2.BÖLÜM

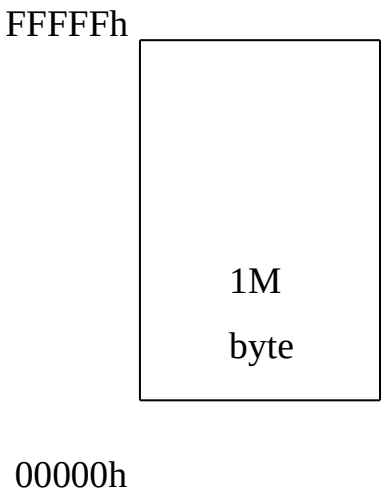
2.ADRESLEME MODLARI

2.1.8086/8088 Hafıza Mimarisi

2.1.1.Lojik ve Fiziksel Hafızalar

8086/8088 mikroişlemcisi 20-bit Adres Yolu ile toplam 1048576 (1M) byte hafıza hücresi adresleyebilmeye karşın, her iki işlemcinin fiziksel hafıza yapıları farklıdır. Bununla beraber bu işlemcilerin lojik hafızaları Şekil 4.13'te görüldüğü gibi aynıdır. ***Lojik hafıza***, yazılım tarafından programcıya görülen hafızaya verilen isimdir. Bu hafıza, donanım tasarımcısı tarafından görülen, gerçek hafıza yapısını oluşturan ***fiziksel hafızadan*** farklı olabilir.

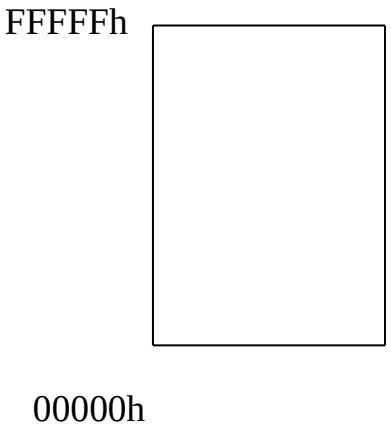
8088 ve 8086 mikroişlemcilerin lojik hafızası 0000h'tan başlayıp FFFFh'a kadar uzanır. Lojik hafıza genişliği bir byte (8-bit) olup bu adreslerin uzunluğu 1M byte hafıza bloğu belirtir. Mikroişlemci tarafından adreslenen 16-bit hafıza kelimesi, her herhangi bir byte adresinden başlar ve peş peşe 2 byte işgal eder.



Şekil 2.1. 8086/8088 lojik hafıza haritası.

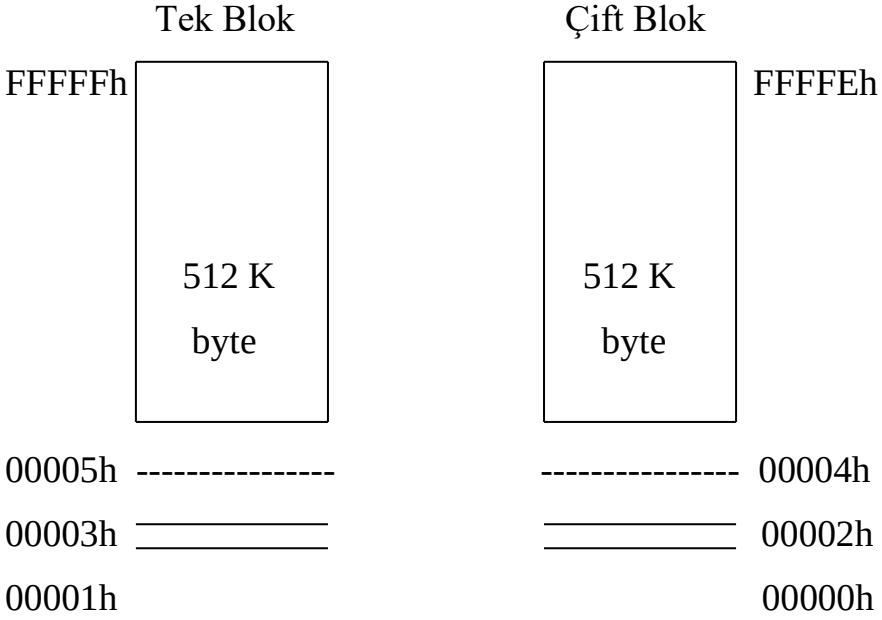
8088 ve 8086 fiziksel haritaları genişlik olarak birbirinden farklıdır. 8088 hafızası 8-bit, 8086 hafızası ise 16-bit genişliğindedir. Programcı için hafıza her zaman 8-bit genişliğinde olmasına rağmen, fark sadece tasarımcısı için bulunur.

Şekil 4.14.8088 mikroişlemcisinin hafıza alanını göstermektedir. Bu hafıza haritası Şekil 4.13'te verilen lojik hafıza haritası ile aynıdır. 8088'in hafıza arabirimi 8085A işlemecisine benzemektedir.



Şekil 3-2. 8088 fiziksel hafıza haritası.

8086 mikroişlemcisinin fiziksel hafıza haritası Şekil 4.15'te görülmektedir. 8088'den farklı olarak iki ayrı hafıza bloğu içermektedir. Tek blok (yüksek hafıza) ve çift blok (düşük hafıza). Her 8086 bloğu 512K*8 olup toplam adreslenebilir, 8086, byte veya kelime (16-bit, Word) verisini doğrudan adresleyebilmektedir. Bundan dolayı, 8086, 16 bit bir kelimeyi bir işlemde okuyup yazabilmektedir (verinin adresinin çift olması sağlandığında). Buna karşın, 8088, 16-bit veri aktarımı için 2 okuma veya yazmaya gerek duyar. 8086 yazılımı daha hızlı çalışır. Çünkü, 8086 bir çok komutu ve 16-bit veriye 8086'in iki katı hızında erişir.

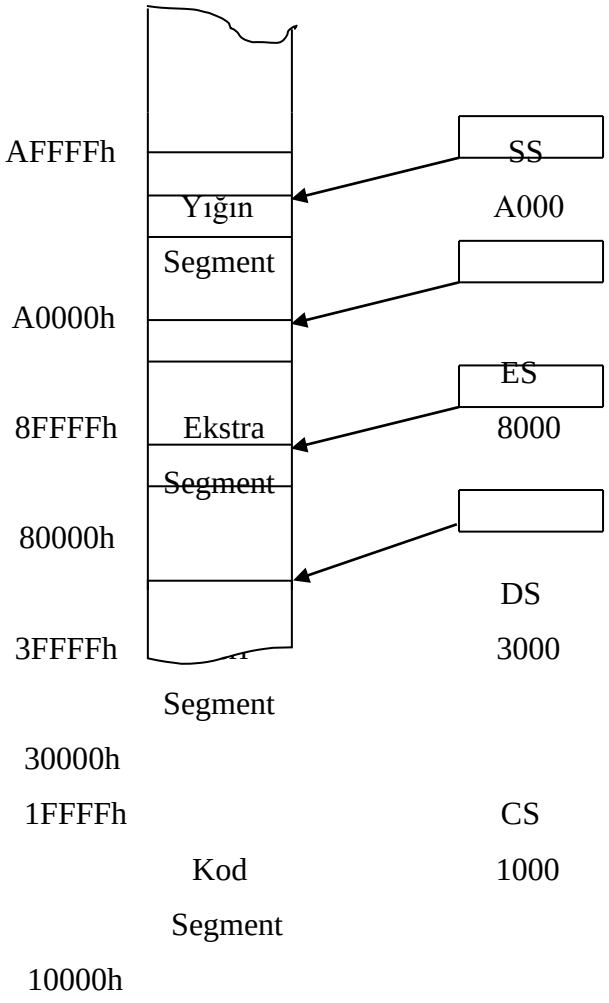


Yüksek Hafıza Düşük Hafıza

Şekil 3-3. 8086 fiziksel (donanım) hafıza haritası

2.1.2. Segment'li Hafıza Yapısı

8086/8088 mikroişlemcilerinde hafızaya erişim segment saklayıcıları yoluyla yapılır. Her bir segment bloğu 64K byte'tır. Şekil 4.16'da segment saklayıcıları ile adreslenen bir hafıza haritası örneği görülmektedir. Hafıza alanının aynı anda 4 farklı segment bulunabilmektedir. Bunlar kod segment (CS), veri segment (DS), ekstra segment (ES), ve yığın segment (SS).



Şekil.2.4. Segment saklayıcıları ile adreslenen bir hafıza haritası

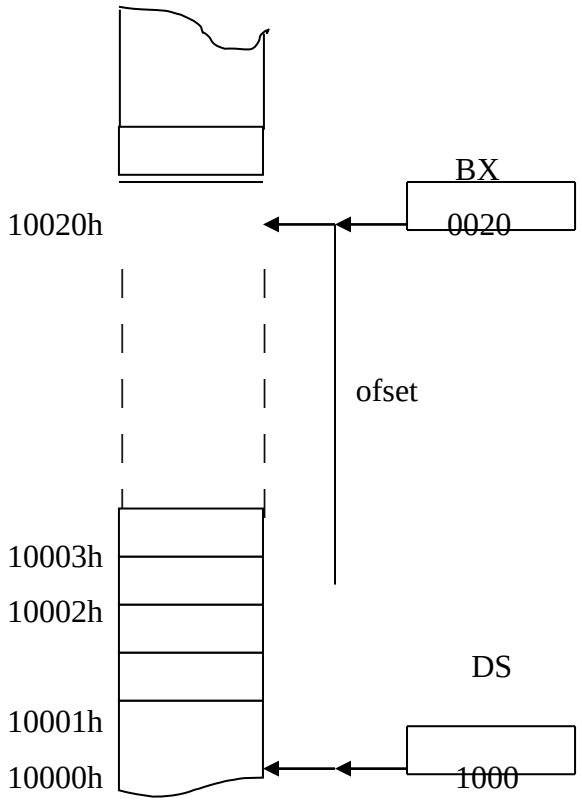
Her segment saklayıcısı, 20-bit adresin 16-bit kısmını tutar. Segment saklayıcıda bulunan 16-bit adresin düşük değerli bölümüne, 0h (0000h) eklenir. Ayrıca bir ofset (indis) bu adresle toplanarak, donanım tarafından otomatik olarak, 20-bit adres elde edilir. Bu işleme lojik adresin fiziksel

adrese çevrilmesi denir. Segment saklayıcılarına göre, aşağıda açıklanacağı gibi, ofset, değişik saklayıcılardan gelir.

Eğer bir 8086/8088 sistemi sadece 64K byte hafıza içeriyorsa, bütün 4 segment saklayıcısı 0000h ile yüklenir ve segment'ler üst üste çakışır. Bu durumda adres aralığı X0000h – XFFFFh arasında değişir. (X herhangi bir 16-lı rakamdır.)

Her bir segment saklayıcısı, özel bir fonksiyona sahiptir ve bir veya daha çok indis veya işaretçi saklayıcısı ile ilişkilidir. Bir hafıza adresi üretmek için, bir segment saklayıcısının içeriği, bir ofset adres tutan bir indis veya işaretçi saklayıcısına eklenir.

Şekil 4.17'de bir komut tarafından adreslenen bir verinin, fiziksel adresinin üretimi görülmektedir. Bu örnekte, veri segment saklayıcısı DS 1000h içermekte, yeni veri segment 10000h fiziksel adresinden itibaren başlamaktadır. Taban saklayıcısı BX'te ofset adres 0020h bulunmakta, ve böylece fiziksel adres 10020h veya 1000h*10h+0020h olmaktadır.



Şekil .2.5. Veri segment'inde bulunana bir hafıza hücresine erişim

Kod segment CS, program ve veri alanı olarak kullanılabilmesine rağmen genelde kodlarının bulunduğu alandır. 8086/8088 tarafından yürütülecek bir sonraki komutun adresi, komut işaretçisi IP'ye CS * 10h içeriğinin toplanmasıyla elde edilir.

Veri segment'i DS, bir çok komut ve adresleme modu tarafından erişilen program verilerini tutar. Hafızadaki verinin adresi, BX,SI veya DI saklayıcılarından birine DS * 10h içeriğinin toplanmasıyla elde edilir.

Yığın (stack) segment'i SS, LIFO (Last In First Out) tarzında çalışmaktadır. Bir yığın hücresinin adresi, SP'nin içeriği artı SS *10h'tır. BP tarafından adreslenen veri de normal yığın segment'inde bulunur.

ES *string* komutlarından kullanılan veri alanıdır. Bir string komutu yürütüldüğünde, hedef adres DI ES * 10h, kaynak veri adresi ise, SI artı ES * 10'tır.

Aşağıda Tablo 4.5'te 8086/8088 mikroişlemcisinin değişik işlemleri ve bunlarda otomatik olarak kullanılan saklayıcılar görülmektedir.

Tablo 3.1. İşlemlere göre değişik segment adres kaynakları.

İşlem Çeşidi	***Segment***	***Alternatif Segment***	***Ofset***
Komut okuma	CS	yok	IP
Yığın işlemi	SS	yok	SP
Veri İşlemi (aşağıdakiler hariç)	DS	CS,ES veya SS	çeşitli
String kaynak	DS	CS,ES veya SS	SI
String hedef	ES	yok	DI
BP taban saklayıcı olarak kullanıldığında	SS	CS,ES veya SS	çeşitli

2.1.2.1. Segment'li Hafıza Yapısının Avantajları

Yukarıda anlatılan segment'li hafıza yapısı, ilk bakışta şaşırtıcı ve zor görülebilir. Bu hafıza yapısı için hatırlanması gerekenler özetle şunlarlar. Program işlem kodları CS alanından okunmakta, program verileri DS ve ES alanlarında saklanmaktadır. Yığın işlemleri ise, SP ve BP saklayıcılarını kullanarak SS üzerinde işlem yapar.

Ayrı kod ve veri alanlarının olmasının ilk avantajı, bir programın, farklı veri blokları üzerinde çalışabilmesidir. Bu işlem, DS saklayıcısına farklı bir bloğa işaret eden yeni bir adresin yüklenmesiyle yapılır.

Segment'li hafıza yapısının en büyük avantajı, lojik adresler üreten x86 programlarının hafızanın herhangi bir yerine yüklenip çalıştırabilmesidir. Bunun nedeni, lojik adreslerin, her zaman, CS taban adresinden bağımsız olarak, 0000h ile FFFFh arasında değişmesidir.

Çok-görevli (multi-tasking) x86 tabanlı bir ortamı veya Windows 95/98/NT işletim sistemini düşünelim. Bir çalışan aktif programın geçici olarak sabit diskte saklandığını ve onun yerine, yeni bir programın getirildiğini farz edelim. Bu çeşit programla, hafızanın herhangi bir yerinde çalışacakları için, ***tekrar yerleştirilebilir (relocatable)*** olarak adlandırılır. Programların bu şekilde çalışabilmesi, segment saklayıcıları yoluyla olmaktadır. Segment saklayıcılarının yani program, veri ve yığın saklayıcılarının taban adresinin değiştirilmesiyle programlar hafızanın herhangi bir yerinde çalışabilmektedir.

8085A ve benzeri birçok 8-bit mikroişlemci ve mikro denetleyicide, bu şekilde bir hafıza adres üretimi olmadığı için, programların tekrar yerleştirilebilirliği mümkün değildir. Yani, bir 8085A işlemcisi için, program ve verilerin başlangıç adresleri, ORG (Origin) gibi assembler ifadeleri ile belirlenip kod üretimi yapıldıktan sonra, bu adreslere, program ve veriler yüklenmelidir.
Bu şekilde, program ve veriler, bir bakıma, hafızaya *kök salar* ve başka yere taşınamaz. Program ve verileri başka bir adrese yükleyebilmek için, ORG ifadesindeki adresler değiştirilir, sonra tekrar derleme yapılır ve yeni ikili kod üretilir. Bu sayede yeni adres yerleşimi için kod elde edilir.

2.2. Adresleme Modları

80286 (Kısa 286) işletmek üzere sizin programınızın operand'larını oluşturmak için yolların çeşitliğini destekler. 286 bunları bir tutucudan oluşturabilir., kendisi bir direktifle, ya da bellek konumundan ya da bir I/O kapısından. Hepsi için adresleme kiplerini 7 gruba ayırabilirsiniz.

1) Saklayıcı adresleme

2) Kesin adresleme

3) Doğrudan adresleme

4) Saklayıcı dolaylı adresleme

5) Tabana dayalı adresleme

6) Doğrudan dizilmiş adresleme

7)Taban dizili adresleme

Mikroişlemci direktifin içindeki bir “mod alanı” ‘nın içeriğini sınayarak. Bu yedi adresleme modunu kullanmak için belirtir. Assembler mod alanını kaynak programdaki operandlarının görünüşlerinin tabanında ayarlar.

Örnek olarak bunu girerseniz.

MOV AX, BX

Assembler her iki operandı (AX ve BX) saklayıcı adresleme modu için kodlar. Bununla beraber, eğer kaynak operandın başına ve sonuna köşeli parantez koyarsanız ve girerseniz.

MOV AX, [BX]

Assembler kaynak operandını (BX) saklayıcı dolaylı adresleme modu için kodlar.

Tablo 3-1 asembler formatını gösterir ve 286’nın 7 operand adresleme kipi için hangi segment tutucusu fiziksel adresini hesaplamak için kullanıldı.

Not: Bütün kipler data segmentin içindeki verinin bu durumda yığın segmentin varsaydığı başvurmayı üstlenir. (DS segment saklayıcısıdır). BP saklayıcısının kapsadığını hariç tutar. (SS segment saklayıcısıdır.)

Not: 286’nın string direktifleri DI noktalarının ekstra segmentteki yerleşimi data segmentdekinden daha iyidir.

Tablo 2.1. 80286 Adresleme Modları:

Adresleme Modu	Operand Biçimi	Segment Tutucu
Tutucu	reg	Yok
En yakın	data	Yok
Doğrudan	disp	DS
	label	DS
Dolaylı saklayıcı	[BX]	DS

	[BP]	SS
	[DI]	DS
	[SI]	DS
Göreceli taban	[BX]+disp	DS
	[BP]+disp	SS
Doğrudan dizilmiş	[DI]+disp	DS
	[SI]+disp	DS
Taban dizilmiş	[BX][SI]+disp	DS
	[BX][DI]+disp	DS
	[BP][SI]+disp	SS
	[BP][DI]+disp	SS

Notlar:1) “disp” taban izilmiş adresleme için seçimliktir.

2) “reg” her 8 veya 16-bit saklayıcı olabilir., IP hariç

3) “data” bir 8 veya 16-bit sabit değerinde olabilir.

4) “disp” bir 8 veya 16-bit işaretlenmiş ayırma değerinde olabilir.

ES segment saklayıcısı gibi kullanıldı. Bütün diğer talimatlar Tablo 3-1’de atanmış olarak gösterildi. Bu bölümün devamında, her zaman tanımladığımız bir adresleme kipini, bir örnekte kullanımını sergileyeceğiz. Genellikle, biz 286’larda MOV (Move) talimatını bunu sergilemek için kullanırız.

2.2.1.Saklayıcı ve Kesin Adresleme

Tutucu adreslemede, 286 operandı tutucudan (ya da içine yüklenenden) gidip getir. (Swap)

Örneğin: MOV AX, CX

16-bit içerikli sayaç tutucusunu (CX), akümülatör tutucusuna (AX) taşır. CX saklayıcısının içeriğini değiştirmez. Burada, 286 adresleme saklayıcısını kaynak operandı CX'den almak ve AX hedef saklayıcısının içine yükler.

"Kesin adresleme" size 8 veya 16-bit sabit değeri kaynak operand gibi belirtmenize izin verir. Bu sabit talimat içindedir. (assembler'ın koymuş olduğu) ve tutucu ya da bellek erişiminden daha iyidir.

Örneğin:

MOV CX, 500

Ondalık değer olan 500'ü CX tutucusuna yükler ve MOV CL, -30

-30 değerini CL tutucusuna yükler.

Yakın operand zaten EQU direktifiyle tanımlanmış bir simge olabilir.Bunun gibi bir form geçerlidir.

```
K    EQU  1024
     -----
     -----
     MOV  CX,K
```

Sorunlardan sakınmak için, 8 bit 127 (7FH)'tan 128 (80H)'a kadar ve 16-bit 32767 (7FFFH)'tan 32768 (8000H)'a kadar. İşaretli numaralandırır. Eğer bunlar işaretsizse en yüksek 8 ve 16 bit değerler.sırasıyla 255 (0FFH) ve 65535 (0FFFFH)'tır.

Kesin Değerler İşaretli Uzatılmıştır:

Assembler her zaman hedefin içindeki kesin değerleri "işaretli uzatır". Bu bütün hedefin 8 veya 16 bitlerine kaynağın en belirgin bit'ini kopyalar.

Örneğin: MOV CX, 500 ; girerseniz assember kaynak değerini (ondalık 500) 10-bit ikilik modelde 0111110100 gibi görür. Bu değer 16-bit hedef saklayıcıya yüklendiğinde (CX), bu modeli "işaretli" bit değerlerinin (0) sekiz kopyasını başa getirerek uzatır. Böylelikle CX, sonunda

0000000111110100 ikilik değerini içerir. İkinci örnekte; 286: 8 bit ikilik modelini –30 (11100010) için CL'nin içine yüklüyor.

Bellek Adresleme Modları:

Erişim Belleği 286'nın Yürütme Birimi (Execution Unit (EU)) ve Yol birimi (Bus Un it (BU)) tarafından ek bir çaba gerektirir. EU biriminin bir bellek operandı okuması ya da yazması gerektiğinde EU ofset değerini BU'ya verir. BU, bu ofseti bir 20-bit fiziksel adres üretmek için (dört "0" ekleyerek) segment tutucusunun içeriğine ekler, sonra bu adresi operanda erişmek için kullanır.

Efektif Adres:

Ofset Yürütme Biriminin hesapladığı, bellek operandı için çağrılan, efektif adrestir (EA). EA segmentin başlangıcından operandın yerine kadar olan byte'ların uzaklığıdır. 16-bit işaretsiz değerin oluşu, EA segmentin içinde nerede yanlış olduğuna başvurabilir. Bu 65,535 (64K) byte üzerinde segmentin başlangıcını geçmiştir.

Zamanın miktarı Yürütme Birimi hesaplamak için aldığı EA talimatını yürütmek için almasında uzunluğun belirlenmesi ilk etkendir. Kullandığınız adresleme kipine bağlı olması, EA'nın bazı şeyleri gerektirebilmesi biraz basitçe talimatın içinden bir ayrıma gidip getirmesiyle (swap) oluşmakta. Sonra tekrar; ayırma ekleme, taban tutucusu ve bir dizin tutucu gibi bazı çok uzun hesaplamalar gerektirebilir.

Yürütme Zamanına aldırmamak programınızda kritik bir durumdur. Adresleme mov tanımından sonra, BU, bu zaman etkenlerini takdir etmelisiniz.

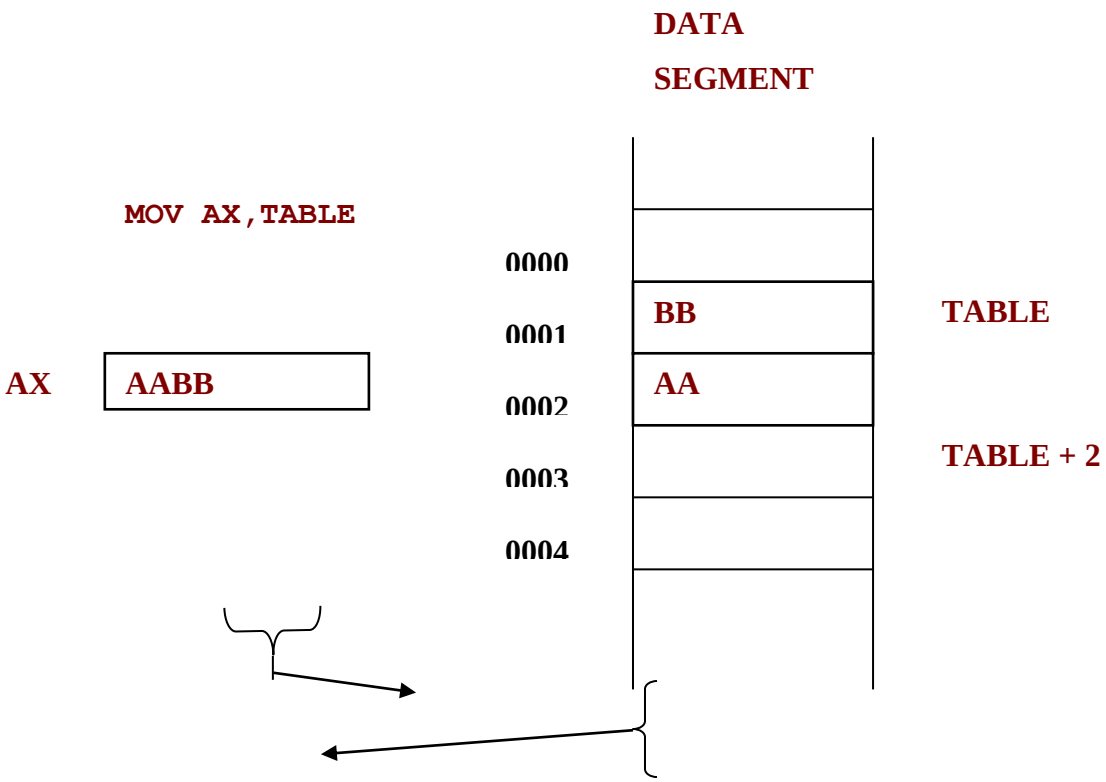
2.2.2.Doğrudan Adresleme

Doğrudan adreslemede, EA kesin veri değerlerinin direktiflerde bulunduğu, direktiflerin içinde bulunur. 286 operand'ın fiziksel adresini üretmek için EA'yı (değişmiş) Data Segment saklayıcısına ekler.

Doğrudan adresleme operandı genellikle etikettir. Örneğin direktif:

MOV AX, TABLE şeklindeyse

TABLE bellek bölgesinin içeriğini AX tutucusuna yükler. Şekil 3-1 talimatın nasıl çalıştığını gösteriyor.



Şekil.2.6. Sayfa 108 Doğrudan Adresleme

Not: 286 veriyi siz hariç tutmak isterseniz zıt düzen içinde bellekte depolar. Bu düşük düzen byte'ından sonra yüksek-düzen byte'ını koyar. Verinin yüksek (en belirgin) parçası, en yüksek bellek adresinin içindedir.

2.2.3.Saklayıcı Dolaylı Adresleme

Saklayıcı Dolaylı adreslemeyle, operandın etkin adresi ya Bx saklayıcı tabanı, ya BP taban işaretçisi (ya da bir dizin tutucusu (SI ya da DI)) içindedir. Dolaylı saklayıcı operandını saklayıcı operandından ayırmak için köşeli parantezle kapatmalısınız. Örneğin:

MOV AX, [BX]

bu talimat BX' le adreslenmiş bellek bölgesinin içeriğini AX saklayıcısının içine yükler. Şekil 3-2 bu örneği resimliyor.

BX'in içine bellek adresine OFFSET öneki girebilmek için bir ofset koymanın bir yolu vardır.

Örneğin: AX içinde TABLE bölgesinde bir sözcük yüklemek için, bu diziyi kullanın.

MOV BX, OFFSET TABLE

MOV AX, [BX]

Bu iki talimatla aynı işi yapmak için

MOV AX, TABLE

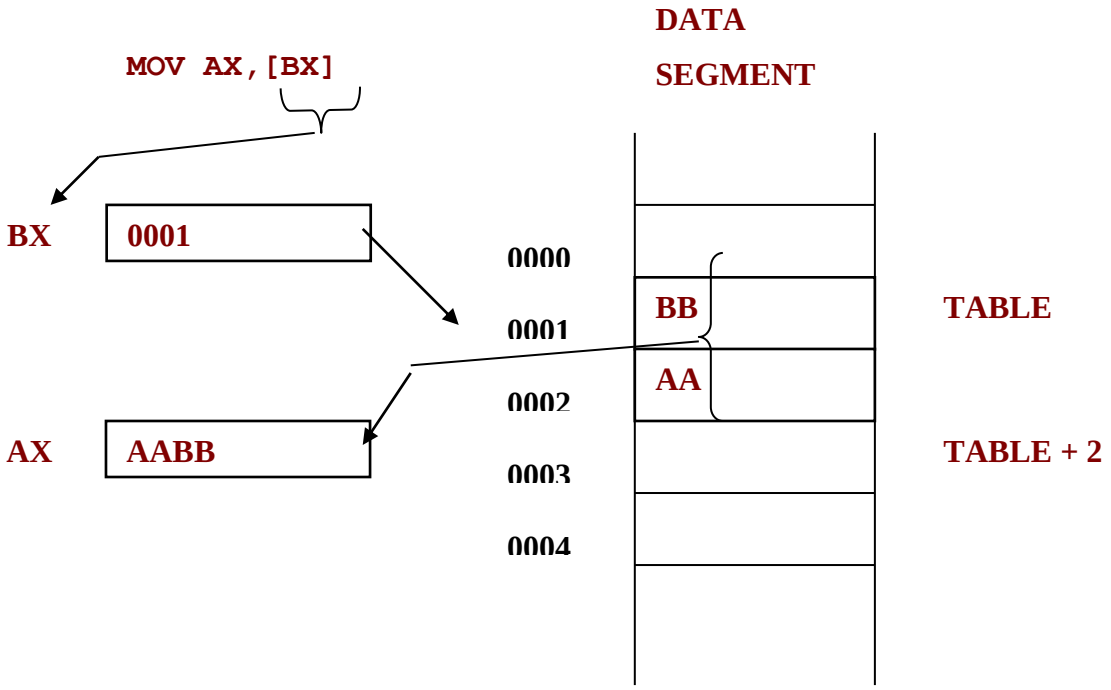
BX'in önceki önceki içeriğini yok etmesi hariç tutulur. Eğer bir yalnız bellek bölgesine (burada TABLE'ın içeriği) erişmek istiyorsanız bu tek talimat daha duyarlı yaklaşım gösterir. Bununla birlikte, bazı taban adresleriyle başlayan, birkaç bölgeye ulaşmak için saklayıcının içinde etkin adres bulunması daha iyidir. Neden böyle? Çünkü her zaman tutucunun içeriklerini getir-götür adressiz üretebilirsiniz.

2.2.4.Göreceli Taban Adresleme

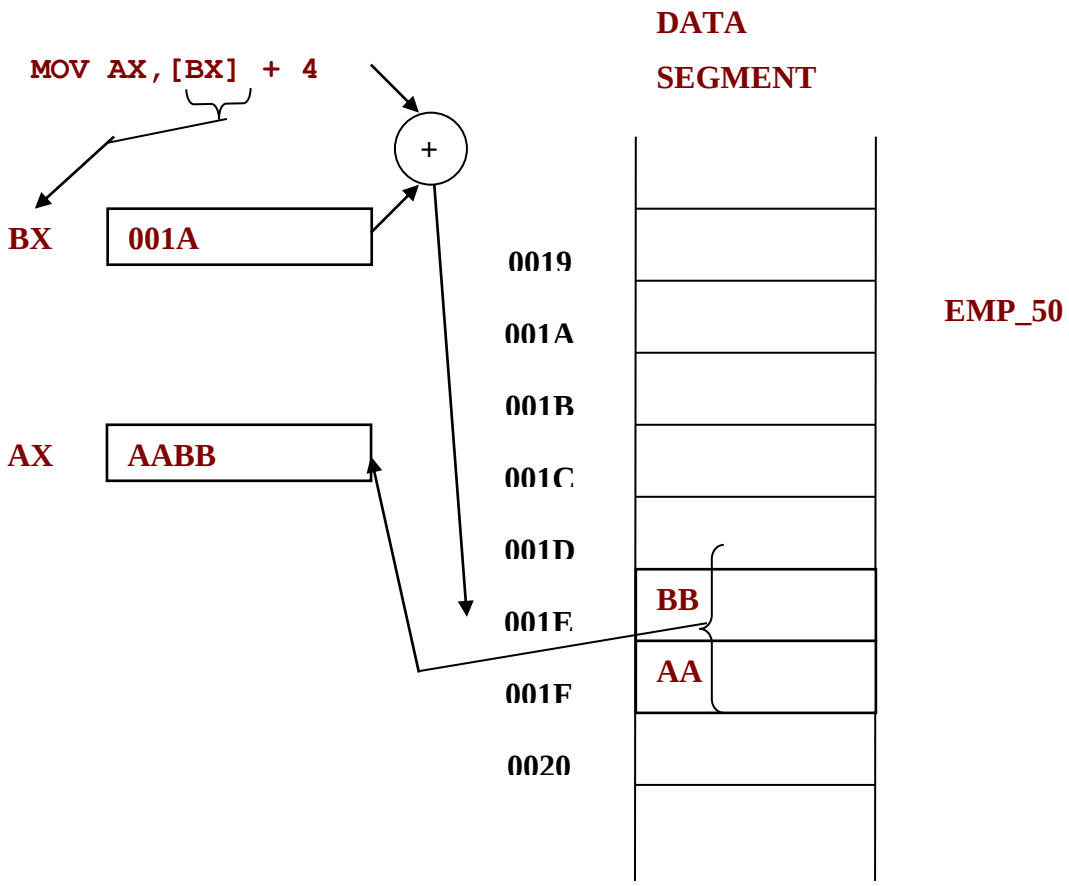
Göreceli taban adreslemeyle assembler etkin adresini BX yada BP tutucusuna bir ayırma değeri ekleyerek hesaplar. BX formu belleğin farklı bölümünde yerleşmiş veri yapısına erişmenin uygun yolunu verir. Bunu yapmak için, yapının taban adresini saklayıcı tabanının içine koyun tabanda ayrılması ile yapının öğelerine başvurun. Bundan sonra, taban saklayıcıda basit değişiklikler yaparak, yapının içinde değişik kayıtlara ulaşabilirsiniz.

Örneğin: Diskten bazı personel kayıtlarını okuduğunuzu varsayın, bulunan her kayıt işçinin kimlik numarasını, bölüm numarasını, bölge numarasını, yaşını, maaşını vb içerir. Eğer böyle numarası kaydın 5. ve 6. byte'larında saklıysa, kaydın adresi BX'le başlıyorsa, direktif böyledir.

MOV AX, [BX] + 4



Şekil .2.7. Dolaylı Register Adresleme



Şekil.2.8. Göreceli Taban Adresleme

Bu talimat işçinin bölge numarasını AX'in içine yükler. (Ayırmada 4 değeri 5'ten daha iyidir, çünkü ilk byte "0" 'dır) Şekil 3-3 bu örneği sergiler.

Macro Assembler göreceli taban operandı belirlemede üç değişik yola izin verir.

Eşdeğer 3 direktif:

```
MOV   AX, [BP]+4    ;  Bu standart formdur, fakat kullanabilirsiniz.
MOV   AX, 4[BP]     ;  ilk ayırmayı koyar
MOV   AX, [BP+4]    ;  ya da köşeli parantezle birlikte
```

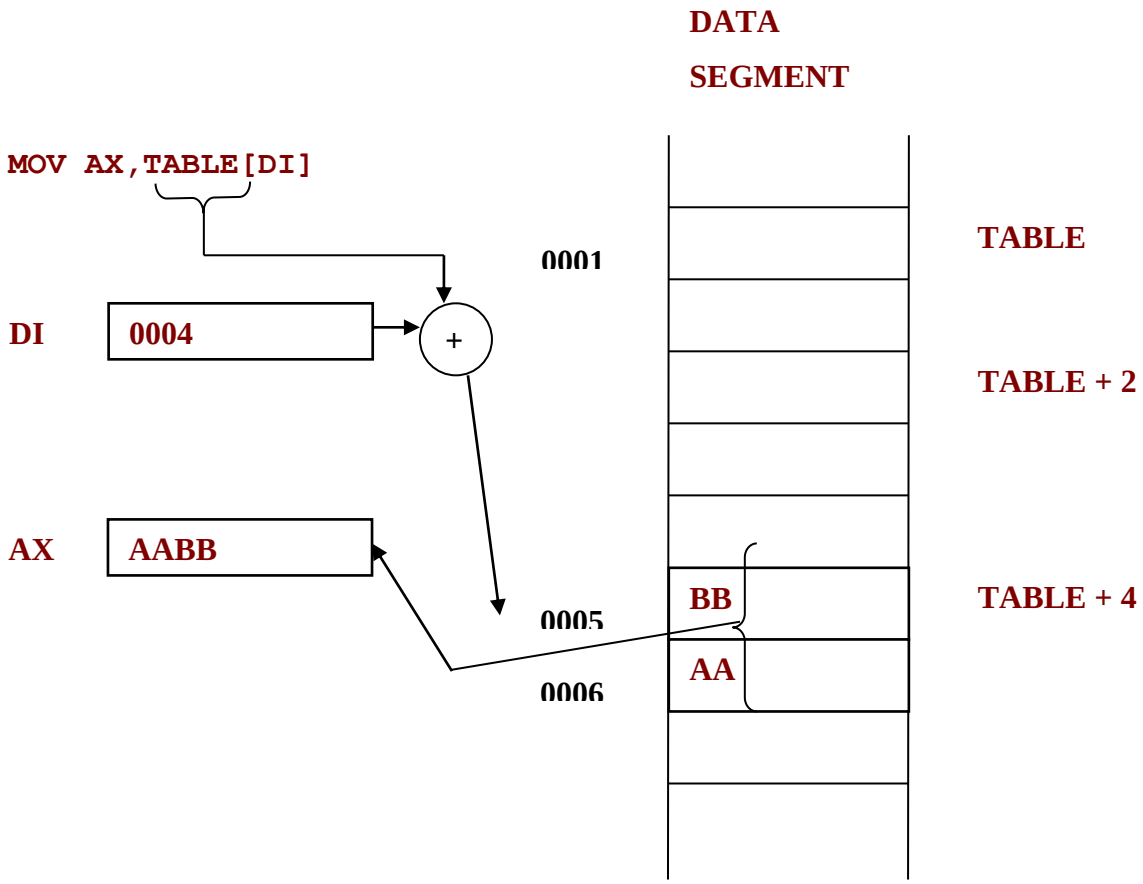
2.2.5.Doğrudan Dizli Adresleme

Doğrudan dizili adreslemeyle, etkin adres ayırma ve dizin saklayıcısının toplamıdır, DI ya da SI'dan biri. Bu adresleme tipi bir tablonun içindeki öğelere erişim için uygundur. Ayırma noktaları tablonun başlangıcına ve dizin saklayıcı noktaları öğenin içinedir.

Örneğin : B_TABLE isminde bir byte tablomuz var. Direktif dizisi şöyledir.

```
MOV     DI,  2
MOV     AL,  B_TABLE [DI]
```

Direktif AL saklayıcısının içine 3. öğeyi yükler.



Şekil.2.8. Doğrudan Dizili Adresleme

Bir sözcük tablosu içinde, öğeler bağımsız 2 byte'tır. Böylece dizin değeri olarak çift öğe numaranız olur. TABLE isimli Word tablosuyla talimat dizisi şöyledir.

```
MOV   DI,  4
MOV   AX, TABLE  [DI]
```

Bu direktif dizisi üçüncü öğeyi AX saklayıcısının içine yükler. Şekil 3-4 bu örneği sergiler.

2.2.6.Taban Dizili Adresleme

Taban dizili adreslemede, EA bir taban saklayıcı, bir index saklayıcı ve (seçimlik olarak) ayırmanın toplamıdır. Çünkü iki ayrı ofset kabul eder. Bu mod iki boyutlu dizilere erişimde yararlıdır. Burada ayırma ve dizin saklayıcısı sıra ve sütun ofsetini sağlarken taban saklayıcı dizinin başlangıç adresini tutar.

Örneğin: bir kimyasal işlem fabrikasında bilgisayarınızın monitöründe altı basınç sübabı var varsayalım. Bu her yarım saatte sübab ayarını okur ve bunları belleğe kaydeder. Birinci hafta içinde bu okuma formu her alt öğe için 336 bloktur. (7 günde her gün için 48 okuma). 2.016 veri değeri toplamıdır.

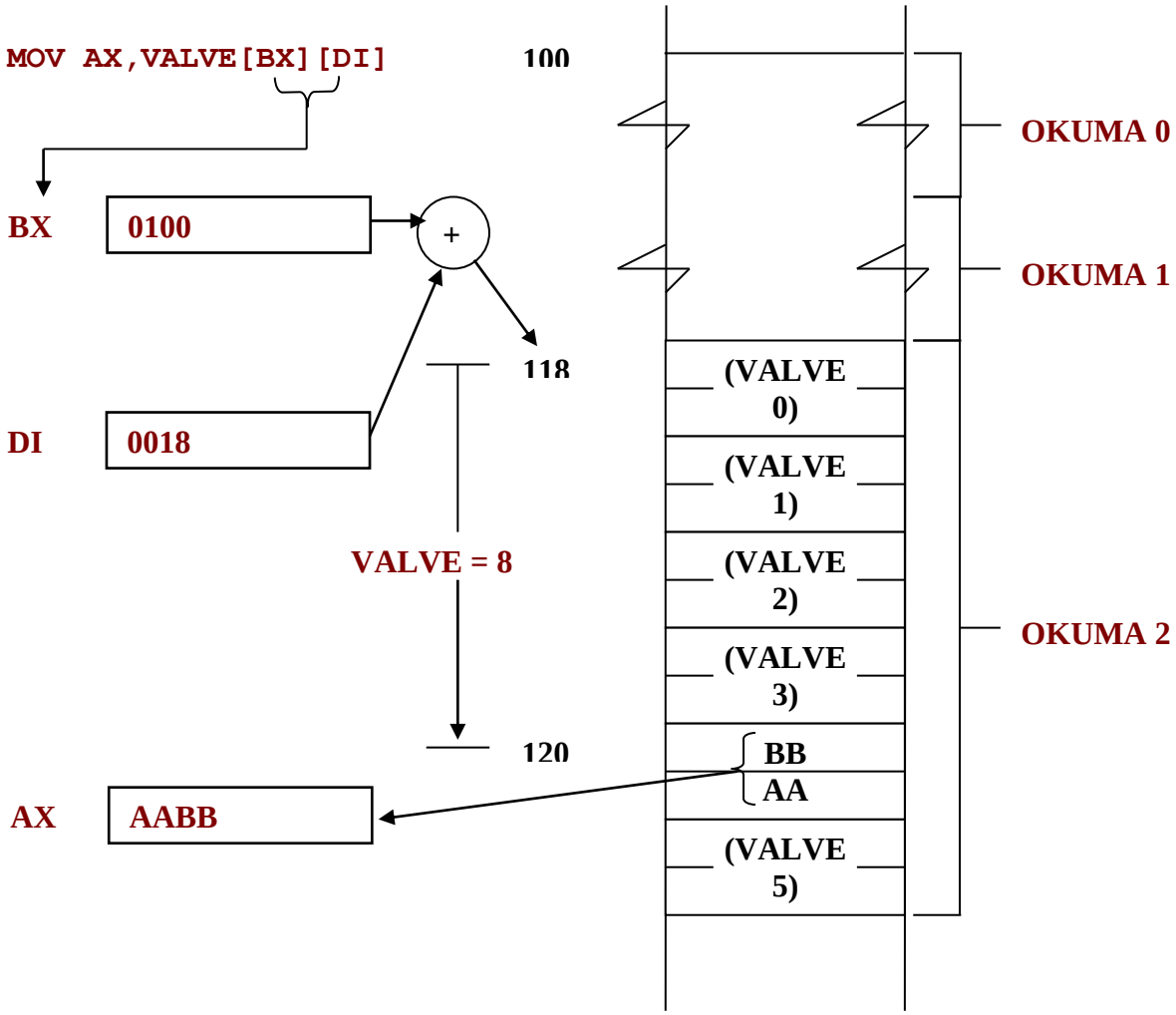
Eğer dizinin başlangıç adresi BX ise, blok ayırması (okuma numarası zamanları 12) DI'nın içindedir; ve sübab numarası ayrımı VALUE değişkeni ile tanımlanmıştır. Bu direktifi kullanabiliriz.

```
MOV   AX,  VALUE [BX] [DI]
```

Bu direktif bazı seçilmiş basınç sübaplarını AX içinden okur. Şekil 3-5'te bu talimat data segmentin 100h ofset adresine sahip bir diziden subab-4 (2.okuma)'ün üç okumasını özetler.

Burada taban dizili adresleme operandları için bazı diğer geçerli biçimleri verilmiştir.

```
MOV   AX,  [BX+2+DI]  ;  bütün üç terimi bir köşeli paranteze koyabilirsiniz.
MOV   AX,  [DI+BX+2]  ;  herhangi düzende
MOV   AX,  [BX+2] [DI]  ;  ya da ayırmayı birleştirebilirsiniz
MOV   AX,  [BX] [DI+2]  ;  başka saklayıcıya
```



Şekil.2.9. Bir iki boyutlu diziden veri değerini çıkarır.