABH**

- Memory is reserved for use in the future by using a question mark (?) as an operand for a DB, DW, or DD directive.
  - when ? is used in place of a numeric or ASCII value, the assembler sets aside a location and does not initialize it to any specific value

  - Ex: STORAGE DW 100 DUP(?)

    Reserve 100 words of storage in memory and give it the name STORAGE, but leave the words uninitialised.

ASSUME

- ASSUME directive is used to tell the assembler the name of the logical segment it should use for a specified segment.
  - Ex: ASSUME CS:CODE tells the assembler that the instructions for a program are in a logical segment named CODE .
  - Ex: ASSUME SS: STACK_HERE

✓ i.e., ASSUME tells the assembler what names have been chosen for the code, data, extra, and stack segments.

EQU

- Equate directive (EQU) equates a numeric, ASCII, or label to another label.
  - Ex: CONTROL_WORD EQU 11001001 ; replacement

    MOV AX, CONTROL_WORD ;assignmeent

Each time the assembler finds the given name in the program, it will replace the name with the value or symbol we equated with that name.

Equates make a program clearer and simplify debugging .

EQU directive

Equate directive equates a symbolic name to a value

```
COUNT    EQU     10
CONST    EQU     20H

MOV        AH, COUNT
MOV        AL, CONST
```

ORG

- The ORG (originate) statement changes the starting offset address of the data in the data segment to a desired location .
- At times, the origin of data or the code must be assigned to an absolute offset address with the ORG statement.
- Ex: ORG 3000H

```
DATA1 DB 25
DATA2 DB 10001001b
DATA3 DB 12h

ORG 0010h
DATA4 DB ‘2591’
```

This is how data is initialized in the data segment

```
0000   19H
0001   89H
0002   12H
0010   32H, 35H, 39H, 31H
```

PROC and ENDP

- Indicate start and end of a procedure (subroutine).

- Ex: SMART_DIVIDE PROC FAR

  it identifies the start of a procedure named SMART_DIVIDE and tells the assembler that the procedure is far (in a segment with a different name from the one that contains the instruction that calls the procedure.)

Ex2: SMART_DIVIDE PROC NEAR

PROC and ENDP

- The PROC directive, which indicates the start of a procedure, must also be followed with a NEAR or FAR.

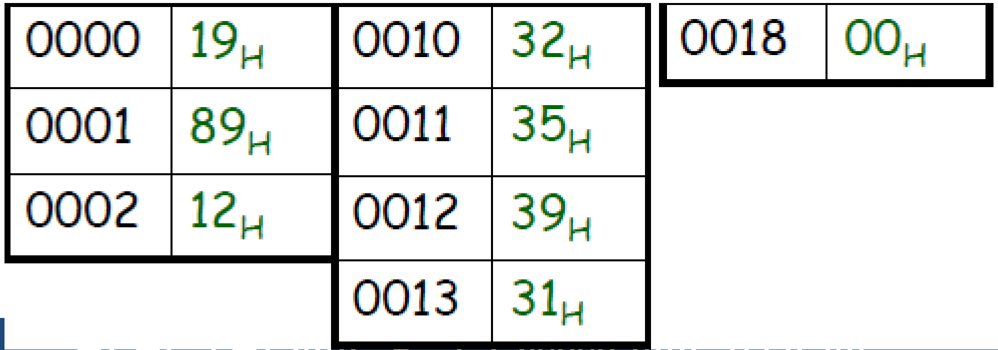
  - A NEAR procedure is one that resides in the same code segment as the program, often considered to be *local*
  - A FAR procedure may reside at any location in the memory system, considered *global*

- The term *global* denotes a procedure that can be used by any program.
- *Local* defines a procedure that is only used by the current program.

Example 1

```
ORG    0000H
DATA1          DB      25
DATA2          DB       10001001b
DATA3          DB      12H
ORG    0010H
DATA4          DB      ‘2591’
ORG    0018H
DATA5          DB      ?
```

This is how data is initialized in the data segment



0000	19_H
0001	89_H
0002	12_H

0010	32_H
0011	35_H
0012	39_H
0013	31_H

0018	00_H

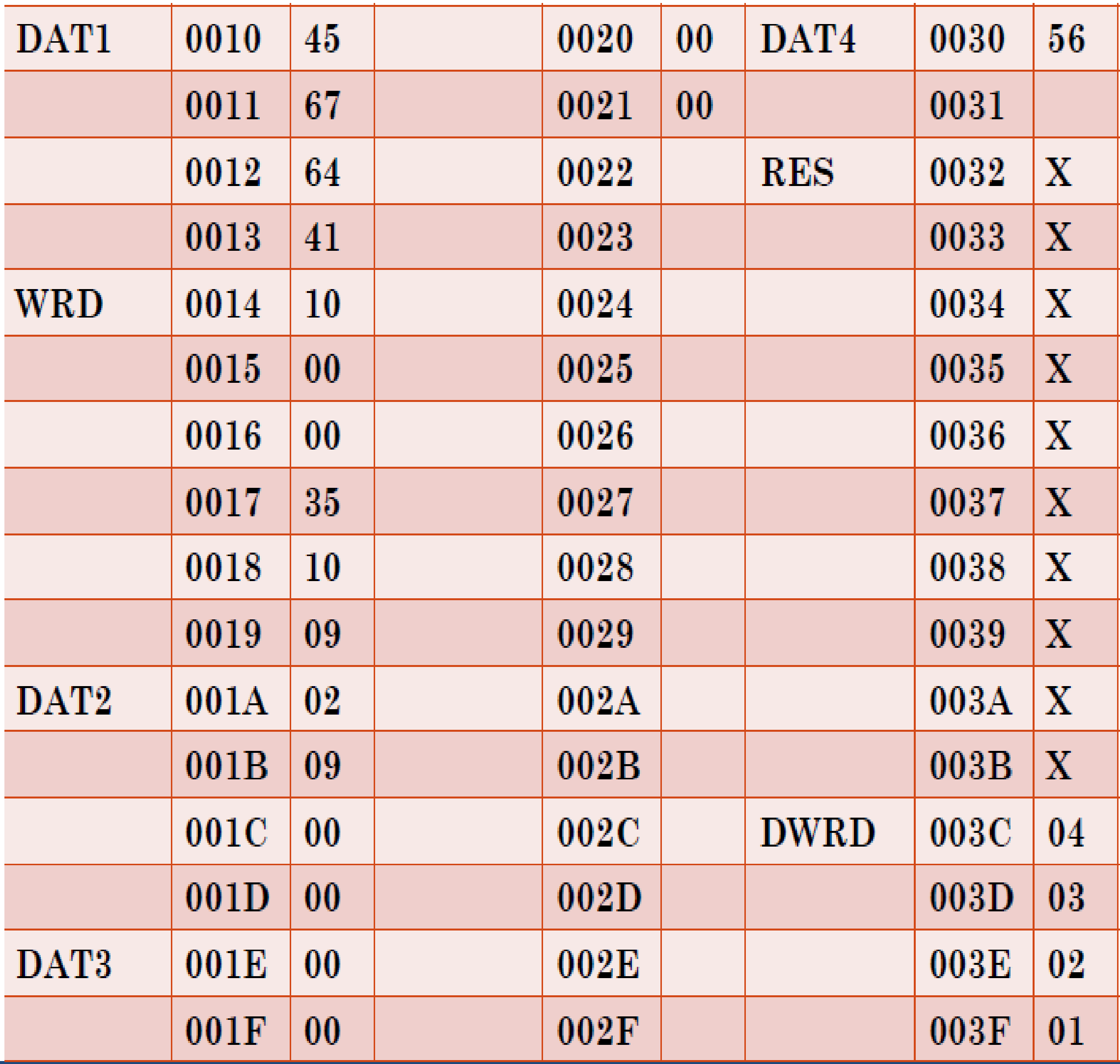
Example 2

```
ORG     0000H
MSG2    DB      '123456'
MSG3    DW      6667H
data1   DB      1,2,3
        DB      'a'
        DB      11110000b
data2   DW      12,13
        DW      2345H
        DD      300H
        DB      9         DUP(FFH)
```

0000	31	0010	00
0001	32	0011	45
0002	33	0012	23
0003	34	0013	00
0004	35	0014	03
0005	36	0015	00
0006	67	0016	00
0007	66	0017	FF
0008	01	0018	FF
0009	02	0019	FF
000A	03	001A	FF
000B	61	001B	FF
000C	F0	001C	FF
000D	0C	001D	FF
000E	00	001E	FF
000F	0D	001F	FF

Example 3

```
        ORG     0010H
COUNT   EQU  32H
VAL1    EQU  0030H
DAT1    DB      45H, 67H ,100,’A’
WRD     DW      10H,3500H,0910H
DAT2    DD      0902H
VAL2    EQU     32H
DAT3    DW      2           DUP(0)
        ORG     VAL1
DAT4    DB      56H
        ORG     VAL2
RES     DB      10          DUP(?)
DWRD    DD      01020304H
```



DAT1	0010	45		0020	00	DAT4	0030	56
	0011	67		0021	00		0031	
	0012	64		0022		RES	0032	X
	0013	41		0023			0033	X
WRD	0014	10		0024			0034	X
	0015	00		0025			0035	X
	0016	00		0026			0036	X
	0017	35		0027			0037	X
	0018	10		0028			0038	X
	0019	09		0029			0039	X
DAT2	001A	02		002A			003A	X
	001B	09		002B			003B	X
	001C	00		002C		DWRD	003C	04
	001D	00		002D			003D	03
DAT3	001E	00		002E			003E	02
	001F	00		002F			003F	01

Example 3 (b) (based on the data stored in memory)

```
MOV   SI,    DAT3
MOV   AL,    DAT1 + 1
MOV   BX,    DAT1+4
ADD   BX,    20H
MOV   AL,    [BX]
LEA   BX,    DAT4
MOV   AL,    [BX]
MOV   BX,    VAL1
MOV   AL,    [BX]
MOV   BX,    OFFSET DAT4
MOV   AL,    [BX]
MOV   AL,    DAT4
```

SI←DAT3,   SI= 0000H
AL ← DAT1 + 1=10+01=11
AL ← 67H
MOV   BX ← DAT1+4=14
BX ← 0010H
BX= BX+20H= 0010H+20H= 0030H
AL ← [BX], AL = 56H
DAT4 = 0030H
BX ← 0030H
AL ← [0030H]=56H
VAL1= 0030H
BX ←   0030H
AL ←   [0030H], AL = 56H
BX ←   0030H
AL ←   [BX]= [0030H], AL = 56H
AL ←   DAT4   AL = 56H

X86 Programming
Program Model

Model Type	Description
Tiny	All the data and code fit in one segment. Tiny programs are written in .COM which means the program must be originated at location 100H
Small	Contains two segments - One DS of 64k bytes and one CS of 64k bytes

Model Type	Description
Medium	Contains one DS of 64kbyte and any number of CS for large programs
Compact	One CS contains the program and any number of DS contains the data
Large	allows any number of CS & DS
Huge	Same as large - but the DSs may contain more than 64k bytes each

```
.Model Tiny

.data
dat1    db      'a'
        align     2
dat2    db      'b'
.code
.startup
      mov     al,dat1
      add     dat2,al
.exit
end
```

```
.Model Small
.stack
.data
dat1    db      'a'
        align     2
dat2    db      'b'
.code
.startup
      mov     al,dat1
      add     dat2,al
.exit
end
```

```
; This is the structure of a main module
; using simplified segment directives

.MODEL SMALL          ; This statement is reqd before
                      ;  you can use other simplified
                      ;  segment directives

.STACK                ; Use default 1-kilobyte stack
.DATA                 ; Begin data segment
                      ; Place data declarations here

.CODE                 ; Begin code segment
.STARTUP              ; Generate start-up code
...........           ; Place instructions here
.EXIT                 ; Generate exit code
 END
```