Bilgisayar Mimarisi

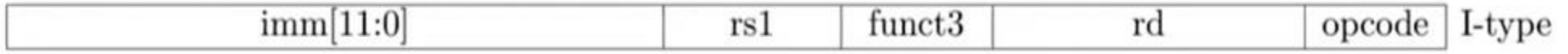
2023-2024 Bahar Dönemi

Hafta-5

Dr. Öğr. Üyesi Nurullah ÖZTÜRK

Anlık Değerler

Programlarda sabit değerler sıklıkla kullanılır. Sabit değerler kullanılacakları zaman **bellekten** `load` buyrukları ile yüklenebilir ya da **anlık değer** olarak buyruklar ile iletilebilir.



`int i = 500;`

`addi`

500	x0	funct3	x1	opcode

Hedef Yazmacı: xxxx xxxx xxxx xxxx xxxx | 0001 1111 0100

31 12 11

Anlık Değerler

Buyrukların Kodlanması

Donanımda buyruklar elektrik sinyalleri ile iletilir.

- Sayısal devrelerdeki iki farklı gerilim değeri:
  - Toprak (0)
  - Kaynak (1)

Basitçe, bir buyruk sayı olarak gösterilen küçük parçalardan oluşur.

Alan: Buyruğun bilgi içeren bir parçası

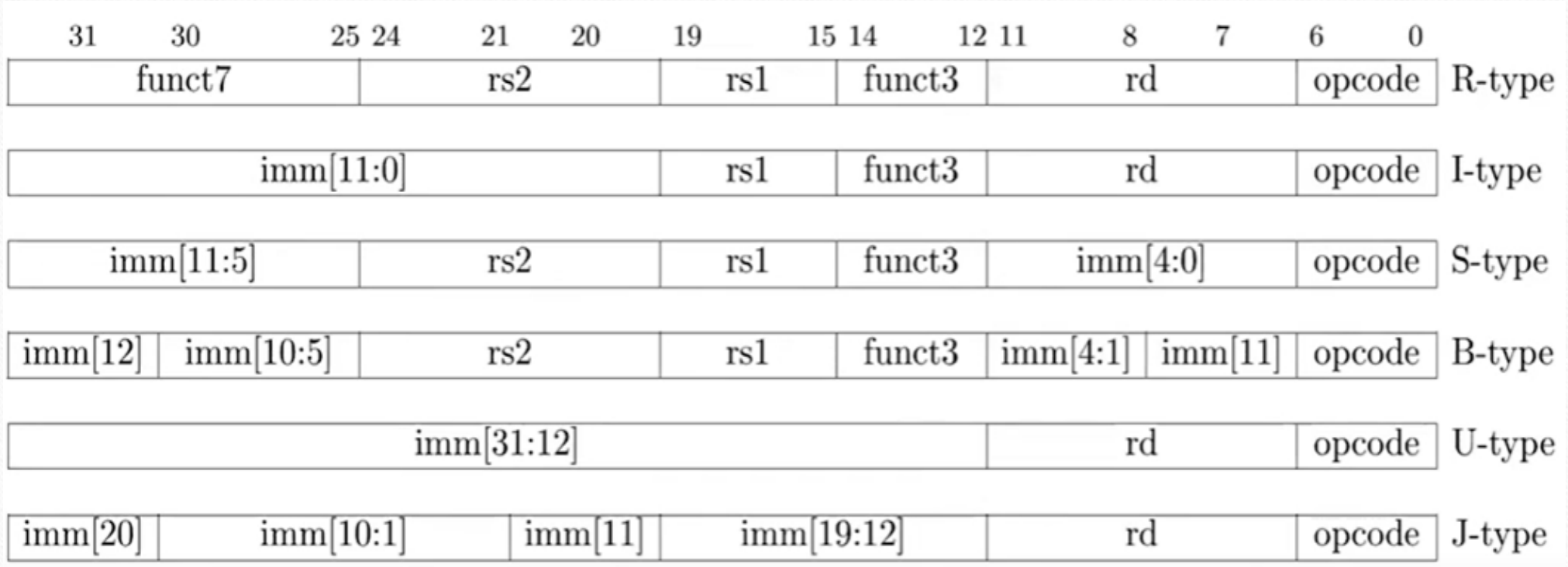
`add x9, x20, x21` →

Onluk Taban:	0	21	20	0	9	51
İkilik Taban:	0000000	10101	10100	000	01001	0110011
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit

Toplama (add) Kaynak Yazmacı (x21) Hedef Yazmacı (x9)

Kaynak Yazmacı (x20)

Tüm RISC-V Buyruk Tipleri



RISC-V R-Type Buyruk Kodlaması

iş7	ky2	ky1	iş3	hy	işkodu
7 bit	5 bit	5 bit	3 bit	5 bit	7 bit

RISC-V Buyruk Alanları

işkodu: İşlemi ve buyruğun formatını belirten alan.

hy: Hedef yazmacı.

iş3: Fazladan işkodu alanı.

ky1: Birinci kaynak yazmacı.

ky2: İkinci kaynak yazmacı.

iş7: Fazladan işkodu alanı.

İngilizce:	funct7	rs2	rs1	funct3	rd	opcode

RISC-V R-Type Buyruk Kodlaması

iş7	ky2	ky1	iş3	hy	işkodu
7 bit	5 bit	5 bit	3 bit	5 bit	7 bit

Farklı aritmetik buyrukların **işkodu alanı aynı iş3 ve iş7 alanları farklıdır**:

Buyruk	Format	iş7	ky2	ky1	iş3	hy	işkodu
topla	R-tipi	**0000000**	yazmaç	yazmaç	000	yazmaç	0110011
çıkar	R-tipi	**0100000**	yazmaç	yazmaç	000	yazmaç	0110011

RISC-V R-Type Buyruk Kodlaması

Bu kodlama her buyruk için uygun mu?

LD buyruğunu nasıl kodlayabiliriz?

iş7	anlık	ky1	iş3	hy	işkodu
7 bit	**5 bit**	5 bit	3 bit	5 bit	7 bit

```
ld x1, 17(x2)        // x1 = bellek[x2+17]
ld x1, 743(x2)       // x1 = bellek[x2+743]
```

5-bit alan büyük anlık değerleri kodlamak için yeterli değil.

- Buyruk boyutlarının **sabit kalması** için **farklı buyruk formatlarına** ihtiyacımız var.

RISC-V I-Type Buyruk Kodlaması

Yeni buyruk tipi: I-tipi (*immediate*). Anlık değer, kaynak ve hedef yazmacı.

- Anlıkla topla (`addi`) ve yükle (`ld`) buyrukları.

anlık	ky1	iş3	hy	işkodu
12 bit	5 bit	3 bit	5 bit	7 bit

12-bit 2'ye tümleyen değeri.

- Negatif ve pozitif sayılar.

RISC-V S-Type Buyruk Kodlaması

Kaydet (`sd`) buyruğu için de yeni bir kodlamaya ihtiyacımız var.

- İki kaynak yazmacı, bir anlık değer.

S-tipi buyruk formatı:

anlık [11:5]	ky2	ky1	iş3	anlık [4:0]	işkodu
7 bit	5 bit	5 bit	3 bit	5 bit	7 bit

Kaynak yazmaçlarının tüm formatlarda aynı yerde bulunması için S-tipi buyruklarda **anlık iki parçaya** ayrılmıştır.

- **Basit donanım.**

RISC-V Buyruk Kodlaması

Her format özel bir işkodu değerine sahiptir.

Donanım işkodu değerine bakarak buyruk formatını anlayabilir.

Şu ana kadar gördüğümüz buyrukların formatları ve buyruk alanları:

Buyruk	Format	iş7	ky2	ky1	iş3	hy	işkodu
topla	R-tipi	0000000	yazmaç	yazmaç	000	yazmaç	0110011
çıkar	R-tipi	0100000	yazmaç	yazmaç	000	yazmaç	0110011

Buyruk	Format	Anlık	ky1	iş3	hy	işkodu
anlıkla topla	I-tipi	sabit değer	yazmaç	000	Yazmaç	0010011
yükle	I-tipi	sabit değer	yazmaç	011	Yazmaç	0000011

Buyruk	Format	anlık	ky2	ky1	iş3	anlık	işkodu
kaydet	S-tipi	adres	yazmaç	yazmaç	011	adres	0100011

C Kodu RISC-V Makine Kodu

```
A[30] = h + A[30] + 1;
```

Buyruk	Format	iş7	ky2	ky1	iş3	hy	işkodu
topla	R-tipi	0000000	yazmaç	yazmaç	000	yazmaç	0110011
çıkar	R-tipi	0100000	yazmaç	yazmaç	000	yazmaç	0110011

Buyruk	Format	Anlık	ky1	iş3	hy	işkodu
a-topla	I-tipi	sabit değer	yazmaç	000	Yazmaç	0010011
yükle	I-tipi	sabit değer	yazmaç	011	Yazmaç	0000011

Buyruk	Format	anlık	ky2	ky1	iş3	anlık	işkodu
kaydet	S-tipi	adres	yazmaç	yazmaç	011	adres	0100011

Derleyici Değişken Eşleştirmesi

A'nın başlangıç adresi → x10

h → x21

C Kodu RISC-V Makine Kodu

```
A[30] = h + A[30] + 1;
```

Buyruk	Format	iş7	ky2	ky1	iş3	hy	işkodu
topla	R-tipi	0000000	yazmaç	yazmaç	000	yazmaç	0110011
çıkar	R-tipi	0100000	yazmaç	yazmaç	000	yazmaç	0110011

Buyruk	Format	Anlık	ky1	iş3	hy	işkodu
a-topla	I-tipi	sabit değer	yazmaç	000	Yazmaç	0010011
yükle	I-tipi	sabit değer	yazmaç	011	Yazmaç	0000011

Buyruk	Format	anlık	ky2	ky1	iş3	anlık	işkodu
kaydet	S-tipi	adres	yazmaç	yazmaç	011	adres	0100011

Derleyici Değişken Eşleştirmesi

A'nın başlangıç adresi → x10

h → x21

C Kodu RISC-V Makine Kodu

```
A[30] = h + A[30] + 1;
```

Derleyici Değişken Eşleştirmesi
A'nın başlangıç adresi → x10
h → x21

```
ld   x9, 240(x10)
add  x9, x21, x9
addi x9, x9, 1
sd   x9, 240(x10)
```

(I) anlık (R) iş7 (S) anlık[11:5]	(R) ky2	ky1	iş3	hy (S) anlık[4:0]	işkodu
240		10	3	9	3
0	9	21	0	9	51
1		9	0	9	19
7	9	10	3	16	35

000011110000		01010	011	01001	0000011
0000000	01001	10101	000	01001	0110011
000000000001		01001	000	01001	0010011
0000111	01001	01010	011	10000	0100011

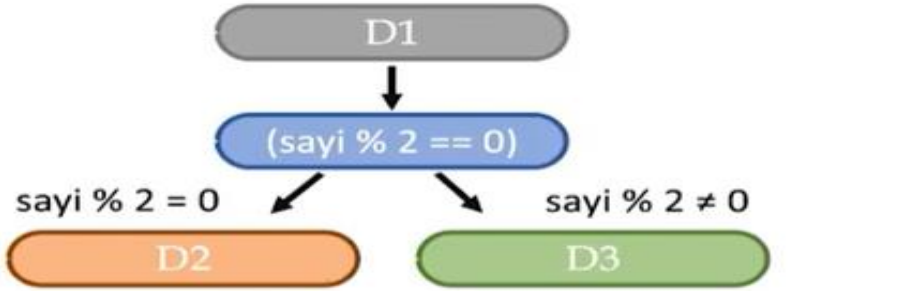
Mantıksal İşlem Buyrukları

Mantıksal İşlemler	C / Java Dili Karşılığı	RISC-V buyrukları
Sola kaydır	<<	sll, slli
Sağa kaydır	>>	srl, srli
Aritmetik sağa kaydır	>>	sra, srai
VE (bit düzeyinde)	&	and, andi
VEYA (bit düzeyinde)	|	or, ori
DIŞLAYAN VEYA (bit düzeyinde)	^	xor, xori
DEĞİL (bit düzeyinde)	~	xori

Dallanma Buyrukları

```
void tekMiCiftMi(int sayi)
{
   if (sayi % 2 == 0)          } B1
      // çifte zıpla
      goto even;
   else
      // teke zıpla
      goto odd;

odd:                           } B2
   printf("%d sayisi tek", sayi);
even:                          } B3
   printf("%d sayisi cift", cift);
   // cift ise don
   return;
}
```



Koşullu Dallanma

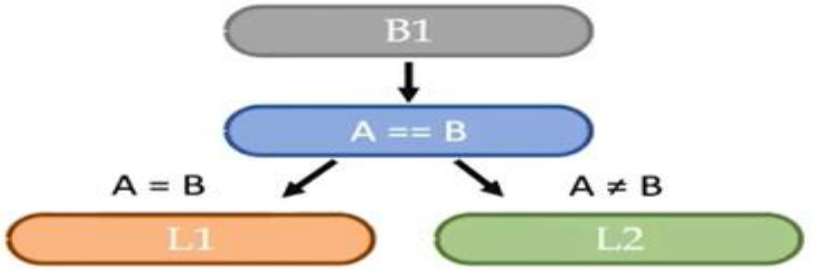
Koşulsuz Atlama

Koşullu Dallanma Buyrukları

eşit ise atla (Branch EQuals)

```
beq rs1, rs2, L1
bne rs1, rs2, L2
```

eşit değilse atla (Branch Not Equals)



Döngü:

```
for (int i = 0; i < 20; i++) {
   ...
}
for (int i = 30; i >= 20; i--) {
   ...
}
```

eşit veya büyükse atla

```
bge rs1, rs2, L1
blt rs1, rs2, L2
```

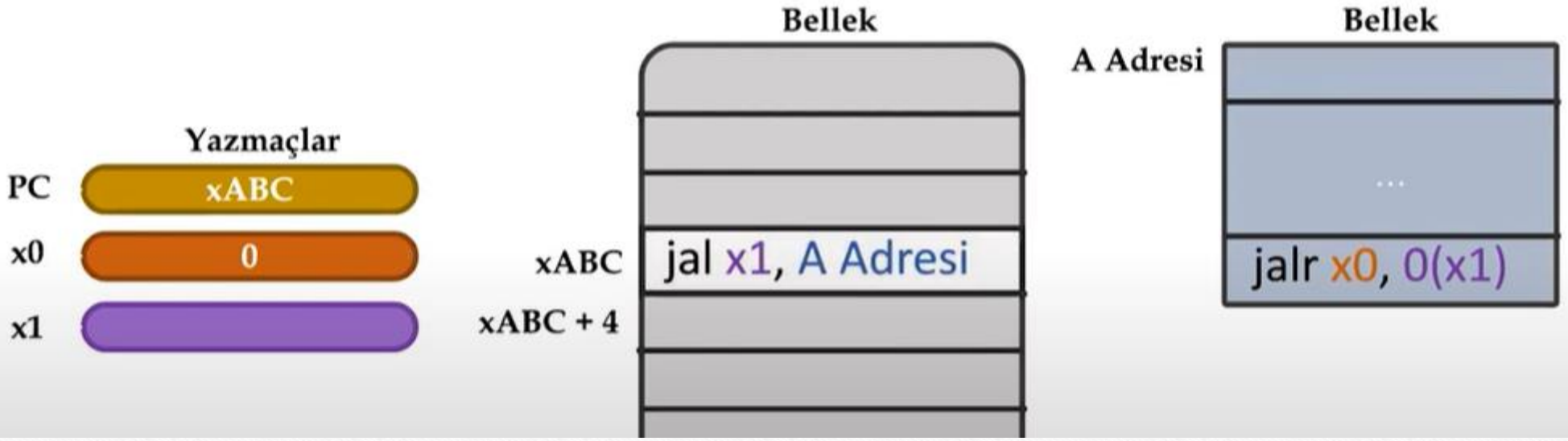
küçükse atla

jal x1, A Adresi

Atla ve kaydet *(jump and link)* buyruğu, A adresine atlar ve dönüş adresini **x1**'e kaydeder.

jalr x0, 0(x1)

Yazmaca atla ve kaydet *(jump and link register)* buyruğu, x1 yazmacındaki adrese atlar ve dönüş adresini **x0**'a kaydeder.



İşlevler

Kodu kolayca **tekrar kullanabilmek** ve kod yazarken **tek bir hedefe** odaklanmak için programcılar **işlev (-ing. function)** adı verilen yapılardan faydalanırlar.

```
int çağıran_işlev()
{
  int a = 5;
  int b = 8;

  çağrılan_işlev(96);

  return a + b;
}
```

→

```
addi x8, x0, #5          // int a = 5
addi x9, x0, #8          // int b = 8

addi x10, x0, #96        // çağırılan_işlev argümanı
jal x1, çağrılan_işlev   // cağırılan_işlev(96)

add x10, x8, x9          // return a + b
```

x10-x17 arasındaki yazmaçlar işlevlere **argüman** olarak verilen değerleri ve işlemlerin **döndüğü değerleri** barındırır.
x1 islevin **döneceği adresi** barındırır.

Program Yığıtı

```
int çağıran_işlev()
{
  int a = 5;
  int b = 8;

  çağrılan_işlev(96);

  return a + b;
}
```

→

```
addi x8, x0, #5          // int a = 5
addi x9, x0, #8          // int b = 8

addi x10, x0, #96        // çağırılan_işlev argümanı
jal x1, cağrılan_işlev   // cağırılan_işlev(96)

add x10, x8, x9
```

Çağrılan işlev x8 ve x9 yazmaçlarını kullanırsa (üzerine yazarsa) ne olur?

Program yanlış çalışır.

→İşlevlere ait **veri** program **yığıtında (-ing. stack)** saklanır.

→Yığıt: Öğelerden **son gelenin ilk işlem görecek biçimde** üst üste yığıldığı varsayılan **veri yapısı**.

Program Yığıtı

Yığıt bellekte tutulur. Yığıtın bellekteki adresi **yığıt işaretçisi** (-ing. stack pointer) ile belirtilir.

→RISC-V'de yığıt işaretçisi **x2 yazmacında** tutulur.

Push: Yığıtın en üstüne veri eklemek.

Pop: Yığıttan en üstteki veriyi çıkarmak.

→Yığıt "yukarıdan aşağıya" doğru genişler.

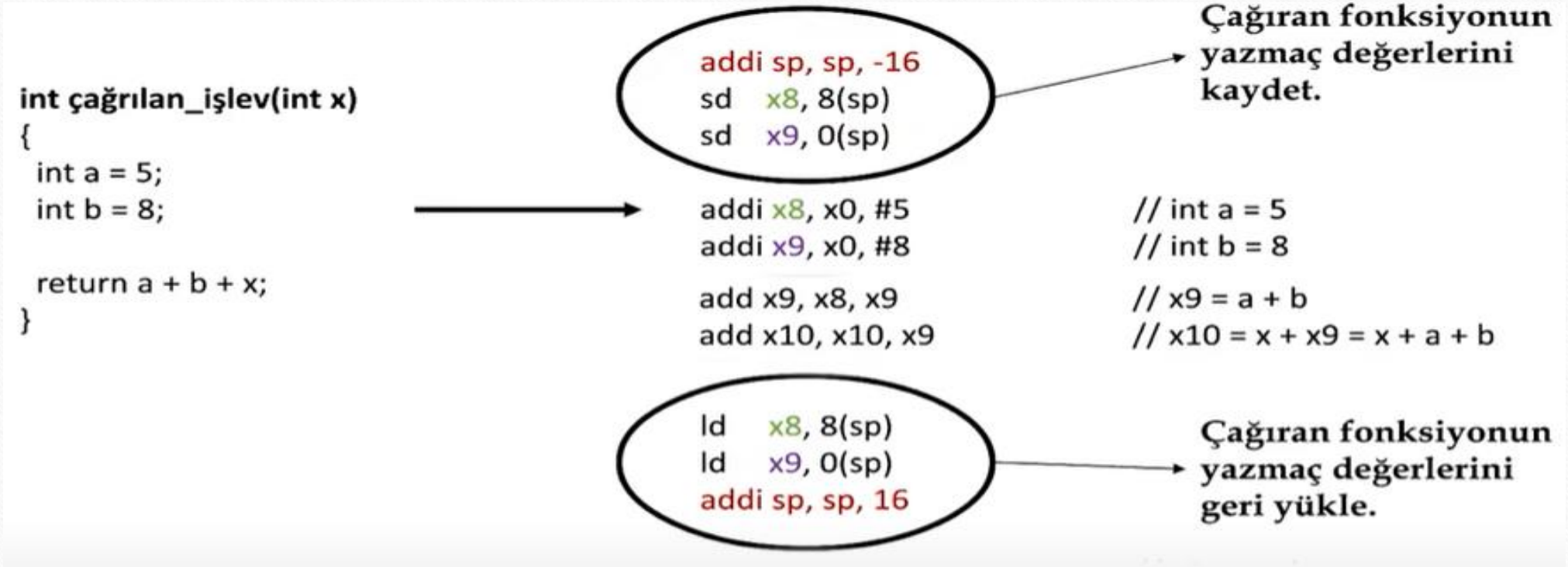
→ Push: Yığıt işaretçisini **azaltır**.

→ Pop: Yığıt işaretçisini **artırır**.

RISC-V'de **push** ve **pop** buyrukları yoktur.

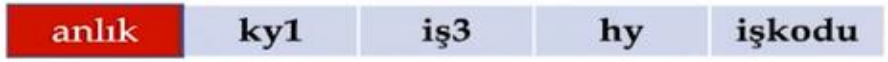
→ Kaydet ve yükle buyrukları kullanılır.

Program Yığıtı



RISC-V Adresleme Kipleri

Anlık adresleme (Immediate addressing)



→ İşlenen buyruğun içinde.

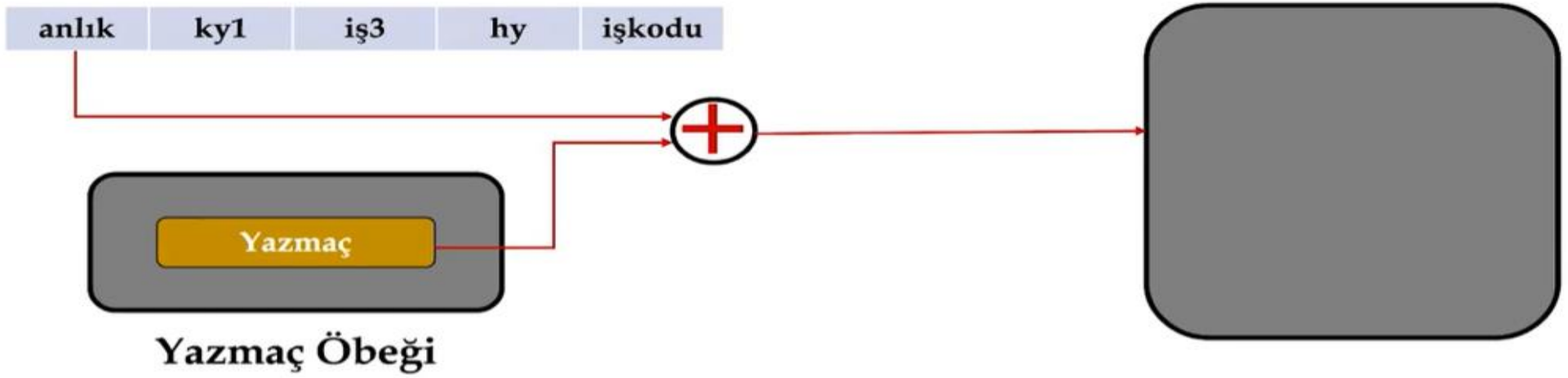
Yazmaç adresleme (Register addressing)

Yazmaç Öbeği

→ İşlenen yazmaç öbeğinde.

RISC-V Adresleme Kipleri

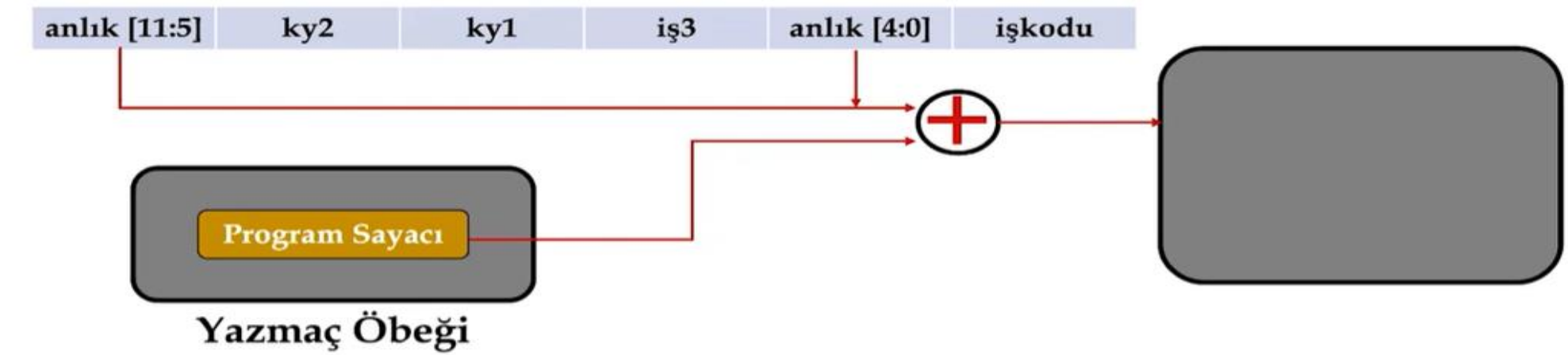
Eklemeli adresleme (Base addressing)



→ İşlenen bellekte ve bir yazmaca ve anlık değerin **eklenmesi** ile adresleniyor.

RISC-V Adresleme Kipleri

Göreceli Adresleme (PC-relative addressing)



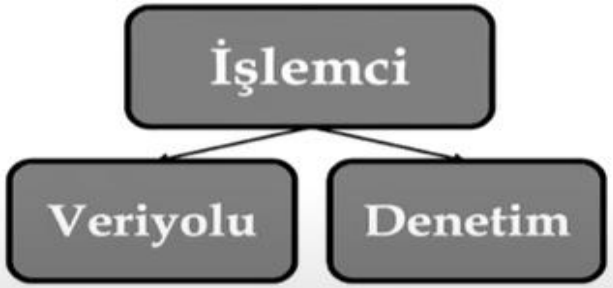
→ **Dallanma adresi** program sayacı ve anlık değerin toplamı olarak hesaplanır.

İŞLEMCİ

İŞLEMCİ

İşlemci bir bilgisayarın **en temel** parçasıdır.

- Aritmetik işlemler,
- Bellekteki veri aktarımları ve
- Giriş çıkış verilerinin kaydedilmesi işlemci tarafından gerçekleştirilir.



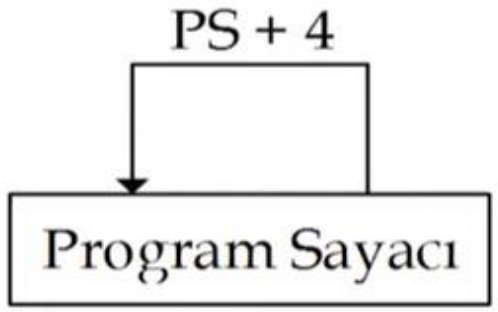
İşlemci Tasarımı Aşamaları

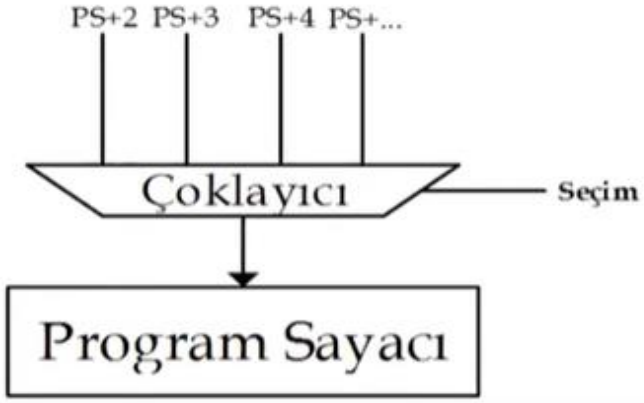
1. **Buyruk Kümesi Mimarisi Belirlenmesi**
2. BKM Gereksinimleri Belirlenmesi
3. Veriyolu Oluşturulması
   - Gereksinimleri karşılayan veri yolu bileşenlerinin belirlenmesi
   - Veri yolu bileşenlerinin birleştirilmesi
4. Denetim Birimlerinin Oluşturulması
   - Denetim noktalarının belirlenmesi
   - Denetim işaretlerini sağlayacak birimin tasarlanması

Buyruk Kümesi Mimarisi

Buyruk kümesi mimarisi işlemci tasarım kararlarını etkiler. Örneğin:

- Değişken boyutta buyruklar **getir (-ing. fetch)** ve **çöz (-ing. decode)** aşamalarının tasarımını etkiler.





- Mimarideki yazmaç sayısı işlemcideki **yazmaç öbeğinin (-ing. register file)** boyutunu belirler.

Buyruk Kümesi Mimarisi

Tipi	Buyruk	İşlenenler	Açıklama
Aritmetik	add	hy, ky1, ky2	**ky1** ve **ky2**'deki değerleri toplayıp **hy** yazmacına yazar.
	sub	hy, ky1, ky2	**ky1** yazmacındaki değerden **ky2** yazmacındaki değeri çıkarır ve **hy** yazmacına yazar.
	addi	hy, ky1, $anlık_{12}$	**ky1** ve **anlık** değeri toplayıp **hy** yazmacına yazar.
	subi	hy, ky1, $anlık_{12}$	**ky1** yazmacındaki değerden **anlık** değeri çıkarıp **hy** yazmacına yazar.
Mantık	and	hy, ky1, ky2	**ky1** ve **ky2**'deki değerlerin **mantıksal ve** sonucunu **hy** yazmacına yazar.
	xor	hy, ky1, ky2	**ky1** ve **ky2**'deki değerlerin **mantıksal dışlayan veya** sonucunu **hy** yazmacına yazar.
Bellek	lw	hy, $anlık_{12}$(ky1)	Bellek'te (**ky1** + **anlık**) ile işaretlenen sözcüğü **hy** yazmacına yazar.
	sw	ky2, $anlık_{12}$(ky1)	Bellek'te (**ky1** + **anlık**) ile işaretlenen sözcüğe **ky** yazmacının değerini yazar.
Dallanma	beq	ky1, ky2, $anlık_{12}$	**ky1** ve **ky2** eşit ise (PS + **anlık**) adresine zıplar, değilse progra sayacını dört artırır.
	jal	hy, $anlık_{20}$	(PS + 4) değerini **hy** yazmacına yazar sonra (PS + **anlık**) a

BKM Gereksinimlerinin Belirlenmesi

Tasarıma başlanmadan önce cevaplanması gereken sorular:

- Yazmaç öbeğinde kaç yazmaç var?
  - RV32-I: 32 yazmaç.
- Yazmaçlar kaç bit genişliğinde?
  - RV**32**-I: 32-bit.
- Aynı anda kaç yazmaç okunabiliyor?
  - RV32-I: En fazla **iki** yazmaç işleneni var (Örn. add rt, ky1, ky2)
- Hangi işlemler desteklenmeli?
  - Aritmetik: Topla (add, addi, **ld**, **st**) , çıkar (sub, subi).
  - Mantık: VE (and), VEYA (or), DIŞLAYAN VEYA (xor).
  - Karşılaştırma: Eşittir (beq).

Adres hesaplaması

Veriyolu Oluşturma

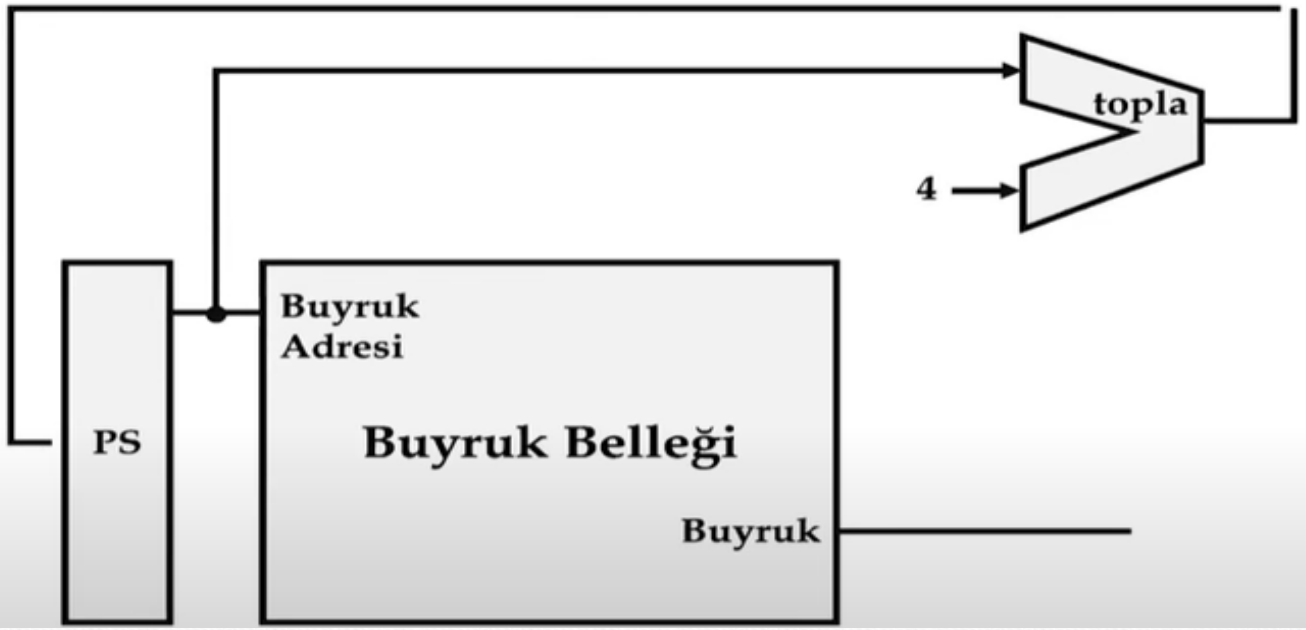
İşlemcide veri saklamak veya veri üzerinde işlem yapmak için kullanılan birimlere **veriyolu birimleri** adı verilir. İşlemci **veriyolunu** bu birimler oluşturur.

RV32-I işlemcisinin veriyolu birimleri:

- Program Sayacı ve Buyruk Belleği
- Yazmaç Öbeği
- Aritmetik mantık birimi
- Anlık genişletme birimi
- Veri Belleği

Program Sayacı ve Buyruk Belleği

Buyruk belleği program buyruklarını tutar ve verilen buyruk adresine karşılık gelen buyruğu çıktı olarak verir.



Program sayacı (PS) ise o anki buyruk adresini tutar. Buyruklar **32-bit sabit genişlikte** olduğundan ve **bayt adresleme** kullanıldığı için **program sayacı** her saat vuruşunda 4 artar.