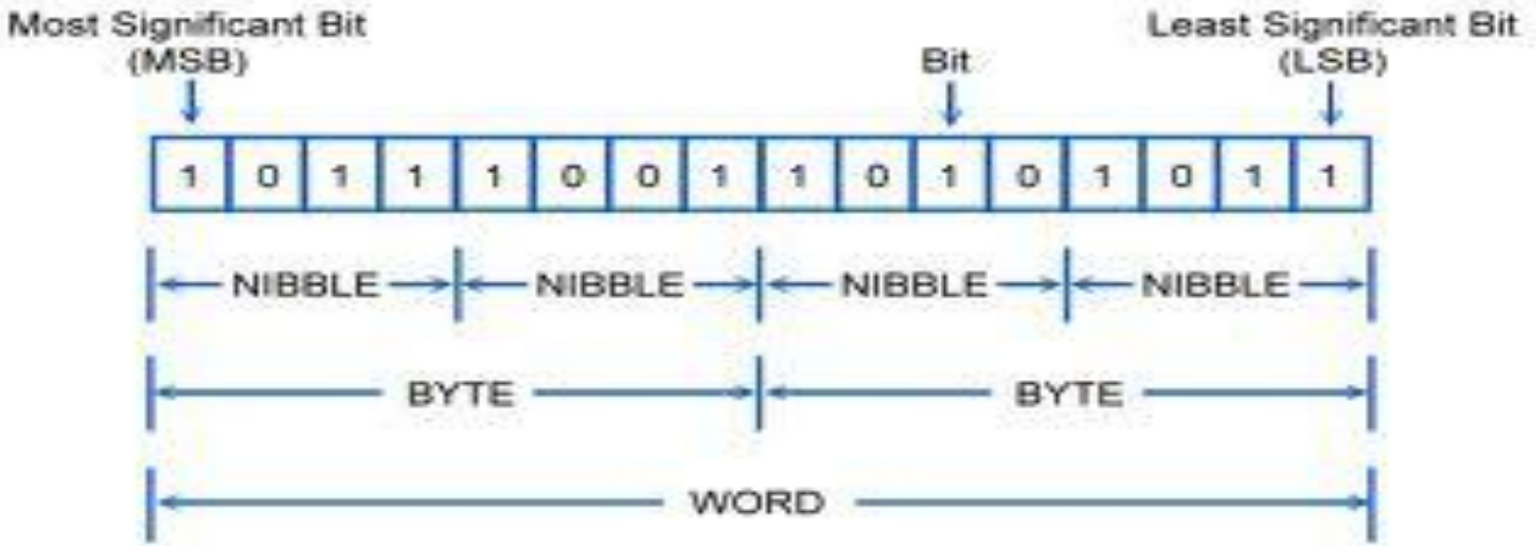
Microprocessor

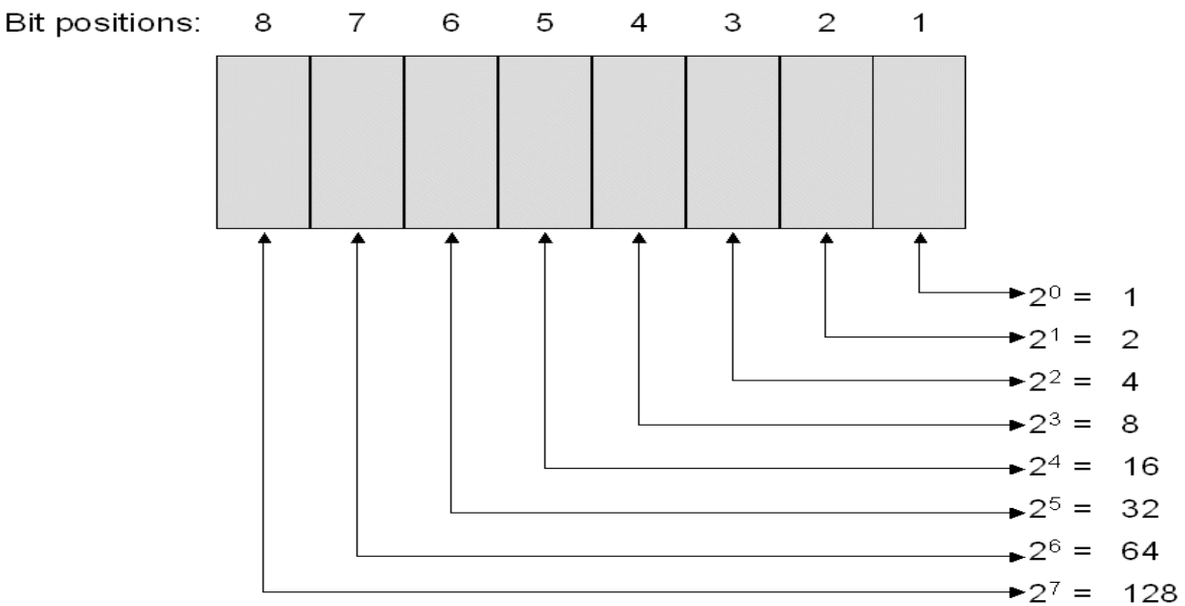
Assembly Addressing Modes

Dr. Cahit Karakuş
Esenyurt Üniversitesi

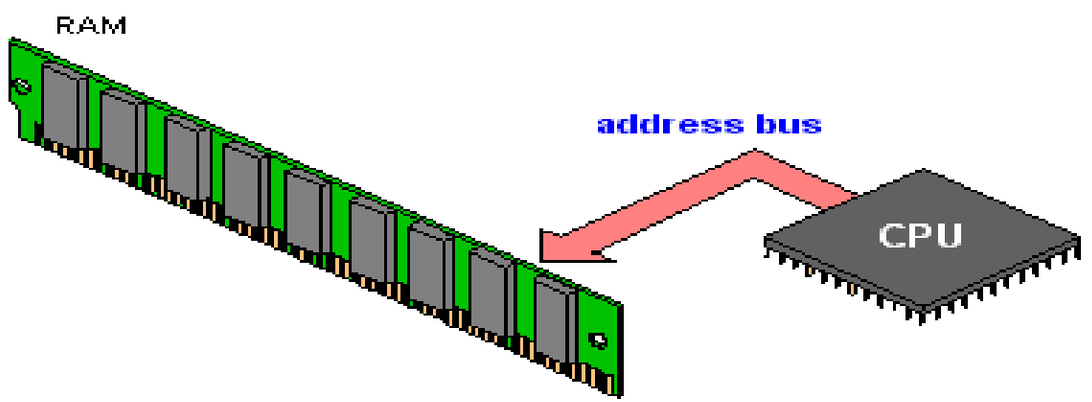
Data Representation Basics

Bits and Bytes

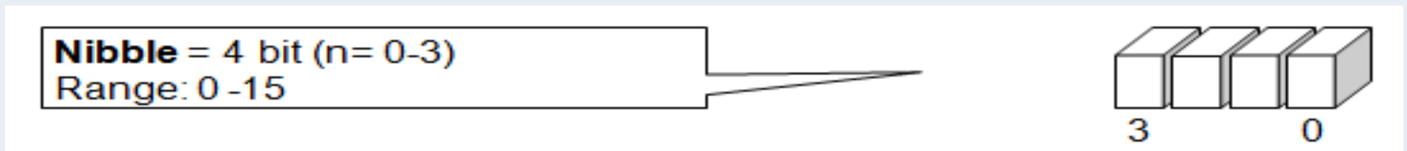
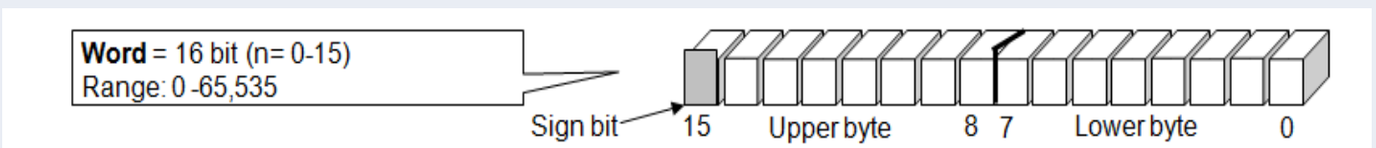
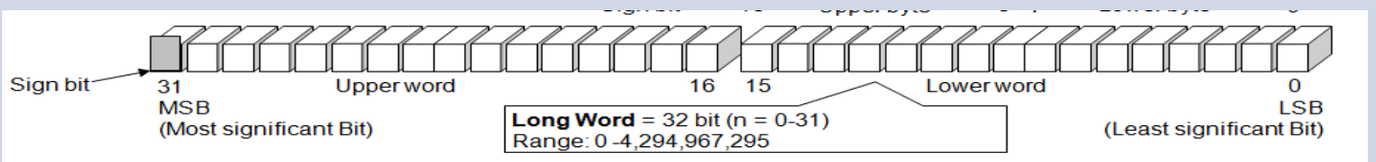




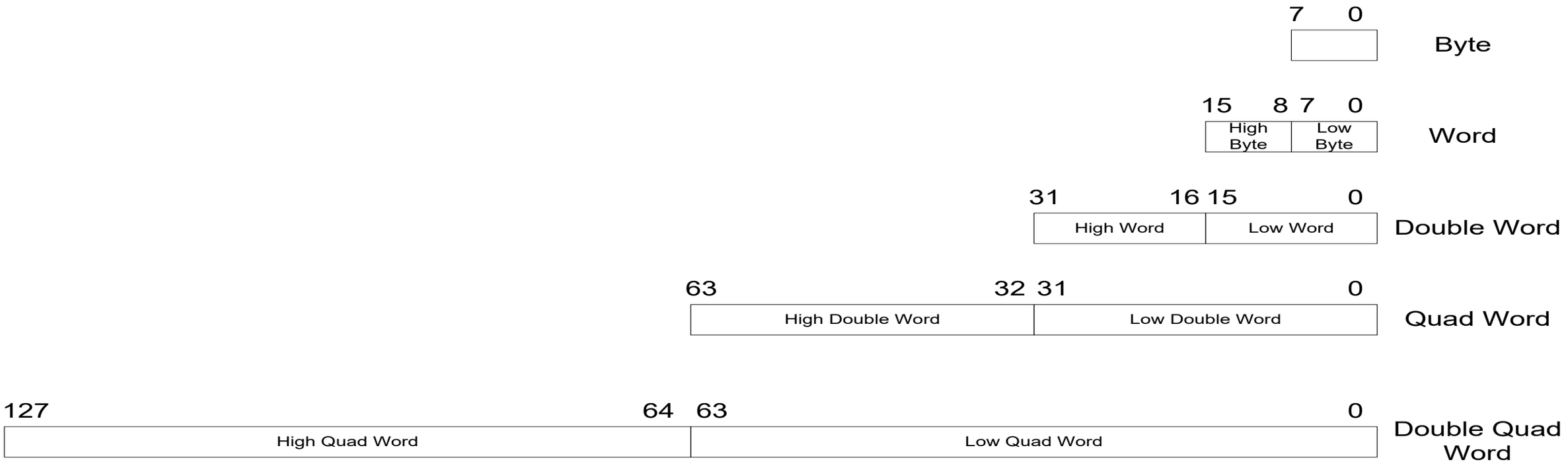
From Computer Desktop Encyclopedia



DATA SIZE

Nibble	**4 bit**	
Byte	**8 bit**	
Word	**16 bit**	
Long word	**32 bit**	

Data Types - size



Always take care of the type of data an instruction accesses!!!!

Bits as ASCII Codes

- ASCII: Each character is represented by a unique 8-bit code.
- 256 unique codes for special characters.
- Each key on the keyboard has an 8-bit equivalent. Each key represents a letter, number, and special character.

ASCII Character Codes

Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char	Hex	Char
20	(Space)	30	0	40	@	50	P	60	‘	70	p
21	!	31	1	41	A	51	Q	61	a	71	q
22	"	32	2	42	B	52	R	62	b	72	r
23	#	33	3	43	C	53	S	63	c	73	s
24	$	34	4	44	D	54	T	64	d	74	t
25	%	35	5	45	E	55	U	65	e	75	u
26	&	36	6	46	F	56	V	66	f	76	v
27	’	37	7	47	G	57	W	67	g	77	w
28	(	38	8	48	H	58	X	68	h	78	x
29	)	39	9	49	I	59	Y	69	i	79	y
2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

Negative Binary Numbers

- Byte: Defines 8 bits of data; refers to an 8-bit memory location.

- Example: db -4
  (4)d= (0000 0100)b
  (-4)d= Binary (4) -> invert all bits of the positive number +1 :
  (-4)d =1111 1011 +1 =1111 1100=FC

- Example: dW -4
  (4)d= 0000 0000 0000 0100
  (-4)d=Word(4) -> invert all bits of the positive number +1
  (-4)d=1111 1111 1111 1011 +1= 1111 1111 1111 1100=FFFC
  (-4)d=(0FFFC)h

Data Representation

Numbers

- If no definition is given, it means the decimal number system is being used.
- 11011 decimal
- (11011)b binary
- 64223 decimal
- (-21843)D decimal
- 1,234 illegal, contains a nondigit character
- (1B4D)H hexadecimal number
- (1B4)D illegal hex number, does not end with “H”
- (FFFF)H illegal hex numbe, does not begin with with digit
- (0FFFF)H hexadecimal number

Signed numbers represented using 2’s complement.

Data Representation

Characters

- must be enclosed in single or double quotes
- e.g. “Hello”, ‘Hello’, “A”, ‘B’
- encoded by ASCII code

- **‘A’ has ASCII code 41H**
- **‘a’ has ASCII code 61H**
- **‘0’ has ASCII code 30H**
- **Line feed has ASCII code 0AH**
- **Carriage Return has ASCII code 0DH**
- **Back Space has ASCII code 08H**
- **Horizontal tab has ASCII code 09H**

Intrinsic data types

- **BYTE**, **SBYTE**
  - 8-bit unsigned integer; 8-bit signed integer
- **WORD**, **SWORD**
  - 16-bit unsigned & signed integer
- **DWORD**, **SDWORD**
  - 32-bit unsigned & signed integer
- **QWORD**
  - 64-bit integer
- **TBYTE**
  - 80-bit integer

Initialized data

Pseudo-instruction	<size> filed	<size> value
DB	byte	1 byte
DW	word	2 bytes
DD	double word	4 bytes
DQ	quadword	8 bytes
DT	tenbyte	10 bytes
DDQ	double quadword	16 bytes
DO	octoword	16 bytes

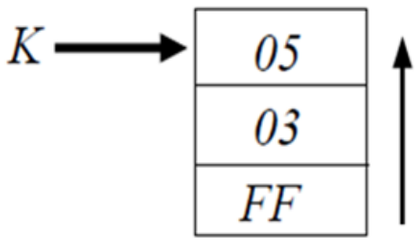
Examples:

```
var1:   db   0x55 ; define a variable ‘var’ of size byte, initialized by 0x55
var2:   db   0x55,0x56,0x57; three bytes in succession
var3:   db   'a‘    ; character constant 0x61 (ascii code of ‘a’)
var4:   db   'hello',13,10,'$‘ ; string constant
var5:   dw    0x1234          ; 0x34 0x12
var6:   dw    ‘A'             ; 0x41 0x00 – complete to word
var7:   dw    ‘AB‘            ; 0x41 0x42
var8:   dw    ‘ABC'           ; 0x41 0x42 0x43 0x00 – complete to word
var9:   dd    0x12345678      ; 0x78 0x56 0x34 0x12
```

Question: What is the difference between Var1 and Var2? Var1: a single variable, Var2: a 3-element array.

Byte Variables

- **Memory operates using the binary number system. Signals (electrical, electromagnetic) and symbols (text, letters, numbers, etc.) are defined in the binary number system.**
- **Assembler directive format defining a byte variable**
  - name DB initial value: 8 bit
  - a question mark (“?”) place in initial value leaves variable uninitialized
- **I DB 4 define variable I with initial value 4**
- **J DB ? Define variable J with uninitialized value, bellekte değişken yer ayırır.**
- **Name DB “Course-1”** allocate 8 bytes for Name. Each character is defined by 8 bits, or 1 byte.
- **K DB 5, 3, -1 allocates 3 bytes**



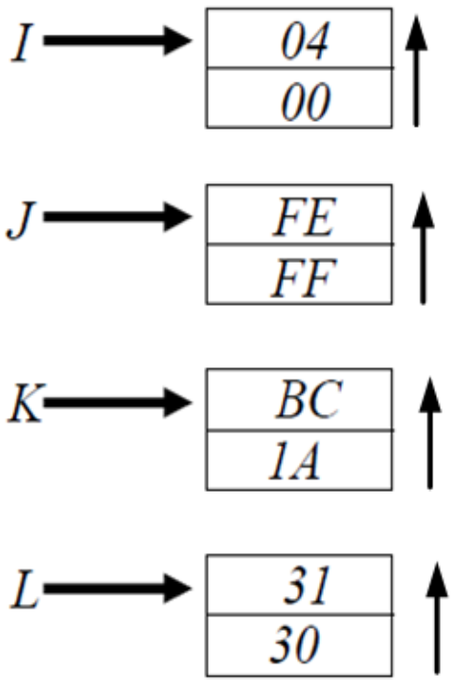
- Array DB -5,-4,-3,-2,-1,0,1,2,3,4,5
- Index the locations of the sequence K in memory from bottom to top. The index should be: 0, 1, 2, ... and so on.
- Index: 0,1,2,3,4,5,6,7,8,9,10
  05,04,03,02,01,00,FF,FE,FD, FC, FB
- K[7]=FEh

- (-5)d=(?)b=(?)h

(5)d=(0000 0101)b, (1111 1010)b+1=(1111 1011)b=(FB)h

Word Variables

- **Assembler directive format defining a word variable**
  - name DW initial value
- **I DW 4**
- **J DW -2**
- **K DW 1ABCH**
- **L DW "01"**

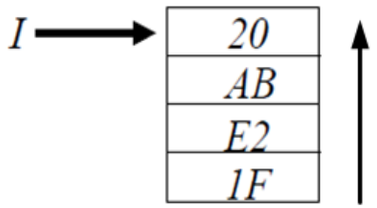


Memory cells are indexed from bottom to top.

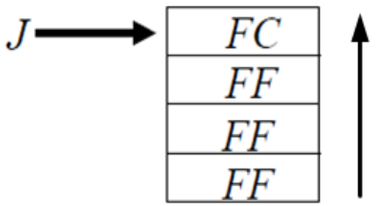
- I DW 4: (0004)h

Double Word Variables

- **Assembler directive format defining a word variable**
  - name DD initial value
- **I DD 1FE2AB20H**



- **J DD -4**



Question: K is defined as DD 77CD8854h.

Draw the location of the variable K in memory.

From bottom to top, it is 77, CD, 88, 54.

How many bytes of memory does it occupy?

The variable K occupies 4 bytes.

Index the locations of the array K in memory. Index: 0, 1, 2, ... continues. Index: 0, 1, 2, 3

a) K[2]=?

K[2]=0CDh

Assembly Language Coding

Assembly Language Programming

- Assembly Language
  - Processor-specific, low-level programming language
    - exposes most of processor features to the programmer
  - Describes
    - Data types and operations
      - Size: Byte, Word, etc
      - Meaning: integer, character
    - Storage
      - Registers
      - Memory
    - Specific instructions

Machine/Assembly Language

- **Machine Language:**
  - Set of fundamental instructions the machine can execute
  - Expressed as a pattern of 1's and 0's
- **Assembly Language:**
  - Alphanumeric equivalent of machine language
  - Mnemonics more human-oriented than 1's and 0's
- **Assembler:**
  - Computer program that transliterates (one-to-one mapping) assembly to machine language
  - Computer's native language is machine/assembly language

Assembly Language

- High level language
  - C, C++, Python, Java language
  - Abstraction
  - Human friendly

    a = b + c;
- Assembly language
  - X86 Assembly language for Intel machine
  - Hard for human to read

    Mov eax, b
    Mov ebx, c
    Add eax, ebx
    Mov a, eax
  - High efficiency to use registers
- Machine language
  - Machine code for a specific machine
  - Machine readable

    00110100100100100100111010101010
    10100100100100100111010101010111
    10101010101111101010101011110010

Key Points

- Registers
- Data Types and Representation
- Variable Declaration
- Memory Segmentation
- Memory Addressing Modes
- Instructions
  - Data flow instructions
  - Arithmetic instructions
  - Bit manipulation instructions
  - Flow control instructions

Assembly Instruction Types

INSTRUCTION SET OF X86

The X86 instructions are categorized into the following main types:

- **Data copy /transfer instructions**: These type of instructions are used to transfer data from source operand to destination operand. All the store, load, move, exchange input and output instructions belong to this category.
- **Arithmetic and Logical instructions:** All the instructions performing arithmetic , logical, increment, decrement, compare and ASCII instructions belong to this category.
- **Branch Instructions:** These instructions transfer control of execution to the specified address. All the call, jump, interrupt and return instruction belong to this class.
- **Loop instructions:** These instructions can be used to implement unconditional and conditional loops. The LOOP, LOOPNZ , LOOPZ instructions belong to this category.
- **Machine control instructions**: These instructions control the machine status. NOP, HLT, WAIT and LOCK instructions belong to this class.
- **Flag manipulation instructions:** All the instructions which directly effect the flag register come under this group of instructions. Instructions like CLD, STD, CLI, STI etc.., belong to this category of instructions.
- **Shift and Rotate instructions:** These instructions involve the bit wise shifting or rotation in either direction with or without a count in CX.
- **String manipulation instructions:** These instructions involve various string manipulation operations like Load, move, scan, compare, store etc..

Instruction Types

- **Data transfer instructions**
  - Transfer information between registers and memory locations or I/O ports.
  - MOV, XCHG, LEA, PUSH, POP, PUSHF, POPF, IN, OUT.

- **Arithmetic instructions**
  - Perform arithmetic operations on binary or binary-codeddecimal (BCD) numbers.
  - ADD, SUB, INC, DEC, ADC, SBB, NEG, CMP, MUL, IMUL, DIV, IDIV, CBW, CWD.

- **Bit manipulation instructions**
  - Perform shift, rotate, and logical operations on memory locations and registers.
  - SHL, SHR, SAR, ROL, ROR, RCL, RCR,
  - Logical operations: NOT, AND, OR, XOR, TEST.

Instruction Types

- **Control transfer instructions**
  - Control sequence of program execution; include jumps and procedure transfers.
  - JMP: kontrolsuz istenilen yere gider.
  - CMP: İki değişkenin çıkarma işlemi sonucunda bayraklar etkin olsur: zero, sign, cary, ...

  JG, JL, JE, JNE, JGE, JLE, JNG, JNL, JC, JS, JA, JB, JAE, JBE, JNB, JNA, JO, JZ, JNZ, JP, JCXZ,

  - **Loop instructions:** LOOP, LOOPE, LOOPZ, LOOPNE, LOOPNZ; Cx register değeri her loop işleminin sonunda 1 eksilir, Cx=0 olduğunda Loop'dan çıkar.
  - Alt Progam: CALL, RET.

- **String instructions**
  - Move, compare, and scan strings of information.
  - MOVS, MOVSB, MOVSW, CMPS, CMPSB, CMPSW. SCAS, SCASB, SCASW, LODS, LODSB, LODSW, STOS, STOSB, STOSW.

Instruction Types

- **Interrupt instructions**
  - Interrupt processor to service specific condition.
  - INT, INTO, IRET.
- **Processor control instructions**
  - Set and clear status flags, and change the processor execution state.
  - STC, STD, STI.
- **Miscellaneous instructions**
  - NOP, WAIT.

Destination - Source

- The source is manipulated with the target, and the result is in the target.
- The source can be an instantaneous value, a general-purpose register, or a memory location.
- The target can be a general-purpose register or a memory location.
- The source and target must be the same size.
- For example:

  Mov ah, bl

  – Mov ax, bx
  – Mov Ax, Cl; Such a command cannot be written; the bit lengths must be the same.

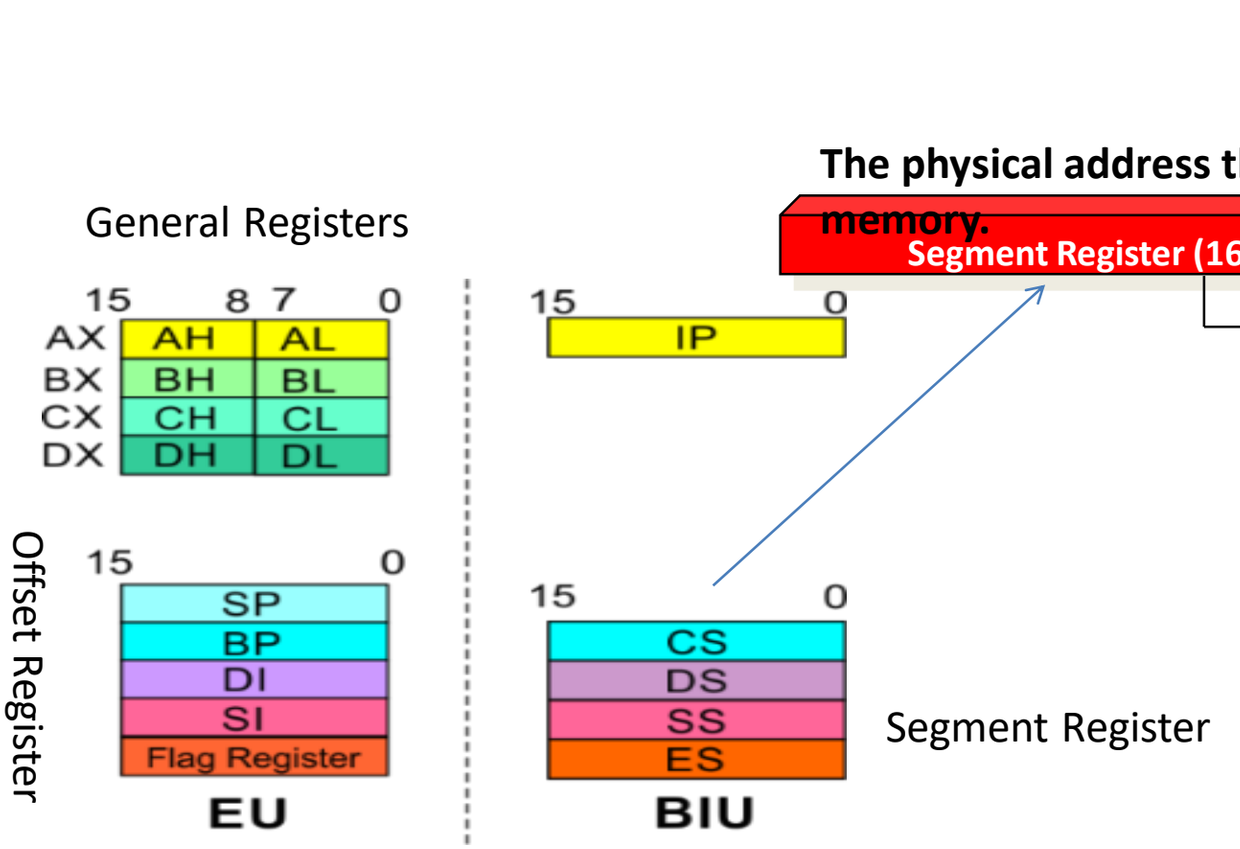
General Rules

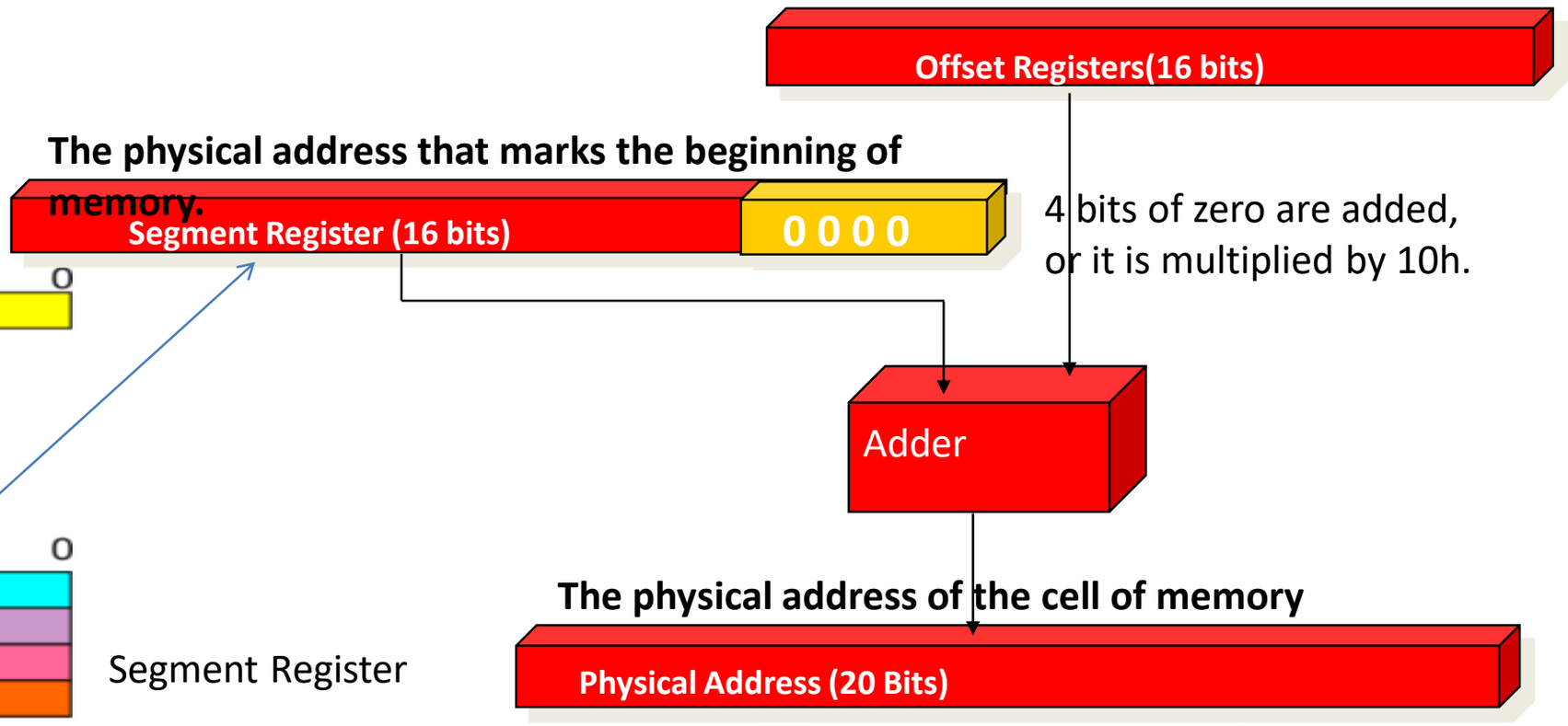
- Mov Target, Source; Source → Destination
- Source and Destination temporary register fields must be the same size.
- Mov AX, BL Illegal; different sizes Ax: 16 bits BL: 8 bits
- Mov AL, BL Legal; Al: 8 bits, BL: 8 bits
- Mov AX, CX Legal
- Mov DS, AX Legal; the starting address of DS is loaded.
- Mov Ax, 0BBCCh Legal
- Mov AL, 0BBCCh Illegal
- Mov DS, 0BBCCh Illegal, numerical values cannot be directly loaded into segment registers.
- Mov i,j i,j: variables (Coefficients or constants) Destination cannot be a variable.
- Mov AL, j Legal

In instruction processing, the length of the registers must be the same. Data registers AX, BX, CX, and DX have a length of 16 bits, while AH, AL, BH, BL, CH, CL, DH, and DL have a length of 8 bits.

Physical Address in Memory Access

Addressing Modes Memory Access





- The starting addresses of memory locations are stored in segment registers. The addresses of memory locations are stored in offset registers.

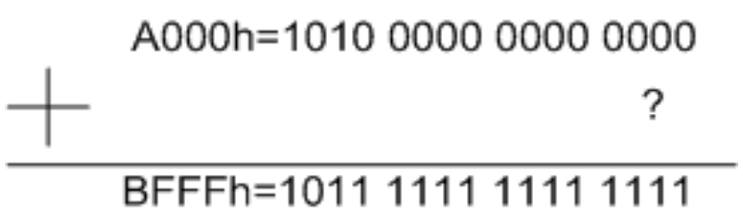
Segment	Offset Registers	Function
CS	IP	Address of the next instruction
DS	BX, DI, SI	Address of data
SS	SP, BP	Address in the stack
ES	BX, DI, SI	Address of destination data (for string operations)

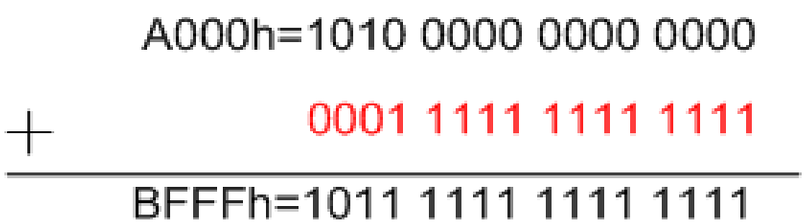
Memory Index and Memory Capacity

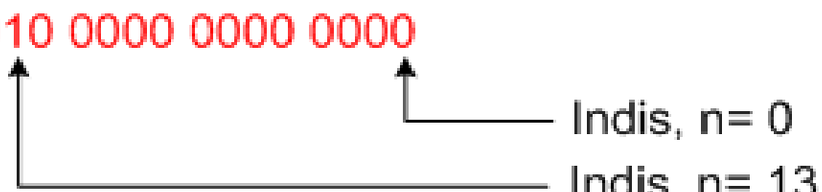
- Question: If the physical starting address of memory is (A000)h and the physical ending address of memory is (A0FF)h, what is the size of the memory?
- Memory capacity = Physical ending address of memory - Physical starting address of memory + 1
- Memory capacity = (A0FF)h – (A000)h + 1 = (FF)h + 1
- Memory capacity: 100h = (0001 0000 0000) = 2^8 = 256 bytes
- Memory address bus index = A7, A6, … , A1, A

Example-3.1: Memory Capacity

- Memory Capacity = Memory End Address – Memory Start Address + 1 (The reason for adding +1 is that the memory start address starts from 0.)
- Example: If Memory Start Address = (A000)h and Memory End Address = (BFFF)h, calculate the memory capacity.
- Memory range = A000h + 1FFFh = BFFFh; What do we add to the starting address to find the ending address?
- Memory Capacity range: (0000)h – (1FFF)h
- Memory capacity = 0001 1111 1111 1111 + 1 = 0010 0000 0000 0000
- Memory Capacity = 2^13 Bytes = 8Kbyte
- Number of address lines = 13
- Address line indexing: A12, A11, … , A1, A0
- Since the starting address of the memory is the physical address, the content of the Data segment register is found by deleting 4 bits from the right. DS becomes 0A00h. The CPU automatically takes the physical address as: 0A000h. Because it adds 4 bits to the right of the value in the segment register.







Example-3.2: Writing to the Memory Cell

- 07
- Write 55h to memory location 1271.
- Our memory is DS
- Mov Ax, 0A00h
- Mov DS, AX
- Mov DS: [1271], 55h; writes the value 55h to memory location 1271d. Memory location indexing: (0,1,2, …., 1271d, …., 1FFFh)
- CPU calculates the physical address. Finds the physical address of the DS segment. DS: 0A000h
- Offset address: 1271=2^10+2^7+2^6+2^5+2^4+2^2+2^1+2^0
- Indexing: 10 9 8 7 6 5 4 3 2 1 0
- Offset address: 1271d= (100 1111 0111)b
- How does the CPU calculate the physical address of the memory location?
- (A000h): (1010 0000 0000 0000)b
- 1271d: (0000 0100 1111 0111)b =(04F7)h
- Physical address of the memory cell=(A000)h+(1271)d=((1010 0100 1111 0111)b=(0A4F7)h

```
Mov Ax,0A00h
Mov DS, AX
Mov DI, 04F7h
Mov DS: [DI], 55h
```

Örnek-4: 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

Soru: Belleğin başlangıç adresi ve bellek gözünün fiziksel adresi

- Aşağıdaki assembly yazılımın işlevini açıklayınız. Belleğin başlangıç adresi ve bellek gözünün fiziksel adresi nedir?
- ORG 100h
- MOV AX, 2000h
- MOV DS, AX
- MOV BX, 3456h
- MOV DS: [BX], 44h
- RET
- **Bu yazılım başlangıç adresi stack segment register'ın içeriğinde saklı olan belleğin indisi BX register'da saklı olan gözünü 44h değerini yazar. Stack Segment register içeriği= 2000h.**
- **Bellek gözünün fiziksel adresi= 20000 + 3456h=23456h**

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 Hedef, Kaynak

MOV Hedef, Kaynak

- **Mov Destination (Hedef), Source (Kaynak)**
- Kaynak (Source) -> Hedef(Destination)
- Kaynak lokasyonundaki veriyi hedef lokasyonuna transfer eder.
- Kaynak:
  - Anlık bir değer, sayısal değerler (Değişkenler: Katsayılar, Sayılar)
  - Genel amaçlı register (Ax, Bx, Cx, Dx: 16 bit; AL, AH, BL, BH, CL, CH, DL, DH: 8bit; SI,DI, SP, BP: 16bit; segment register)
  - 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).

MOV Operand1, Operand2

Algorithm: operand1 = operand2

Opeand1,	**Operand2:**
• REG,	memory
• REG,	REG
• REG,	SREG
• REG,	immediate
• memory,	REG
• memory,	immediate
• memory,	SREG
• SREG,	memory
• SREG,	REG

Not: MOV komutu CS ve IP register değerini transfer edemez. MOV komutu anlık veri değerini segment register'a doğrudan transfer edemez, transfer edebilmesi için önce anlık veri değeri genel amaçlı bir register'a transfer edilir, ardından genel amaçlı register içeriği segment register'a transfer edilir.

MOV Operand1, Operand2

Algorithm: operand1 = operand2

- **Example:**

```
ORG 100h
MOV AX, 0B800h ; ( 1- Komut doğru, sayısal  değer genel amaçlı register'a transfer edilmektedir.
                   2- Boyutları doğru, 16bit,
                   3- 0B800h sayısal değeri AX register'a transfer edilmektedir.)
MOV DS, AX ; copy value of AX to DS.
MOV CL, 'A' ; CL = 41h (ASCII code).
MOV CH, 01011111b ;
MOV BX, 15Eh ; BX = belleğin başlangıç adresinden itibaren nerede olduğunu gönderen indis register'dır.
MOV [BX], CX ; DS[0B8000h+015Eh] = CX.
RET ; returns to operating system.
```

Not: Her bir komut için

- Komut doğru mu?
- Boyutları doğru mu?
- Komutun işlevi nedir? Sorularının yanıtlanması gerekmektedir.

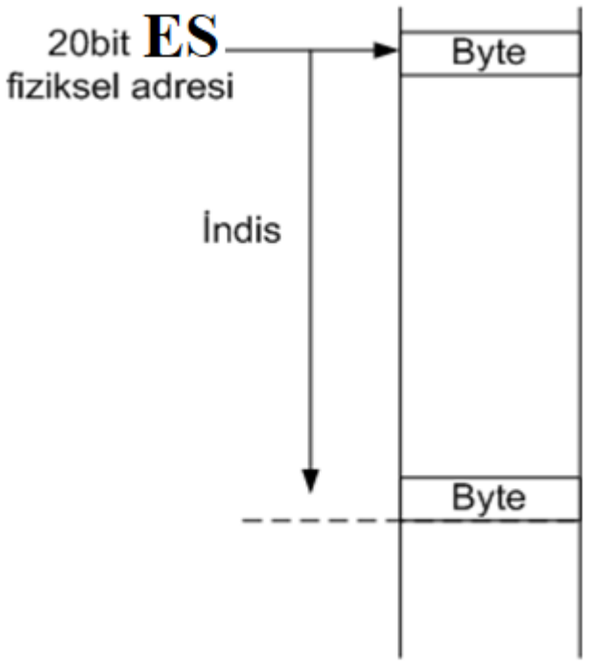
Bellek Gözüne Erişim
Fiziksel Adres

20 bit Fiziksel Adresin Hesaplanması

- CPU, segment register içeriğini 10h ile çarparak ve buna genel amaçlı BX ve/veya indis register SI, DI değerleri eklenerek (1230h * 10h + 45h = 12345h) fiziksel adres hesaplaması yapar.
- Burada Segment adresi: 123h, Indis kaydı: 45h
- Segment Register: CS, DS, SS, ES
- Indis Register: CS (IP), DS(SI, DI, BX), SS(SP, BP), ES(SI, DI, BX).
- [SI, DI, SP, BP, BX, Sayısal değer] ile gösterilen 2 kayıttan oluşan adrese etkin adres denir.
- Varsayılan olarak BX, SI ve DI kayıtları DS segment kaydıyla çalışır; BP ve SP, SS segment kaydıyla çalışır.
- Diğer genel amaçlı register (CX, AX, DX) etkili bir adres oluşturamaz! ayrıca BX etkili bir adres oluşturabilse de BH ve BL bunu yapamaz.

Örnek-2: Bellek Gözünün Fiziksel Adresi

Soru: Fiziksel adresi bc150h olan bellek gözüne bl register'ın içeriği yazılacaktır. İndis: 150h, SI register'ında kayıtlı olacaktır. Segment register ES olacaktır. Assembly yazılımını yazınız.



- Belleklerin fiziksel başlangıç adresleri 20bit = ES*10h + SI
- Segment register'lar 16 bittir. Segment register'ların içeriği temel belleklerin başlangıç adresini gösterir.
- Bellek gözünün fiziksel başlangıç adresleri =Segment register * 10h + SI
  - Segment Regsiter*10h=bc150h - 150h=bC00
  - Segment register, ES= BC000h/10h= bC00h, çünkü extra segment register 16 bit'tir.
  - İndis, si: 150h
  - Bellek gözünün fiziksel adresi=bC000h + 150h = bc150h

```
Mov ax, 0bC00h
Mov es, ax
Mov si, 150h
Mov bl,20h
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

- Belleğe erişmek için offset regster'lar kullanabilir: BX, SI, DI, BP, SP. bu kayıtları [ ] sembolleri içinde birleştirerek farklı hafıza konumları elde edebiliriz. Bu kombinasyonlar desteklenen adresleme modlarıdır.
- 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:

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

Bellek Erişim

- Segment kaydındaki (CS, DS, SS, ES) 16 bitlit değerlere segment register adı verilir ve 20 bitlik bellek başlangıç adresini tanımlarlar. indis (BX, SI, DI, BP) register değerlere ofset denir. Yer değiştirme, bir değişken değeri veya bir değişkenin ofseti veya hatta her ikisi olabilir. Birden fazla değer varsa, tüm değerler toplanarak değerlendirilir ve tek bir değişken değeri hesaplar.

- Belleğe erişmek için şu dört kaydı kullanabiliriz: BX, SI, DI, BP, SP bu kayıtları [ ] sembolleri içinde birleştirerek farklı hafıza konumları elde edebiliriz. Bu kombinasyonlar desteklenir (adresleme modları):

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

Bellek Erişim

- Mov [SI], 55h ifadesinde DS: b800h değerini ve SI: 10h değerini içerdiğinde b800:10 olarak da kaydedilebilir. [SI]=Fiziksel adres b8000h + 10h = b8010h olacaktır.

- d8 - 8 bit işaretli değişken veri değişken için kalır (örneğin: 22, 55h, -1, vb...)
- d16 - 16 bitlik işaretli veri değişken için kalır (örneğin: 300, 5517h, -259, vb...).
- d16 (yalnızca değişken ofset)

- Sayısal değerler işaretli bir değerdir, dolayısıyla hem pozitif hem de negatif olabilir. Genellikle derleyici d8 ve d16 arasındaki farkla ilgilenir ve gerekli makine kodunu üretir.

- 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.

Bellek Erişim

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

- Varsayılan olarak DS segment kaydı, BP kaydı olan modlar dışındaki tüm modlar için kullanılır; bu SS segment kaydı için kullanılır. Bu tabloyu kullanarak tüm olası kombinasyonları hatırlamanın kolay bir yolu var:

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]

Physical address

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

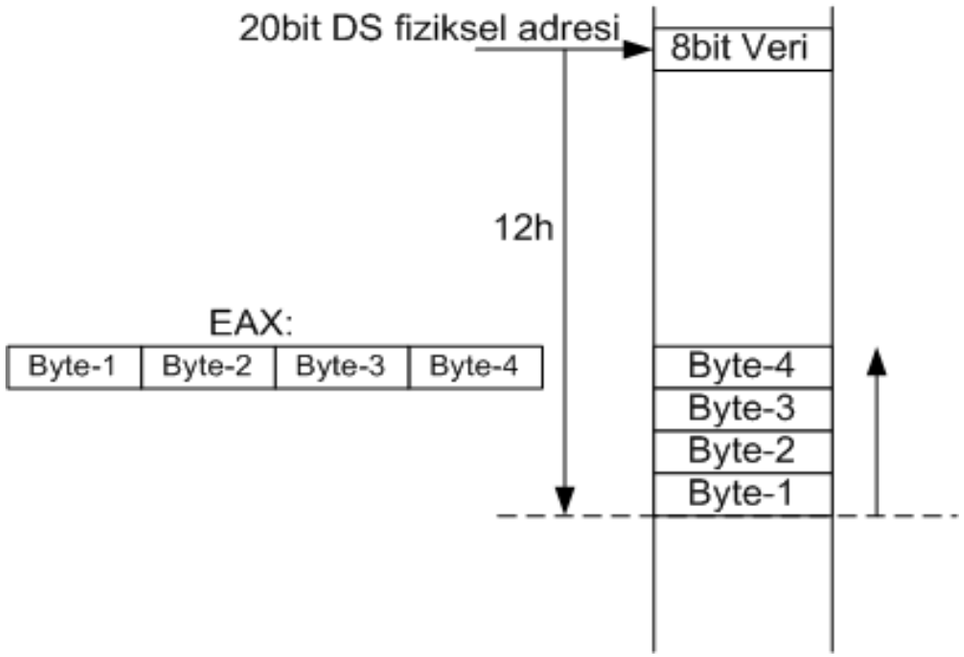
Answer:

(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 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:

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

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**

Kayıt edilen değişkenlerin ofset adresini gösterir.

- 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: LEA, Değişkenlerin bellek gözündeki yerleri

```
org 100h
Mov ax, 700h
Mov ds, ax
lea si, I
lea bx, J
lea di, K


mov ch, [si]
mov cl, [bx+7]
mov dh, [di+4]
ret

I db 4
J db "Cahit Karakus - 1960"
K db 5,3,-1,-5,-10, 0Ah, 01010101b
```

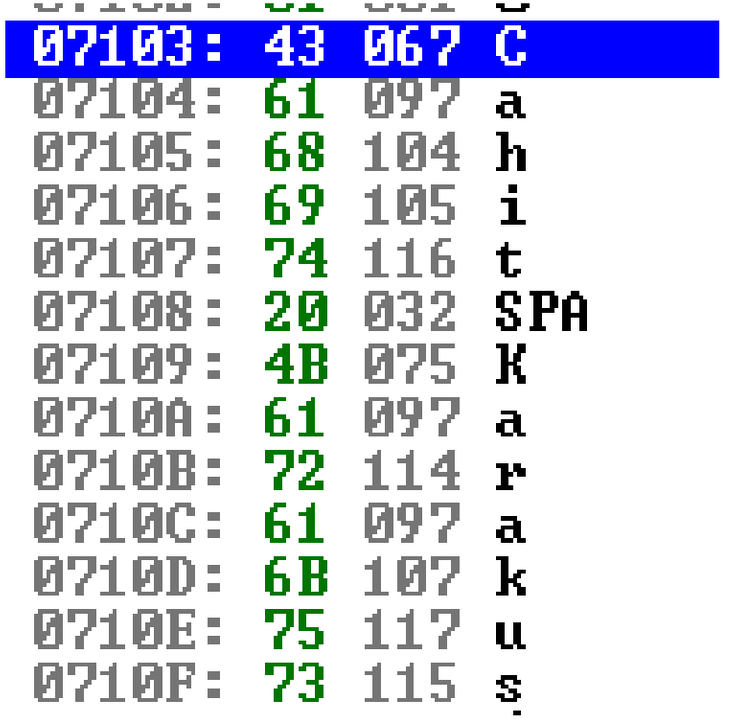
DS: 0700h
Belleğin fiziksel baslangıç adresi=07000h

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



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

Mov BX, 0AA55h
Mov Ax, 0CCDDh
XCHG AX, BX
RET

Data Transfer Komutları (Push, Pop)

Segment: SS, Komutlar: PUSH, POP

İlgili register: AX, BX, CX, DX
(AH, AL; BH,BL; CH,CL;DH,DL)

PUSH reg16/ mem

PUSH reg16

$(SP) \leftarrow (SP) - 2$
Fiziksel Adres = (SS) x 10h + SP
$(MA_S ; MA_S + 1) \leftarrow (reg16)$

SI, DI
BP, SP
IP
DS, SS,CS, ES

PUSH mem

$(SP) \leftarrow (SP) - 2$
$MA_S = (SS) \times 16_{10} + SP$
$(MA_S ; MA_S + 1) \leftarrow (mem)$

Push komutu ile Ram bellkte saklanır, geri getirilmek istendiğinde pop kullanılır.

POP reg16/ mem

POP reg16

$MA_S = (SS) \times 16_{10} + SP$
$(reg16) \leftarrow (MA_S ; MA_S + 1)$
$(SP) \leftarrow (SP) + 2$

POP mem

$MA_S = (SS) \times 16_{10} + SP$
$(mem) \leftarrow (MA_S ; MA_S + 1)$
$(SP) \leftarrow (SP) + 2$

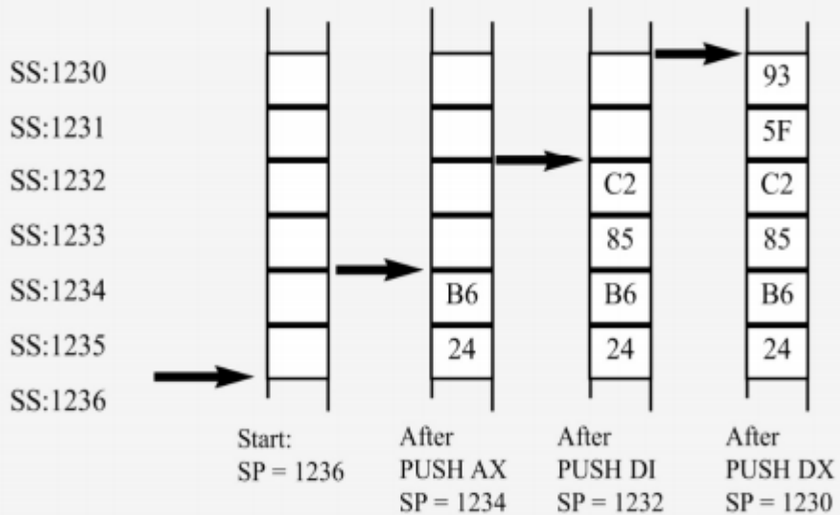
```
mov ax, 55aah
mov bx, 4488h
push ax
push bx

mov ax, 2222h
mov bx, 3333h

pop bx
pop ax
```

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, 1000h
 Mov ss, ax
 Mov sp, 60h
 MOV AX, 0102h ; store 1212h in AX.
MOV BX, 0304h ; store 3434h in BX
PUSH AX ; store value of AX in stack segment.
PUSH BX ; store value of BX in stack segment.
MOV AX, 55aah
MOV BX, 0aa55h
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
```

Push yaptıktan sonra bellekteki yerini bulabilmek için ss: sp (1000: 5a) işlemi yapılmalıdır.

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	√

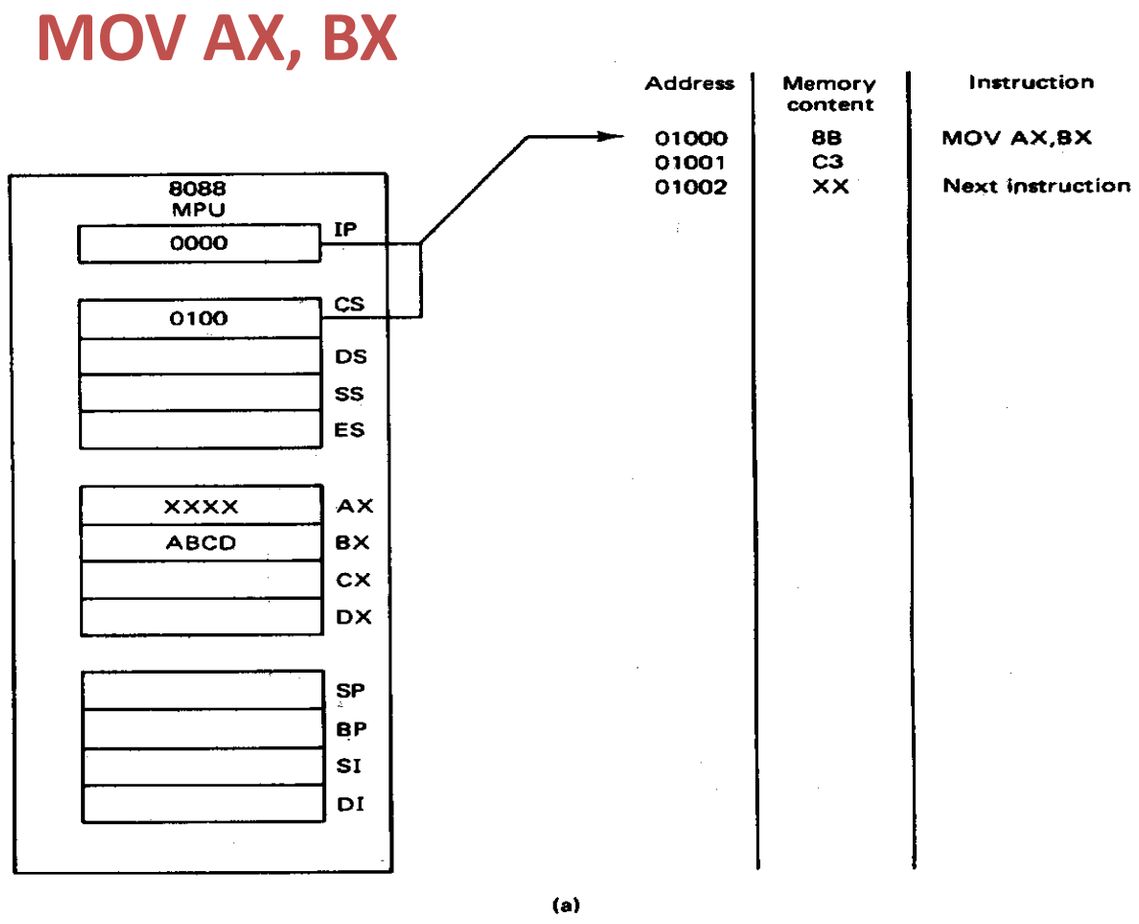
- Register'lara değer atama işlemleri MOV komutu ile yapılır.
- MOV komutlarında register uzunlukları heriki tarafta da aynı olmak zorundadır.
- Segment register'lara doğrudan değer atanmaz. Segment register'a değer atanması için, değer önce MOV komutuyla AX register'ına atanır , sonra AX register'ın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

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

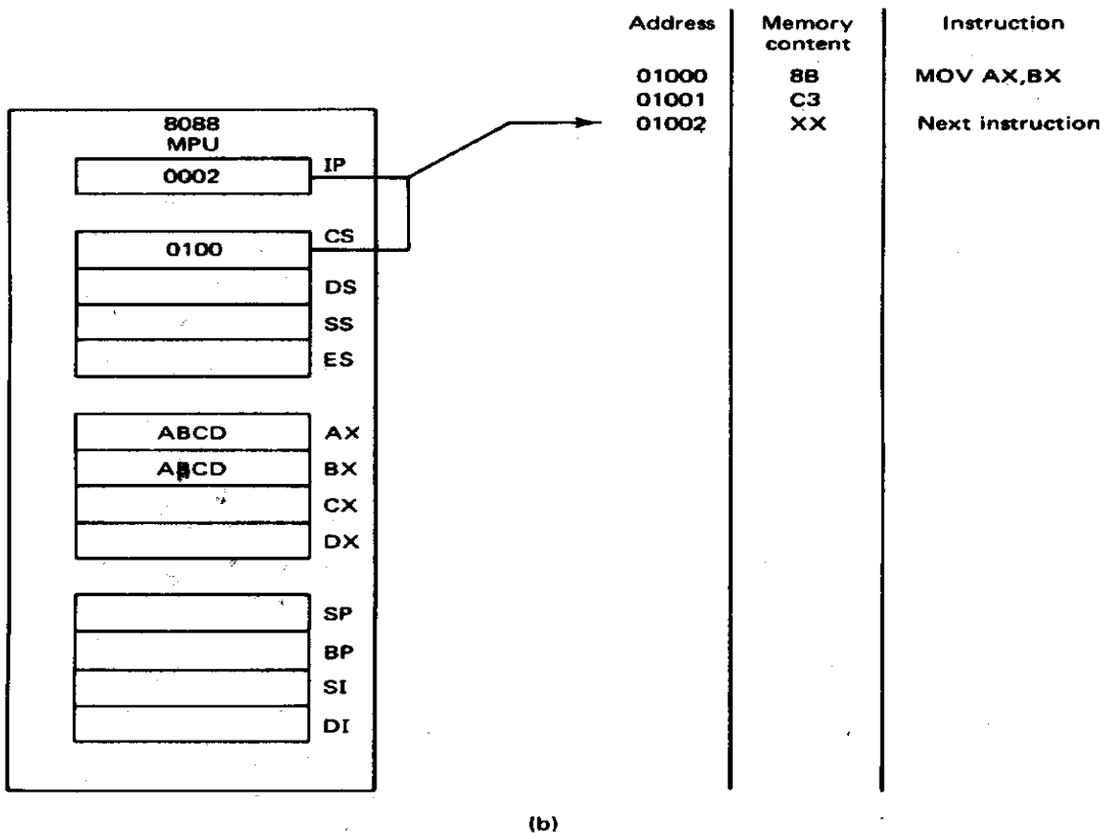
- AX,BX,CX,DX regster'ların 16bittir. AH-AL, BH -BL, CH-CL, DH-DL register'ların kapasitesi 8 bittir.

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, 45h
ADD AL, cl
NOT AX
OR AX, 0000h
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

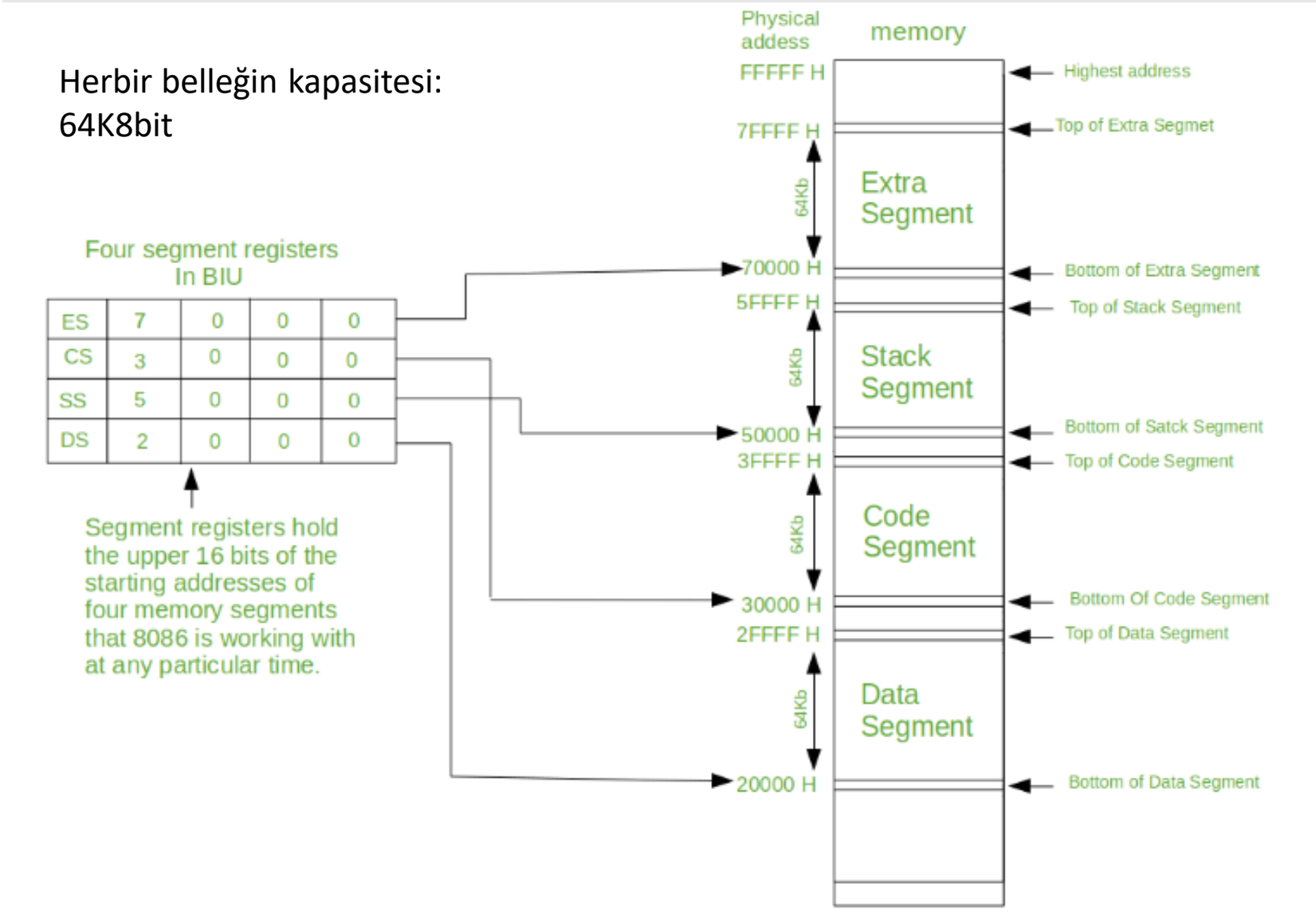
Group II:
Bellek Erişim Adres Modları

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 → 89AB → 89AB0 (Paragraph to byte → 89AB x 10 = 89AB0)
F012 → 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.

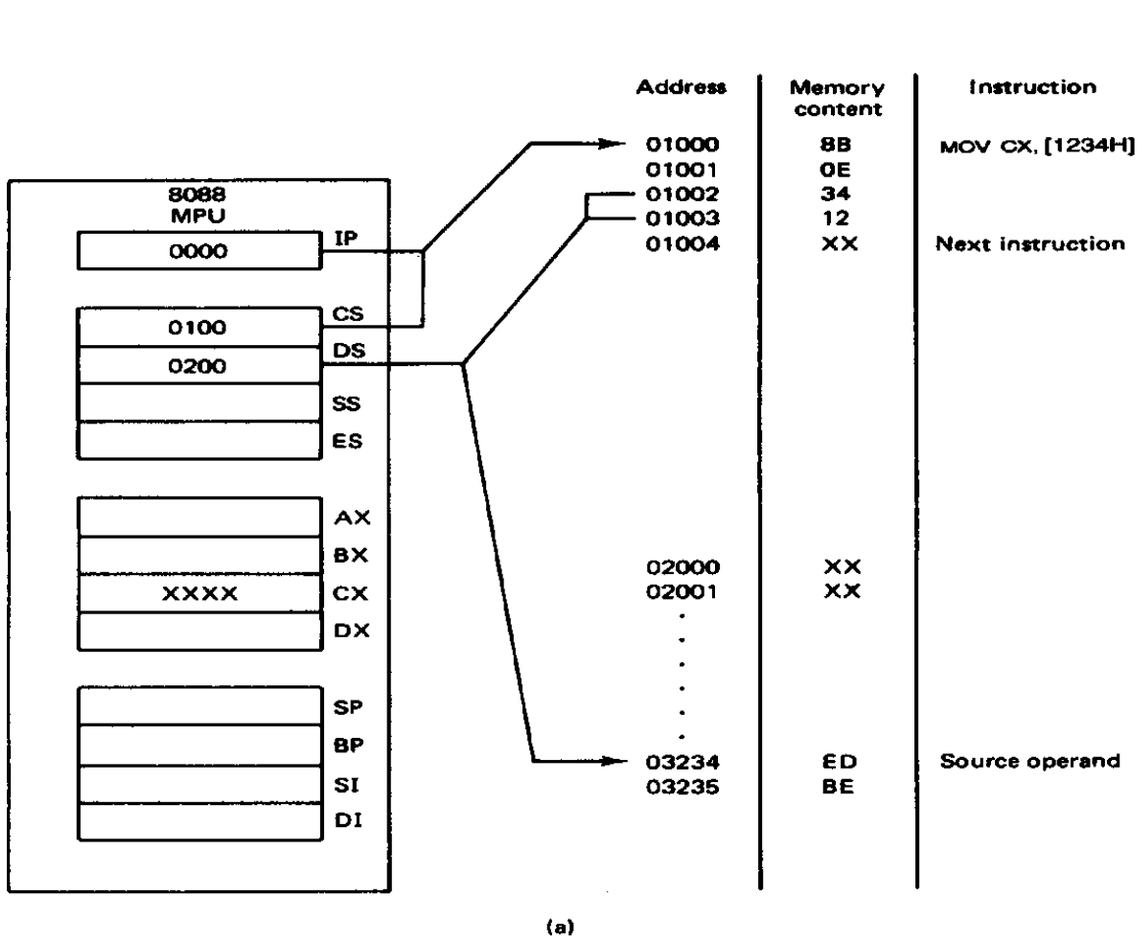
Example:

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

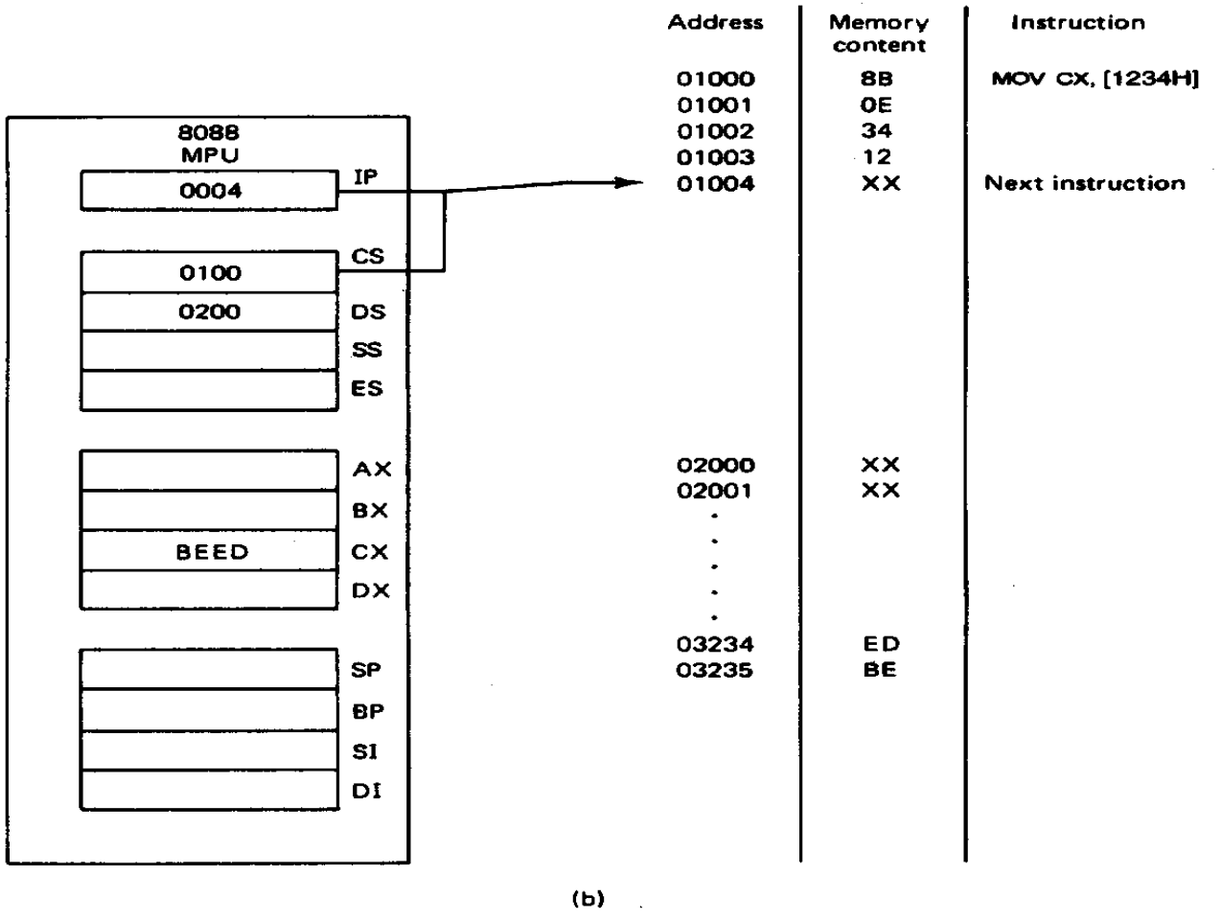
- Köşeli parantezin önünde segment register ifadesi yok ise default olarak DS olduğunu gösterir.
- 1354H'nin etrafındaki köşeli parantezler bellek konumunun içeriğini gösterir.
- Köşeli parentizin önünde SS: [...], ES:[...] olarak segment register belirtilmemişse varsayılan ofset adres, data segment'tir.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

Direct Addressing Mode

MOV CX, [1234h]



(a)



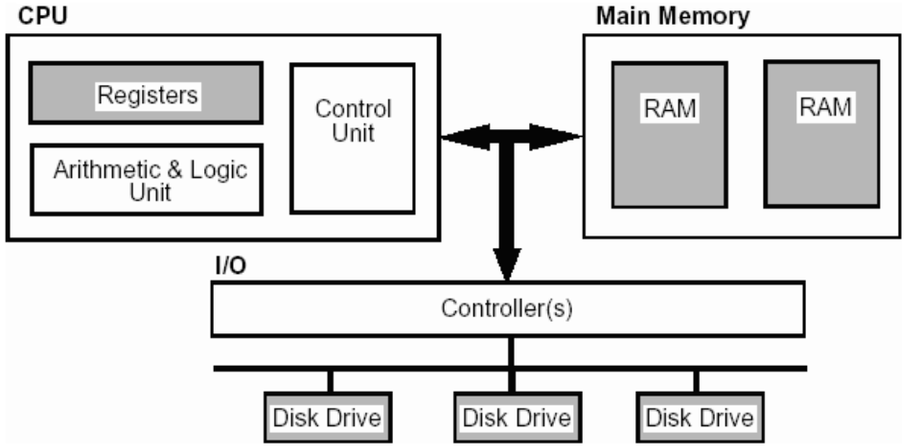
(b)

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)

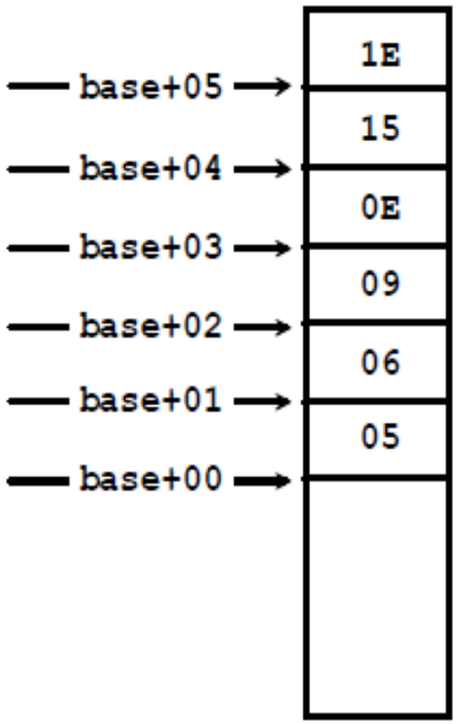
(CH) ← (MA +1)

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

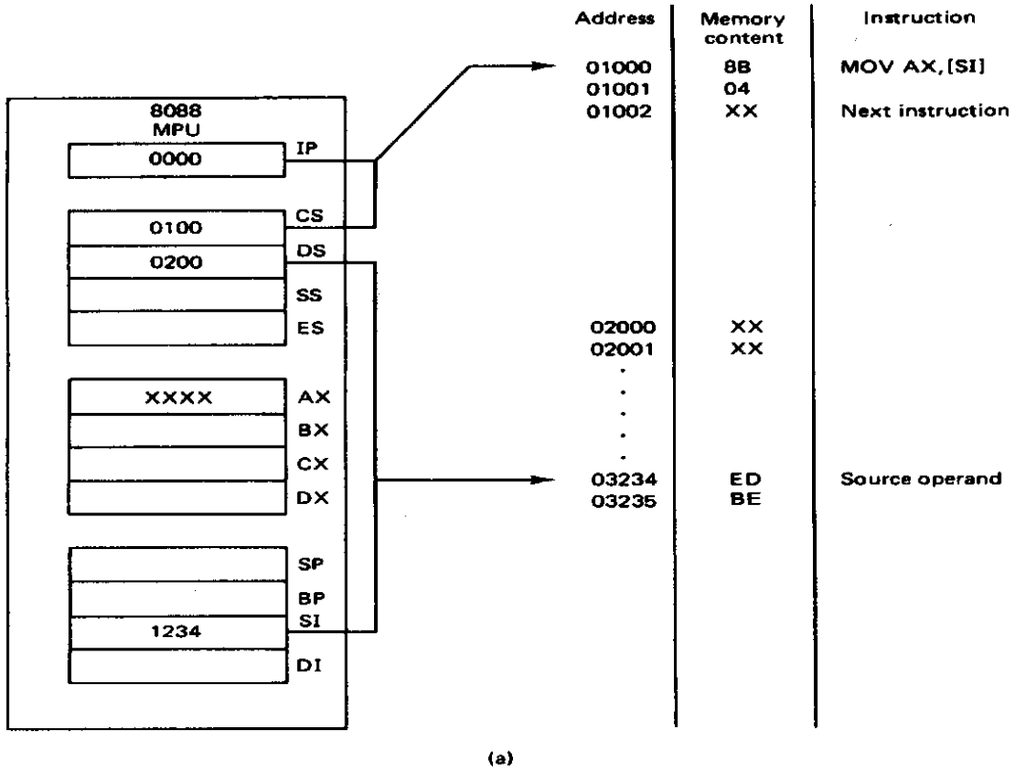
```
    ORG 0x100
section .data
    table dw 1,2,3,4,5,6,7,8,9,10,0
section .text
    MOV     SI,table
    MOV     AX,0000
L1: ADD     AX,[SI]
    ADD     SI,2
    CMP     WORD [SI],0

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

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

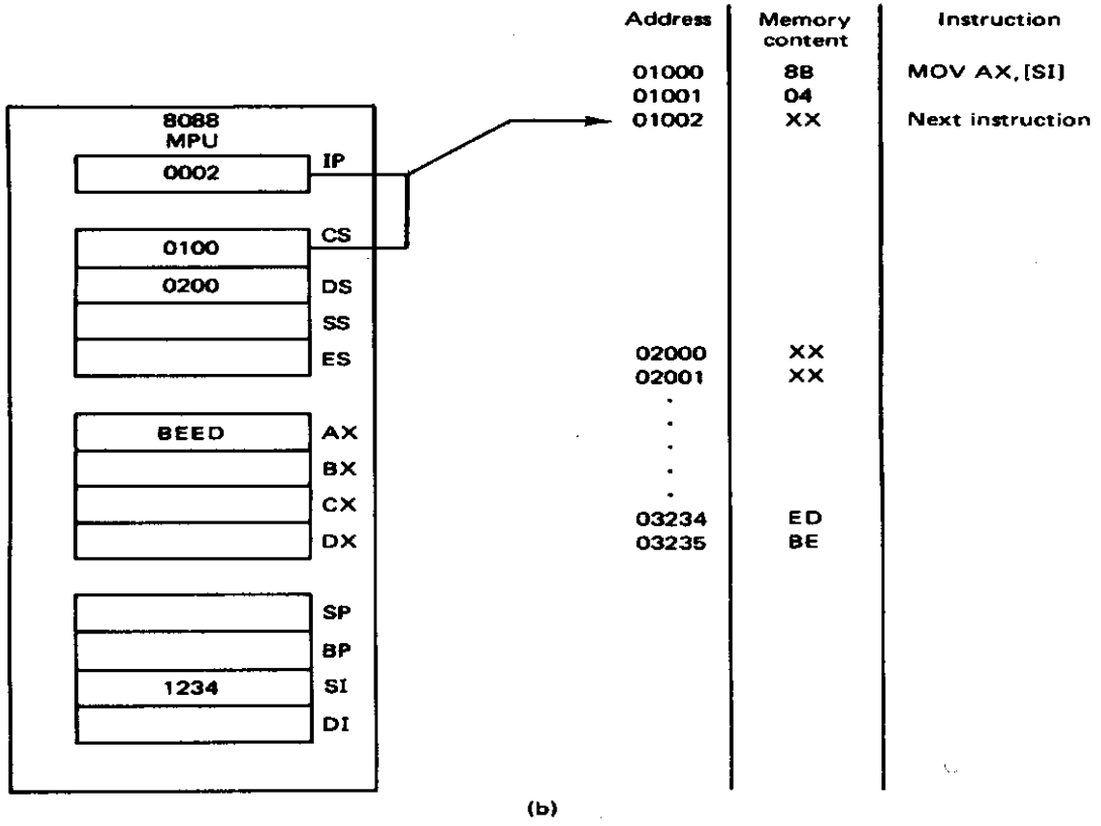
AX = 0 → 1 → 3 → 6 → 10 → 15 → 21 → 28 → 36 → 45 → 55

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:

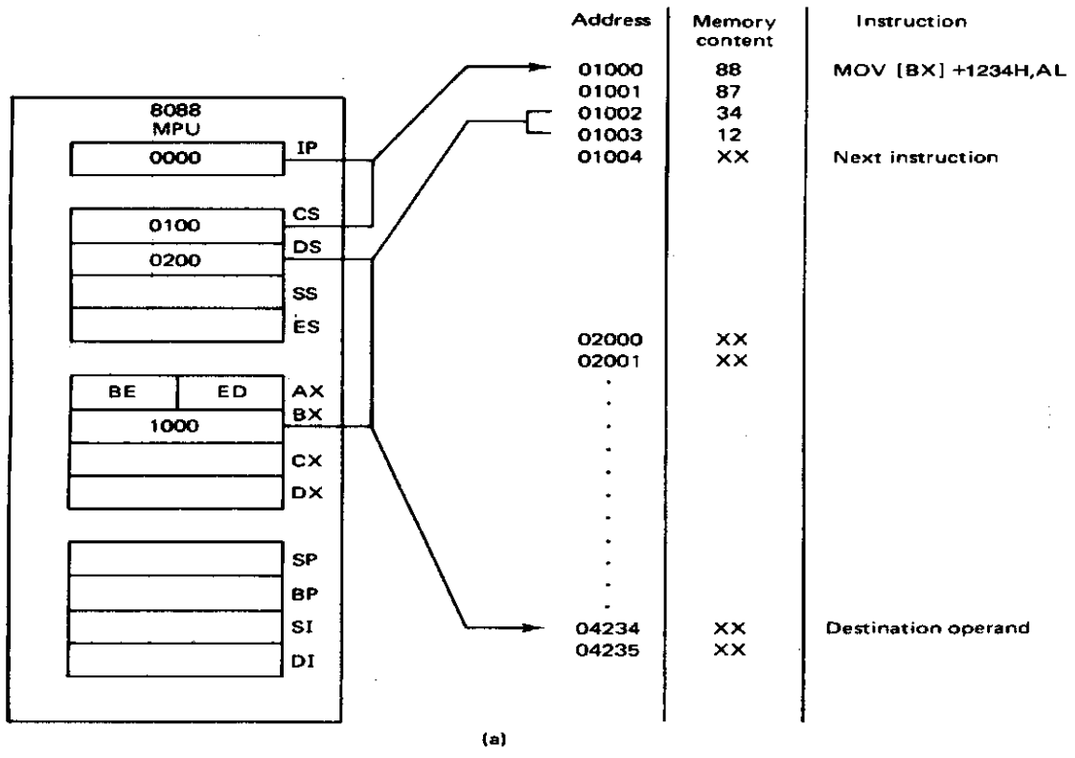
$0008_H \leftarrow 08_H$ (Sign extended)
EA = (BX) + 0008_H
BA = (DS) x 16_{10}
MA = BA + EA

(AX) ← (MA) or,

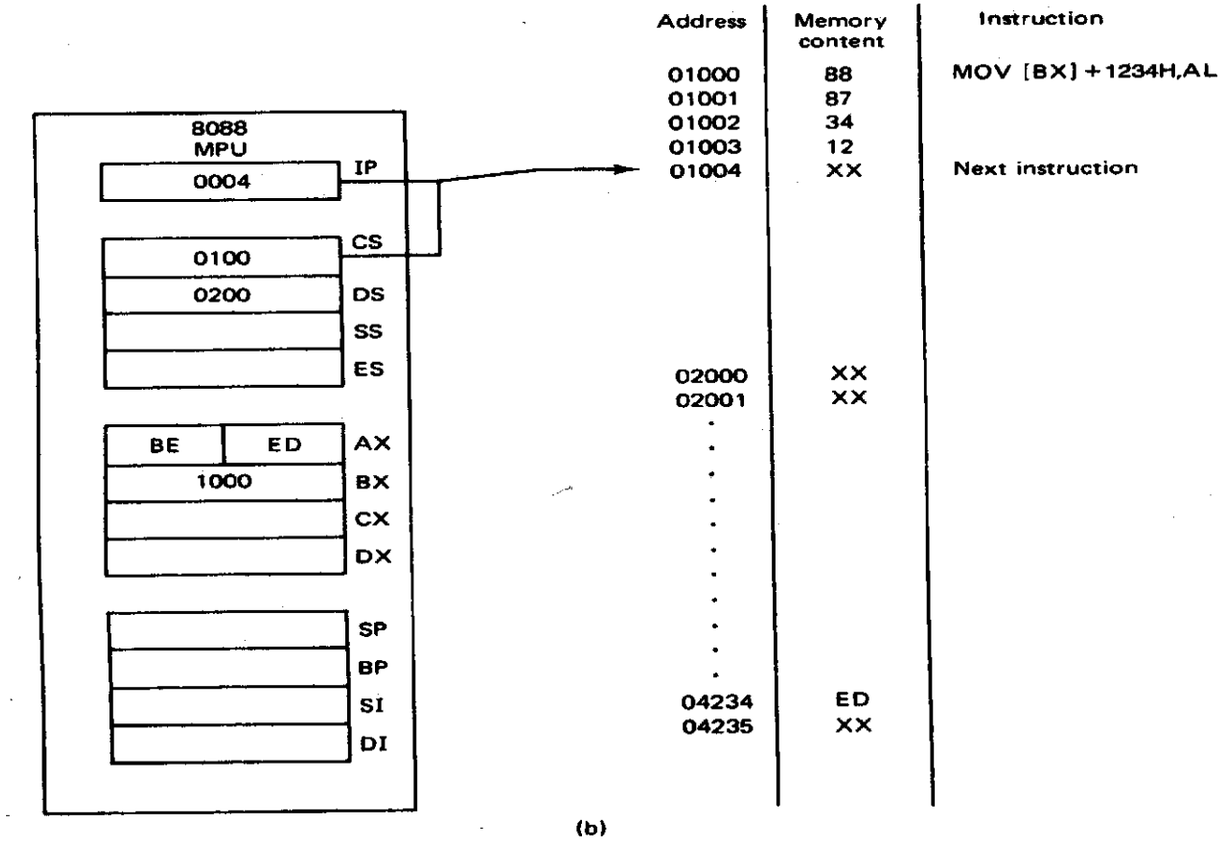
(AL) ← (MA)
(AH) ← (MA + 1)

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,

$(CL) \leftarrow (MA)$

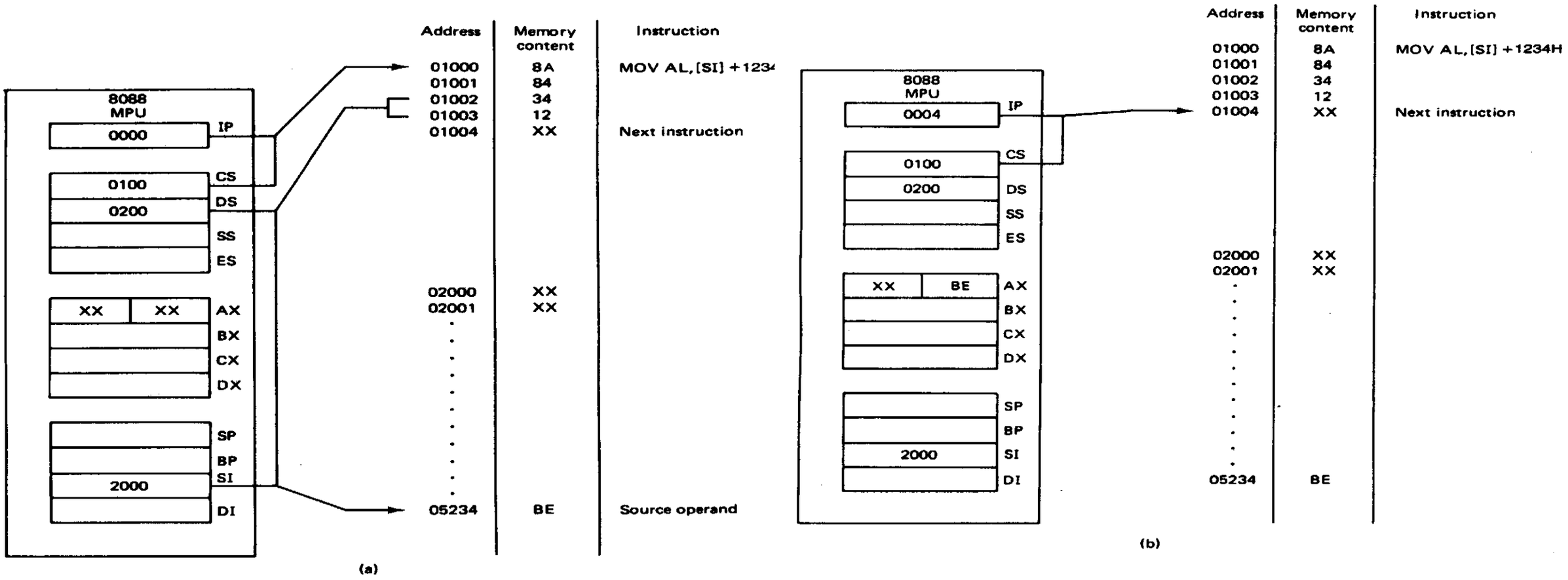
$(CH) \leftarrow (MA + 1)$

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:

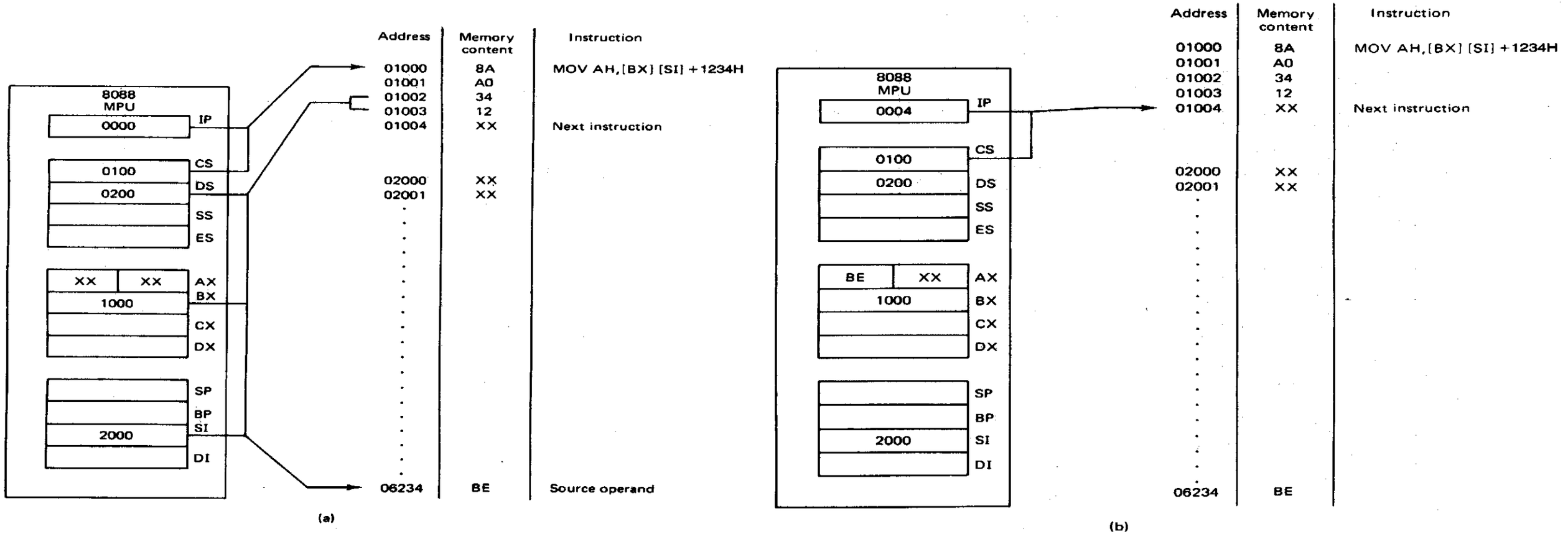
$000A_H \leftarrow 0A_H$ (Sign extended)
EA = (BX) + (SI) + $000A_H$
BA = (DS) x 16_{10}
MA = BA + EA

(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

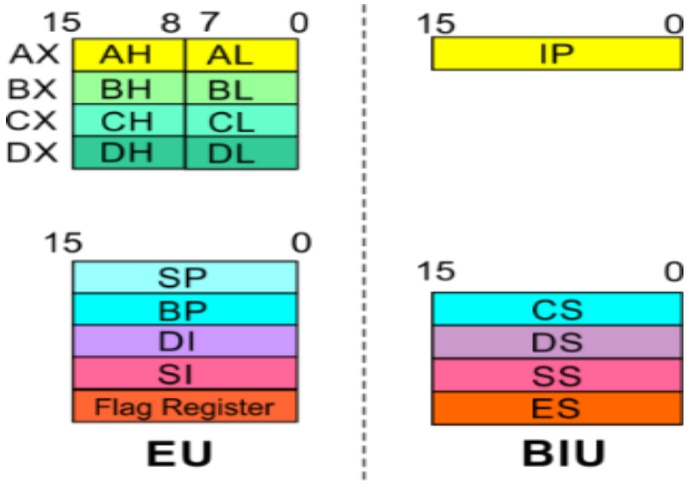
These addressing modes are used to access data from standard I/O mapped devices or ports. In direct port addressing mode, an 8-bit port address is directly specified in the instruction.

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: $PORT_{addr} = 09_H$
$(AL) \leftarrow (PORT)$

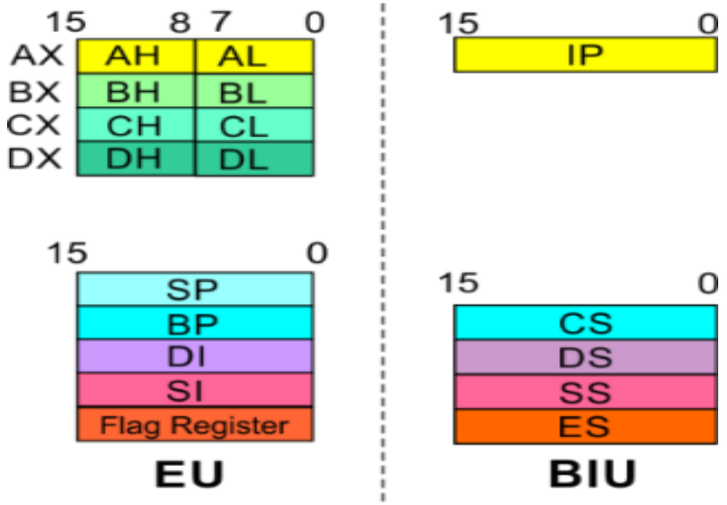
Content of port with address 09_H is moved to AL register

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)

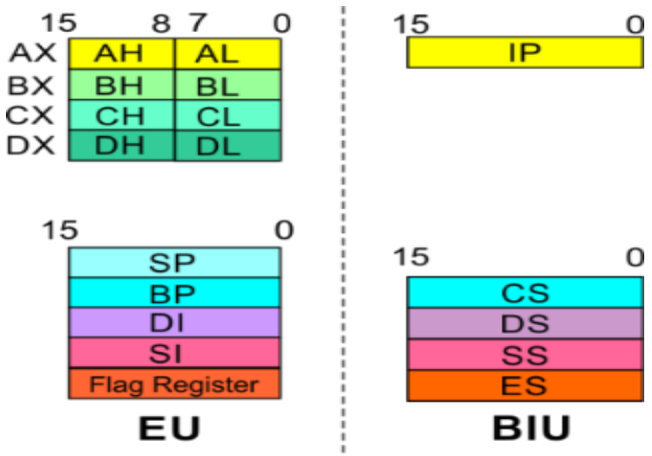
If ZF = 1, then

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

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

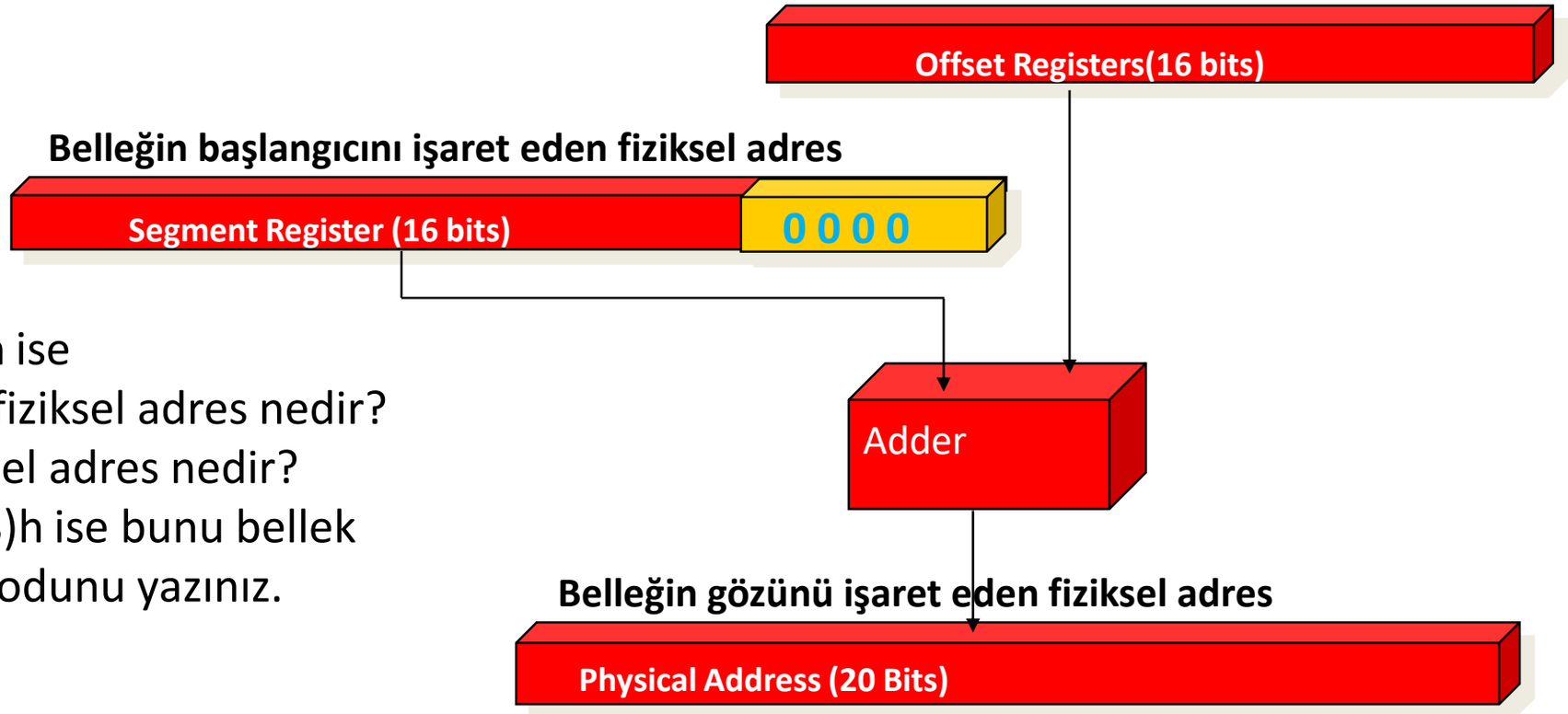
b) Belleğe 8bit yani byte olarak yazar okur. Data bus hat sayısı 8 adettir. Data hatları indisi: D7, D6, D5, ..., D1, D0

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

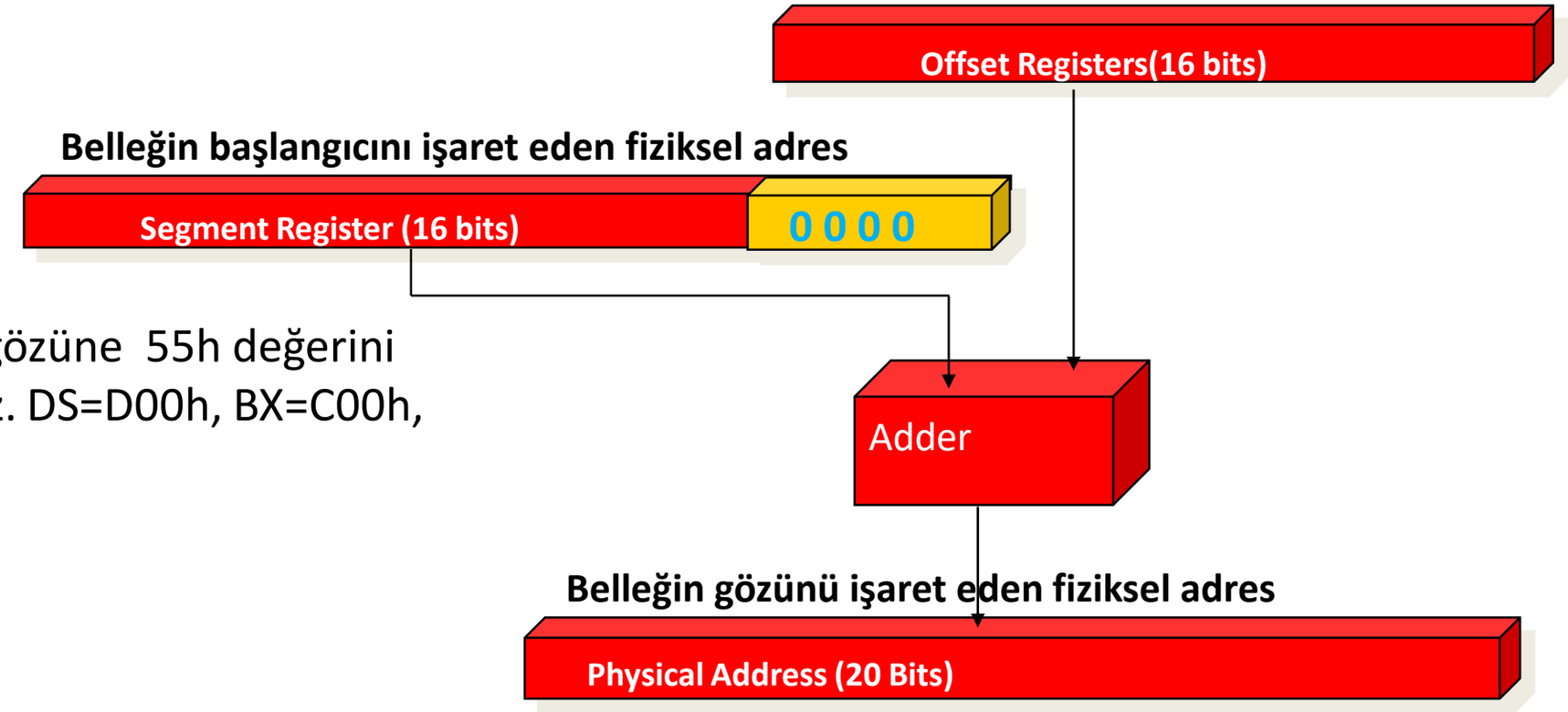
a) Belleğin başlangıcını işaret eden fiziksel adres nedir?
b) Belleğin gözünü işaret eden fiziksel adres nedir?
c) AL, data register'ındaki değer (88)h ise bunu bellek gözüne transfer eden assemble kodunu yazınız.

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

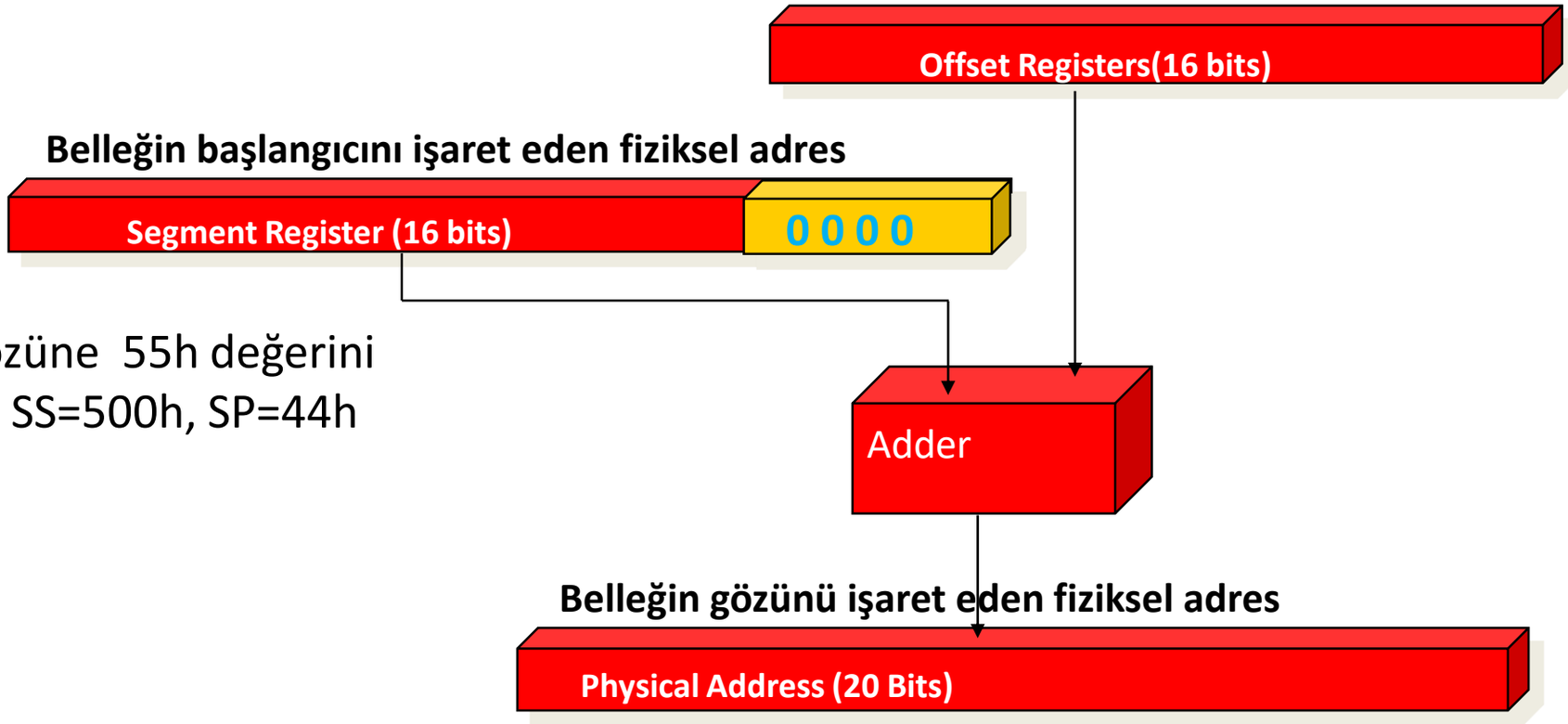


Soru:
Fiziksel adresi DCCCh olan belleğin gözüne 55h değerini yazan assemble programının yazınız. DS=D00h, BX=C00h, SI=C0h alınacaktır.

Yanıt:

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

Soru: Bellek Erişim Adresinin Hesaplanması

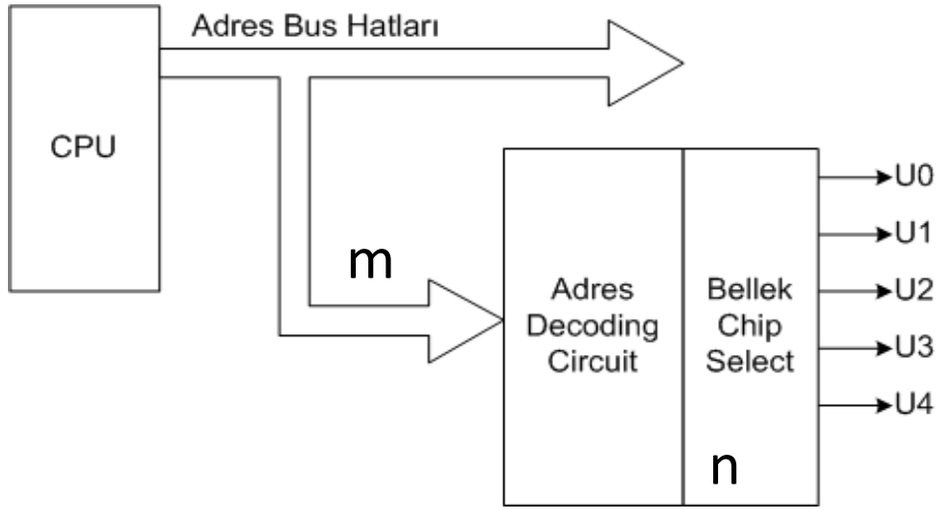


Soru:
Fiziksel adresi 5044h olan belleğin gözüne 55h değerini yazan assemble programının yazınız. SS=500h, SP=44h alınacaktır.

Yanıt:
MOV AX, 500h
MOV SS, AX
MOV BP, 44h
MOV SS:[BP], 55h

Stack segment register'ın gösterdiği bellek gözünü veri yazılırken BP, okunurken SP ofset register kullanılır.

Örnek 1.1

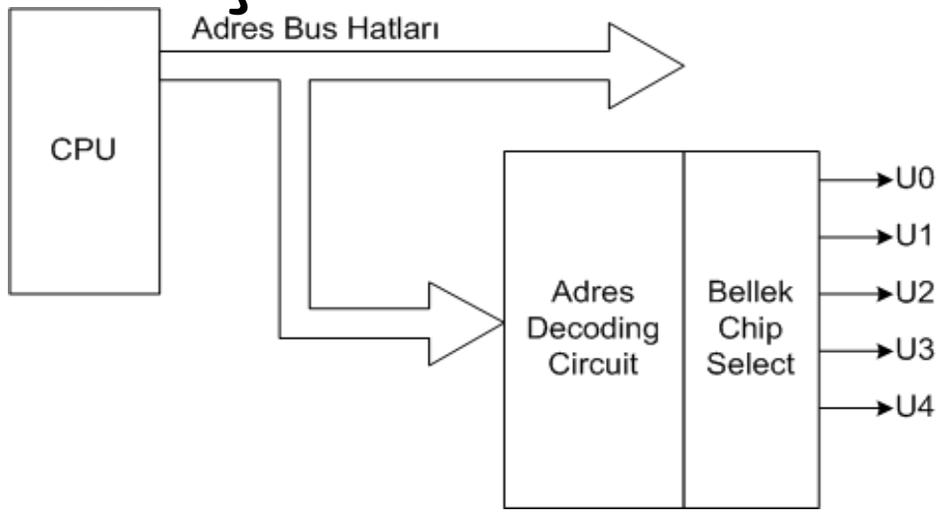


- U0=3KByte, U1=4Kbyte, U2=5Kbyte, U3=7Kbyte, U4=8Kbyte

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

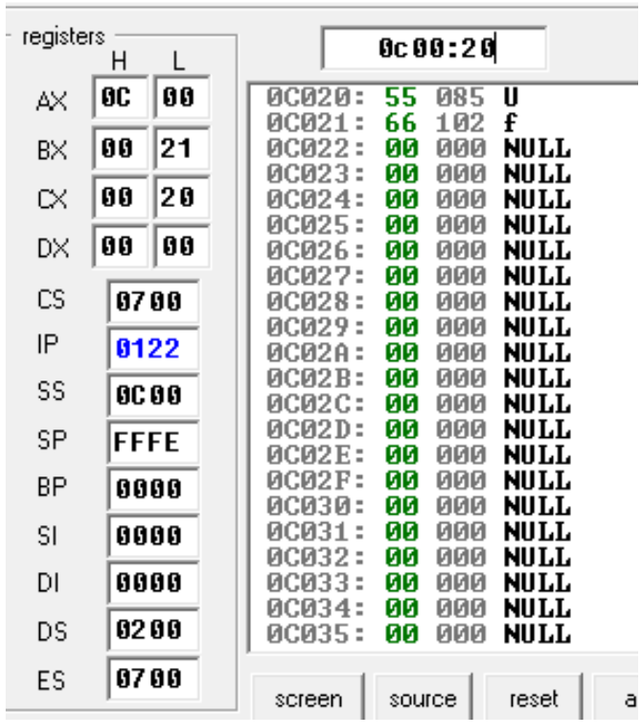
l) Aşağıda verilen kod yazılımını açıklayınız. 20 bitlik fiziksel adresini heaplayınız. Kod hangi belleği tanımlamaktadır? Indis register aralığı ilgili belleğin tanımlı aralığında mı?

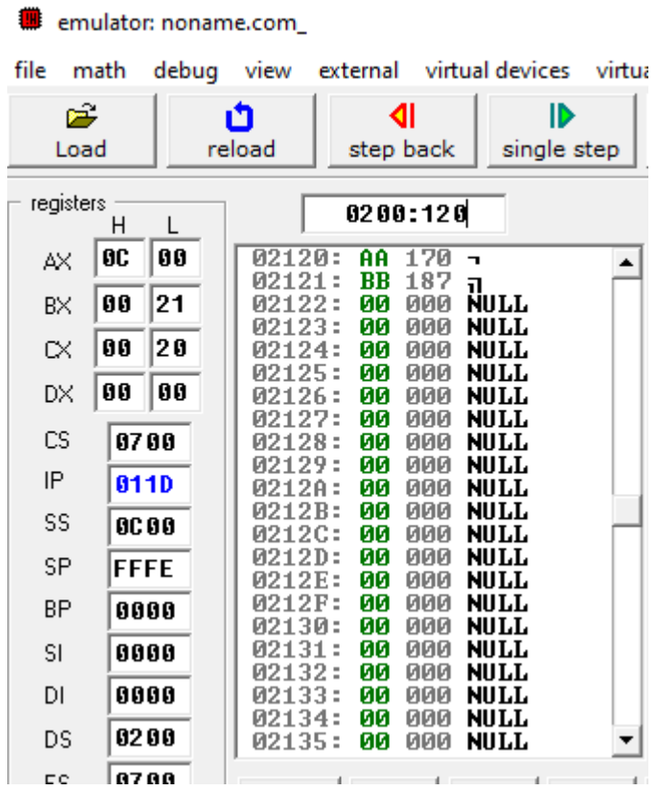
```
Mov Ax, 0400h
Mov ES, Ax
Mov Bx, 0DCCh
Mov [Bx], 0AAh
```

(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
```





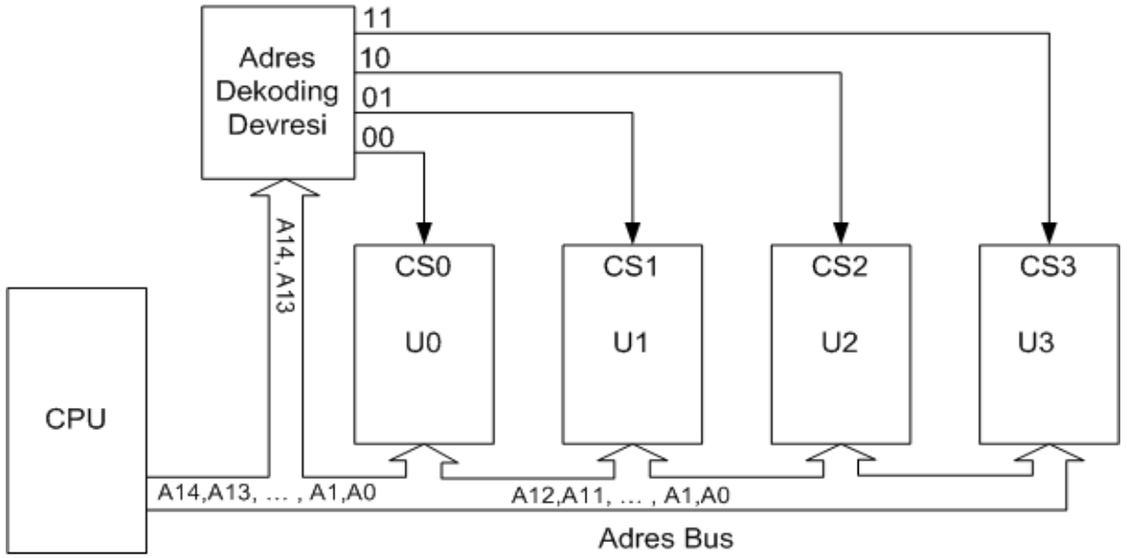
Data segment registerında gösterilen belleğin fiziksel adresi nedir?
Fiziksel adres=(DS)*10h
Fiziksel ades=200h*10h=2000h
Segment register'lara doğrudan veri yazılamaz. Data register'lar üzerinden yazılır.
Bellek gözünün adresi=2000h+120h=2120h
Bu bellek gözüne AA datası kayıt edilir.
Emulator: 200:0120;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

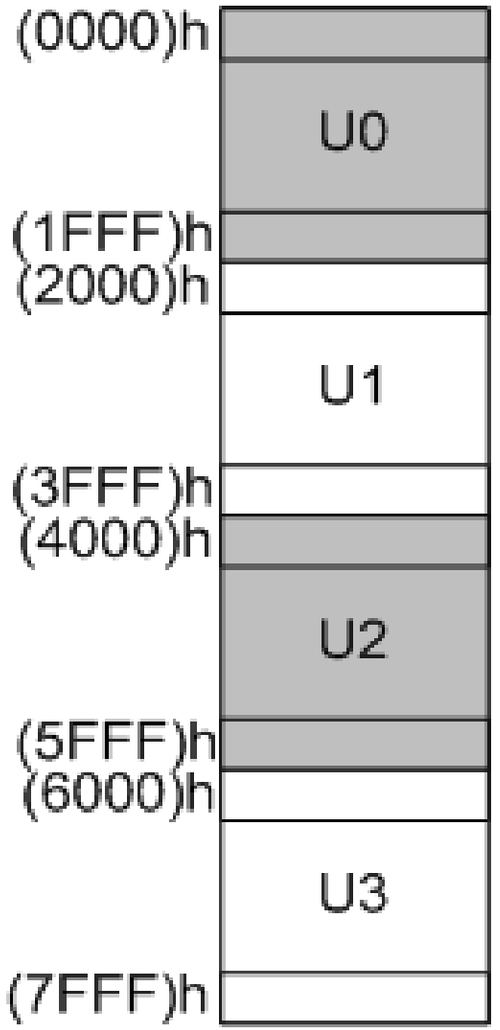
- Adres dekoding devresine gelen adres bus hatları indisi: A14, A13 ise, doğruluk tablosunu oluşturun. Hangi giriş durumunda hangi bellek seçilir.

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

A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0		Fiziksel (20bit) Başlangıç - Bitiş adresleri	(16bit) 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