

# *Microprocessors*

## *Hafta 5, 6, 7*

*Ö. Etkin*

### **CALL İfadesi**

CALL instruction'u, bir prosedür çağrılmak için kullanılır. CALL, sıkılıkla kullanılması gereken görevleri yapan prosedürleri çağrılmak için kullanılır. Bu, programı daha yapılandırılmış duruma getirir.

CALL instruction'ını kullanırken hedef adres mevcut segment içinde olabilir. Bu durumda **NEAR call(çağırı)** olacaktır. Hedef adres mevcut segment dışında olduğunda **FAR call** olacaktır. Çağrılan altprogramın yürütülmesinden sonra mikroişlemcinin nereye-doneceğini bilmesini sağlamak için mikroişlemci otomatik olarak CALL instructiondan sonraki komutun adresini stack üzerine kaydeder. NEAR call ile sadece IP(instruction pointer) stack'e kaydedilirken FAR call ile hem CS(code segment) hem de IP stack'e kaydedilir. Bir altprogram çağrıldığında(call işlemi), kontrol alt programa(subroutine) verilir ve işlemci IP'yi kaydeder ve yeni konumdan instruction'lar fetch edilir. Alt programın execute aşaması tamamlandığında kontrol, çağrıran komuta iade edilir. Kontrolün alt programdan ana programa iadesi yapılması için **RET(return)** instruction'u kullanılması zorunludur.

Program FAR olarak ayarlandıysa assembler CS ve IP'yi call işlemindeki konumuna geri getirir. Near olarak ayarlandıysa assembler sadece IP'yi call işlemindeki konumuna geri getirir.

Örneğin:

SP=FFEHE olsun ve aşağıdaki program kod parçası DEBUG işlemine sokulsun:

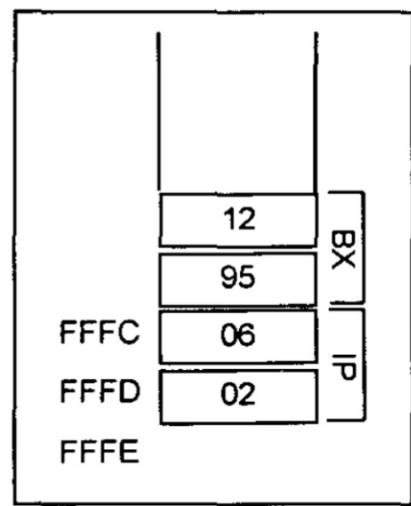


Figure 2-5. IP in the Stack

12B0:0200 BB1295 MOV BX,9512

12B0:0203 E8FA00 CALL 0300

12B0:0206 B82F14 MOV AX,142F

CALL instruction'u NEAR call olduğunda code segment değişmeyeceği için sadece IP stack üzerine kaydedilir. Yukarıdaki durumda CALL ifadesinden sonraki IP adresi stack üzerine kaydedilir. Yani "MOV AX,142F" instruction'ını taşıyan 0206 adres numarası IP'ye taşınır.

Alt programın son instruction'u RET olmak zorundadır. RET instruction'u CPU'ya stack'in en üstündeki 2 byte veriyi IP içine POP eder(yani stack içerisindeki 2 byte'lık değeri çıkarıp IP'ye yerleştirir) ve 0206 offset adresindeki instruction'u execute eder. Bu sebeple SP'yi değiştiren PUSH ve POP instruction'ları birbiriyle eşleşmelidir. Diğer değişle : her PUSH işlemi için POP işlemi yapmak gereklidir.

```
12B0:0300 53 PUSH BX  
12B0:0301 ... .....  
..... .... ..  
12B0:0309 5B POP BX  
12B0:030A C3 RET
```

## Assembly Dili Alt Programları

Assembly dili programlamada bir tane main ve main içerisinde çalışan birden fazla alt programların(subroutine) olması yaygındır. Bu durum her alt programı birbirinden farklı modüller haline getirebilme imkanı vermektedir.

```
.CODE  
MAIN PROC FAR ;THIS IS THE ENTRY POINT FOR DOS  
MOV AX,@DATA  
MOV DS,AX  
CALL SUBR1  
CALL SUBR2  
CALL SUBR3  
MOV AH,4CH  
INT 21H  
ENDP  
  
SUBR1 PROC  
...  
RET ;Main'e geri dönmemek için gerekli  
ENDP ;Prosedürü bitirmek için  
  
SUBR2 PROC  
...  
RET  
ENDP  
  
SUBR3 PROC  
...  
RET  
ENDP  
  
END MAIN ;THIS IS THE EXIT POINT
```

Figure 2-6. Shell of Assembly Language Subroutines

## Veri Tipleri ve Veri Tanımı

Assemblerler, 80x86 mikroişlemcinin çeşitli veri tiplerini destekler. Ayrıca onları tanımlayan ve bellek ayıran veri direktifleri sağlar. Bu bölümde, bu direktifleri ve 80x86'nın farklı veri veri tiplerini temsil etmek için nasıl kullanıldıklarını inceleyeceğiz.

### 80x86 Veri Tipleri

Intel 8088/86 mikroişlemcisi çeşitli veri tiplerini desteklemektedir. Ancak hiçbir veri 16 bit uzunluğu geçemez. Bunun sebebi register'ların maksimum 16 bit veri tutabilmeleridir.

Bir programcının işi ise büyük verileri 16 bitlik parçalara ayırarak register'lar içine yerleştirip CPU'nun yapabilmesi için hazırlamaktır.

8088/86 işlemcideki verip tipleri 8 ya da 16 bit uzunluğunda, olabilir ve pozitif ya da negatif değer alabilirler. Eğer bir sayı 8 bit uzunluktan daha az bir değerse bu değer yüksek değerli bitler 0 olacak şekilde yazılmalıdır. Aynı şekilde eğer bir sayı 16 bir uzunluktan daha az değerse bu değer yüksek değerli bitler 0 olarak şekilde yazılmalıdır. Örneğin 5 sayısı binary sistemde 3 bit uzunluğundadır (101). Ancak 8088/86 işlemci sayıyı "05" olarak ya da "0000 0101" binary sistem olarak kabul etmektedir. 514 sayısı binary sistemde "10 0000 1010" olarak yazılır ama 8088/86 işlemci bunu "0000 0010 0000 0010" olarak kabul etmektedir.

## Assembler Data Direktifleri

80x86 mimarisine göre tasarlanmış tüm assembler'lar (8088,8086,80188,80186, pentium vs.) data gösterimleri için standard haline getirilmiştir. Bazı 80x86 mimarisinde kullanılan data direktifleri tüm yazılım ve donanım üreticileri tarafından desteklenmektedir.

### ORG (Origin)

ORG, offset adresinin başlangıcını belirtmek için kullanılır. ORG ifadesinden gelen sayı decimal veya hexadecimal olabilir. Eğer verilen sayının sonu "H" ile bitmiyorsa bu, sayının decimal olduğunu belirtir ve assembler bunu hexadecimal'e çevirir. ORG direktifi genel olarak data alanlarını ayırmak için kullanılsa da code segment'teki offset değerini(IP) değiştirmek için de kullanılır

### DB(Define Byte)

DC direktifi, assembler'da en yaygın kullanılan direktiflerden birisidir. Bu direktif bellek ayırma işlemindeki en küçük tanımlanabilir bellek boyutudur. DB; decimal, binary, hex ve ASCII sayılarını tanımlamak için kullanılabilirmektedir. Decimal ifadelerin sonuna "D" eklemek opsyoneldir ancak sayının sonuna binary için "B" , hexadecimal için "H" koymak zorunludur. ASCII'yi belirtmek için string ifadesini tek tırnak ile yazmak yeterlidir ('bunun gibi'). Assembler, sayılar ve karakterler için gerekli ASCII kodunu otomatik atayacaktır.

Aşağıda bazı DB örnekleri bulunmaktadır:

```
DATA1    DB    25      ;DECIMAL
DATA2    DB    10001001B ;BINARY
DATA3    DB    12H      ;HEX
          ORG   0010H
DATA4    DB    '2591'   ;ASCII NUMBERS
          ORG   0018H
DATA5    DB    ?       ;SET ASIDE A BYTE
          ORG   0020H
DATA6    DB    'My name is Joe' ;ASCII CHARACTERS
```

0000 19	DATA1	DB 25	;DECIMAL
0001 89	DATA2	DB 10001001B	;BINARY
0002 12	DATA3	DB 12H	;HEX
0010	ORG	0010H	
0010 32 35 39 31	DATA4	DB '259'	;ASCII NUMBERS
0018	ORG	0018H	
0018 00	DATA5	DB ?	;SET ASIDE A BYTE
0020	ORG	0020H	
0020 4D 79 20 6E 61 6D 65 20 69 73 20 4A 6F 65	DATA6	DB 'My name is Joe'	;ASCII CHARACTERS

List File for DB Examples

ASCII stringlerinin etrafında tek veya çift tırnak kullanılabilir. Bu, "N'aber" gibi tek tırnak içermesi gereken stringler için yararlı olabilmektedir.

## DUP (Duplicate)

DUP, verilen karakter sayısı kadar ifadenin çoğaltılması için kullanılır. Bu gereksiz fazla kod yazmayı engellemektedir. Örneğin aşağıdaki 2 kodu kıyaslayabilirsiniz:

```
ORG 0030H
DATA7 DB 0FFH,0FFH,0FFH,0FFH,0FFH,0FFH ;FILL 6 BYTES WITH FF
ORG 38H
DATA8 DB 6 DUP(0FFH) ;FILL 6 BYTES WITH FF
; the following reserves 32 bytes of memory with no initial value given
ORG 40H
DATA9 DB 32 DUP (?) ;SET ASIDE 32 BYTES
;DUP can be used inside another DUP
; the following fills 10 bytes with 99
DATA10 DB 5 DUP (2 DUP (99)) ;FILL 10 BYTES WITH 99
```

0030	ORG	0030H	
0030 FF FF FF FF FF FF	DATA7	DB 0FFH,0FFH,0FFH,0FFH,0FFH,0FFH	; 6 BYTES = FF
0038	ORG	38H	
0038 0006[ FF ]	DATA8	DB 6 DUP(0FFH)	;FILL 6 BYTES WITH FF
0040	ORG	40H	
0040 0020 [ ?? ]	DATA9	DB 32 DUP (?)	;SET ASIDE 32 BYTES
0060	ORG	60H	
0060 0005[ 0002[ 63 ] ]	DATA10	DB 5 DUP (2 DUP (99))	;FILL 10 BYTES WITH 99

List File for DUP Examples

## DW (Define Word)

DW, bellekte 2 byte alan ayırmak için kullanılır. DW register'ların 16 bit genişliğinde olan işlemlerde yaygın olarak kullanılır.

```
        ORG 70H
DATA11 DW 954          ;DECIMAL
DATA12 DW 100101010100B ;BINARY
DATA13 DW 253FH         ;HEX
        ORG 78H
DATA14 DW 9,2,7,0CH,00100000B,5,'HI' ;MISC. DATA
DATA15 DW 8 DUP (?)      ;SET ASIDE 8 WORDS
```

0070	ORG 70H
0070 03BA	DATA11 DW 954 ;DECIMAL
0072 0954	DATA12 DW 100101010100B ;BINARY
0074 253F	DATA13 DW 253FH ;HEX
0078 0009 0002 0007 000C 0020 0005 4849	ORG 78H
0086 0008[ ????]	DATA14 DW 9,2,7,0CH,00100000B,5,'HI' ;MISC. DATA
	DATA15 DW 8 DUP (?) ;SET ASIDE 8 WORDS

List File for DW Examples

## EQU(Equate)

Bu ifade bellek konumunu işgal etmeden sabit bir değer atamak için kullanılır. EQU data segment dışında, code segment ortasında kullanılabilir. EQU kullanılırken immediate adresleme modundan yararlanılır.

COUNT EQU 25

instruction'lar execute edildiği sırada "MOV CX, COUNT" instruction'ının çalıştığını varsayıdığımızda CX register'ına COUNT'un değeri olan 25 aktarılacaktır.

## DD(Define Doubleword)

DD direktifi, bellek üzerinden bir değişkene 4 byte'luk (2 word'lük) alan ayırır. Bu değişkenler decimal, binary ve hexadecimal olabilir. Verilerin bitlerinin yerleştirilmesi DW'deki kısımda bahsettiğim gibidir. Verilerin bellek üzerine yerleşmesi de daha önceden bahsedilen kurala göredir. Küçük byte değeri küçük adres değerine, büyük byte değeri büyük adres değerine taşınır. Aşağıda bazı DD örnekleri verilmiştir.

```
ORG 00A0H
DATA16 DD 1023          ;DECIMAL
DATA17 DD 10001001011001011100B ;BINARY
DATA18 DD 5C2A57F2H        ;HEX
DATA19 DD 23H,34789H,65533
```

00A0	000003FF	ORG 00A0H	
00A0	0008965C	DATA16 DD 1023	;DECIMAL
00A4	5C2A57F2	DATA17 DD 10001001011001011100B	;BINARY
00A8	00000023 00034789	DATA18 DD 5C2A57F2H	;HEX
00AC	0000FFF0	DATA19 DD 23H,34789H,65533	

List File for DD Examples

## DQ (Define Quadword)

DQ, bellekte 8 byte'luk (4 word'lük) alan ayırır. DQ 64 bit uzunluğundaki verileri temsil edebilir.

```
ORG 00A0H
DATA16 DD 1023          ;DECIMAL
DATA17 DD 10001001011001011100B ;BINARY
DATA18 DD 5C2A57F2H        ;HEX
DATA19 DD 23H,34789H,65533
```

00A0	000003FF	ORG 00A0H	
00A0	0008965C	DATA16 DD 1023	;DECIMAL
00A4	5C2A57F2	DATA17 DD 10001001011001011100B	;BINARY
00A8	00000023 00034789	DATA18 DD 5C2A57F2H	;HEX
00AC	0000FFF0	DATA19 DD 23H,34789H,65533	

List File for DD Examples

## DT (Define Ten Bytes)

DT, bellek alanı ayırma işleminde BCD sayılarına yer ayırmak için kullanılır. DT'nin kullanımıyla ilgili alıştırmalar daha sonra gösterilecektir. Şimdilik DT'nin bellekte nasıl yer kapladığını gözlemleyiniz. Fark edildiği üzere verinin sonuna "H" sembolü koymak gerekli değildir. Bu bellekte 10 byte alan ayırır ama en fazla 18 haneli sayı girilebilir.

```
        ORG 00E0H  
DATA23  DT 867943569829 ;BCD  
DATA24  DT ?           ;NOTHING
```

00E0	ORG 00E0H			
00E0	299856437986000000	DATA23	DT 867943569829	;BCD
00EA	00	DATA24	DT ?	;NOTHING
00	00000000000000000000			

### List File for DT Examples

DT işlemi ayrıca 10 byte'luk integer değer tanımlamak için de kullanılabilir. Bunun için sayının sonuna "D" işaretini konur.

DEC DT 65535d ; assembler bunu decimal sayıdan  
; hexadecimal sayıya çevirir ve saklar

Aşağıdaki örnekte bir veri kısmının memory dump'ı bulunmaktadır. Bu dump örneği verilerin bellekte nerede tutulduğuna öğrenmeye dair çok önemlidir. Memory dump'a bakıldığından verilerin little endian dönüşümüne göre sıralı bir şekilde durduğu görülmektedir. Yani least significant byte(en düşük değerli byte) düşük adres konumuna yerlesirken most significant byte büyük adres konumuna yerlesir.

Örneğin "DATA20 DQ 4523C2" ifadesinin bellek adresindeki yerine bakalım. Bellek offset adresi 00C0H ile başlasın. 00C0H offset adresine least significant byte olan "C2" atılır. Daha sonra "23" sayısı 00C1H offset adresine atılır. En son ise most significant bit olan "45" sayısı 00C2H offset adresine atılır.

Ayrıca bilinmelidir ki ASCII verilerini saklamak için sadece DB direktifi kullanılmalıdır. DD, DQ, DW, DT direktifleri 2 byte'dan daha büyük değerli olduğu için assembly hata verecektir. DB ASCII sayıları için kullanıldığından belleğe ters şekilde yerleştirilir. Örneğin "DATA4 DB '2591'" kodunun origin'i 10H olsun. 10H'de 32(ASCII karakterinde 2'yi temsil eder), 11H'de 35 bulunur ve böyle devam eder.

		ASCII '2591' değeri																ASCII String	
-D	1066:0	100	19	89	12	00	00	00	00-00	00	00	00	00	00	00	00	00	.....	
1066:0000		32	35	39	31	00	00	00	00-00	00	00	00	00	00	00	00	00	2591.....	
1066:0010		4D	79	20	6E	61	6D	65	20-69	73	20	4A	6F	65	00	00	My name is Joe..		
1066:0020		FF	FF	FF	FF	FF	FF	00	00-FF	FF	FF	FF	FF	FF	00	00	00	.....	
1066:0030		00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	00	.....	
1066:0040		63	63	63	63	63	63	63	63-63	63	00	00	00	00	00	00	00	cccccccccc.....	
1066:0050		BA	03	54	09	3F	25	00	00-09	00	02	00	07	00	0C	00	00	:.T.?8.....	
1066:0060		20	00	05	00	4F	48	00	00-00	00	00	00	00	00	00	00	00	...OH.....	
1066:0070		00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	00	.....	
1066:0080		FF	03	00	00	5C	96	08	00-F2	57	2A	5C	23	00	00	00	00	....\...rW*\#...	
1066:0090		89	47	03	00	FD	FF	00	00-00	00	00	00	00	00	00	00	00	B#E.....IH.....	
1066:00A0		C2	23	45	00	00	00	00	00-49	48	00	00	00	00	00	00	00	.....	
1066:00B0		00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	00	.....	
1066:00C0		29	98	56	43	79	86	00	00-00	00	00	00	00	00	00	00	00	9.VCy6.....	
1066:00D0																			
1066:00E0																			

Figure 2-7. DEBUG Dump of Data Segment

## Tam Segment Tanımı

### Segmentlerin Tanımı

"SEGMENT" ve "ENDS" direktifleri assembl'er'a segment'in başlangıcını ve bitişini belirtirler. Aşağıdaki formatta yazılırlar:

label SEGMENT [options]

;Segmentte kullanılacak ifadeler buraya yazılır

label ENDS

Label (Etiket de denir) isimlendirme kurallarına uymak zorundadır ve eşsiz olmalıdır. [options] alanı assembl'er'a segmenti düzenlemesi için önemli bilgiler verir. [options] yazmak zorunlu değildir. ENDS label'ı SEGMENT label direktifi içerisinde olmak zorundadır. Full segment tanımında ".MODEL" direktifi kullanılmaz. Dahası ".STACK" , ".DATA" ve ".CODE" direktifleri yerine SEGMENT ve ENDS ifadeleri önceki direktifleri kapsadığı için kullanılır.

### Stack Segment Tanımı

Stack segment "DB 64 DUP (?)" ifadesini içeren kod satırında gösterilmektedir. Bu komut stack için 64 byte boyutunda bellekten alan ayırır.

STSEG SEGMENT ; "SEGMENT" direktifi segmenti  
; başlatır

DB 64 DUP (?) ; bu segment sadece 1 satır içerir

STSEG ENDS ; "ENDS" segmenti sonlandırır

<b>;FULL SEGMENT DEFINITION</b>	<b>;SIMPLIFIED FORMAT</b>
:— stack segment —	.MODEL SMALL
name1 SEGMENT	.STACK 64
DB 64 DUP (?)	:
name1 ENDS	—
:— data segment —	.DATA
name2 SEGMENT	—
;data definitions are placed here	;data definitions are placed here
name2 ENDS	:
:— code segment —	—
name3 SEGMENT	.CODE
MAIN PROC FAR	MAIN PROC FAR
ASSUME ...	MOV AX,@DATA
MOV AX,name2	MOV DS,AX
MOV DS,AX	...
...	...
MAIN ENDP	MAIN ENDP
name3 ENDS	END MAIN
END MAIN	

Figure 2-8. Full versus Simplified Segment Definition

TITLE	PURPOSE: ADDS 4 WORDS OF DATA
PAGE 60,132	
STSEG	SEGMENT
	DB 32 DUP (?)
STSEG	ENDS
DTSEG	SEGMENT
DATA_IN	DW 234DH,1DE6H,3BC7H,566AH
	ORG 10H
SUM	DW ?
DTSEG	ENDS
:	
CDSEG	SEGMENT
MAIN	PROC FAR
	ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
	MOV AX,DTSEG
	MOV DS,AX
	MOV CX,04                         ;set up loop counter CX=4
	MOV DI,OFFSET DATA_IN          ;set up data pointer DI
	MOV BX,00                         ;initialize BX
ADD_LP:	ADD BX,[DI]                     ;add contents pointed at by [DI] to BX
	INC DI                             ;increment DI twice
	INC DI                             ;to point to next word
	DEC CX                             ;decrement loop counter
	JNZ ADD_LP                         ;jump if loop counter not zero
	MOV SI,OFFSET SUM                 ;load pointer for sum
	MOV [SI],BX                         ;store in data segment
	MOV AH,4CH                         ;set up return
	INT 21H                             {return to DOS}
MAIN	ENDP
CDSEG	ENDS
END MAIN	

Program 2-2, rewritten with full segment definition

```

TITLE PURPOSE: TRANSFERS 6 BYTES OF DATA
PAGE 60,132
STSEG SEGMENT
    DB      32 DUP (?)
    ENDS
;
DTSEG SEGMENT
    ORG 10H
DATA_IN DB      25H,4FH,85H,1FH,2BH,0C4H
    ORG 28H
COPY   DB      6 DUP(?)
    ENDS
;
CDSEG SEGMENT
MAIN  PROC FAR
ASSUME CS:CDSEG,DS:DTSEG,SS:STSEG
MOV   AX,DTSEG
MOV   DS,AX
MOV   SI,OFFSET DATA_IN ;SI points to data to be copied
MOV   DI,OFFSET COPY   ;DI points to copy of data
MOV   CX,06H            ;loop counter = 6
MOV_LOOP: MOV  AL,[SI]  ;move the next byte from DATA area to AL
        MOV  [DI],AL  ;move the next byte to COPY area
        INC  SI       ;increment DATA pointer
        INC  DI       ;increment COPY pointer
        DEC  CX       ;decrement LOOP counter
        JNZ  MOV_LOOP ;jump if loop counter not zero
        MOV  AH,4CH   ;set up to return
        INT  21H     ;return to DOS
MAIN
CDSEG ENDP
ENDS
END  MAIN

```

**Program 2-3, rewritten with full segment definition**

## Data Segment Tanımı

Full segment tanımına göre SEGMENT direktifi data segmenti belirtmek için isimlendirilir ve datalar burada olmalıdır. Örneğin:

DTSEG SEGMENT ; SEGMENT direktifi segmenti başlatır  
; datalarını burada tanımla  
DTSEG ENDS ; ENDS direktifi segmenti sonlandırır.

## Code Segment Tanımı

Full segment tanımına göre SEGMENT direktifi data segmenti belirtmek için isimlendirilir ve kodlar burada olmalıdır. Örneğin:

```
CDSSEG SEGMENT ; SEGMENT direktifi segmenti başlatır  
; kodlar buraya yazılır  
CDSSEG ENDS ; ENDS segmenti sonlandırır
```

Full segment tanımında PROC direktifinden hemen sonra (programda kullanılan segment etiketlerinin eşit olduğunu varsayıarak) segment kayıtlarını belirli segmentlerle ilişkilendiren ASSUME direktifi bulunur. Eğer ekstra segment kullanılırsa ES de ayrıca ASSUME ifadesine eklenmelidir. Assume ifadesi gereklidir. Çünkü verilen assembly dili programında birden fazla kod segmentleri, 1, 2, 3 veya daha fazla data segmentleri ve birden fazla stack segment olabilir. Ancak CPU, her segment tipinden sadece 1 tanesine erişebilir. CPU, belirli zamanda seçili segmentlerden sadece 1 tanesini işleme alabilmektedir.

ASSUME assembler'a hangi tanımlı segment direktifinin kullanılması gerektiğini belirtir. Bu ayrıca assembler'a offset adresini hesaplamasında yardımcı olur.

Örneğin "MOV AL, [BX]" kodundaki BX register'i data segmentinin offset adresini tutmaktadır.

Bir DOS programındaki üç segmentten 2 tanesinin uygun değerleri vardır. DS değerleri (ES de kullanılabilir) program tarafından tanımlanmalıdır. Bu tanımlama full segment tanımlama ile yapılabilir:

```
MOV AX, DTSEG ;DTSEG data segment label'ıdır  
MOV DS, AX
```

## EXE VE COM DOSYALARI

Şimdiye kadarki tüm kod örnekleri EXE dosyalarına assemble edilip linker tarafından EXE'lere bağlanmak üzere tasarlanmıştır. Bu bölüm, EXE dosyası gibi yürütülebilir makine kodunu içeren ve DOS seviyesinde çalıştırılabilen COM dosyalarını ele almaktadır. Bu bölümün sonunda bir dosyadan diğerine dönüştürme işlemi gösterilmektedir.

### Neden COM Dosyaları var ?

Bazı durumlarda, sınırlı bellek miktarı nedeniyle çok sıkışık kodlara ihtiyaç duyulabilir. Bu tür durumlarda COM dosyası kullanışlıdır. EXE dosyasının herhangi bir boyutta olabileceğinden, COM dosyalarının geniş çapta kullanılmasının ana nedenlerinden birisidir. Öte yandan COM dosyaları, sınırlı olmalarından dolayı kompaktırlar ve bu sebeple kullanılmaktadır. COM dosyaları 64KB'ı geçemezler 64 KB sınırının nedeni COM dosyasının tek bir segmente sığması gerektigidir. COM dosyasını 64KB boyutuna sınırlamak, verileri kod segmenti içinde tanımlamayı ve ayrıca kod

segmentinin bir bölümünü (son bölüm) stack için kullanmayı gerektirir. COM dosyasının ayırt edici özelliklerinden biri EXE dosyalarının aksine ayrı bir veri segmenti tanımının olmamasıdır.

**Table 2-2: EXE vs. COM File Format**

EXE File	COM File
unlimited size	maximum size 64K bytes
stack segment is defined	no stack segment definition
data segment is defined	data segment defined in code segment
code, data defined at any offset address	code and data begin at offset 0100H
larger file (takes more memory)	smaller file (takes less memory)

TITLE PROG2-4 COM PROGRAM TO ADD TWO WORDS

PAGE 60,132

```
CODSG SEGMENT
        ORG 100H
        ASSUME CS:CODSG,DS:CODSG,ES:CODSG
;—THIS IS THE CODE AREA
PROGCODE PROC NEAR
        MOV AX,DATA1      ;move the first word into AX
        MOV SUM,AX        ;move the sum
        MOV AH,4CH         ;return to DOS
        INT 21H
PROGCODE ENDP
;—THIS IS THE DATA AREA
DATA1    DW 2390
DATA2    DW 3456
SUM      DW ?
;
CODSG ENDS
END    PROGCODE
```

3 Burada  
Once  
datalar  
alinir

Program 2-4

TITLE PROG2-5 COM PROGRAM TO ADD TWO WORDS

PAGE 60,132

CODSG SEGMENT

ASSUME CS:CODSG,DS:CODSG,ES:CODSG

ORG 100H

START: JMP PROGCODE ;go around the data area

;—THIS IS THE DATA AREA

DATA1 DW 2390

DATA2 DW 3456

SUM DW ?

;—THIS IS THE CODE AREA

PROGCODE: MOV AX,DATA1 ;move the first word into AX
ADD AX,DATA1 ;add the second word
MOV SUM,AX ;move the sum
MOV AH,4CH
INT 21H

;

CODSB ENDS

END START

Burada ise jump  
ile code segmente atlanır.  
datalar kodların çalışma  
sırasında ihtiyacı varsa  
getirilir. Program 2-4'e  
göre daha yavaş çalışır.

Program 2-5

# Aritmetik ve Lojik Instruction'lar

## İşaretsiz Toplama ve Çıkarma İşlemi

### İşaretsiz Toplama İşlemi

ADD instruction'ının kullanım şekli şu şekildedir:

ADD hedef, kaynak ; hedef = hedef + kaynak

ADD ve ADC instruction'ları 2 operandı birbirine eklemek için kullanılır. Hedef operand register veya bellek olabilir. Kaynak operand ise register, bellek ya da immediate olabilir.

Memory'den memory'ye aritmetik işlemler 80x86 assembly dilinde asla izin verilmemektedir. Instruction ZF, SF, AF, CF veya PF gibi flag registerleri hangi operandların kullanıldığına göre değiştirebilir.

#### Example 3-1

Show how the flag register is affected by

```
MOV      AL,0F5H  
ADD      AL,0BH
```

#### Solution:

$$\begin{array}{r} F5H \\ + 0BH \\ \hline 100H \end{array} \quad \begin{array}{r} 1111\ 0101 \\ + 0000\ 1011 \\ \hline 0000\ 0000 \end{array}$$

After the addition, the AL register (destination) contains 00 and the flags are as follows:

CF = 1 since there is a carry out from D7

SF = 0 the status of D7 of the result

PF = 1 the number of 1s is zero (zero is an even number)

AF = 1 there is a carry from D3 to D4

ZF = 1 the result of the action is zero (for the 8 bits)

Write a program to calculate the total sum of 5 bytes of data. Each byte represents the daily wages of a worker. This person does not make more than \$255 (FFH) a day. The decimal data is as follows: 125, 235, 197, 91, and 48.

```

TITLE      PROG3-1A (EXE) ADDING 5 BYTES
PAGE       60,132
.MODEL SMALL
.STACK 64
;
        .DATA
COUNT    EQU    05
DATA      DB     125,235,197,91,48
          ORG    0008H
SUM      DW     ?
;
        .CODE
MAIN     PROC   FAR
          MOV    AX,@DATA
          MOV    DS,AX
          MOV    CX,COUNT ;CX is the loop counter
          MOV    SI,OFFSET DATA;SI is the data pointer
          MOV    AX,00 ;AX will hold the sum
BACK:    ADD    AL,[SI] ;add the next byte to AL
          JNC    OVER ;If no carry, continue
          INC    AH ;else accumulate carry in AH
OVER:    INC    SI ;increment data pointer
          DEC    CX ;decrement loop counter
          JNZ    BACK ;if not finished, go add next byte
          MOV    SUM,AX ;store sum
          MOV    AH,4CH
          INT    21H ;go back to DOS
MAIN     ENDP
END     MAIN

```

### Program 3-1a

Write a program to calculate the total sum of five words of data. Each data value represents the yearly wages of a worker. This person does not make more than \$65,555 (FFFFH) a year. The decimal data is as follows: 27345, 28521, 29533, 30105, and 32375.

```

TITLE      PROG3-1B (EXE) ADDING 5 WORDS
PAGE       60,132
.MODEL SMALL
.STACK 64
;
        .DATA
COUNT    EQU    05
DATA      DW     27345,28521,29533,30105,32375
          ORG    0010H
SUM      DW     2 DUP(?)
;
        .CODE
MAIN     PROC   FAR
          MOV    AX,@DATA
          MOV    DS,AX
          MOV    CX,COUNT ;CX is the loop counter
          MOV    SI,OFFSET DATA;SI is the data pointer
          MOV    AX,00 ;AX will hold the sum
          MOV    BX,AX ;BX will hold the carries
BACK:    ADD    AX,[SI] ;add the next word to AX
          ADC    BX,0 ;add carry to BX
          INC    SI ;increment data pointer twice
          INC    SI ;to point to next word
          DEC    CX ;decrement loop counter
          JNZ    BACK ;if not finished, continue adding
          MOV    SUM,AX ;store the sum
          MOV    SUM+2,BX ;store the carries
          MOV    AH,4CH
          INT    21H ;go back to DOS
MAIN     ENDP
END     MAIN

```

### Program 3-1b

## İşaretsiz Çıkarma İşlemi

SUB hedef, kaynak ; hedef = hedef - kaynak

Çıkarma işleminde 80x86 mikroişlemciler (neredeyse tüm modern işlemciler de) 2'nin tümleyeni metodunu kullanır. Dahası her CPU'da toplama devresi vardır. Çıkarma işlemi için ayrı bir çıkarma devresi yapmak maaliyetli olacağı için Intel mühendisleri çıkışma işlemini özel bir toplama işlemiyle gerçekleştirirler.

İşaretsiz çıkışma işlemi 3 aşamada yapılır:

- 1) Kaynak operandın 2'ye tümleyeni alınır
- 2) Kaynak operand hedef operanda eklenir
- 3) Elde (carry) tersine çevrilir

### Example 3-2

Show the steps involved in the following:

MOV	AL,3FH	;load AL=3FH
MOV	BH,23H	;load BH=23H
SUB	AL,BH	;subtract BH from AL. Place result in AL.

#### Solution:

AL	3F	0011 1111	0011 1111	
- BH	- 23	- 0010 0011	+1101 1101	(2's complement)
		1C	1 0001 1100	CF=0 (step 3)

The flags would be set as follows: CF = 0, ZF = 0, AF = 0, PF = 0, and SF = 0. The programmer must look at the carry flag (not the sign flag) to determine if the result is positive or negative.

SUB instruction'ın çalışmasından sonra CF = 0 ise işlem sonucun pozitif olduğunu, CF=1 ise işlem sonucunun negatif olduğunu gösterir. Normalde sonuç 2'nin tümleyeni şeklinde olmasına rağmen NOT ve INC instructionları bunu

değiştirebilir. NOT instruction'u operand'ın 1'in tümleyenini yapar. NOT ile 1'e tümleyeni yapıldıktan sonra INC instruction'u ile operand arttırılarak 2'nin tümleyeni yapılır. Örnekte de gösterilmektedir:

### Example 3-3

Analyze the following program:

;from the data segment:

```
DATA1    DB 4CH
DATA2    DB 6EH
DATA3    DB ?
```

;from the code segment:

MOV	DH,DATA1	;load DH with DATA1 value (4CH)
SUB	DH,DATA2	;subtract DATA2 (6E) from DH (4CH)
JNC	NEXT	;if CF=0 jump to NEXT target
NOT	DH	;if CF=1 then take 1's complement
INC	DH	;and increment to get 2's complement
NEXT:	MOV DATA3,DH	;save DH in DATA3

### Solution:

Following the three steps for "SUB DH,DATA2":

4C	0100 1100	0100 1100	
- 6E	0110 1110	2's comp	+1001 0010
- 22			0 1101 1110

CF=1 (step 3) the result is negative

## SBB (Subtract With Borrow)

Bu instruction çok byte'lı (çok word'lü) sayılarda kullanılır ve alt aperand'ın ödünç alınmasını ele alır. Carry flag 0 ise SBB instructionı SUB gibi davranıştır. Carry flag 1 ise SBB sonuç ifadeden 1 çıkarır. Örneğin aşağıdaki örnekte "PTR" operandına dikkat edin. PTR data pointer'i, tanımlanan boyuttan farklı olduğunda operandın boyutunu belirtmek için yaygın olarak kullanılır.

### Example 3-4

Analyze the following program:

```
DATA_A    DD 62562FAH
DATA_B    DD 412963BH
RESULT   DD ?
```

...

MOV	AX,WORD PTR DATA_A	;AX=62FA
SUB	AX,WORD PTR DATA_B	;SUB 963B from AX
MOV	WORD PTR RESULT,AX	;save the result
MOV	AX,WORD PTR DATA_A +2	;AX=0625
SBB	AX,WORD PTR DATA_B +2	;SUB 0412 with borrow
MOV	WORD PTR RESULT+2,AX	;save the result

### Solution:

After the SUB, AX = 62FA - 963B = CCBF and the carry flag is set. Since CF = 1, when SBB is executed, AX = 625 - 412 - 1 = 212. Therefore, the value stored in RESULT is 0212CCBF.

## İşaretsiz Çarpma İşlemi

**Table 3-1: Unsigned Multiplication Summary**

Multiplication	Operand 1	Operand 2	Result
byte × byte	AL	register or memory	AX
word × word	AX	register or memory	DX AX
word × byte	AL = byte, AH = 0	register or memory	DX AX

## İşaretsiz Bölme İşlemi

**Table 3-2: Unsigned Division Summary**

Division	Numerator	Denominator	Quotient	Rem
byte/byte	AL = byte, AH = 0	register or memory	AL <sup>1</sup>	AH
word/word	AX = word, DX = 0	register or memory	AX <sup>2</sup>	DX
word/byte	AX = word	register or memory	AL <sup>1</sup>	AH
doubleword/word	DXAX = doubleword	register or memory	AX <sup>2</sup>	DX

Notes:

1. Divide error interrupt if AL > FFH.
2. Divide error interrupt if AX > FFFFH.