Microprocessor

Assembly Addressing Modes

Dr. Cahit Karakuş
Esenyurt Üniversitesi

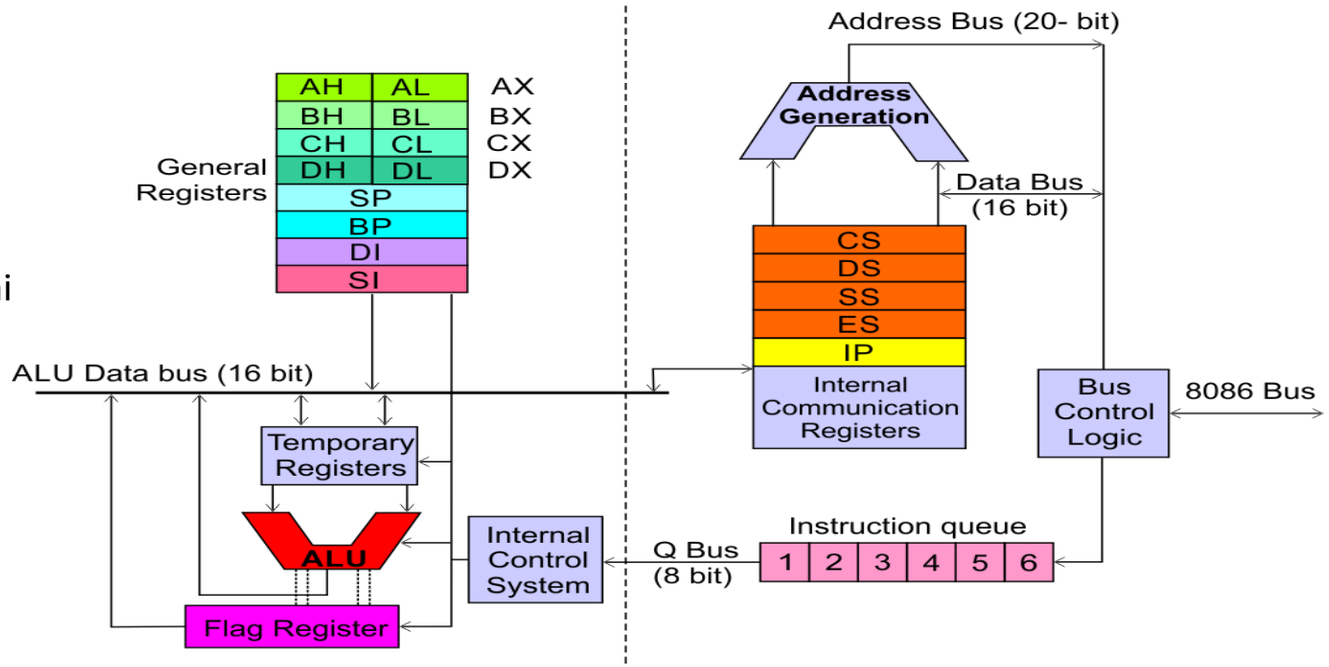
CPU Summary

CPU İç Mimarisi - Registers

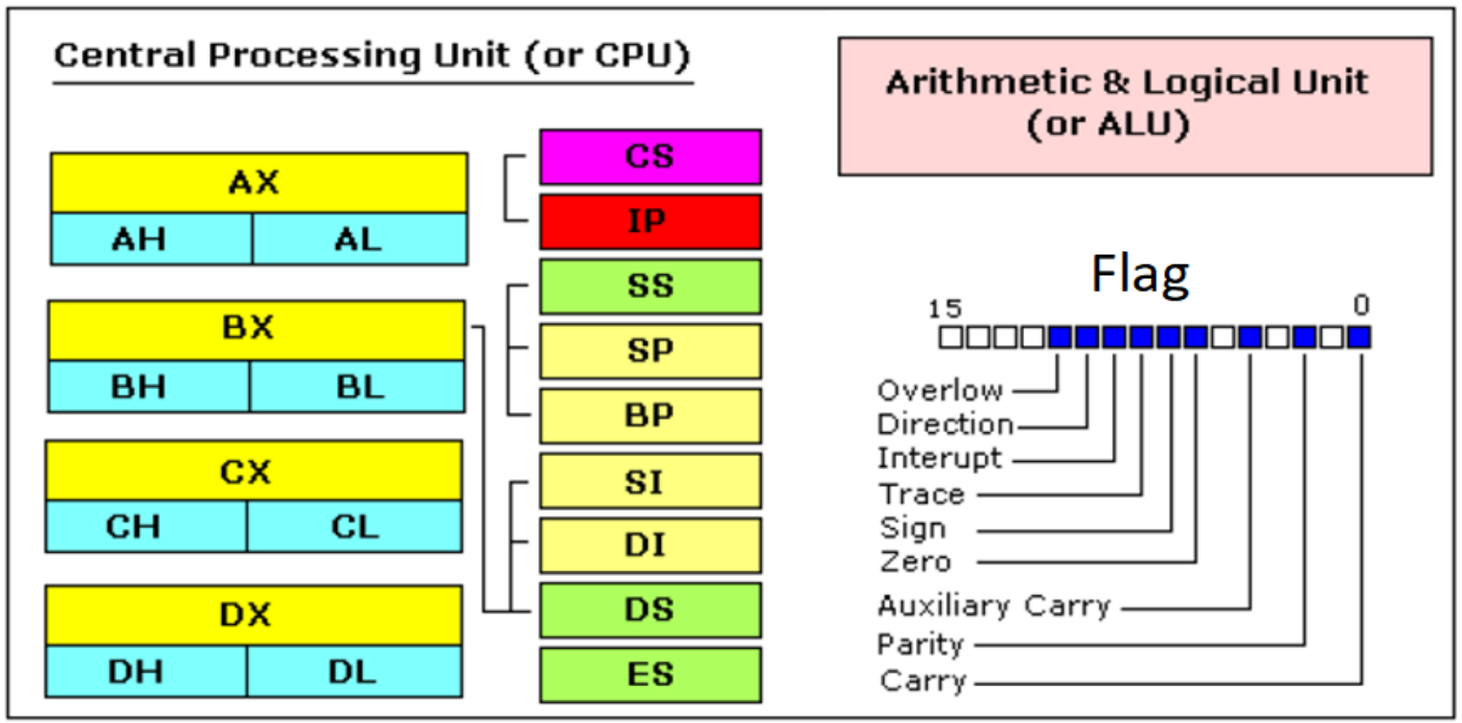
- CPU'nun iç mimarisinin ana bileşenleri (EU, BIU): ALU, Kontrol Birimi, Komut Kuyruğu ve Register'lar.

Registers (Özel Amaçlı Geçici Saklayıcılar): Bellek hiyerarşisinde 1 numaradır. Yüksek hızlı veri işlenir ve transfer edilir. Geçici depolama alanıdır. CPU içinde bulunur ve CPU'nun ana bileşenleridir. x86 mikroişlemcisinde toplam 14 adet 16bitlik register bulunmaktadır.

- Data Register: Aritmetik, mantık, karşılaştırma, ve veri transfer işlemleri bu register'ların üzerinde yapılır. AX (AH-AL), BX(BH-BL), CX(CH-CL), DX (DH_DL).
- AH, AL, BH, BL, CH, CL, DH, DL 8 bit Özel amaçlı Geçici Saklayıcılardır
- Segmet Register: CS, DS, SS, ES: belleklerin başlangıç adresleri burdadır.
- Pointer (SP, BP): Başlangıç adresi SS'de bulunan belleğin gözlerini işaret eder.
- Index (SI, DI, BX): Başlangıç adresi DS, ES'de bulanan belleklerin gözlerini işaret eder.
- Flag: İşlem yapılırken değişen durumların göstergesidir. Kesme, pozitif/negatif, sonuç sıfır, bit sayıları, …
- ALU: Aritmetik ve mantıksal işlem birimidir.
- IP, Başlangıç adresi CS'de bulunan programın işlenecek bir sonraki komutunun yerini işaret eder.



Veri Belleğini Adresleme



- Çoğu Aritmetik ve Mantık Komutları, işlemci durum kaydını (veya Bayrakları) etkiler.
- CS: Program kodlarının saklandığı belleğin başlangıç adresini kayıt edildiği register.
- IP, CS ile birlikte bir sonraki kodun fiziksel adresini gösterir.
- DS, ES, SS ise veri velleğin başlangıç adresi gösterir.
- SI, DI, BX indis göstergesidir, DS ve ES ile birlikte verinin kayıt edildiği belleğin fiziksel adresini gösterir.
- BP, SP indis göstergesidir., SS ile birlikte verinin kayıt edildiği belleğin fiziksel adresini gösterir.

Flags

- **Elde (Taşıma) Bayrağı (CF: Carry Flag)** - işaretsiz bir taşma olduğunda bu bayrak 1'e ayarlanır. Örneğin bayt 255 + 1 eklediğinizde (sonuç 0...255 aralığında değildir). Taşma olduğundan bu bayrak 1'e ayarlanır. Taşma olmadığında bu bayrak 0'a ayarlanır.
- **Sıfır Bayrağı (ZF: Zero Flag)** - sonuç sıfır olduğunda 1'e ayarlanır. Sıfır olamayan sonuçlar için bu bayrak 0'a ayarlanır.
- **İşaret Bayrağı (SF: Sign Flag)** - sonuç negatif olduğunda 1'e ayarlayın. Sonuç pozitif olduğunda 0'a ayarlanır. Aslında bu bayrak en anlamlı bitin değerini alır.
- **Taşma Bayrağı (OF: Overflow Flag) -** işaretli bir taşma olduğunda 1'e ayarlanır. Örneğin, 100 + 50 bayt eklediğinizde (sonuç -128...127 aralığında değildir).
- **Eşlik Bayrağı (PF: Parity Flag)** - bu bayrak, sonuçta bir bit sayısı çift olduğunda 1'e ve tek sayıda bit olduğunda 0'a ayarlanır. Sonuç bir kelime olsa bile sadece 8 düşük bit analiz edilir!
- **Yardımcı Bayrak (AF: Auxiliary Flag) -** (4 bit) için işaretsiz bir taşma olduğunda 1'e ayarlanır.
- **Kesme etkinleştirme İşareti (IF: Interrupt enable Flag)** - bu işaret 1 olarak ayarlandığında, CPU harici cihazlardan gelen kesmelere tepki verir.
- **Yön Bayrağı (DF: Direction Flag) -** bu bayrak, veri zincirlerini işlemek için bazı talimatlar tarafından kullanılır, bu bayrak 0'a ayarlandığında - işleme ileriye doğru yapılır, bu bayrak 1'e ayarlandığında işleme geriye doğru yapılır.

Special Purpose Registers (IP / Flags)

- IP - the instruction pointer.
- flags register - determines the current state of the microprocessor.
- IP register always works together with CS segment register and it points to currently executing instruction.

- Flags register is modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program.
- generally you cannot access these registers directly, the way you can access AX and other general registers, but it is possible to change values of system registers using some tricks that you will learn a little bit later.

Kesme

- Kesme geldiğine
- AX, BX, CX, DX, SI, DI, SP, BP, CS, SS, DS, ES, IP, Flags register'ların içeriklere belleğe kaydedilir.
- Kesme isteğine ilişkin alt programını çalıştırır.
- İşlev tamamlandığında, belleğe kayıt edilen değerler yeniden register'lara aktarılır.
- CPU kaldığı yerden çalışmasına devam eder.

Data Types

Bit - Binary

- Her türlü semboller ikili (binary) sayı sistemi (0/1) ile gösterilir.
- Sembol: Metin, Rakamlar, Resim, Video, Ses, …
- Sembol: Klavyedeki tüm tuşların gösterimi semboldür ve 8bit ile temsil edilir. Sembol m=8bit, toplam sembol sayısı=2^m=256 dır.
- Klavyedeki her bir tuşun 8 bit ile gösterimi ASCII standardı ile verilmektedir.
- Hex=4bit ile temsil edilir: 0,1,2, … , 9, A, B,C,D,E,F. Toplam, 2^4= 16 sembol 4 bit ile temsil edilir.
- (AC)h=(1010 1100)b= 2^7+2^5+2^3+2^2=128+32+8+4=(172)d
- Byte=8bit datayı temsil eder. Indis: 7,6,5, …, 2,1,0 dır.
- Bit toplanması: 0+0=0, 0+1=1, 1+0=1, 1+1=10, 1+1+1=10+1=11

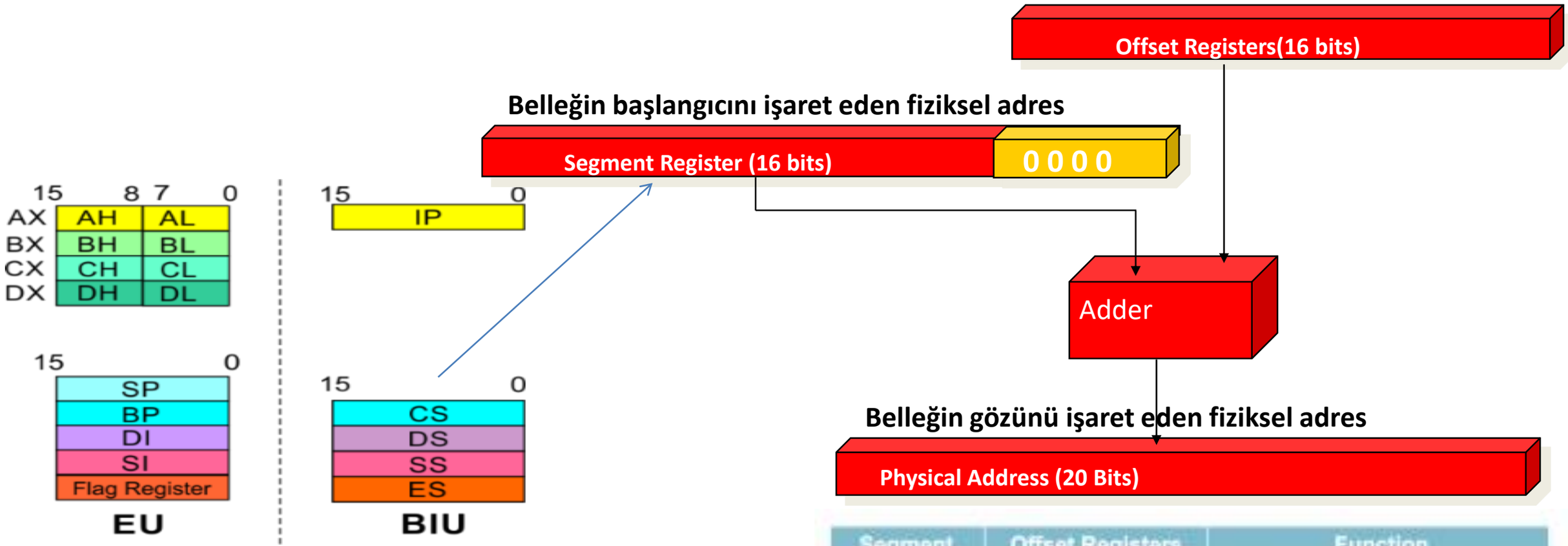
Genel Kurallar

- Mov Varış, Kaynak; Kaynak → Varış
- Kaynak ve Varş saklayıcıları aynı boyutta olmak zorundadır
- Mov AX, BL İllegal; boyut farklı Ax: 16 bit BL: 8 bit
- Mov AL, BL Legal ; Al: 8Bit, BL: 8bitr
- Mov AX, CX Legal
- Mov DS, AX Lega; DS'in başlangıç adsresi yüklenir.
- Mov Ax, 0BBCCh Legal
- Mov AL, 0BBCCh İllegal
- Mov DS, 0BBCCh İllegal, segment register'lara doğrudan sayısal değer yüklenemez.
- Mov i,j i,j: değişkenler (Katsayı ya da sabitler) Varış noktası değişken olamaz.
- Mov AL, j Legal

Komut işlemlerinde, Register'ların kayıt alanı uzunlukları aynı olmalıdır. AX, BX, CX, DX data register'ları 16 bit kayıt uzunluğunz sahiptir. Oysa AH, AL, BH, BL, CH, CL, DH, DL 8 bit kayıt uznuluğuna sahiptir.

Bellek Erişimde Fiziksel Adres

Addressing Modes Memory Access



Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

- Data Segment Register'ın Offset Register'ları nedir?
- Code Segment Register'ın Offset Register'ları nedir?
- Stack Segment Register'ın Offset Register'ları nedir?
- Extra Segment Register'ın Offset Register'ları nedir?

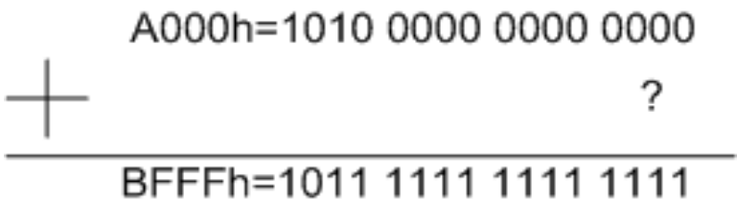
Pozitif veya negatif tam sayılar

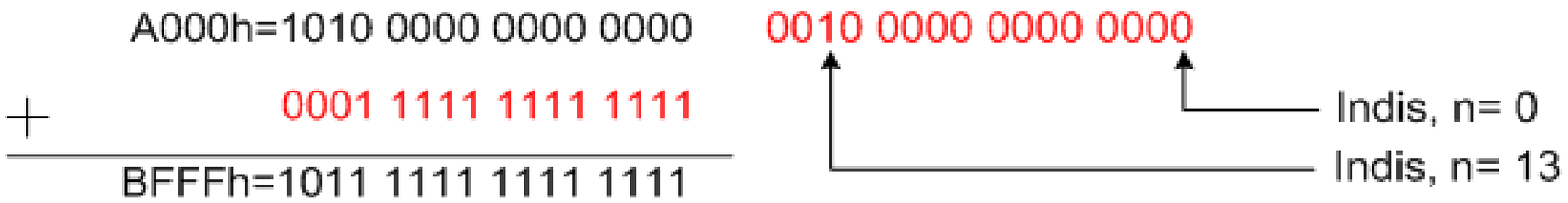
Bellek İndisi ve Kapasitesi

- Soru: Bellek fiziksel başlangıç adresi (A000)h, bellek fiziksel bitiş adresi (A0FF)h ise bellek boyutu nedir?
- Bellek kapasitesi= Bellek fiziksel bitiş adresi - Bellek fiziksel başlangıç adresi +1
- Bellek kapasitesi= (A0FF)h –(A000)h +1 =(FF)h +1
- Bellek kapasitesi: 100h=(0001 0000 0000)=2^8=256 byte
- Bellek adres bus indisi=A7,A6, … , A1, A0

Örnek-3: Bellek Kapasitesi

- Bellek Kapasitesi= Bellek Bitiş Adresi – Bellek Başlangıç Adresi +1 (+1 konmasının nedeni bellek başlangıç adresinin 0'dan başlamasıdır.)
- Örnek: Bellek Başlangıç Adresi=(A000)h, Bellek Bitiş Adresi=(BFFF)h ise Bellek kapasitesini hesaplayınız.
- Bellek kapasitesi = BFFFh – A000h=1FFFh +1
- Bellek kapasitesi=0001 1111 1111 1111 + 1=0010 0000 0000 0000
- Bellek Kapasitesi aralığı: (0000)h – (1FFF)h
- Bellek Kapasitesi=2^13 Byte=8Kbyte
- Adres hat sayısı=13
- Adres hatları indisleme: A12, A11, … , A1, A0





Örnek-3: Bellek Kapasitesi

- Bellek Kapasitesi= Bellek Bitiş Adresi – Bellek Başlangıç Adresi +1 (+1 konmasının nedeni bellek başlangıç adresinin 0'dan başlamasıdır.)
- Örnek: Belleği başlangıcı için CPU'dan çıkacak fiziksel başlangıç adresi=(A000)h, ve Bellek Kapasitesi=32KiloByte ise Belleğin bitişi için CPU'dan çıkacak bitiş adresini bulunuz.
- 32Kbyte = 2^5*2^10=2^15byte
- Adres bus hat sayısı=15
- İndis=A14, A13, A12, … , A1, A0
- Bellek başlangıç adresi= (0000 0000 0000 0000)b
- Bellek bitiş adresi=(0111 1111 1111 1111)b
- Belleği başlangıcı için CPU'dan çıkacak fiziksel başlangıç adresi= (1010 0000 0000 0000)b
- Belleğin bitişi için CPU'dan çıkacak bitiş adresi= Belleği başlangıcı için CPU'dan çıkacak fiziksel başlangıç adresi + Bellek bitiş adresi=(11FFF)h
- (1010 0000 0000 0000)b + (0111 1111 1111 1111)b = (1 0001 1111 1111 1111)b= (11FFF)h

Addressing Modes

Addressing Modes

- Bir programın her komutu bir veri üzerinde çalışmak zorundadır.
- Kaynak işlenenin bir talimatta belirtilmesinin farklı yolları, adresleme modları olarak bilinir.

- Group I : Addressing modes for register and immediate data
- Group II : Addressing modes for memory data
- Group III : Addressing modes for I/O ports
- Group IV : Relative and Implied Addressing mode

MOV Kodu

- **Mov Destination (Hedef), Source (Kaynak)**
- source -> Destination
- Kaynak lokasyonundaki veriyi hedef lokasyonuna transfer eder.
- Kaynak,
  - Anlık bir değer (Değişkenler: Katsayılar, Sayılar)
  - Genel amaçlı register (Ax, Bx, Cx, Dx; AL, AH, BL, BH, CL, CH, DL, DH)
  - Bellek gözleri: Segment register'larındaki RAM bellek başlangıç adresinden (DS,SS,ES) gösterge veya indeks register'lar ile belirlenmiş bellek gözündeki veriler.
- Hedef
  - register, genel amaçlı bir register
  - bellek gözleri.
- Kaynak ve hedefteki veri boyutları aynı olmaldır. Byte (8bit), word (16bit), dword(32bit).

Veri Transfer Komutları (Bellek)

Komutlar: MOV, LEA, XCHG

MOV reg2, reg1	(reg2) ← (reg1)
MOV mem, reg1	(mem) ← (reg1)
MOV reg2, mem	(reg2) ← (mem)
MOV reg, data	(reg) ← data
MOV mem, data	(mem) ← data
XCHG reg2, reg1	(reg2) ↔ (reg1)
XCHG mem, reg1	(mem) ↔ (reg1)

Dikkat: Mov komutu ile bellekten belleğe veri transferi yapılamaz.

MOV

- **MOV Operand1, Operand2**
- Algorithm: operand1 = operand2

Opeand1, Operand2:

- REG, memory
- memory, REG
- REG, REG
- memory, immediate
- REG, immediate
- SREG, memory
- memory, SREG
- REG, SREG
- SREG, REG

Not: The MOV instruction cannot set the value of the CS and IP registers. The MOV instruction copy value of one segment register to another. The segment register should copy to general register first. The MOV instruction copy immediate value to segment register but, firstly immediate value should copy to general register first).

- **Example:**

```
ORG 100h
MOV AX, 0B800h ; set AX = B800h (VGA memory).
MOV DS, AX ; copy value of AX to DS.
MOV CL, 'A' ; CL = 41h (ASCII code).
MOV CH, 01011111b ; CL = color attribute.
MOV BX, 15Eh ; BX = position on screen.
MOV [BX], CX ; w.[0B800h:015Eh] = CX.
RET ; returns to operating system.
```

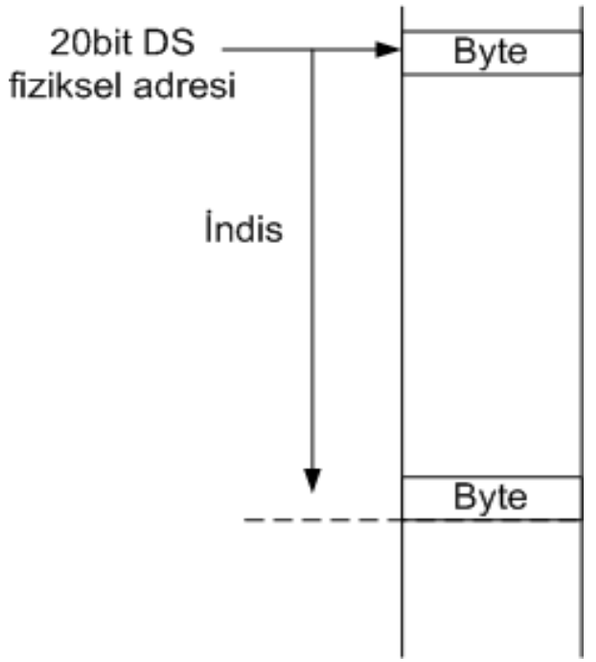
Bellek Gözüne Erişim
Fiziksel Adres

Calculation of Physical Address

- CPU makes a calculation of physical address by multiplying the segment register by 10h and adding general purpose register to it (1230h * 10h + 45h = 12345h). Burada Segment adres: 1230h, Indis register: 45h
- Segment Registers: CS, DS, SS, ES
- Indis Registers: CS (IP), DS(SI, DI, BX), SS(SP, BP), ES(SI, DI, BX).
- the address formed with 2 registers is called an **effective address**. by default **BX, SI** and **DI** registers work with **DS** segment register; **BP** and **SP** work with **SS** segment register.
- Other general purpose registers cannot form an effective address! also, although **BX** can form an effective address, **BH** and **BL** can not.

Örnek-2: Bellek Gözünün Fiziksel Başlangıç Adresi

Soru: fiziksel adresi bc150h olan bellek gözüne bl register'ın içeriği yazılacaktır.
İndis: 150h ve si register'ında kayıtlı olacaktır. Segment register es olacaktır.
Assembly yazılımını yazınız.



- Belleklerin fiziksel başlangıç adresleri 20bit
- Segment register'lar 16 bittir. Segment register'ların içeriği temel belleklerin başlangıç adresini gösterir.
- Belleklerin fiziksel başlangıç adresleri =Segment register * 10h
  - Es belleğin fiziksel başlangıç adresi= bellek gözünün fiziksel adresi – indis=bc150h -150h=bc000
  - es: bc00h, çünkü extra segment register 16 bit'tir.
  - İndis, si: 150h
  - Bellek gözünün fiziksel adresi=bc000h + 150h = bc150h

```
Mov ax, bc00h
Mov es, ax
Mov si, 150h
mov es:[si], bl
```

[]: Köşeli parantez bellek gözünün fiziksel adresini tanımlar.
Önünde hiçbir şey yok ise DS tanımlıdır.
Köşeli parantez segment register'ın içeriğinin fiziksel adres karşılığıdır. Yani 16bitlik segment register İçeriğinin sağ tarafına 4 bit 0 eklenerek 20 bit'e dönüşümüdür.

Bellek Erişimi

- To access memory we can use these four registers: BX, SI, DI, BP. combining these registers inside [ ] symbols, we can get different memory locations. These combinations are supported addressing modes.
- d8 - stays for 8 bit signed immediate displacement (for example: 22, 55h, -1, etc...)
- d16 - stays for 16 bit signed immediate displacement (for example: 300, 5517h, -259, etc...).

[BX + SI] [BX + DI] [BP + SI] [BP + DI]	[SI] [DI] d16 (variable offset only) [BX]	[BX + SI + d8] [BX + DI + d8] [BP + SI + d8] [BP + DI + d8]
[SI + d8] [DI + d8] [BP + d8] [BX + d8]	[BX + SI + d16] [BX + DI + d16] [BP + SI + d16] [BP + DI + d16]	[SI + d16] [DI + d16] [BP + d16] [BX + d16]

Addressing Modes

Addressing modes for 16-bit x86 processors can be summarized by this formula:

$$\left\{\begin{matrix} CS: \\ DS: \\ SS: \\ ES: \end{matrix}\right\} \left[\left\{\begin{matrix} BX \\ BP \end{matrix}\right\}\right] + \left[\left\{\begin{matrix} SI \\ DI \end{matrix}\right\}\right] + [\text{displacement}]$$

Memory Access

- The value in segment register (CS, DS, SS, ES) is called a segment, and the value in purpose register (BX, SI, DI, BP) is called an offset. Displacement can be a değişken value or offset of a variable, or even both. if there are several values, assembler evaluates all values and calculates a single değişken value. Displacement can be inside or outside of the [ ] symbols, assembler generates the same machine code for both ways.

- To access memory we can use these four registers: BX, SI, DI, BP, SP combining these registers inside [ ] symbols, we can get different memory locations. these combinations are supported (addressing modes):

- [SI], [DI], [BX]
- [BX + SI]
- (DS-16bit, sağ tarafa 4bit 0 konur, 20 bit fiziksel adres elde edilir.)
- Bx+ SI gerçek fiziksel adres bulunur.

Memory Access

- When DS contains value b800h and SI contains the value 10h it can be also recorded as b800:10. [SI]=The physical address will be b8000h + 10h = b8010h.

- d8 - stays for 8 bit signed değişken displacement (for example: 22, 55h, -1, etc...)
- d16 - stays for 16 bit signed değişken displacement (for example: 300, 5517h, -259, etc...).
- d16 (variable offset only)

- Displacement is a signed value, so it can be both positive or negative. Generally the compiler takes care about difference between d8 and d16, and generates the required machine code.

- for example, let's assume that DS = 100, BX = 30, SI = 70.
- The following addressing mode: [BX + SI] + 25 is calculated by processor to this physical address: 100 * 16 + 30 + 70 + 25 = 1725.

Memory Access

- [BX + DI], [BP + DI], [SI + d8]
- [DI + d8], [BP + d8], [BX + d8]

- By default DS segment register is used for all modes except those with BP register, for these SS segment register is used. There is an easy way to remember all those possible combinations using this chart:
-

BX	SI	+ disp
BP	DI	

- [BX + SI + d16], [BX + DI + d16], [BP + SI + d16], [BP + DI + d16]
- [BX + SI + d8], [BX + DI + d8], [BP + SI + d8], [BP + DI + d8]
- [SI + d16], [DI + d16], [BP + d16], [BX + d16]

Memory Access

- You can form all valid combinations by taking only one item from each column or skipping the column by not taking anything from it. as you see BX and BP never go together. SI and DI also don't go together. here are an examples of a valid addressing modes: [BX+5] , [BX+SI] , [DI+BX-4]

- if zero is added to a decimal number it is multiplied by 10, however 10h = 16, so if zero is added to a hexadecimal value, it is multiplied by 16, for example: 7h = 7; 70h = (112)d

- In order to say the compiler about data type, these prefixes should be used: byte ptr - for byte. word ptr - for word (two bytes).

- For example: byte ptr [BX] ; byte access. or word ptr [BX] ; word access.
- assembler supports shorter prefixes as well:

- b. - for byte ptr
- w. - for word ptr

- in certain cases the assembler can calculate the data type automatically.

Memory Access

- Stack is used by CALL instruction to keep return address for procedure, RET instruction gets this value from the stack and returns to that offset.
- Quite the same thing happens when INT instruction calls an interrupt, it stores in stack flag register, code segment and offset.
- IRET instruction is used to return from interrupt call.
- We can also use the stack to keep any other data, there are two instructions that work with the stack.

Bellek Gözüne Erişim Adresi

Belleğin başlangıçının fiziksel adresi nasıl hesaplanır?

Belleğin başlangıç adresleri Segment regster'larda kayıt edilmiştir. 16 bit. Oysa fiziksel adres 20bit. Bu nedenle 16bit değerinin sağına 4 bit 0 eklenerek 20 bit'lik fiziksel adres elde edilir.

Örneğin DS değeri (2000)h ise fiziksel adresi nedir?

(2000h)=(0010 0000 0000 0000)b

Fiziksel adres=(0010 0000 0000 0000 0000)b= (20000)h

Bellek gözünün erişim adres ise belleğin başlangıç fiziksel adresi (20bit) ile indeks ya da pointer register'ların içeriği ve var ise tam sayısal değerler toplanarak hesaplanır.

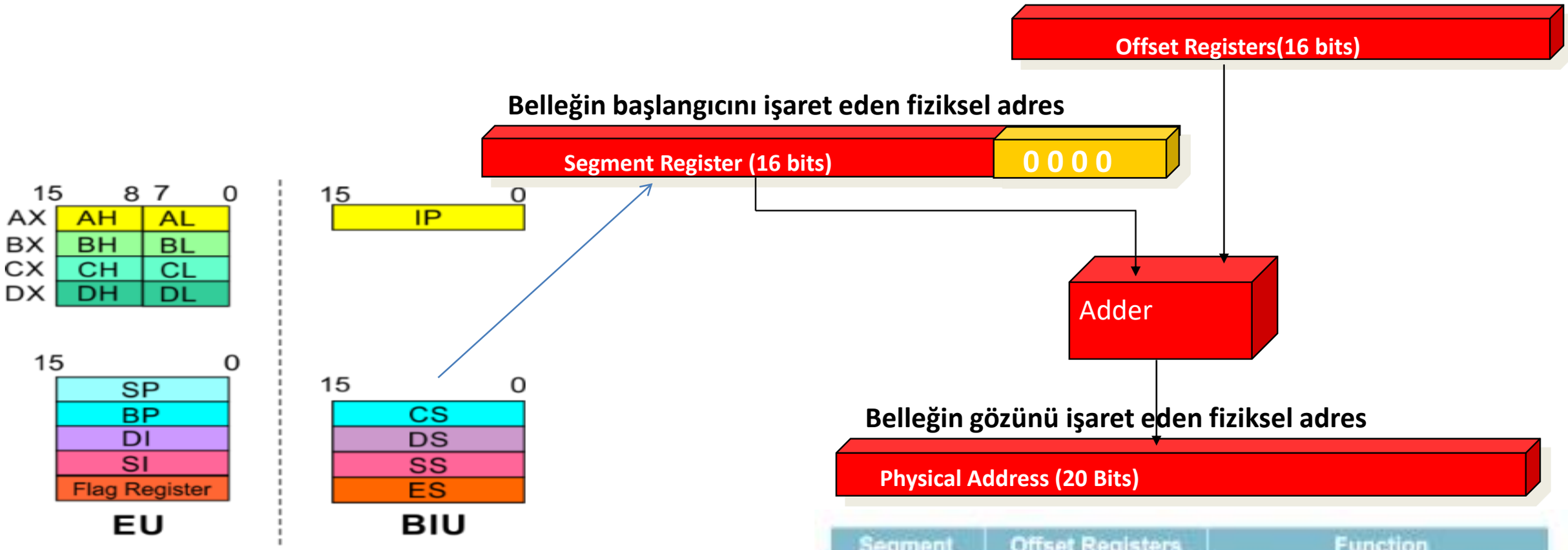
CS → IP

DS, ES → SI, DI, BX

SS → SP, BP

INT Sayılar da eklenebilir.

Addressing Modes Memory Access



Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

- Data Segment Register'ın Offset Register'ları nedir?
- Code Segment Register'ın Offset Register'ları nedir?
- Stack Segment Register'ın Offset Register'ları nedir?
- Extra Segment Register'ın Offset Register'ları nedir?

Örnek

- MOV SI, 1000h ; DS ya da ES register'ları ile ilgili
- MOV AX, 4000h ; AX register'ına 400h değeri yazılır.
- MOV ES, AX ; ES içeriği 4000h olur. ES ile gösterilen belleğin başladığı konumun fiziksel adresi= (40000)h
- MOV AX, ES:[SI] ; 40000 + 1000h=41000h 20 bitlik bellek gözünün fiziksel adresi bulunmuş oldu. MOV komutu ile bu adresten 16 bitlik veri alacak ve AX register'ına kayıt edilecek.

Physical address

Question: What physical address corresponds to DS:(103F)h if DS=94D0h

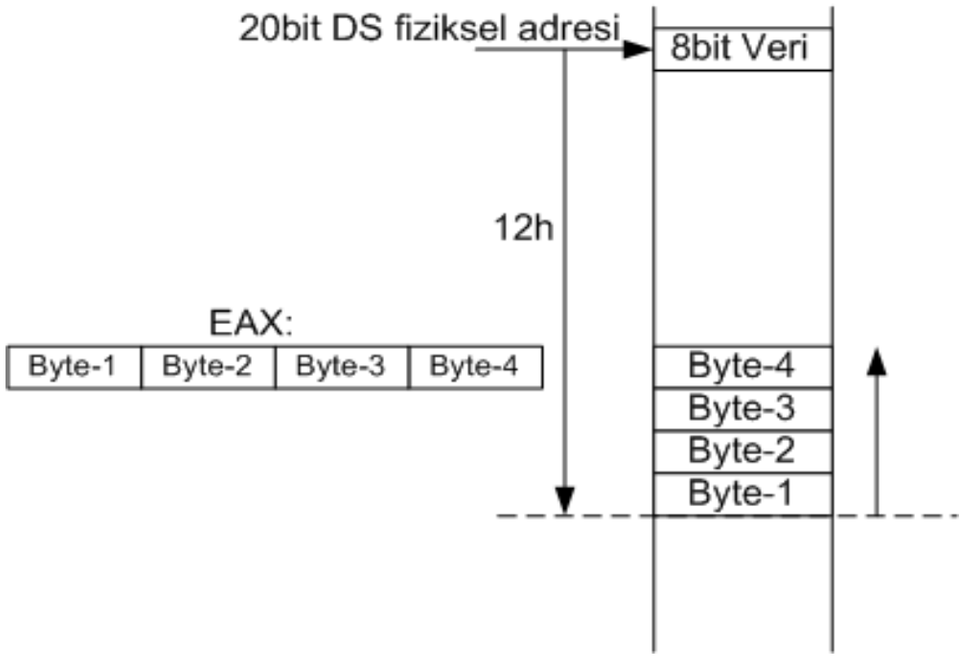
Answer:

(103F)h değeri DS ile belirlenmiş belleğin başlangıç adresinden verinin bulunduğu yeri işaret eder.

- DS: 94D0h * 10h = 95D00h (20 bitlik fiziksel adres elde edilir. Bu RAM belleğin başlangıç adresidir.)
- Bellk gözünün adresi: 95D00h + 103Fh=95E3Fh elde edilir.
- and it's equivalent to effective address: 95D3h:000Fh

Addressing modes

- **DS:2000h ve MOV EAX, [12h] ise bellek gözünün konumunu belirleyiniz ve veri transferini gerçekleyiniz.**
- MOV Komutunda hedef ve kaynak 32 bit olduğu görülmektedir.
- Köşeli parentezin önünde segment register göstergesi olmadığından default olarak belleğin RAM bellek olduğu görülmektedeir.
- DS registerîndaki değer 10h ile çarpılarak 20 bitlik fizksel adres elde edilir. 2000h*10h=20000h
- Bu değere köşeli parentezin içerisindeki değer toplanarak ilgili bellek gözünün konumu belirlenmiş olur.
- Fiziksel adres=20000h + 12h =20012h
- MOV komutu ile belirlenmiş bellek gözünden 4 byte veri EAX register'ına transfer edilir.



Örnek

- CPU’dan bellek gözlerini seçmek için A15,A14,A13,A12,A11, …, A1, A0 adres hatları gelmektedir. 16 adet.
- 4 adet bellek olduğu için CPU’dan Adres dekoding devresi girişine 2 adres hattı (2 bit)gelir.
- 00 gelirse ROM
- 01 gelirse RAM1
- 10 gelirse RAM2
- 11 gelirse RAM3 seçilir.

- 2 adet bellek seçmek için CPU dan adres hattı geliyorsa indisleri nedir? A17, A16 dır.
- CPU’nun bellek erişim kapasitesi kaç byte dır? 2^18=256Kbyte. Toplam adres hattı sayısı=18 adet.

Örnek:

Mov ‘segment regsiter’, 16bit

Mov Segment Register: [IP, BX, SI, DI, BP, SP, int Sayı], ‘Data’

- Köşeli parentez görünce, bir belleğin başlangıç adresini tanımlar (16bit).
- Köşeli parentezin içi ise belleğin gözünün adresini tanımlar.
- Bellek gözü adresi nedir? CPU’dan çıkan adres bus hatlarının bir kısmı adres dekoding devresine gider, bellekleri seçer. CPU’nun iç mimarisindeki karşılığı nedir? Segment Registers: CS, DS, SS, ES.
- Bellek gözü nasıl seçilir? Köşeli parentezin içinde: Değişkenler, Bx, SI, DI, SP, BP, tam sayısaln rakamlar
  - SS→ SP, BP
  - DS, ES → Bx, SI, DI
  - CS → IP
- Hangi bellek? Köşeli parentezin önünde segment register indisi olmadığı durumlarda default olarak DS tanımlanır.

Örnek: Mov DS, 0A000h

Mov [12h], 55h

- Hangi bellek? Köşeli parentezin önünde segment register indisi olmadığı için default olarak DS tanımlanır.
- Bellek gözünü adresini bulun? Data segment register 16 bit: (1010 0000 0000 0000)b
- Bellek gözünün fiziksel adresi hesaplanırken, ilgili segment register'ın sağına 4 bit 0 eklenir.
- Data segment register fiziksel adres, 20 bit: (1010 0000 0000 0000 0000)b=(A0000)h
- Bellek gözünün fiziksel adresi: Data segment register 20 bitlik fiziksel adres + Köşeli parentezin içindeki değer 16 bit
- Bellek gözünün fiziksel adresi:
- Belleğin başlangıç adresi=(1010 0000 0000 0000 0000)b
- + (0001 0010)b
- = (1010 0000 0000 0001 0010)b=(A0012)h
- Bellek gözünün adresi bulunduktan sonra 55h değeri bu göze yazılır.

Pentium Addressing modes

Pentium Addressing modes

- Direct: **MOV EAX, [12h]**
  - *Copy* value located at address 10h
- Indirect: **MOV EAX, [EBX]**
  - *Copy* value pointed to by register BX
- Indexed: **MOV AL, [EBX + ECX * 4 + 10h]**
  - *Copy* value from array (BX[4 * CX + 0x10])
- Pointers can be associated to type
  - **MOV AL, byte ptr [BX]**

- Köşeli parentez, belleği tanımlar.
- Eğer köşeli parentezin önünde segment register tanımı yok ise default olarak DS vardır. Diğer segment register'larda Segment Register: **[…] şeklinde tanımlanır. Örnek: ES: [12h]**
- Köşeli parentez içindeki değer verinin bulunduğu belleğin konumunu işaret eder.
- Pentium CPU olmasından dolayı 32 bitlik bellek konumunun adresi Segment register'ın 32 bitlik değerinin sağına 4 bit 0 ilave edilerek 36 bitlik fiziksel adres elde edilir. Böylece erişilecek belleğin başlangıç adresi tam olarak belirlenmiş olur.
- Bu başlangıç adresine köşeli parentezin içerisindeki değerler ilave edierek, bellek gözünün konumu belirlenmiş olur.

Pentium Addressing modes

- Direct: **MOV EAX, [12h]** ; EAX register'I 32 bit olduğu için Data Regment register değeri 32 bit olur, sağına 4 bit eklenerek 36 bitlik fiziksel adres elde edilir. Böylece RAM belleğin başlangıç adres bulunmuş olur. Bu değere 12h değeri eklenerek transfer edilecek verinni konumu belirlenmiş olur.
  - *Copy* value located at address 12h
- Indirect: **MOV EAX, [EBX]** ; EAX register'I 32 bit olduğu için Data Regment register değeri 32 bit olur, sağına 4 bit eklenerek 36 bitlik fiziksel adres elde edilir. Böylece RAM belleğin başlangıç adres bulunmuş olur. Bu değere EBX regiter'ının 32 bit'lik değeri eklenerek transfer edilecek verinni konumu belirlenmiş olur.
  - *Copy* value pointed to by register EBX
- Indexed: **MOV AL, [EBX + ECX * 4 + 10h] ;** 1 byte okunacak (AL), nereden okuncak? Data segmentten (32bit) fiziksel adres bulunur (36bit) bu değere EBX + ECX * 4 + 10h sayısal değeri toplanarak bellek gözünün fiziksel adresi elde edilmiş olur.
  - *Copy* value from array (EBX+[4 * CX + 0x10])
- Pointers can be associated to type
  - **MOV AL, byte ptr [BX]**

Addressing Modes

Addressing modes for 32-bit address size on 32-bit or 64-bit x86 processors can be summarized by this formula:

$$\begin{Bmatrix} CS: \\ DS: \\ SS: \\ ES: \\ FS: \\ GS: \end{Bmatrix} \left[\left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ ESP \\ EBP \\ ESI \\ EDI \end{Bmatrix} \right] + \left[\begin{Bmatrix} EAX \\ EBX \\ ECX \\ EDX \\ EBP \\ ESI \\ EDI \end{Bmatrix} * \begin{Bmatrix} 1 \\ 2 \\ 4 \\ 8 \end{Bmatrix} \right] + [\text{displacement}]$$

Addressing Modes

Addressing modes for 64-bit code on 64-bit x86 processors can be summarized by these formulas:

$$\left\{\begin{matrix} : \\ FS: \\ GS: \end{matrix}\right\} [\text{general register}] + \left[\text{general register} * \left\{\begin{matrix} 1 \\ 2 \\ 4 \\ 8 \end{matrix}\right\}\right] + [\text{displacement}]$$

And RIP + [displacement]

Addressing Modes

- The 8086 had 64 KB of 8-bit (or alternatively 32 K-word of 16-bit) I/O space, and a 64 KB (one segment) stack in memory supported by hardware.
- Only words (2 bytes) can be pushed to the stack.
- The stack grows downwards (toward numerically lower addresses), its bottom being pointed by SS:SP.
- There are 256 interrupts, which can be invoked by both hardware and software.
- The interrupts can cascade, using the stack to store the return address.

LEA – XCHG – PUSH /POP

LEA: Load Effective Address

- LEA REG, memory

- Algorithm: REG = address of memory (offset). Generally this instruction is replaced by MOV when assembling when possible.

Example:
- ORG 100h
- LEA AX, m
- RET
- m DW 1234h
- END

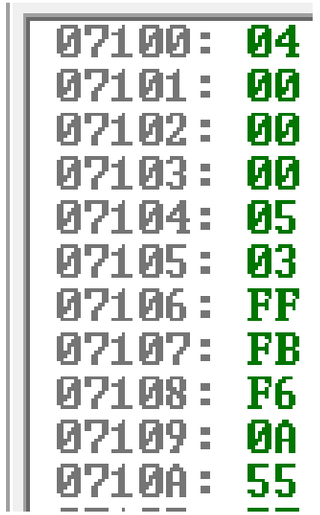
Not:
- AX is set to: 0104h.
- LEA instruction takes 3 bytes, RET takes 1 byte, we start at
- 100h, so the address of 'm' is 104h.

Örnek: Değişkenlerin bellek gözündeki yerleri

```
org 100h
lea si, I
lea bx, J
lea di, K
ret

I db 4
J db ?
K db 5,3,-1,-5,-10, 0Ah, 01010101b
```

SI: 0109h
DI: 010Bh
bx: 010Ah

DS: 0700h
Belleğin fiziksel baslangıç adresi=07000h



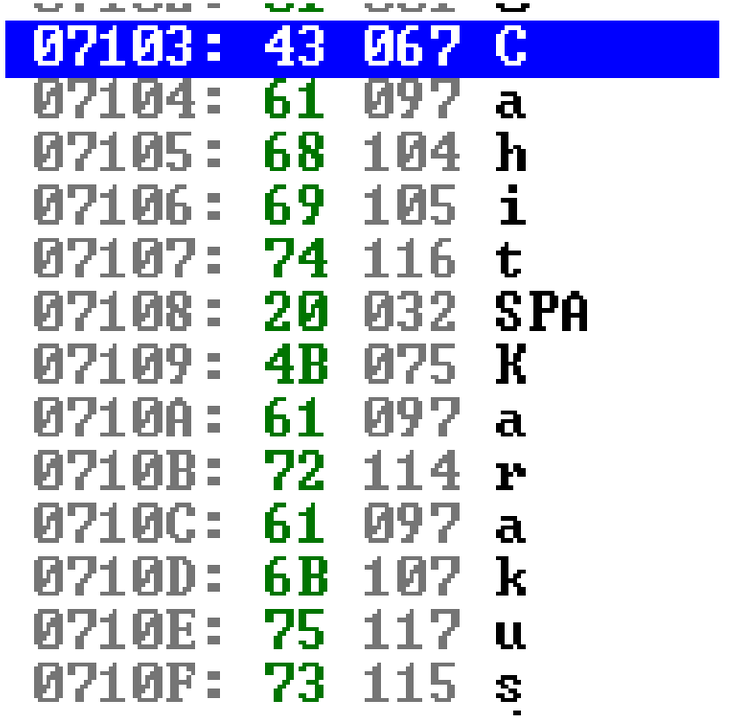
I değişkenin kayıt edildiği bellek gözü=7000h+SI=7109h
K değişkenlerinin(3 adet?) kayıt edildiği bellek gözü = 710Bh + DI
J değişkenin kayıt edildiği bellek gözü =7000h+BX=710Ah

LEA komutu değişkenlerin ayıt edildiği belleğin indislerini transfer eder.

Örnek: Değişkenleri bellek gözündeki yerleri

```
org 100h
lea si, I
I db "Cahit Karakus"
ret
```

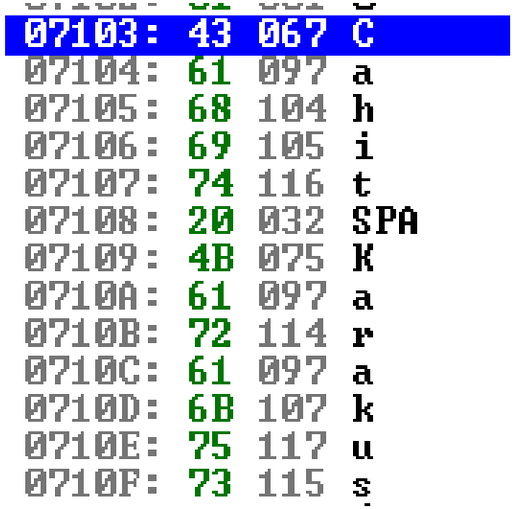
SI: 0103h
DS: 0700h
Belleğin baslangic adresi=07000h
I değişkenin kayıt edildiği bellek gözü=7000h+SI=7103h



Örnek: Değişkenleri bellek gözündeki yerleri

```
org 100h
lea si, I
I db "Cahit Karakus"
ret
```

SI: 0103h
DS: 0700h
Belleğin baslangic adresi=07000h
I değişkenin kayıt edildiği bellek gözü=7000h+SI=7103h



XCHG:Exchange values of two operands.

- XCHG Operand1, Operand2
- Algorithm: operand1 < - > operand2

Operand1, operand2:

- REG, memory
- memory, REG
- REG, REG

- **Example:**

```
MOV AL, 5
MOV AH, 2
XCHG AL, AH ; AL = 2, AH = 5
XCHG AL, AH ; AL = 5, AH = 2
RET
```

Data Transfer Komutları (Push, Pop)

Segment: SS, Komutlar: PUSH, POP

PUSH reg16/ mem

PUSH reg16

(SP) ← (SP) – 2
Fiziksel Adres = (SS) x 10h + SP
(MA $_S$; MA $_S$ + 1) ← (reg16)

PUSH mem

(SP) ← (SP) – 2
MA $_S$ = (SS) x 16_{10} + SP
(MA $_S$; MA $_S$ + 1) ← (mem)

POP reg16/ mem

POP reg16

MA $_S$ = (SS) x 16_{10} + SP
(reg16) ← (MA $_S$; MA $_S$ + 1)
(SP) ← (SP) + 2

POP mem

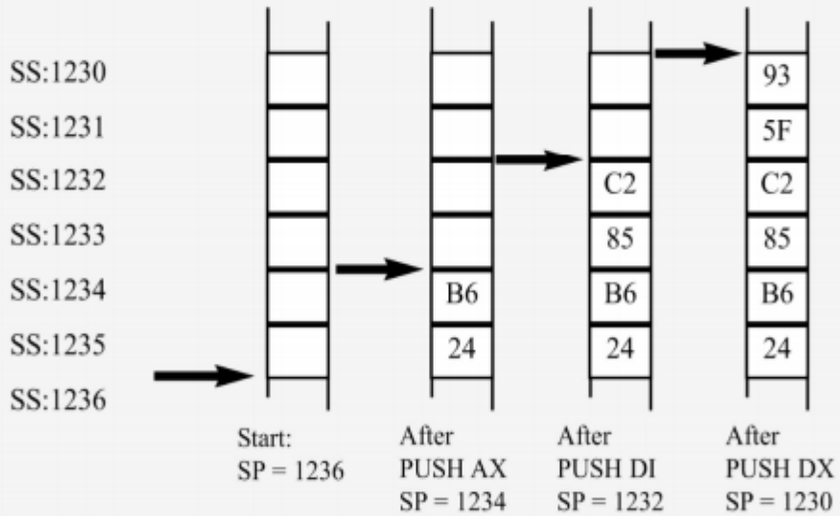
MA $_S$ = (SS) x 16_{10} + SP
(mem) ← (MA $_S$; MA $_S$ + 1)
(SP) ← (SP) + 2

Push - Pop

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

```
PUSH  AX
PUSH  DI
PUSH  DX
```

Solution:



- Push, 16 bitlik bir kaydın içeriğini kaydettiğinden, iki kez azalır.
- AH'nin içeriği olan 24H, 1235 adresli bellek konumuna kaydedilir. AL, 1234 konumunda saklanır.
- Her pop ile yığının ilk 2 baytı talimat tarafından belirtilen CPU kaydına kopyalanır ve yığın işaretçisi iki kez artırılır.
- Yığının tam fiziksel konumu, yığın segmenti (SS) kaydının ve yığın işaretçisi SP'nin değerine bağlıdır.
- Yığın için fiziksel adresleri hesaplamak için SS'yi sola kaydırın, ardından yığın işaretçi kaydı olan ofset SP'yi ekleyin.

Example

If SS = 3500H and the SP is FFFEH,

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

Solution:

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

Push - Pop

PUSH source" instruction does the following:

- Subtract 2 from SP register.
- Write the value of source to the address SS:SP.

"POP destination" instruction does the following:

- Write the value at the address SS:SP to destination.
- Add 2 to SP register.
- The current address pointed by SS:SP is called the top of the stack.

Örnek: Push – Pop (Stack Segment: SP)

Örnek-1:

```
ORG 100h
MOV AX, 1212h ; store 1212h in AX.
MOV BX, 3434h ; store 3434h in BX
PUSH AX ; store value of AX in stack segment.
PUSH BX ; store value of BX in stack segment.
MOV AX, 00ACCh
MOV BX, 00FDDh
OR AX, BX
MOV DX, AX
POP BX ; set AX to original value of BX.
POP AX ; set BX to original value of AX.
RET
END
```

```
ORG 100h
MOV AX, 1234h
PUSH AX
POP DX ; DX = 1234h
RET
```

Group I:
Register ve anlık değerler için adresleme modları

1. Register Addressing

Register mode – In this type of addressing mode both the operands are registers.

- MOV AX, BX ; veri transferi
- XOR AX, DX; mantıksal
- ADD AL, BL ; toplama AL=AL+BL

MOV Instruction

- MOV destination,source
  - **8 bit moves**
    - MOV CL,55h
    - MOV DL,CL
    - MOV BH,CL
  - **16 bit moves**
    - MOV CX,468Fh
    - MOV AX,CX
    - MOV BP,DI

MOV Instruction

• MOV AX,58FCH	√
• MOV DX,6678H	√
• MOV SI,924BH	√
• MOV BP,2459H	√
• MOV DS,2341H	x
• MOV CX,8876H	√
• MOV CS,3F47H	x
• MOV BH,99H	√

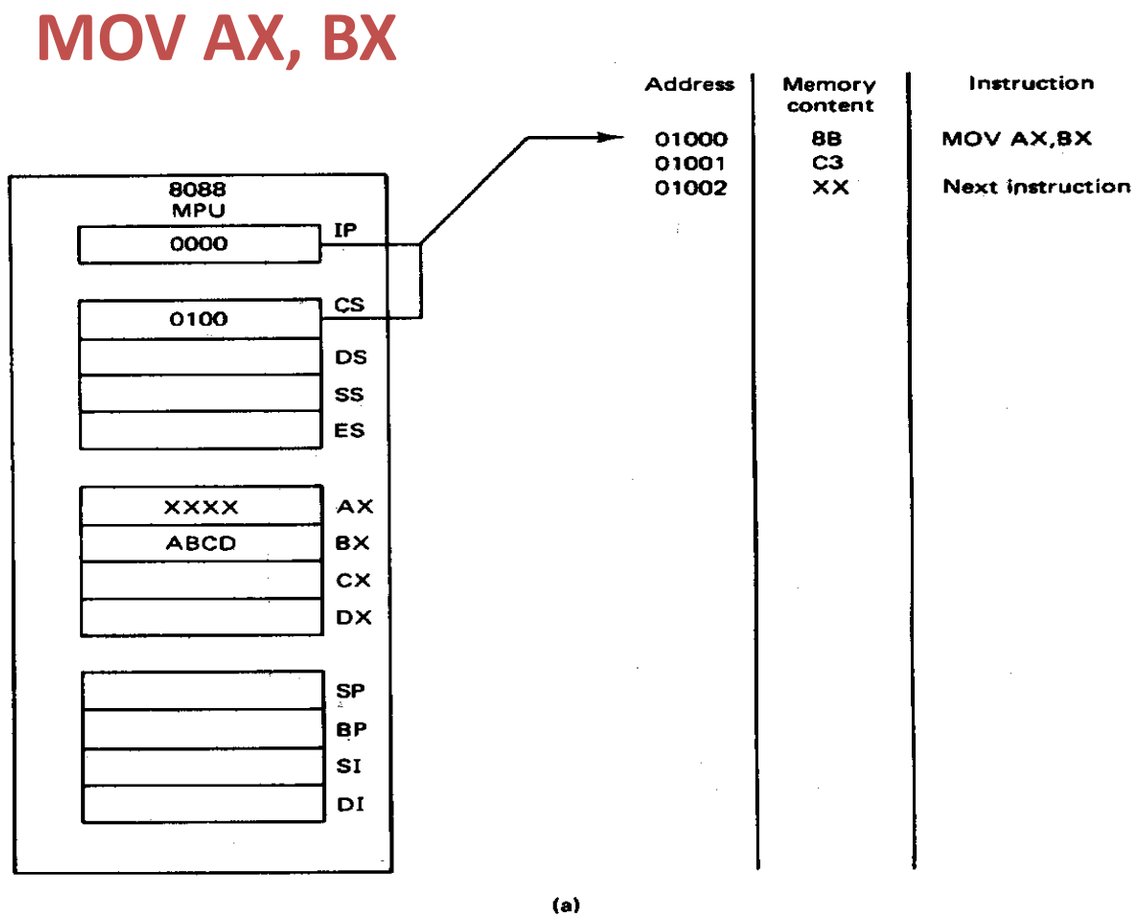
- Register'lara değer atama işlemleri MOV komutu ile yapılır.
- MOV komutlarında register uzunlukları heriki tarafta da aynı olmak zorundadır.
- Segment register'lara doğrudan değer atanmaz. Segment register'a değer atanması için, değer önce MOV komutuyla AX register'ına atanır , sonra AX register'ın içeriği segment register'a transfer edilir .

```
MOV AX, 0C000h
MOV DS, AX
```

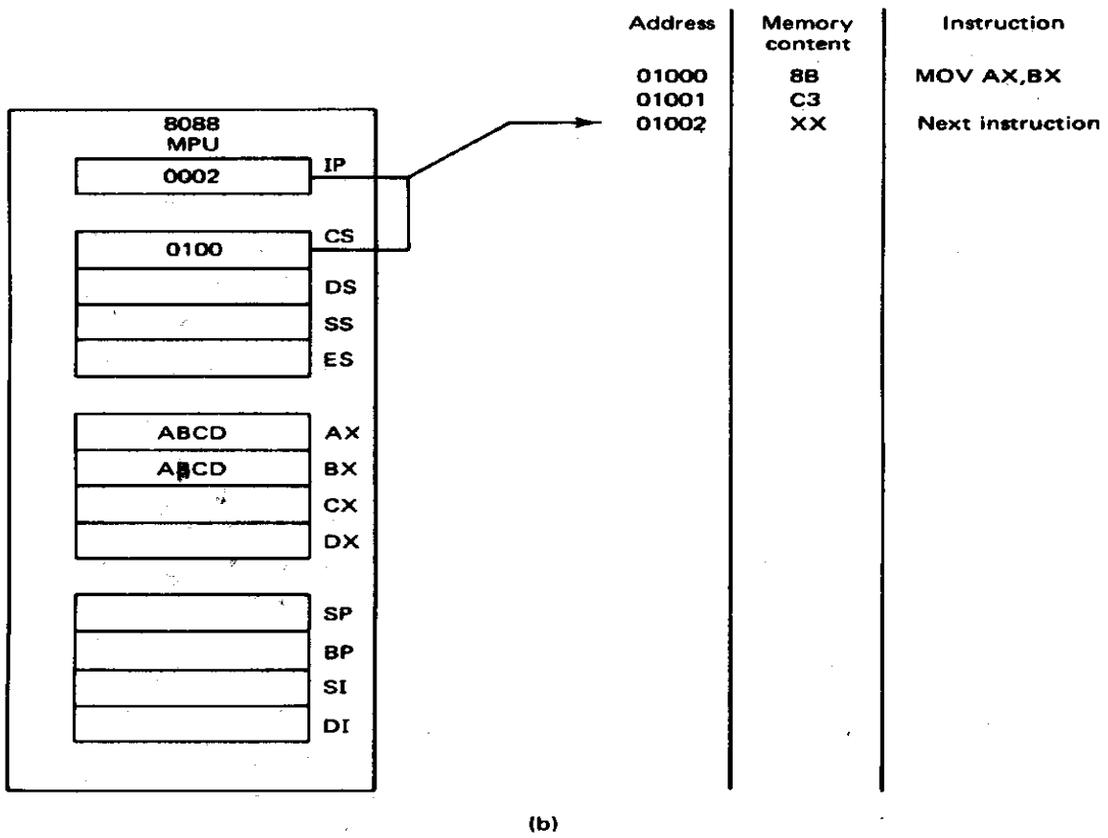
- AX,BX,CX,DX regster'ların 16bittir. AH-AL, BH -BL, CH-CL, DH-DL register'ların kapasitesi 8 bittir.

Register Addressing

MOV AX, BX



Before execution



After execution

2. Immediate mode

- Immediate mode – Bu tip adresleme modunda kaynak komutu 8 bitlik veya 16 bitlik bir veridir. Hedef komut hiçbir zaman anlık veri olamaz.

MOV AX, 2000h
MOV CL, 0Ah
ADD AL, 45h
AND AX, 0000h

- Segment Register'a değer atamak için bir data register'ın gerekli olduğuna dikkat edin. Segment Register'larına doğrudan sayısal değer atanmaz. Data register üzerinden değer atanır.

MOV AX, 2000h
MOV CS, AX

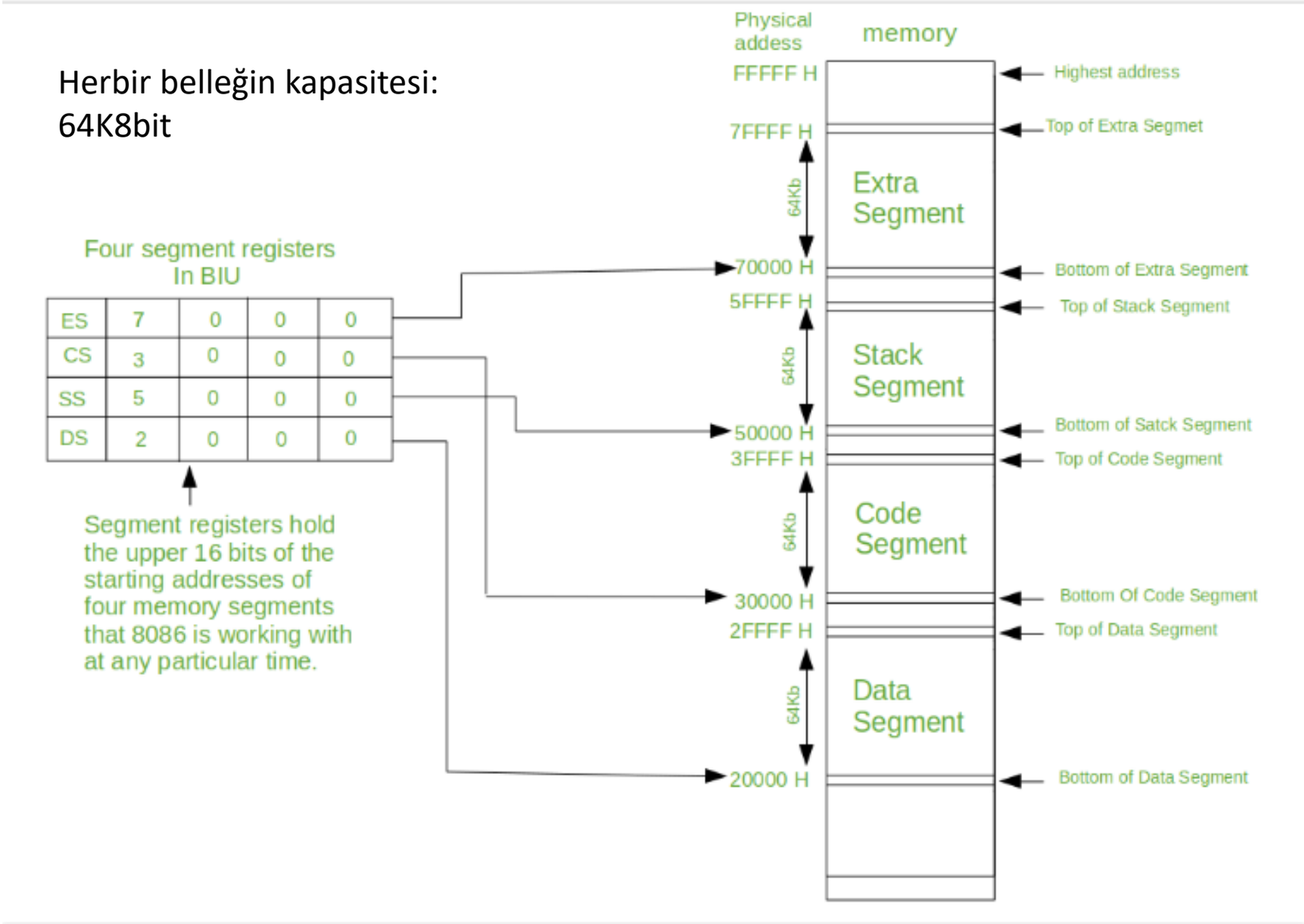
Group II:
Bellek Erişim Adres Modları

Segmentasyon

- Segment register’lar 16 bit.
- Belleğin başlangıç adreslerini gösterir.
- Fiziksel boyut: 20 bit.

Assembly dilinde yazılım yazmaya başlamadan önce, beleklerin başlangıç ve bitiş adreslerinin belirlenmesi gerekmektedir.

Herbir belleğin kapasitesi:
64K8bit



Segmentasyonun ana avantajları

- Güçlü bir bellek yönetim mekanizması sağlar.
- Veri ile ilgili veya yığınla ilgili işlemler farklı segmentlerde gerçekleştirilebilir.
- Kodla ilgili işlem ayrı kod bölümlerinde yapılabilir.
- İşlemlerin verileri kolayca paylaşmasına izin verir.
- İşlemcinin adres yeteneğini genişletmesini izin verir, yani segmentasyon, 1 Megabayt adresleme yeteneği vermek için 16 bit kayıtların kullanılmasına izin verir. Segmentasyon olmadan 20 bit kayıt gerektirir.
- Her alan için birden fazla segment ayırarak kod verilerinin bellek boyutunu veya yığın segmentlerini 64 KB'den daha fazla artırmak mümkündür.

Addressing Modes : Memory Access

- 20 Address lines $\Rightarrow$ 8086 can address up to 2^{20} = 1M bytes of memory
  However, the largest register is only 16 bits
- Physical Address will have to be calculated Physical Address : Actual address of a byte in memory. i.e. the value which goes out onto the address bus.
- Memory Address represented in the form – Seg : Offset (Eg - 89AB:F012)
- Each time the processor wants to access memory, it takes the contents of a segment register, shifts it one hexadecimal place to the left (same as multiplying by 16_{10}), then add the required offset to form the 20- bit address

16 bytes of contiguous memory

89AB : F012 $\rightarrow$ 89AB $\rightarrow$ 89AB0 (Paragraph to byte $\rightarrow$ 89AB x 10 = 89AB0)
F012 $\rightarrow$ 0F012 (Offset is already in byte unit)
+ -------
98AC2 (The absolute address)

Segment Addressing Registers

- Köşeli parentez segment adresini belirtmektedir.
- Köşeli parentezin önünde segment adresi yok ise varsayılan olarak data segmentin başlangış adresini işaret eder.
- PA: Fiziksel adres
- MOV CX,[BX][DI]+8h
- Komutunda fiziksel adres, PA= (DS)h x 10h + (BX)h + (DI)h + 8h

Not:

- (DS)h x 10h : DS segment register'larındaki 16 bitlik sayısal ifadenin sağına 4 bit 0 eklenir; 20 bitlik fiziksel adres elde edilir. Böylece 16 desimal değeri ile çarpılmış olur.
- Bu 20 bitlik değer Bx, DI register'ların içeriği ve 8 sayısal değeri ile toplanarak verinin bulunduğu bellek gözüne erişilir ve bu bellek gözünden 16 bit WORD (2 adet byte) değeri Cx register'ına transfer edilir.

Örnek: Mov Bx, [300h] bu işlemin sonucunda Bx nedir? (DS: 200h, Ram belleğin fiziksel adresi 2300h'de kayıtlı (0A0B)h değeri bulunmaktadır.

  - DS:200h ise fiziksel adres=(2000)h + (300)h=(2300)h, o halde sonuç Bx=(0AoB)h olur.

1. Direct Addressing Mode

- Bu adresleme modu doğrudan olarak adlandırılır çünkü işlenenin segment tabanından yer değiştirmesi doğrudan komutta belirtilir.

- Burada, veri işleneninin depolandığı bellek yerinin etkin adresi talimatta verilmiştir. Etkili adres, doğrudan talimatta yazılan sadece 16 bitlik bir sayıdır.

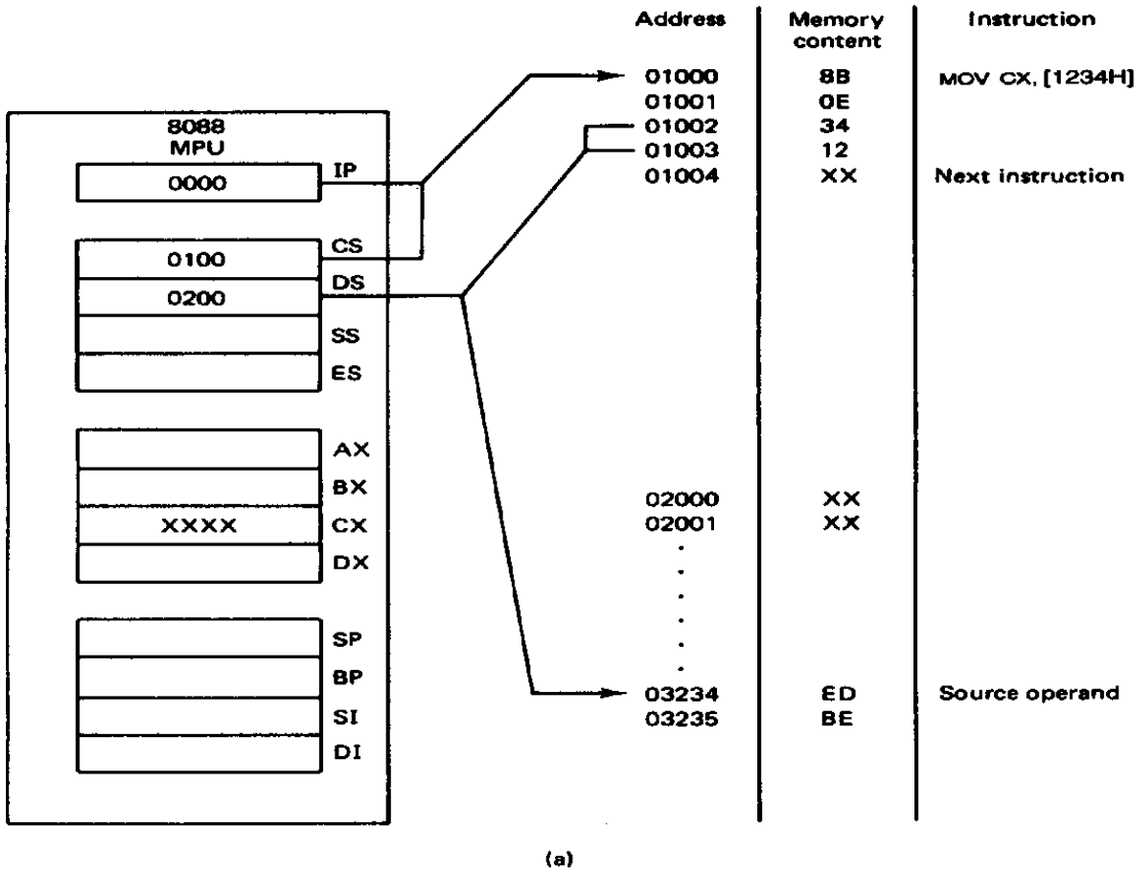
Example:

```
MOV   BX, [1354H]
MOV   BL,  [0400H]
```

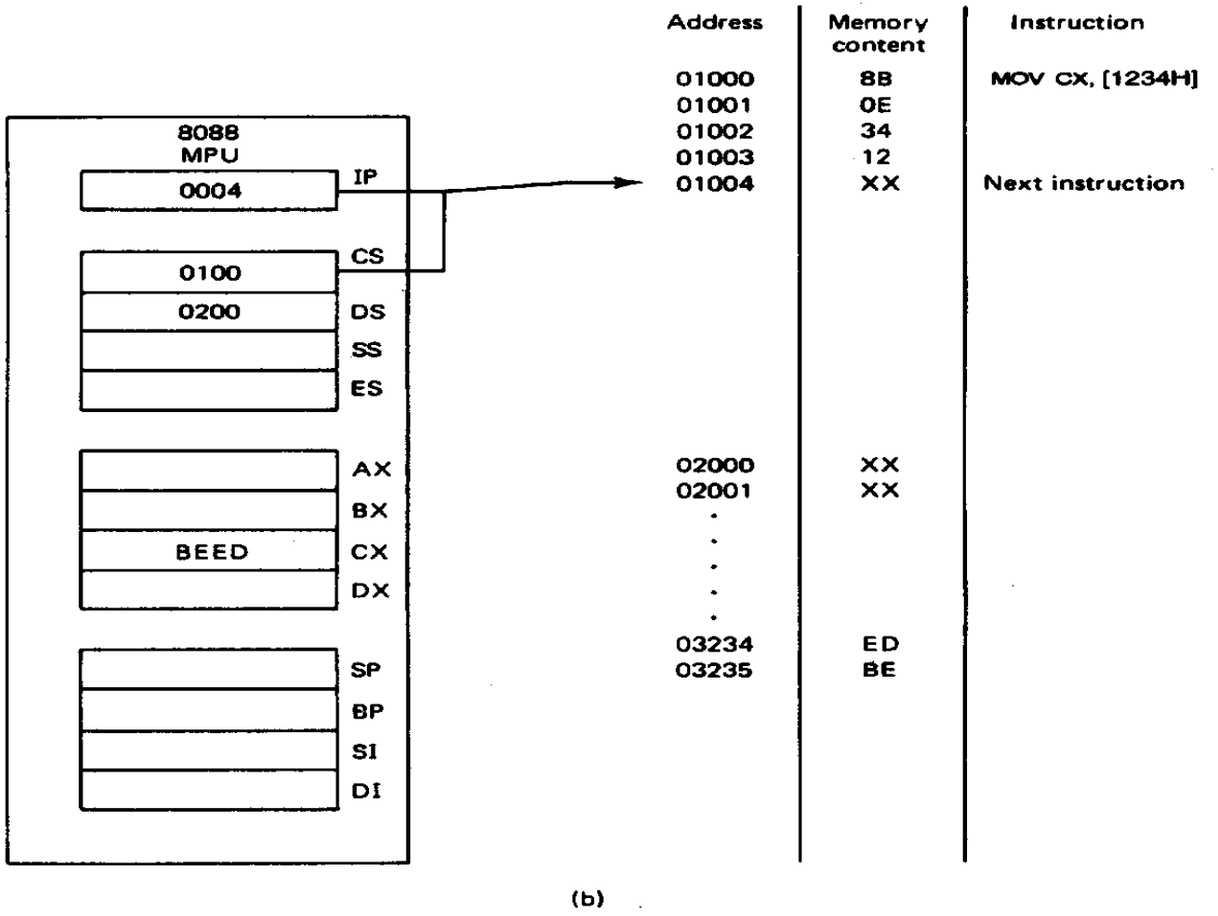
- Köşeli parantezin önünde segment register ifadesi yok ise default olarak DS olduğunu gösterir.
- 1354H'nin etrafındaki köşeli parantezler bellek konumunun içeriğini gösterir.
- Köşeli parentizin önünde SS: [...], ES:[...] olarak segment register belirtilmemişse varsayılan ofset adres, data segment'tir.
- Komut yürütüldüğünde, bellek konumunun içeriğini BX saklayısına kopyalanacaktır.

Direct Addressing Mode

MOV CX, [1234h]



(a)



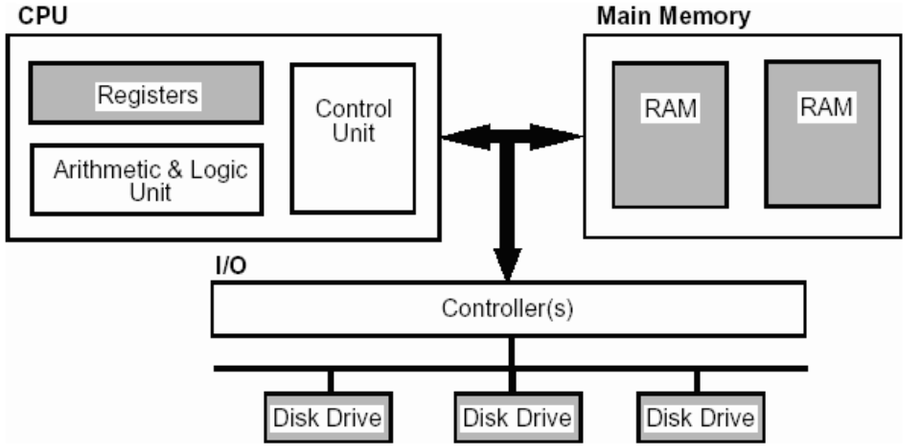
(b)

Bellekte aşağıdan yukarıya doğru ilerlenir.

2. Register Indirect Adressing Mode

Register indirect mode

- İşlenenin komutun ofset adresi bir register'da bulunur.
- Register, bellek konumunda bir işaretçi görevi görür.
- Yalnızca BX, SI, DI veya BP Register'larına izin verilir.
- BX, SI, DI için segment numarası DS'dedir.
- BP için segment numarası SS'dir.
- Operand formatı [Register]



Example: suppose SI=0100h and [0100h]=1234h

- MOV AX, SI ; AX=0100h
- MOV AX, [SI] ;AX=1234h

Register Indirect Addresing Mode

Segment Registers	CS	DS	ES	SS
Offset Register	IP	SI,DI,BX	SI,DI,BX	SP,BP

Instruction Examples	Override Segment Used	Default Segment
MOV AX,CS:[BP]	CS:BP	SS:BP
MOV DX,SS:[SI]	SS:SI	DS:SI
MOV AX,DS:[BP]	DS:BP	SS:BP
MOV CX,ES:[BX]+12	ES:BX+12	DS:BX+12
MOV SS:[BX][DI]+32,AX	SS:BX+DI+32	DS:BX+DI+32

Register Indirect Addressing

In Register indirect addressing, name of the register which holds the effective address (EA) will be specified in the instruction. Content of the DS register is used for base address calculation. In this addressing mode the effective address (EA) is in SI, DI or BX.

Example:

MOV CX, [BX]

Operations:

EA = (BX)

BA = (DS) x 16_{10}

MA = BA + EA

(CX) ← (MA) or,

(CL) ← (MA)

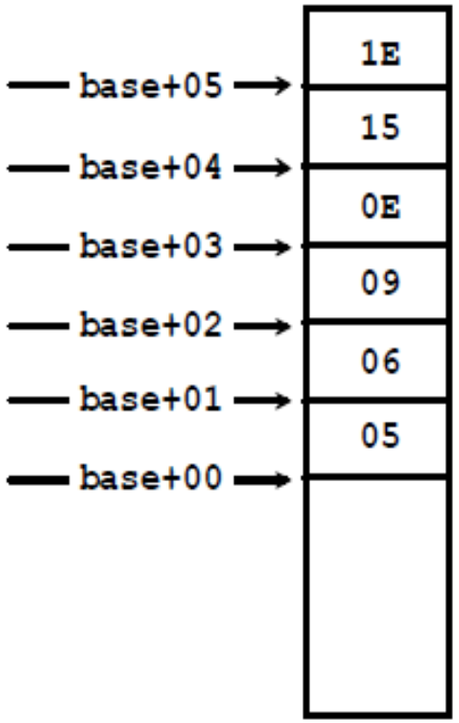
(CH) ← (MA +1)

Register Indirect Addressing

Example

Writing Values into a Table

```
    ORG 0x100
section .bss
    base resb 6
section .text
    MOV     SI, 0
L1: MOV     AX,SI
    IMUL    AL
    ADD     AL,5
    MOV     [base + SI],AL
    INC     SI
    CMP     SI,5
    JLE     L1
    mov     ax,4C00h
    int     21h
```



Arithmetic With Stored Data

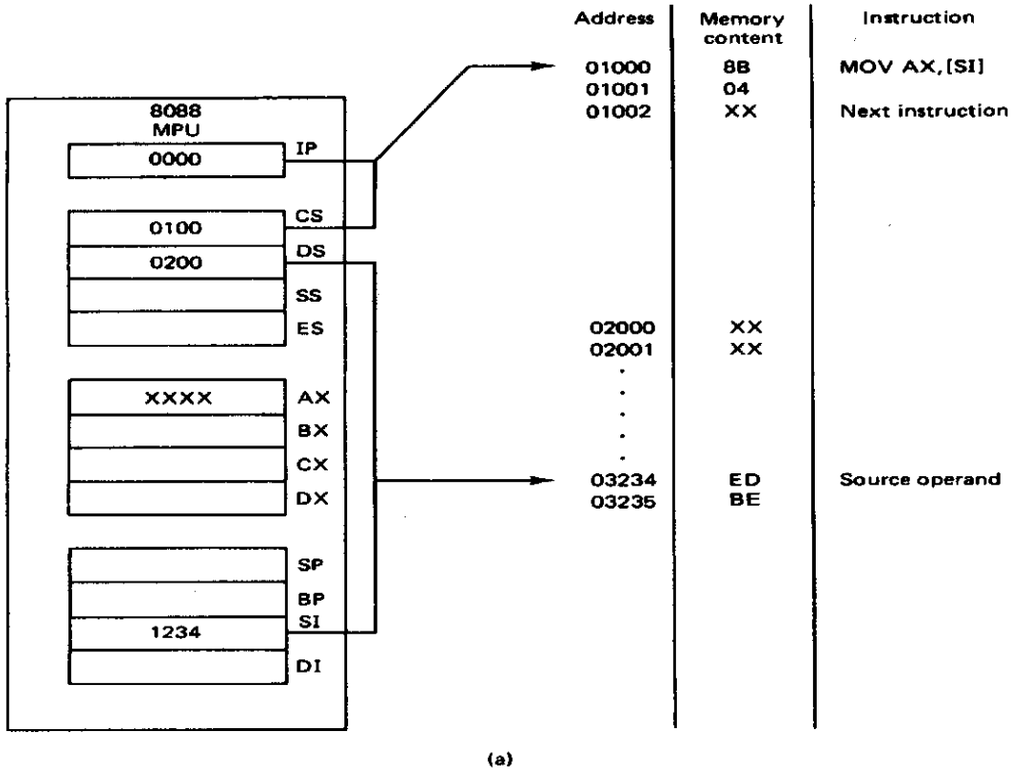
```
    ORG 0x100
section .data
    table dw 1,2,3,4,5,6,7,8,9,10,0
section .text
    MOV     SI,table
    MOV     AX,0000
L1: ADD     AX,[SI]
    ADD     SI,2
    CMP     WORD [SI],0

    JNZ     L1
    mov     ax,4C00h
    int     21h
```

```
; WORD – operate on
; 16-bit word = 2 bytes
; from [SI+1].[SI]
```

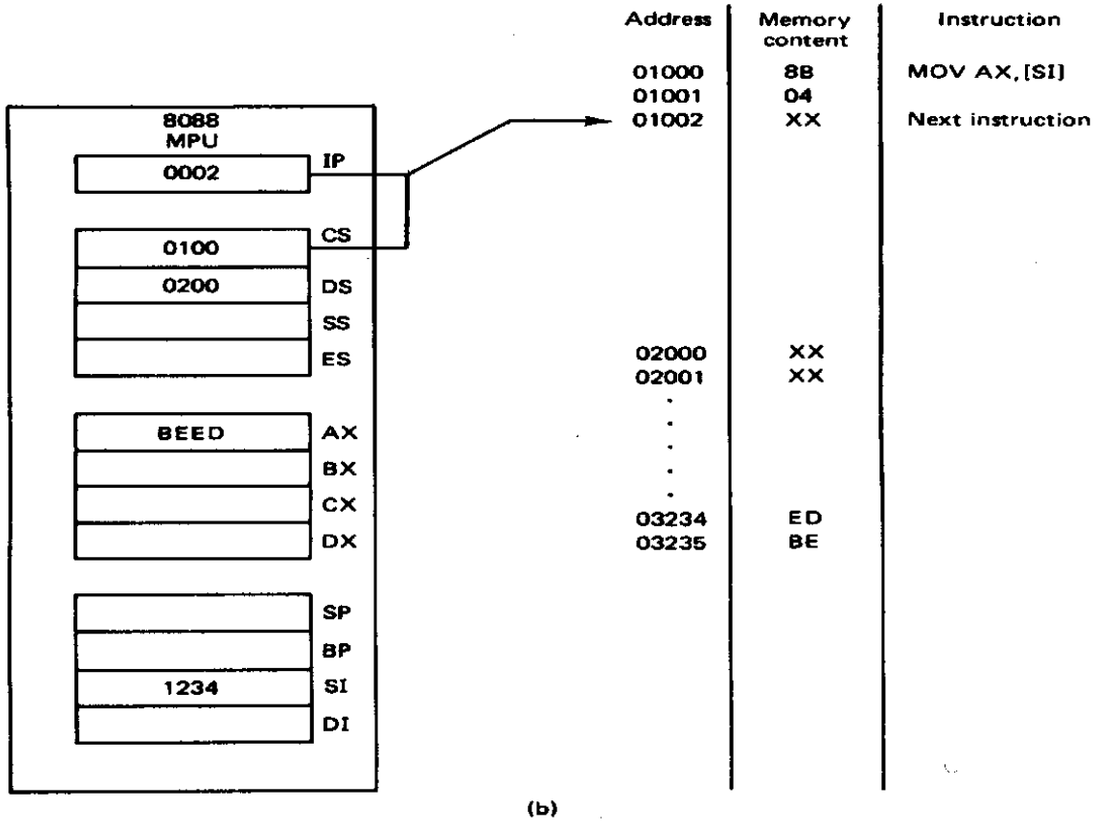
AX = 0 → 1 → 3 → 6 → 10 → 15 → 21 → 28 → 36 → 45 → 55

Register Indirect Addressing Mode

MOV AX, [SI]



(a)



(b)

3. Based Addressing

In Based Addressing, BX or BP is used to hold the base value for effective address and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction. In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value. When BX holds the base value of EA, 20-bit physical address is calculated from BX and DS. When BP holds the base value of EA, BP and SS is used.

Example: MOV AX, [BX + 08H]

Base register: BX, BP

Index register: SI, DI

Operations:

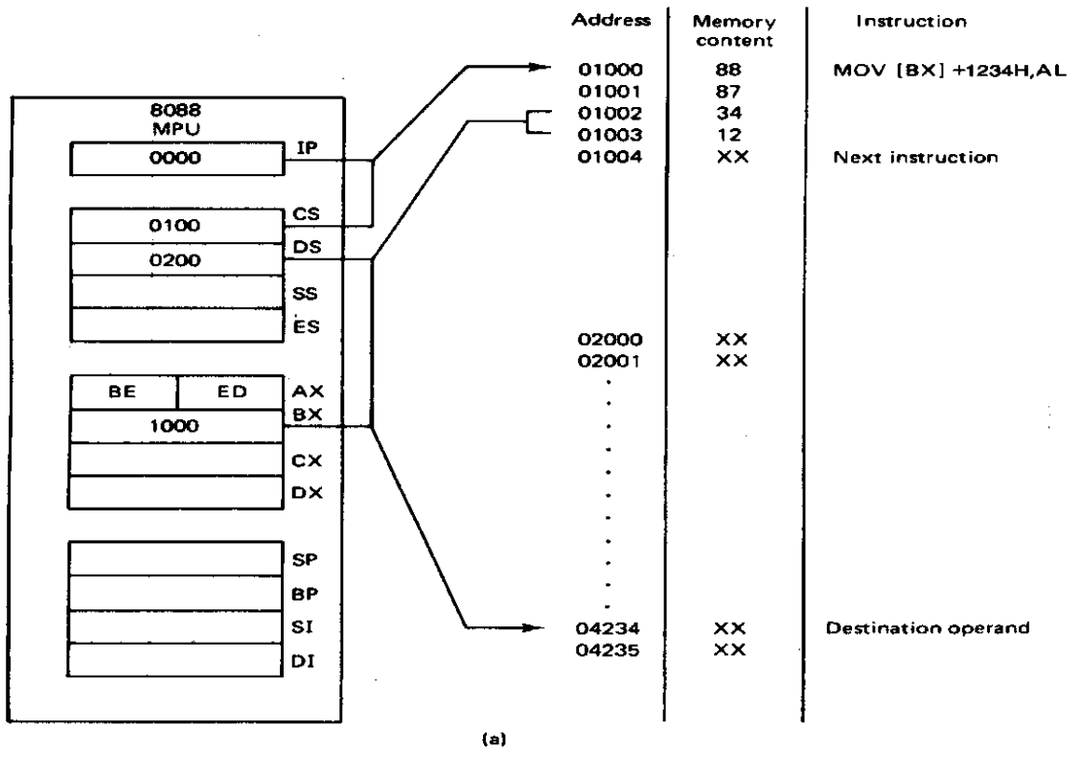
$0008_H \leftarrow 08_H$ (Sign extended)
EA = (BX) + 0008_H
BA = (DS) x 16_{10}
MA = BA + EA

(AX) ← (MA) or,

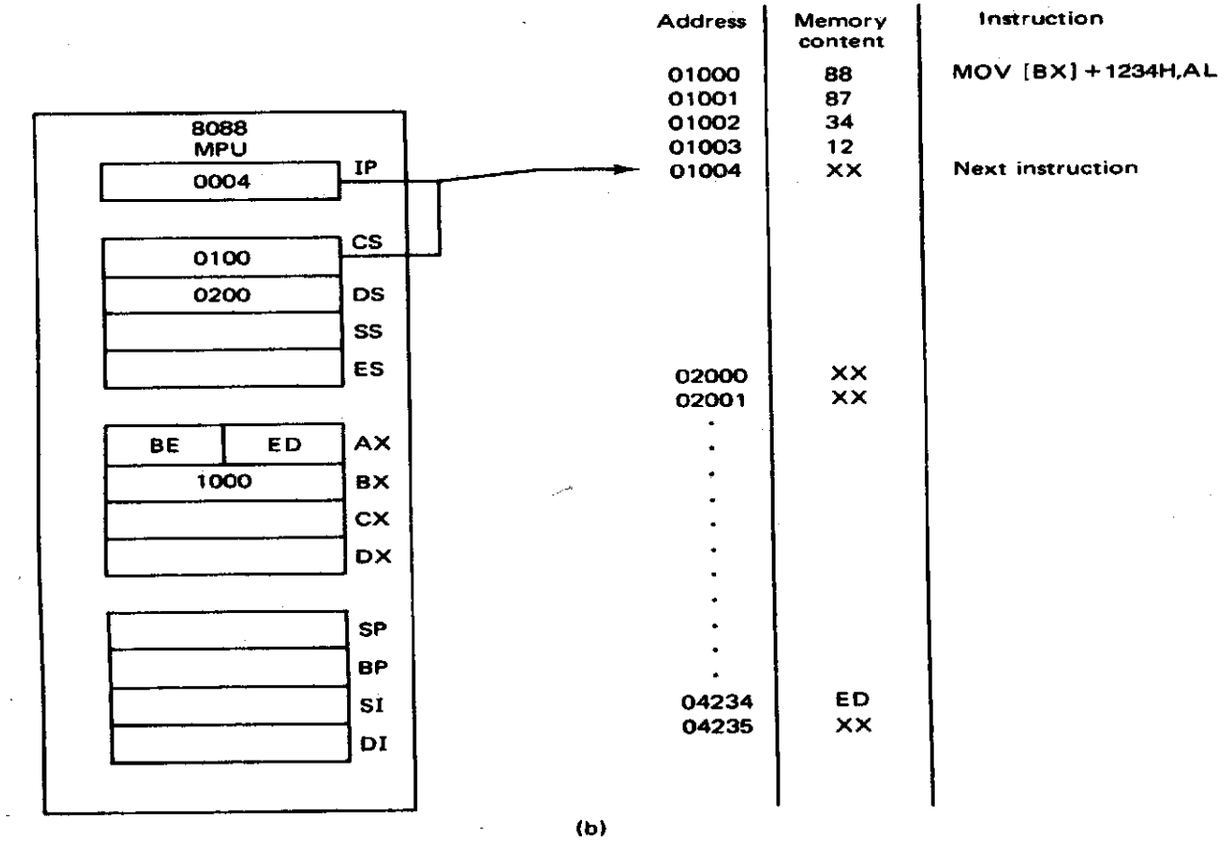
(AL) ← (MA)
(AH) ← (MA + 1)

Based Addressing

MOV [BX]+1234H,AL



(a)



(b)

4. Indexed Addressing

In this type of addressing mode the effective address is sum of index register and displacement. SI or DI register is used to hold an index value for memory data and a signed 8-bit or unsigned 16-bit displacement will be specified in the instruction. Displacement is added to the index value in SI or DI register to obtain the EA. In case of 8-bit displacement, it is sign extended to 16-bit before adding to the base value.

Example:

MOV CX, [SI + 0A2H]

Operations:

$FFA2_H \leftarrow A2_H$ (Sign extended)

$EA = (SI) + FFA2_H$

$BA = (DS) \times 16_{10}$

$MA = BA + EA$

$(CX) \leftarrow (MA)$ or,

$(CL) \leftarrow (MA)$

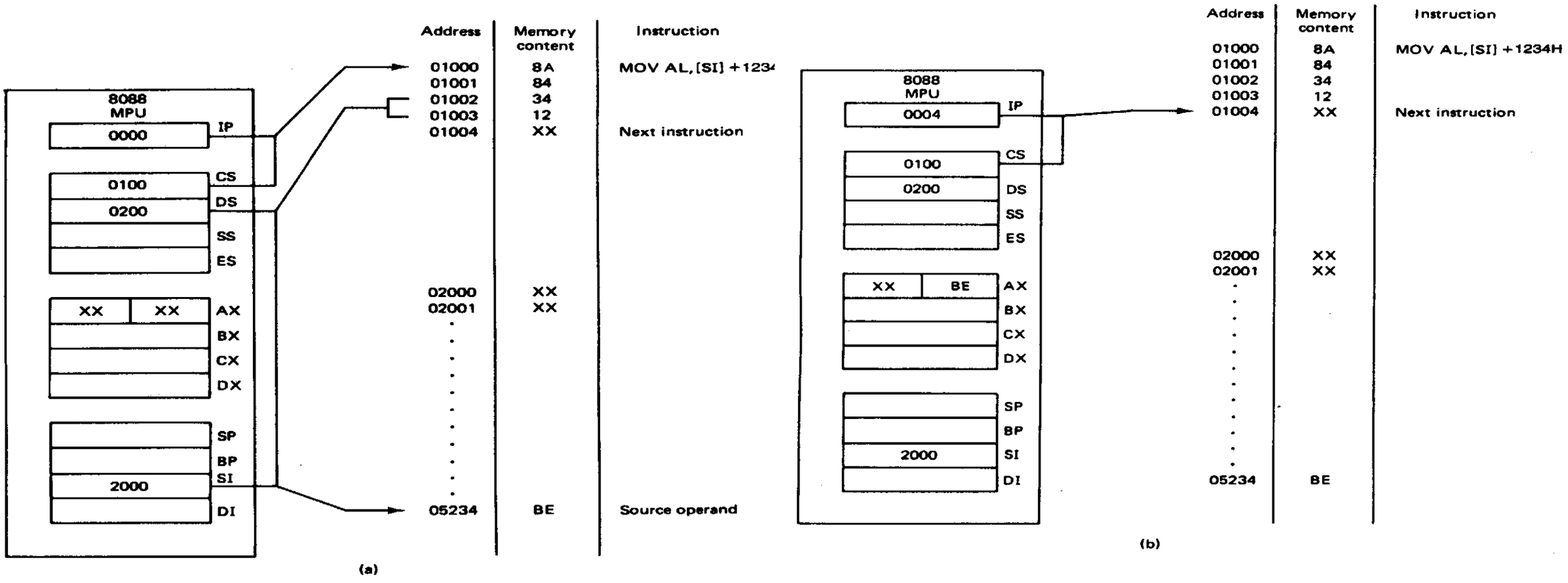
$(CH) \leftarrow (MA + 1)$

MOV AX, [SI+2000]

MOV AL, [DI+3000]

Indexed Addressing

MOV AL, [SI] +1234H



5. Based Index Addressing

In this the effective address is the sum of base register and displacement. In Based Index Addressing, the effective address is computed from the sum of a base register (BX or BP), an index register (SI or DI) and a displacement. MOV AL, [BP+ 0100]

Example: MOV DX, [BX + SI + 0AH]

Operations:

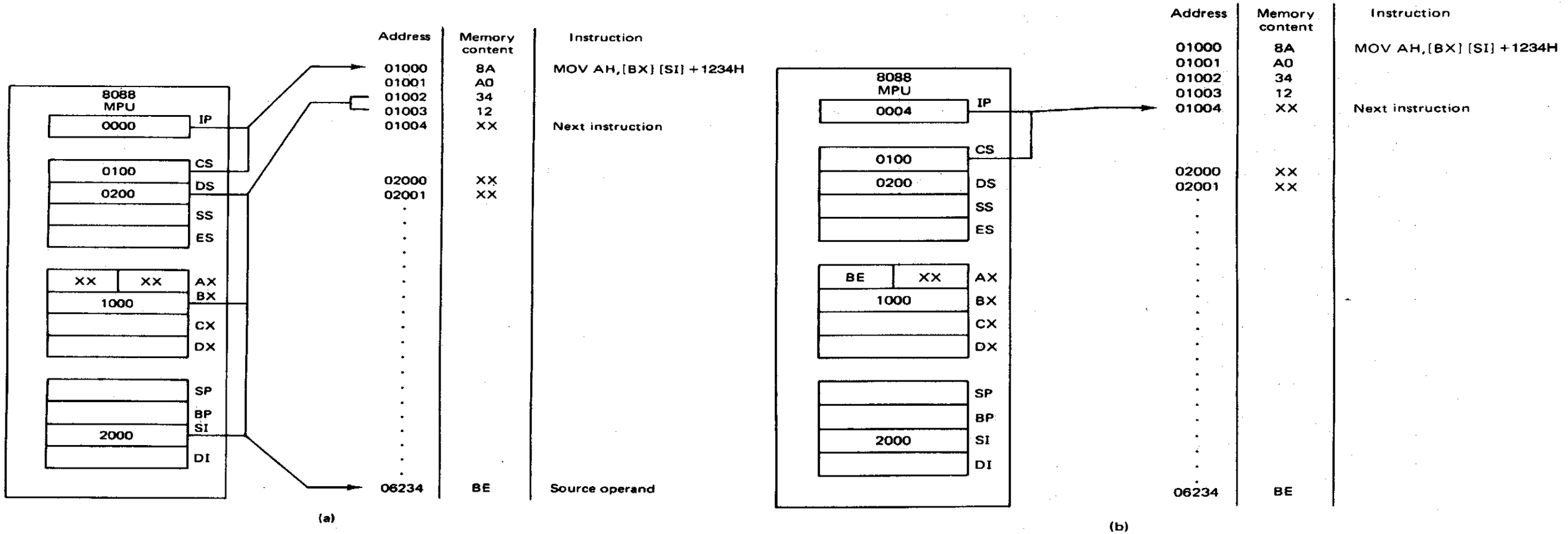
$000A_H \leftarrow 0A_H$ (Sign extended)
EA = (BX) + (SI) + $000A_H$
BA = (DS) x 16_{10}
MA = BA + EA

(DX) ← (MA) or,
(DL) ← (MA)
(DH) ← (MA + 1)

Based indexed displacement mode – In this type of addressing mode the effective address is the sum of index register, base register and displacement.
MOV AL, [SI+BP+2000]

Based-Indexed Addressing Modes

MOV AH, [BX][SI] +1234H



6. String Addressing

Employed in string operations to operate on string data. The effective address (EA) of source data is stored in SI register and the EA of destination is stored in DI register. Segment register for calculating base address of source data is DS and that of the destination data is ES.

Example: MOVS BYTE

Operations:

Calculation of source memory location:
EA = (SI) BA = (DS) x 16_{10} MA = BA + EA

Calculation of destination memory location:
EA_E = (DI) BA_E = (ES) x 16_{10} $MA_E = BA_E + EA_E$

(MAE) ← (MA)

If DF = 1, then (SI) ← (SI) – 1 and (DI) = (DI) - 1
If DF = 0, then (SI) ← (SI) +1 and (DI) = (DI) + 1

String Addressing

Employed in string operations to operate on string data.

The effective address (EA) of source data is stored in SI register and the EA of destination is stored in DI register.

Segment register for calculating base address of source data is DS and that of the destination data is ES

Example: MOVS BYTE
Operations:

Calculation of source memory location:
EA = (SI) BA = (DS) x 16_{10} MA = BA + EA

Calculation of destination memory location:
EA_E = (DI) BA_E = (ES) x 16_{10} $MA_E = BA_E + EA_E$

(MAE) ← (MA)

If DF = 1, then (SI) ← (SI) – 1 and (DI) = (DI) - 1
If DF = 0, then (SI) ← (SI) +1 and (DI) = (DI) + 1

Note : Effective address of the Extra segment register

Group III :
Addressing modes for I/O ports

Data Transfer Instructions (I/O)

Mnemonics: IN, OUT …

IN AL, [DX]	I/O $PORT_{addr}$ = (DX) (AL) ← (PORT Buffer) ; 1 Byte
IN AX, [DX]	I/O $PORT_{addr}$ = (DX) (AX) ← (PORT Buffer); 2 Byte, 1 Word
OUT [DX], AL	I/O $PORT_{addr}$ = (DX) (PORT Buffer) ← (AL); 1 Byte
OUT [DX], AX	I/O $PORT_{addr}$ = (DX) (PORT Buffer) ← (AX); 2 Byte, 1Word

- I/O işlemlerinde sadece DX, AX, AL register'ları kullanılır.
- DX: I/O Port Buffer adres, 16bittir.
- Fiziksel adres DX *10h
- AX, AL : Veri

- Hatırlatma:
- AX: Data Register, MUL, DIV, I/O işlemlerinde register olarak.
- BX: Data Register, 20 bitlik fiziksel adres hesaplamada DS ve ES ile birlikte offset Register.
- CX: Data Register, Counter
- DX: Data Register, MUL, DIV, işlemlerinde register olarak. I/O port buffer adres bilgisi.
- SI, DI: 20 bitlik fiziksel adres hesaplamada DS ve ES ile birlikte offset Register.
- BP, SP:20 bitlik fiziksel adres hesaplamada SS ile birlikte offset Register.

Direct I/O port Addressing

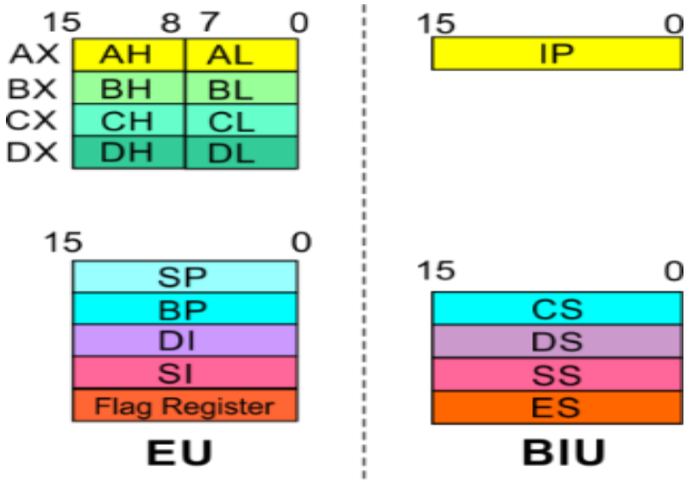
These addressing modes are used to access data from standard I/O mapped devices or ports. In direct port addressing mode, an 8-bit port address is directly specified in the instruction.

Example: IN AL, [09H]
Operations: $PORT_{addr} = 09_H$
(AL) ← (PORT) ,Content of port with address 09_H is moved to AL register

In indirect port addressing mode, the instruction will specify the name of the register which holds the port address. In 8086, the 16-bit port address is stored in the DX register.

Example: OUT [DX], AX
Operations: $PORT_{addr} = (DX)$
(PORT) ← (AX) , Content of AX is moved to port whose address is specified by DX register.

Direct I/O port Addressing
Indirect I/O port Addressing



These addressing modes are used to access data from standard I/O mapped devices or ports.

In direct port addressing mode, an 8-bit port address is directly specified in the instruction.

Example: IN AL, [09H]
Operations: $PORT_{addr} = 09_H$
$(AL) \leftarrow (PORT)$

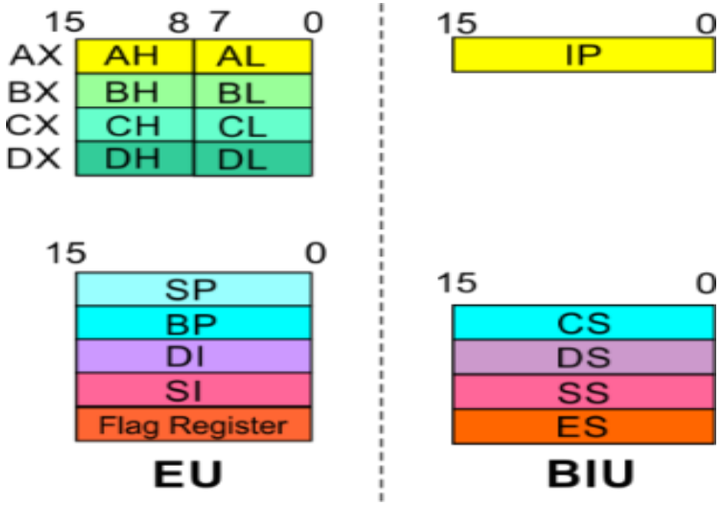
Content of port with address 09_H is moved to AL register

In indirect port addressing mode, the instruction will specify the name of the register which holds the port address. In 8086, the 16-bit port address is stored in the DX register.

Example: OUT [DX], AX
Operations: $PORT_{addr} = (DX)$
$(PORT) \leftarrow (AX)$
Content of AX is moved to port whose address is specified by DX register.

Group IV : Relative and Implied Addressing mode

Relative Addressing



In this addressing mode, the effective address of a program instruction is specified relative to Instruction Pointer (IP) by an 8-bit signed displacement.

Example: JZ 0AH

Operations:

$000A_H \leftarrow 0A_H$ (sign extend)

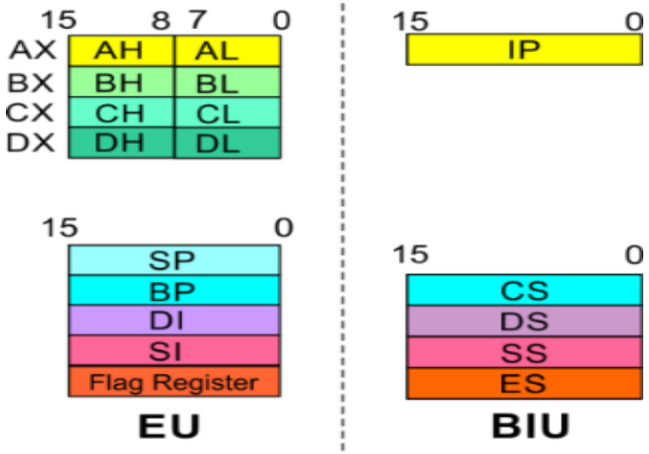
If ZF = 1, then

$EA = (IP) + 000A_H$
$BA = (CS) \times 16_{10}$
$MA = BA + EA$

If ZF = 1, then the program control jumps to new address calculated above.

If ZF = 0, then next instruction of the program is executed.

Implied Addressing



Instructions using this mode have no operands. The instruction itself will specify the data to be operated by the instruction.

Example: CLC

This clears the carry flag to zero.

The PTR Operator

- MOV AL,[20h] ; 8 bit data copied into AL
- MOV AX,[20h] ; 16 bit data copied into AX
- INC [20h] ; 8 bit or 16 bits incremented"?

- Byte or word or doubleword?
- To clarify we use the PTR operator
  - INC BYTE PTR [20h]
  - INC WORD PTR [20h]
  - INC DWORD PTR [20h]

Bellek Erişim Uygulamalar

Example

20100H konumundan başlayan 16 byte bellek bloğunun içeriğini 20120H'den başlayan başka bir bellek bloğuna kopyalayın.

```
        MOV AX, 2000H
        MOV DS, AX
        MOV SI, 100h
        MOV DI, 120h
        MOV CX, 16
NXTPT:  MOV, AL, [SI]
        MOV [DI], AL
        INC SI
        INC DI
        DEC CX
        JNZ NXTPT
```

Örnek: Range Calculate – Memory Capacity

Example: 64Kbyte x 8 bit belleğin

a) Adres hattı sayısı kaç adettir? Adres hatlarını indisleyin

b) Data hattı sayısı kaç adettir?

c) Belleğin fiziksel başlangıç ve bitiş adresini belirleyiniz.

Yanıt:

a) 64Kbyte=2^6*2^{10} byte=2^{16} byte, belleğin adres hattı sayısı=16 adettir.

Adres hatları indisi: A15, A14, A13, ..., A2, A1, A0

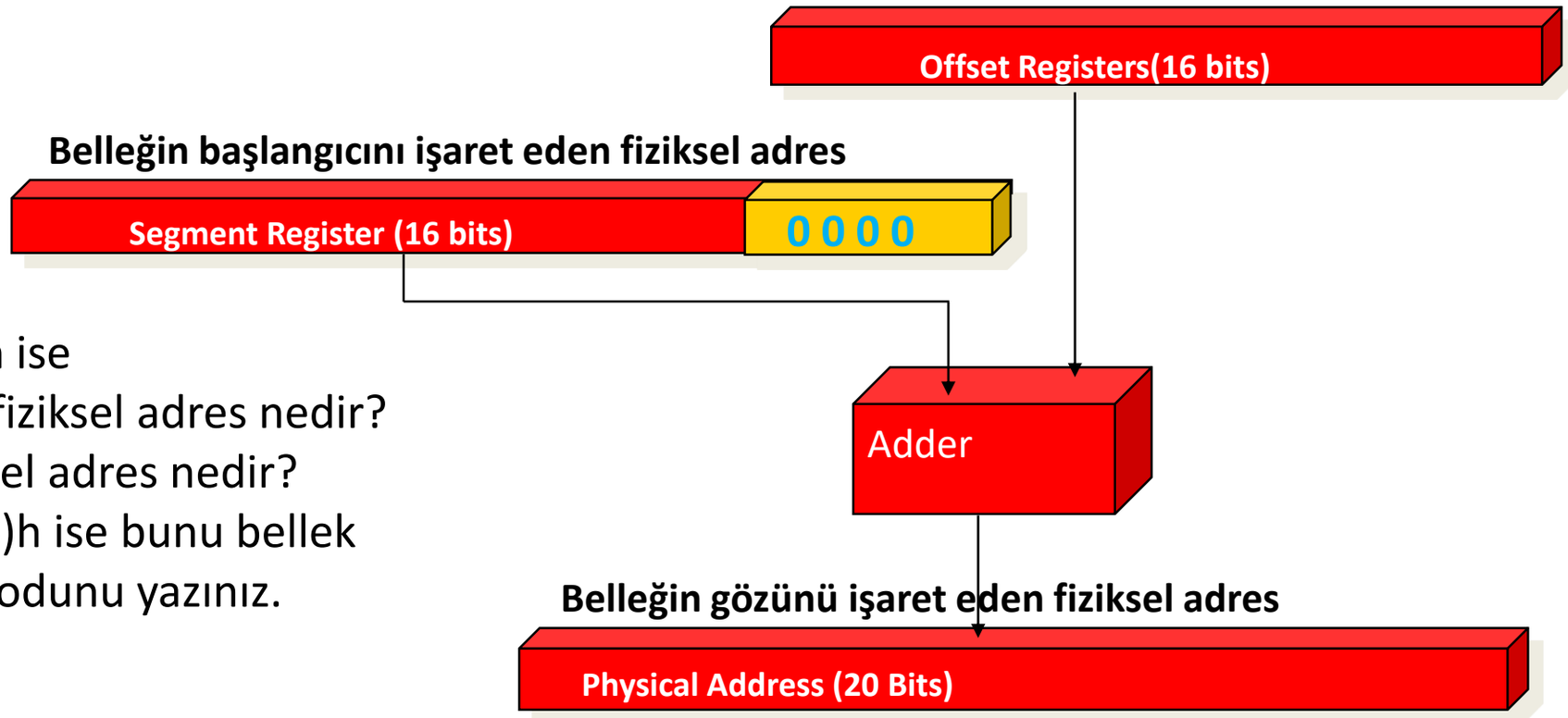
b) Belleğe 8bit yani byte olarak yazar okur. Data bus hat sayısı 8 adettir. Data hatları indisi: D7, D6, D5, ..., D1, D0

c) Belleğin başlangıç adresi = (0000 0000 0000 0000)b=(0000)h

Belleğin bitiş adresi = (1111 1111 1111 1111)b=(FFFF)h

- Bir belleğin bitiş adresi, (1100 1111 1111 1111 1111)b ise bunun hex değeri nedir? (CFFFF)h

Soru: Bellek Erişim Adresinin Hesaplanması



Soru:

DS: (A000)h, Offset Register: (A000)h ise

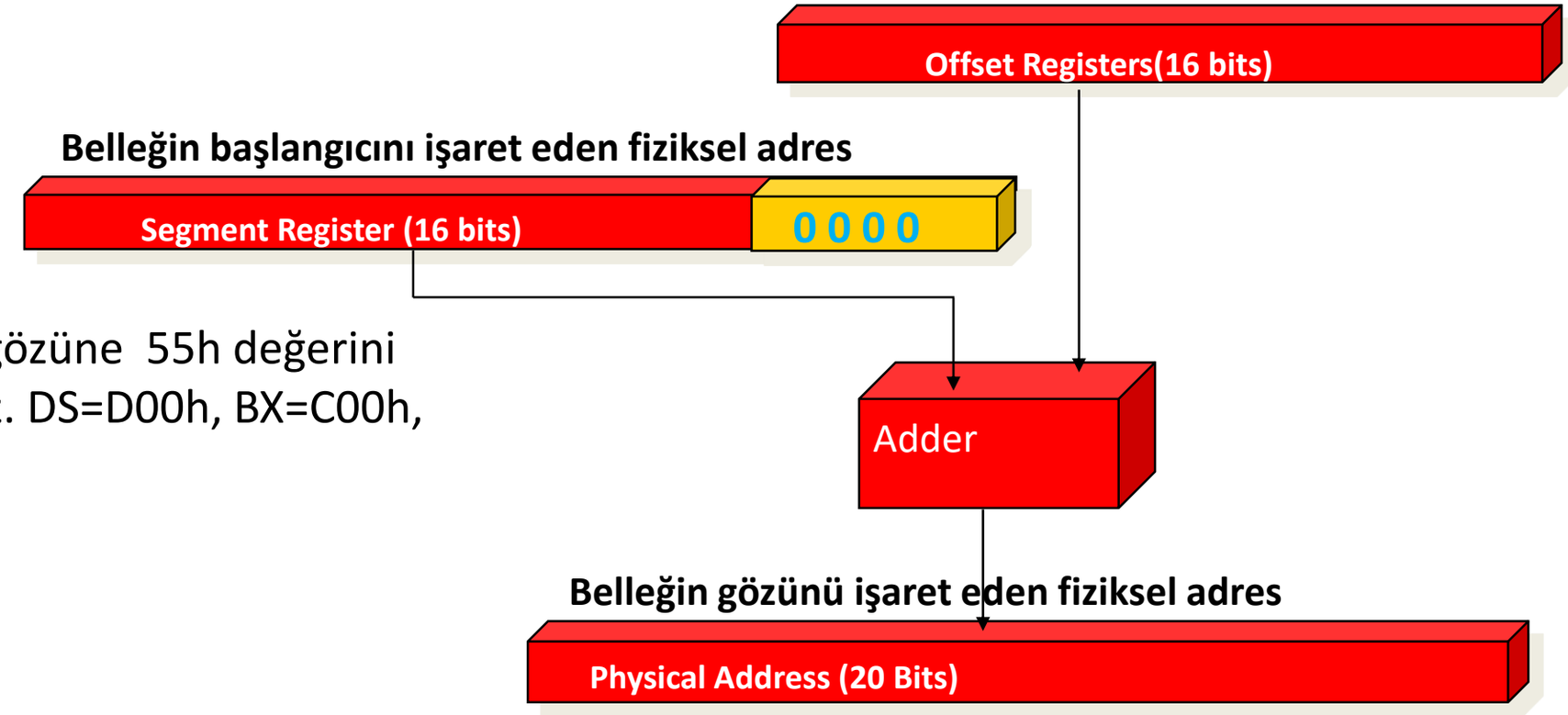
a) Belleğin başlangıcını işaret eden fiziksel adres nedir?
b) Belleğin gözünü işaret eden fiziksel adres nedir?
c) AL, data register'ındaki değer (88)h ise bunu bellek gözüne transfer eden assemble kodunu yazınız.

Yanıt:

a) Belleğin başlangıcını işaret eden 20 bit fiziksel adres: (A0000)h
b) Belleğin gözünü işaret eden fiziksel adres: (A0000)h + (A000)h=(AA000)h

```
MOV AX, 0A000h
MOV DS, AX
MOV AL, 88h
MOV DI, 0A000h
MOV [DI], AL
```

Soru: Bellek Erişim Adresinin Hesaplanması

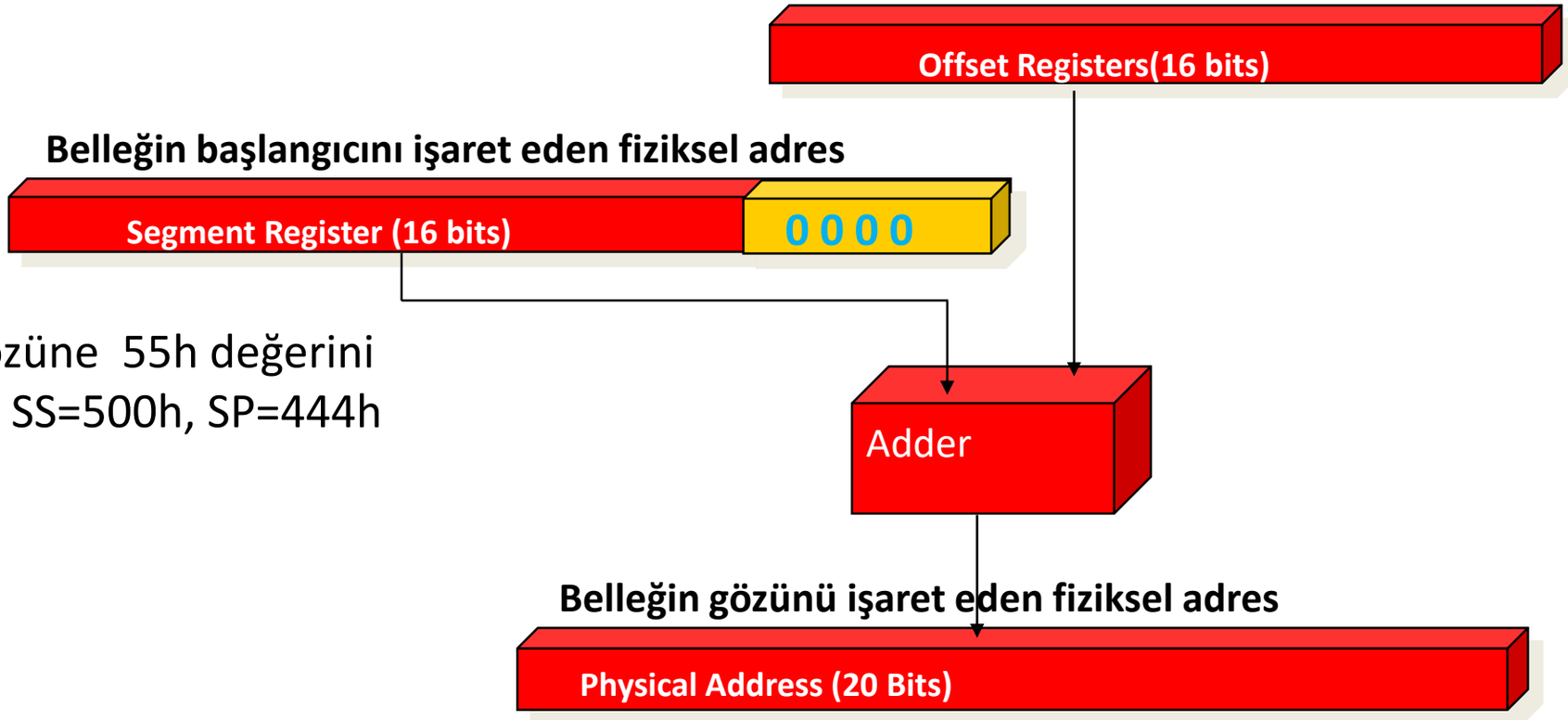


Soru:
Fiziksel adresi DCCCh olan belleğin gözüne 55h değerini yazan assemble programının yazınız. DS=D00h, BX=C00h, SI=C0h alınacaktır.

Yanıt:

```
MOV AX, 0D00h
MOV DS, AX
MOV  BX, 0C00h
MOV SI, 0C0h
MOV [BX + SI+ 0Ch], 55h
```

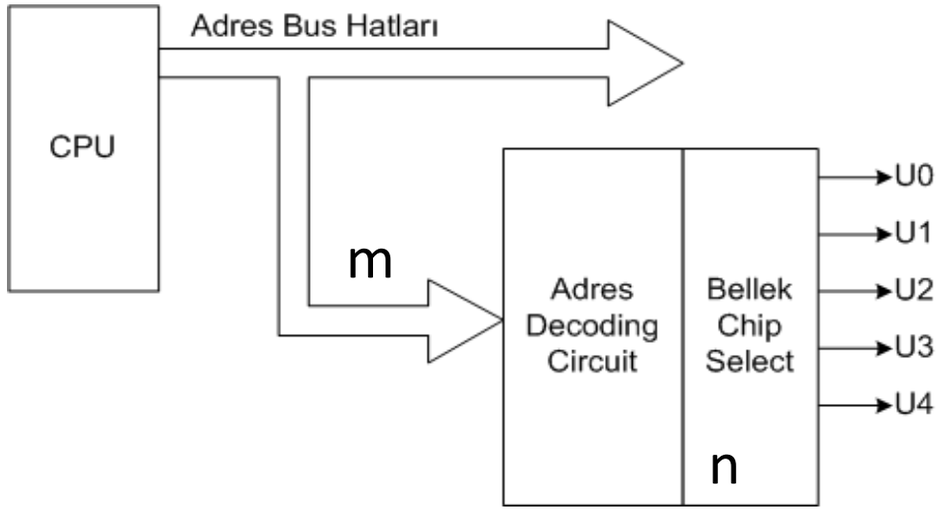
Soru: Bellek Erişim Adresinin Hesaplanması



Soru:
Fiziksel adresi 5444h olan belleğin gözüne 55h değerini yazan assemble programının yazınız. SS=500h, SP=444h alınacaktır.

Yanıt:
MOV AX, 500h
MOV SS, AX
MOV SP, 444h
MOV SS:[SP], 55h

Örnek 1.1

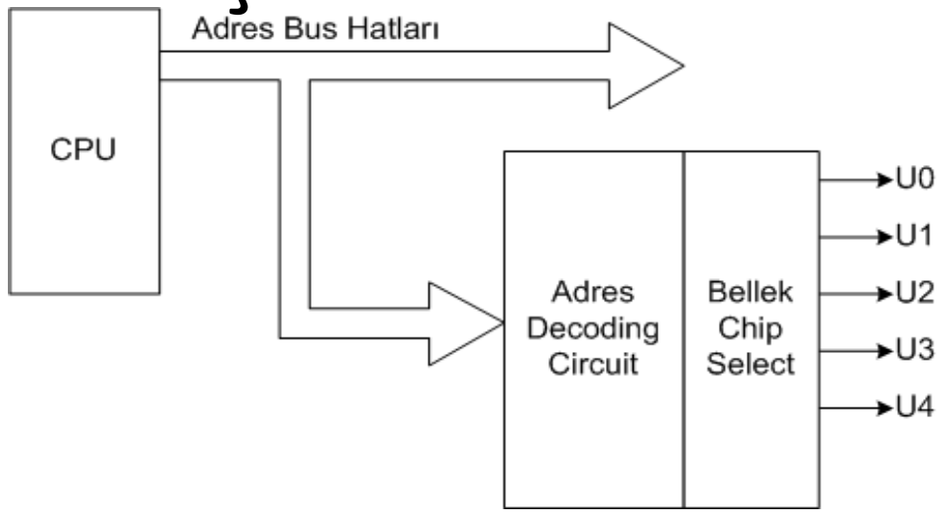


- U0=3KByte, U1=4Kbyte, U2=5Kbyte, U3=7Kbyte, U4=8Kbyte

a) CPU'dan adres decoding circuit için kaç adet adres bus hattına ihtiyaç vardır?

- CPU'dan adres decoding circuit devresi için çıkacak adres hatt sayısı, m ise, $2^m \geq n$ dir. Burada n: bellek ve I/O birimlerinin toplam sayısıdır.
- Burada n=5 ise , m=3 olur. O halde adres dekoding devresinin girişine CPU'dan 3 adet adres bus hattı gelirse, seçilecek maksimum bellek sayısı $n_{max}= 2^3=8$ dir.

Örnek 1.2: Adres Dekoding Devresinin Girişi



- b) Adres dekoding lojik devresinin giriş durumlarına göre çıkış durumlarını belirleyiniz.
- Aynı anda bir adet bellek ya da I/O birim seçilir. Seçilmez ise çakışma ve üst üste yazılma olur. Veri, aynı anda iki belleğin gözüne ya yazılır ya da okunur.

adr2	adr1	adr0	Bellek - I/O	Durumu
0	0	0	U0	Aktif
0	0	1	U1	Aktif
0	1	0	U2	Aktif
0	1	1	U3	Aktif
1	0	0	U4	Aktif
1	0	1	U5	Yedek
1	1	0	U6	Yedek
1	1	1	U7	Yedek

Örnek 1.3

c) Herbir belleğin adres hattı sayısını belirleyiniz ve indisleyiniz.

- Bellek gözlerine erişime yönelik adres hat sayısı belirlemede 2^n li tanımlama kullanıldığından,
- U0=3KByte yerine U0=4Kbyte alınır. Yazılım kodu oluşturulurken 3Kbyte'lık alan tanımlanır.
- U1=4Kbyte alınır.
- U2=5Kbyte yerine U2=8Kbyte alınır. Yazılım kodu oluşturulurken 5Kbyte'lık alan tanımlanır.
- U3=7Kbyte yerine U3=8Kbyte alınır. Yazılım kodu oluşturulurken 7Kbyte'lık alan tanımlanır.
- U4=8Kbyte alınır.
- Bu durumda
- U0=4Kyte=$2^2*2^{10} = 2^{12}$ byte bulunur. Adres hattı sayısı=12, indisleme: A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0
- U1=4Kyte=$2^2*2^{10} = 2^{12}$ byte bulunur. Adres hattı sayısı=12, indisleme: A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0
- U2=8Kyte=$2^3*2^{10} = 2^{13}$ byte bulunur. Adres hattı sayısı=13, indisleme: A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0
- U3=8Kyte=$2^3*2^{10} = 2^{13}$ byte bulunur. Adres hattı sayısı=13, indisleme: A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0
- U4=8Kyte=$2^3*2^{10} = 2^{13}$ byte bulunur. Adres hattı sayısı=13, indisleme: A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0

Örnek 1.4

d) CPU’dan Adres dekoding devresine gelen adres hatlarını indisleyiniz.

- CPU’dan Adres dekoding devresine gelen adres hatlarını indisleme, maksimum bellek indisinden sonra devam eder.
- Maksimum bellek indisi=A12 olduğundan, ADC girişine 3 adet adres indisi geldiğinden
- adr0=A13
- adr1=A14
- adr2=A15 olur.

e) CPU’dan çıkan adres hatlarını indislemesini yapınız.

- A15, A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0 olur.

f) CPU’dan çıkan toplam adres hat sayısını bulunuz. Toplam hat sayısı, k= CPU’dan çıkan adres hatlarından maksimum indis + 1=15+1 =16 dır.

g) CPU’nun adresleme kapasitesi kaç kbyte dır?

- CPU’nun adresleme kapasitesi=2^k byte=2^16byte=(2^6)*(2^10)=64Kbyte eder.

Örnek 1.5

h) Belleklerin başlangıç adresini gösteren 16 bit Segment register içeriklerini belirleyiniz.

20 bit fiziksel adres, 16 bit segment register içeriğinin (10)h ile çarpılmasından (sağ tarafa 4 bit 0 eklenmesi) elde edildiğinden A3, A2, A1, A0 göz önüne alınmaz.

Tüm bitler sıfırlanır. Sadece adres dekoding devresine gelen adres hatlarının durumları yazılır.

adr2-A15	adr1-A14	adr0-A13	Bellek - I/O	Durumu
0	0	0	U0	Aktif
0	0	1	U1	Aktif
0	1	0	U2	Aktif
0	1	1	U3	Aktif
1	0	0	U4	Aktif
1	0	1	U5	Yedek
1	1	0	U6	Yedek
1	1	1	U7	Yedek

	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	Hex
U0 segment	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(0000)h
U1 segment	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	(0200)h
U2 segment	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	(0400)h
U3 segment	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	(0600)h
U4 segment	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	(0800)h

Örnek 1.6

i) Herbir belleğin offset register aralığını hesaplayınız.

	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Hex
U0 start	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**(0000)h**
U0 stop	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**(0FFF)h**
U1 start	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**(0000)h**
U1 stop	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**(0FFF)h**
U2 start	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**(0000)h**
U2 stop	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**(1FFF)h**
U3 start	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**(0000)h**
U3 stop	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**(1FFF)h**
U4 start	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**(0000)h**
U4 stop	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**(1FFF)h**

Örnek 1.7

i) Herbir belleğin 20 bit fiziksel dares aralığını belirleyiniz.

	A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Hex
U0 start	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(00000)h
U0 stop	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	(00FFF)h
U1 start	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	(02000)h
U1 stop	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	(02FFF)h
U2 start	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(04000)h
U2 stop	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	(05FFF)h
U3 start	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	(06000)h
U3 stop	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	(07FFF)h
U4 start	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(08000)h
U4 stop	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	(09FFF)h

Örnek 1.8

j) Bir belleğin 20bit fiziksel başlangıç adresi (06000)h, bitiş adresi (07FFF)h ise belleğin kapasitesi kaç Kbyte'dır?

Belleğin kapasitesi (Byte) bulmak için Bitiş adresi – Başlangıç adres + 1 , formulü kullanılı. Bulunan Hex değerinin indislerinden 2^k ların toplamı ile Byte olarak hesaplanır.

Bitiş adresi – Başlangıç adres + 1 = (07FFF)h – (06000)h +1 = (1FFF)h +1 = (2000)h

Bitiş adresi – Başlangıç adres + 1 =(2000)h=(0010 0000 0000 0000)b

1'in olduğu indis, k=12

Belleğin kapasitesi (Byte)= 2^k=2^12=4Kbyte olarak hesaplanır.

Örnek 1.9

k) U4 belleği data segment ise bu belleğin 7532 inci gözüne (AA)h değerini yazan kodu yazınız. Yazılan bu değer tanımla bellek aralığı içerisinde midir? Offset indis register olarak BX yerine hangi register'lar kullanılır?

Mov Ax, 0800h

Mov DS, Ax

Mov Bx, 1D6Ch

Mov [Bx], 0AAh

A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Hex
1	1	1	0	1	0	1	1	0	1	1	0	0	(1D6C)h

(7532)d=(1 1101 0110 1100)b=(1D6C)h

7532	4096	2^12
3436	2048	2^11
1388	1024	2^10
364	256	2^8
108	64	2^6
44	32	2^5
12	8	2^3
4	4	2^2
0		

- U4 Belleğin indis aralığı i) şıkkkında hesaplanmıştı ve bu aralık (0000)h ile (1FFF)h olarak bulunmuştu. 7532 desimal değerinin Hex karşılığı olan (1D6C)h bu aralığın içerisinde olduğu görülmektir.
- Offset indis register olarak BX yerine SI, DI kullanılır.

Örnek 1.10

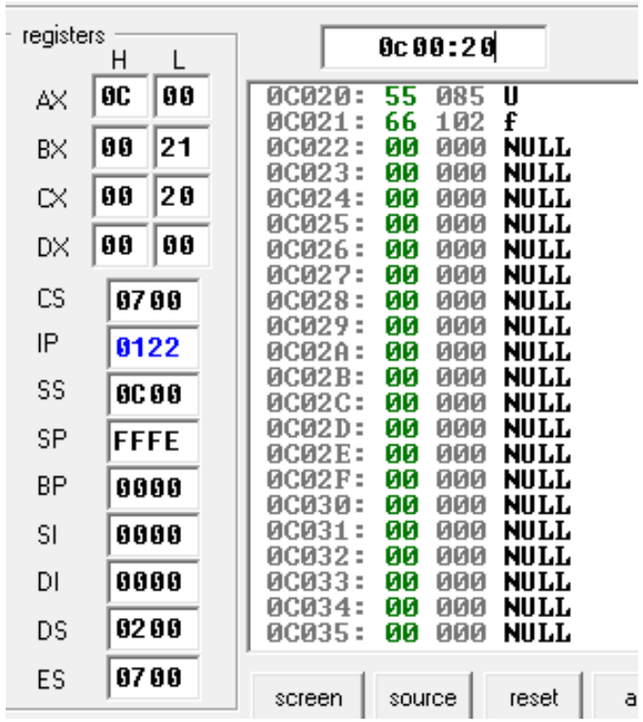
l) Aşağıda verilen kod yazılımını açıklayınız. 20 bitlik fiziksel adresini heaplayınız. Kod hangi belleği tanımlamaktadır? Indis register aralığı ilgili belleğin tanımlı aralığında mı?

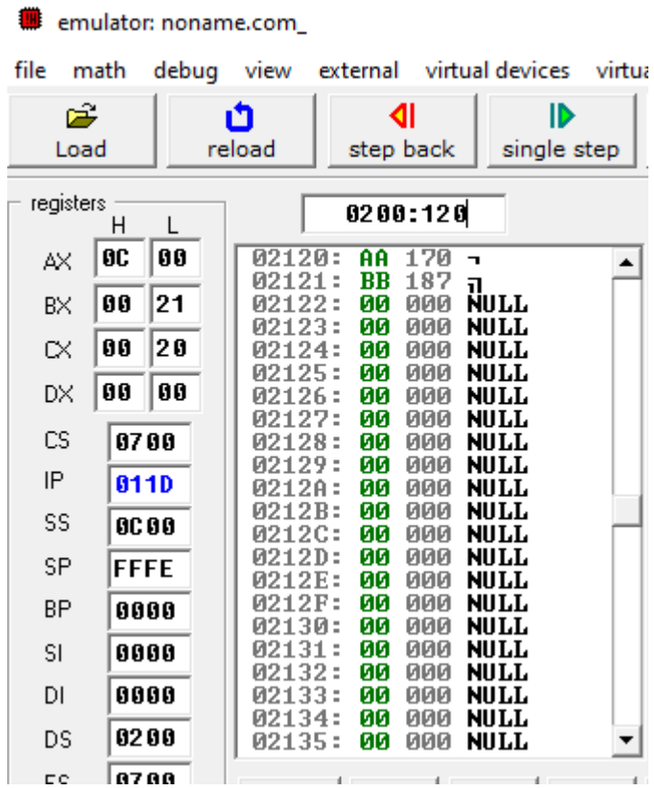
```
Mov Ax, 0400h
Mov ES, Ax
Mov Bx, 0DCCh
Mov [Bx], 0AAh
```

(0DCC)h=(0000 1101 1100 1100)b=2^11+2^10+2^8+2^7+2^6+2^3+2^2
(0DCC)h=2048+1024+256+128+64+8+4=(3532)d

- 20 bitlik fiziksel adres= (0400)*10h+ (0DCC)h=(04000)h + (0DCC)h=(04DCC)h
- Segment Register içeriğinden kodun U2 belleği tanımladığı görülmektedir.
- Indis register aralığı (0DCC)h değeri, ilgili U2 belleğin tanımlı olan (0000)h – (1FFF)h aralığı içerisndedir.

```
org 100h
mov ax,200h
Mov DS, ax
Mov Bx, 120h  ;index
mov [Bx], 0AAh


mov ax,0c00h
Mov SS, ax
Mov Bx, 120h ; index
mov SS:[Bx], 55h


endp
```





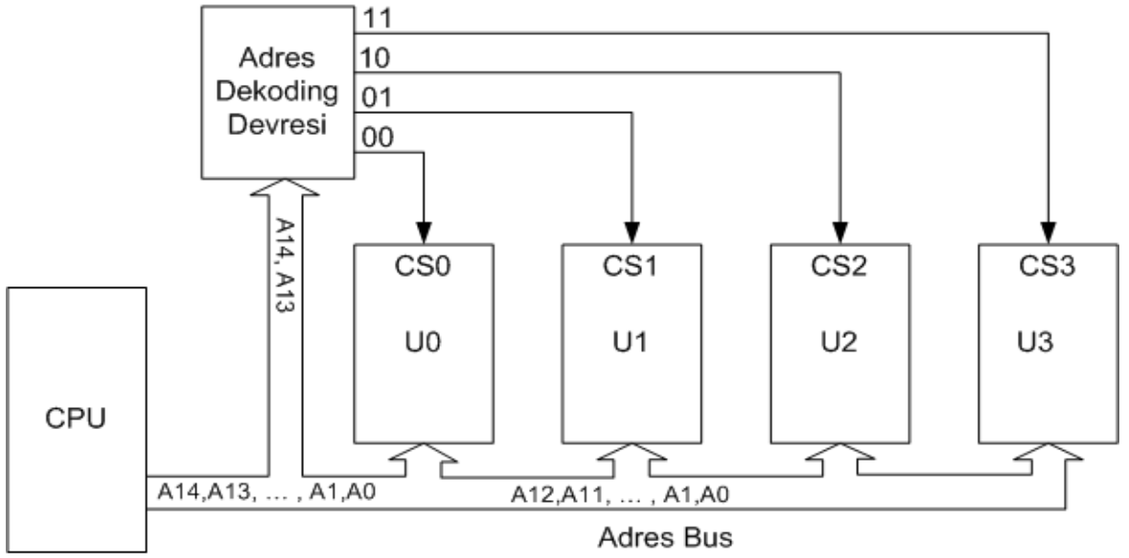
Data segment registerında gösterilen belleğin fiziksel adresi nedir?
Fiziksel adres=(DS)*10h
Fiziksel ades=200h*10h=2000h
Segment register'lara doğrudan veri yazılamaz. Data register'lar üzerinden yazılır.
Bellek gözünün adresi=2000h+120h=2120h
Bu bellek gözüne AA datası kayıt edilir.
Emulator: 200:0120;
DS:200h, indis: 120h

SS ile gösterilen belleğin başlangıç adres nedir?
C00h*10h=C000h

Örnek-4: Memory Mapping

- 4 belleğin başlangıç adresleri segment register'larda verilmiştir. Bellek kapasiteleri verildiğine göre bellekleri fiziksel başlangıç adreslerini ve segment register içerik değerlerini bulunuz.
- CS, ROM bellek: 8Kbyte
- DS, RAM bellek: 8Kbyte
- SS, RAM bellek: 8Kbyte
- ES, RAM bellek: 8Kbyte
- Bellek mapping görünümünü çiziniz.

Örnek-4: Memory Mapping



- CS, ROM bellek: U0= 8Kbyte=2^13byte; Adres Bus İndis: A12, A11, … , A1, A0
- DS, RAM bellek: U1= 8Kbyte=2^13byte ; Adres Bus İndis: A12, A11, … , A1, A0
- SS, RAM bellek: U2= 8Kbyte=2^13byte ; Adres Bus İndis: A12, A11, … , A1, A0
- ES, RAM bellek: U3= 8Kbyte=2^13byte ; Adres Bus İndis: A12, A11, … , A1, A0
- Adres Dekoding devresi girişi, n=2^m; Burada n: bellek sayısı, m= CPU'dan çıkan ve adres dekoding devresine giriş yapan adres hattı sayısıdır. 4=2^m; m=2 bulunur.
- Maksimum Adres Bus indis: A12 olduğundan adres dekoding devresine gelen adres bus hatları indisi: A14, A13 olur.
- CPU'dan çıkan toplam adres hatları indisleri: A14, A13, A12, …, A1, A0
- CPU'dan çıkan toplam adres hattı sayısı=15
- CPU bellek adresleme kapasitesi= 2^15=32 Kbyte

Örnek 4: Adres Dekoding Devresi

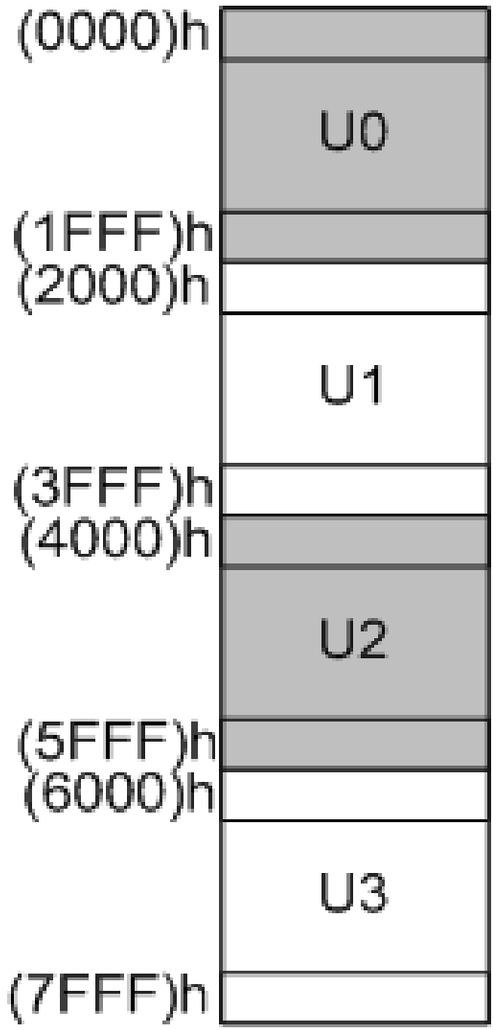
- Adres dekoding devresine gelen adres bus hatları indisi: A14, A13 ise, doğruluk tablosunu oluşturun. Hangi giriş durumunda hangi bellek seçilir.

Doğruluk Tablosu					
Girişler		Çıkışlar			
A14	A13	U0	U1	U2	U3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

A14	A13	Bellek Seçim
0	0	U0
0	1	U1
1	0	U2
1	1	U3

Örnek-4: Belleklerin fiziksel başlangıç ve bitiş adresleri

	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Fiziksel Adresi	Segment Register (16 Bit)	Segment Register
U0 Başlangıç adresi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(0000)h	(0000)h	CS
U0 Bitiş adresi	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	(1FFF)h		
U1 Başlangıç adresi	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	(2000)h	(0200)h	DS
U1 Bitiş adresi	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	(3FFF)h		
U2 Başlangıç adresi	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	(4000)h	(0400)h	SS
U2 Bitiş adresi	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	(5FFF)h		
U3 Başlangıç adresi	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	(6000)h	(0600)h	ES
U3 Bitiş adresi	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	(7FFF)h		



- Bellek gözü (507F)h adresi hangi bellektedir? U2
- Bu belleğin başlangıç adresi hangi segment register'ın içeriğindedir. SS
- Segment register içeriği nedir? (0400)h
- İndis Register değeri nedir? İndis register değeri= bellek gözünün adresi – belleğin fiziksel başlangıç adresi= (507F)h-(4000)h=(107F)h

Örnek-4: Assembly Veri transfer

(507F)h adresindeki bellek gözüne (AA)h değerini yazan kodu oluşturunuz.

- MOV AX, 0400h
- MOV SS, AX
- MOV SI, 107Fh
- MOV SS:[SI], 0AAh

Örnek-5: Memory Mapping

- CS, ROM bellek: U0= 7Kbyte; 8Kbyte=2^13byte; Adres hattı sayısı=13adet; Adres Bus İndis: A12, A11, … , A1, A0
- DS, RAM bellek: U1= 12Kbyte; 16Kbyte=2^14byte; Adres hattı sayısı=14adet; Adres Bus İndis: A13, A11, … , A1, A0
- SS, RAM bellek: U2= 24Kbyte; 32Kbyte=2^15byte; Adres hattı sayısı=15adet; Adres Bus İndis: A14, A11, … , A1, A0
- ES, RAM bellek: U3= 37Kbyte;64Kbyte=2^16byte; Adres hattı sayısı=16adet; Adres Bus İndis: A15, A11, … , A1, A0
- Adres Dekoding devresi girişi, n=2^m; Burada n: bellek sayısı, m= CPU'dan çıkan ve adres dekoding devresine giriş yapan adres hattı sayısıdır. 4=2^m; m=2 bulunur. 2 adet bellekleri seçmek adres bus hattı CPU'dan çıkacak.
- Maksimum Adres Bus indis: A15 olduğundan adres dekoding devresine gelen adres bus hatları indisi: A17, A16 olur.
- CPU'dan çıkan toplam adres hatları indisleri: A17, A16, A15, …, A1, A0
- CPU'dan çıkan toplam adres hattı sayısı=18
- CPU bellek adresleme kapasitesi= 2^18=256 Kbyte

Örnek-5

																		Fiziksel (20bit)	(16bit)	
A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0		Başlangıç - Bitiş adresleri	Segment Adresleri	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U0 ilk bellek gözü	(00000)h	(0000)h	CS
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	U0 son bellek gözü	(01FFF)h		
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U1 ilk bellek gözü	(10000)h	(1000)h	DS
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	U1 son bellek gözü	(13FFF)h		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U2 ilk bellek gözü	(20000)h	(2000)h	SS
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	U2 son bellek gözü	(27FFF)h		
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	U3 ilk bellek gözü	(30000)h	(3000)h	ES
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	U3 son bellek gözü	(3FFFF)h		

Not: doldurulmamış yerlere 0 yazılır.

Not: Başlangıç adreslerini hex'e çevrilir. Sağdan itibaren 4'er 4'er ayrılır.

(20CD0)h bellek gözüne 55h değerini yazın.

SS:2000h, Indis: (CD0)h

Mov Ax, 2000h

Mov SS, Ax

Mov SP, 0CD0h

Mov SS:[SP], 55h

Örnek-6: Memory Mapping

Segment register'ların içerikleri aşağıda verilmiştir.

- U0, CS: (0000)h
- U1, DS: (4000)h
- U2, SS: (A000)h
- U3, ES: (D000)h

- Herbir belleğin bitiş indis register değer (FFFF)h ise herbir belleğin fiziksel başlangıç ve bitiş adresleri ile kapasiterlerini bulunuz.
- Adres dekoding devresine gelen adres indislerini bulunuz.
- Herbir belleği seçen maantıksal dvre tablosunu oluşturun.
- Adres dekoding devresini çiziniz.

Usage Notes

- A lot of slides are adopted from the presentations and documents published on internet by experts who know the subject very well.
- I would like to thank who prepared slides and documents.
- Also, these slides are made publicly available on the web for anyone to use
- If you choose to use them, I ask that you alert me of any mistakes which were made and allow me the option of incorporating such changes (with an acknowledgment) in my set of slides.

Sincerely,

Dr. Cahit Karakuş

cahitkarakus@esenyurt.edu.tr