8086-80486 – Inst Set & ALP

MOV Revisted

EX 1. Swap the word at memory location 24000_H with 25000_H

24000_H	
24001_H	
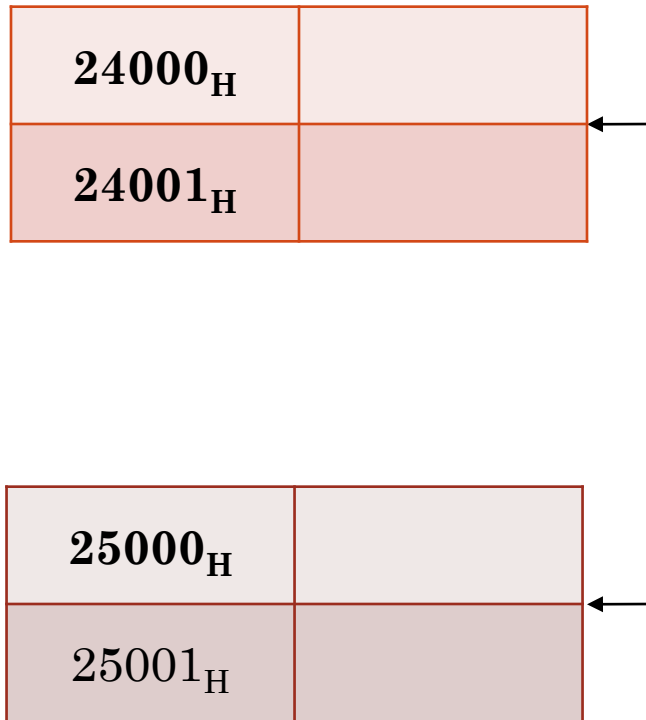
25000_H	
25001_H	

MOV AX, 2000$_H$

MOV DS, AX

MOV SI, 4000$_H$

MOV DI, 5000$_H$

MOV BX, [SI]

MOV DX, [DI]

MOV [SI], DX

MOV [DI], BX

- Initialise Segment Register

- Initialise Offset Registers

- Transfer data from reg to mem temporarily

- Store back the data in mem

Ex 2

MOV BX, 2000$_H$

MOV DI, 10$_H$

MOV AL, [BX+DI]

MOV DI, 20$_H$

MOV [BX+DI], AL

mov instruction can not
transfer data from one memory location to another
directly. It needs an intermediate register

DS: 2020 ← DS: 2010

Different mov options

R ← M

M ← R

R ← R

M ← I

R ← I

MOV   DST,SRC

- Copies the contents of source to destination

- No Flags Affected   flags affected when alu is used

- Size of source and destination must be the same   source register and destination must be of same size

- Source can be register, memory, or immediate data

- Destination can be register or memory location

Types of Instruction

- Data Transfer Instructions

- Arithmetic Instructions

- Logical Instructions

- Branch and Program control Instructions

8086-80486 – Inst Set & ALP

A Simple Program

INC Destination

- Destination – Register or

  memory location (specified in 24 diff ways)

- (AF, OF, PF, SF, ZF affected, CF not affected)

- INC BL

- INC BX

- DEC Destination

Inc / Dec the contents of a Memory location

- Specify the data size in memory

- use directive Assembler directive

- BYTE PTR, WORD PTR, DWORD PTR

- INC WORD PTR [BX]

- INC BYTE PTR[BX]

- BX-1000_H DS- 2000_H

21000	00
21001	00

LSB is 21000 and
MSB is 21001

Flags

INC WORD PTR [BX]

- OF – 0

- SF – 0

- ZF - 0

- PF – 1(follows only low byte

- AF - 1

INC BYTE PTR[BX]

- OF – 0

- SF – 0

- ZF -1

- PF -1

- AF -1

Branch Instructions

Jump if equal to - JE
Jump if zero - JZ (examine zero flag)

we give the displacement in this range -128 to 127

JE/JZ   Displacement (-128 to +127)

JNE/ JNZ   Displacement (-128 to +127)

Copy a block of data from one memory area to another memory area- from 42000_H 50 data to 44000_H

MOV	AX, 4000_H	10111000	00	40
MOV	DS,AX	10001110	11011000	
MOV	SI,2000_H	10111110	00	20
MOV	DI,4000_H	10111111	00	40
MOV	CX,0032_H	10111001	32	00

X1:	MOV	BL,[SI]	10001010	00011100
	MOV	[DI],BL	10001000	00011101
	INC	SI	01000110	
	INC	DI	01000111	
	DEC	CX	01001001	
	JNZ	X1	01110101	displacement

B8 00 40

8E D8

BE 00 20

BF 00 40

B9 32 00

8A 1C

88 1D

46

47

49

75   displacement

00000	B8
00001	00
00002	40
00003	8E
00004	D8
00005	BE
00006	00
00007	20
00008	BF
00009	00
0000A	40
0000B	B9
0000C	32
0000D	00
0000E	8A
0000F	1C

00010	88
00011	1D
00012	46
00013	47
00014	49
00015	75
00016	XX
00017	

$$0000E_H$$
$$-\ 00017_H$$
$$F7_H$$

8086-80486 – Inst Set & ALP

A Simple Program

INC Destination

- Destination – Register or

  memory location (specified in 24 diff ways)

- (AF, OF, PF, SF, ZF affected, CF not affected)

- INC BL

- INC BX

- DEC Destination

Branch Instructions

JE/JZ   Displacement (-128 to +127)

JNE/ JNZ   Displacement (-128 to +127)

Copy a block of data from one memory area to another memory area- from 42000_H 50 data to 44000_H

Data Transfer of 32 bytes

This is in hexadecima so, 42000H + 50Decimal = 42031H

42000_H	21
42001_H	22
42003_H	32
42004_H	34
42030_H	45
42031_H	67

Typo should be 42002

44000_H	
44001_H	
44003_H	
44030_H	
44031_H	

Src Addr 42000_H
Dst Addr 44000_H

42000 = 4000:2000
44000 = 4000:4000

MOV	AX, 4000$_H$
MOV	DS,AX
MOV	SI,2000$_H$
MOV	DI,4000$_H$
MOV	CX,0032$_H$

10111000	00	40
10001110	11011000	
10111110	00	20
10111111	00	40
10111001	32	00

X1:	MOV	BL,[SI]
	MOV	[DI],BL
	INC	SI
	INC	DI
	DEC	CX
	JNZ	X1
NXT:		

10001010   00011100

10001000   00011101

01000110

01000111

01001001

01110101   displacement

B8	00	40
8E	D8	
BE	00	20
BF	00	40
B9	32	00
8A	1C	
88	1D	
46		
47		
49		
75	displacement	

00000	B8
00001	00
00002	40
00003	8E
00004	D8
00005	BE
00006	00
00007	20
00008	BF
00009	00
0000A	40
0000B	B9
0000C	32
0000D	00
0000E	8A
0000F	1C

00010	88
00011	1D
00012	46
00013	47
00014	49
00015	75
00016	XX
00017	

$0000E_H$

$- \underline{00017_H}$

$\underline{F7_H}$

8086-80486 – Inst Set & ALP

Arithmetic Instructions – ADD& SUB

ADD Destination, Source

(Source) + (Destination) $\longrightarrow$ (Destination)

- Source may be an immediate no. /register/ a memory location specified by any one of the 24 addressing modes
- Destination may be a register / memory location specified by any one of the 24 addressing methods
- Both source and destination in an instruction cannot be memory locations
- Source and Destination must be of same size
- **All Flags Affected**

Example

MOV CL, 73_H 0111 0011

MOV BL, $4F_H$ 0100 1111

ADD CL, BL

Result in CL = $C2_H$ 11000010_b

(194_d if unsigned -62_d if signed)

CF = 0, PF = 0, AF = 1, ZF = 0, SF = 1, OF = 1

ADC   DESTINATION, SOURCE

(Source) + (Destination) + (CF) $\longrightarrow$ (Destination)
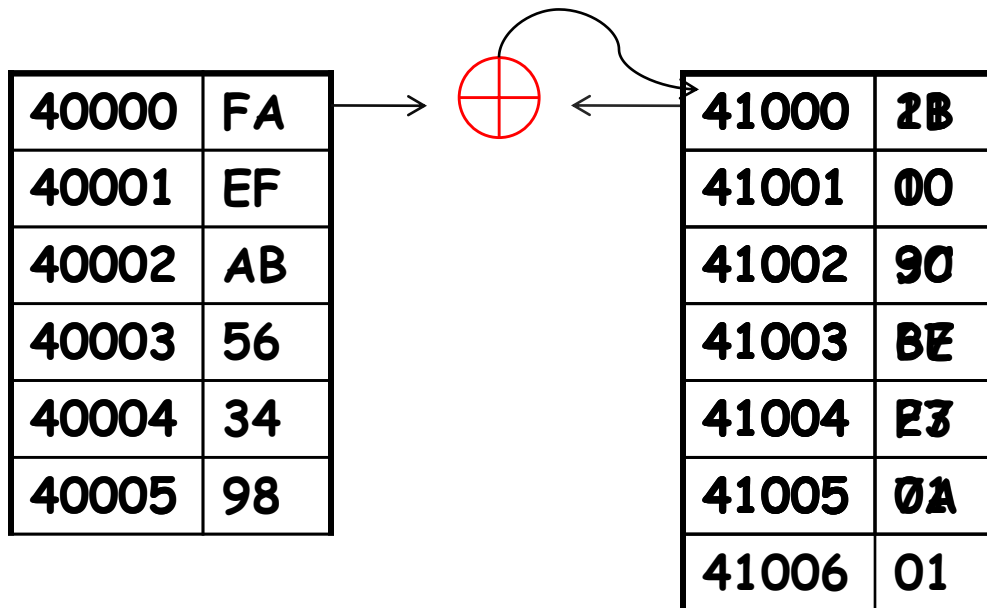
Useful for muti-byte addition of data

ADD two data of size each 6 bytes

ADC – How it works ???

1	0	1	1	1	
98	34	56	AB	EF	FA
71	F3	67	90	10	21
0A	27	BE	3C	00	1B

1

40000	FA
40001	EF
40002	AB
40003	56
40004	34
40005	98

⊕

41000	1B
41001	00
41002	3C
41003	BE
41004	27
41005	0A
41006	01

CLC clears the carry flag
(to begin with 0 carry)

CLC

Add 2 – 6 byte nos stored in location 20000_H and 21000_H store the result starting from location 21000_H
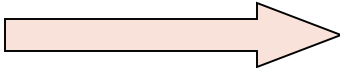
```
         MOV        AX, 2000$_H$
         MOV        DS,AX
         MOV        SI,0000$_H$
         MOV        DI,1000$_H$
         MOV        CL,06$_H$
         MOV        BL,00
         CLC
X1:      MOV        AL,[SI]
         ADC        [DI],AL
         INC        SI
         INC        DI
         DEC        CL
         JNZ        X1
         JNC        X2
         INC        BL
X2:      MOV        [DI],BL
```

Subtract

SUB DESINATION, SOURCE

(Destination) – (Source) $\longrightarrow$ (Destination)

SBB DESTINATION, SOURCE

(Destination) - (Source) – (CF) $\longrightarrow$ (Destination)

Example

MOV BX,8021$_H$

SUB BX,8190$_H$

8	0	2	1
8	1	9	0
F	E	9	1

8	0	2	1
7	E	7	0
F	E	9	1

Two's compliment addition

Z -0 C -1 A-0 P-X S-1 O-0

Auxilary flag - doesn't matter since 16 bit number
parity flag doesn't matter since it represent
last 8 bits

Write a Program to add an array of bytes stored in Memory starting at **40000$_H$** and store the result in **41000$_H$**

Number of elements in the array **50$_d$**

8086-80486 – Inst Set & ALP

Arithmetic Instructions – ADD& SUB

Write a Program to add an array of bytes stored in Memory starting at **40000$_H$** and store the result in **41000$_H$**

Number of elements in the array **50$_d$**

Example – Add 5 Numbers

40000	FF
40001	FF
40002	FF
40003	FF
40004	01

3   FD

MOV	AX, 4000_H
MOV	DS,AX
MOV	SI,0000_H
MOV	DI,1000_H
MOV	CX,0032_H
MOV	BL,00
MOV	BH,BL

B8	00	40
8E	D8	
BE	00	10
BF	00	10
B9	32	00
B3	00	
8A	FB	

X1:	ADD	BL,[SI]
	JNC	X2
	INC	BH
X2:	INC	SI
	DEC	CX
	JNZ	X1
	MOV	[DI],BX

02	1C
73	X2
FE	C7
46	
49	
75	X1
89	1D

B8   00   40
8E   D8
BE   00   10
BF   00   10
B9   32   00
B3   00
8A   FB
02   1C
73   YY
FE   C7
46
49
75   XX
89   1D

00018_H
$- \underline{00016_H}$
02_H

00000	B8	00010	8A
00001	00	00011	FB
00002	40	00012	02
00003	8E	00013	1C
00004	D8	00014	73
00005	BE	00015	YY
00006	00	0016	FE
00007	00	0017	C7
00008	BF	0018	46
00009	00	0019	49
0000A	10	001A	75
0000B	B9	001B	XX
0000C	32	001C	89
0000D	00	001D	1D
0000E	B3		
0000F	00		

00012_H
$- \underline{0001C_H}$
$F6_H$

YY – 02
XX – F6

8086-80486 – Inst Set & ALP

MASM Assembler Directives

Data Declaration

DB,   DW,   DD

DB - define byte
DW - define word
DD - define double word

DATA1   DB $45_H, 35_H, 74_H$

DATA2   DW $2000_H, 37_H, 2222_H$

DATA3   DD $234567AB_H$

ORG 0000_H

	ORG	0000_H
DATA1	DB	25
DATA2	DB	10001001_b
DATA3	DB	12_H
	ORG	0010_H
DATA4	DB	'2591'
ORG	0018_H	
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

Addr	Val	Addr	Val
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

	ORG	0000_H
MSG2	DB	'123456'
MSG3	DW	6667_H
data1	db	1,2,3
	db	'a'
	db	11110000_b
data2	dw	12,13
	dw	2345_H
	dd	300_H
	DB	9 DUP(FF_H)

EQU Directive

Equate directive equates a symbolic name to a value

COUNT   EQU 10

CONST   EQU 20_H

define stores value in memory
but EQU doesn't store any thing in memory
whenever it sees count, it replaces it with value

MOV AH, COUNT

MOV AL, CONST

ADD AH,AL

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

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

```
MOV        SI,DAT3
MOV        AL, DAT1 + 1
MOV        BX, DAT1+4
ADD        BX,20$_H$
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
```

8086-80486 – Inst Set & ALP

MASM Program Models

Physical Vs. Logical Segmentation

- A block of memory of discrete size-called a "physical segment"

- The number of bytes in a physical memory segment is 64K for 16-bit processors or 4 gigabytes for 32-bit processors

- A variable-sized block of memory- called a "logical segment" occupied by a program's code or data

Logical Segments

- Logical segments contain the 3 components of a program:
  - code
  - data
  - stack

- Mapping of Logical segments to actual physical segments in memory

- Load Segment Registers with actual physical addresses

- MASM has assembler directives to do the same

Logical Segments

- Segments defined in two ways
  - Simplified segment directives
  - Full segment definitions

Models

- There are many models available to MASM Assembler ranging from Tiny to Huge

- To designate a model use the .MODEL statement followed by the size of the memory system

- Ex: .MODEL TINY

- TINY Model requires that all program and data fit into one 64K segment

. CODE defines code segment
. DATA   defines data segment
. STARTUP
. EXIT
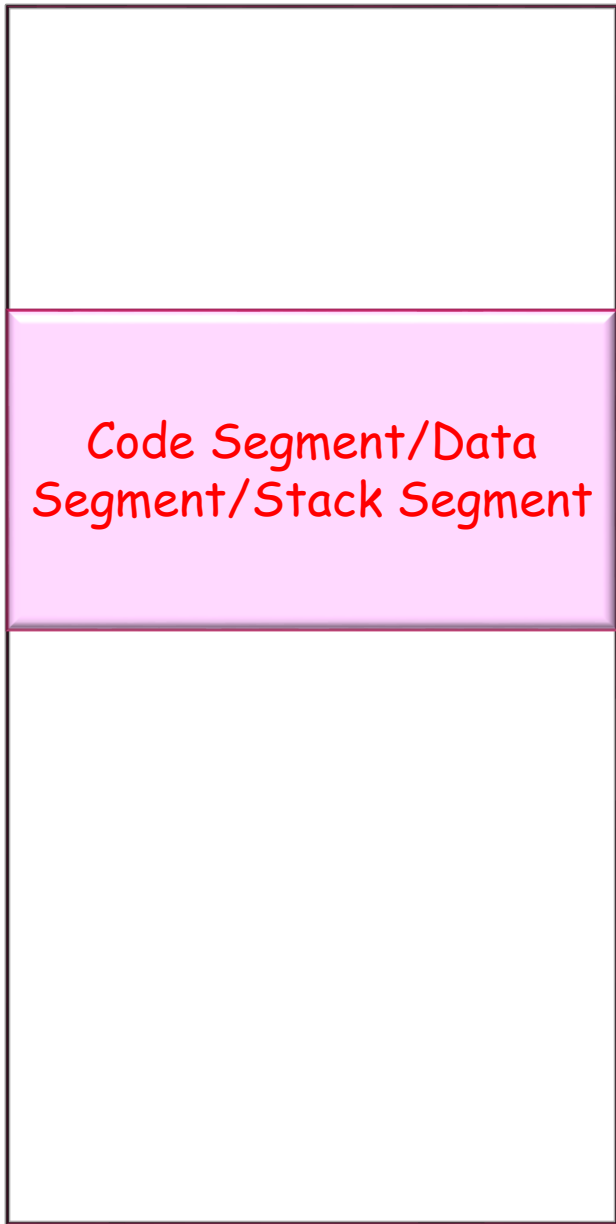
30000_H

Code Segment/Data Segment/Stack Segment

$3FFFF_H$

Ex1

```
.model tiny
.data
DATA1  DB      23
DATA2  DW      9999h
DATA3  DW      9999
ARRAY  DW      01,02,03,04,05,06,07,08
.code
.startup
        MOV    BX,DATA2
        MOV    CX,DATA3
        MOV    DATA1,BL
        MOV    DL,DATA1
        MOV    DI,0002H
        MOV    AX, ARRAY [DI]
.exit
end
```

Ex 2

.DATA
DATA1 DB   23H
ARRAY DW   01,02,03,04,05,06,07,08

.CODE

        MOV   DX, ARRAY
        MOV   CL, DATA1

        MOV   BX, OFFSET ARRAY   BX will get effective address of Array
        MOV   AL,[BX]

Write an ALP to find the greatest signed no. From a set of 10 bytes stored at <u>array</u>. The greatest no. must be stored at location *RES*

80X86 ISA & PROGRAMMING

COMPARE INSTRUCTIONS

COMPARE INSTRUCTION

- Compare instruction is a subtraction that changes only - the flag bits

- CMP Destination, Source
- CMP CL, [BX]
- CMP AX, 2000_H
- CMP [DI], CH

CMP   CX, BX

CX = BX   CF-0 ZF-1 SF-0

CX < BX   CF-1 ZF-0 SF-1

CX > BX   CF-0 ZF-0 SF-0

OF, PF, ACF –depends on data

Unsigned Comparision

TO CHECK RESULT OF COMPARE - LOGICAL

JA/JNBE	CF-0 AND ZF-0
JAE/JNB/JNC	CF-0
JB/JC/JNAE	CF-1
JBE/JNA	CF-1 OR ZF-1
JE/JZ	ZF-1
JNE/JNZ	ZF-0

Unsigned

TO CHECK RESULT OF COMPARE -ARITHMETIC

Signed

JG/JNLE	SF $\oplus$ OF-0, AND ZF-0
JGE/JNL	SF $\oplus$ OF-0
JL/JNGE	SF $\oplus$ OF-1
JLE/JNG	SF $\oplus$ OF-1 OR ZF-1
JE/JZ	ZF-1
JNE/JNZ	ZF-0

WRITE AN ALP TO FIND THE GREATEST SIGNED NO. FROM A SET OF 10 BYTES STORED AT **ARRAY**. THE GREATEST NO. MUST BE STORED AT LOCATION *RES*

```
.model tiny
.data
ARRAY          DB          91_H,02_H,83_H,FF_H,75_H,06_H,07_H,47_H ,12_H ,90_H
RES            DB          ?
.code
.startup

               LEA         BX, ARRAY
               MOV         CL,0A_H
               MOV         AL,[BX]
               DEC         CL
               INC         BX
X2:            CMP         AL,[BX]
               JGE         X1
               MOV         AL,[BX]
X1:            INC         BX
               DEC         CL
               JNZ         X2
               MOV         RES,AL

 .exit
 end
```

80X86 ISA & PROGRAMMING

PROGRAM MODELS

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

* Flat Model –Special type of Tiny Model for 32-bit

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
```

align
is assembler
directive which
tells it to store any future data at only
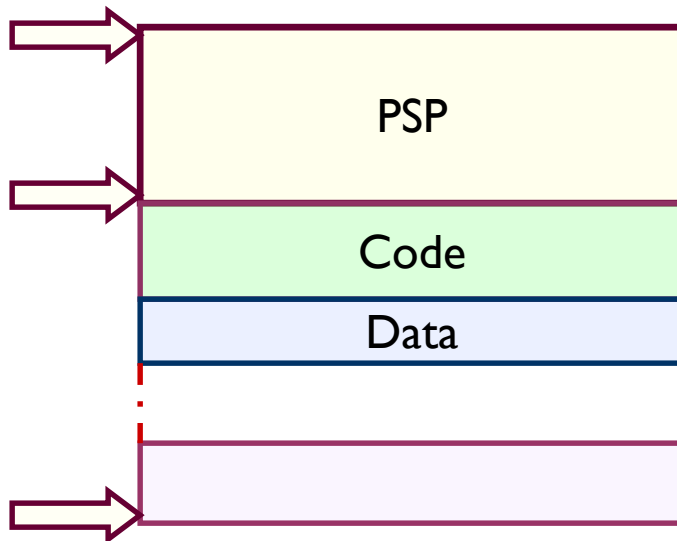even index

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

Memory Map

Model Tiny

CS, DS, SS, ES
$0B4F_H$

IP

SP

PSP

Code 0100_H

Data $010C_H$

$FFFE_H$

code seg, data seg, stack seg, ES all are stored in same 64k area psp

.asm file- extension of assembly code
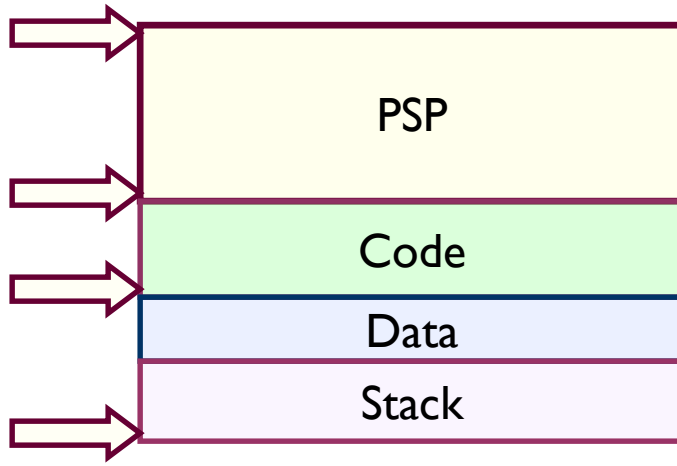executabable formed by .tiny is .comfile

Memory Map

Model Small

$0B50_H$ ES

$0B60_H$ CS:IP

$0B62_H$ DS

$0B62_H$ SS:SP

PSP

Code 0000_H

Data 0020_H

Stack

0410_H

```
; 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
```