

Microprocessors

Hafıza 3 ve 4

Özgür Erkili

8086 Adresleme Modları

Cpu, operandlara(yani datalara) birkaç farklı yol ile erişmektedir. Erişmek için kullanılan bu yollar, CPU'nun tasarımı ile ilgilidir ve cpu üretildikten sonra değiştirilemezdir.

80x86 işlemciler toplamda 7 adresleme moduna sahiptir:

1. Register
2. Immediate
3. Direct
4. Register Indirect
5. Based Relative
6. Indexed Relative
7. Based Indexed Relative

1. Register Adresleme Modu

Register adresleme modunda verilerin tutulmasında ve manipule edilmesinde registerlar kullanılmaktadır. Bu adresleme modu çalıştırıldığında belleğe erişim olmaz.

Ayrıca register adresleme modu diğer modlara göre daha hızlıdır.

Örneğin:

MOV BX,DX ; DX register içeriğini BX'e kopyala
MOV ES,AX ; AX register içeriğini ES'ye kopyala
ADD AL,BH ; AL = AL + BH işlemi yapılır

NOT : Bilinmelidir ki kaynak ve hedef register'inin ("MOV BX,DX" kodundaki BX hedef, DX kaynak register'dır.) boyutu birbiri ile uyusmalıdır. Yani 16 bit bir register kaynak olarak kullanılacaksa hedef register da 16 bit olmalıdır. "MOV CL,AX" kodunu denediğimizde hata alma sebebimiz budur.

2. Immediate Adresleme Modu

Immediate adresleme modunda kaynak operandlar sabit değerlidir. İsminden de anlaşılacağı gibi Immediate adresleme modunda instruction assemble edilirken operand anında opcode ile birlikte gelir. Bundan dolayı bu adresleme modu hızlı execute edilir. Ancak kullanımı kısıtlıdır.

Immediate adresleme modu segment ve flag registerları dışındaki registerlara veri yüklemek için kullanılır.

Örneğin:

MOV AX,2550H ; 2550H değerini AX'e taşı
MOV CX,625 ; 625 decimal değeri CX'e taşı
MOV BL,40H ; 40H değerini BL'ye taşı

Segment registerlara veri taşımak için, veri ilk önce genel amaçlı registerlara daha sonra ise segment registerlara taşınmalıdır.

Örneğin :

MOV AX,2550H

MOV DS,AX

Aşağıdaki kullanım hatalıdır :

MOV DS,0123H ; ILLEGAL

Göstermiş olduğum bu iki adresleme modlarında operandlar ya mikroişlemci içerisindeindedir ya da instruction'ın içerisinde ilistirilmiştir. Çoğu programda işleme girecek veriler, işlemci dışarısındaki bir hafızadan alınır(Buradaki hafıza RAM olarak geçmektedir). Veriye data segment üzerinde erişmenin birkaç yolu bulunmaktadır.

3. Direct Adresleme Modu

Direct adresleme modunda veri bellekteki herhangi bir konumda durmaktadır ve verinin bulunduğu konumun adresi instruction ile birlikte gelmektedir. Immediate adreslemede operand'ın değeri instruction ile gelirken direct adreslemede operand'ın bellekte bulunan adresi instruction ile gelmektedir. Aralarındaki temel fark budur.

Direct adreslemede instruction ile birlikte gelen adres offset adresidir. Data segment register'ında bulunan değer sola shift edilip offset adres ile toplanır ve verinin fiziksel bellek adresine erişilir.

Örneğin:

MOV DL,[2400] ; DS:2400H adresindeki veriyi DL'ye kopyala

Yukarıdaki durumda 2400 offset adresi DS(Data segment) ile birleştirilip fiziksel adres hesaplanır. Fark edildiği üzere adres değerleri kapalı parantez ile yazılmaktadır. Bu parantezin olmaması durumunda 8 bitlik DL register'ına 16 bitlik 2400 değerini kopyalamaya çalıştığımız için hata verecektir.

Örnek :

Find the physical address of the memory location and its contents after the execution of the following, assuming that DS = 1512H.

MOV AL,99H
MOV [3518],AL

İlk önce AL register'ına 99H değeri atanır. İkinci satırda ise AL register'ının içerisindeki değer DS:3518 (yani 1512:3518) mantıksal adres değerine taşınır. DS sola shift edilip offset adres ile toplandıktan sonra 18638H fiziksel adres'i elde edilir. ($15120H + 3518H = 18638H$). Yukarıdaki kodun ikinci satırı çalıştırıldıkten sonra 18638H fiziksel adresindeki değer 99H olacaktır.

4. Register Indirect Adresleme Modu

Register indirect Adresleme modunda operand'ın değerinin bulunduğu bellek adresi register'da tutulur. Bu amaçla kullanılan registerlar sırasıyla SI, DI ve BX'tir. Bu üç register pointer olarak kullanıldığında bellek konumunun offset değerini tutacaklardır. Bu sebeple DS ile birleştirilip 20 bitlik fiziksel adresin elde edilmesi gerekmektedir.

Örneğin:

MOV AL,[BX] ; AL register'ına DS:BX'in işaret ettiği bellek
; adresindeki değer kopyalanır.

NOT: Register'ın offset'i gösterdiğini belirtmek için köşeli parantezin içine yazılmalıdır.

Yukarıdaki kodda BX, sola shift edilmiş DS ile toplanır ve fiziksel adres elde edilir.

Aynı kural SI ve DI registerları kullanılarak da yapılabilir:

MOV CL,[SI] ; DS:SI içerisindeki değeri CL'ye kopyala
MOV [DI],AH ; AH içerisindeki değeri DS:DI adresine kopyala

Örnek :

Assume that DS = 1120, SI = 2498, and AX = 17FE. Show the contents of memory locations after the execution of

MOV [SI],AX

AX register'inin içerisindeki değer DS:SI ve DS:SI +1 mantıksal adresine taşınır. Burada hem DS:SI ve hem DS:SI +1 adresine taşınmasının sebebi belleğin en fazla 8 bit değer taşımasından dolayıdır. AX register'ı 16 bitlik bir register olduğu için veri, Little endian dönüşümüne göre yerleştirilecektir. Fiziksel adres DS sola shift edilip SI eklendikten sonra 13698H olacaktır. Little endian dönüşümüne göre 13698H adresi küçük byte değeri olan FE değerini tutarken 13699H adresi en büyük byte değeri olan 17 değerini tutacaktır.

5. Based Relative Adresleme Modu

Based relative adresleme modunda BX ve BP base register'ları efektif adresin hesaplanması için kullanılırlar. Fiziksel adres hesaplamasında kullanılan varsayılan segmentler BX için DS, BP için SS'dir.

Örneğin:

MOV CX,[BX]+10 ; DS:BX+10 ve DS:BX+10+1 adresindeki
; değeri CX'e taşı.
; Fiziksel Adres = DS + BX + 10

CX, 16 bitlik bir register olduğu için CL ve CH kısmı sırayla doldurulacaktır. Yani CL'nin içine DS:BX+10 adresindeki değer taşınırken CH'nin içine DS:BX+10+1 adresindeki değer taşınır.

Alternatif olarak "MOV AL,[BP+5]" ya da "MOV AL,5[BP]" kodu kullanılabilir.

6. Indexed Relative Adresleme Modu

Indexed relative adresleme modu base adresleme moduna çok benzemektedir. Temel farkı DI ve SI registerleri offset adresini tutarlar.

Örneğin :

MOV DX,[SI]+5 ; PA(Fiziksel adres) = DS + SI + 5

MOV CL,[DI]+20 ; PA = DS + DI + 20

Örnek:

Assume that DS = 4500, SS = 2000, BX = 2100, SI = 1486, DI = 8500, BP = 7814, and AX = 2512. Show the exact physical memory location where AX is stored in each of the following. All values are in hex.

- (a) MOV [BX]+20,AX (b) MOV [SI]+10,AX
- (c) MOV [DI]+4,AX (d) MOV [BP]+12,AX

Her madde için "PA = Segment register(sola shift) + offset register + yer değiştirme değeri(displacement)" kuralı uygulanacaktır.

- a) DS:BX+20 = 47120 -> 12 ve 47121 -> 25
- b) DS:SI+10 = 46496 -> 12 ve 46497 -> 25
- c) DS:DI+4 = 4D504 -> 12 ve 4D505 -> 25
- d) SS:BP+12 = 27826 -> 12 ve 27827 -> 25

7. Based Indexed Adresleme Modu

Based indexed adresleme modu based adresleme ile indexed adresleme modunun birleştirilmiş halidir. Bu adresleme modunda bir base register bir de index register kullanılır.

Örneğin :

```
MOV CL,[BX][DI]+8 ; PA = DS(sol shift) + BX + DI + 8  
MOV CH,[BX][SI]+20 ; PA = DS(sol shift) + BX + SI +20  
MOV AH,[BP][DI]+12 ; PA = DS(sol shift) + BP + DI +12  
MOV AH,[BP][SI]+29 ; PA = DS(sol shift) + BP + SI +29
```

Yukarıdaki kodlar daha farklı şekilde de yazılabilir. Örneğin son örnekteki kod "MOV AH,[BP+SI+29]" ya da "MOV AH,[SI+BP+29]" olarak yazılabilir.

NOT: "MOV AX,[SI][DI]+ displacement" kullanımı yasaktır.

Yukarıdaki çoğuörnekte MOV instruction'unun kullanılma sebebi konuyu daha iyi açıklamak içindir. Diğer instruction'lar adresleme modlarını destekliyorsa yukarıdaki örnekler kullanılabilir. "ADD DL,[BX]" kodu örnek verilebilir.

Table 1-3: Offset Registers for Various Segments

Segment register:	CS	DS	ES	SS
Offset register(s):	IP	SI, DI, BX	SI, DI, BX	SP, BP

Segment Override

Table 1-3, 80x86 işlemcide offset registerların 4 segment registerları ile kullanılabileceğini özetlemektedir. 80x86 CPU segmentlerin override edilmesine (varsayılan atama segmentlerini geçersiz kılıp yerine farklı segment kullanmasına) izin vermektedir. Bunu yapmak için kullanılacak segmentin kod üzerinde belirtilmesi gereklidir. Örneğin "MOV AL,[BX]" komutunda fiziksel adrese erişmek için DS kullanılır ve DS:BX mantıksal adresinden fiziksel adres hesaplanır. Daha önce belirtildiği gibi DS, BX'i göstermek için kullanılan varsayılan segmenttir. Bunu override etmek için istenilen segment instruction içerisinde belirtilmelidir. Örneğin "MOV AL,ES:[BX]" komutunda AL register'ına DS:BX'deki adres değeri taşınmak yerine ES:BX'deki adres değeri taşınır. Table 1-4 segment override hakkında daha çok örnek verirken table 1-5 8086/88'deki adresleme modlarını özetlemektedir.

Table 1-4: Sample Segment Overrides

Instruction	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

Table 1-5: Summary of 80x86 Addressing Modes

Addressing Mode	Operand	Default Segment
Register	reg	none
Immediate	data	none
Direct	[offset]	DS
Register indirect	[BX] [SI] [DI]	DS DS DS
Based relative	[BX]+disp [BP]+disp	DS SS
Indexed relative	[DI]+disp [SI]+disp	DS DS
Based indexed relative	[BX][SI]+disp [BX][DI]+disp [BP][SI]+ disp [BP][DI]+ disp	DS DS SS SS

Direktifler ve Örnek Programlar

Direktifler, assembler'a assembly dili instructionlarını makine koduna çevirirken nasıl davranışması gereği konusunda yönergeler veren özel ifadelerdir.

Bir assembly dili instruction'u dört alan içerir:

[label:] mnemonic [operandlar] [;Yorum satırı]

Label alanı programın satır konumunu belirtmek için kullanılır ve 31 karakteri geçmemelidir. Direktifler için kullanılan Label'ların sonuna iki nokta(:) koymaya gerek yoktur ama satır opcode oluşturan instruction içeriyorsa label'dan sonra iki nokta konulmalıdır.

Assembly dilinde mnemonic(instruction) ve operand alanları programın gerçek işi yaptığı ve görevlerin yerine getirildiği kısımlardır. Örneğin:

ADD AL,BL
MOV AX,6764

ADD ve MOV mnemonic opcode iken "AL,BL" ve "AX,6764" ise operand'lardır. Mnemonic ve operand içermek yerine bu iki alan assembler psödo-instruction'ları ya da direktifleri içerebilir. Bunlar programı assembler'da düzenlemek için kullanılır. **Direktifler makine kodu üretmezler ve sadece assembler tarafından kullanılır.**

Instruction'lar ise makine kodu üretirler. DB, END ve ENDP gibi ifadeler direktiflere örnek verilebilir.

Yorum satırı ";" ile başlar. Diğer program dillerindeki görevde sahiptir.

Model Tanımı

MODEL direktifi, programda kullanılacak bellek modelini belirtmekte kullanılır.

.MODEL SMALL

SMALL, assembly dilinde en yaygın kullanılan bellek modelidir. SMALL; kod için maksimum 64KB, veri(data) için maksimum 64KB alan kullanır.

.MODEL MEDIUM

Data kısmı maksimum 64KB bellek alanına sahipken kod kısmı 64KB belleği aşabilir.

.MODEL COMPACT

Data kısmı 64KB belleği aşabilirken kod kısmı maksimum 64KB alana sahiptir.

.MODEL LARGE

Data ve kod kısmı 64KB belleği aşabilir ancak tek bir veri kümesi 64KB değeri aşmamalıdır. Bu model, programın büyük boyutlu verilerle çalışabilecegi anlamına gelmektedir ancak tek bir veri setinin 64KB'ı aşmaması sınırlaması getirir.

.MODEL HUGE

Data ve kod kısmı 64KB belleği aşabilir. Data itemleri(array gibi) 64KB belleği aşabilir.

.MODEL TINY

COM dosyalarında kullanılan bu model tipinde data ve kod kısmı 64KB alana sığmalıdır.

Segment Tanımı

Daha önce bahsettiğim üzere 80x86 cpu'nundört segment register'i bulunmaktadır. Bunlar CS, DS, SS ve ES'dir. Bir assembly dil programının her satırı bu segmentlerden birine karşılık gelmelidir. Basitleştirilmiş segment tanımı formatı üç basit direktif kullanır : ".CODE" , " .DATA" ve " .STACK" . Bu direktifler sırasıyla CS, DS ve SS register'larına karşılık gelmektedir.

Programdaki Segmentler

;THE FORM OF AN ASSEMBLY LANGUAGE PROGRAM

;NOTE: USING SIMPLIFIED SEGMENT DEFINITION

Define Byte, .MODEL SMALL
data için ayrılan .STACK 64 \Rightarrow Bu direktit bellekten stack için 64 byte yer ayırm
bellek boyutunu itade eder

	.DATA	DATA segment
DATA1	DB 52H	
DATA2	DB 29H	Değişkenin taarflanmış ama değer atanması almadığını belirtir.
SUM	DB ?	
	.CODE	Code segment başlangıcı
MAIN	PROC FAR	;this is the program entry point
	MOV AX,@DATA	;load the data segment address
	MOV DS,AX	;assign value to DS
	MOV AL,DATA1	;get the first operand
	MOV BL,DATA2	;get the second operand
	ADD AL,BL	;add the operands
	MOV SUM,AL	;store the result in location SUM
	MOV AH,4CH	;set up to return to DOS
	INT 21H	;
MAIN	ENDP	
	END MAIN	;this is the program exit point

Write, run, and analyze a program that adds 5 bytes of data and saves the result. The data should be the following hex numbers: 25, 12, 15, 1F, and 2B.

PAGE 60,132 *=> Debug sırasında yazıcıya parametre olarek gönderilir (Satırlar 10 ile 255 arası) iken satır 60 ile 132 arası*
TITLE PROG2-1 (EXE) PURPOSE: ADDS 5 BYTES OF DATA
.MODEL SMALL
.STACK 64 *→ Programın İsmi*

DATA_IN DB 25H,12H,15H,1FH,2BH
SUM DB ?

.CODE

MAIN PROC FAR
MOV AX,@DATA
MOV DS,AX
MOV CX,05 ;set up loop counter CX=5
MOV BX,OFFSET DATA_IN ;set up data pointer BX
MOV AL,0 ;initialize AL
AGAIN: ADD AL,[BX] ;add next data item to AL
INC BX ;make BX point to next data item
DEC CX ;decrement loop counter
JNZ AGAIN ;jump if loop counter not zero
MOV SUM,AL ;load result into sum
MOV AH,4CH ;set up return
INT 21H ;return to DOS

MAIN ENDP
END MAIN

Write and run a program that adds four words of data and saves the result. The values will be 234DH, 1DE6H, 3BC7H, and 566AH. Use DEBUG to verify the sum is D364.

TITLE PROG2-2 (EXE) PURPOSE: ADDS 4 WORDS OF DATA
PAGE 60,132

.MODEL SMALL
.STACK 64

DATA_IN .DATA
DW 234DH,1DE6H,3BC7H,566AH
ORG 10H
SUM DW ?

MAIN .CODE
PROC FAR
MOV AX,@DATA
MOV DS,AX
MOV CX,04
MOV DI,OFFSET DATA_IN
MOV BX,00
ADD LP: ADD BX,[DI]
INC DI
INC DI
DEC CX
JNZ ADD_LP
MOV SI,OFFSET SUM
MOV [SI],BX
MOV AH,4CH
INT 21H
ENDP
END MAIN

;set up loop counter CX=4
;set up data pointer DI
;initialize BX
;add contents pointed at by [DI] to BX
;increment DI twice
;to point to next word
;decrement loop counter
;jump if loop counter not zero
;load pointer for sum
;store in data segment
;set up return
;return to DOS

DW 4 byte alan ayırr. → Offset 0(s,j,r) kabul edilir.

→ ORG(Origin) direktiti datanın offset adresini ayırmak için kullanılır

DATA_IN yerine SUM olsaydı DI ya 10 değer yüklenirdi.

Write and run a program that transfers 6 bytes of data from memory locations with offset of 0010H to memory locations with offset of 0028H.

TITLE PROG2-3 (EXE) PURPOSE: TRANSFERS 6 BYTES OF DATA
PAGE 60,132

.MODEL SMALL
.STACK 64
.DATA
ORG 10H
DATA_IN DB 25H,4FH,85H,1FH,2BH,
ORG 28H
COPY DB 0C4H

MAIN .CODE
PROC FAR
MOV AX,@DATA
MOV DS,AX
MOV SI,OFFSET DATA_IN
MOV DI,OFFSET COPY
MOV CX,06H
MOV_LOOP: MOV AL,[SI]
MOV [DI],AL
INC SI
INC DI
DEC CX
JNZ MOV_LOOP
MOV AH,4CH
INT 21H
ENDP
END MAIN

;SI points to data to be copied
;DI points to copy of data
;loop counter = 6
;move the next byte from DATA area to AL
;move the next byte to COPY area
;increment DATA pointer
;increment COPY pointer
;decrement LOOP counter
;jump if loop counter not zero
;set up to return
;return to DOS

→ Basında 0(s,j,r) olmasıın
Sebebi sayı olduğunu belirtmek
icindir.

6 DUP(?) → 6 data "?" yazmak yerine kullanıldı

FAR and NEAR Jump

Kontrol ifadesindeki sıçrama code segment'in kapsadığı bellek alanının içerisinde ise buna **NEAR jump**, Kontrol ifadesindeki sıçrama code segment'in kapsadığı bellek alanının dışında ise buna **FAR jump** denir.

NEAR jump'da CS:IP mantıksal adresinde CS sabit kalırken IP değişir. FAR jump'da CS:IP mantıksal adresinde CS ve IP değişir.

Conditional Jumps (ÖNEMLİ, EZBERLE!)

Table 2-1: 8086 Conditional Jump Instructions

Mnemonic	Condition Tested	"Jump IF ..."
JA/JNBE	(CF = 0) and (ZF = 0)	above/not below nor zero
JAE/JNB	CF = 0	above or equal/not below
JB/JNAE	CF = 1	below/not above nor equal
JBE/JNA	(CF or ZF) = 1	below or equal/not above
JC	CF = 1	carry
JE/JZ	ZF = 1	equal/zero
JG/JNLE	((SF xor OF) or ZF) = 0	greater/not less nor equal
JGE/JNL	(SF xor OF) = 0	greater or equal/not less
JL/JNGE	(SF xor OR) = 1	less/not greater nor equal
JLE/JNG	((SF xor OF) or ZF) = 1	less or equal/not greater
JNC	CF = 0	not carry
JNE/JNZ	ZF = 0	not equal/not zero
JNO	OF = 0	not overflow
JNP/JPO	PF = 0	not parity/parity odd
JNS	SF = 0	not sign
JO	OF = 1	overflow
JP/JPE	PF = 1	parity/parity equal
JS	SF = 1	sign

CMP komutu AL'den 61H değerini çıkarır ama değeri yazmaz. Onun yerine çıkarma işlemi sonucu 0 ise flag register'i değiştirir.

0005	8A 47 02	AGAIN:	MOV	AL,[BX]+2
0008	3C 61		CMP	AL,61H
000A	72 06	Conditional Jump	JB	NEXT
000C	3C 7A		CMP	AL,7AH
000E	77 02		JA	NEXT
0010	24 DF		AND	AL,ODFH
0012	88 04	NEXT:	MOV	[SI],AL

Unconditional JUMP (JMP)

1. SHORT JUMP, which is specified by the format "JMP SHORT label". This is a jump in which the address of the target location is within -128 to +127 bytes of memory relative to the address of the current IP. In this case, the opcode is EB and the operand is 1 byte in the range 00 to FF. The operand byte is added to the current IP to calculate the target address. If the jump is backward, the operand is in 2's complement. This is exactly like the J condition case. Coding the directive "short" makes the jump more efficient in that it will be assembled into a 2-byte instruction instead of a 3-byte instruction.

changes IP
↗

2. NEAR JUMP, which is the default, has the format "JMP label". This is a near jump (within the current code segment) and has the opcode E9. The target address can be any of the addressing modes of direct, register, register indirect, or memory indirect:
 - (a) Direct JUMP is exactly like the short jump explained earlier, except that the target address can be anywhere in the segment within the range +32767 to -32768 of the current IP.
 - (b) Register indirect JUMP; the target address is in a register. For example, in "JMP BX", IP takes the value BX.
 - (c) Memory indirect JMP; the target address is the contents of two memory locations pointed at by the register. Example: "JMP [DI]" will replace the IP with the contents of memory locations pointed at by DI and DI+1.

3. FAR JUMP which has the format "JMP FAR PTR label". This is a jump out of the current code segment, meaning that not only the IP but also the CS is replaced with new values.

Natımlı legendiyseniz
beğenmeyi ve kanalıma
abane olmayın umutmayın