4.BÖLÜM

4.ASSEMBLER KOMUTLARI

4.1.Veri Transfer Komutları:

4.1.1.Adres Yükleme Komutları:

Bu komutlar, bir saklayıcıya veya bir saklayıcı ile bir segment saklayıcısına bir adres yüklemede kullanılmaktadır. Tablo.4.1.de komutların 3 değişik şeklini göstermektedir.

4.1.1.1. LEA (Load Effective Address):

LEA komutu, bir saklayıcıya operand ile belirtilen adresi yükler. Tablo.4.1'de birinci örnekte, AX saklayıcısı, operand SUBADR içeriği ile değil (yani bu adresteki veri ile değil) SUBADR adresiyle yüklenmektedir.

Tablo.4.1.Adres yükleme komutlarının değişik kullanımları

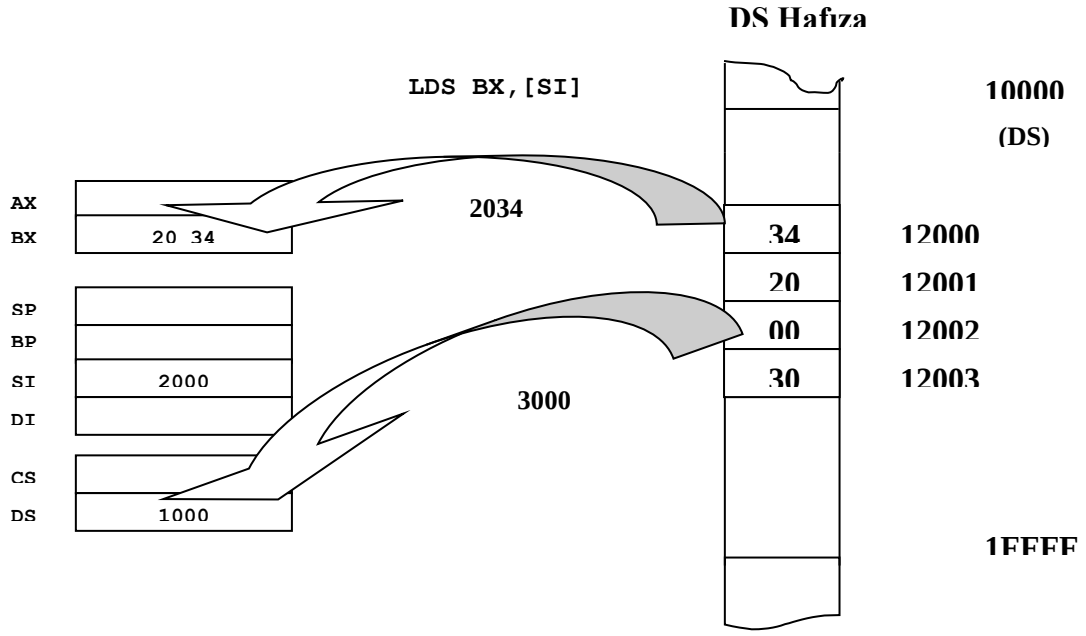
Asembly Dili	***Yapılan İşlem***
LEA AX,SUBADR	AX SUBADR adresiyle yüklenir
LDS DI,LIST	DI ve DS LIST'teki adres ile yüklenir
LES BX,VEC1	BX ve ES VEC1'deki adres ile yüklenir

4.1.1.2. LDS ve LES:

LDS ve LES komutları, bir 16-bit saklayıcıya bir ofset adres ve DS (LDS ile) veya ES (LES ile) segment saklayıcısına 6yeni bir segment adresi yükler. Bu komutlar, yeni ofset ve segment numarasını seçmede, değişik geçerli adresleme modlarından herhangi birisi kullanılır. Bu komutlardan her birinde hafızadan mikroişlemciye iki tane 16-bit kelime, yani toplam 4-byte veri transferi olur. LDS ve LES komutları ile, bir program içinde farklı bir DS ve ES alanlarına işaret edilirken, ofset saklayıcılara da yeni ofset yüklenir. Şekil.4.1'de LDS BX,[SI] komutu ile SI ile işaretli hafıza alanından bir 32-bit sayı BX ve DS saklayıcılarına kopyalanmaktadır.

4.2. Dizi (String) Komutları:

Üç çeşit dizi (string) veri transfer komutu vardır. Bunlar: LODS, STOS ve MOVS. Bu komutlar, mikroişlemci ile hafıza arasında, bir blok veya tek bir byte veya kelime transferinde kullanılır. Bu komutları tanıtmadan, aşağıda önce, bu komutların kullanımında önemli olan, yön (direction) bayrağı (D) ve dizi işlemlerinde kullanılan DI ve SI saklayıcılarının fonksiyonları anlatılacaktır.



Şekil.4.1 LDS BX,[SI] komutu yürütülürken bellek saklayıcılarının durumları.

4.2.1.Yön Bayrağı (D):

Yön bayrağı dizi işlemleri sırasında, DI ve SI saklayıcıları için otomatik-arttırma (D = 0) veya otomatik-azalma (D = 1) çalışma modunu seçer. D bayrağı CLD komutu ile 0'lanır ve STD komutu ile 1'lenir. Yani CLD otomatik-arttırmayı ve STD otomatik-azalmayı seçer. Bir dizi komutu ile yapılan veri transferinden sonra, D bayrağıyla seçilen çalışma moduna göre, eğer veri 1-byte ise DI ve/veya SI 1 arttırılır veya azaltılır. Benzeri şekilde, transfer edilen veri 2 byte ise, bu kez DI ve/veya SI 2 arttırılır ya da azaltılır.DI ve SI

Bir dizi komutunda DI, SI veya her ikisi kullanılabilir. Normalde SI, DS için ve DI, ES için ofset adres olmaktadır. SI için segment atanması, bir segment ön eki (override prefix) ile değiştirilebilir. Bununla beraber DI segment ataması her zaman ES'tir bu atama değiştirilemez.

4.2.2.LODS (Load String)

LODZ komutu SI ile işaretli hafıza hücresinden AL'ye 1 byte veya AX'e 2 byte (kelime) yüklemektedir. Tablo.4.2, LODS komutunun bazı kullanım örneklerini göstermektedir. LODSB ve LODSW komutları bir byte veya bir kelime transferine neden olur. Örneğin, LODSB komutu yürütülürken, önce SI ile işaretli DS alanında bir byte veri AL saklayıcısına kopyalanır. Hemen sonra, SI, D bayrağının durumuna göre, 1 arttırılır veya azaltılır.

4.2.3.STOS (Store String):

STOS komutu, AL veya AX'i DI ile işaretli hafıza hücresine saklanmaktadır. Tablo.4.3 STOS komutunun bazı kullanım örneklerini göstermektedir. LODS'te olduğu gibi, STOS'tan sonra B veya W kullanılarak bir byte veya kelime yapıldığı belirtilir.

Tablo.4.2. LODS komutunun bazı değişik kullanım örnekleri.

Asembly Dili	***Yapılan İşlem***
LODSB	AL=[SI],SI=SI±1
LODSW	AX=[SI],SI=SI±2
LODS BUFFER	AL=[SI],SI=SI±1 (Eğer BUFFER byte ise)
LODS DATA	AX=[SI],SI=SI±2 (Eğer DATA Word ise)

Tablo.4.3. STOS komutunun bazı değişik kullanım örnekleri.

Asembly Dili	***Yapılan İşlem***
STOSB	DI=AL,DI=DI±1
STOSW	DI=AX,DI=DI±2
STOS BUFFER	AL=AL,DI=DI±1 (Eğer BUFFER byte ise)
STOS DATA	AX=AX,DI=DI±2 (Eğer DATA Word ise)

4.2.4.REP ile STOS Komutu:

Tekrar (repeat) REP öneki (prefix) herhangi bir dizi komutuna eklenebilir. Bu önek, bir dizi işleminin, CX sayma değeri sıfır oluncaya kadar devan etmesine neden olur.

Aşağıda verilen örnek program parçasında, BUFFER adresli yerden başlayan 10 byte, sıfır ile temizlenmektedir. Bu işlem peş peşe 10 tane STOS komutuyla veya REP öneki ile bir tane STOS komutuyla yapılabilir. Program ES ve DI saklayıcılarını hafıza adresiyle yükleyen LES komutuyla başlamaktadır. Daha sonra, CX sayacı yüklenmekte ve DI bayrağı sıfırlanarak, her STOS komutu yürütüldükten sonra DI'nın otomatik arttırılması sağlanır.

```
        ; STOS komutunu kullanarak bir hafıza bloğunu temizleme
    LES  DI, BUFFER   ; BUFFER adresini yükle
    MOV  CX, 10       ; sayacı yükle
    CLD               ; otomatik-arttırmayı seç
    MOV  AL, 10       ; hafıza temizlenecek
REP  STOSB            ; BUFFER alanı temizle
```

Yukarıda verilen programda, LES komutuyla yüklenen adres BUFFER kullanılan assembler'da bir yol ile tanımlanmalıdır. Genellikle DD (Define-Double Word) yalancı-işlemi (pseudo- operation) 32-bit hafıza işaretçisi tanımlamada kullanılır.

Hafıza bloğunu temizlemenin, daha hızlı bir yolu REP STOSW komutunu kullanmaktır. Bu durumda sayaç 5 ile yüklenmelidir. Çünkü 5 tane 0000h kelimesi 10 byte hafıza temizler. Aşağıda bu işlemi yapan program parçası verilmiştir.

```
        ; STOS komutunu kullanarak bir hafıza bloğunu temizleme
    LES  DI, BUFFER   ; BUFFER adresini yükle
    MOV  CX, 5        ; sayacı yükle
    CLD               ; otomatik-arttırmayı seç
    MOV  AL, 0        ; hafıza temizlenecek
REP  STOSW            ; BUFFER alanı temizle
```

4.2.5.MOVS (Move String):

En güçlü dizi veri transfer komutu olan MOVS hafızanın bir alanından diğer bir alanına bir byte veya bir kelime aktarımı yapılır. Diğer bir deyişle, bu komut *hafızadan hafızaya* veri

transferi yapar. Bir MOVS komutu, data segment'te (DS) bulunan ve SI saklayıcısı ile adreslenen hafıza hücresini, ES'de bulunan ve DI ile adreslenen hafıza alanına aktarır. Tablo.4.4'te MOVS komutunun bazı değişik kullanımları gösterilmektedir.

Tablo.4.4. MOVS komutunun bazı değişik kullanım örnekleri.

Asembly Dili	***Yapılan İşlem***
MOVSB	[DI]=[SI],DI=DI±1,SI=SI±1
MOVSW	[DI]=[SI],DI=DI±2,SI=SI±2
MOVS BYTE1,BYTE2	[DI]=[SI],DI=DI±1,SI=SI±1
MOVS WORD1,WORD2	[DI]=[SI],DI=DI±2,SI=SI±2

Aşağıdaki verilen örnek program parçasında, BUFFER2 alanından 100 byte BUFFER1 lanına aktarılmaktadır. MOVS kullanılmadan önce, LES ve LDS komutlarıyla hafıza işaretçileri DI ve SI yüklenerek, BUFFER1 ve BUFFER2 alanlarını işaret eder. Daha sonra, MOVS komutu yürütülürken, SI ve DI'da bulunan adreslerin otomatik arttırılması, CLD komutuyla seçilmiş ve CX sayacı 100 ilk değeriyle yüklenmiştir. Bu ön işlemlerden sonra, REP komutuyla kullanılan MOVS komutu veri transferini gerçekleştirir.

```
    LES  DI, BUFFER1   ; BUFFER1 adresini yükle
    LES  SI, BUFFER2   ; BUFFER2 adresini yükle
    CLD                ; otomatik-arttırmayı seç
    MOV  CX, 100       ; 100 byte veri transferi
REP  MOVSB             ; BUFFER2'den BUFFER1'e aktarım
```

4.3.Diğer Veri Transfer Komutları:

Bu gruptaki veri transfer komutaları, çok önemli olmalarına ve çok kullanılmalarına karşın daha önceki gruplara girmemektedir. Bu komutlar IN, OUT; XCHG, XLAT, LAHF ve SAHF 'dir.

4.3.1. IN ve OUT:

Mikroişlemci bu komutlar ile giriş/çıkış cihazları (port'ları) ile haberleşir. 8085A mikroişlemcisinde IN ve OUT komutlarından her biri 2 byte olup birinci byte işlem kodu

sonraki byte prot adresidir. Bu sayede, 8085A, 256 tane giriş ve 256 tane çıkış olmak üzere toplam 512 tane port'a sahip olabilir. 8085A'da veri aktarımı, mikroişlemcinin A (accumulator) saklayıcısı ile 8-bit bir *sabit* adres ile belirtilen (adreslenen) bir dış port arasında olur.

8086/8088 mikroişlemcilerinde veri aktarımı mikroişlemcinin AL veya AX saklayıcısı ile bir *sabit* adres ile veya DX ile belirtilen (adreslenen) ir dış port arasında olur. Tablo.4.5'te IN ve OUT komutlarının bazı değişik kullanım örnekleri verilmektedir.

Tablo.4.5. IN ve OUT komutlarının bazı değişik kullanım örnekleri.

Asembly Dili	***Yapılan İşlem***
IN AL,IPORT	IPORT'tan 8-bit veri AL'ye okunur
IN AX,IPORT	IPORT'tan 16-bit veri AX'ye okunur
IN AL,DX	DX ile işaretli port'tan 8-bit veri AL'ye okunur
IN AX,DX	DX ile işaretli port'tan 16-bit veri AX'e okunur
OUT OPORT,AL	AL'nin içeriği OPORT'a gönderilir
OUT OPORT,AX	AX'in içeriği OPORT'a gönderilir
OUT DX,AL	AL'nin içeriği DX ile işaretli port'a gönderilir
OUT DX,AX	AX'in içeriği DX ile işaretli port'a gönderilir

4.3.2.XCHG:

Bu komut, herhangi bir saklayıcının içeriğini diğer bir saklayıcı veya hafıza hücresiyle değiştirmektedir. XCHG komutu, segment saklayıcılarının içeriğini ve hafızadan hafızaya veri içeriği değiştirmede kullanılmaz.

XCHG komutunu 16-bit AX saklayıcısı ile diğer bir 16-bit saklayıcının içeriklerini değiştirmede kullanılan en etkin ve en hızlı yer değiştirme yönetimidir. Çünkü,bu komut hafızada bir byte yer kaplar. Diğer XCHG komutları, kullanılan adresleme moduna göre, 2 veya daha fazla byte gerektirir. Tablo.4.6, XCHG komutunun bazı değişik kullanım örnekleri verilmektedir.

Tablo.4.6. XCHG komutunun değişik şekilleri.	
Asembly Dili	***Yapılan İşlem***
XCHG saklayıcı,saklayıcı	Saklayıcı içeriklerini değiştir.
XCHG saklayıcı,hafıza	Saklayıcı ile hafıza içeriklerini değiştir.

4.3.3.XLAT:

XLAT (translate) komutu, AL'nin içeriğini hafızada bir tabloda saklı sayıya çevirmektedir. Bu komut, bir kodu diğerine çeviren, doğrudan ***tablo arama (table lookup)*** tekniğinde kullanılır. XLAT komutu, önce AL'nin içeriğini BX saklayıcısına ekleyerek DS için bir hafıza adresi oluşturur. Daha sonra bu adreste saklı veriyi AL saklayıcısına yükler.

Örneğin: bir 7-segment LED gösterge arama tablosunun DISPLAY adresine saklandığını varsayalım. XLAT komutu, AL saklayıcısında bulunacak 0 ile 9 arasında değişecek BCD (binary Coded Decimal) sayıların 7-segment kod karşılığı bulmada kullanılır.

```
MOV   BX, OFFSET DISPLAY   ; DISPLAY adresini yükle
XLAT                       ; AL'deki BCD kodu 7-segment koda çevir
```

4.3.4.LAHF ve SAHF:

Bu komutlar 8085 mikroişlemcisinin yazılımını 8086/8088 çevirmek için tasarlanmıştır ve seyrek olarak kullanılmaktadır. LAHF komutu bayrak saklayıcısının en sağdaki 8-bit'ini AH saklayıcısına yüklemektedir. SAHF komutu ise AH saklayıcısını bayrak saklayıcısının en sağdaki 8-bit'ine aktarmaktadır.

4.4.Aritmetik ve Lojik Komutlar:(Toplama, Çıkarma ve Karşılaştırma)

Bir mikroişlemcinin komut kümesindeki en temel aritmetik komutlar, toplama, çıkarma, ve karşılaştırma işlemleridir. Bu bölümde toplama komutları ADD, ADC ve INC; çıkarma komutları SUB, SBB, DEC ve karşılaştırma komutu CMP sunulacaktır.

4.4.1.Toplama(ADD, ADC ve INC Komutları):

Temel olarak ADD ve ADC olmak üzere iki toplama komutu olmasına karşın, kullanılan adresleme modlarına göre, toplama işlemi çok değişik şekiller almaktadır. İkinci elde ile toplama komutu ADC (Add with Carry) değişik uzunlukta (16-bit, 32-bit, 64-bit veya daha

uzun) operandların toplama işleminde kullanılır.INC(arttırma) komutu bir operand'a 1 toplayan diğer bir toplama komutu sayılabilir.

4.4.1.1.ADD Komutu:

Tablo.4.7, ADD toplama komutunun değişik adresleme modlarına göre bazı kullanım örneklerini göstermektedir. Tablonun başında ***saklayıcı toplamasına (register addition)*** çeşitli örnekler verilmiştir. Bu toplamada her iki operand bir saklayıcıdan gelmektedir.

Tablo .4.7. Toplama komutunun değişik kullanım örnekleri.

Asembly Dili	***Yapılan İşlem***
ADD AL,CL	AL = AL + CL
ADD DX,SI	DX = DX + SI
ADD BL,20H	BL = BL + 20H
ADD CX,4000H	CX = CX + 4000H
ADD [BX],CL	CL,BX ile adreslenen DS'deki hücre ile toplanır, sonuç yine bu hücreye yazılır.
ADD CX,[SI+2]	CX,SI+2 ile adreslenen DS'de bulunan hücre ile toplanır, sonuç CX'e yazılır.
ADD AL,BUF	AL,DS'de bulunan BUF hücresi ile toplanır,sonuç AL'ye yazılır.
ADD AL,BUF[DI]	AL,DS'de bulunan BUF ve ofset DI'nın toplamıyla adreslenen hücre ile toplanır, sonuç AL'e yazılır.
ADD [BX+DI],DL	DS'de bulunan BX ve ofset DI'nın toplamı ile adreslenen hücre, DL ile toplanır ve sonuç yine bu hafıza hücresini yazılır.

Diğer bir adresleme ***ivedi toplama (immediate addition)*** adreslemesidir. Bu adreslemede, bir operand saklayıcıdan,diğer bir operand hafızadan gelen sabit bir sayıdır. SI, DI, BP veya BX saklayıcılarıyla yapılan adreslemeyle ***hafızadan saklayıcıya toplama (memory-to-register addition)*** gerçekleştirilir. Bir taban ofset değerinin toplanmasıyla

oluşan adresle işaretlenen hücrenin bir saklayıcı ile toplanması ile ***dizi toplaması (array addition)*** gerçekleştirilir. Aşağıda verilen örnekte, bir byte dizisi olan ARRAY'in ilk 3 elemanı ARRAY[0], ARRAY[1] ve ARRAY[2] toplanmaktadır.

```
MOV  AL, 0          ; DISPLAY adresini yükle
MOV  SI, 0          ; DISPLAY adresini yükle
ADD  AL, ARRAY[SI]    ; ilk eleman
ADD  AL, ARRAY[SI+1]  ; ikinci eleman
ADD  AL, ARRAY[SI+1]  ; üçüncü eleman
```

4.4.1.2.ADC Komutu:

Elde ile toplama komutu olan ADC, operand ile beraber elde bayrağını (C) toplamada kullanır. Bu komut 8086-8088 'da 16-bit'ten ve 80386 – Pentium 'da ise 32-bit'ten daha geniş sayıları toplamada kullanılır. Tablo.4.8, ADC komutunun bazı kullanım şekillerini göstermektedir.

Tablo.4.8. ADC komutunun değişik kullanım örnekleri

Asembly Dili	***Yapılan İşlem***
ADC AL,CL	AL = AL + CL + C
ADC AX,BX	AX = AX + BX + C
ADC [BX],AL	DS'de bulunan BX ile adreslenen byte hücresi,AL ve C ile toplanır, sonuç hafızada saklanır.
ADC AX,[BP+2]	SS'deki BP+2 ile adreslenen 16-bit hafıza hücresi,AX ve C ile toplanır, sonuç AX'de saklanır.

4.4.1.3.INC Komutu:

Bu komut bir segment saklayıcısı dışında herhangi bir saklayıcıya veya bir hafıza hücresine 1 eklemektedir. Tablo.4.9, INC komutunun bazı kullanım şekillerini göstermektedir.

Tablo.4.9. Arttırma komutunun değişik kullanım örnekleri	
Asembly Dili	***Yapılan İşlem***
INC CL	CL = CL + 1
INC BP	BP = BP + 1
INC BYTE PTR[BX]	DS'de bulunan BX ile adreslenen bir byte hafıza içeriği arttırılır.
INC WORD PTR[SI]	DS'de bulunan SI ile adreslenen bir 16-bit hafıza içeriği arttırılır.

4.4.2.Çıkarma(SUB, SBB ve DEC Komutları):

Temel olarak SUB ve SBB olmak üzere iki çıkarma komutu olmasına karşın kullanılan adresleme modlarına göre çıkarma işlemi çok değişik şekiller almaktadır. İkinci ödünç ile çıkarma komutu SBB (Subtract with Borrow) değişik uzunlukta (16-bit, 32-bit, 64-bit veya daha uzun) operandların çıkarma işleminde kullanılır. DEC (azaltma) komutu bir operand'tan 1 çıkaran diğer bir çıkarma komutu sayılabilir.

4.4.2.1.SUB Komutu:

Tablo.4.10, SUB komutunun değişik adresleme modlarına göre bazı kullanım örneklerini göstermektedir.

Tablo.4.10. Çıkarma komutunun değişik kullanım örnekleri	
Asembly Dili	***Yapılan İşlem***
SUB AL,CL	AL = AL - CL
SUB DX,SI	DX = DX - SI
SUB BL,20H	BL = BL - 20H
SUB CX,4000H	CX = CX - 4000H

SUB [BX],CL	BX ile adreslenen DS'deki hücreden CL çıkarılır, sonuç yine bu hücreye yazılır.
SUB CX,[SI+2]	CX'den SI+2 ile adreslenen DS'de bulunan hücre çıkartılır, sonuç CX'e yazılır.
SUB AL,BUF	AL'den DS'de bulunan BUF hücresi çıkartılır,sonuç AL'ye yazılır.
SUB AL,BUF[DI]	AL'den ,DS'de bulunan BUF ve ofset DI'nın toplamıyla adreslenen hücre çıkartılır, sonuç AL'e yazılır.
SUB [BX+DI],DL	DS'de bulunan BX ve ofset DI'nın toplamı ile adreslenen hücreden DL çıkartılır ve sonuç yine bu hafıza hücresini yazılır.

4.4.2.2.SBB Komutu:

Ödünç ile toplama komutu olan SBB, operand ile beraber elde bayrağını (C) çıkarmada kullanır. Bu komut Bu komut 8086-8088 'da 16-bit'ten ve 80386 – Pentium 'da ise 32-bit'ten daha geniş sayıları toplamada kullanılır. Tablo.4.11, SBB komutunun bazı kullanım şekillerini göstermektedir.

Tablo.4.11. SBB komutunun değişik kullanım örnekleri

Asembly Dili	***Yapılan İşlem***
SBB AL,CL	AL = AL - CL - C
SBB AX,BX	AX = AX - BX - C
SBB [BX],AL	DS'de bulunan BX ile adreslenen byte hücresinden AL ve C çıkartılır, sonuç hafızada saklanır.
SBB AX,[BP+2]	AX'ten SS'deki BP+2 ile adreslenen 16-bit hafıza hücre ve C çıkartılır, sonuç AX'te saklanır.

4.4.2.3.DEC Komutu:

Bu komut bir segment saklayıcısı dışında herhangi bir saklayıcıdan veya bir hafıza hücresinden 1 çıkarmaktadır.Tablo.4.12,DEC komutunun bazı kullanım şekillerini göstermektedir.

Tablo.4.12. Azaltma komutunun değişik kullanım örnekleri

Asembly Dili	***Yapılan İşlem***
DEC AL	AL = AL - 1
DEC SP	SP = SP - 1
DEC BYTE PTR[BX]	DS'de bulunan BX ile adreslenen bir byte hafıza içeriği azaltılır.
DEC WORD PTR[SI]	DS'de bulunan SI ile adreslenen bir 16-bit hafıza içeriği azaltılır.

4.4.3.Karşılaştırma:

Karşılaştırma komutu olan CMP (Compare) sadece bayrak bit'lerini değiştiren bir çıkarma işlemidir. Karşılaştırma işlemi, bir saklayıcı içeriğiyle diğer bir saklayıcı veya hafıza içeriği arasında yapılır. Genelde, bu karşılaştırma komutundan hemen sonra bayrakların durumunu test eden bir dallanma komutu gelir.

Tablo.4.13, CMP komutunun değişik adresleme modlarına göre bazı kullanım örneklerini göstermektedir. Karşılaştırma komutu daha önce verilen toplama ve çıkarma komutunda olduğu gibi aynı adresleme modlarını kullanmaktadır. Sadece hafızadan-hafızaya ve segment saklayıcı karşılaştırmasına izin verilmez. Karşılaştırma işlemine giren operand'larda bir değişme olmaz. Sadece bayraklar etkilenir.

Tablo .4.13. Karşılaştırma komutunun değişik kullanım örnekleri	
Asembly Dili	***Yapılan İşlem***
SUB AL,CL	AL-CL işlemi yapılır.
SUB DX,SI	DX-SI işlemi yapılır.
SUB BL,20H	BL-20H işlemi yapılır.
SUB CX,4000H	CX-4000H işlemi yapılır.
SUB [BX],CL	BX ile adreslenen DS'deki hücreden CL çıkartılır.
SUB CX,[SI+2]	CX'ten SI+2 ile adreslenen DS'te bulunan hücre çıkartılır.
SUB AL,BUF	AL'den DS'de bulunan BUF hücresi çıkartılır.
SUB AL,BUF[DI]	Al'den DS'de bulunan BUF ve ofset DI'nın toplamı ile adreslenen hücre çıkartılır.
SUB [BX+DI],DL	DS'de bulunan BX ve ofset DI'nın toplamı ile adreslenen hücreden DL çıkartılır.

4.5.Çarpma ve Bölme:

Eski 8-bit mikroişlemciler donanım çarpma-bölme birimine sahip değildi. Bu birim 16-bit mikroişlemcilere eklenen ilk önemli donanım birimi oldu. Bu temel birim, bununla beraber, programlara sadece ***tamsayı (integer)*** çarpma ve bölme desteği sağlamaktadır. Kayan nokta işlemleri ya yazılım alt programları ile veya ilk zamanlar, 8087 (8086/8088 için), 80287 (80286 için) ve 80387 (80386 için) gibi bir dış nümerik işlemciyle sağlandı. 80486 ve daha sonraki Pentium işlemcilerde, kayan nokta aritmetik işlemleri, artık tüm devre üzerinde gerçekleştirildi.

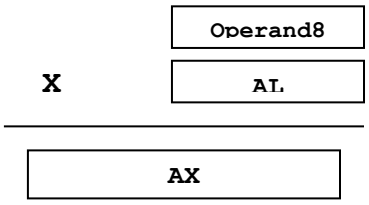
4.5.1.Çarpma:

Çarpma işlemi, 8-bit veya 16-bit işaretsiz (MUL) veya işaretli (IMUL) tamsayılar üzerinde yapılır. Çarpma sonucu (çarpım), operand'lar 8-bit ise, 16-bit; 16-bit ise 32-bit olur. Çarpma komutlarında, ivedi adresleme modunun dışındaki adresleme modları

kullanılabilir. Çarpma ve bölme komutlarında ilk operand her zaman AL (8-bit işlemde) veya Ax (16-bit işlemde) saklayıcısında bulunur.

a) 8-Bit Çarpma:

8-bit çarpma işleminde çarpanlardan biri her zaman AL saklayıcısında bulunur. Programcı ikinci çarpanı komut ile belirtir. Şekil.4.1'de görüldüğü gibi 16-bit çarpım sonucu Ax saklayıcısında bulunur.

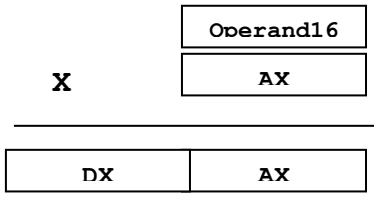


Şekil.4.1. 8-bit çarpma işlemi:

İşaretli çarpmada operand'lar donanım tarafından işaretli (2'nin tümleyeni) olarak yorumlanır ve çarpım yine 2'nin tümleyeni şeklinde olur. Bu tür çarpma işleminde IMUL komutu kullanılır. Sayıların yorumlanması ve işlemleri programcıya bağlıdır.

b) 16-Bit Çarpma:

16-bit çarpma işleminde çarpanlardan biri her zaman AZ saklayıcısında bulunur. Programcı ikinci çarpanı komut ile belirtir. Şekil.4.2' de görüldüğü gibi 32-bit çarpım sonucu DX ve AX saklayıcılarında bulunur. Tablo.4.14, MUL ve IMUL komutlarının değişik adresleme modlarına göre bazı kullanım örnekleri gösterilmektedir.



Şekil.4.2. 16-bit çarpma işlemi

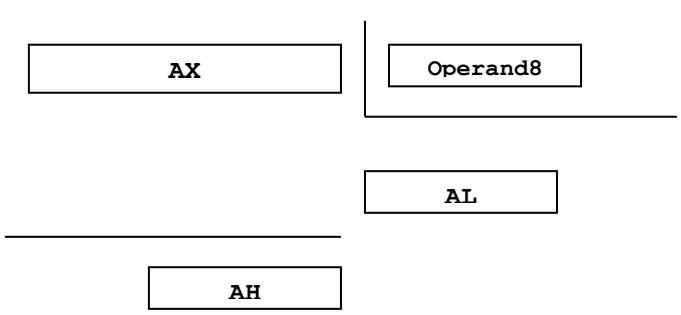
Tablo.4.14. Çarpma komutunun değişik kullanım örnekleri	
Asembly Dili	***Yapılan İşlem***
MUL DL	AL'de bulunan işaretsiz tamsayı DL ile çarpılır. Çarpım AX'te bulunur.
IMUL CH	AL'de bulunan işaretli tamsayı CH ile çarpılır. Çarpım AX'te bulunur.
MUL BYTE PTR[BX]	DS'de bulunan BX ile adreslenen hafıza hücresi AL ile çarpılır. Çarpım AX'te bulunur.
IMUL BYTE PTR OP2	DS'de OP2 adresinde bulunan işaretli tamsayı AL ile çarpılır. Çarpım AX'te bulunur.
IMUL CX	AX'deki işaretli tamsayı CX ile çarpılır. Çarpım DX:AX'te bulunur.
MUL WORD PTR[SI]	DS'de bulunan ve SI ile adreslenen 16-bit hafıza hücresi AX ile çarpılır. Çarpım DX:AX'te bulunur.

4.5.2.Bölme:

8086/8088'de çarpma işlemi gibi bölme işlemi de, işaretsiz (DIV) veya işaretli (IDIV) 8-bit veya 16-bit sayılar üzerinde yapılır.

a) 8-Bit Bölme:

8-bit bölme işleminde bölünen her zaman AX saklayıcısında bulunur. Programcı bölen sayıyı komut ile belirtir. Şekil.4.3'te görüldüğü gibi 8-bit bölüm sonucu AL ve AH saklayıcılarında bulunur.



Şekil.4.3. 8-bit bölme işlemi

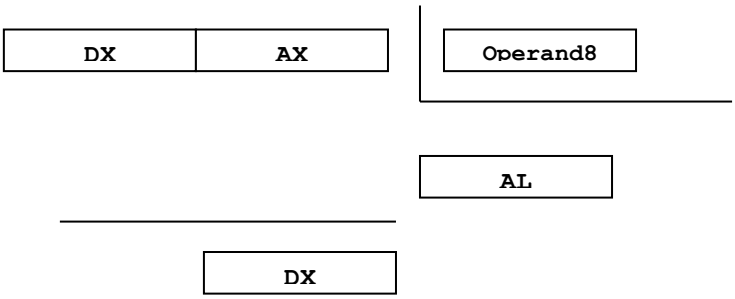
Aşağıda verilen bölme işlemi örneğinde, AL saklayıcısındaki 12h sayısı CL saklayıcısındaki 2 sayısına bölünmektedir. Bölme işleminden önce kalan saklandığı AH saklayıcısı temizlenmektedir.

```
MOV  AL, 12H
MOV  CL, 2
MOV  AH, 0
DIV  CL
```

8-bit bölme işleminde, bölünecek sayı 8-bit ise bu sayıyı 16-bit'e çevirmek için CBW (Convert Byte to Word) komutu kullanılır. Bu komut Al saklayıcısında bulunan işaretli 8-bit sayıyı AX saklayıcısına 16-bit olarak çevirir. Bu komut genellikle 8-bit bölme işleminden önce kullanılır.

b)16-Bit Bölme:

16-bit bölme işleminde bölünen her zaman DX:AX saklayıcı çiftinde bulunur. Programcı bölen sayıyı komut ile belirtir. Şekil.4.4'de görüldüğü gibi, 16-bit bölüm sonucu AX ve DX saklayıcılarında bulunur.



Şekil.4.4. 16-bit bölme işlemi:

16-bit bölme işleminde, bölünecek sayı 16-bit ise bu sayıyı 32-bit'e çevirmek için CWD (Convert Word to Double Word) komutu kullanılır. Bu komut AX saklayıcısına bulunan işaretli 16-bit sayıyı DX:AX saklayıcı çiftine 32-bit olarak çevirir. Bu komut genellikle 16-bit bölme işleminden önce kullanılır.

Aşağıda verilen örnekte, AX'teki –200 CX'te bulunan 9 ile bölünmektedir. Bölme işlemindeki önce CWD komutu ile AX'teki işaretli sayı DX:AX saklayıcı çiftine 32-bit olarak çevrilir. Bölme işleminden sonra AX'te bölüm (-22) ve DX'te kalan (-2) bulunur. Tablo.4.15''de 8-bit ve 16-bit bölme işlemlerine ait bazı örnekler verilmektedir.

MOV AX, -200

MOV CX, 9

CWD

IDIV CX

Tablo.4.15. Bölme komutunun değişik kullanım örnekleri

Asembly Dili	***Yapılan İşlem***
DIV DL	AL'de bulunan işaretsiz tamsayı DL ile bölünür. Çarpım AX'te bulunur.
IDIV CH	AL'de bulunan işaretli tamsayı CH ile bölünür. Çarpım AX'te bulunur.
DIV BYTE PTR[BX]	DS'de bulunan BX ile adreslenen hafıza hücresi AL ile bölünür. Çarpım AX'te bulunur.
IDIV BYTE PTR OP2	DS'de OP2 adresinde bulunan işaretli tamsayı AL ile bölünür. Çarpım AX'te bulunur.
IDIV CX	AX'deki işaretli tamsayı CX ile bölünür. Çarpım DX:AX'te bulunur.
DIV WORD PTR[SI]	DS'de bulunan ve SI ile adreslenen 16-bit hafıza hücresi AX ile bölünür. Çarpım DX:AX'te bulunur.

4.6.BCD ve ASCII Aritmetik:

8086/8088 ikili kodlanmış ondalık ***(Binary Coded Decimal - BCD)*** ve ASCII sayılar üzerinde işlem yapan özel konutlara sahiptir. Bu komutlar, bu tür kodları kullanan nümerik veri göstergelerini sürmede veya benzeri diğer veri işlemlerinde faydalıdır.

4.6.1.BCD Aritmetik:

BCD aritmetikte kullanılan 4 komut vardır. Bu komutlar şunlardır:

DAA (Decimal Adjust after Addtion)

DAS (Decimal Adjust after Subtraction)

AAM (Adjust After Mutiplication)

AAD (Adjust Before Division)

DAA, DAS, AAM komutları BCD sayıların toplama, çıkarma, çarpma işlemlerinden sonra; ADD ise DIV komutu kullanmadan *önce* sayıları ayarlama için kullanılır.

Toplama ve çıkarmada, toplanacak ve çıkartılacak sayılar, bir byte içinde, 2 hane olarak AL'de saklanır. Bu işleme *paketlenmiş BCD formatı* denir. Örneğin, eğer bir byte 25H ise ve bu bir BCD sayıyı belirtiyorsa, bu sayı BCD kodlu 25 sayıdır. Yani bu byte 2 ve 5 BCD hanelerini işlemektedir. Çarpma ve bölmede, sayılar 1 byte'ta 1 hane olarak paketlenmiş BCD sayıları olarak saklanır. Örneğin, AL 04 ise, sayı BCD 4 'e karşılık gelir.

4.6.1.1.DAA:

DAA komutunun kullanımına ait aşağıda verilen örnek, BX ve CX saklayıcılarında bulunan 4 haneli BCD sayıları toplamakta, sonucu DX saklayıcısına saklamaktadır. İlk DAA komutunun kullanımından önce, 99 ve 34 sayıları toplanarak BCD olmayan CD sayısı üretilmiştir. DAA komutu bu işlemi C bayrağında 1 ve AL'de 33 yaparak düzeltmektedir. Daha sonra 12 ve 30 elde bayrağıyla beraber toplanmaktadır. Bu sayının haneleri BCD olduğundan, ikinci DAA komutundan sonra bu sayıda bir değişiklik olmaz.sonuç olarak, DX saklayıcısında 4333H veya 4333 BCD kodu bulunur.

```
MOV  CX, 1234H
MOV  CX, 3099H
MOV  AL, BL
DIV  AL, CL
DAA
MOV  DL, AL
MOV  AL, BH
ADC  AL, DH
DAA
MOV  DH, AL
```

Yukarıdaki toplama işlemi DAA komutu kullanılmadan yapılsaydı çok daya uzun olacaktı. DAA komutu, otomatik olarak AL'de bulunan sayıları düzelttiği için programcıya bu işlemi kontrol yükü düşmez.

4.6.1.2.DAS:

DAS komutu DAA gibi çalışmaktadır. Bu komut çıkarma işleminden sonra AL'yi ayarlamada kullanılır. Aşağıdaki program DAS komutu kullanılarak, iki BCD sayının çıkartma işlemini gerçekleştirmektedir. Bu program yukarıda verilen programa çok benzemektedir. ADD ve ADC komutları SUB ve SBB ile DAA komutu DAS ile yer değiştirilmiştir. Bu program çalıştırıldığında DX saklayıcısında 1865 BCD sayısı bulunur.

```
MOV  CX, 1234H
MOV  CX, 3099H
MOV  AL, BL
DIV  AL, CL
DAS
MOV  DL, AL
MOV  AL, BH
ADC  AL, DH
DAS
MOV  DH, AL
```

4.6.1.3.AAM:

Bu komut iki tane tek haneli paketlenmiş BCD sayının çarpma işleminden sonra kullanılır. Örneğin, AL saklayıcısında bulunan 5 ile CL'deki 6 sayılarını çarpan aşağıdaki örneği düşünelim. Çarpma işleminden sonra AX saklayıcısında 1EH (30) bulunur. Bu BCD olarak doğru bir gösterim değildir. AAM komutundan sonra AX saklayıcısı 0300H veya 2 haneli paketlenmiş 30 BCD sonucuna çevrilir.

```
MOV  AL, 5
DIV  CL, 6
MUL
AAM
```

4.6.1.4.AAD:

Bir aritmetik işlemden ***sonra*** kullanılan diğer 3 BCD komutundan farklı olarak, bu komut bölme işleminden ***önce*** kullanılmaktadır. AAD komutu, AX saklayıcısında 2-haneli paketlenmiş bir BCD sayı gerektirmekte olup AH en değerli haneyi, AL ise diğer haneyi tutar. Bu 2-haneli BCD sayı, tek-haneli bir BCD sayı ile bölünmeden önce AAD komutuyla ayarlanmalıdır. Bu sayede yapılan bölme işlemi sonucunda, AL saklayıcısında tek-haneli sonuç ve AH'te kalan üretir. AAD komutu paketlenmiş BCD sayıları 00 ile 99 arasında ikili sayılara çevirir. Aşağıda verilen örnek paketlenmiş BCD 72 sayısının 9 ile bölünme işlemini gerçekleştirmektedir. AAD komutuyla AX saklayıcısına bulunan 0702H sayısı, yine AX'te 0048H sayısına çevrilir. Bu şekilde BCD sayı ikili sayıya çevrilir. Böylece, ikili bölme komutu DIV ile doğru bölme işlemi yapılarak, AL'de 08H ve AH'ta 00H bulunur.

```
MOV  BL, 9H
DIV  AX, 0702H
AAD
DIV  BL
```

4.6.2.ASCII Aritmetik:

ASCII aritmetik komutları, ASCII-kodlu sayılarla kullanılır. Bu sayılar 30H ile 39H arasında değişir ve 0 ile 9 arasında sayıları gösterir. ASCII-kodlu sayılarla kullanılan iki komut vardır. Bu komutlar şunlardır.

AAA (Adjust for ASCII Addition)

AAS (Adjust for ASCII Substraction)

Bu komutlar her zaman AX saklayıcısını ayarlama işleminden önce kaynak ve sonra hedef olarak kullanılır.

4.6.2.1. AAA:

İki 1-haneli ASCII-kodlu sayının toplanması faydalı bir veri üretmez. Örneğin, eğer 31H ile 39H sayılarını toplarsak, sonuç 6AH olur. bu ASCII toplama (1+9) işlemi, 10 ondalık sayısına karşılık gelen, 31H ve 30H ASCII-kodlu 2-haneli bir ASCII sonucu üretmeli. Eğer bu toplamadan sonra, AAA komutu yürütülürse, AX saklayıcısı 0100H içerir. Bu sonuç ASCII olmamakla beraber, 3030H eklenerek, ASCII koda çevrilir ve 3130H elde edilir. Aşağıda verilen kod parçası yukarıda anlatılan işlemi gerçekleştirmektedir.

```
MOV  AX, 031H      ; ASCII 1 yükle, AH2i temizle
DIV  AL, 39H       ; ASCII 9 ile topla
AAA                ; ASCII toplama sonu ilk ayarlama
ADD  AX, 3030H     ; Sonucu ASCII olarak ver
```

4.7.Lojik İşlemler:

Lojik işlemler, düşük-seviyeli bir programda, ikili bit kontrolü sağlar. Lojik işlemler bit'lerin temizlenmesini (0'lanmasını), 1'lenesini veya tersinin alınmasını sağlar. Bütün lojik işlemler bayrak bit'lerini etkiler. Lojik işlemler her zaman elde ve taşma bit'lerini temizleyip diğer bayraklar sonucun durumunu yansıtır.

Temel lojik komutları AND, OR, XOR (Exclusive-OR) ve NOT işlemleridir. Diğer bir lojik komut olan TEST, AND komutunun özel bir şeklidir. NOT komutuna benzeyen NEG de bu bölümde anlatılacaktır.

4.7.1.AND, OR ve XOR Komutları:

AND işlemi lojik çarpma işlemini gerçekleştirir. AND işlemi, bir ikili sayıdaki bit'leri temizleme işleminde kullanılabilir. Bir ikili sayıdaki bit'i temizleme (0'lama) işlemine ***maskeleme (masking)*** denir. Bu işlem bir veri kelimesinde istenmeyen bit'ler temizlenir.

Maskeleme işlemine bir örnek olarak bir 16-bit ASCII bilginin BCD koda çevrilmesini düşünelim. Aşağıdaki örnekte olduğu gibi, eğer ASCII koddaki her bir veri hanesinin en sol 4 bit'i temizlenirse (maskelenirse) BCD kod elde edilir.

```
MOV  CX, 3531H    ; 2 haneli ASCII sayıyı yükle
AND  CX, 0F0FH    ; CX'i maskele
```

OR işlemi lojik toplama işlemini gerçekleştirir. OR işlemi ikili sayıdaki istenilen bit'leri 1'leme işleminde kullanılabilir.

Aşağıda verilen örnekte, iki sayı çarpıldıktan sonra AAM komutu kullanılarak 2-haneli paketlenmiş BCD sayıya çevrilmektedir. En son satırda yapılan OR işlemiyle, bu sayı 2-haneli ASCII-kodlu sayıya çevrilir. OR komutundan önce AX saklayıcısı 0305H içermektedir. OR komutundan sonra AX 3335H içerir.

```
MOV  AL, 5       ; Veriyi yükle
AND  BL, 7
MUL  BL          ; AX = 5 * 7
AAM              ; Paketlenmiş 2-haneli BCD ayar
OR   AX, 3030    ; ASCII'ye çevir
```

XOR işlemi, bazen karşılaştırma işlemi olarak adlandırılır.XOR işleminde girişlerin farklı olma durumunda; örneğin, γ = A XOR B işleminde, A = 1, B = 0, veya, A = 0, B = 1 ise, sonuç γ lojik 1 olur. Diğer durumlarda γ lojik 0'dır.

XOR işlemi bir saklayıcının veya bir hafıza hücresinin bit'lerini ***tersleme (invert)*** işleminde kullanılabilir. Bu kelimedeki tersini alınacak bit'lerin karşılığı 1 ile XOR işlemine girer ise, istenen bit'ler terslenir. Bu işlem, bir kontrol uygulamasında açma/kapama bit'lerinin konumlarını değiştirmede yararlıdır. Örneğin, AX saklayıcısında bulunan bit'lerin, en sağdaki 6 bit'i değiştirilmeden en-solundaki 10 bit'inin tersi alınmak istensin. Aşağıdaki komut bu işlemi gerçekleştirmektedir.

```
XOR  AX, 0FFC0H
```

Bir saklayıcının veya bir hafıza hücresinin bit'leri, kendisi ile XOR işlemine girer ise o kelime temizlenir. Bu işlem en hızlı kelime temizleme işlemidir. Örneğin, aşağıda verilen kod örneğinde AL saklayıcısı temizlenmektedir.

```
XOR  AL, AL
```

AL saklayıcısını temizleyen MOV komutu ile verilen aşağıdaki örneği hafızadan işlem kodundan sonra, 0 verisini okuma için bir hafıza okuma çevrimi daha gerçektir. Bu yüzden, sadece bir XOR işlem kodundan okumasıyla AL'yi temizleyen yukarıdaki koda göre yürütme süresi daha uzun.

```
MOV  AL, 0
```

Özet olarak, AND komutu bit'leri temizlemekte (0'lamakta), OR komutu 1'lemekte ve XOR komutu ise bit'lerin tersini almaktadır. Bu üç komut bir programcıya, herhangi bir saklayıcıda veya hafıza hücresinde bulunan bit bit üzerinde kontrol sağlar.

Tablo.4.16, AND, OR ve XOR komutlarının değişik adresleme modlarına göre bazı kullanım örnekleri gösterilmektedir.

Tablo.4.16. AND, OR ve XOR örnekleri (OP[Operation]: AND, OR ya da XOR)	
Asembly Dili	***Yapılan İşlem***
OP AL,CL	AL = AL OP CL
OP CX,DX	CX = CX OP DX
OP BL,0FH	BL = BL OP 0FH
OP SI,00FFH	SI = SI OP 00FFH
OP AX,[DI]	AX, data segment hafızada DI ile işaret edilen 16-bir kelime ile OP'lanır, sonuç AX'te saklanır.
OP BUF[SI],AL	Data segment hafızada BUF artı SI ile işaret edilen byte içerik OP'lanır, sonuç hafızada saklanır.

4.7.2.TEST Komutu:

TEST komutu bir AND işlemi gerçekleştirir. Daha önce gördüğümüz, AND komutu, hedef operand'ı değiştirmektedir, fakat TEST komutu değiştirmez. Bu komut, testin sonucunu belirten, sadece bayrak saklayıcısının durumunu etkiler. TEST komutu AND komutu gibi aynı adresleme modlarını kullanır. Tablo.4.17, TEST komutunun bazı şekillerini göstermektedir.

Tablo.4.17. TEST komutunun bazı kullanım örnekleri	
Asembly Dili	***Yapılan İşlem***
TEST AL,CL	AL ile CL AND'lenir. Ne AL ne de CL değişir. Sadece bayraklar değişir.
TEST CX,DX	CX ile DX AND'lenir. Ne CX ne de DX değişir. Sadece bayraklar değişir.
TEST BL,5	AL ile 5 AND'lenir. AL değişmez. Sadece bayraklar değişir.
TEST SI,00FFH	DX ile 00FH AND'lenir. DX değişmez. Sadece bayraklar değişir.

TEST komutu bir karşılaştırma CMP (Compare) komutu gibi kullanılır. Bu komut normalde tek bir bit'i test eder; buna karşın, CMP komutu bütün bir byte'ı veya kelimeyi test eder. Sıfır bayrağı (Z), eğer test edilen bir sıfır ise, lojik 1'dir (Z=1, sonuç sıfır); eğer test edilen bit sıfır değil ise, lojik 0'dır (Z=0, sonuç sıfır değil).

Genelde TEST komutundan sonra JZ (Jump Zero) veya JNZ (Jump Not Zero) gibi bir dallanma komutu kullanır. Aşağıda verilen kısa program parçasında, AL saklayıcısında bulunan en-sağdaki ve en-soldaki bit durumları test edilmektedir. 1 ile en-sağdaki 128 ile en-soldaki bit pozisyonu seçilir. Her testi bir dallanma komutu takip eder. Eğer test edilen bit 1 değil ise, dallanma komutuyla belirtilen adrese (RIGHT veya LEFT) program akışı yönlenir.

```
TEST  AL, 1     ; en-sağdaki bit'i test et
JNZ   RIGHT     ; eğer 1 ise dallan
   TEST  AL, 128   ; en-soldaki bit'i test et
JNZ   LEFT      ; eğer 1 ise dallan
```

4.7.3.NOT ve NEG Komutları:

Lojik tersini alma işlemi (veya 1'in tümleyeni) NOT komutuyla ve aritmetik işaret tersleme işlemi (veya'nın tümleyeni) NEG komutuyla yapılır. Bu komutlar, sadece tek operand kullanan çok az komuttan ikisidir. Tablo.4.18, NOT ve NEG komutlarının değişik adresleme modlarına göre bazı kullanım örneklerini göstermektedir.

NOT komutu, bir byte'taki veya kelimedeki bit'lerin tersini alır. NEG komutu ise, bir sayının 2'nin tümleyeni eşini bulur. Yani bir işaretli sayının aritmetik işareti pozitiften negatife veya negatiften pozitife değişir. NOT fonksiyonu, bir lojik işlem olarak, NEG fonksiyonu, bir aritmetik işlem olarak düşünülür.

Tablo.4.18. NOT ve NEG komutlarının bazı kullanım örnekleri

Asembly Dili	***Yapılan İşlem***
NOT CL	CL = CL'
NEG CL	CL = CL' + 1 (2'nin tümleyeni)
NEG AX	AX = AX' + 1 (2'nin tümleyeni)

NOT TEMP	DS'de bulunan TEMP ile adreslenen verinin bit'leri terslenir. TEMP'le adreslenen verinin boyu TEMP'in tanımlanmasına bağlıdır.
NOT BYTE PTR[BX]	DS'de bulunan BX ile adreslenen 1-byte verinin bit'leri terslenir.

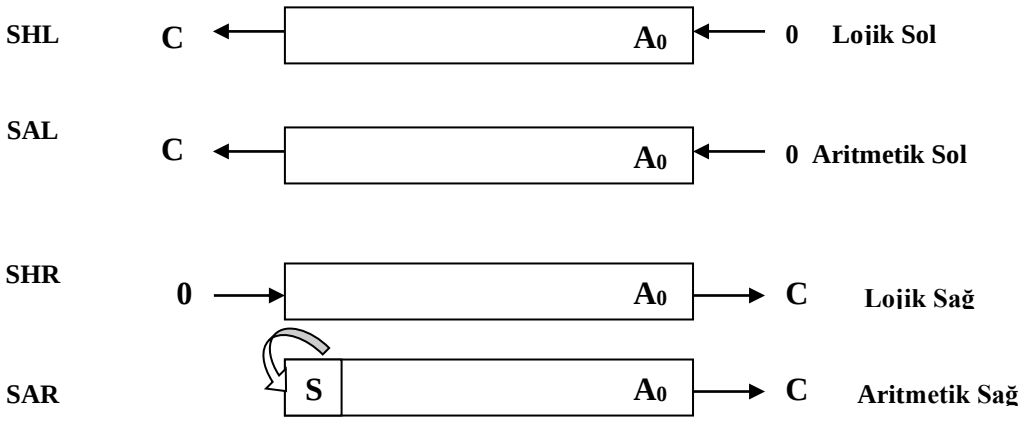
4.7.4.Kaydırma ve Döndürme:

Kaydırma (***Shift***) ve döndürme (***Rotate***) komutları, AND, OR, XOR ve NOT komutlarında olduğu gibi, ikili sayılar üzerinde, ikili bit seviyesinde işlem yapmaktadır.

4.7.4.1.Kaydırma Komutları:

Kaydırma komutları, bir saklayıcı veya hafıza hücresi üzerinde işlem yapar. Bu komutlar, 2'nin katları ile çarpma (sola kaydırma) ve bölme (sağa kaydırma) işlemlerinde kullanılır.8086/8088 komut kümesi, Şekil.4.5'te görüldüğü gibi, 2 tane lojik kaydırma ve 2 tane aritmetik kaydırma olmak üzere, toplam 4 tane kaydırma komutuna sahiptir.

Aritmetik sağa kaydırma komutu olan SAR'da, eğer sayı pozitif ise, sağa '0' kaydırılır; eğer sayı negatif ise, sağa '1' kaydırır. Bu sağa kaydırma işleminde işaret bit'i yine yine aynı konuma kopyalanacaktır.Aritmetik sağa kaydırma ile lojik sağa kaydırma, işaretli sayılarda farklılık gösterir. SAR komutu işaretli sayıyı 2'ye böler. Buna karşın SHR komutu işaretsiz sayıyı 2'ye böler. Lojik kaydırma ise sayıyı 2 ile çarpar. Eğer 1 haneden fazla kaydırma istenirse, CL saklayıcısı kaydırma sayısını tutar.



Şekil.4.5. Kaydırma komutları

Tablo.4.19, kaydırma komutlarının değişik adresleme modlarına göre bazı kullanım örneklerini göstermektedir.

Tablo.4.19. Kaydırma komutlarının bazı kullanım örnekleri

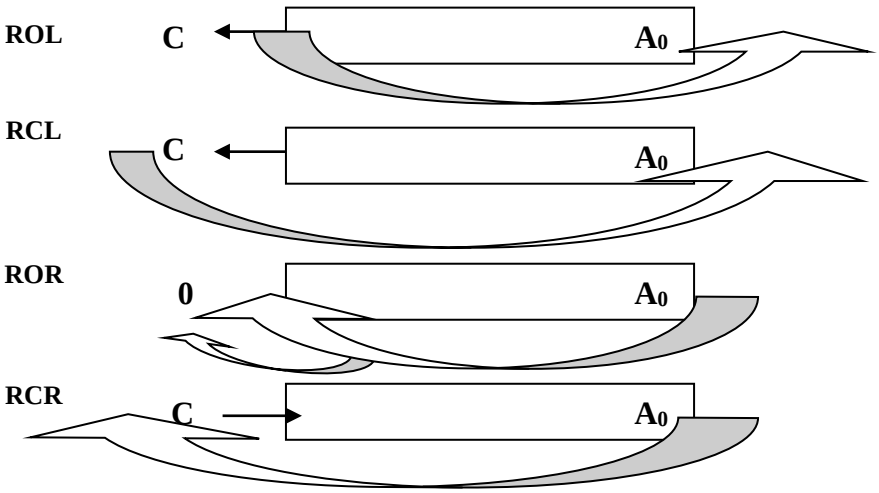
Asembly Dili	***Yapılan İşlem***
SHL CX	Lojik olarak CX'i sola kaydır
SHR AX	Lojik olarak AX'i sola kaydır
SAL BUFFER,CL	Aritmetik olarak DS'de bulunan BUFFER'ı sola CL'de bulunan sayı kadar kaydır
SAR SI	Aritmetik olarak SI'yı sağa kaydır

4.7.4.2.Döndürme Komutları:

8086/8088 mikroişlemcisinde yer alan 4 döndürme komutu, bir saklayıcıdaki veya hafıza hücresindeki bilgiyi, bir ucundan diğerine veya elde bayrağı üzerinden, Şekil.4.6'da görüldüğü gibi döndürmektedir. Eğer 1 haneden fazla bir döndürme işlemi istenirse, CL saklayıcısı kaydırma sayısını tutar. Tablo.4.20, döndürme komutlarının değişik adresleme modlarına göre bazı kullanım örneklerini göstermektedir.

Tablo.4.20. Döndürme komutlarının bazı kullanım örnekleri

Asembly Dili	***Yapılan İşlem***
ROL DI	DI sola bir bit döner.
RCL CL	CL sola elde üzerinden 1 bit döner.
ROR AH,CL	AH sola elde üzerinden CL'de bulunan sayı kadar döner.
RCR WORD PTR[BP]	SS'de bulunan ve BP ile adreslenen 16-bit Word sağa 1 bit döner.



Şekil.4.6. Döndürme komutları

4.8.Dizi Karşılaştırmaları:

Daha önceden gördüğümüz gibi, 8086/8088'de yer alan dizi komutları çok güzlüdür.; çünkü, çok az komut ile programcıya, bir veri bloğunu, hafızanın bir bölümünden diğerine aktarmayı sağlar.Bu bölümde, bir hafıza bloğunu, bir veri ile karşılaştırma işlemini yapan SCAS (Dizi Tarama) komutu ile, iki hafıza bloğunu karşılaştıran CMPS (Dizi Karşılaştırma) ek dizi komutları sunulacaktır.

Daha önce sunulan, MOVS, LODS, ve STOS dizi komutları gibi, SCAS ve CMPS komutları SI ve/veya DI için otomatik arttırma veya otomatik-azaltma işlemlerini seçmede yön bayrağını (D) kullanır. Ayrıca, REPNE, REPE gibi duruma bağlı tekrar öneki ile tekrar edilebilir.

4.8.1.SCAS Komutu:

Dizi tarama (Scan String) komutu, AL saklayıcısı ile bir byte hafıza bloğunu veya AX saklayıcısı ile bir 16-bit hafıza bloğunu karşılaştırır. Byte karşılaştırma işleminde SCASB, kelime karşılaştırma işleminde SCASW kullanılır.

Aşağıdaki verilen dizi tarama örneğinde, BLOCK adresli yerden başlayan 100 byte veri içinde 7 sayısı aranmaktadır. SCASB komutu blok içinde 7 bulana kadar veya CX sayacı sıfır oluncaya kadar devam eder.

```
MOV  DI, OFFSET BLOCK   ; veri bloğuna işaret et
CLD                     ; otomatik arttırma
MOV  CX, 100            ; blok uzunluğu
```

```
    MOV  AL, 7          ; blok içinde aranacak veri
REPNE  SCASB            ; bulunana kadar karşılaştır
```

Diğer durumda bağlı tekrar ön eki, ***eşit olduğu sürece tekrar (repeat while equal)*** REPE ekidir. Duruma bağlı tekrarlarda, dizi işlemi, durum doğru oldukça ve CX sayacı sıfır olmadığı sürece tekrar eder. Sayaç sıfır olduğunda veya durum artık doğru değil ise tekrardan çıkılır. Her bir çevrim, sayacı bir azaltır. Sayacın azalması bayrakları etkilemez, fakat bir hafıza içeriği ile AL veya AX saklayıcısını karşılaştıran SCAS komutu bayrakları etkiler.

Yukarıda verilen örnekte 7 rakamın blok içinde bulunup bulunmadığını anlamak için, SCASB komutundan sonra Z bayrağına bakmak yeterlidir. Eğer Z = 1 ise, 7, blok içindedir ve DI,7 sayının bulunduğu hafıza hücresinden bir sonraki yere işaret eder.

4.8.2.CMPS Komutu:

Dizi karşılaştırma (***Compare String***) komutu, aynı olup olmadığını bulmak için iki hafıza bloğunu karşılaştırır. Byte Karşılaştırma işleminde CMPB, keline karşılaştırma işleminde CMPW kullanılır. Karşılaştırma işleminde, ES alanında DI ile işaretli bir hafıza hücresi, DS alanındaki SI ile adreslenen bir hücresinden çıkartılır. Bu çıkarma işleminden ne hafıza, ne saklayıcılar ne de bayraklar etkilenir. CMPS komutu, otomatik olarak SI ve DI saklayıcılarını arttırır veya azaltır. Bu komut normalde REPE veya REPNE ön eki ile beraber kullanılır.

Aşağıda verilen örnek, 50 byte uzunluğunda iki hafıza bloklarının birbirlerine eşitlik olup olmadığını bulmak için, blokları karşılaştırmaktadır. CMPB komutu REPE ön eki ile kullanılmaktadır. Bu ek eşitlik durumu olduğu sürece araştırmanın devam etmesini sağlar Cx saklayıcısı 0 olduğunda veya eşit olmayan bir durumda, CMPB komutu yürütmesini durdurur. Eğer CX sıfır ise veya bayraklar eşit durum belirtir. İse, iki hafıza dizisi birbirine eşittir.

```
    MOV  SI, OFFSET BLOCK1   ; veri bloğuna işaret et
    MOV  DI, OFFSET BLOCK2   ; veri bloğuna işaret et
    CLD                      ; otomatik arttırma
    MOV  CX, 50              ; blok uzunluğu
REPE   CMPSB                 ; dizileri karşılaştır
```

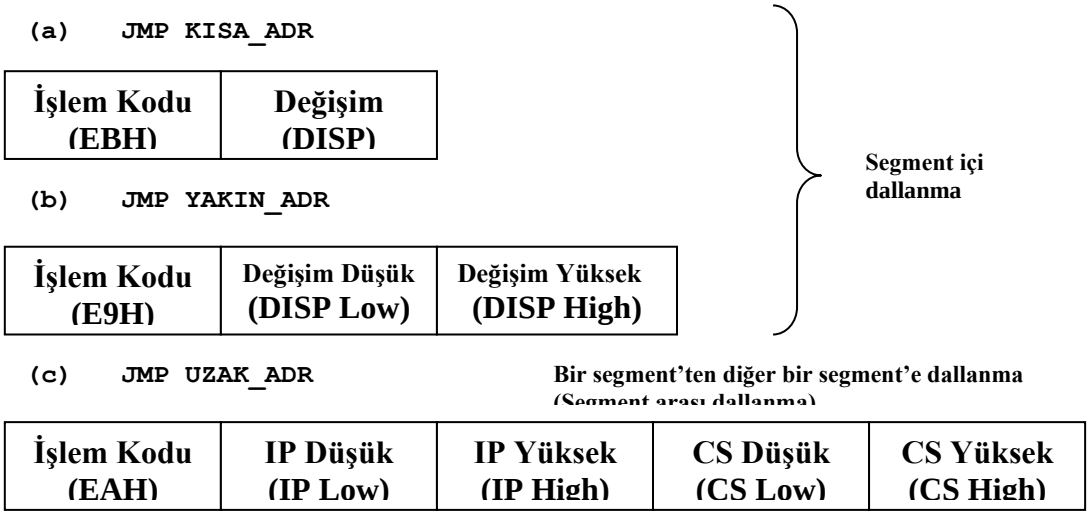
4.9.Program Kontrol Komutları:

4.9.1.Dallanma Komutları:

Temel program kontrol komutu olan JMP (Jump), hafızanın bir bölümünden diğer bir bölümüne program akışını yönlendirir. Duruma bağlı dallanma komutları, bayrak bit'lerinin durumuna göre, yani bir durum testi sonucu program akışını değiştirir.

4.9.1.1.Doğrudan Dallanmalar:

JMP komutunun 3 değişik şekli vardır. Bunlar: ***Kısa (short)*** dallanma, ***yakın (near)*** dallanma ve ***uzak (far)*** dallanmadır. Şekil.4.7, dallanma komutlarını, hafızadaki yer alış yapılarını ve uzunluklarını göstermektedir.



Şekil.4.7.Dallanma komutları: (a) kısa dallanma (2 byte), (b) yakın dallanma (3 byte) ve (c) uzak dallanma (5byte)

4.9.1.2.Kısa Dallanma:

Kısa dallanma, 2-byte bir komuttur ve işlem kodundan sonraki ikinci byte, işaretli adres bilgisidir. Bu komut, kendisini takip eden hafıza adresinden itibaren, +127 ile –128 byte arasında bir hafıza hücresinde dallanmayı sağlar. Kısa dallanmalar, göreceli (relative) dallanmalar olarak adlandırılır. Çünkü, bu komutlar hafızada herhangi bir yere, bir değişiklik yapılmadan taşınabilir. Bu dallanma komutun içinde, işlem kodundan sonra, bir ***adres*** olmayıp ***yer değişim (displacement)*** bulunur.

4.9.1.3.Yakın Dallanma:

Kısa dallanmaya benzeyen yakın dallanma 3-byte bir komuttur ve o anki CS içinde herhangi bir adrese dallanmayı sağlar. Ofset adresi 16-bit'tir.

4.9.1.4.Uzak Dallanma:

Uzak dallanma, mikroişlemcinin hafıza adres alanındaki herhangi bir adrese dallanmayı sağlar. Kısa ve yakın dallanmalar genellikle, ***segment içi dallanma (intrasegment jump)***, uzak dallanma ise ***segment arası dallanma (intersegment jump)*** olarak adlandırılır. Programcı, dallanmalarda, JMP komutundan sonra bir adres etiketi (label) kullanır. Kullanılan assembler, dallanmanın uzunluğuna göre işlem kodu ve adres üretir. Bu yüzden programcı için yapılacak tek şey JMP'den sonra dallanılacak yerin adresini yazmaktır.

4.9.1.5.Saklayıcı Operandlı Dallanma:

Dallanma komutu, bir 16-bit saklayıcı operand olarak belirlemede kullanılabilir. Bu tür dallanma, dolaylı dallanmadır ve saklayıcının içindeki adres dallanma adresi olarak kullanılır. Saklayıcının adresi IP olarak alınır ve program bu adresten itibaren devam eder. Bu çeşit bir dallanma, bir dallanma tablosu kullanmada çok yararlıdır.

Aşağıda, TABLO adresinden başlayan bir dallanma tablosunda bulunan 4 farklı altprogram adresine erişen bir örnek verilmektedir. Altprogram adresleri 16-bit olduğu için bu adresler, TABLO başlangıç adresinden itibaren TABLO+0, TABLO+2, TABLO+4 be TABLO+6 adreslerinde yer almaktadır. Program başında, SI'da 0,1,2 veya 3 olduğu varsayılmakta ve iki katı alınmaktadır. Daha sonra, TABLO taban adresiyle bu adres toplanarak dallanılacak altprogram adresi bulunur. JMP AX ile bu adrese dallanır.

```
    ; Dallanma tablosu örneği
    ADD  SI, SI            ; SI'nın iki katı
    ADD  SI, OFFSET TABLO   ; alt program adresi
    MOV  AX, CS:[SI]        ; adresi AX'e al
    JMP  AX                 ; alt programa dallan
    ...
TABLO:  DW  SUB0            ; 4 alt program adres tablosu
    DW  SUB1
    DW  SUB2
    DW  SUB3
```

4.9.1.6.İndisli Adresleme Kullanan Dolaylı Dallanma:

Dallanma komutu indisli adresleme kullanarak bir dallanma tablosuna erişebilir. Dallanma tablosu, ***yakın (near)*** dolaylı dallanma, ofset adresleri ve ***uzak (far)*** dolaylı dallanma segment ve ofset adresleri içerir. Daha önce verilen programın aşağıda indisli adresleme kullanan şekli verilmiştir.

```
; Dallanma tablosu örneği
ADD  SI, SI            ; SI'nın iki katı
ADD  SI, OFFSET TABLO   ; alt program adresi
MOV  CS:[SI]           ; alt programa dallan
...
TABLO:  DW  SUB0              ; 4 alt program adres tablosu
DW  SUB1
DW  SUB2
DW  SUB3
```

Yukarıda verilen programda, dallanma tablosuna erişim, normal hafızaya erişim gibidir. JMP CS:[SI] komutu CS hafızada bulunan ve SI ile adreslenen hücreye işaret eder. Program akışı bu hafıza adresinde saklı olan adreste yer alan adrese dallanır. Saklayıcı ve indisli adreslemeli dallanma komutunun her ikisi de ***yakın*** dallanmadır. Yani bu dallanmalarda 16-bit ofset (IP) kullanılır ve dallanma segment içidir. Eğer, segment arası dallanma yapılmak isteniyor ise, FAR PTR eki kullanılır. Örneğin, JMP FAR PTR [SI] komutunda, mikroişlemci SI ile işaretli hafıza bölgesinde 32-bit adres (IP ve CS) varsayar.

4.9.1.7.Duruma Bağlı Dallanmalar:

8086/8088 mikroişlemcisinde duruma bağlı dallanmalar kısa dallanmadır. Yani, dallanmalar daima bir sonraki komuttan sonra +127 ile –128 byte arasındadır. Duruma bağlı dallanma komutları tablo.4.21'de test durumları ile verilmiştir.

Tablo.4.21. Duruma bağlı dallanma komutları		
Komut	***Test Durumu***	***Açıklama***
JA	C = 0 ve Z = 0	Yukarı ise dallan
JAE	C = 0	Yukarı ya da eşit ise dallan
JB	C = 1	Aşağı ise dallan
JBE	C = 1 ya da Z = 1	Aşağı ya da eşit ise dallan
JC	C = 1	Elde 1 ise dallan
JE ya da JZ	Z = 1	Eşit ya da sıfır ise dallan
JG	Z = 0 ve S = 0	Büyük ise dallan
JGE	S = 0	Büyük ya da eşit ise dallan
JL	S ≠ 0	Küçük ise dallan
JLE	Z = 1 ve S ≠ 0	Küçük ya da eşit ise dallan
JNC	C = 0	Elde 0 ise dallan
JNE ya da JNZ	Z = 0	Eşit değil ya da sıfır değil ise dallan
JNO	O = 0 ve Z = 0	Taşma yok ise dallan
JNS	S = 0	S = 0 ise dallan
JNP/JPO	P = 0 (parity)	Eşlik yok ise / eşlik tek ise dallan
JO	O = 1	Taşma var ise dallan
JP/JPE	P = 1	Eşlik var ise / eşlik çift ise dallan
JS	S = 1	İşaret S = 1 ise dallan
JCXZ	CX = 0	CX = 0 ise dallan

Duruma bağlı dallanmalar şu bayrak bit'lerini test eder. İşaret (S), sıfır (Z), elde (C), eşlik (P) ve taşma (O). Eğer test edilen durum doğru ise, program akışı dallanma komutuyla belirtilen adresten devam eder. Eğer durum yanlış ise, dallanma olmaz ve bir sonraki sıradaki komut yürütülür.

İşaretli sayıların karşılaştırılmasında, JG, JGE, JE, JNE, JL, JLE komutları kullanılır. İşaretsiz sayılarda ise JA, JAE, JB, JBE, JE, JNE komutları kullanılır.Tablo.4.21'de görüldüğü gibi bazı komutlar alternatif komutlara sahiptir. Örneğin, JE alternatif JZ komutuna sahiptir. Alternatif komutların çoğu programlarda kullanılmaz, çünkü bir anlam ifade etmezler.

JCXZ komutu bayrakları test etmeyip CX saklayıcısının içeriğini ele alır. Eğer CX = 0 ise dallanma gerçekleşir, aksi durumda sıradaki komut yürütülür. Bu komut SCAS komutuyla yapılan bir dizi tarama işleminden sonra CX saklayıcısının durumuna göre dallanmada kullanılabilir. Aşağıda daha önce verilen örneğin sonuna JCXZ komutu eklenmiştir. Bu örnekte, 100 byte uzunluğunda bir hafıza bloğu içinde 7 sayısı aranmaktadır. Çevrim sonunda 7 bulunabilir veya bulunmayabilir. JCXZ komutu bu durumun testi için çok yararlıdır. Eğer bütün tablo aranmış ve 7 bulunmamış ise CX sıfır olur, aksi halde sıfır değildir.

```
    MOV    DI, OFFSET BLOCK   ; veri bloğuna işaret et
    CLD                       ; otomatik arttırma
    MOV    CX, 100            ; blok uzunluğu
    MOV    AL, 7              ; blok içinde arttırılacak veri
REPNE  SCASB                  ; bulunan kadar karşılaştır
    JCXZ   BULUNMADI
    ...                       ; bulundu ise
```

4.10.LOOP Komutu:

LOOP komutu CX saklayıcısının azaltma ile duruma bağlı dallanma işlemlerinin birleşimidir. Bu komut CX saklayıcısını bir azaltır ve eğer CX≠0, komut ile belirtilen adrese dallanır. CX sıfır olduğunda bir sonraki komut yürütülür.

Aşağıda verilen örnekte, BLOCK1 ve BLOCK2 adresli yerden başlayan, 100 kelime uzunluğunda iki dizinin toplanmasında LOOP komutu kullanılmıştır. LODSW komutu BLOCK1 verisine ve STOSW komutu ise BLOCK2 verisine erişmektedir. ADD AX, ES:[DI] komutu ES alanında bulunan BLOCK2 verisine erişmektedir. BLOCK2'nin ES alanında olmasının tek nedeni, STOSW komutundaki DI saklayıcısının ES alanındaki veriyi adreslemiş olmasıdır.

```
    MOV   SI, OFFSET BLOCK1   ; veri bloğuna işaret et
    MOV   DI, OFFSET BLOCK2   ; veri bloğuna işaret et
    MOV   CX, 100             ; blok uzunluğu
    MOV   AL, 7               ; blok içinde arttırılacak veri
TEKRAR:
    LODSW                     ; BLOCK1 verisi oku
    ADD   AX,ES:[DI]          ; BLOCK2 verisi ile topla
    STOSW                     ; BLOCK2 alanında sakla
    LOOP TEKRAR               ; 100 kere tekrar et
```

4.10.1.Duruma Bağlı LOOP Komutları:

LOOP komutu durma bağlı olabilir. Sık olarak kullanılan iki tane duruma bağlı LOOP komutu vardır. Bunlar: LOOPE ve LOOPNE komutlarıdır.

LOOPE (LOOP while Equal) eşit olduğu sürece çevrim komutu, CX sıfır değil ise ve bir eşitlik durumu olmadığı sürece, adresle belirtilen yere dallanır. CX sıfır olduğunda veya bayrak bit'leri ile bir eşitsizlik durumu belirtildiğinde çevrimden çıkar.

LOOPNE (LOOP while Not Equal) eşit olmadığı sürece çevrim komutu, CX sıfır değil ise ve eşitsizlik durumu olduğu sürece, adresle belirtilen yere dallanır. CX sıfır olduğunda veya bayrak bit'leri ile bir eşitlik durumu belirtildiğinde çevrimden çıkar.

4.11.Altprogramlar:

Altprogram, bilgisayar yazılım mimarisinde önemli bir yer taşır. Genelde bir görev yerine getiren komutlar kümesidir. Ve hafızada bir kere yer aldıktan sonra, bir programda birçok kere kullanılır. Bu, hafıza alanından tasarruf sağlar ve altprogramlar içeren bir programı yazmak daha az zaman aldığı için programcılığı kolaylaştırır. Altprogramın bir dezavantajı,

altprogram ile ana program arasındaki parametre aktarımdaki ve altprogram çağırma (CALL) ve altprogramdan dönüş (RET) işlemlerindeki ek zaman kaybıdır.

Bir altprogram CALL komutu ile çağrılırken, yığın (stack) hafıza, CALL komutundan sonraki komutun adresini saklamada kullanılır. RET komutu ile, yığından okunan dönüş adresinden itibaren dönüş devam eder.

Bir x86 ***Assembler*** programı kullanırken alt program yazmanın bazı kuralları bulunmaktadır. Örneğin, bir altprogram, Pascal gibi yüksek seviyeli bir dildeki gibi ***procedure*** olarak adlandırılır. Altprogram PROC yönlendiricisi (directive) ile başlar ve ENDP yönlendiricisi ile biter. Bu, bir altprogramı diğer kodlardan ayırmayı sağlar. Bu iki yönlendiriciden önce altprogramın adı yazılır. PROC yönlendiricisini, altprogramın tipini belirten NEAR (segment içi) veya FAR (segmentler arası) yönlendiricisi takip eder. Aşağıda NEAR ve FAR yönlendiricisi kullanan, iki örnek altprogram yapısı verilmiştir.

```
 ; SUB0 adında NEAR altprogram yapısı
SUB0  PROC NEAR   ; altprogram adı ve başlangıcı
    ...
    RET
SUB0  ENDP        ; altprogram sonu

 ; SUB1 adında NEAR altprogram yapısı
SUB1  PROC NEAR   ; altprogram adı ve başlangıcı
    ...
    RET
SUB1  ENDP        ; altprogram sonu
```

Yukarıdaki iki altprogram arasındaki fark, assembler tarafından RET komutu için üretilen kodda bulunmaktadır. NEAR RET komutu C3H işlem kodu kullanır buna karşın FAR RET komutu ise CBH işlem kodu kullanır. Yakın RET, yığından bir 16-bit sayı çeker ve bunu alt programdan dönmek için IP saklayıcısına yerleştirir. Uzak RET ise, yığından bir 32-bit sayı çeker ve bunu altprogramdan hafızanın herhangi bir yerine dönmek için IP ve CS saklayıcılarına yerleştirir.

Bir MASM (Microsoft Assembler) programında NEAR ve FAR tipini USES ifadesi takip eder. Bu USES ifadesi, istenen saklayıcıların otomatik olarak yığın hafızaya atılmasını ve alt program sonu yığından çekilmesini sağlar.Birçok program tarafından kullanılacak (global) altprogramlar FAR olarak tanımlanmalıdır. Bir görev tarafından kullanılan altprogramlar, normalde NEAR olarak tanımlanır.

4.12.CALL Komutu:

CALL (çağırma) komutu program akışını bir altprograma aktarır. Bu komutun JMP komutundan farkı, bir CALL komutu yürütülürken mikroişlemci tarafından otomatik olarak dönüş adresi yığında saklanır. Altprogram sonundaki RET komutuyla bu dönüş adresi yığından çekilir ve programda CALL komutundan sonraki komuta dönülmüş olur.

4.12.1.Yakın CALL:

Yakın CALL komutunda, 8086-80286'da, ilk byte işlem kodundan ikinci ve üçüncü byte'lar, uzunluğu ±32K değişim adresi olup toplam 3-byte'tır. 80386 üstü işlemciler, korumalı modda çalıştıklarında, 32-bit değişim kullanılarak ±2G byte'lık bir alan sağlar. Yakın CALL daha önce gördüğümüz yakın JMP komutuna benzemektedir. Kısa CALL komutu bulunmamaktadır.

4.12.2.Uzak CALL:

Uzak CALL, 5-byte uzunluğundadır ve işlem kodundan sonra dallanacak altprogramın IP ve CS adresleri komut içinde bulunur. FAR CALL komutu yürütülürken, dallanmadan önce, yığına o anki IP ve CS yerleştirilir. Bu sayede hafızanın herhangi bir yerine yerleştirilmiş bir altprogram çağrılır.

4.12.3.Saklayıcı Operand'lı CALL:

JMP komutunda olduğu gibi, CALL da bir saklayıcıyı bir operand olarak kullanabilir. Örneğin, CALL BX komutunda, önce o anki IP yığına atılır. Daha sonra, BX saklayıcısındaki ofset adres, o anki kod segment'te bulunan bir altprogram çağrımı için IP adresi olarak alınır. Bu tip bir CALL komutunda, bir16-bit saklayıcıda bulunan sayı 16-bit ofset olarak kullanılır.

4.12.4.Dolaylı Hafıza Adresi Kullanan CALL:

Bu tür bir altprogram çağırma, değişik altprogramları bir parametreye göre seçip çağırmada yararlıdır. Aşağıda verilen örnekte, 3 altprogramdan biri, DI'daki bir parametreye göre çağırılmaktadır. Altprogramlar, DI'daki 0,1 ve 2 sayılarına göre belirtilmektedir.

Eğer, bu indisler 1'den başlasaydı, programın başında DI saklayıcısını 1 azaltmamız gerekecekti. Altprogram adresleri, TABLO adresinden itibaren 2-byte aralıklarla saklandığından, bu tabloya erişirken DI saklayıcısındaki indisin 2 katı alınır. 'CS:' öneki CALL komutundaki operand'ın önüne gelmektedir. Çünkü [BX+DI] ile CS alanında bulunan TABLO'ya erişilmektedir.

```
; alt program çağırma tablosu örneği
    TABLO  DW   SUB0            ; Altprogram look-up tablosu
    DW   SUB1
    DW   SUB2
; Altprogram çağırma komutları
    ADD  DI, DI             ; DI'nın iki katı
    MOV  BX, OFFSET TABLO   ; TABLO'ya işaret et
    CALL CS:[BX:DI]         ; Alt program çağır
SUB0  PROC NEAR
    ...
    RET
SUB0  ENDP
 SUB1  PROC NEAR
    ...
    RET
SUB1  ENDP

SUB2  PROC NEAR
    ...
    RET
SUB2  ENDP
```

CALL komutuyla uzak adresler uzak adreslerde kullanılabilir. Bu durumda CALL komutundan sonra FAR PTR kullanılır. Örneğin, CALL FAR PTR [SI] gibi. Bu durumda, DS alanında SI ile işaretli alandan 32-bit bir adres okunur ve bu adres FAR olarak tanımlı bir altprogram adresi olarak işlenir.

4.13.RET Komutu:

Dönüş RET komutu, yığın hafızadaki yakın dönüş için 16-bit bir sayı veya uzak dönüş için 32-bit bir sayı çeker. Bu sayıyı IP saklayıcısına (yakın dönüşte) ve IP ve CS saklayıcılarına (uzak dönüşte) yerleştirir. NEAR ve FAR RET komutları, altprogram tamamlanmasında PROC yönlendirmesiyle belirlenir ve buna göre kullanılan assembler yakın ve uzak RET komutu için işlem kodu üretir.

RET komutunun bir operand kullanan bir şekli daha vardır. Bu RET komutunda altprogramdan dönmeden önce SP içeriğine RET komutundan sonra yazılan operand eklenir. Bu işlem, yığındaki verinin geçilmesi veya silinmesi anlamına gelir. Örneğin, aşağıda verilen kod örneğindeki altprograma girildikten sonraki iki PUSH işlemiyle yığına iki tane 16-bit sayı yazılarak yığın 4 byte büyümüştür. RET 4 komutu, SP'ye 4 ekleyerek bu 4 byte'ın atılmasına neden olur. Bu şekilde RET komutuyla, bu altprogramı çağıran CALL komutundan sonraki satıra doğru olarak dönmüş oluruz.

```
TEST  PROC NEAR
    PUSH AX
    PUSH BX
    ...
    RET  4
TEST  ENDP
```

Eğer, bir altprogramda yapılan PUSH ve POP'ların sayısı birbirine eşit değil ise, bu şekilde bir RET komutu kullanılabilir. Yığın hafızanın, RET komutundan önce, düzgün olarak bırakılması, yani yığının en üstünde dönüş adresinin olması gerekmektedir. Aksi durumda, altprogramdan dönüş istenmeyen bir hafıza adresine olur.

4.14.Kesmelere Giriş:

Kesme, bir donanım (bir ***harici*** donanım sinyalinden) veya yazılımın ürettiği bir CALL (bir komut ile ***dahili*** olarak üretilen) işlemidir. Her iki durumda da bir ***kesme hizmet programı (Interrupt Service Routine - ISR)*** çağrılır. Bu bölümde, CALL komutunun özel

çeşitleri olan yazılım kesmeleri sunulacaktır. Yazılım kesmelerinde üçü (INT, INTO ve INT 3) tanıtılacak, kesme vektörleri anlatılacak ve özel kesme dönüş komutu IRET'in çalıştırılması açıklanacaktır.

4.14.1.Kesme Vektörleri:

Kesme vektörü, mikroişlemci gerçek modda çalışırken, hafızanın ilk 1024 byte'lık alanında (003FFH-00000H) saklı olan 4-byte bir adrestir. Her bir kesme vektörü, bir kesme hizmet programının adresini oluşturan bir IP ve CS içerir. İlk 2 byte IP ve son 2 byte CS adresidir. 256 tane farklı kesme vektörü bulunur. Her vektör, kesme ile çağrıları bir kesme hizmet programının adresini tutar. Tablo.4.22, açıklamalarıyla beraber, kesme vektörlerini ve gerçek modda her vektöre karşı gelen adresleri göstermektedir.

Intel ilk 32 kesme vektörünü (0-31) 8086-80486 ve gelecek ürünler için ayırmaktadır. Geri kalan kesme vektörleri (32-255) kullanıcı içindir. Ayrılı vektörlerden bazıları, bölme hatası gibi, program yürütme sırasında oluşan hatalar içindir. Bazıları yardımcı işlemci için ayrılmıştır. Diğerleri sistemdeki diğer durumlar ve korumalı modda çalışmada oluşan hatalar içindir. Korumalı modda kesme vektör tablosu yerine, daha önce gördüğümüz gibi, her kesme için 8-byte içeren, kesme tanımlayıcı (descriptor) tablosu yer alır.

4.14.2.Kesme Komutları:

8086-80486 üç farklı kesme komutuna sahiptir. INT, INTO ve INT3. Gerçek modda, bu komutlardan her biri, vektör tablosundan bir vektör okur ve sonra bu vektörle adreslenen kesme hizmet programını çağırır. Korumalı modda bu komutlardan her biri kesme tanımlayıcı tablosundan bir kesme tanımlayıcısı okur. Bu okunan tanımlayıcı hizmet programını belirtir. Kesme çağırma, uzak CALL komutuna benzer; çünkü dönüş adresi (IP/EIP ve CS) yığına yerleştirilir.

Tablo.4.22. Kesme Vektörleri

Kesme no	***Adres***	***Mikroişlemci***	***İşlev***
0	0H-3H	Bütün	Bölme hatası
1	4H-7H	Bütün	Tek adım
2	8H-BH	Bütün	NMI

3	CH-FH	Bütün	Durma noktası
4	10H-13H	Bütün	Taşma kesmesi
5	14H-17H	80186-Pentium	BOUND kesmesi
6	18H-1BH	80186-Pentium	Geçersiz işlem kodu
7	1CH-1FH	80186-Pentium	Yardımcı işlemci (emulation) kesmesi
8	20H-23H	80386-Pentium	Çift hatası (Double fault)
9	24H-27H	80386	Yardımcı işlemci segment overrun
10	28H-2BH	80386-Pentium	Geçersiz durum segment'i
11	2CH-2FH	80386-Pentium	Segment mevcut değil
12	30H-33H	80386-Pentium	Yığın hatası
13	34H-37H	80386-Pentium	Genel koruma hatası
14	38H-3BH	80386-Pentium	Sayfa hatası
15	3CH-3FH	----	Ayrılmış*
16	40H-43H	80286-Pentium	Kayan-nokta hatası
17	44H-47H	80486SX	Ayarlama (alignment) kontrol kesmesi
18	48H-4FH	Pentium	Makine kontrol
19-31	50H-7HH	----	Ayrılmış*
32-255	80H-3FFH	Bütün	Kullanıcı kesmeleri

4.14.2.1.INT:

Programcının kullanabileceği 256 tane farklı yazılım kesme komutu (INT) bulunmaktadır. Her INT komutu, 2-byte uzunluğunda olup değeri 0 ile 255 (FFH-00H) arasında değişen bir nümerik operand'a sahiptir. İlk byte işlem kodu, ikinci byte vektör tipi numarasıdır. Yalnızca yazılım kesmesi INT3 farklılık gösterir ve bu komut 1-byte uzunluğundadır.

INT komutu ile yapılan işlem, FAR CALL komutu ile yapılana benzemektedir. INT komutu 2 byte olmasına karşın FAR CALL 5 byte uzunluğundadır. CALL yerine INT komutunun kullanılması, INT komutunun çok kullanıldığı programlarda, önemli bir hafıza tasarrufu sağlar.Kesme vektörünün adresini hesaplarken, kesme tipi numarası 4 ile çarpılır. Örneğin, INT 10H komutu, gerçek modda, adresi hafızanın 40H hücresinde saklı olan bir kesme hizmet programını çağırır. Korumalı modda ise, bu sayı 4 yerine 8 ile çarpılır. Çünkü her kesme tanımlayıcısı 8-byte uzunluğundadır.

Bir yazılım kesmesi yürütüldüğünde:

- Bayrak saklayıcısı yığına atılır.
- T ve I bayrakları temizlenir.
- CS yığına atılır ve CS için vektörlerden yeni bir değer okunur.
- IP/EIP yığına atılır ve IP/EIP için vektörden yeni bir değer okunur.
- Yeni CS:IP/EIP ile adreslenen yere dallanır.

INT komutu yürütüldüğünde, harici donanım kesme giriş ucu INTR'i (Interrupt Request) kontrol eden kesme bayrağı (I) temizlenir. I = 0 olduğunda mikroişlemci INTR ucunu pasifler (disabled), yani bu uçtan gelecek kesmelere cevap vermez. I bayrağı yeniden 1 olduğunda bu kesme giriş ucu aktif duruma gelir.

Yazılım kesmeleri, genellikle sistem programları çağırmada kullanılır. Sistem programları, bütün sistem ve uygulama programları tarafından kullanılabilen programlardır. Örneğin, IBM PC'de yazılım kesmeleri, yazıcıları, video birimlerini ve disk sürücüleri gibi donanım birimlerini kontrol etmede kullanılır.

4.14.2.2.IRET/IRETD:

IRET komutu gerçek modda, IRETD komutu ise korumalı modda kullanılır. Kesme dönüş komutu IRET yalnız yazılım veya donanım kesme hizmet programlarında kullanılır. Basit dönüş RET komutundan farklı olarak, IRET komutu, yığından IP ve CS olarak alınacak 2 tane 16-bit veri çektikten sonra bayrak saklayıcısına bir 16-bit veri yığından okur. IRET komutu yürütüldüğünden, I ve T eski değerlerine döner. 80286-80486 mikroişlemcilerinde, korumalı modda çağrılmış bir kesme hizmet programından dönüş için IRET komutu kullanılır. IRET komutu komutundan farklı olarak, yığın hafızadan 32-bit IP (EIP) çekilir.

4.14.2.3.INT3:

Özel bir yazılım kesmesi olan INT3, programlarda durma noktası (break point) oluşturmada kullanılır ve uzunluğu 1-byte'tır. Genellikle program hatalarını bulurken program akışını kırmada ve kesmede kullanılır.

4.14.2.4.INTO:

Taşıma durumunda oluşan bu kesme, taşıma bayrağını (O) test eden duruma bağlı bir yazılım kesmesidir. Eğer O=0 ise INTO komutu bir işleme neden olmaz, fakat O=1 ise ve INTO komutu yürütülmüş ise, vektör tip numarası 4 olan bir kesme oluşur.

INTO komutu işaretli ikili sayıları toplayan veya çıkaran program işaretli iki sayıyı toplayan veya çıkaran programlarda kullanılır. JO veya INTO komutu bir taşma durumunu bulmak için kullanılabilir.

4.14.2.5.Kesme Hizmet Programı:

Bir kesme kesme altprogramının tanımlanması daha önce verilen, bir alt program yazma işlemi gibidir. Tek fark, altprogramda kullanılan RET komutunun yerine, kesme hizmet programında IRET komutunun kullanılması ve bu kesme alt programının adresinin, programın başında ilk işlemler yapılırken, kesme tablosunda yerleştirilmesidir. Aşağıda bir kesme hizmet programının genel çerçevesi verilmiştir.

```
INTS  PROC FAR
    ...
    RET  4
INTS  ENDP
```

4.14.2.6.Kesme Kontrol:

INTR donanım kesmesini kontrol eden iki komut bulunmaktadır. ***STI (Set Interrupt flag)*** komutu I bayrağını 1'lemekte ve bu da INTR girişini aktif duruma getirmektedir. ***CLI (CLear Interrupt flag)*** komutu I bayrağını 0'lamakta ve bu da INTR girişini pasif yapmaktadır. Bir yazılım kesme hizmet programı içinde, ilk adım olarak, genellikle donanım kesmeleri STI komutu kullanılarak aktif yapılır.

4.14.2.7.İşlemci Kontrol ve Diğer Komutlar:

8086-80486 komut kümesindeki son grup, işlemci kontrol komutları ile şimdiye kadar olan gruplara girmeyen diğer komutlardır. Bu komutlar elde bayrağını kontol, BUSY / TEST ucunu örnekleme, mikroişlemciyi durdurma gibi fonksiyonlar sağlar.

4.14.2.8.Elde Bayrağını Kontrol:

Elde bayrağını uzun Word toplama ve çıkarma işlemlerinde kullanılır. Ayrıca, bir altprogramdan dönerken, bir hata durumu veya başka bir durum, 1 veya 0 ile çağrılan program bildirmede boolean bayrak olarak ta kullanılır.

Elde bayrağını (C) kontrol etmeye yönelik 3 tane komut bulunmaktadır. Bu komutlar: elde bayrağını 1'le, STC (*Set Carry*); elde bayrağını temizle, CLC (Clear Carry); elde bayrağının tersini al, CMC (Complement Carry).

4.14.2.9.WAIT:

WAIT komutu 80286-80386'da $\overline{\text{BUSY}}$ donanım ucunu, 8086/8088'de ise $\overline{\text{TEST}}$ ucunu gözler. Bu ucun adı 80286 işlemcisinden itibaren $\overline{\text{TEST}}$'ten $\overline{\text{BUSY}}$'ye değişti. WAIT komutu yürütüldüğünde, bu uç 0 ise, bir şey olmaz ve bir sonraki komut yürütülür. Eğer bu uç 1 ve WAIT komutu yürütülmüş ise, mikroişlemci bu ucun lojik 0 seviyesinde dönmesini bekler. IBM PC'de, mikroişlemcinin $\overline{\text{BUSY}}$ ucu, genellikle 8087-87387gibi nümerik yardımcı işlemcinin BUSY ucuna bağlanır. Bu bağlantı ve WAIT komutu, yardımcı işlemci görevini bitirene kadar, 8086-80386 işlemcisinin beklemesini sağlar.

4.14.2.10.HLT:

HLT komutu program yürütmesini durdurur. Bir durma işleminde çıkmak için 3 yol vardır: bir kesme ile, bir donanım kesmesi veya bir DMA işlemi sırasında. Bu komut, normalde bir kesmeyi beklemek amacıyla kullanılır. Harici donanım kesmeleri ile yazılım sistemini senkronize etmeye de yararlıdır.

4.14.2.11.NOP:

Mikroişlemci NOP (***No Operation***) komutuna rastladığı zaman, bir işlem yapmayıp sadece çok kısa bir zaman harcar. Bu zaman, bu komutun hafızadan okunma (fetch) süresidir. Program geliştirilirken, bu komut ileride başka komutlarla değiştirilmek üzere, program aralarında çeşitli yerlere konulabilir. Bu sayede NOP konulduğu yerler, programcıya eklerin yapılacağı önemli noktaları hatırlatır. NOP komutu aracılığıyla, yazılım gecikmeleri sağlamak amacıyla sık olarak kullanılır.

4.14.2.12.LOCK Ön Eki:

LOCK ön eki bir komuta ilave edildiği zaman $\overline{\text{LOCK}}$ donanım ucu lojik 0 olur. bu uç genellikle harici yol işlemlerini düzenleyici tüm devreleri (***bus masters***) ve diğer sistem birimlerini pasif duruma getirir. $\overline{\text{LOCK}}$ ön eki bir ya da sıralı birden fazla komutun önünde olduğunda, bu komut ya da komutlar *kilitli* durumda olur. bu komutların yürütülmesi sırasında, LOCK ucu lojik 0 seviyesinde kalır. Örneğin, LOCCCCKKK komutu kilitli bir komuta örnektir.

4.14.2.13.ESC:

ESC (***Escape***) komutu, 8087-80387 nümerik işlemcisine mikroişlemciden bilgi aktarmada kullanılır. ESC komutu yürütüldüğünde, mikroişlemci, istenirse, bir hafıza adresi sağlar, aksi durumda, bir NOP işlemi yapar.