

Microprocessors

Hafta 2

ADD Instruction

ADD instruction, şu şekilde kullanılır.

ADD hedef, kaynak ; kaynak, hedefe eklenir. Hedefin değeri değişir.

ADD instruction; CPU'ya hedef ve kaynak operandlarının toplamayı ve sonucu hedefte yazdırması için komut gönderir. 25H ve 34H sayılarını toplamak için her iki sayı registerlara taşınır ve toplanır.

MOV AL, 25H ; 25 sayısını AL registerine kopyala
MOV BL, 34H ; 34 sayısını BL registerine kopyala
ADD AL, BL ; AL = AL + BL

Yukarıdaki program çalıştırıldığında AL'nin sonuca 59 olurken BL'nin sonuçu 34H'dir. Fark edildiği üzere AL'nin değeri değişirken BL'nin değeri değişmemiştir. Bunun sebebi ADD instruction'unda kaynağı(BL) hedef(AL) eklenmesidir. Yukarıdaki program, kullanılan registerlara göre

daha farklı yazılabılır:

MOV DH, 25H ; 25'i DH'ye kopyala
MOV CL, 34H ; 34'ü CL'ye kopyala
ADD DH, CL ; DH = DH + CL

Yukarıdaki program çalıştırıldığında DH=59H ve CL=34H olacaktır.

ADD instruction'ı kullanabilmek için her zaman 2 tane sayı, 2 farklı register'lara taşımamız şart değildir. Aynı işlem şu şekilde de yapılabilir.

MOV DH, 25H ; ilk operandı DH'ye kopyala
ADD DH, 34H ; DH'ye 34H sayısını ekle.

Yukarıdaki ADD işleminde, bir register'da değer varken, kaynak değer operand olarak gelmiştir. Bu operand'a "immediate operand" denir. MOV ve ADD instructionlarında kaynak değer register da olabilir immediate operand da olabilir.

8 bit bir register'in taşıyabileceği en büyük değer FFH'dır. FFH'den daha büyük olan değerler AX, BX, CX veya DX gibi 16 bit register'larda tutulmalıdır. Örneğin 34EH ve 6A5 sayısının toplamak istersenkağıdaki program gibi kod yazmalıyız:

MOV AX, 34EH

MOV DX, 6A5H

ADD AX, DX ; AX = AX + DX = 9F3H

Su sekilde de yazılabilir:

MOV CX, 34EH

ADD CX, 6A5H ; $CX = CX + 6A5H = 9F3H$

★ Ek bilgi: 8 bit bir register 16 bit register ile toplanırken 16 bitlik register'in en düşük değerli 8 bit'i ile 8 bit register toplanır. $AX \Rightarrow AH AL$ dir. Örneğin;

o ADD AX, AL

$$\begin{array}{r} AX \Rightarrow AH\ AL \\ +\ AL \Rightarrow \underline{AL} \\ \hline AH(AL+AL) \end{array}$$

o ADD BX, BH

$$\begin{array}{r} BX \Rightarrow BH\ BL \\ +\ BH \Rightarrow \underline{BH} \\ \hline BH(BL+BH) \end{array}$$

Program Segment'lerine Giriş

Tipik bir assembly programı en az üç segment içermektedir. Bunlar code segment, data segment ve stack segment idir. **Code segment**, programın gerçekleştirilmek üzere tasarlandığı görevleri gerçekleştiren assembly dili instructionlarını içerir. **Data segment**, code segmentındaki instructionları işleyeceği verileri (data) saklamak için kullanılır. Bu segment, programın çalışması sırasında kullanılan değişkenleri ve veri yapılarını içerir. **Stack segment** ise geçici verilerin depolandığı ve yönetildiği segment'tır.

Segment'in Kökeni ve Tanımı

Segment, 64k byte'a kadar alana sahip ve 16'ya bölünebilen bir adresle başlayan yapıdır. Segment boyutunun 64k byte olmasının nedeni 8085 mikroişlemcisinin maksimum $64KB^2$ kadar fiziksel belleği desteklemesinden dolayıdır. (16 adres yolu olduğu için $2^{16} = 64KB$ adreslerebilir). 8085'deki bu sınırlama uyumluluk için 8086/88'in tasarımına aktarıldı. 8085'te code, data ve stack için 64KB bellek bulunurken 8086/88'de her kategoriye en fazla 64KB bellek atanabilir. Assembly dili programlamada bu kategoriler code segment, data segment ve stack segment olarak adlandırılır. Bu sebeple 8086/88 mikroişlemcisi herhangi bir zamanda maksimum 64KB code, 64KB data ve 64KB stack isteyebilir. Ancak 20 tane adresleme yoluna sahip olduğu için 1MB (2^{20} byte) bellek aralığına sahiptir.

Fiziksel ve Mantıksal Adres

Intel'in 8086 işlemcisini anlatırken üç tip adresten sıkılıkla bahsedeceğiz. Bunlar **physical address**, **offset address** ve **logical address** idir. **Physical (Fiziksel) address**, 8086 işlemcisinin adres pinterine yerleştirilen ve bellek aragüz devreleri tarafından çözümlenen 20 bitlik adresdir. Bu adres 00000H ile FFFFFFFH arasında mesafe sahip olabilir. RAM veya ROM'daki 1MB alondaki gerçek konumu ifade eder. **Offset address**, 64 kB'lık bir segment aralığı içindeki bir konumu ifade etmektedir. Dolayısıyla bir offset adresi 0000H ile FFFFH arasında olabilir. **Logical (mantıksal) address**, bir segment değeri ve offset adresinden oluşur. Bu bellek erişimini segment:offset çifti olarak ifade eder.

Code Segment

Bir program'ı execute ederken 8086 mikroişlemci instructionları code segment'ten fetch eder. Logical address her zaman CS (code segment) ve IP (instruction pointer) içerir. CS:IP formatında gösterilir. Instruction'un bulunduğu fiziksel adres CS'yi bir heksadesimal basamak sola kaydırıp (Shift işlemi de denir) IP'ye eklenerek oluşturulur. IP, offset adresini içermektedir. Bu işlem sonucunda elde edilen 20 bitlik adres, DİS adres bus'a gönderilir ve bellek çözme devreleri tarafından çözümlenir. Bu önemli konımı açıklamak için CS ve IP değerlerini değerlerini diyagramda gösterildiği gibi versayalım. Offset adresi IP içerisinde bulunur ve değeri 95F3H'dır. Mantıksal adres ise CS:IP'dir ve değeri 2500:95F3H'dır. Fiziksel adres ise 25000 ile 95F3'ün toplamı olan 2E5F3H olacaktır.

Bir instruction'ın fiziksel adresini şu şekilde hesaplayabiliriz:

1. Start with CS.

2	5	0	0
---	---	---	---

2. Shift left CS.

2	5	0	0	0
---	---	---	---	---

3. Add IP.

2	E	5	F	3
---	---	---	---	---

Mikroişlemci, 2E5F3 bellek konumundan instruction'u alacaktır. IP' nin minimum değeri 0000H ve maksimum değeri FFFFH dir. Bu sebeple yukarıdaki örnekteki logical adres aralığı 2500:0000 ile 2500:FFFF arasıdır. Yukarıdaki kod segmentinin en düşük bellek konumunun 25000H ($25000 + 0000$) ve en yüksek bellek konumunun 34FFFFH ($25000 + FFFF$) olacağı anlamına gelmektedir.

Örnek

CS=24F6H ve IP 6341AH iken

a) logical address'i nedir? $\Rightarrow 24F6:6341A$

b) offset address'i nedir? $\Rightarrow 6341A$

c) Fiziksel adresi hesaplayınız. $\Rightarrow 24F60 + 6341A = 2B2AA$

d) alt sınırı bulunuz $\Rightarrow 24F60 + 0000 = 24F60$

e) code segment'in üst sınırını bulunuz $\Rightarrow 24F60 + FFFF$
 $= 34F5F$

Little Endian Dönüşümü

Daha önceki örneklerde 8 bit veya 1 byte data kullanmıştık.
16 bit data kullanmıştık ne olurdu?

Örnek:

MOV AX, 35F3H ; AX'e 35F3H değerini yükle
MOV [1500], AX ; AX registerinin içeriğini 1500H
offset değerindeki yere kopyala

Bunun gibi durumlarda düşük byte değeri düşük
değerli bellek adresine giderken büyük byte değeri
büyük bellek adresine gider. Yukarıdaki örnekte DS:1500
bellek adresinde F3H bulunurken DS:1501 bellek
adresinde 35H bulunur.

Örnek:

DS:6826 = 48 ve DS:6827= 22 olsun.
"MOV BX, [6826]" komutundan sonraki
register içeriğini gösteriniz.

Little endian dönüşümüne göre (Tüm 80x86'tarda
kullanılır) BL, low offset adresi olan DS:6826'nın değerini;
BH ise high offset adresi olan DS:6827'nın değerini içermelidir.

Memory map of the IBM PC

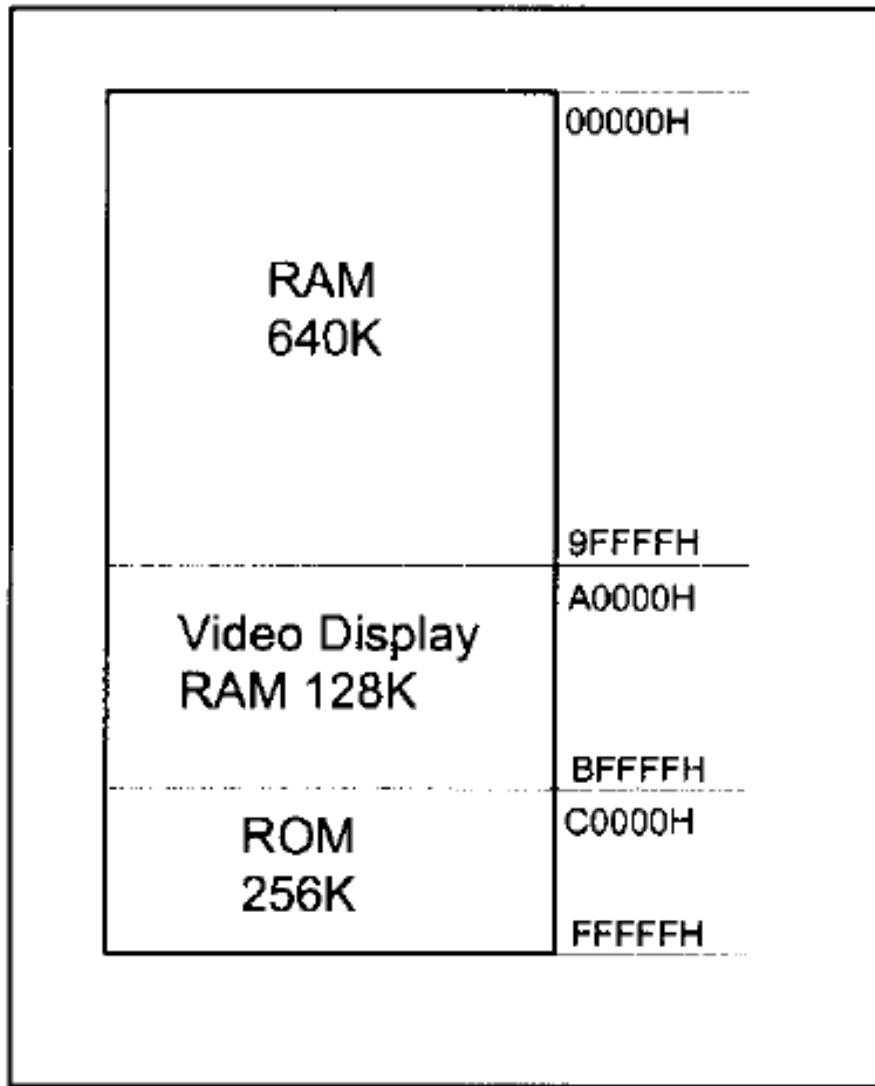


Figure 1-3. Memory Allocation in the PC

★ Intel 8086'daki 1MB'lık adresleme alanı Sadece RAM'e ait degildir. ROM, VRAM gibi verileri adreslemek için 1MB'lık alan içerisinde pay verilir.

Stacks

Example 1-6

Assuming that SP = 1236, AX = 24B6, DI = 85C2, and DX = 5F93, show the contents of the stack as each of the following instructions is executed:

PUSH AX

PUSH DI

PUSH DX

Solution:

SS:1230

SS:1231

SS:1232

SS:1233

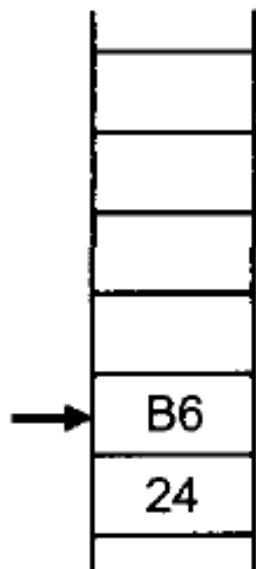
SS:1234

SS:1235

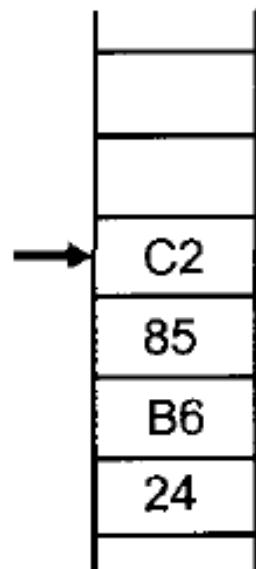
SS:1236

START

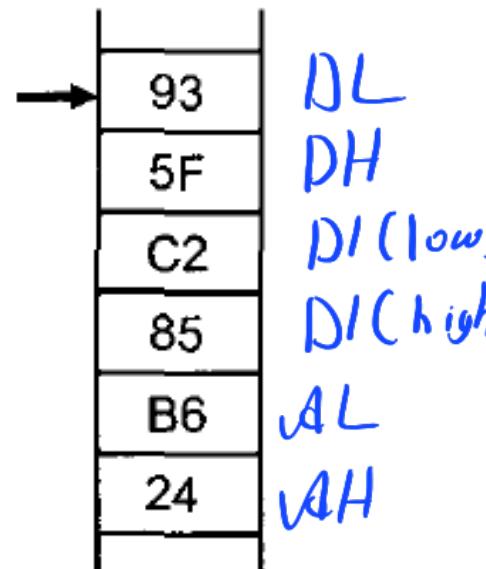
SP = 1236



After
PUSH AX
SP = 1234



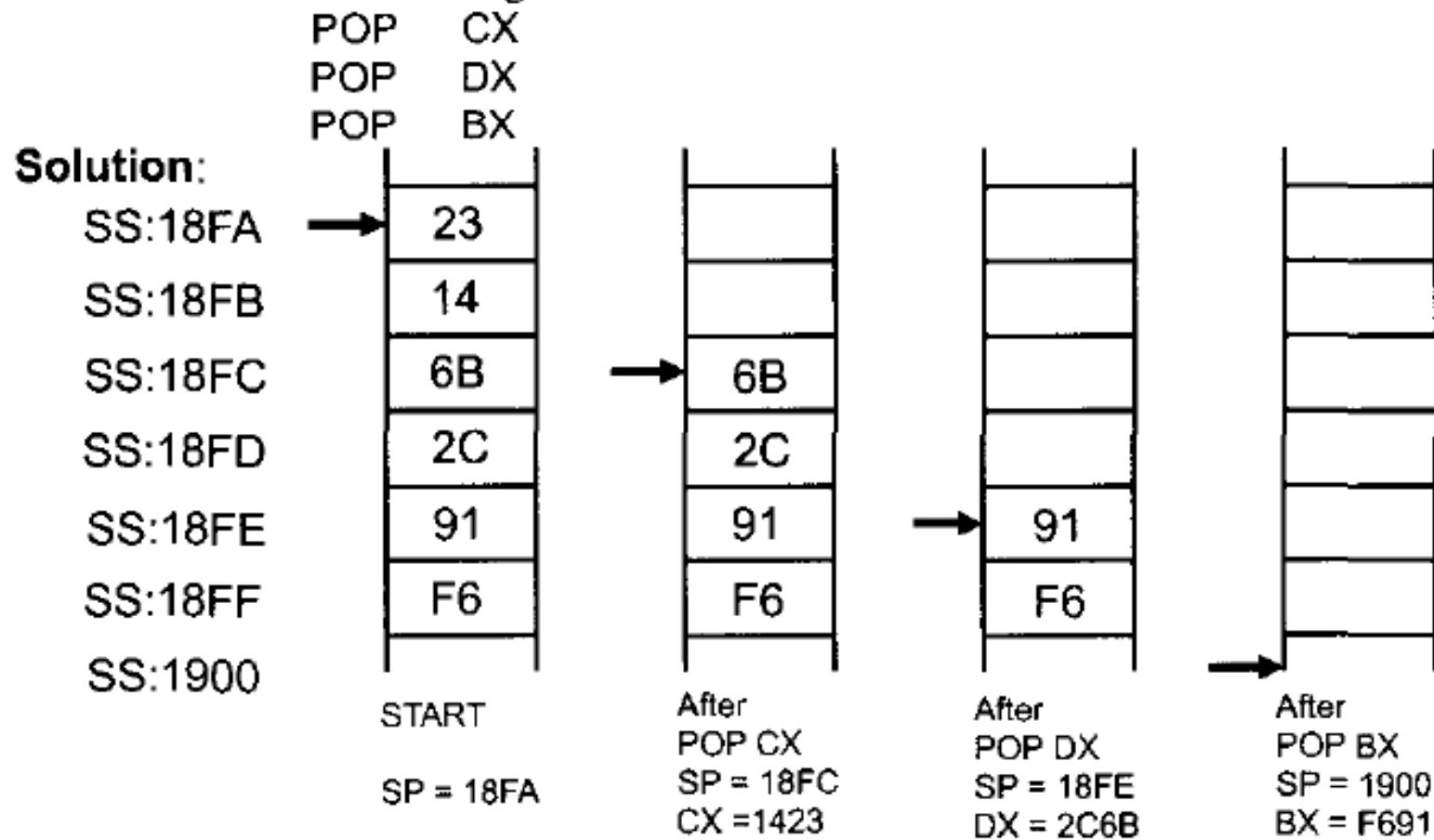
After
PUSH DI
SP = 1232



After
PUSH DX
SP = 1230

Example 1-7

Assuming that the stack is as shown below, and SP = 18FA, show the contents of the stack and registers as each of the following instructions is executed:



Example 1-8

If SS = 3500H and the SP is FFFEH,

- | | |
|---|--|
| (a) Calculate the physical address of the stack. | (b) Calculate the lower range. |
| (c) Calculate the upper range of the stack segment. | (d) Show the logical address of the stack. |

Solution:

- | | |
|--------------------------|--------------------------|
| (a) 44FFE (35000 + FFFE) | (b) 35000 (35000 + 0000) |
| (c) 44FFF (35000 + FFFF) | (d) 3500:FFFE |

Segment Properties

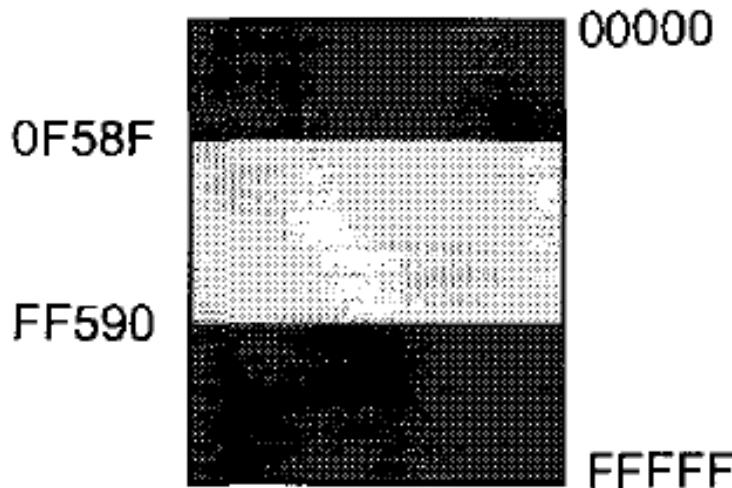
<u>Logical address (hex)</u>	<u>Physical address (hex)</u>
1000:5020	15020
1500:0020	15020
1502:0000	15020
1400:1020	15020
1302:2000	15020

Example 1-9

What is the range of physical addresses if CS = FF59?

Solution:

The low range is FF590 (FF590 + 0000). The range goes to FFFF and wraps around, from 00000 to 0F58F (FF590 + FFFF = 0F58F), which is illustrated below.



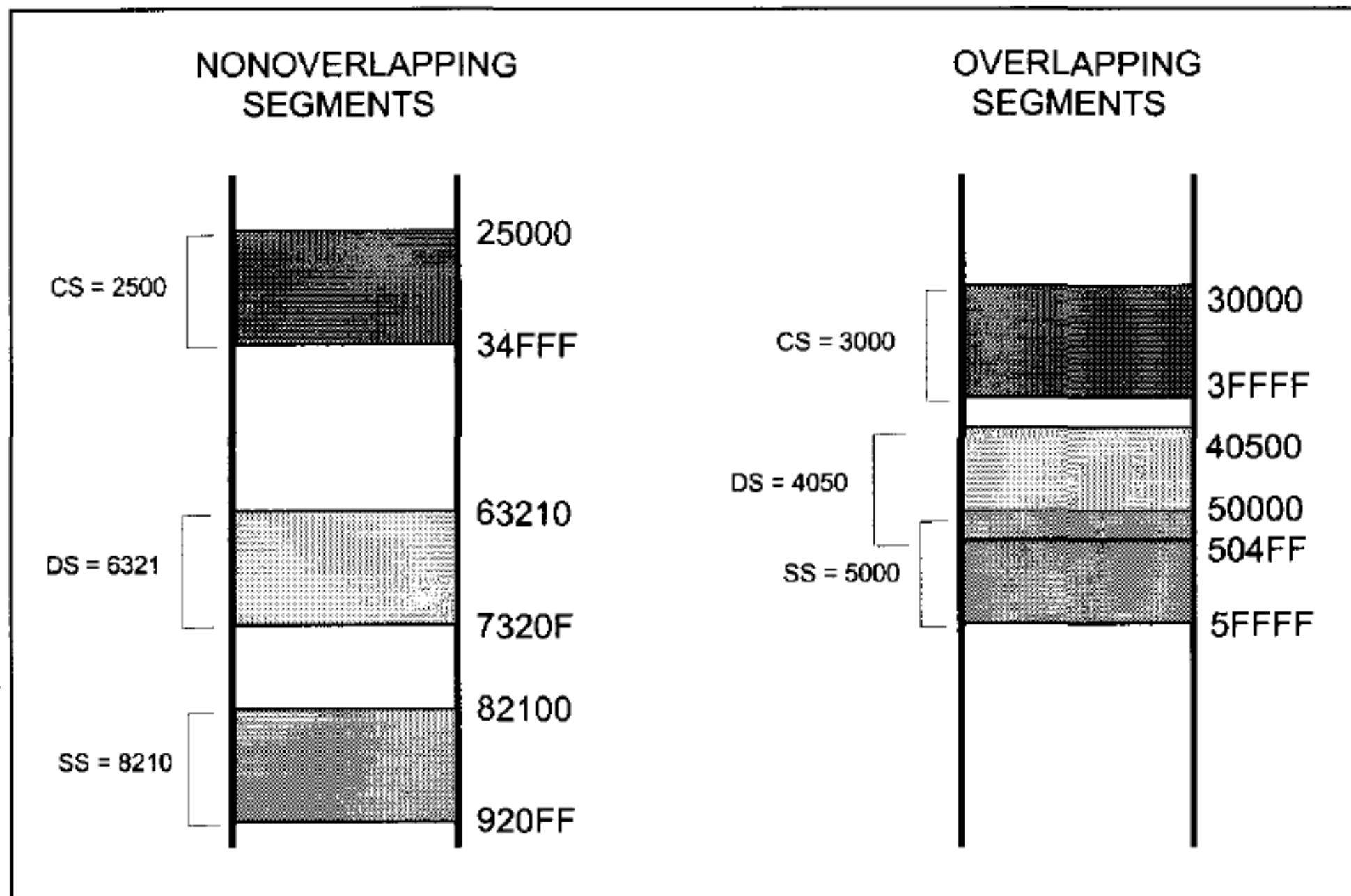


Figure 1-4. Nonoverlapping vs. Overlapping Segments

Flag register

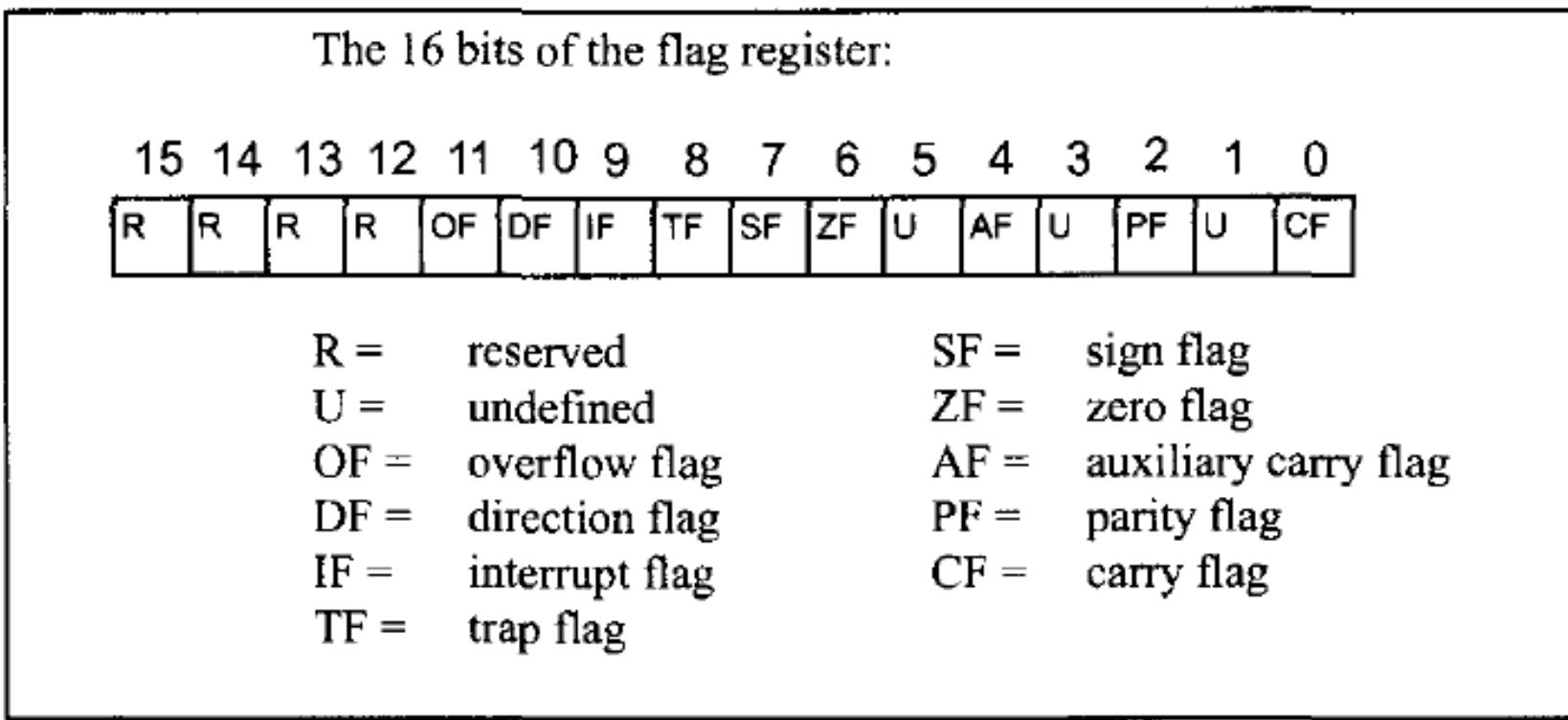


Figure 1-5. Flag Register

(Reprinted by permission of Intel Corporation, Copyright Intel Corp. 1989)

CF, the Carry Flag. This flag is set whenever there is a carry out, either from d7 after an 8-bit operation, or from d15 after a 16-bit data operation.

PF, the Parity Flag. After certain operations, the parity of the result's low-order byte is checked. If the byte has an even number of 1s, the parity flag is set to 1; otherwise, it is cleared.

AF, Auxiliary Carry Flag. If there is a carry from d3 to d4 of an operation, this bit is set; otherwise, it is cleared (set equal to zero). This flag is used by the instructions that perform BCD (binary coded decimal) arithmetic.

ZF, the Zero Flag. The zero flag is set to 1 if the result of an arithmetic or logical operation is zero; otherwise, it is cleared.

SF, the Sign Flag. Binary representation of signed numbers uses the most significant bit as the sign bit. After arithmetic or logic operations, the status of this sign bit is copied into the SF, thereby indicating the sign of the result.

TF, the Trap Flag. When this flag is set it allows the program to single-step, meaning to execute one instruction at a time. Single-stepping is used for debugging purposes.

IF, Interrupt Enable Flag. This bit is set or cleared to enable or disable only the external maskable interrupt requests.

DF, the Direction Flag. This bit is used to control the direction of string operations, which are described in Chapter 6.

OF, the Overflow Flag. This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit. In general, the carry flag is used to detect errors in unsigned arithmetic operations. The overflow flag is only used to detect errors in signed arithmetic operations.

Example 1-10

Show how the flag register is affected by the addition of 38H and 2FH.

Solution:

MOV	BH,38H	;BH= 38H
ADD	BH,2FH	;add 2F to BH, now BH=67H

$$\begin{array}{r} & \text{38} & 0011 & 1000 \\ + & \underline{\text{2F}} & \underline{0010} & \underline{1111} \\ & \text{67} & 0110 & 0111 \end{array}$$

CF = 0 since there is no carry beyond d7

PF = 0 since there is an odd number of 1s in the result

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 0 since d7 of the result is zero

Example 1-11

Show how the flag register is affected by

MOV	AL,9CH	;AL=9CH
MOV	DH,64H	;DH=64H
ADD	AL,DH	;now AL=0

Solution:

$$\begin{array}{r} & \text{9C} & 1001 & 1100 \\ + & \underline{\text{64}} & \underline{0110} & \underline{0100} \\ & \text{00} & 0000 & 0000 \end{array}$$

CF=1 since there is a carry beyond d7

PF=1 since there is an even number of 1s in the result

AF=1 since there is a carry from d3 to d4

ZF=1 since the result is zero

SF=0 since d7 of the result is zero

Example 1-12

Show how the flag register is affected by

MOV	AX,34F5H	;AX= 34F5H
ADD	AX,95EBH	;now AX= CAE0H

Solution:

	34F5	0011	0100	1111	0101
+	<u>95EB</u>	1001	0101	1110	<u>1011</u>
	CAE0	1100	1010	1110	0000

CF = 0 since there is no carry beyond d15

PF = 0 since there is an odd number of 1s in the lower byte

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 1 since d15 of the result is one

Example 1-13

Show how the flag register is affected by

```
MOV BX,AAAAH ;BX= AAAAH  
ADD BX,5556H ;now BX= 0000H
```

Solution:

$$\begin{array}{r} \text{AAAA} \\ + \text{5556} \\ \hline \text{0000} \end{array} \quad \begin{array}{cccccc} 1010 & 1010 & 1010 & 1010 \\ 0101 & 0101 & 0101 & 0110 \\ \hline 0000 & 0000 & 0000 & 0000 \end{array}$$

CF = 1 since there is a carry beyond d15

PF = 1 since there is an even number of 1s in the lower byte

AF = 1 since there is a carry from d3 to d4

ZF = 1 since the result is zero

SF = 0 since d15 of the result is zero

Example 1-14

Show how the flag register is affected by

MOV	AX,94C2H	;AX=94C2H
MOV	BX,323EH	;BX=323EH
ADD	AX,BX	;now AX=C700H
MOV	DX,AX	;now DX=C700H
MOV	CX,DX	;now CX=C700H

Solution:

	94C2	1001	0100	1100	0010
+	<u>323E</u>	0011	0010	0011	1110
	C700	1100	0111	0000	0000

After the ADD operation, the following are the flag bits:

CF = 0 since there is no carry beyond d15

PF = 1 since there is an even number of 1s in the lower byte

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 1 since d15 of the result is 1

Use of the zero flag for looping

```
        MOV  CX,05      ;CX holds the loop count
        MOV  BX,0200H    ;BX holds the offset data address
        MOV  AL,00      ;initialize AL
ADD_LP: ADD  AL,[BX]    ;add the next byte to AL
        INC  BX        ;increment the data pointer
        DEC  CX        ;decrement the loop counter
        JNZ ADD_LP     ;jump to next iteration if counter not zero
```