

Bölüm 1: Programlama Dilleri

Mikroişlemciler

Yazılım ve dil seviyeleri

Makine dili

Örnek komut biçimi

Assembly dilleri

Farklı platformlarda "Merhaba"

Derlenen diller

Yorumlanan diller

Aynı program 14 dilde + 8086

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

Öğrenme Çıktıları

Dil seviyeleri

Makine dili, assembly dili ve yüksek seviyeli diller arasındaki farkı açıklamak.

Makine kodu

Bir komut biçimine göre ikili komutları kodlamak ve çözmek.

Assembly

Etiket, komut, işlenen, direktif kavramlarını tanımak; sistem çağrısı yapan kısa programları okumak.

Platform bağımlılığı

Aynı programın x86, 68000, ARM, PowerPC ve farklı işletim sistemlerinde neden farklı yazıldığını açıklamak.

Derleme ve yorumlama

Derleyici, bağlayıcı, yorumlayıcı ve JIT kavramlarını karşılaştırmak.

Diller arası okuma

Aynı algoritmayı farklı dillerde ve 8086 assembly ile izlemek.

Yazılım

- Bilgisayar donanımını **düzgün, koordineli** ve istenen şekilde çalıştırır.
- **İşletim sistemi**
 - bilgisayarın temel yazılımıdır,
 - donanımı yönetir, diğer yazılımların çalışmasını denetler.
- **Programlama dilleri**
 - yazılım geliştirmek için icat edilmiştir,
 - farklı alanlara yönelik çok sayıda dil vardır.
- Programlama dillerinin evrimi, bilgisayar teknolojisindeki önemli gelişmeleri yansıtır.

Kullanıcı

Uygulamalar

tarayıcı, derleyici, oyun

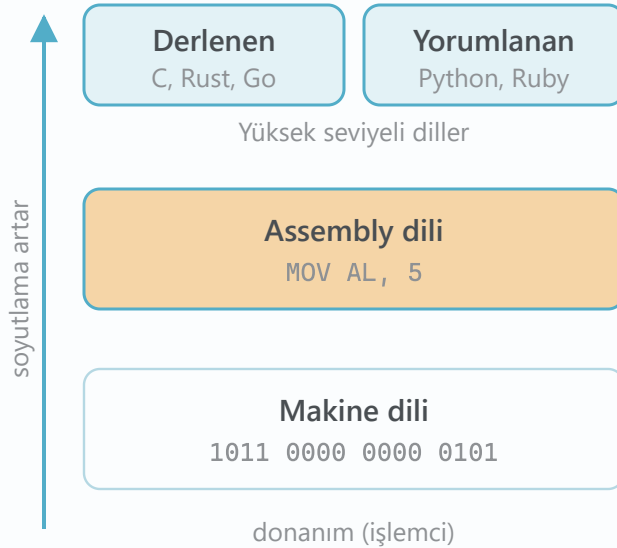
İşletim sistemi

sistem çağrıları, sürücüler

Donanım

CPU, bellek, G/Ç

Programlama Dilleri: Seviyeler



1. **Makine dilleri:** işlemcinin doğrudan çalıştırdığı 0 ve 1'ler.
2. **Assembly dilleri:** makine komutlarının sembolik (anımsatıcı) yazımı.
3. **Yüksek seviyeli diller**
 - **derlenen:** önce makine diline çevrilir, sonra çalışır;
 - **yorumlanan:** çalışma anında satır satır işlenir.

Seviye yükseldikçe **taşınabilirlik** ve **okunabilirlik** artar, donanım üzerindeki **doğrudan denetim** azalır.

Aynı İşlem Üç Seviyede: $x = a + b$

Yüksek seviye (C)

$x = a + b;$

Değişkenlerin nerede durduğu, hangi yazmacın kullanılacağı derleyiciye bırakılır.

Aynı İşlem Üç Seviyede: $x = a + b$

Yüksek seviye (C)

$x = a + b;$

Değişkenlerin nerede durduğu, hangi yazmacın kullanılacağı derleyiciye bırakılır.

Assembly dili (8086)

```
MOV AX, [a]      ; a'yı AX yazmacına yükle  
ADD AX, [b]      ; AX = AX + b  
MOV [x], AX      ; sonucu x'e yaz
```

Her satır tek bir makine komutuna karşılık gelir.

Aynı İşlem Üç Seviyede: $x = a + b$

Yüksek seviye (C)

```
x = a + b;
```

Değişkenlerin nerede durduğu, hangi yazmacın kullanılacağı derleyiciye bırakılır.

Assembly dili (8086)

```
MOV AX, [a]      ; a'yı AX yazmacına yükle  
ADD AX, [b]      ; AX = AX + b  
MOV [x], AX      ; sonucu x'e yaz
```

Her satır tek bir makine komutuna karşılık gelir.

Makine dili (a = 0200h, b = 0202h, x = 0204h)

```
A1 00 02        ; MOV AX, [0200h]  
03 06 02 02     ; ADD AX, [0202h]  
A3 04 02        ; MOV [0204h], AX
```

Adresler bellekte düşük bayt önce (little-endian) saklanır: 0200h → 00 02 .

Kod İzleme: $x = a + b$ Bellekte Nasıl Çalışır?

```
1 org 100h
```

```
► mov ax, [0200h] ; A1 00 02
```

```
3 add ax, [0202h] ; 03 06 02 02
```

```
4 mov [0204h], ax ; A3 04 02
```

```
5 ret ; C3
```

kod (0100h–010Ah)

A1	00	02	03	06	02	02	A3	04	02	C3
----	----	----	----	----	----	----	----	----	----	----

IP = 0100h

veri

15	00	27	00	??	??
----	----	----	----	----	----

0200h

0201h

0202h

0203h

0204h

0205h

a

b

x

GETİR

AX

????

IP = 0100h. **Getir:** A1 00 02 okunur. A1 = "AX'e bellekten sözcük yükle", adres baytları 00 02 → **0200h** (little-endian).

Adım 1 / 7

DEĞİŞKENLER

IP = 0100h

AX = ????

x = ????

Varsayım: a = 0015h (21), b = 0027h (39); veri kesimi kodla aynı (COM programı).

Kod İzleme: $x = a + b$ Bellekte Nasıl Çalışır?

```
1 org 100h
```

```
► mov ax, [0200h] ; A1 00 02
```

```
3 add ax, [0202h] ; 03 06 02 02
```

```
4 mov [0204h], ax ; A3 04 02
```

```
5 ret ; C3
```

kod (0100h–010Ah)

A1	00	02	03	06	02	02	A3	04	02	C3
----	----	----	----	----	----	----	----	----	----	----

IP = 0103h

veri

15	00	27	00	??	??
----	----	----	----	----	----

0200h

0201h

0202h

0203h

0204h

0205h

a

b

x

YÜRÜT

AX

0015h

Yürüt: 0200h ve 0201h'deki baytlar 15 00 → AX = **0015h** (a = 21). IP = 0103h.

Adım 2 / 7

DEĞİŞKENLER

IP = 0103h

AX = 0015h

x = ????

Varsayım: a = 0015h (21), b = 0027h (39); veri kesimi kodla aynı (COM programı).

Kod İzleme: $x = a + b$ Bellekte Nasıl Çalışır?

```

1 org 100h
2 mov ax, [0200h] ; A1 00 02
3 ► add ax, [0202h] ; 03 06 02 02
4 mov [0204h], ax ; A3 04 02
5 ret ; C3

```

kod (0100h–010Ah)

A1	00	02	03	06	02	02	A3	04	02	C3
----	----	----	----	----	----	----	----	----	----	----

IP = 0103h

GETİR

AX

veri

15	00	27	00	??	??
----	----	----	----	----	----

0015h

0200h

0201h

0202h

0203h

0204h

0205h

a

b

x

Getir: **03 06 02 02** (4 bayt). 03 = ADD r16, r/m16; ModR/M
06 = "AX, doğrudan adres"; adres 0202h.

Adım 3 / 7

DEĞİŞKENLER

IP = 0103h

AX = 0015h

x = ????

Varsayım: a = 0015h (21), b = 0027h (39); veri kesimi kodla aynı (COM programı).

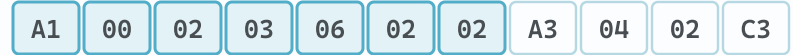
Kod İzleme: $x = a + b$ Bellekte Nasıl Çalışır?

```

1 org 100h
2 mov ax, [0200h] ; A1 00 02
3 ► add ax, [0202h] ; 03 06 02 02
4 mov [0204h], ax ; A3 04 02
5 ret ; C3

```

kod (0100h–010Ah)



IP = 0107h

YÜRÜT

AX

veri



0200h

0201h

0202h

0203h

0204h

0205h

a

b

x

003Ch

Yürüt: b = 0027h (39) okunur, AX = 15h + 27h = **003Ch** (60).
IP = 0107h.

Adım 4 / 7

DEĞİŞKENLER

IP = 0107h

AX = 003Ch

x = ????

Varsayım: a = 0015h (21), b = 0027h (39); veri kesimi kodla aynı (COM programı).

Kod İzleme: $x = a + b$ Bellekte Nasıl Çalışır?

```

1 org 100h
2 mov ax, [0200h] ; A1 00 02
3 add ax, [0202h] ; 03 06 02 02
4 ► mov [0204h], ax ; A3 04 02
5 ret ; C3

```

kod (0100h–010Ah)

A1	00	02	03	06	02	02	A3	04	02	C3
----	----	----	----	----	----	----	----	----	----	----

IP = 0107h

GETİR

AX

veri

15	00	27	00	??	??
----	----	----	----	----	----

003Ch

0200h

0201h

0202h

0203h

0204h

0205h

a

b

x

Getir: **A3 04 02**: A3 = "AX'i belleğe yaz", hedef adres 0204h.

Adım 5 / 7

DEĞİŞKENLER

IP = 0107h

AX = 003Ch

x = ????

Varsayım: a = 0015h (21), b = 0027h (39); veri kesimi kodla aynı (COM programı).

Kod İzleme: $x = a + b$ Bellekte Nasıl Çalışır?

```

1 org 100h
2 mov ax, [0200h] ; A1 00 02
3 add ax, [0202h] ; 03 06 02 02
▶ 4 mov [0204h], ax ; A3 04 02
5 ret ; C3

```

kod (0100h–010Ah)

A1	00	02	03	06	02	02	A3	04	02	C3
----	----	----	----	----	----	----	----	----	----	----

IP = 010Ah

YÜRÜT

AX

veri

15	00	27	00	3C	00
----	----	----	----	----	----

0200h

0201h

0202h

0203h

0204h

0205h

a

b

x

003Ch

Yürüt: AX belleğe düşük bayt önce yazılır: 0204h ← **3C**, 0205h ← **00**. $x = 60$.

Adım 6 / 7

DEĞİŞKENLER

IP = 010Ah

AX = 003Ch

x = 003Ch

Varsayım: $a = 0015h$ (21), $b = 0027h$ (39); veri kesimi kodla aynı (COM programı).

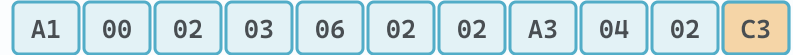
Kod İzleme: $x = a + b$ Bellekte Nasıl Çalışır?

```

1 org 100h
2 mov ax, [0200h] ; A1 00 02
3 add ax, [0202h] ; 03 06 02 02
4 mov [0204h], ax ; A3 04 02
▶ ret ; C3

```

kod (0100h–010Ah)



IP = 010Ah

GETİR

AX

veri



003Ch

0200h

0201h

0202h

0203h

0204h

0205h

a

b

x

C3 = RET. C'deki tek satır $x = a + b;$ burada 3 komut, 10 bayt makine kodu oldu.

Adım 7 / 7

DEĞİŞKENLER

IP = 010Ah

AX = 003Ch

x = 003Ch

Varsayım: a = 0015h (21), b = 0027h (39); veri kesimi kodla aynı (COM programı).

Makine Dilleri

- 0 ve 1'lerden oluşan **en düşük seviyeli** dildir; işlemci yalnızca bunu anlar.
- Tamamen **işlemciye ve donanıma bağımlıdır**: 8086 için yazılan makine kodu ARM'da çalışmaz.
- İlk genel amaçlı elektronik bilgisayarlardan **ENIAC** (1945), kablolar ve anahtarlarla doğrudan donanım üzerinde programlanıyordu.
- Günümüzde
 - her işlemci ailesinin kendine özgü bir makine dili (komut kümesi) vardır,
 - okuması ve yazması pratik olmadığı için doğrudan kullanılmaz.
- **Assembly dilleri** makine diline bağımlılığı azaltır ve daha insan dostu bir programlama sağlar.

8086 · MOV AL, 5

1011	0	000	00000101
------	---	-----	----------

 = B0 05
MOV sabit w=0 AL sabit = 5

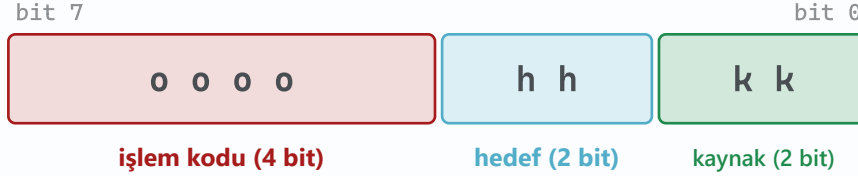
ARM · MOV r0, #5

1110 0011 1010 0000 0000 0000 0000 0101

 = E3A00005 (32 bit)

8086'da B0 05 baytları "AL'ye 5 yükle" demektir. Aynı iş ARM'da 32 bitlik tamamen farklı bir sözcükle kodlanır.

Örnek Bir Makine Dili Tasarımı



İşlem kodu	Anlamı
0000	HALT: programı durdur
0001	ADD: toplama
0010	SUB: çıkarma
0011	AND: mantıksal VE
0100	OR: mantıksal VEYA

- Komutlar **8 bitliktir**.
- 4 yazmaç: R0 = 00, R1 = 01, R2 = 10, R3 = 11.
- Anlam: **hedef** ← **hedef işlem kaynak**.
- 4 bitlik işlem kodu en çok $2^4 = 16$ işlem tanımlayabilir; burada 5'i kullanılıyor.

Örnek: 0001 10 11 → ADD R2, R3 → R2 ← R2 + R3

MAKİNE DİLİ

Örnek Program: Alanları Ayırmak

Makine kodu	İşlem	Hedef	Kaynak	Anlamı
0001 0001	ADD	R0	R1	$R0 \leftarrow R0 + R1$
0011 1011	AND	R2	R3	$R2 \leftarrow R2 \text{ AND } R3$
0100 0110	OR	R1	R2	$R1 \leftarrow R1 \text{ OR } R2$
0000 0000	HALT	—	—	programı durdur

Tanımlanan biçimde ilk 2 bitlik alan **hedef** olduğu için sonuç **ilk** yazmaca (R0, R2, R1) yazılır.

Onaltılık karşılıkları: 11h, 3Bh, 46h, 00h.

Kod İzleme: Örnek Makine Programı

```
1 0001 0001 ; ADD R0, R1 → R0 ← R0 + R1
2 0011 1011 ; AND R2, R3 → R2 ← R2 AND R3
3 0100 0110 ; OR R1, R2 → R1 ← R1 OR R2
4 0000 0000 ; HALT
```

Başlangıç değerleri (örnek). PC = 0: ilk komut getirilecek.

Adım 1 / 5

DEĞİŞKENLER

PC = 0

R0 = 0000 0101 (5)

R1 = 0000 0011 (3)

R2 = 0000 1100 (12)

R3 = 0000 1010 (10)

Kod İzleme: Örnek Makine Programı

```
▶ 0001 0001 ; ADD R0, R1 → R0 ← R0 + R1
2 0011 1011 ; AND R2, R3 → R2 ← R2 AND R3
3 0100 0110 ; OR R1, R2 → R1 ← R1 OR R2
4 0000 0000 ; HALT
```

ADD R0, R1: $5 + 3 = 8 \rightarrow R0$. Kaynak (R1) değişmez.

Adım 2 / 5

DEĞİŞKENLER

PC = 1

R0 = 0000 1000 (8)

R1 = 0000 0011 (3)

R2 = 0000 1100 (12)

R3 = 0000 1010 (10)

Kod İzleme: Örnek Makine Programı

```
1 0001 0001 ; ADD R0, R1 → R0 ← R0 + R1
▶ 0011 1011 ; AND R2, R3 → R2 ← R2 AND R3
3 0100 0110 ; OR R1, R2 → R1 ← R1 OR R2
4 0000 0000 ; HALT
```

AND R2, R3: $1100 \text{ AND } 1010 = \mathbf{1000} \rightarrow R2 = 8$. Yalnızca iki tarafta da 1 olan bit kalır.

Adım 3 / 5

DEĞİŞKENLER

PC = 2

R0 = 0000 1000 (8)

R1 = 0000 0011 (3)

R2 = 0000 1000 (8)

R3 = 0000 1010 (10)

Kod İzleme: Örnek Makine Programı

```
1 0001 0001 ; ADD R0, R1 → R0 ← R0 + R1
2 0011 1011 ; AND R2, R3 → R2 ← R2 AND R3
▶ 0100 0110 ; OR R1, R2 → R1 ← R1 OR R2
4 0000 0000 ; HALT
```

OR R1, R2: 0011 OR 1000 = **1011** → R1 = 11.

Adım 4 / 5

DEĞİŞKENLER

PC = 3

R0 = 0000 1000 (8)

R1 = 0000 1011 (11)

R2 = 0000 1000 (8)

R3 = 0000 1010 (10)

Kod İzleme: Örnek Makine Programı

```
1 0001 0001 ; ADD R0, R1 → R0 ← R0 + R1
2 0011 1011 ; AND R2, R3 → R2 ← R2 AND R3
3 0100 0110 ; OR R1, R2 → R1 ← R1 OR R2
▶ 0000 0000 ; HALT
```

HALT: işlemci durur. Sonuç: R0 = 8, R1 = 11, R2 = 8, R3 = 10.

Adım 5 / 5

DEĞİŞKENLER

PC = 3

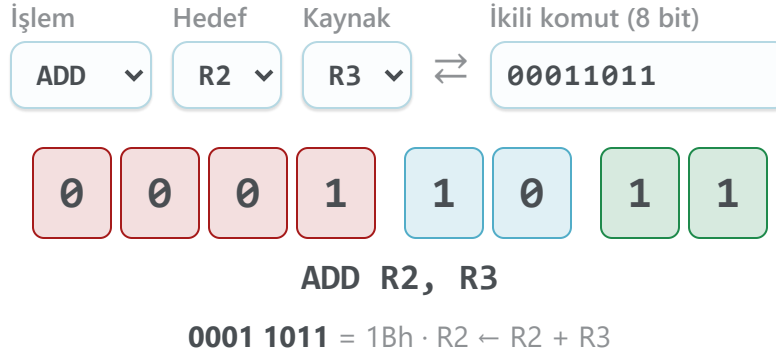
R0 = 0000 1000 (8)

R1 = 0000 1011 (11)

R2 = 0000 1000 (8)

R3 = 0000 1010 (10)

Kodlayıcı ve Çözücü



Üstteki seçimleri değiştir (kodlama) ya da sağdaki kutuya 8 bit yaz (çözme). 0101 –

1111 işlem kodları bu tasarımda tanımsızdır.

Makine Dili: Artılar ve Eksiler

Artılar

- + Çeviri gerekmez; işlemci doğrudan çalıştırır.
- + Donanım üzerinde tam denetim.
- + En küçük ve (doğru yazılırsa) en hızlı kod.

Eksiler

- Okuması, yazması ve hata ayıklaması çok zor.
- Adresler elle hesaplanır; araya bir komut eklemek tüm atlama adreslerini bozar.
- Taşınabilir değil: her işlemci için baştan yazılır.
- Tek bir bit hatası programı tamamen değiştirir.



Bu sorunları çözmek için önce **assembly dilleri** (1950'ler), ardından **FORTTRAN** (1957) gibi yüksek seviyeli diller geliştirildi.

Assembly Dilleri

- Her makine komutu, akılda kalır bir **anımsatıcıyla** (mnemonic) yazılır: `10110000 00000101` yerine `MOV AL, 5`.
- Farklı işlemci mimarilerine ve komut kümelerine göre tasarlanır: **Intel x86, Motorola 68000, ARM, PIC, AVR** için ayrı assembly dilleri vardır.
- Mikrodenetleyicilere özel assembly dilleri **gömülü sistemlerde** ve elektronik uygulamalarda sıkça kullanılır.
- Bir işlemci ailesi içinde uyumu sağlar ve donanım özelliklerinden en iyi şekilde yararlanmayı amaçlar.
- Kaynak kodu makine diline çeviren programa **assembler** denir.



Nerede kullanılır?

önyükleyici, aygıt sürücüleri, kesme yordamları, işletim sistemi çekirdeği

Neden öğrenilir?

işlemcinin gerçekte nasıl çalıştığını anlamak, hata ayıklama, tersine mühendislik

Performans

derleyicinin üretemediği özel komutlar (SIMD, kript) ve kritik döngüler

ASSEMBLY DİLLERİ

Kaynak Koddan Çalışan Programa



Assembler'ın **liste** (listing) çıktısı, her satırın adresini ve makine kodunu yan yana gösterir:

Adres	Makine kodu	Kaynak
0100	B4 09	mov ah, 09h
0102	BA 0C 01	mov dx, offset mesaj
0105	CD 21	int 21h
0107	B8 00 4C	mov ax, 4C00h
010A	CD 21	int 21h
010C	4D 65 72 ...	mesaj db 'Merhaba!\$'

emu8086 bu adımları tek tuşla yapar ve COM/EXE dosyasını doğrudan üretir.

Kod İzleme: Assembler'ın İki Geçişi

► **org 100h**

```
2 mov ah, 09h
3 mov dx, offset mesaj
4 int 21h
5 mov ax, 4C00h
6 int 21h
7 mesaj db 'Merhaba!$'
```

1. GEÇİŞ

konum sayacı (LC)

0100h

üretilen baytlar

(1. geçişte bayt üretilmez)

sembol tablosu

mesaj

?

satır 1

1. geçiş yalnız adres sayar. **org 100h** konum sayacını (LC) **0100h** yapar; direktif bayt üretmez.

Adım 1 / 9

DEĞİŞKENLER

geçiş = 1

LC = 0100h

mesaj = ?

Kod İzleme: Assembler'ın İki Geçişi

```
1 org 100h
▶ mov ah, 09h
3 mov dx, offset mesaj
4 int 21h
5 mov ax, 4C00h
6 int 21h
7 mesaj db 'Merhaba!$'
```

1. GEÇİŞ

konum sayacı (LC)

0102h

üretilen baytlar

(1. geçişte bayt üretilmez)

sembol tablosu

mesaj

?

satır 2

`mov ah, 09h` 2 bayt tutar → LC = **0102h**. Baytların kendisi henüz yazılmaz.

Adım 2 / 9

DEĞİŞKENLER

geçiş = 1

LC = **0102h**

mesaj = ?

Kod İzleme: Assembler'ın İki Geçişi

```
1 org 100h
2 mov ah, 09h
3 ► mov dx, offset mesaj
4 int 21h
5 mov ax, 4C00h
6 int 21h
7 mesaj db 'Merhaba!$'
```

1. GEÇİŞ

konum sayacı (LC)

0105h

üretilen baytlar

(1. geçişte bayt üretilmez)

sembol tablosu

mesaj**?**

satır 3

offset mesaj ileri başvurudur: mesaj henüz tanımlanmadı ama komutun 3 bayt olduğu bilinir → LC = **0105h**.

Adım 3 / 9

DEĞİŞKENLER

geçiş = 1

LC = **0105h**

mesaj = ?

Kod İzleme: Assembler'ın İki Geçişi

```
1 org 100h
2 mov ah, 09h
3 mov dx, offset mesaj
▶ int 21h
▶ mov ax, 4C00h
▶ int 21h
7 mesaj db 'Merhaba!$'
```

1. GEÇİŞ

konum sayacı (LC)

010Ch

üretilen baytlar

(1. geçişte bayt üretilmez)

sembol tablosu

mesaj

?

satır 4,5,6

int 21h (2), mov ax, 4C00h (3), int 21h (2): LC = **010Ch**.

Adım 4 / 9

DEĞİŞKENLER

geçiş = 1

LC = **010Ch**

mesaj = ?

Kod İzleme: Assembler'ın İki Geçişi

```
1 org 100h
2 mov ah, 09h
3 mov dx, offset mesaj
4 int 21h
5 mov ax, 4C00h
6 int 21h
▶ mesaj db 'Merhaba!$'
```

1. GEÇİŞ

konum sayacı (LC)

0115h

üretilen baytlar

(1. geçişte bayt üretilmez)

sembol tablosu

mesaj 010Ch

satır 7

Etiket **mesaj** şu anki LC ile **sembol tablosuna** girer: mesaj = 010Ch. 9 baytlık dizgi → LC = 0115h.

Adım 5 / 9

DEĞİŞKENLER

geçiş = 1

LC = 0115h

mesaj = 010Ch

Kod İzleme: Assembler'ın İki Geçişi

```

1 org 100h
2 ► mov ah, 09h
3 mov dx, offset mesaj
4 int 21h
5 mov ax, 4C00h
6 int 21h
7 mesaj db 'Merhaba!$'
    
```

2. GEÇİŞ

konum sayacı (LC)

0100h

üretilen baytlar

B4 09

sembol tablosu

mesaj 010Ch

satır 2

2. **geçiş** baştan başlar ve baytları üretir: `mov ah, 09h` → **B4 09**.

Adım 6 / 9

DEĞİŞKENLER

geçiş = 2

LC = 0100h

mesaj = 010Ch

Kod İzleme: Assembler'ın İki Geçişi

```
1 org 100h
2 mov ah, 09h
3 ► mov dx, offset mesaj
4 int 21h
5 mov ax, 4C00h
6 int 21h
7 mesaj db 'Merhaba!$'
```

2. GEÇİŞ

konum sayacı (LC)

0102h

üretilen baytlar

B4 09 BA 0C 01

sembol tablosu

mesaj 010Ch

satır 3

offset mesaj artık tablodan bulunur: 010Ch → **BA 0C 01**
(adres little-endian).

Adım 7 / 9

DEĞİŞKENLER

geçiş = 2

LC = 0102h

mesaj = 010Ch

Kod İzleme: Assembler'ın İki Geçişi

```
1 org 100h
2 mov ah, 09h
3 mov dx, offset mesaj
▶ int 21h
▶ mov ax, 4C00h
▶ int 21h
7 mesaj db 'Merhaba!$'
```

2. GEÇİŞ

konum sayacı (LC)

0105h

sembol tablosu

mesaj 010Ch

satır 4,5,6

üretilen baytlar

B4	09	BA	0C	01	CD	21	B8	00	4C	CD	21
----	----	----	----	----	----	----	----	----	----	----	----

Kalan komutlar: **CD 21, B8 00 4C, CD 21.**

Adım 8 / 9

DEĞİŞKENLER

geçiş = 2

LC = 0105h

mesaj = 010Ch

Kod İzleme: Assembler'ın İki Geçişi

```
1 org 100h
2 mov ah, 09h
3 mov dx, offset mesaj
4 int 21h
5 mov ax, 4C00h
6 int 21h
▶ mesaj db 'Merhaba!$'
```

2. GEÇİŞ

konum sayacı (LC)

010Ch

sembol tablosu

mesaj 010Ch

satır 7

üretilen baytlar

B4	09	BA	0C	01	CD	21	B8	00	4C	CD	21	4D	65	72	68
61	62	61	21	24											

Dizginin ASCII kodları yazılır: 4D 65 72 ... 24 ('\$'). Çıktı 21 baytlık COM dosyası; çalıştırılınca ekrana **Merhaba!** yazar.

Adım 9 / 9

DEĞİŞKENLER

geçiş = 2

LC = **010Ch**mesaj = **010Ch**

ASSEMBLY DİLLERİ

Bir Assembly Satırının Yapısı

dongu: **add** **ax**, **cx** ; toplama ekle

etiket

animsatıcı

işlenenler

yorum (assembler yok sayar)

Öge	Görevi	Örnek
Etiket	bir adrese isim verir; atlama ve veri hedefi	<code>dongu:</code> , <code>mesaj</code>
Animsatıcı	makine komutunun adı	<code>MOV</code> , <code>ADD</code> , <code>JMP</code> , <code>INT</code>
İşlenenler	yazmaç, sabit ya da bellek; 8086'da önce hedef , sonra kaynak	<code>AX</code> , <code>5</code> · <code>[BX]</code> , <code>AL</code>
Direktif (directive)	makine kodu üretmez, assembler'a bilgi verir	<code>ORG 100h</code> , <code>DB</code> , <code>SECTION</code> , <code>EQU</code>
Yorum	<code>;</code> ile başlar (ARM GNU'da <code>@</code> , PPC'de <code>#</code>)	<code>;</code> açıklama

Assembler'lar ve Sözdizimleri

Assembler	Sözdizimi · kullanım
MASM	Intel · Microsoft, Windows
TASM	Intel · Borland, DOS
NASM	Intel · açık kaynak, çok platform
GAS (as)	AT&T · GNU/Linux, gcc
emu8086	Intel (MASM benzeri) · 8086 benzetimcisi, bu ders

Aynı komut, iki yazım

```
; Intel:  hedef, kaynak  
mov eax, 5  
add eax, [ebx+4]
```

```
# AT&T:  kaynak, hedef  
movl $5, %eax  
addl 4(%ebx), %eax
```

AT&T'de yazmaçlar `%`, sabitler `$` ile yazılır; boyut komut sonesinde (`l` = 32 bit) belirtilir.

x86 Assembly: Linux (32 bit, NASM)

```
1 section .data
▶ hello db 'Merhaba, Linux!', 10
▶ len equ $ - hello ; = 16 bayt
4 section .text
5 global _start
6 _start:
7 mov eax, 4 ; sys_write
8 mov ebx, 1 ; stdout
9 mov ecx, hello ; dizgenin adresi
10 mov edx, len ; bayt sayısı
11 int 0x80 ; çekirdeği çağır
12 mov eax, 1 ; sys_exit
13 xor ebx, ebx ; çıkış kodu 0
14 int 0x80
```

Veri bölümü: dizgi ve sonuna satır sonu (10).

`len` , `$ - hello` ile **16** olarak hesaplanır.

Adım 1 / 9

DEĞİŞKENLER

EAX = ??????????

EBX = ??????????

ECX = ??????????

EDX = ??????????

ÇIKTI

"Merhaba, Dünya!" metninde ü harfi UTF-8'de 2 bayt olduğundan metin 16 bayttır; `equ $ - hello` uzunluğu assembler'a hesaplatır.

x86 Assembly: Linux (32 bit, NASM)

```
1 section .data
2 hello db 'Merhaba, Linux!', 10
3 len equ $ - hello ; = 16 bayt
4 section .text
5 global _start
6 _start:
7 ► mov eax, 4 ; sys_write
8 mov ebx, 1 ; stdout
9 mov ecx, hello ; dizgenin adresi
10 mov edx, len ; bayt sayısı
11 int 0x80 ; çekirdeği çağır
12 mov eax, 1 ; sys_exit
13 xor ebx, ebx ; çıkış kodu 0
14 int 0x80
```

Program `_start` 'tan başlar. EAX = 4:
sys_write sistem çağırısı numarası.

Adım 2 / 9

DEĞİŞKENLER

EAX = 00000004

EBX = ????????

ECX = ????????

EDX = ????????

ÇIKTI

"Merhaba, Dünya!" metninde ü harfi UTF-8'de 2 bayt olduğundan metin 16 bayttır; `equ $`
`- hello` uzunluğu assembler'a hesaplatır.

x86 Assembly: Linux (32 bit, NASM)

```
1 section .data
2 hello db 'Merhaba, Linux!', 10
3 len equ $ - hello ; = 16 bayt
4 section .text
5 global _start
6 _start:
7 mov eax, 4 ; sys_write
8 ► mov ebx, 1 ; stdout
9 mov ecx, hello ; dizgenin adresi
10 mov edx, len ; bayt sayısı
11 int 0x80 ; çekirdeği çağır
12 mov eax, 1 ; sys_exit
13 xor ebx, ebx ; çıkış kodu 0
14 int 0x80
```

EBX = 1: yazılacak dosya tanıtıcısı **stdout** (ekran).

Adım 3 / 9

DEĞİŞKENLER

EAX = 00000004

EBX = 00000001

ECX = ?????????

EDX = ?????????

ÇIKTI

"Merhaba, Dünya!" metninde ü harfi UTF-8'de 2 bayt olduğundan metin 16 bayttır; `equ $ - hello` uzunluğu assembler'a hesaplatır.

x86 Assembly: Linux (32 bit, NASM)

```
1 section .data
2 hello db 'Merhaba, Linux!', 10
3 len equ $ - hello ; = 16 bayt
4 section .text
5 global _start
6 _start:
7 mov eax, 4 ; sys_write
8 mov ebx, 1 ; stdout
9 ► mov ecx, hello ; dizginin adresi
10 mov edx, len ; bayt sayısı
11 int 0x80 ; çekirdeği çağır
12 mov eax, 1 ; sys_exit
13 xor ebx, ebx ; çıkış kodu 0
14 int 0x80
```

ECX = dizginin **adres**i (içeriği değil).

Adım 4 / 9

DEĞİŞKENLER

EAX = 00000004

EBX = 00000001

ECX = hello (adres)

EDX = ?????????

ÇIKTI

"Merhaba, Dünya!" metninde ü harfi UTF-8'de 2 bayt olduğundan metin 16 bayttır; `equ $`
- `hello` uzunluğu assembler'a hesaplatır.

x86 Assembly: Linux (32 bit, NASM)

```
1 section .data
2 hello db 'Merhaba, Linux!', 10
3 len equ $ - hello ; = 16 bayt
4 section .text
5 global _start
6 _start:
7 mov eax, 4 ; sys_write
8 mov ebx, 1 ; stdout
9 mov ecx, hello ; dizgenin adresi
10 ► mov edx, len ; bayt sayısı
11 int 0x80 ; çekirdeği çağır
12 mov eax, 1 ; sys_exit
13 xor ebx, ebx ; çıkış kodu 0
14 int 0x80
```

EDX = 16 (10h): yazılacak bayt sayısı.

Adım 5 / 9

DEĞİŞKENLER

EAX = 00000004

EBX = 00000001

ECX = hello (adres)

EDX = 00000010

ÇIKTI

"Merhaba, Dünya!" metninde ü harfi UTF-8'de 2 bayt olduğundan metin 16 bayttır; `equ $ - hello` uzunluğu assembler'a hesaplatır.

x86 Assembly: Linux (32 bit, NASM)

```
1 section .data
2 hello db 'Merhaba, Linux!', 10
3 len equ $ - hello ; = 16 bayt
4 section .text
5 global _start
6 _start:
7 mov eax, 4 ; sys_write
8 mov ebx, 1 ; stdout
9 mov ecx, hello ; dizgenin adresi
10 mov edx, len ; bayt sayısı
11 ► int 0x80 ; çekirdeği çağır
12 mov eax, 1 ; sys_exit
13 xor ebx, ebx ; çıkış kodu 0
14 int 0x80
```

`int 0x80` : çekirdek çağırılır, metin yazılır.
Dönüşte EAX = yazılan bayt sayısı (16).

Adım 6 / 9

DEĞİŞKENLER

EAX = 00000010

EBX = 00000001

ECX = hello (adres)

EDX = 00000010

ÇIKTI

Merhaba, Linux!

"Merhaba, Dünya!" metninde ü harfi UTF-8'de 2 bayt olduğundan metin 16 bayttır; `equ $ - hello` uzunluğu assembler'a hesaplatır.

x86 Assembly: Linux (32 bit, NASM)

```
1 section .data
2 hello db 'Merhaba, Linux!', 10
3 len equ $ - hello ; = 16 bayt
4 section .text
5 global _start
6 _start:
7 mov eax, 4 ; sys_write
8 mov ebx, 1 ; stdout
9 mov ecx, hello ; dizgenin adresi
10 mov edx, len ; bayt sayısı
11 int 0x80 ; çekirdeği çağır
12 ► mov eax, 1 ; sys_exit
13 xor ebx, ebx ; çıkış kodu 0
14 int 0x80
```

EAX = 1: **sys_exit**.

Adım 7 / 9

DEĞİŞKENLER

EAX = 00000001

EBX = 00000001

ECX = hello (adres)

EDX = 00000010

ÇIKTI

Merhaba, Linux!

"Merhaba, Dünya!" metninde ü harfi UTF-8'de 2 bayt olduğundan metin 16 bayttır; `equ $ - hello` uzunluğu assembler'a hesaplatır.

x86 Assembly: Linux (32 bit, NASM)

```
1 section .data
2 hello db 'Merhaba, Linux!', 10
3 len equ $ - hello ; = 16 bayt
4 section .text
5 global _start
6 _start:
7 mov eax, 4 ; sys_write
8 mov ebx, 1 ; stdout
9 mov ecx, hello ; dizgenin adresi
10 mov edx, len ; bayt sayısı
11 int 0x80 ; çekirdeği çağır
12 mov eax, 1 ; sys_exit
13 xor ebx, ebx ; çıkış kodu 0
14 int 0x80
```

`xor ebx, ebx` : bir sayı kendisiyle XOR'lanınca 0 olur → çıkış kodu 0 (başarı).

`mov ebx, 0` 'dan kısadır.

Adım 8 / 9

DEĞİŞKENLER

EAX = 00000001

EBX = 00000000

ECX = hello (adres)

EDX = 00000010

ZF = 1

CF = 0

ÇIKTI

Merhaba, Linux!

"Merhaba, Dünya!" metninde ü harfi UTF-8'de 2 bayt olduğundan metin 16 bayttır; `equ $ - hello` uzunluğu assembler'a hesaplatır.

x86 Assembly: Linux (32 bit, NASM)

```
1 section .data
2 hello db 'Merhaba, Linux!', 10
3 len equ $ - hello ; = 16 bayt
4 section .text
5 global _start
6 _start:
7 mov eax, 4 ; sys_write
8 mov ebx, 1 ; stdout
9 mov ecx, hello ; dizgenin adresi
10 mov edx, len ; bayt sayısı
11 int 0x80 ; çekirdeği çağır
12 mov eax, 1 ; sys_exit
13 xor ebx, ebx ; çıkış kodu 0
▶ int 0x80
```

Çekirdek süreci 0 çıkış koduyla sonlandırır; bu çağrı geri dönmez.

Adım 9 / 9

DEĞİŞKENLER

EAX = 00000001

EBX = 00000000

ECX = hello (adres)

EDX = 00000010

ZF = 1

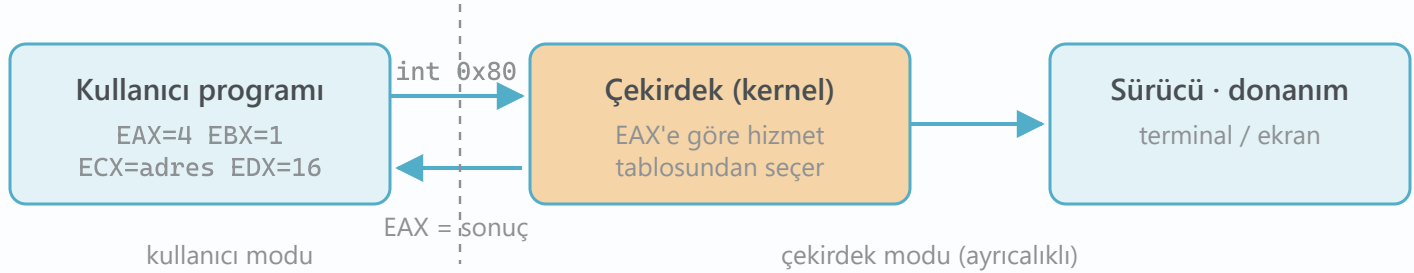
CF = 0

ÇIKTI

Merhaba, Linux!

"Merhaba, Dünya!" metninde ü harfi UTF-8'de 2 bayt olduğundan metin 16 bayttır; `equ $ - hello` uzunluğu assembler'a hesaplatır.

Sistem Çağrısı Nasıl Çalışır?



Ortam	Çağrı	Hizmet numarası	Parametreler
Linux 32 bit	<code>int 0x80</code>	EAX (4 = write, 1 = exit)	EBX, ECX, EDX
Linux 64 bit	<code>syscall</code>	RAX (1 = write, 60 = exit)	RDI, RSI, RDX
MS-DOS	<code>int 21h</code>	AH (09h = yaz, 4Ch = çık)	DX, AL ...
BIOS	<code>int 10h</code>	AH (0Eh = karakter yaz)	AL, BH ...

x86 Assembly: DOS (emu8086, COM)

```
► org 100h          ; COM programı
2 mov ah, 09h       ; DOS 09h: dizge yaz
3 lea dx, mesaj     ; DS:DX = adres
4 int 21h           ; DOS hizmetini çağır
5 mov ax, 4C00h     ; DOS 4Ch: sonlandır
6 int 21h
7 mesaj db 'Merhaba, DOS!$'
```

org 100h : COM programı kesimin 100h adresine yüklenir (önceki 256 bayt PSP). CS = DS = ES = SS.

Adım 1 / 6

DEĞİŞKENLER

IP = 0100h

AH = ??

DX = ????

ÇIKTI

09h hizmeti '\$' sonlandırıcı bekler; veri kodun arkasına konur, önde olsaydı komut gibi çalıştırılırdı. **mov ax, 4C00h** AL'yi (çıkış kodu) de sıfırlar.

x86 Assembly: DOS (emu8086, COM)

```
1 org 100h          ; COM programı
▶ mov ah, 09h        ; DOS 09h: dizge yaz
3 lea dx, mesaj      ; DS:DX = adres
4 int 21h            ; DOS hizmetini çağır
5 mov ax, 4C00h      ; DOS 4Ch: sonlandır
6 int 21h
7 mesaj db 'Merhaba, DOS!$'
```

AH = 09h: DOS'un "\$ ile biten dizgiyi yaz" hizmeti.

Adım 2 / 6

DEĞİŞKENLER

IP = 0102h

AH = 09h

DX = ????

ÇIKTI

09h hizmeti '\$' sonlandırıcı bekler; veri kodun arkasına konur, önde olsaydı komut gibi çalıştırılırdı. `mov ax, 4C00h` AL'yi (çıkış kodu) de sıfırlar.

x86 Assembly: DOS (emu8086, COM)

```
1 org 100h          ; COM programı
2 mov ah, 09h        ; DOS 09h: dizge yaz
3 ▶ lea dx, mesaj    ; DS:DX = adres
4 int 21h            ; DOS hizmetini çağır
5 mov ax, 4C00h      ; DOS 4Ch: sonlandır
6 int 21h
7 mesaj db 'Merhaba, DOS!$'
```

`lea dx, mesaj` : DS:DX dizginin adresini gösterir. Kod 13 bayt (0100h–010Ch) olduğundan mesaj **010Dh**'de.

Adım 3 / 6

DEĞİŞKENLER

IP = 0106h

AH = 09h

DX = 010Dh

ÇIKTI

09h hizmeti '\$' sonlandırıcı bekler; veri kodun arkasına konur, önde olsaydı komut gibi çalıştırılırdı. `mov ax, 4C00h` AL'yi (çıkış kodu) de sıfırlar.

x86 Assembly: DOS (emu8086, COM)

```
1 org 100h          ; COM programı
2 mov ah, 09h        ; DOS 09h: dizge yaz
3 lea dx, mesaj      ; DS:DX = adres
4 ► int 21h          ; DOS hizmetini çağır
5 mov ax, 4C00h      ; DOS 4Ch: sonlandır
6 int 21h
7 mesaj db 'Merhaba, DOS!$'
```

int 21h : DOS, '\$' görene kadar karakterleri ekrana yazar ('\$ yazılmaz).

Adım 4 / 6

DEĞİŞKENLER

IP = 0108h

AH = 09h

DX = 010Dh

ÇIKTI

Merhaba, DOS!

09h hizmeti '\$' sonlandırıcı bekler; veri kodun arkasına konur, önde olsaydı komut gibi çalıştırılırdı. `mov ax, 4C00h` AL'yi (çıkış kodu) de sıfırlar.

x86 Assembly: DOS (emu8086, COM)

```
1 org 100h          ; COM programı
2 mov ah, 09h        ; DOS 09h: dizge yaz
3 lea dx, mesaj      ; DS:DX = adres
4 int 21h            ; DOS hizmetini çağır
▶ mov ax, 4C00h      ; DOS 4Ch: sonlandır
6 int 21h
7 mesaj db 'Merhaba, DOS!$'
```

AX = 4C00h: AH = 4Ch "programı sonlandır", AL = 00h çıkış kodu.

Adım 5 / 6

DEĞİŞKENLER

IP = 010Bh

AH = 4Ch

AL = 00h

DX = 010Dh

ÇIKTI

Merhaba, DOS!

09h hizmeti '\$' sonlandırıcı bekler; veri kodun arkasına konur, önde olsaydı komut gibi çalıştırılırdı. `mov ax, 4C00h` AL'yi (çıkış kodu) de sıfırlar.

x86 Assembly: DOS (emu8086, COM)

```
1 org 100h          ; COM programı
2 mov ah, 09h        ; DOS 09h: dizge yaz
3 lea dx, mesaj      ; DS:DX = adres
4 int 21h            ; DOS hizmetini çağır
5 mov ax, 4C00h      ; DOS 4Ch: sonlandır
▶ int 21h
7 mesaj db 'Merhaba, DOS!$'
```

Program biter, denetim DOS'a döner. Veri **kodun arkasında** olduğu için işlemci hiçbir zaman veriyi komut sanıp çalıştırmaz.

Adım 6 / 6

DEĞİŞKENLER

IP = 010Dh

AH = 4Ch

AL = 00h

DX = 010Dh

ÇIKTI

Merhaba, DOS!

09h hizmeti '\$' sonlandırıcı bekler; veri kodun arkasına konur, önde olsaydı komut gibi çalıştırılırdı. `mov ax, 4C00h` AL'yi (çıkış kodu) de sıfırlar.

ASSEMBLY DİLLERİ

x86 Assembly: Windows (Win32 API, NASM)

```
extern MessageBoxA    ; user32.dll
extern ExitProcess    ; kernel32.dll

section .data
mesaj db 'Merhaba, Windows!', 0

section .text
global _start
_start:
    push 0             ; uType = MB_OK
    push mesaj         ; lpCaption (başlık)
    push mesaj         ; lpText (metin)
    push 0             ; hWnd = NULL
    call MessageBoxA
    push 0             ; uExitCode = 0
    call ExitProcess
```

Farklar

- Sistem çağırısı yok: işletim sistemi hizmetleri **DLL fonksiyonları** olarak çağrılır.
- Parametreler **yığına** sağdan sola itilir: C'deki `MessageBoxA(hWnd, lpText, lpCaption, uType)`.
- **stdcall**: yığını çağrılan fonksiyon temizler.
- Dizgiler **0 ile** biter (C dizgisi).
- Bağlarken süslenmiş ad gerekebilir: `_MessageBoxA@16` (16 = parametre baytı).

Kod İzleme: Parametreler Yığına

```
1 extern MessageBoxA
2 extern ExitProcess
3 section .data
4 mesaj db 'Merhaba...', 0
5 section .text
6 global _start
7
8 ► _start:
9     push 0          ; uType
10    push mesaj      ; lpCaption
11    push mesaj      ; lpText
12    push 0          ; hWnd
13    call MessageBoxA
14    push 0          ; uExitCode
15    call ExitProcess
```

yüksek adres ↑

ESP → 0012FF80h

0012FF7Ch

0012FF78h

0012FF74h

0012FF70h

0012FF6Ch

düşük adres ↓ (yığın bu yöne büyür)

Başlangıç: ESP = **0012FF80h** (örnek değer). Yığın düşük adreslere doğru büyür.

Adım 1 / 8

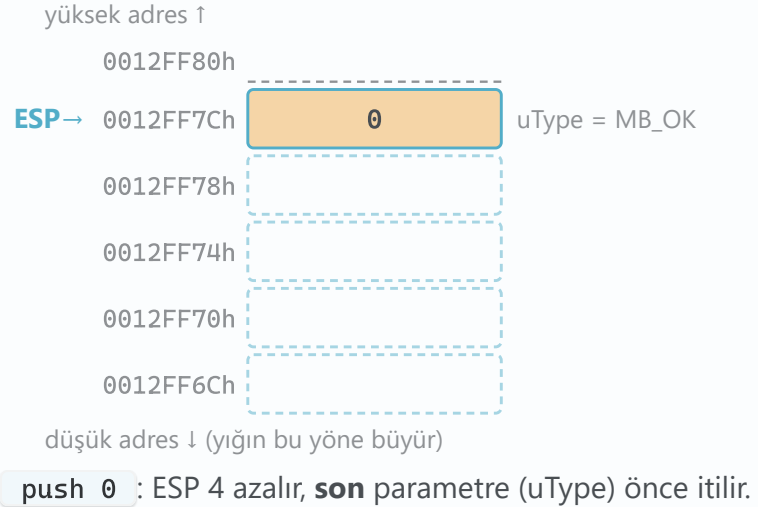
DEĞİŞKENLER

ESP = **0012FF80h**

EAX = ????????h

Kod İzleme: Parametreler Yığına

```
1 extern MessageBoxA
2 extern ExitProcess
3 section .data
4 mesaj db 'Merhaba...', 0
5 section .text
6 global _start
7 _start:
8   ► push 0          ; uType
9   push mesaj        ; lpCaption
10  push mesaj         ; lpText
11  push 0             ; hWnd
12  call MessageBoxA
13  push 0             ; uExitCode
14  call ExitProcess
```



Adım 2 / 8

DEĞİŞKENLER

ESP = 0012FF7Ch

EAX = ????????h

Kod İzleme: Parametreler Yığına

```
1 extern MessageBoxA
2 extern ExitProcess
3 section .data
4 mesaj db 'Merhaba...', 0
5 section .text
6 global _start
7 _start:
8   push 0           ; uType
9   ► push mesaj     ; lpCaption
10  push mesaj       ; lpText
11  push 0           ; hWnd
12  call MessageBoxA
13  push 0           ; uExitCode
14  call ExitProcess
```



`push mesaj` : başlık dizgisinin **adres**i itilir.

Adım 3 / 8

DEĞİŞKENLER

ESP = 0012FF78h

EAX = ?????????h

Kod İzleme: Parametreler Yığına

```
1 extern MessageBoxA
2 extern ExitProcess
3 section .data
4 mesaj db 'Merhaba...', 0
5 section .text
6 global _start
7 _start:
8   push 0          ; uType
9   push mesaj      ; lpCaption
10  ► push mesaj     ; lpText
11   push 0          ; hWnd
12   call MessageBoxA
13   push 0          ; uExitCode
14   call ExitProcess
```



Adım 4 / 8

DEĞİŞKENLER

ESP = 0012FF74h

EAX = ????????h

Kod İzleme: Parametreler Yığına

```

1 extern MessageBoxA
2 extern ExitProcess
3 section .data
4 mesaj db 'Merhaba...', 0
5 section .text
6 global _start
7 _start:
8   push 0          ; uType
9   push mesaj      ; lpCaption
10  push mesaj       ; lpText
11  ► push 0         ; hWnd
12  call MessageBoxA
13  push 0           ; uExitCode
14  call ExitProcess

```

yüksek adres ↑

0012FF80h

0012FF7Ch

0

uType = MB_OK

0012FF78h

&mesaj

lpCaption

0012FF74h

&mesaj

lpText

ESP → 0012FF70h

0

hWnd = NULL

0012FF6Ch

düşük adres ↓ (yığın bu yöne büyür)

push 0 : ilk parametre (hWnd) en son itilir → yığının tepesinde. Parametreler **sağdan sola** itilir.

Adım 5 / 8

DEĞİŞKENLER

ESP = 0012FF70h

EAX = ????????h

Kod İzleme: Parametreler Yığına

```
1 extern MessageBoxA
2 extern ExitProcess
3 section .data
4 mesaj db 'Merhaba...', 0
5 section .text
6 global _start
7 _start:
8   push 0          ; uType
9   push mesaj      ; lpCaption
10  push mesaj      ; lpText
11  push 0          ; hWnd
12  ► call MessageBoxA
13  push 0          ; uExitCode
14  call ExitProcess
```

yüksek adres ↑

0012FF80h

0012FF7Ch

0

uType = MB_OK

0012FF78h

&mesaj

lpCaption

0012FF74h

&mesaj

lpText

0012FF70h

0

hWnd = NULL

ESP → 0012FF6Ch

dönüş adr.

call'dan sonraki komut

düşük adres ↓ (yığın bu yöne büyür)

`call MessageBoxA` : dönüş adresi itilir ve fonksiyona atlanır;
ekranda ileti kutusu açılır.

Adım 6 / 8

DEĞİŞKENLER

ESP = 0012FF6Ch

EAX = ????????h

Kod İzleme: Parametreler Yığına

```
1 extern MessageBoxA
2 extern ExitProcess
3 section .data
4 mesaj db 'Merhaba...', 0
5 section .text
6 global _start
7 _start:
8     push 0          ; uType
9     push mesaj      ; lpCaption
10    push mesaj      ; lpText
11    push 0          ; hWnd
12    ► call MessageBoxA
13    push 0          ; uExitCode
14    call ExitProcess
```

yüksek adres ↑

ESP → 0012FF80h

0012FF7Ch

0012FF78h

0012FF74h

0012FF70h

0012FF6Ch

düşük adres ↓ (yığın bu yöne büyür)

Kullanıcı Tamam'a basınca fonksiyon **ret 16** ile döner: **stdcall**'da yığını çağrılan temizler, ESP yeniden 0012FF80h. EAX = 1 (IDOK).

Adım 7 / 8

DEĞİŞKENLER

ESP = 0012FF80h

EAX = 00000001h

Kod İzleme: Parametreler Yığına

```
1 extern MessageBoxA
2 extern ExitProcess
3 section .data
4 mesaj db 'Merhaba...', 0
5 section .text
6 global _start
7 _start:
8     push 0          ; uType
9     push mesaj      ; lpCaption
10    push mesaj      ; lpText
11    push 0          ; hWnd
12    call MessageBoxA
13    ► push 0          ; uExitCode
14    ► call ExitProcess
```



`push 0` + `call ExitProcess` : süreç 0 çıkış koduyla sonlanır; bu çağrı geri dönmez.

Adım 8 / 8

DEĞİŞKENLER

ESP = 0012FF7Ch

EAX = 00000001h

Motorola 68000 Assembly: Amiga

```
move.l 4.w,a6      ; a6 = ExecBase
lea    dosname(pc),a1 ; kütüphane adı
moveq  #0,d0       ; herhangi bir sürüm
jsr    -552(a6)     ; OpenLibrary()
tst.l  d0
beq.s  cik         ; açılmadı → çık
move.l d0,a6       ; a6 = DOSBase
jsr    -60(a6)      ; Output() → d0
move.l d0,d1       ; dosya tanıttıcısı
move.l #mesaj,d2    ; dizgenin adresi
moveq  #16,d3       ; bayt sayısı
jsr    -48(a6)      ; Write(d1,d2,d3)
move.l a6,a1
move.l 4.w,a6
jsr    -414(a6)     ; CloseLibrary()
cik moveq #0,d0      ; dönüş kodu 0
rts
dosname dc.b 'dos.library',0
mesaj   dc.b 'Merhaba, Amiga!',10
```

Farklar

- 8 veri (`d0–d7`) ve 8 adres (`a0–a7`) yazmacı; **kaynak, hedef** sırası.
- Boyut soneki: `.b` , `.w` , `.l` ; sabitler `#` ile.
- Hizmetler kütüphane tabanından negatif uzaklıkla (`-48(a6)`) çağrılır.
- Kütüphane açılır, kullanılır, kapatılır.

ARM Assembly: Linux (32 bit, GNU as)

```
.data
mesaj: .ascii "Merhaba, ARM!\n"
uzun = . - mesaj      @ 14 bayt

.text
.global _start
_start:
    mov    r7, #4       @ sys_write
    mov    r0, #1       @ stdout
    ldr    r1, =mesaj    @ dizgenin adresi
    mov    r2, #uzun     @ bayt sayısı
    svc    #0           @ çekirdeği çağır
    mov    r7, #1       @ sys_exit
    mov    r0, #0       @ çıkış kodu
    svc    #0
```

Farklar

- RISC: 16 yazmaç (**r0-r15**), sabit 32 bitlik komutlar.
- Hizmet numarası **r7**'de, parametreler **r0-r2**'de.
- Çekirdeğe geçiş: **svc #0** (eski adı **swi**).
- Bellekten yalnızca **ldr** / **str** ile okunur/yazılır.

GNU as'ta **db** değil **.ascii** kullanılır; **. - mesaj** uzunluğu assembler'a hesaplanır.

PowerPC (PPC) Assembly: Linux (32 bit)

```
.data
mesaj: .ascii "Merhaba, PowerPC!\n"
uzun = . - mesaj      # 18 bayt

.text
.globl _start
_start:
li    0, 4             # r0 = sys_write
li    3, 1             # r3 = stdout
lis   4, mesaj@ha      # r4 = adres (üst)
addi  4, 4, mesaj@l    # + alt yarı
li    5, uzun          # r5 = bayt sayısı
sc                      # çekirdeği çağır
li    0, 1             # r0 = sys_exit
li    3, 0             # r3 = çıkış kodu
sc
```

Farklar

- RISC: 32 genel yazmaç; GNU as'ta yazmaçlar düz sayıyla yazılır (`3` = r3).
- Hizmet numarası **r0**'da, parametreler **r3, r4, r5**'te; geçiş `sc`.
- 32 bitlik adres, 16 bitlik iki parçada yüklenir (`@ha` , `@l`).

ASSEMBLY DİLLERİ

Aynı “Merhaba”, Altı Ortam

Ortam	İşlemci	OS'a geçiş	Hizmet no.	Parametreler	Dizgi sonu
Linux	x86 (32)	<code>int 0x80</code>	EAX	EBX, ECX, EDX	uzunluk verilir
DOS	8086	<code>int 21h</code>	AH	DX (adres)	<code>'\$'</code>
Windows	x86 (32)	<code>call</code> DLL	fonksiyon adı	yığın	<code>0</code>
AmigaOS	68000	<code>jsr -48(a6)</code>	kütüphane uzaklığı	d1, d2, d3	uzunluk verilir
Linux	ARM	<code>svc #0</code>	r7	r0, r1, r2	uzunluk verilir
Linux	PowerPC	<code>sc</code>	r0	r3, r4, r5	uzunluk verilir

Assembly kodu hem **işlemciye** (komut kümesi, yazmaçlar) hem de **işletim sistemine** (çağrı yolu, numaralar) bağlıdır. Yüksek seviyeli dillerin en büyük kazancı bu bağımlılığı derleyiciye/yorumlayıcıya devretmektir.

Yüksek Seviyeli Diller

Derlenen diller

- Kaynak kod önceden **makine diline** çevrilir.
- Hataların çoğu **derleme sırasında** yakalanır.
- **Performans** avantajı sağlar.
- Aşamalar: **derleme** (compiler) ve **bağlama** (linker).
- Çıktı belirli bir işlemci + işletim sistemine özgüdür.

Yorumlanan diller

- Kaynak kod **çalışma anında** yorumlayıcı tarafından işlenir.
- **Hızlı geliştirme** ve deneme imkânı sağlar.
- Kaynak kod **platformdan bağımsızdır**: yorumlayıcı olan her yerde çalışır.
- Hatalar ancak ilgili satır çalışınca ortaya çıkabilir.
- Genelde daha yavaştır.



Hibrit yaklaşım: Java ve C# önce ara koda (bytecode/IL) derlenir; sanal makine bu kodu çalışırken makine diline çevirir (**JIT**, just-in-time). Python ve JavaScript motorları da içeride bytecode/JIT kullanır.

Derleme Süreci (C, gcc)

1 · **Önişlemci** `#include` ve `#define` açılır → `tekcift.i` (gcc -E)

Adım 1 / 5

Tek komutla: `gcc tecift.c -o tecift` . Ara assembly'yi görmek için `gcc -S -masm=intel tecift.c` .

Derleme Süreci (C, gcc)

1 • **Önişlemci** `#include` ve `#define` açılır → `tekcift.i` (gcc -E)

2 • **Derleyici** C kodu sözdizimi ve tip denetiminden geçer, **assembly**'ye çevrilir → `tekcift.s` (gcc -S)

Adım 2 / 5

Tek komutla: `gcc tekcraft.c -o tekcraft`. Ara assembly'yi görmek için `gcc -S -masm=intel tekcraft.c`.

Derleme Süreci (C, gcc)

1 · Önışlemci `#include` ve `#define` açılır → `tekcift.i` (gcc -E)

2 · Derleyici C kodu sözdizimi ve tip denetiminden geçer, **assembly**'ye çevrilir → `tekcift.s` (gcc -S)

3 · Assembler Assembly, makine koduna çevrilir → nesne dosyası `tekcift.o` (gcc -c)

Adım 3 / 5

Tek komutla: `gcc tecift.c -o tecift` . Ara assembly'yi görmek için `gcc -S -masm=intel tecift.c` .

Derleme Süreci (C, gcc)

1 • Önışlemci `#include` ve `#define` açılır → `tekcraft.i` (gcc -E)

2 • Derleyici C kodu sözdizimi ve tip denetiminden geçer, **assembly**'ye çevrilir → `tekcraft.s` (gcc -S)

3 • Assembler Assembly, makine koduna çevrilir → nesne dosyası `tekcraft.o` (gcc -c)

4 • Bağlayıcı Nesne dosyası, `printf` / `scanf` gibi kütüphane kodlarıyla birleştirilir → çalıştırılabilir `tekcraft` / `tekcraft.exe`

Adım 4 / 5

Tek komutla: `gcc tekcraft.c -o tekcraft`. Ara assembly'yi görmek için `gcc -S -masm=intel tekcraft.c`.

Derleme Süreci (C, gcc)

1 • Önışlemci `#include` ve `#define` açılır → `tekift.i` (gcc -E)

2 • Derleyici C kodu sözdizimi ve tip denetiminden geçer, **assembly**'ye çevrilir → `tekift.s` (gcc -S)

3 • Assembler Assembly, makine koduna çevrilir → nesne dosyası `tekift.o` (gcc -c)

4 • Bağlayıcı Nesne dosyası, `printf` / `scanf` gibi kütüphane kodlarıyla birleştirilir → çalıştırılabilir `tekift` / `tekift.exe`

5 • Yükleyici İşletim sistemi programı belleğe yükler ve `main` 'e atlar.

Adım 5 / 5

Tek komutla: `gcc tekift.c -o tekift` . Ara assembly'yi görmek için `gcc -S -masm=intel tekift.c` .

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
```

```
▶ mov bx, 0 ; toplam = 0
```

```
▶ mov cx, 10 ; i = 10
```

```
4 kosul: cmp cx, 0 ; i > 0 ?
```

```
5 jle bitti
```

```
6 add bx, cx ; toplam += i
```

```
7 sub cx, 3 ; i -= 3
```

```
8 jmp kosul
```

```
9 bitti: mov toplam, bx
```

```
10 ret
```

```
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
```

```
i = 10;
```

```
while (i > 0) {
```

```
    toplam += i;
```

```
    i -= 3;
```

```
}
```

toplam (BX)

0

i (CX)

10

$10 + 7 + 4 + 1 = 22$

Derleyici değişkenleri yazmaçlara yerleştirdi: toplam → **BX**, i → **CX**. while koşulu döngünün başında sınanır.

Adım 1 / 11

DEĞİŞKENLER

BX = 0

CX = 10

ZF = 0

SF = 0

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0      ; toplam = 0
3 mov cx, 10     ; i = 10
▶ kosul: cmp cx, 0 ; i > 0 ?
▶ jle bitti
6 add bx, cx     ; toplam += i
7 sub cx, 3      ; i -= 3
8 jmp kosul
9 bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

0

i (CX)

10

cmp: i - 0 → ZF=0 SF=0 OF=0

jle atlamaz → gövde

10 + 7 + 4 + 1 = 22

cmp cx, 0 + jle : i = 10 > 0 olduğundan atlanmaz, gövdeye girilir (C'deki koşul tersine çevrilip "çık" atlamasına dönüştü).

Adım 2 / 11

DEĞİŞKENLER

BX = 0

CX = 10

ZF = 0

SF = 0

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0          ; toplam = 0
3 mov cx, 10         ; i = 10
4 kosul: cmp cx, 0    ; i > 0 ?
5 jle bitti
▶ add bx, cx          ; toplam += i
▶ sub cx, 3           ; i -= 3
▶ jmp kosul
9 bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

10

i (CX)

7

 $10 + 7 + 4 + 1 = 22$

Gövde: toplam = **10**, i = **7**; **jmp kosul** ile koşula dönülür.

Adım 3 / 11

DEĞİŞKENLER

BX = 10

CX = 7

ZF = 0

SF = 0

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0          ; toplam = 0
3 mov cx, 10         ; i = 10
▶ kosul: cmp cx, 0    ; i > 0 ?
▶ jle bitti
6 add bx, cx         ; toplam += i
7 sub cx, 3          ; i -= 3
8 jmp kosul
9 bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

10

i (CX)

7

cmp: i - 0 → ZF=0 SF=0 OF=0

jle atlamaz → gövde

10 + 7 + 4 + 1 = 22

cmp cx, 0 + jle : i = 7 > 0 olduğundan atlanmaz, gövdeye girilir (C'deki koşul tersine çevrilip "çık" atlamasına dönüştü).

Adım 4 / 11

DEĞİŞKENLER

BX = 10

CX = 7

ZF = 0

SF = 0

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0          ; toplam = 0
3 mov cx, 10         ; i = 10
4 kosul: cmp cx, 0    ; i > 0 ?
5 jle bitti
▶ add bx, cx          ; toplam += i
▶ sub cx, 3           ; i -= 3
▶ jmp kosul
9 bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

17

i (CX)

4 $10 + 7 + 4 + 1 = 22$

Gövde: toplam = **17**, i = **4**; **jmp kosul** ile koşula dönülür.

Adım 5 / 11

DEĞİŞKENLER**BX = 17****CX = 4****ZF = 0****SF = 0**

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0          ; toplam = 0
3 mov cx, 10         ; i = 10
▶ kosul: cmp cx, 0    ; i > 0 ?
▶ jle bitti
6 add bx, cx         ; toplam += i
7 sub cx, 3          ; i -= 3
8 jmp kosul
9 bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

17

i (CX)

4

cmp: i - 0 → ZF=0 SF=0 OF=0

jle atlamaz → gövde

10 + 7 + 4 + 1 = 22

cmp cx, 0 + jle : i = 4 > 0 olduğundan atlanmaz, gövdeye girilir (C'deki koşul tersine çevrilip "çık" atlamasına dönüştü).

Adım 6 / 11

DEĞİŞKENLER

BX = 17

CX = 4

ZF = 0

SF = 0

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0          ; toplam = 0
3 mov cx, 10         ; i = 10
4 kosul: cmp cx, 0    ; i > 0 ?
5 jle bitti
▶ add bx, cx          ; toplam += i
▶ sub cx, 3           ; i -= 3
▶ jmp kosul
9 bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

21

i (CX)

1 $10 + 7 + 4 + 1 = 22$

Gövde: toplam = **21**, i = **1**; `jmp kosul` ile koşula dönülür.

Adım 7 / 11

DEĞİŞKENLER

BX = **21**CX = **1**ZF = **0**SF = **0**

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0          ; toplam = 0
3 mov cx, 10         ; i = 10
4 ▶ kosul: cmp cx, 0  ; i > 0 ?
5 ▶ jle bitti
6 add bx, cx         ; toplam += i
7 sub cx, 3          ; i -= 3
8 jmp kosul
9 bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

21

i (CX)

1

cmp: i - 0 → ZF=0 SF=0 OF=0

jle atlamaz → gövde

10 + 7 + 4 + 1 = 22

cmp cx, 0 + jle : i = 1 > 0 olduğundan atlanmaz, gövdeye girilir (C'deki koşul tersine çevrilip "çık" atlamasına dönüştü).

Adım 8 / 11

DEĞİŞKENLER

BX = 21

CX = 1

ZF = 0

SF = 0

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0          ; toplam = 0
3 mov cx, 10         ; i = 10
4 kosul: cmp cx, 0    ; i > 0 ?
5 jle bitti
▶ add bx, cx          ; toplam += i
▶ sub cx, 3           ; i -= 3
▶ jmp kosul
9 bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

22

i (CX)

-2

 $10 + 7 + 4 + 1 = 22$ Gövde: toplam = **22**, i = **-2**; `jmp kosul` ile koşula dönülür.

Adım 9 / 11

DEĞİŞKENLER

BX = 22

CX = -2

ZF = 0

SF = 1

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0          ; toplam = 0
3 mov cx, 10         ; i = 10
▶ kosul: cmp cx, 0    ; i > 0 ?
▶ jle bitti
6 add bx, cx          ; toplam += i
7 sub cx, 3           ; i -= 3
8 jmp kosul
9 bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

22

i (CX)

-2

cmp: i - 0 → ZF=0 SF=1 OF=0

jle atlar → bitti

10 + 7 + 4 + 1 = 22

i = -2: koşul i > 0 yanlış (i ≤ 0, SF ≠ OF) → **jle bitti**
atlar, döngüden çıkılır.

Adım 10 / 11

DEĞİŞKENLER

BX = 22

CX = -2

ZF = 0

SF = 1

Kod İzleme: Derleyicinin Ürettiği Döngü

```
1 org 100h
2 mov bx, 0          ; toplam = 0
3 mov cx, 10         ; i = 10
4 kosul: cmp cx, 0    ; i > 0 ?
5 jle bitti
6 add bx, cx          ; toplam += i
7 sub cx, 3           ; i -= 3
8 jmp kosul
► bitti: mov toplam, bx
10 ret
11 toplam dw ?
```

C kaynağı

```
toplam = 0;
i = 10;
while (i > 0) {
    toplam += i;
    i -= 3;
}
```

toplam (BX)

22

i (CX)

-2**toplam ← BX = 22** $10 + 7 + 4 + 1 = 22$

Sonuç belleğe yazılır: toplam = **22**. Tek bir C döngüsü; karşılaştırma, koşullu atlama ve koşulsuz atlamaya çevrildi.

Adım 11 / 11

DEĞİŞKENLERBX = **22**CX = **-2**ZF = **0**SF = **1**

Derlenen Diller

- Kaynak kod çalışmadan önce **derleyici** tarafından bütünüyle incelenir.
- Derlenmiş kod doğrudan işlemcide çalıştığı için **daha hızlıdır**.
- Tip ve sözdizimi hataları **derleme aşamasında** tespit edilir.

C ve C++

sistem programlaması,
işletim sistemleri, gömülü
yazılım ve yüksek
performanslı uygulamalar

Rust

güvenli ve performanslı
sistem programlaması
(bellek güvenliği derleyicide
denetlenir)

Fortran

bilimsel ve sayısal
hesaplamalar; ilk yüksek
seviyeli dil (1957)

Go

sunucu ve ağ yazılımı; hızlı
derleme, eşzamanlılık

Ada

havacılık, savunma,
demiryolu gibi güvenlik-
kritik sistemler

Java, C# (hibrit)

ara koda derlenir, sanal
makinede JIT ile çalışır

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7     for (int i = 1; i <= number; i++) {
8         if (i % 2 == 0) {
9             printf("%d sayisi bir cift sayidir.\n", i);
10        } else {
11            printf("%d sayisi bir tek sayidir.\n", i);
12        }
13    }
14    return 0;
15 }
```

Bütün örnek programlar aynı işi yapar: bir sayı okur, 1'den o sayıya kadar her sayının tek mi çift mi olduğunu yazar.

Adım 1 / 10

DEĞİŞKENLER

number = ?

ÇIKTI

Bir sayi girin:

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7     for (int i = 1; i <= number; i++) {
8         if (i % 2 == 0) {
9             printf("%d sayisi bir cift sayidir.\n", i);
10        } else {
11            printf("%d sayisi bir tek sayidir.\n", i);
12        }
13    }
14    return 0;
15 }
```

Kullanıcı 3 girer. `scanf` 'e
değişkenin **adres**i (`&number`) verilir.

Adım 2 / 10

DEĞİŞKENLER

`number = 3`

ÇIKTI

Bir sayi girin: 3

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7     for (int i = 1; i <= number; i++) {
8         if (i % 2 == 0) {
9             printf("%d sayisi bir cift sayidir.\n", i);
10        } else {
11            printf("%d sayisi bir tek sayidir.\n", i);
12        }
13    }
14    return 0;
15 }
```

Döngü başlar: $i = 1$, koşul $1 \leq 3$ doğru.

Adım 3 / 10

DEĞİŞKENLER

number = 3

i = 1

ÇIKTI

Bir sayi girin: 3

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7     for (int i = 1; i <= number; i++) {
8         if (i % 2 == 0) {
9             printf("%d sayisi bir cift sayidir.\n", i);
10        } else {
11            printf("%d sayisi bir tek sayidir.\n", i);
12        }
13    }
14    return 0;
15 }
```

$1 \% 2 = 1 \neq 0 \rightarrow$ **else** dalı: "tek".

Adım 4 / 10

DEĞİŞKENLER

number = 3

i = 1

$i \% 2 = 1$

ÇIKTI

```
Bir sayi girin: 3
1 sayisi bir tek sayidir.
```

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7
8     for (int i = 1; i <= number; i++) {
9         if (i % 2 == 0) {
10             printf("%d sayisi bir cift sayidir.\n", i);
11         } else {
12             printf("%d sayisi bir tek sayidir.\n", i);
13         }
14     }
15     return 0;
16 }
```

`i++` → 2; $2 \leq 3$ doğru.

Adım 5 / 10

DEĞİŞKENLER

number = 3

i = 2

ÇIKTI

```
Bir sayi girin: 3
1 sayisi bir tek sayidir.
```

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7     for (int i = 1; i <= number; i++) {
8         if (i % 2 == 0) {
9             printf("%d sayisi bir cift sayidir.\n", i);
10        } else {
11            printf("%d sayisi bir tek sayidir.\n", i);
12        }
13    }
14    return 0;
15 }
```

$2 \% 2 = 0 \rightarrow$ "çift".

Adım 6 / 10

DEĞİŞKENLER

number = 3

i = 2

$i \% 2 = 0$

ÇIKTI

```
Bir sayi girin: 3
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
```

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7
8     for (int i = 1; i <= number; i++) {
9         if (i % 2 == 0) {
10             printf("%d sayisi bir cift sayidir.\n", i);
11         } else {
12             printf("%d sayisi bir tek sayidir.\n", i);
13         }
14     }
15     return 0;
16 }
```

$i = 3; 3 \leq 3$ doğru.

Adım 7 / 10

DEĞİŞKENLER

number = 3

i = 3

ÇIKTI

```
Bir sayi girin: 3
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
```

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7     for (int i = 1; i <= number; i++) {
8         if (i % 2 == 0) {
9             printf("%d sayisi bir cift sayidir.\n", i);
10        } else {
11            printf("%d sayisi bir tek sayidir.\n", i);
12        }
13    }
14    return 0;
15 }
```

$3 \% 2 = 1 \rightarrow \text{"tek"}$.

Adım 8 / 10

DEĞİŞKENLER

number = 3

i = 3

$i \% 2 = 1$

ÇIKTI

```
Bir sayi girin: 3
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
3 sayisi bir tek sayidir.
```

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7     for (int i = 1; i <= number; i++) {
8         if (i % 2 == 0) {
9             printf("%d sayisi bir cift sayidir.\n", i);
10        } else {
11            printf("%d sayisi bir tek sayidir.\n", i);
12        }
13    }
14    return 0;
15 }
```

$i = 4; 4 \leq 3$ **yanlış** → döngü biter.

Adım 9 / 10

DEĞİŞKENLER

number = 3

i = 4

ÇIKTI

```
Bir sayi girin: 3
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
3 sayisi bir tek sayidir.
```

Görev: 1'den n'ye Tek/Çift — C

```
1 #include <stdio.h>
2
3 int main() {
4     int number;
5     printf("Bir sayi girin: ");
6     scanf("%d", &number);
7     for (int i = 1; i <= number; i++) {
8         if (i % 2 == 0) {
9             printf("%d sayisi bir cift sayidir.\n", i);
10        } else {
11            printf("%d sayisi bir tek sayidir.\n", i);
12        }
13    }
14    ▶ return 0;
15 }
```

return 0 : işletim sistemine
"başarılı" çıkış kodu.

Adım 10 / 10

DEĞİŞKENLER

number = 3

dönüş = 0

ÇIKTI

```
Bir sayi girin: 3
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
3 sayisi bir tek sayidir.
```

C++

```
#include <iostream>

int main() {
    int number;
    std::cout << "Bir sayı girin: ";
    std::cin >> number;
    for (int i = 1; i <= number; i++) {
        if (i % 2 == 0) {
            std::cout << i << " sayısı bir çift sayıdır."
                << std::endl;
        } else {
            std::cout << i << " sayısı bir tek sayıdır."
                << std::endl;
        }
    }
    return 0;
}
```

C++ (1985, Bjarne Stroustrup)

- C'nin üst kümesi; sınıflar, şablonlar, istisnalar.
- G/Ç için akışlar: `std::cout <<` yazar, `std::cin >>` okur.
- Derleme: `g++ tekciift.cpp -o tekciift`
- `std::endl` satır sonu ekler ve tamponu boşaltır; çok satırdaki `'\n'` daha hızlıdır.

YÜKSEK SEVİYELİ DİLLER · DERLENEN (ARA KODA)

C#

```
using System;

class Program {
    static void Main() {
        Console.Write("Bir sayı girin: ");
        int number = int.Parse(Console.ReadLine());
        for (int i = 1; i <= number; i++) {
            if (i % 2 == 0) {
                Console.WriteLine($"{i} sayısı bir çift sayıdır.");
            } else {
                Console.WriteLine($"{i} sayısı bir tek sayıdır.");
            }
        }
    }
}
```

C# (2000, Microsoft)

- .NET platformu; **IL** ara diline derlenir, CLR çalışırken JIT ile makine koduna çevirir.
- `($"{i} ..."` : dizge içine değer yerleştirme (interpolation).
- Çalıştırma: `dotnet run`
- `int.Parse` sayı olmayan girdide istisna fırlatır; `int.TryParse` daha güvenlidir.

YÜKSEK SEVİYELİ DİLLER · DERLENEN (BYTECODE)

Java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Bir sayı girin: ");
        int number = scanner.nextInt();
        for (int i = 1; i <= number; i++) {
            if (i % 2 == 0) {
                System.out.println(i + " sayisi bir cift sayidir.");
            } else {
                System.out.println(i + " sayisi bir tek sayidir.");
            }
        }
    }
}
```

Java (1995, Sun Microsystems)

- `javac` → platformdan bağımsız **bytecode** (`.class`) → JVM'de JIT.
- "Bir kez yaz, her yerde çalıştır."
- Çalıştırma: `javac`
`Main.java` ve `java Main`
- Her şey bir sınıf içinde; dosya adı = `public` sınıf adı.

Go

```
package main

import "fmt"

func main() {
    var number int
    fmt.Print("Bir sayi girin: ")
    fmt.Scan(&number)
    for i := 1; i <= number; i++ {
        if i%2 == 0 {
            fmt.Printf("%d sayisi bir cift sayidir.\n", i)
        } else {
            fmt.Printf("%d sayisi bir tek sayidir.\n", i)
        }
    }
}
```

Go (2009, Google)

- Doğrudan makine koduna derlenir; tek, bağımsız çalıştırılabilir dosya.
- Çöp toplayıcı ve hafif iş parçacıkları (**goroutine**).
- Çalıştırma: `go run tekcift.go`
- Koşul ve döngülerde parantez yok; `:=` tipi çıkarır.

Rust

```
use std::io;

fn main() {
    let mut input = String::new();
    println!("Bir sayi girin:");
    io::stdin().read_line(&mut input)
        .expect("okuma hatasi");
    let number: i32 = input.trim().parse()
        .expect("cevirme hatasi");
    for i in 1..=number {
        if i % 2 == 0 {
            println!("{}", sayisi bir cift sayidir.", i);
        } else {
            println!("{}", sayisi bir tek sayidir.", i);
        }
    }
}
```

Rust (2015, Mozilla)

- Çöp toplayıcı olmadan **bellek güvenliği**: sahiplik (ownership) kuralları derleyicide denetlenir.
- Değişkenler varsayılan olarak değiştirilemez; **mut** gerekir.
- **1..=number** : üst sınır dahil aralık.
- Çalıştırma: **rustc tekcift.rs** ya da **cargo run**

Fortran

```
program Main
  implicit none
  integer :: number, i
  write(*,*) 'Bir sayi girin:'
  read(*,*) number
  do i = 1, number
    if (mod(i, 2) == 0) then
      write(*,*) i, ' sayisi bir cift sayidir.'
    else
      write(*,*) i, ' sayisi bir tek sayidir.'
    end if
  end do
end program Main
```

Fortran (1957, IBM · John Backus)

- İlk yaygın yüksek seviyeli dil: FORMula TRANslation.
- Bilimsel hesaplamada hâlâ yaygın (hava tahmini, fizik).
- `implicit none` : tüm değişkenler açıkça tanımlanmalı.
- Çalıştırma: `gfortran tekcift.f90 -o tekcift`
- `write(*,*)` serbest biçim yazar; sayıların önünde boşluk görülebilir.

Ada

```
with Ada.Text_IO;           use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
```

```
procedure Main is
  Number : Integer;
begin
  Put ("Bir sayi girin: ");
  Get (Number);
  for I in 1 .. Number loop
    Put (I, Width => 0);
    if I mod 2 = 0 then
      Put_Line (" sayisi bir cift sayidir.");
    else
      Put_Line (" sayisi bir tek sayidir.");
    end if;
  end loop;
end Main;
```

Ada (1983, ABD Savunma Bakanlığı)

- Güçlü tip denetimi; havacılık ve savunma yazılımı.
- Çalıştırma: `gnatmake main.adb`
- Tamsayı `Get / Put` için `Ada.Integer_Text_IO` gerekir; `Put_Line` satır sonu ekler.

Yorumlanan Diller

- Kaynak kod **çalışma anında** yorumlanır; ayrı bir derleme adımı yoktur.
- Hızlı geliştirme ve deneme (etkileşimli kabuk, REPL).

Python

genel amaçlı, yüksek seviyeli; veri bilimi, yapay zekâ

JavaScript

web geliştirme; tarayıcıda ve Node.js'te çalışır

Ruby

kullanım kolaylığı ve esneklik; Ruby on Rails



Python

```
# Kullanıcıdan girdi alma
number = int(input("Bir sayi girin: "))

# 1'den number'a kadar sayıları kontrol et
for i in range(1, number + 1):
    if i % 2 == 0:
        print(f"{i} sayisi bir cift sayidir.")
    else:
        print(f"{i} sayisi bir tek sayidir.")
```

Python (1991, Guido van Rossum)

- Bloklar süslü parantezle değil **girintiyle** belirlenir.
- `range(1, number + 1)` :
üst sınır **dahil değildir**, bu yüzden +1.
- `input` metin döndürür;
`int( ... )` sayıya çevirir.
- Çalıştırma: `python tekcift.py`
- Dinamik tipli: değişkenin tipi çalışırken belirlenir.

C sürümünde 15 satır gereken iş 6 satırda: yüksek seviyeli dillerin **ifade gücü**. Bedeli: CPython aynı döngüyü C'den onlarca kat yavaş çalıştırır.

JavaScript

```
// Kullanıcıdan girdi alma
let number = Number(prompt("Bir sayi girin:"));

// 1'den number'a kadar sayıları kontrol et
for (let i = 1; i <= number; i++) {
  if (i % 2 === 0) {
    console.log(i + " sayisi bir cift sayidir.");
  } else {
    console.log(i + " sayisi bir tek sayidir.");
  }
}
```

JavaScript (1995, Brendan Eich)

- Tarayıcıların dili; Node.js ile sunucuda da çalışır.
- Modern motorlar (V8) kodu JIT ile makine koduna çevirir.
- `prompt` **metin** döndürür; `Number( ... )` ile çevirmek güvenlidir.
- `===` : tip dönüştürmeden eşitlik.

Ruby

```
# Kullanıcıdan girdi alma
print "Bir sayi girin: "
number = gets.chomp.to_i

# 1'den number'a kadar sayıları kontrol et
for i in 1..number do
  if i % 2 == 0
    puts "#{i} sayisi bir cift sayidir."
  else
    puts "#{i} sayisi bir tek sayidir."
  end
end
```

Ruby (1995, Yukihiro Matsumoto)

- Her şey bir nesnedir:
`gets.chomp.to_i` art arda metod çağrısıdır.
- `chomp` satır sonunu atar,
`to_i` tamsayıya çevirir.
- `1..number` : üst sınır dahil aralık.
- `"#{i}"` : dizge içine değer yerleştirme.
- Çalıştırma: `ruby tekciift.rb`

PHP

```
<?php
// Kullanıcıdan girdi alma
echo "Bir sayi girin: ";
$number = (int) trim(fgets(STDIN));

// 1'den number'a kadar sayıları kontrol et
for ($i = 1; $i <= $number; $i++) {
    if ($i % 2 == 0) {
        echo $i . " sayisi bir cift sayidir.\n";
    } else {
        echo $i . " sayisi bir tek sayidir.\n";
    }
}
?>
```

PHP (1995, Rasmus Lerdorf)

- Sunucu tarafı web dili; HTML içine gömülebilir.
- Değişkenler `$` ile başlar; `.` dizge birleştirir.
- `fgets(STDIN)` komut satırından bir satır okur; `trim` satır sonunu atar.
- Çalıştırma: `php tekcift.php`

Perl

```
use strict;
use warnings;

# Kullanıcıdan girdi alma
print "Bir sayi girin: ";
my $number = <STDIN>;
chomp $number;

# 1'den number'a kadar sayıları kontrol et
for my $i (1..$number) {
    if ($i % 2 == 0) {
        print "$i sayisi bir cift sayidir.\n";
    } else {
        print "$i sayisi bir tek sayidir.\n";
    }
}
```

Perl (1987, Larry Wall)

- Metin işleme ve sistem yönetimi; güçlü düzenli ifadeler.
- `<STDIN>` bir satır okur, `chomp` satır sonunu atar.
- Çift tırnaklı dizgede `$i` doğrudan değerle değiştirilir.
- `use strict; use warnings;` yaygın hataları yakalar.
- Çalıştırma: `perl tekcift.pl`

MATLAB

```
% Kullanıcıdan girdi alma
number = input('Bir sayi girin: ');

% 1'den number'a kadar sayıları kontrol et
for i = 1:number
    if mod(i, 2) == 0
        fprintf('%d sayisi bir cift sayidir.\n', i);
    else
        fprintf('%d sayisi bir tek sayidir.\n', i);
    end
end
```

MATLAB (1984, MathWorks)

- Matris tabanlı sayısal hesaplama; mühendislikte, özellikle sinyal işleme ve denetimde yaygın.
- `1:number` : 1, 2, ..., number vektörü.
- Satır sonundaki `;` sonucun ekrana basılmasını engeller.
- Ücretsiz uyumlu alternatif: GNU Octave.

Aynı Program, On Dört Dil

Dil	Model	Döngü başlığı	Kalan	Yazdırma
C	derlenen	<code>for (int i = 1; i ≤ n; i++)</code>	<code>i % 2</code>	<code>printf</code>
C++	derlenen	<code>for (int i = 1; i ≤ n; i++)</code>	<code>i % 2</code>	<code>std::cout <<</code>
C#	IL + JIT	<code>for (int i = 1; i ≤ n; i++)</code>	<code>i % 2</code>	<code>Console.WriteLine</code>
Java	bytecode + JIT	<code>for (int i = 1; i ≤ n; i++)</code>	<code>i % 2</code>	<code>System.out.println</code>
Go	derlenen	<code>for i := 1; i ≤ n; i++</code>	<code>i%2</code>	<code>fmt.Printf</code>
Rust	derlenen	<code>for i in 1..=n</code>	<code>i % 2</code>	<code>println!</code>
Fortran	derlenen	<code>do i = 1, n</code>	<code>mod(i, 2)</code>	<code>write(*,*)</code>
Ada	derlenen	<code>for I in 1 .. N loop</code>	<code>I mod 2</code>	<code>Put_Line</code>
Python	yorumlanan	<code>for i in range(1, n + 1):</code>	<code>i % 2</code>	<code>print</code>
JavaScript	yorumlanan/JIT	<code>for (let i = 1; i ≤ n; i++)</code>	<code>i % 2</code>	<code>console.log</code>
Ruby	yorumlanan	<code>for i in 1..n do</code>	<code>i % 2</code>	<code>puts</code>
PHP	yorumlanan	<code>for (\$i = 1; \$i ≤ \$n; \$i++)</code>	<code>\$i % 2</code>	<code>echo</code>
Perl	yorumlanan	<code>for my \$i (1..\$n)</code>	<code>\$i % 2</code>	<code>print</code>
MATLAB	yorumlanan/JIT	<code>for i = 1:n</code>	<code>mod(i, 2)</code>	<code>fprintf</code>

Derlenen, Yorumlanan, Hibrit

Ölçüt	Derlenen	Yorumlanan	Hibrit (bytecode + JIT)
Çeviri zamanı	çalışmadan önce, bir kez	her çalıştırmada	önce ara koda, sonra çalışırken
Hız	en yüksek	en düşük	yüksek (ısındıktan sonra)
Dağıtılan	makine kodu	kaynak kod	ara kod (.class, .dll)
Taşınabilirlik	her platform için yeniden derle	yorumlayıcı olan her yer	sanal makine olan her yer
Hata yakalama	çoğu derlemede	çoğu çalışırken	tip hataları derlemede
Örnek	C, C++, Go, Rust, Fortran, Ada	Python, Ruby, PHP, Perl, MATLAB	Java, C#, JavaScript (V8)

“Derlenen/yorumlanan” dilin değil **gerçeklemenin** özelliğidir: C için yorumlayıcı, Python için derleyici (Cython, PyPy JIT) de vardır.

Kod İzleme: Ara Kod (Bytecode) Nasıl Çalışır?

```
1 // x = (a + b) * c;
```

```
► iload a // a'yı yığına it
```

```
3 iload b // b'yi yığına it
```

```
4 iadd // üst iki öğeyi topla
```

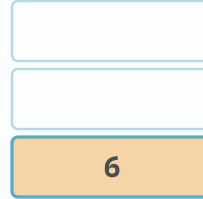
```
5 iload c
```

```
6 imul // üst iki öğeyi çarp
```

```
7 istore x // yığından x'e yaz
```

```
8 return
```

işlenen yığını



yerel değişkenler

a	6
b	7
c	3
x	?

Derleyici `x = (a + b) * c` için yığın tabanlı **ara kod** (bytecode) üretti. Yerel değişkenler: a = 6, b = 7, c = 3.

`iload a` : yerel değişken a = 6 yığının üstüne itilir.

Adım 1 / 7

DEĞİŞKENLER

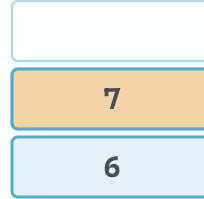
yığın = [6]

x = ?

Kod İzleme: Ara Kod (Bytecode) Nasıl Çalışır?

```
1 // x = (a + b) * c;  
2 iload a      // a'yı yığına it  
3 ► iload b    // b'yi yığına it  
4 iadd         // üst iki öğeyi topla  
5 iload c  
6 imul         // üst iki öğeyi çarp  
7 istore x     // yığından x'e yaz  
8 return
```

işlenen yığını



← üst

yerel değişkenler

a	6
b	7
c	3
x	?

`iload b` : yerel değişken b = 7 yığının üstüne itilir.

Adım 2 / 7

DEĞİŞKENLER

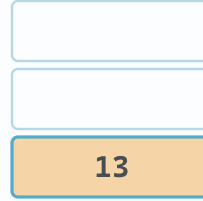
yığın = [6, 7]

x = ?

Kod İzleme: Ara Kod (Bytecode) Nasıl Çalışır?

```
1 // x = (a + b) * c;  
2 iload a      // a'yı yığına it  
3 iload b      // b'yi yığına it  
▶ iadd         // üst iki öğeyi topla  
5 iload c  
6 imul         // üst iki öğeyi çarp  
7 istore x     // yığından x'e yaz  
8 return
```

işlenen yığını



yerel değişkenler

a	6
b	7
c	3
x	?

iadd : üstteki iki öğe (6, 7) çekilir, toplam **13** itilir. İşlenen adı yok; hepsi yığında.

Adım 3 / 7

DEĞİŞKENLER

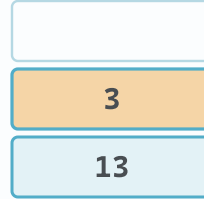
yığın = [13]

x = ?

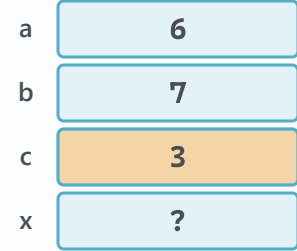
Kod İzleme: Ara Kod (Bytecode) Nasıl Çalışır?

```
1 // x = (a + b) * c;  
2 iload a      // a'yı yığına it  
3 iload b      // b'yi yığına it  
4 iadd        // üst iki öğeyi topla  
► iload c  
6 imul        // üst iki öğeyi çarp  
7 istore x     // yığından x'e yaz  
8 return
```

işlenen yığını



yerel değişkenler



iload c : yerel değişken c = 3 yığının üstüne itilir.

Adım 4 / 7

DEĞİŞKENLER

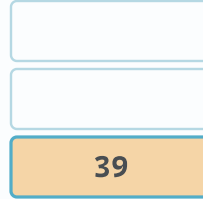
yığın = [13, 3]

x = ?

Kod İzleme: Ara Kod (Bytecode) Nasıl Çalışır?

```
1 // x = (a + b) * c;  
2 iload a      // a'yı yığına it  
3 iload b      // b'yi yığına it  
4 iadd         // üst iki öğeyi topla  
5 iload c  
▶ imul         // üst iki öğeyi çarp  
7 istore x     // yığından x'e yaz  
8 return
```

işlenen yığını



yerel değişkenler

a	6
b	7
c	3
x	?

imul : $13 \times 3 = 39$ yığına itilir.

Adım 5 / 7

DEĞİŞKENLER

yığın = [39]

x = ?

Kod İzleme: Ara Kod (Bytecode) Nasıl Çalışır?

```
1 // x = (a + b) * c;  
2 iload a      // a'yı yığına it  
3 iload b      // b'yi yığına it  
4 iadd         // üst iki öğeyi topla  
5 iload c  
6 imul         // üst iki öğeyi çarp  
► istore x     // yığından x'e yaz  
8 return
```

işlenen yığını

yerel değişkenler

a

6

b

7

c

3

x

39

istore x : yığının üstü x'e yazılır → x = **39**; yığın boşaldı.

Adım 6 / 7

DEĞİŞKENLER

yığın = []

x = 39

Kod İzleme: Ara Kod (Bytecode) Nasıl Çalışır?

```
1 // x = (a + b) * c;  
2 iload a      // a'yı yığına it  
3 iload b      // b'yi yığına it  
4 iadd         // üst iki öğeyi topla  
5 iload c  
6 imul         // üst iki öğeyi çarp  
7 istore x     // yığından x'e yaz  
► return
```

işlenen yığını

yerel değişkenler

a

6

b

7

c

3

x

39

Bu ara kod hiçbir işlemciye bağlı değil: sanal makine (JVM, CLR) her komutu ya yorumlar ya da JIT ile 8086/x86 gibi gerçek makine koduna çevirir.

Adım 7 / 7

DEĞİŞKENLER

yığın = []

x = 39

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2 ► mov cx, 1 ; i = 1
3 dongu: cmp cx, 3 ; i ile n karşılaştır
4 ja bitti ; i > n ise bitir
5 mov dl, cl
6 add dl, '0' ; rakam → ASCII
7 mov ah, 02h
8 int 21h ; DL'yi yaz
9 test cl, 1 ; en düşük bit
10 jnz tek ; 1 ise tek
11 lea dx, cift_m
12 jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15 int 21h ; dizgeyi yaz
16 inc cx ; i++
17 jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

CX sayaç (i) olarak kullanılır: i = 1. n = 3
sabit (okuma işlemi sonraki bölümlerde).

Adım 1 / 19

DEĞİŞKENLER

CX = 0001h

ÇIKTI



Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3     ▶ dongu: cmp cx, 3      ; i ile n karşılaştır
4     ▶     ja bitti      ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h         ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h         ; dizgeyi yaz
16     inc cx          ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

`cmp cx, 3` : 1 – 3 hesaplanır (sonuç saklanmaz) → ZF=0, CF=1 (ödünç).

`ja` yalnız CF=0 ve ZF=0 iken atlar → **atlamaz.**

Adım 2 / 19

DEĞİŞKENLER

CX = 0001h

ZF = 0

CF = 1

ÇIKTI



Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     ▶ mov dl, cl
6     ▶ add dl, '0'   ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15    int 21h        ; dizgeyi yaz
16    inc cx          ; i++
17    jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

DL = 1 + 30h = 31h: rakam 1'in ASCII kodu. ADD bayrakları günceller: CF=0.

Adım 3 / 19

DEĞİŞKENLER

CX = 0001h

DL = 31h ('1')

ZF = 0

CF = 0

ÇIKTI



Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja  bitti      ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     ▶     mov ah, 02h
8     ▶     int 21h    ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h        ; dizgeyi yaz
16     inc cx         ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

DOS 02h hizmeti DL'deki karakteri yazar.

Adım 4 / 19

DEĞİŞKENLER

CX = 0001h

DL = 31h ('1')

AH = 02h

ZF = 0

CF = 0

ÇIKTI

1

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     ► test cl, 1    ; en düşük bit
10    ► jnz tek       ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h        ; dizgeyi yaz
16     inc cx         ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

test cl, 1 : 1 AND 1 = 1 ≠ 0 →
ZF=0 → jnz tek atlar. (i % 2 yerine
en düşük bit.)

Adım 5 / 19

DEĞİŞKENLER

CX = 0001h

DL = 31h ('1')

AH = 02h

ZF = 0

CF = 0

ÇIKTI

1

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek        ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 ▶ tek: lea dx, tek_m
14 ▶ yaz: mov ah, 09h
15 ▶     int 21h      ; dizgeyi yaz
16    inc cx          ; i++
17    jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah','$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah','$'
```

DX = tek_m dizgisinin adresi; 09h hizmeti '\$'a kadar yazar. 0Dh, 0Ah satır başı + satır sonudur.

Adım 6 / 19

DEĞİŞKENLER

CX = 0001h

DX = offset tek_m

AH = 09h

ZF = 0

CF = 0

ÇIKTI

1 sayisi bir tek sayidir.

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek:  lea dx, tek_m
14 yaz:  mov ah, 09h
15    int 21h        ; dizgeyi yaz
16    ► inc cx        ; i++
17    ► jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

`inc cx` → 2 (INC, CF'yi değiştirmez);
döngü başına dönülür.

Adım 7 / 19

DEĞİŞKENLER

CX = 0002h

DX = offset tek_m

AH = 09h

ZF = 0

CF = 0

ÇIKTI

1 sayisi bir tek sayidir.

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3     ▶ dongu: cmp cx, 3      ; i ile n karşılaştır
4     ▶     ja bitti      ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h         ; DL'yi yaz
9     test cl, 1       ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h         ; dizgeyi yaz
16     inc cx          ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

2 – 3: CF=1 → devam.

Adım 8 / 19

DEĞİŞKENLER

CX = 0002h

DX = offset tek_m

AH = 09h

ZF = 0

CF = 1

ÇIKTI

1 sayisi bir tek sayidir.

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja  bitti      ; i > n ise bitir
5     ► mov dl, cl
6     ► add dl, '0'   ; rakam → ASCII
7     ► mov ah, 02h
8     ► int 21h       ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h        ; dizgeyi yaz
16     inc cx         ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

DL = 32h ('2') yazılır.

Adım 9 / 19

DEĞİŞKENLER

CX = 0002h

DL = 32h ('2')

AH = 02h

ZF = 0

CF = 0

ÇIKTI

1 sayisi bir tek sayidir.

2

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     ► test cl, 1    ; en düşük bit
10    ► jnz tek       ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h        ; dizgeyi yaz
16     inc cx         ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

2 AND 1 = 0 → ZF=1 → **jnz** **atlamaz**:
sayı çift.

Adım 10 / 19

DEĞİŞKENLER

CX = 0002h

DL = 32h ('2')

AH = 02h

ZF = 1

CF = 0

ÇIKTI

1 sayisi bir tek sayidir.

2

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek        ; 1 ise tek
11    ▶ lea dx, cift_m
12    ▶ jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h        ; dizgeyi yaz
16     inc cx         ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

DX = cift_m; **jmp yaz** ile ortak yazdırma koduna gidilir.

Adım 11 / 19

DEĞİŞKENLER

CX = 0002h

DX = offset cift_m

AH = 02h

ZF = 1

CF = 0

ÇIKTI

1 sayisi bir tek sayidir.

2

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek        ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek:  lea dx, tek_m
14    ► yaz: mov ah, 09h
15    ► int 21h      ; dizgeyi yaz
16    inc cx         ; i++
17    jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

“çift” iletisi yazılır.

Adım 12 / 19

DEĞİŞKENLER

CX = 0002h

DX = offset cift_m

AH = 09h

ZF = 1

CF = 0

ÇIKTI

1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek:  lea dx, tek_m
14 yaz:  mov ah, 09h
15    int 21h        ; dizgeyi yaz
16    ► inc cx       ; i++
17    ► jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

CX = 3.

Adım 13 / 19

DEĞİŞKENLER

CX = 0003h

DX = offset cift_m

AH = 09h

ZF = 0

CF = 0

ÇIKTI

```
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
```

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3     ▶ dongu: cmp cx, 3      ; i ile n karşılaştır
4     ▶     ja bitti      ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h         ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h         ; dizgeyi yaz
16     inc cx          ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

$3 - 3 = 0 \rightarrow ZF=1, CF=0$. $ZF=1$ olduğu için **ja** **atlamaz** ($3 \leq 3$).

Adım 14 / 19

DEĞİŞKENLER

CX = 0003h

DX = offset cift_m

AH = 09h

ZF = 1

CF = 0

ÇIKTI

1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     ▶ mov dl, cl
6     ▶ add dl, '0'    ; rakam → ASCII
7     ▶ mov ah, 02h
8     ▶ int 21h        ; DL'yi yaz
9     ▶ test cl, 1     ; en düşük bit
10    ▶ jnz tek        ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h        ; dizgeyi yaz
16     inc cx         ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

'3' yazılır; $3 \text{ AND } 1 = 1 \rightarrow \text{ZF}=0 \rightarrow \text{tek}$.

Adım 15 / 19

DEĞİŞKENLER

CX = 0003h

DL = 33h ('3')

AH = 02h

ZF = 0

CF = 0

ÇIKTI

```
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
3
```

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 ▶ tek: lea dx, tek_m
14 ▶ yaz: mov ah, 09h
15 ▶     int 21h      ; dizgeyi yaz
16    inc cx          ; i++
17    jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

“tek” iletisi yazılır.

Adım 16 / 19

DEĞİŞKENLER

CX = 0003h

DX = offset tek_m

AH = 09h

ZF = 0

CF = 0

ÇIKTI

```
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
3 sayisi bir tek sayidir.
```

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek:  lea dx, tek_m
14 yaz:  mov ah, 09h
15    int 21h        ; dizgeyi yaz
16    ► inc cx        ; i++
17    ► jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

CX = 4.

Adım 17 / 19

DEĞİŞKENLER

CX = 0004h

DX = offset tek_m

AH = 09h

ZF = 0

CF = 0

ÇIKTI

```
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
3 sayisi bir tek sayidir.
```

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3     ▶ dongu: cmp cx, 3      ; i ile n karşılaştır
4     ▶     ja bitti      ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h         ; DL'yi yaz
9     test cl, 1       ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek: lea dx, tek_m
14 yaz: mov ah, 09h
15     int 21h         ; dizgeyi yaz
16     inc cx          ; i++
17     jmp dongu
18 bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

$4 - 3 = 1 \rightarrow ZF=0, CF=0 \rightarrow$ **ja bitti**
atlar ($4 > 3$).

Adım 18 / 19

DEĞİŞKENLER

CX = 0004h

DX = offset tek_m

AH = 09h

ZF = 0

CF = 0

ÇIKTI

```
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
3 sayisi bir tek sayidir.
```

Aynı Görev 8086 Assembly ile (n = 3)

```
1 org 100h
2     mov cx, 1      ; i = 1
3 dongu: cmp cx, 3    ; i ile n karşılaştır
4     ja bitti       ; i > n ise bitir
5     mov dl, cl
6     add dl, '0'     ; rakam → ASCII
7     mov ah, 02h
8     int 21h        ; DL'yi yaz
9     test cl, 1      ; en düşük bit
10    jnz tek         ; 1 ise tek
11    lea dx, cift_m
12    jmp yaz
13 tek:  lea dx, tek_m
14 yaz:  mov ah, 09h
15    int 21h        ; dizgeyi yaz
16    inc cx          ; i++
17    jmp dongu
► bitti: ret
19 tek_m db ' sayisi bir tek sayidir.',0Dh,0Ah,'$'
20 cift_m db ' sayisi bir cift sayidir.',0Dh,0Ah,'$'
```

ret : COM programı DOS'a döner.
C'deki **for** + **if**, burada
karşılaştırma + **koşullu atlama**
çiftleriyle kuruldu.

Adım 19 / 19

DEĞİŞKENLER

CX = 0004h

DX = offset tek_m

AH = 09h

ZF = 0

CF = 0

ÇIKTI

```
1 sayisi bir tek sayidir.
2 sayisi bir cift sayidir.
3 sayisi bir tek sayidir.
```

ÖZET

Özet

- Yazılım donanımı yönetir; işletim sistemi temel yazılımdır, programlama dilleri yazılım geliştirmek için kullanılır.
- **Makine dili** işlemcinin tek anladığı dildir: işlem kodu ve işlenen alanlarından oluşan ikili komutlar; donanıma tamamen bağımlıdır.
- **Assembly dili** makine komutlarını anımsatıcılarla yazar; assembler ve bağlayıcı ile çalıştırılabilir dosyaya dönüşür. Kod hem işlemciye hem işletim sistemine bağlıdır (int 0x80, int 21h, DLL çağrısı, svc, sc).
- **Derlenen diller** (C, C++, Go, Rust, Fortran, Ada) önceden makine koduna çevrilir: hızlı, ama platforma özgü.
- **Yorumlanan diller** (Python, JavaScript, Ruby, PHP, Perl, MATLAB) çalışırken işlenir: hızlı geliştirme, taşınabilir kaynak, daha yavaş çalışma. Java/C# ara kod + JIT ile ikisinin arasındadır.
- Yüksek seviyedeki `for` / `if`, assembly'de **karşılaştırma + koşullu atlama** olarak gerçekleşir.

ÖZET

Alıştırmalar

1. Örnek makine dilinde `SUB R3, R0` ve `OR R0, R2` komutlarını ikili ve onaltılık olarak kodlayın.
2. `0010 0111` komutunu çözün. $R1 = 9$, $R3 = 4$ ise sonuç nedir?
3. Bu tasarımda kaç farklı işlem kodu tanımlanabilir? Yazmaç sayısını 8'e çıkarmak için komut biçimi nasıl değişmeli?
4. DOS programında dizgi sonundaki `'$'` unutulursa ne olur?
5. Linux örneğinde `len` elle 15 yazılısaydı ekrana ne çıkardı?
6. Windows örneğinde parametreler neden ters sırada itilir? `call` 'dan hemen sonra ESP kaçır?
7. Java neden "derlenen" ya da "yorumlanan" diye tek kelimeyle sınıflanamaz?

► İpuçları / cevaplar

ÖZET

Kaynaklar

- Tanenbaum ve Bos, *Modern Operating Systems*, 3. baskı, Pearson.
- Brey, *The Intel Microprocessors*, 8. baskı, Pearson.
- Irvine, *Assembly Language for x86 Processors*, 7. baskı, Pearson.
- NASM belgeleri (nasm.us) ve GNU Binutils *as* kılavuzu.
- Ralf Brown's Interrupt List (DOS INT 21h hizmetleri); Microsoft Win32 API belgeleri (`MessageBoxA` , `ExitProcess`).
- emu8086 belgeleri ve 8086 komut kümesi başvurusu.

Son

Bölüm 1: Programlama Dilleri

Sonraki bölüm: Bölüm 2: 8086 Pinler · [Terim sözlüğü](#)