Bölüm 3.2: Assembly

Mikroişlemciler

Assembly dili | Komut satırının yapısı | Komut ve direktif | İşlenen türleri | Örtük işlenenler | Komut süreleri | IP ve bayraklar | İlk program

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Assembly'nin yeri
Assembly dilinin makine dili ve yüksek düzeyli dillerle ilişkisini açıklamak.

Satır yapısı
Etiket, komut, işlenen ve açıklamadan oluşan satırları okumak; komutu direktiften ayırmak.

İşlenenler
Yazmaç, anlık değer ve bellek işlenenlerini ve komutların örtük işlenenlerini tanımak.

Komut süresi
Yazmaç ve bellek işlenenli komutların süre farkını saat turuyla hesaplamak.

IP ve bayraklar
IP'nin nasıl ilerlediğini ve hangi komutların bayrakları değiştirdiğini izlemek.

İlk program
emu8086 ile basit bir .COM programını yazıp adım adım çalıştırmak.

Assembly Dili

- **Düşük düzeyli** bir programlama dilidir: her komut, işlemcinin tek bir makine komutuna karşılık gelir.
- Makine kodundaki sayılar yerine akılda kalıcı **komut kısaltmaları** (mnemonic) kullanılır: `MOV`, `ADD`, `JMP`...
- Bilgisayarın yapısını (yazmaçlar, bellek, sistem yolu) **bilmeyi** gerektirir.
- Bilgisayarın temel yapısını ve bir programın gerçekte nasıl çalıştığını **anlamak** açısından önemlidir.
- Her işlemci ailesinin kendi assembly dili vardır; bu derste **Intel 8086** ve **emu8086** kullanılır.

Tek satır, üç görünüm

Assembly: `MOV AX, 5`

Onaltılık makine kodu: `B8 05 00`

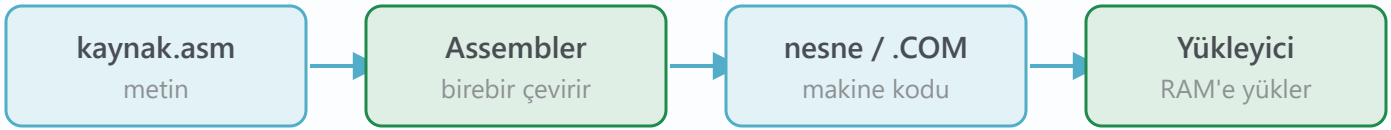
CPU'nun gördüğü:

`10111000 00000101 00000000`

16 bitlik 0005h, bellekte düşük bayt önce (05 00) saklanır.

ASSEMBLY DİLİ

Assembler ve emu8086



EXE için araya bağlayıcı (linker) girer: .obj → .exe

Assembler

Komut kısaltmalarını ve etiketleri makine koduna, değişken adlarını adreslere çevirir. Örnek: MASM, TASM, NASM.

emu8086

Editör + assembler + 8086 öykünücüsü (emulator). Yazmaçları, belleği ve bayrakları *adım adım* gösterir.

.COM / .EXE

.COM tek kesimli, en fazla 64 KB'lık basit biçimdir (ORG 100h). .EXE çok kesimli olabilir.

Bu derste örnekler emu8086 söz dizimiyle yazılır; çoğu MASM/TASM ile de uyumludur.

Neden Assembly?

Assembly dili ile oluşturulmuş bir program:

Artıları

+ Daha **az yer** kaplar (gereksiz kod yok).
+ Yalnızca yapması gereken işlemleri yaptığından **daha hızlıdır**.
+ Yüksek düzeyli dillerin yapamadığı işlemleri yapabilir: port G/Ç, kesme işleme, bayraklara doğrudan erişim.
+ Önyükleyici, aygıt sürücüsü, gömülü sistem gibi alanlarda vazgeçilmezdir.

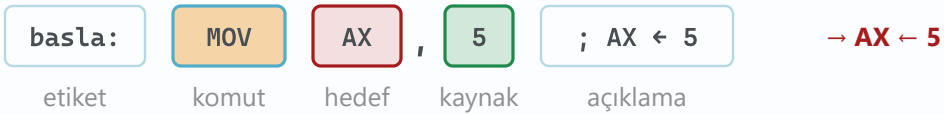
Eksileri

– **Taşınabilir değildir**: 8086 kodu ARM'da çalışmaz.
– Yazması, okuması ve hata ayıklaması zahmetlidir.
– Büyük projelerde verim düşüktür; modern derleyiciler çoğu zaman yeterince hızlı kod üretir.

Uygulamada genellikle **karma** yaklaşım kullanılır: program yüksek düzeyli dilde, zamanlaması kritik küçük parçalar assembly ile yazılır.

Komut Satırının Yapısı

```
[etiket:]  komut  [hedef], [kaynak]  [; açıklama]
```



Satır	Etiket	Komut	İşlenenler	Açıklama
`basla: MOV AX, 5 ; AX ← 5`	`basla`	`MOV`	`AX`, `5`	`; AX ← 5`
`INC AX`	—	`INC`	`AX`	—
`RET`	—	`RET`	yok	—

- İki işlenenli komutlarda sıra **hedef, kaynak**tır: `MOV AX, BX` → AX ← BX.
- Assembler büyük-küçük harfe duyarsızdır: `mov ax, bx` ile `MOV AX, BX` aynıdır.
- `;` işaretinden satır sonuna kadar her şey açıklamadır ve makine koduna çevrilmez.

Komut mu, Direktif mi?

Satır	Türü	Bellekte ne üretir?
`MOV AX, 5`	komut	makine kodu: B8 05 00
`ORG 100h`	direktif	hiçbir şey; konum sayacını 0100h yapar
`sayi DB 0`	direktif (veri tanımı)	1 bayt veri: 00
`k EQU 5`	direktif	hiçbir şey; k adı 5 değerini alır
`END`	direktif	hiçbir şey; kaynak burada biter
`PUTC 'A'`	makro çağrısı	makro gövdesindeki komutların kodu

Komut

İşlemcinin çalışma anında yürüttüğü işlem. Her biri bir makine komutuna çevrilir.

Direktif

Assembler'a verilen bilgi: nereye yerleşecek, ne kadar yer ayrılacak, adlar neyi gösterecek.

Makro

Derleme anında yerine birkaç satır yazılan, adı olan metin kalıbı.

Etiketler ve Adlar

```
basla:  MOV CX, 3    ; kod etiketi
don:    INC AX
        LOOP don     ; don = adres
        RET
sayi    DB 7         ; veri adı
```

- Kod etiketi iki nokta üst üste ile biter ve bir komutun adresini (CS içindeki bağıl konumunu) temsil eder; kendisi makine kodu üretmez.
- Değişken adı iki noktasız yazılır; adı, türü (DB, DW…) ve adresi birlikte saklanır.
- Ad harfle ya da alt çizgiyle başlar, rakamla başlayamaz; devamında harf, rakam ve alt çizgi olabilir.
- Komut, yazmaç ve direktif adları (MOV, AX, DB, END…) ad olarak kullanılamaz.
- Büyük-küçük harf ayrılmaz: `don` ile `DON` aynı etikettir; ikisini birden tanımlamak “iki kez tanımlı” hatası verir.

Ad	`_sayac`	`toplam2`	`2toplam`	`cx`	`son-deger`
Geçerli mi?	evet	evet	rakamla başlıyor	yazmaç adı	– bir işleçtir

YAZMAÇLARLA ÇALIŞMA

İşlenen Türleri ve Örtük İşlenenler

Yazmaç
`MOV AX, BX` : değer işlemcinin içinden gelir; en hızlı tür.

Anlık değer
`MOV AX, 5` : değer komutun baytlarının içindedir; yalnızca kaynak olabilir.

Bellek
`MOV AX, [BX]` : değer bellekten bir yol döngüsüyle okunur.

Komut	Yazılan işlenen	Örtük işlenen(ler)
`MUL BL`	BL	AL (çarpılan), AX (sonuç)
`DIV BL`	BL	AX (bölünen), AL (bölüm), AH (kalan)
`LOOP don`	don	CX (sayaç)
`CBW`	—	AL → AX
`PUSH AX`	AX	SS:SP (yığın tepesi)
`XLAT`	—	AL (indis ve sonuç), BX (tablo adresi)

Örtük işlenen komutta yazılmaz ama değişir: `MUL BL` AH'yi de siler. Bir yazmacın değerini korumak isteyen program, kullandığı komutların örtük işlenenlerini bilmelidir.

Kod İzleme: MUL, SHR ve LOOP'un Örtük İşlenenleri

```
1  ORG 100h
▶ MOV AL, 12
▶ MOV BL, 10
4  MUL BL          ; AX = AL × BL
5  MOV CL, 2
6  SHR AX, CL      ; sayı CL'de
7  MOV CX, 3       ; LOOP sayacı
8  don:
9  ADD AX, 1
10 LOOP don        ; CX azalır
11 RET
```

AX	BX	CX	CF	ZF
000Ch	000Ah	0000h	0	0

AX = 12 (onluk)

AL = 12, BL = 10. AH başlangıçta 0.

Adım 1 / 8

DEĞİŞKENLER

AX = 000Ch BX = 000Ah CX = 0000h

AX, CL ve CX bu programda komutların “gizli” işlenenleriydi; yazmaçların özel görevleri bu örtük kullanımlardan gelir.

Kod İzleme: MUL, SHR ve LOOP'un Örtük İşlenenleri

```
1  ORG 100h
2  MOV AL, 12
3  MOV BL, 10
▶ 4  MUL BL          ; AX = AL × BL
5  MOV CL, 2
6  SHR AX, CL        ; sayı CL'de
7  MOV CX, 3         ; LOOP sayacı
8  don:
9  ADD AX, 1
10 LOOP don          ; CX azalır
11 RET
```

AX	BX	CX	CF	ZF
0078h	000Ah	0000h	0	?

`AX = 120 (onluk)`

örtük: AL, AX

`MUL BL` yalnız BL'yi yazar; çarpılan örtük olarak AL, sonuç örtük olarak AX'tir: AX = 120 = **0078h**. Sonuç 8 bite sığdığı için CF = 0.

Adım 2 / 8

DEĞİŞKENLER

`AX = 0078h` `BX = 000Ah` `CX = 0000h`

AX, CL ve CX bu programda komutların “gizli” işlenenleriydi; yazmaçların özel görevleri bu örtük kullanımlardan gelir.

Kod İzleme: MUL, SHR ve LOOP'un Örtük İşlenenleri

```
1  ORG 100h
2  MOV AL, 12
3  MOV BL, 10
4  MUL BL          ; AX = AL × BL
▶  MOV CL, 2
▶  SHR AX, CL      ; sayı CL'de
7  MOV CX, 3       ; LOOP sayacı
8  don:
9  ADD AX, 1
10 LOOP don        ; CX azalır
11 RET
```

AX	BX	CX	CF	ZF
001Eh	000Ah	0002h	0	0

`AX = 30 (onluk)`

sayı CL'de

`SHR AX, CL` : 1'den büyük kaydırma sayısı yalnızca CL'de verilebilir. 120 ÷ 4 = 30 → AX = **001Eh**.

Adım 3 / 8

DEĞİŞKENLER

`AX = 001Eh` `BX = 000Ah` `CX = 0002h`

AX, CL ve CX bu programda komutların “gizli” işlenenleriydi; yazmaçların özel görevleri bu örtük kullanımlardan gelir.

Kod İzleme: MUL, SHR ve LOOP'un Örtük İşlenenleri

```
1  ORG 100h
2  MOV AL, 12
3  MOV BL, 10
4  MUL BL          ; AX = AL × BL
5  MOV CL, 2
6  SHR AX, CL      ; sayı CL'de
▶  MOV CX, 3       ; LOOP sayacı
8  don:
9  ADD AX, 1
10 LOOP don        ; CX azalır
11 RET
```

AX	BX	CX	CF	ZF
001Eh	000Ah	0003h	0	0

AX = 30 (onluk)

CX = 3: LOOP'un örtük sayacı. CL de CX'in parçası olduğundan önceki 2 değeri silindi.

Adım 4 / 8

DEĞİŞKENLER

AX = 001Eh | BX = 000Ah | CX = 0003h

AX, CL ve CX bu programda komutların “gizli” işlenenleriydi; yazmaçların özel görevleri bu örtük kullanımlardan gelir.

Kod İzleme: MUL, SHR ve LOOP'un Örtük İşlenenleri

```
1  ORG 100h
2  MOV AL, 12
3  MOV BL, 10
4  MUL BL          ; AX = AL × BL
5  MOV CL, 2
6  SHR AX, CL      ; sayı CL'de
7  MOV CX, 3       ; LOOP sayacı
8  don:
▶  ADD AX, 1
▶  LOOP don        ; CX azalır
11 RET
```

AX	BX	CX	CF	ZF
001Fh	000Ah	0002h	0	0

`AX = 31 (onluk)`

1. tur: AX + 1 = 31; `LOOP` CX'i 1 azaltır: CX = 2, sıfır değil → `don`.

Adım 5 / 8

DEĞİŞKENLER

`AX = 001Fh` `BX = 000Ah` `CX = 0002h`

AX, CL ve CX bu programda komutların “gizli” işlenenleriydi; yazmaçların özel görevleri bu örtük kullanımlardan gelir.

Kod İzleme: MUL, SHR ve LOOP'un Örtük İşlenenleri

```
1  ORG 100h
2  MOV AL, 12
3  MOV BL, 10
4  MUL BL          ; AX = AL × BL
5  MOV CL, 2
6  SHR AX, CL      ; sayı CL'de
7  MOV CX, 3       ; LOOP sayacı
8  don:
▶  ADD AX, 1
▶  LOOP don        ; CX azalır
11 RET
```

AX	BX	CX	CF	ZF
0020h	000Ah	0001h	0	0

AX = 32 (onluk)

2. tur: AX + 1 = 32; `LOOP` CX'i 1 azaltır: CX = 1, sıfır değil → `don`.

Adım 6 / 8

DEĞİŞKENLER

AX = 0020h | BX = 000Ah | CX = 0001h

AX, CL ve CX bu programda komutların “gizli” işlenenleriydi; yazmaçların özel görevleri bu örtük kullanımlardan gelir.

Kod İzleme: MUL, SHR ve LOOP'un Örtük İşlenenleri

```
1  ORG 100h
2  MOV AL, 12
3  MOV BL, 10
4  MUL BL          ; AX = AL × BL
5  MOV CL, 2
6  SHR AX, CL      ; sayı CL'de
7  MOV CX, 3       ; LOOP sayacı
8  don:
▶  ADD AX, 1
▶  LOOP don        ; CX azalır
11 RET
```

AX	BX	CX	CF	ZF
0021h	000Ah	0000h	0	0

AX = 33 (onluk)

3. tur: AX + 1 = 33; `LOOP` CX'i 1 azaltır: CX = 0, döngü biter.

Adım 7 / 8

DEĞİŞKENLER

AX = 0021h | BX = 000Ah | CX = 0000h

AX, CL ve CX bu programda komutların “gizli” işlenenleriydi; yazmaçların özel görevleri bu örtük kullanımlardan gelir.

Kod İzleme: MUL, SHR ve LOOP'un Örtük İşlenenleri

```
1  ORG 100h
2  MOV AL, 12
3  MOV BL, 10
4  MUL BL          ; AX = AL × BL
5  MOV CL, 2
6  SHR AX, CL      ; sayı CL'de
7  MOV CX, 3       ; LOOP sayacı
8  don:
9  ADD AX, 1
10 LOOP don        ; CX azalır
▶  RET
```

AX	BX	CX	CF	ZF
0021h	000Ah	0000h	0	0

AX = 33 (onluk)

RET. Programda CX hiç azaltılmadı gibi görünse de LOOP onu üç kez değiştirdi: AX = **0021h** = 33.

Adım 8 / 8

DEĞİŞKENLER

AX = 0021h | BX = 000Ah | CX = 0000h

AX, CL ve CX bu programda komutların “gizli” işlenenleriydi; yazmaçların özel görevleri bu örtük kullanımlardan gelir.

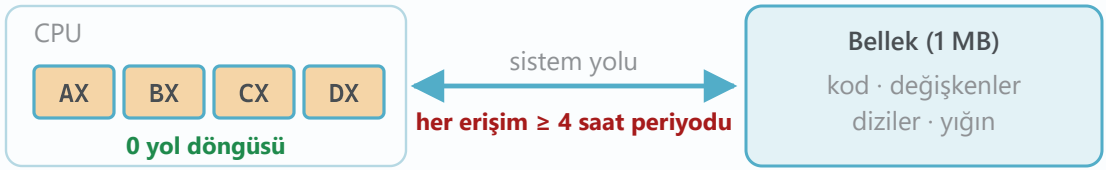
Yazmaç mı, Bellek mi?

Yazmaç

- CPU'nun içindedir; erişim için yol döngüsü gerekmez.
- Sayısı çok azdır (8086'da 14 adet 16 bitlik yazmaç).
- Ara sonuçlar ve sayaçlar için idealdir.

Bellek

- CPU'nun dışındadır; her erişim en az bir yol döngüsü (8086'da 4 saat periyodu) sürer.
- 1 MB'a kadar büyüktür.
- Program kodu, değişkenler, diziler ve yığın buradadır.



İyi assembly kodu sık kullanılan değerleri **yazmaçlarda** tutar, belleğe yalnızca gerektiğinde gider. Örneğin bir döngü sayacını bellekte değil `CX` 'te saklamak döngüyü hızlandırır.

14 yazmaç: 8 genel amaçlı + 4 kesim + IP + FLAGS.

Komut Süreleri: Yazmaç ve Bellek

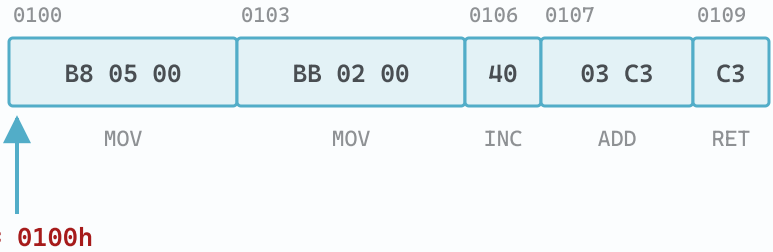
Komut (8086)	Saat turu
`MOV AX, BX`	2
`MOV AX, 5`	4
`MOV AX, sayi`	10
`MOV CX, [BX]`	8 + 5 = 13
`ADD AX, BX`	3
`ADD AX, sayi`	9 + 6 = 15
`INC CX`	2
`INC sayac` (sözcük)	15 + 6 = 21

- Bellek işlenenli komutlarda süreye **etkin adres hesabı** (EA) eklenir: doğrudan adres 6, `[BX]` 5, `[BX+SI]` 7 tur.
- Belleğe yazan komutlar daha uzundur: `INC sayac` önce okur, sonra yazar (iki yol döngüsü).
- AX'e doğrudan adresten yükleme kısa akümülatör biçimini kullanır: 10 tur.
- 1000 kez dönen döngüde sayaç CX'teyse artırma 1000 × 2 = 2000 saat turu, bellekteyse 1000 × 21 = 21 000 tur sürer; 5 MHz'de 0,4 ms yerine 4,2 ms.

Değerler Intel 8086 veri sayfasındaki en iyi durum süreleridir; tek adresteki sözcük erişimi ve önden getirme kuyruğunun boşalması süreyi uzatır. Oran değişmez: bellek işlenenli bir komut yazmaç işlenenli eşinden birkaç kat yavaştır.

Kod İzleme: IP Nasıl İlerler?

```
▶ ORG 100h
2 MOV AX, 5      ; 3 bayt
3 MOV BX, 2      ; 3 bayt
4 INC AX         ; 1 bayt
5 ADD AX, BX     ; 2 bayt
6 RET            ; 1 bayt
```



Program CS:0100h'ye yüklenir; **IP = 0100h** ilk komutu gösterir.

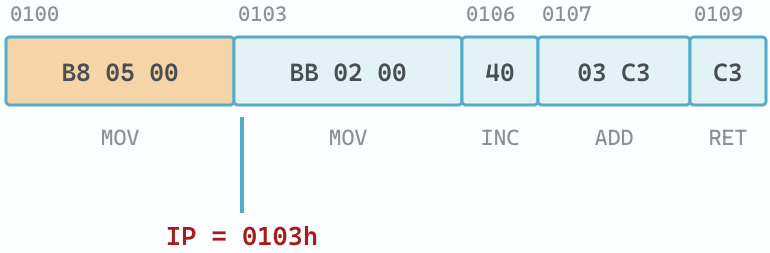
Adım 1 / 6

DEĞİŞKENLER

IP = **0100h** AX = **0000h** BX = **0000h** SP = **FFFEh**

Kod İzleme: IP Nasıl İlerler?

```
1  ORG 100h
▶  MOV AX, 5      ; 3 bayt
3  MOV BX, 2      ; 3 bayt
4  INC AX         ; 1 bayt
5  ADD AX, BX     ; 2 bayt
6  RET            ; 1 bayt
```



MOV AX, 5 (B8 05 00) getirilir: **IP = 0103h**, yürütülünce AX = 0005h.

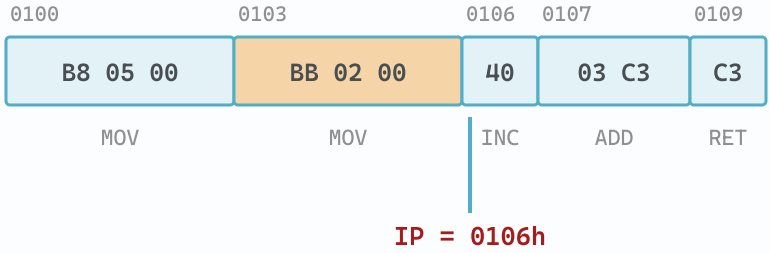
Adım 2 / 6

DEĞİŞKENLER

IP = 0103h | AX = 0005h | BX = 0000h | SP = FFFEh

Kod İzleme: IP Nasıl İlerler?

```
1 ORG 100h
2 MOV AX, 5      ; 3 bayt
▶ MOV BX, 2      ; 3 bayt
4 INC AX         ; 1 bayt
5 ADD AX, BX     ; 2 bayt
6 RET            ; 1 bayt
```



3 bayt daha: **IP = 0106h**, BX = 0002h.

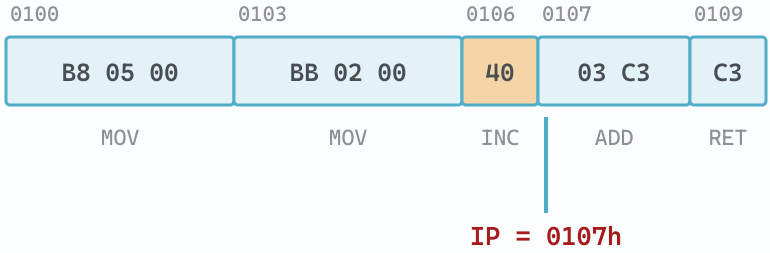
Adım 3 / 6

DEĞİŞKENLER

IP = 0106h | AX = 0005h | BX = 0002h | SP = FFFEh

Kod İzleme: IP Nasıl İlerler?

```
1 ORG 100h
2 MOV AX, 5      ; 3 bayt
3 MOV BX, 2      ; 3 bayt
▶ INC AX         ; 1 bayt
5 ADD AX, BX     ; 2 bayt
6 RET            ; 1 bayt
```



INC AX tek bayttır (40h): **IP = 0107h**, AX = 0006h.

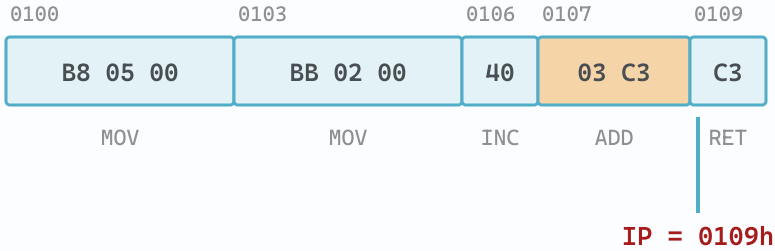
Adım 4 / 6

DEĞİŞKENLER

IP = 0107h | AX = 0006h | BX = 0002h | SP = FFFEh

Kod İzleme: IP Nasıl İlerler?

```
1 ORG 100h
2 MOV AX, 5      ; 3 bayt
3 MOV BX, 2      ; 3 bayt
4 INC AX         ; 1 bayt
▶ ADD AX, BX     ; 2 bayt
6 RET            ; 1 bayt
```



ADD AX, BX 2 bayt: **IP = 0109h**, AX = 0008h.

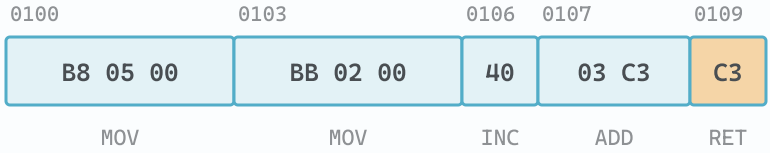
Adım 5 / 6

DEĞİŞKENLER

IP = 0109h | AX = 0008h | BX = 0002h | SP = FFFEh

Kod İzleme: IP Nasıl İlerler?

```
1 ORG 100h
2 MOV AX, 5      ; 3 bayt
3 MOV BX, 2      ; 3 bayt
4 INC AX         ; 1 bayt
5 ADD AX, BX     ; 2 bayt
▶ RET            ; 1 bayt
```



IP = 0000h (PSP'ye dönüş)

RET yığından 0000h'yi çeker: **IP = 0000h**, SP = 0000h. PSP:0000'daki INT 20h programı bitirir.

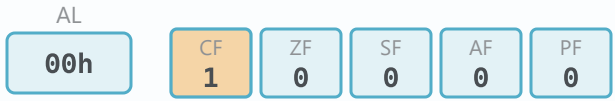
Adım 6 / 6

DEĞİŞKENLER

IP = 0000h | AX = 0008h | BX = 0002h | SP = 0000h

Kod İzleme: Hangi Komut Bayrağa Dokunur?

```
1  ORG 100h
▶ STC                ; CF = 1
3  MOV AL, 0FFh      ; bayraklar aynı
4  INC AL            ; CF'ye dokunmaz
5  MOV CX, 0         ; ZF aynı kalır
6  DEC AL
7  ADD AL, 1         ; CF yeni
8  CLC
9  NOT AL            ; bayraklar aynı
10 RET
```



yalnız CF

`STC` : CF = 1. Bayrağı doğrudan değiştiren komutlardan biri.

Adım 1 / 8

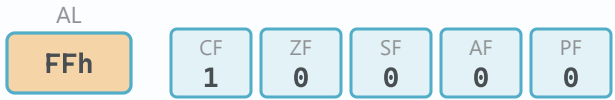
DEĞİŞKENLER

`AL = 00h` `CF = 1` `ZF = 0` `SF = 0`

MOV, NOT, XCHG, LEA, PUSH ve POP bayrakları değiştirmez; INC ve DEC CF dışındakileri değiştirir. Koşullu atlamadan hemen önce bayrağı hangi komutun ayarladığına bakılır.

Kod İzleme: Hangi Komut Bayrağa Dokunur?

```
1  ORG 100h
2  STC              ; CF = 1
▶ MOV AL, 0FFh      ; bayraklar aynı
4  INC AL           ; CF'ye dokunmaz
5  MOV CX, 0        ; ZF aynı kalır
6  DEC AL
7  ADD AL, 1        ; CF yeni
8  CLC
9  NOT AL           ; bayraklar aynı
10 RET
```



bayrak yok

`MOV AL, 0FFh`: AL değişti, bayrakların hiçbiri değişmedi. MOV bayrak etkilemez.

Adım 2 / 8

DEĞİŞKENLER

AL = FFh | CF = 1 | ZF = 0 | SF = 0

MOV, NOT, XCHG, LEA, PUSH ve POP bayrakları değiştirmez; INC ve DEC CF dışındakileri değiştirir. Koşullu atlamadan hemen önce bayrağı hangi komutun ayarladığına bakılır.

Kod İzleme: Hangi Komut Bayrağa Dokunur?

```
1  ORG 100h
2  STC               ; CF = 1
3  MOV AL, 0FFh      ; bayraklar aynı
▶  INC AL            ; CF'ye dokunmaz
5  MOV CX, 0         ; ZF aynı kalır
6  DEC AL
7  ADD AL, 1         ; CF yeni
8  CLC
9  NOT AL            ; bayraklar aynı
10 RET
```



CF korunur

`INC AL` : FFh + 1 = 00h → ZF = 1, AF = 1, PF = 1. CF hâlâ 1: INC ve DEC elde bayrağına dokunmaz.

Adım 3 / 8

DEĞİŞKENLER

AL = 00h | CF = 1 | ZF = 1 | SF = 0

MOV, NOT, XCHG, LEA, PUSH ve POP bayrakları değiştirmez; INC ve DEC CF dışındakileri değiştirir. Koşullu atlamadan hemen önce bayrağı hangi komutun ayarladığına bakılır.

Kod İzleme: Hangi Komut Bayrağa Dokunur?

```
1  ORG 100h
2  STC              ; CF = 1
3  MOV AL, 0FFh     ; bayraklar aynı
4  INC AL           ; CF'ye dokunmaz
▶  MOV CX, 0        ; ZF aynı kalır
6  DEC AL
7  ADD AL, 1        ; CF yeni
8  CLC
9  NOT AL           ; bayraklar aynı
10 RET
```

AL

00h

CF	ZF	SF	AF	PF
1	1	0	1	1

bayrak yok

`MOV CX, 0` : sonuç sıfır olsa da ZF yeniden hesaplanmaz; önceki komutun ZF'si kalır.

Adım 4 / 8

DEĞİŞKENLER

`AL = 00h` `CF = 1` `ZF = 1` `SF = 0`

MOV, NOT, XCHG, LEA, PUSH ve POP bayrakları değiştirmez; INC ve DEC CF dışındakileri değiştirir. Koşullu atlamadan hemen önce bayrağı hangi komutun ayarladığına bakılır.

Kod İzleme: Hangi Komut Bayrağa Dokunur?

```
1  ORG 100h
2  STC              ; CF = 1
3  MOV AL, 0FFh     ; bayraklar aynı
4  INC AL           ; CF'ye dokunmaz
5  MOV CX, 0        ; ZF aynı kalır
▶  DEC AL
7  ADD AL, 1        ; CF yeni
8  CLC
9  NOT AL           ; bayraklar aynı
10 RET
```



CF korunur

`DEC AL` : 00h − 1 = FFh → SF = 1, ZF = 0. CF yine 1: DEC ödünç almayı CF'ye yazmaz.

Adım 5 / 8

DEĞİŞKENLER

AL = FFh | CF = 1 | ZF = 0 | SF = 1

MOV, NOT, XCHG, LEA, PUSH ve POP bayrakları değiştirmez; INC ve DEC CF dışındakileri değiştirir. Koşullu atlamadan hemen önce bayrağı hangi komutun ayarladığına bakılır.

Kod İzleme: Hangi Komut Bayrağa Dokunur?

```
1  ORG 100h
2  STC              ; CF = 1
3  MOV AL, 0FFh     ; bayraklar aynı
4  INC AL           ; CF'ye dokunmaz
5  MOV CX, 0        ; ZF aynı kalır
6  DEC AL
▶  ADD AL, 1        ; CF yeni
8  CLC
9  NOT AL           ; bayraklar aynı
10 RET
```



hepsi yeni

`ADD AL, 1` : FFh + 1 = 100h → AL = 00h. ADD bütün durum bayraklarını yeniden yazar: CF = 1 bu kez ADD'in eldesi.

Adım 6 / 8

DEĞİŞKENLER

AL = 00h | CF = 1 | ZF = 1 | SF = 0

MOV, NOT, XCHG, LEA, PUSH ve POP bayrakları değiştirmez; INC ve DEC CF dışındakileri değiştirir. Koşullu atlamadan hemen önce bayrağı hangi komutun ayarladığına bakılır.

Kod İzleme: Hangi Komut Bayrağa Dokunur?

```
1  ORG 100h
2  STC              ; CF = 1
3  MOV AL, 0FFh     ; bayraklar aynı
4  INC AL           ; CF'ye dokunmaz
5  MOV CX, 0        ; ZF aynı kalır
6  DEC AL
7  ADD AL, 1        ; CF yeni
▶  CLC
9  NOT AL           ; bayraklar aynı
10 RET
```

AL

00h

CF	ZF	SF	AF	PF
0	1	0	1	1

yalnız CF

`CLC` : CF = 0.

Adım 7 / 8

DEĞİŞKENLER

AL = 00h | CF = 0 | ZF = 1 | SF = 0

MOV, NOT, XCHG, LEA, PUSH ve POP bayrakları değiştirmez; INC ve DEC CF dışındakileri değiştirir. Koşullu atlamadan hemen önce bayrağı hangi komutun ayarladığına bakılır.

Kod İzleme: Hangi Komut Bayrağa Dokunur?

```
1  ORG 100h
2  STC              ; CF = 1
3  MOV AL, 0FFh     ; bayraklar aynı
4  INC AL           ; CF'ye dokunmaz
5  MOV CX, 0        ; ZF aynı kalır
6  DEC AL
7  ADD AL, 1        ; CF yeni
8  CLC
▶  NOT AL           ; bayraklar aynı
10 RET
```



`NOT AL` : AL = FFh, ama ZF hâlâ 1. NOT hiçbir bayrağı değiştirmez; sonucu sınamak için ardından `TEST AL, AL` ya da `CMP` gerekir.

Adım 8 / 8

DEĞİŞKENLER

`AL = FFh` `CF = 0` `ZF = 1` `SF = 0`

MOV, NOT, XCHG, LEA, PUSH ve POP bayrakları değiştirmez; INC ve DEC CF dışındakileri değiştirir. Koşullu atlamadan hemen önce bayrağı hangi komutun ayarladığına bakılır.

İLK PROGRAM

.COM Program Şablonu

```
ORG 100h            ; .COM: kod 0100h'den başlar

MOV AX, 5           ; AX = 5
MOV BX, 3           ; BX = 3
ADD AX, BX          ; AX = AX + BX = 8
MOV CL, AL          ; CL = 08h

RET                 ; DOS'a dön

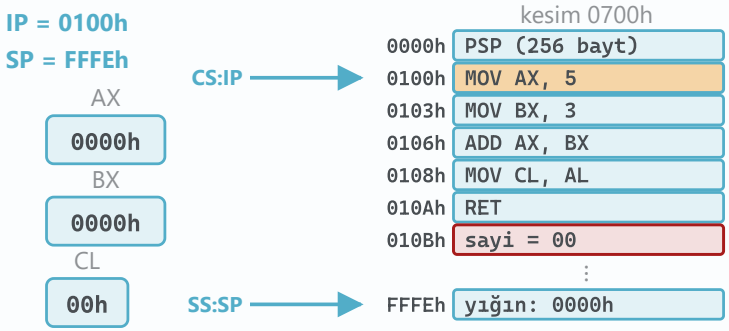
sayi DB 0           ; değişkenler RET'ten sonra
```

- `ORG 100h` : .COM programı kesimin 0100h adresine yüklenir; ilk 256 bayt (PSP) DOS'a aittir.
- Komutlar yukarıdan aşağı, **sırayla** yürütülür.
- `RET` : işletim sistemine geri döner.
- Değişkenler kodun **sonuna** (RET'ten sonra) konur ki CPU onları komut sanıp yürütmesin.
- .COM'da başlangıçta `CS = DS = ES = SS` , `IP = 0100h` , `SP = FFFEh` .

emu8086'da **new** → **COM template** şablonu açar; **emulate** programı öykünücüye yükler, **single step** her komuttan sonra yazmaçların nasıl değiştiğini gösterir.

Kod İzleme: .COM Programı Bellekte

```
▶ ORG 100h          ; kod 0100h'den
2 MOV AX, 5
3 MOV BX, 3
4 ADD AX, BX        ; AX = 8
5 MOV CL, AL        ; CL = 08h
6 RET               ; DOS'a dön
7 sayi DB 0         ; veri
```



Program 0700:0100'e yüklendi. CS = DS = ES = SS = 0700h, **IP = 0100h**, **SP = FFFEh**; yığın tepesinde 0000h var.

Adım 1 / 6

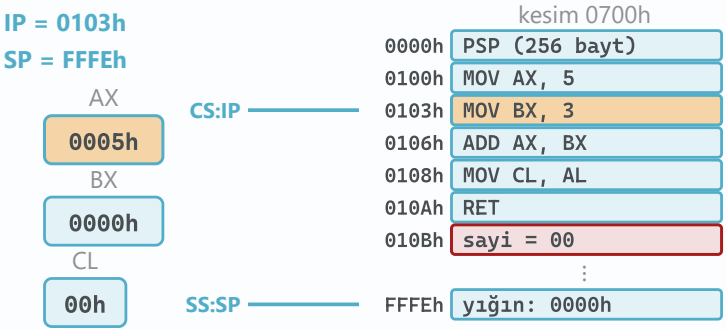
DEĞİŞKENLER

`AX = 0000h` `BX = 0000h` `CL = 00h` `IP = 0100h` `SP = FFFEh`

Komutlar 3, 3, 2, 2 ve 1 bayttır; IP her komutta uzunluğu kadar ilerler. Veriler RET'ten sonra durduğu için komut olarak yürütülmez.

Kod İzleme: .COM Programı Bellekte

```
1 ORG 100h          ; kod 0100h'den
▶ MOV AX, 5
3 MOV BX, 3
4 ADD AX, BX        ; AX = 8
5 MOV CL, AL        ; CL = 08h
6 RET               ; DOS'a dön
7 sayi DB 0         ; veri
```



`MOV AX, 5` (3 bayt): AX = 0005h, IP = **0103h**.

Adım 2 / 6

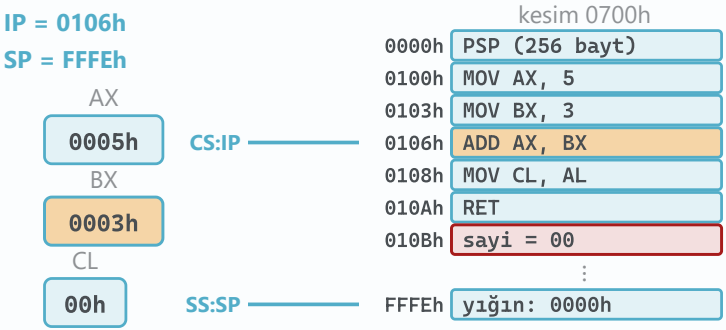
DEĞİŞKENLER

AX = 0005h | BX = 0000h | CL = 00h | IP = 0103h | SP = FFFEh

Komutlar 3, 3, 2, 2 ve 1 bayttır; IP her komutta uzunluğu kadar ilerler. Veriler RET'ten sonra durduğu için komut olarak yürütülmez.

Kod İzleme: .COM Programı Bellekte

```
1 ORG 100h        ; kod 0100h'den
2 MOV AX, 5
▶ MOV BX, 3
4 ADD AX, BX      ; AX = 8
5 MOV CL, AL      ; CL = 08h
6 RET             ; DOS'a dön
7 sayi DB 0       ; veri
```



`MOV BX, 3` (3 bayt): BX = 0003h, IP = **0106h**.

Adım 3 / 6

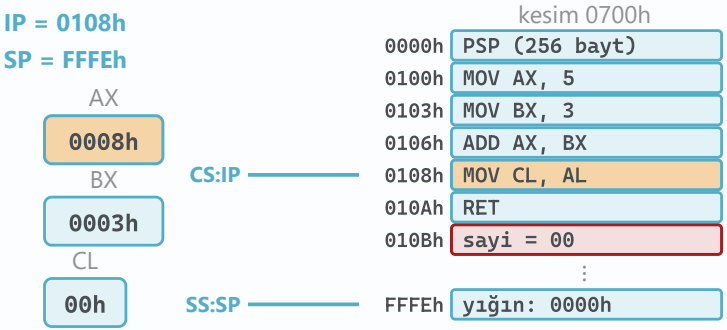
DEĞİŞKENLER

`AX = 0005h` `BX = 0003h` `CL = 00h` `IP = 0106h` `SP = FFFEh`

Komutlar 3, 3, 2, 2 ve 1 bayttır; IP her komutta uzunluğu kadar ilerler. Veriler RET'ten sonra durduğu için komut olarak yürütülmez.

Kod İzleme: .COM Programı Bellekte

```
1 ORG 100h        ; kod 0100h'den
2 MOV AX, 5
3 MOV BX, 3
▶ ADD AX, BX      ; AX = 8
5 MOV CL, AL      ; CL = 08h
6 RET             ; DOS'a dön
7 sayi DB 0       ; veri
```



`ADD AX, BX` (2 bayt): AX = **0008h**, IP = 0108h.

Adım 4 / 6

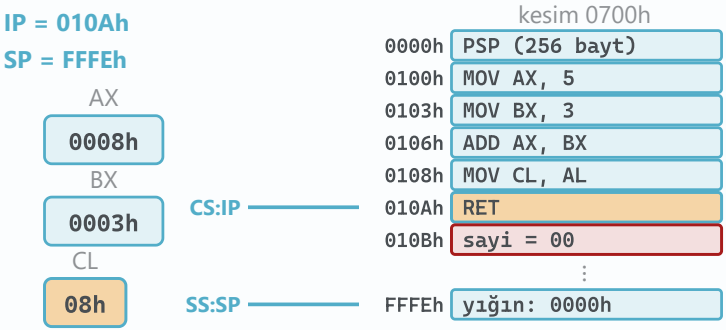
DEĞİŞKENLER

AX = 0008h | BX = 0003h | CL = 00h | IP = 0108h | SP = FFFEh

Komutlar 3, 3, 2, 2 ve 1 bayttır; IP her komutta uzunluğu kadar ilerler. Veriler RET'ten sonra durduğu için komut olarak yürütülmez.

Kod İzleme: .COM Programı Bellekte

```
1 ORG 100h          ; kod 0100h'den
2 MOV AX, 5
3 MOV BX, 3
4 ADD AX, BX        ; AX = 8
▶ MOV CL, AL        ; CL = 08h
6 RET               ; DOS'a dön
7 sayi DB 0         ; veri
```



`MOV CL, AL` (2 bayt): CL = **08h**, IP = 010Ah.

Adım 5 / 6

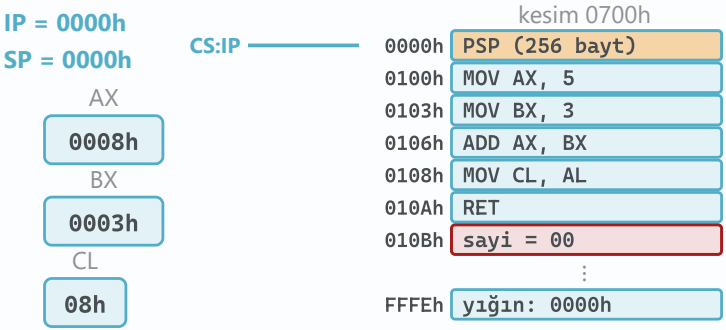
DEĞİŞKENLER

`AX = 0008h` `BX = 0003h` `CL = 08h` `IP = 010Ah` `SP = FFFEh`

Komutlar 3, 3, 2, 2 ve 1 bayttır; IP her komutta uzunluğu kadar ilerler. Veriler RET'ten sonra durduğu için komut olarak yürütülmez.

Kod İzleme: .COM Programı Bellekte

```
1 ORG 100h        ; kod 0100h'den
2 MOV AX, 5
3 MOV BX, 3
4 ADD AX, BX      ; AX = 8
5 MOV CL, AL      ; CL = 08h
▶ RET             ; DOS'a dön
7 sayi DB 0       ; veri
```



`RET` yığından 0000h'yi alır: IP = 0000h, SP = 0000h. PSP:0000'daki `INT 20h` programı bitirir; 010Bh'deki `sayi` hiç yürütülmedi.

Adım 6 / 6

DEĞİŞKENLER

`AX = 0008h` `BX = 0003h` `CL = 08h` `IP = 0000h` `SP = 0000h`

Komutlar 3, 3, 2, 2 ve 1 bayttır; IP her komutta uzunluğu kadar ilerler. Veriler RET'ten sonra durduğu için komut olarak yürütülmez.

İLK PROGRAM

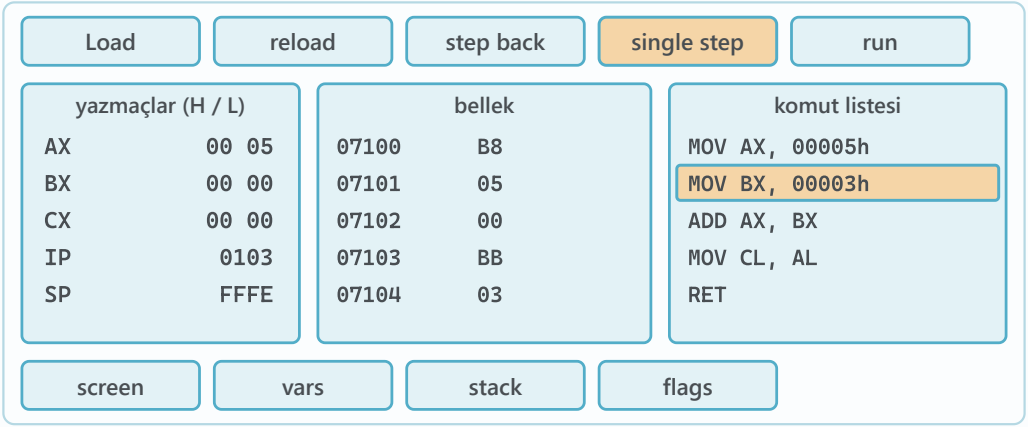
Program Bellekte: Bayt Bayt

Bağıl konum	Baytlar	Kaynak satırı	Not
0100h	`B8 05 00`	`MOV AX, 5`	B8 + yazmaç no (AX = 0); sabit little-endian
0103h	`BB 03 00`	`MOV BX, 3`	B8 + 3 (BX)
0106h	`03 C3`	`ADD AX, BX`	C3 = 11 000 011: yazmaç–yazmaç, AX, BX
0108h	`8A C8`	`MOV CL, AL`	C8 = 11 001 000: CL, AL
010Ah	`C3`	`RET`	tek baytlık komut
010Bh	`00`	`sayi DB 0`	veri; komut değil

- .COM dosyası tam olarak bu 12 bayttır: başlık yoktur, DOS baytları 0100h'den itibaren belleğe kopyalar.
- `ORG 100h` satırının karşılığında bayt yoktur; yalnızca ilk baytın konumunu belirler.
- Aynı komutun birden çok kodlaması olabilir: `ADD AX, BX` için `01 D8` de geçerlidir. MASM 03 C3'ü seçer; ikisi de aynı işi yapar.
- Bellekte komut ile veri arasında hiçbir işaret yoktur: 010Bh'deki 00, IP oraya gelseydi komut olarak çözülürdü.

İLK PROGRAM

emu8086 Öykünücü Penceresi



- single step bir komut yürütür; değişen yazmaçlar ve bellek hemen görünür.
- step back son adımı geri alır; bir hatanın nedenine geri dönmeyi kolaylaştırır.
- Komut listesindeki vurgulu satır **sıradaki** komuttur (CS:IP), az önce yürütülen değil.
- Komut listesi sabitleri onaltılık ve 5 basamakla gösterir (00005h).
- Ek pencereler: ekran çıktısı, değişkenler, yığın ve bayraklar.

Bellek sütunu fiziksel adresleri gösterir: 0700:0100 = 07100h. Çizim pencerenin yalnızca ana bölümlerini gösterir.

İLK PROGRAM

Yazım Kuralları

Sayı tabanları

`25` veya `25d` onluk · `19h` onaltılık · `00011001b` ikili · `31o` sekizli. Hepsi aynı değer: 25.

Harfle başlayan onaltılık

Başına `0` konur: `0FFh`, `0B800h`. `FFh` yazılırsa assembler onu bir *ad* (etiket/değişken) sanır.

Karakterler

`'A'` → ASCII kodu 41h. `MOV CL, 'A'` ile `MOV CL, 41h` aynı makine kodunu üretir.

İşaretli sayılar

`-1` 8 bitte FFh, 16 bitte FFFFh olarak saklanır (ikiye tümleyen).

Büyük-küçük harf

Assembler duyarsızdır: `mov`, `MOV`, `Mov` aynıdır; `var1` ile `VAR1` aynı addır.

Açıklamalar

`;` sonrası açıklamadır. Her satıra “ne” değil “neden” yazmak iyi alışkanlıktır.

İLK PROGRAM

Sık Yapılan Hatalar

Hatalı	Neden?	Doğrusu
`MOV AL, BX`	Boyutlar farklı (8 ↔ 16 bit)	`MOV AL, BL` veya `MOV AX, BX`
`MOV DS, 1234h`	Kesim yazmacına sabit atanamaz	`MOV AX, 1234h` · `MOV DS, AX`
`MOV AL, [AX]`	Adreslemede yalnız BX, SI, DI, BP	`MOV BX, AX` · `MOV AL, [BX]`
`MOV AL, FFh`	FFh bir ad sanılır	`MOV AL, 0FFh`
`MOV IP, 200h`	IP'ye MOV ile yazılamaz	`JMP` ile atla
`MOV AL, 300`	300 > 255, 8 bite sığmaz	`MOV AX, 300`

Bu kurallar MOV komutunun işlenen kısıtlarından gelir.

İLK PROGRAM

Daha Fazla Söz Dizimi Hatası

Hatalı	Neden?	Doğrusu
`MOV CS, AX`	CS yalnız JMP, CALL, RET, INT ile değişir	uzak atlama
`MOV ES, DS`	kesimden kesime MOV yok	`MOV AX, DS` · `MOV ES, AX`
`MOV [BX], [SI]`	iki bellek işleneni	`MOV AL, [SI]` · `MOV [BX], AL`
`INC [BX]`	boyut belirsiz	`INC BYTE PTR [BX]`
`MOV AL, [BX+BP]`	iki taban yazmacı birlikte olmaz	`[BX+SI]` ya da `[BP+DI]`
`ADD 5, AX`	hedef sabit olamaz	`ADD AX, 5`
`PUSH 5`	80186'dan önce yok	`MOV AX, 5` · `PUSH AX`
`SHL AX, 2`	8086'da sayı yalnız 1 ya da CL	`MOV CL, 2` · `SHL AX, CL`

PUSH sabit ve SHL AX, 2 80186 ve sonrasında geçerlidir. MASM 8086 için derlerken hata verir; emu8086 kabul eder, ama bu kod gerçek bir 8086'da çalışmaz.

İLK PROGRAM

Derleme Hatası mı, Mantık Hatası mı?

Derleme (söz dizimi) hatası

Assembler bulur, satır numarasını verir, program hiç oluşmaz.

- `MOV AL, BX` : boyut uyuşmazlığı
- `JMP bsla` : tanımsız etiket
- `MOV AL, 300` : sığmayan sabit

Mantık hatası

Program derlenir ve çalışır, ama sonuç yanlıştır; assembler bunu bilemez.

- RET'i unutmak: veri komut sanılır
- İşaretli sayı için `JA` kullanmak
- ADC yerine ADD: elde kaybolur
- .EXE'de DS'yi yüklememek

Mantık hatası tek adım çalıştırmayla bulunur: her komuttan sonra beklenen yazmaç ve bayrak değeri öykünücüdeki değerle karşılaştırılır. Değerlerin ayrıldığı ilk komut, hatanın bulunduğu yerdir.

Laboratuvar: İlk Programlar

.COM programı

KOD · **12 bayt**

```
1 ORG 100h          ; kod 0100h…
2 MOV AX, 5
3 MOV BX, 3
4 ADD AX, BX        ; AX = 8
5 MOV CL, AL        ; CL = 08h
6 RET               ; DOS'a dön
7 sayi DB 0         ; veri
```

YAZMAÇLAR

AX	0000	AH	00	AL	00
BX	0000	BH	00	BL	00
CX	000C	CH	00	CL	0C
DX	0000	DH	00	DL	00

SI	0000	DI	0000	BP	0000
SP	FFFE	CS	0700	IP	0100
DS	0700	ES	0700	SS	0700

BAYRAKLAR

O	D	I	T	S	Z	A	P	C
0	0	1	0	0	0	0	0	0

CS:IP = 0700:0100 · adım 0

BELLEK **DS = 0700h**

```
0108 8A C8 C3 00 00 00 00 00
0110 00 00 00 00 00 00 00 00
0118 00 00 00 00 00 00 00 00
0120 00 00 00 00 00 00 00 00
0128 00 00 00 00 00 00 00 00
```

YIĞIN **SS:SP**

```
FFF6 0000
FFF8 0000
FFFA 0000
FFFC 0000
FFFE 0000 ◂ SP
```

ÇIKTI

Program yüklendi: 12 bayt, CS:IP = 0700:0100h. emu8086'daki gibi CX başlangıçta program boyutunu (000Ch) gösterir. Adım düğmesiyle komutlar tek tek yürütülür; vurgulu satır sıradaki komuttur.

ÖZET

Özet

Assembly
Makine koduyla birebir eşleşen düşük düzeyli dil; küçük ve hızlı kod, donanıma doğrudan erişim, ama taşınabilir değil.

Satır yapısı
`[etiket:] komut hedef, kaynak ; açıklama`. Direktifler (ORG, DB, EQU) makine kodu üretmez.

İşlenenler
Yazmaç en hızlısıdır; anlık değer yalnız kaynak olur; bellek işleneni yol döngüsü ister. MUL, LOOP, CBW örtük işlenen kullanır.

İlk program
.COM programı `ORG 100h` ile 0100h'den başlar; veriler RET'ten sonra durur.

IP
CS:IP sıradaki komutu gösterir; komut uzunluğu kadar ilerler.

Bayraklar
MOV, NOT, XCHG bayrakları değiştirmez; INC ve DEC CF'ye dokunmaz; ADD hepsini yeniden yazar.

ÖZET

Alıştırmalar

1. AX = 0ABCDh iken `MOV AH, 0` sonrası AX, AL ve BX değerleri nedir?
2. AX = 1234h, BL = 03h iken `MUL BL` sonrası AX kaçtır? AH'deki 12h'ye ne olur?
3. 8 MHz'lik bir 8086'da 2000 kez dönen bir döngüde sayaç `INC sayac` (21 tur) yerine `INC CX` (2 tur) ile artırılırsa kaç saat turu ve kaç milisaniye kazanılır?
4. AL = 80h iken `ADD AL, 80h` sonrası AL, CF, ZF, SF, OF?
5. Program 0100h'de başlar: 3 + 2 + 1 + 4 baytlık dört komut yürütülünce IP ne olur?
6. `MOV CX, AL` ve `MOV ES, 0B800h` neden hatalıdır? Düzeltin.

▶ **Cevaplar**

ÖZET

Kaynaklar

- emu8086 belgeleri, *8086 assembler tutorial for beginners* (bölüm 1: numbering systems, bölüm 2: memory access).
- Intel, *8086 16-Bit HMOS Microprocessor* veri sayfası.
- Barry B. Brey, *The Intel Microprocessors*, 8. baskı, Pearson.
- Muhammad Ali Mazidi, Janice G. Mazidi, *The 80x86 IBM PC and Compatible Computers*, Pearson.

Son

Bölüm 3.2: Assembly

Sonraki bölüm: Bölüm 4.1: Bellek Erişimi · Terim sözlüğü