



# BİLGİSAYAR MİMARİSİ

2021-2022

4. HAFTA

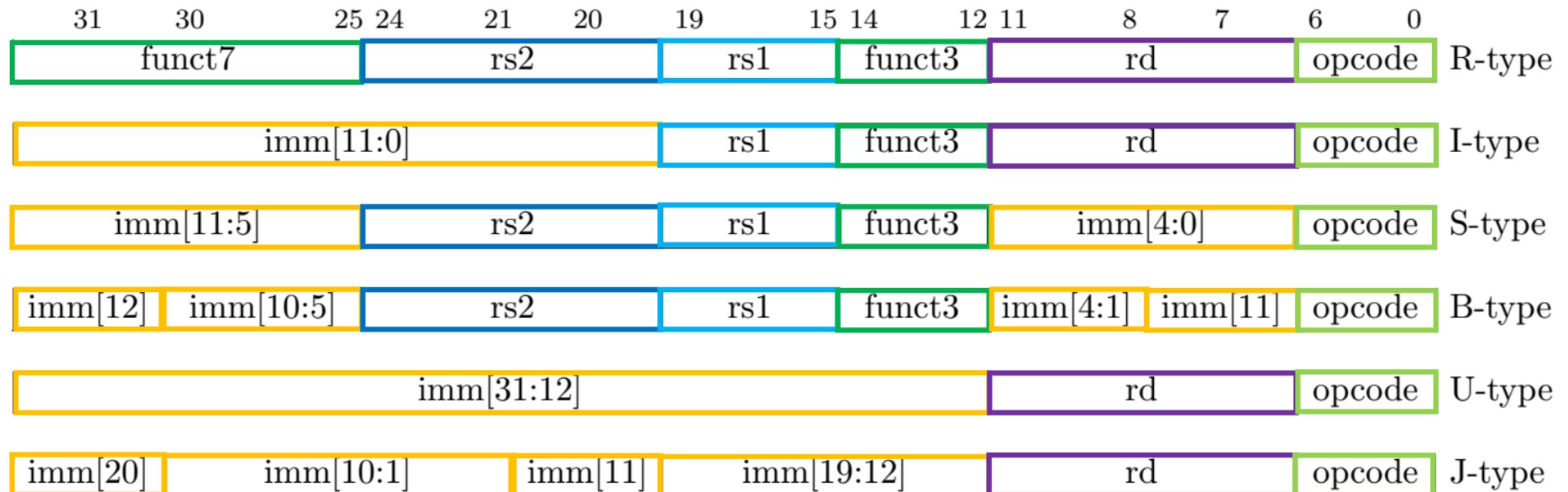
Dr. Öğr. Üyesi Ümit ŞENTÜRK

# Okuma Listesi

## Gerekli

- Computer Organization and Design: The Hardware Software Interface [RISC-V Edition] David A. Patterson, John L. Hennessy
  - 3. Bölüm sonuna kadar
- TOBB üniversitesi, Prof Dr. Oğuz ERGİN, Bilgisayar Mimarisi ve Organizasyonu dersi ders sunumları

# RISC-V Buyruk Türleri



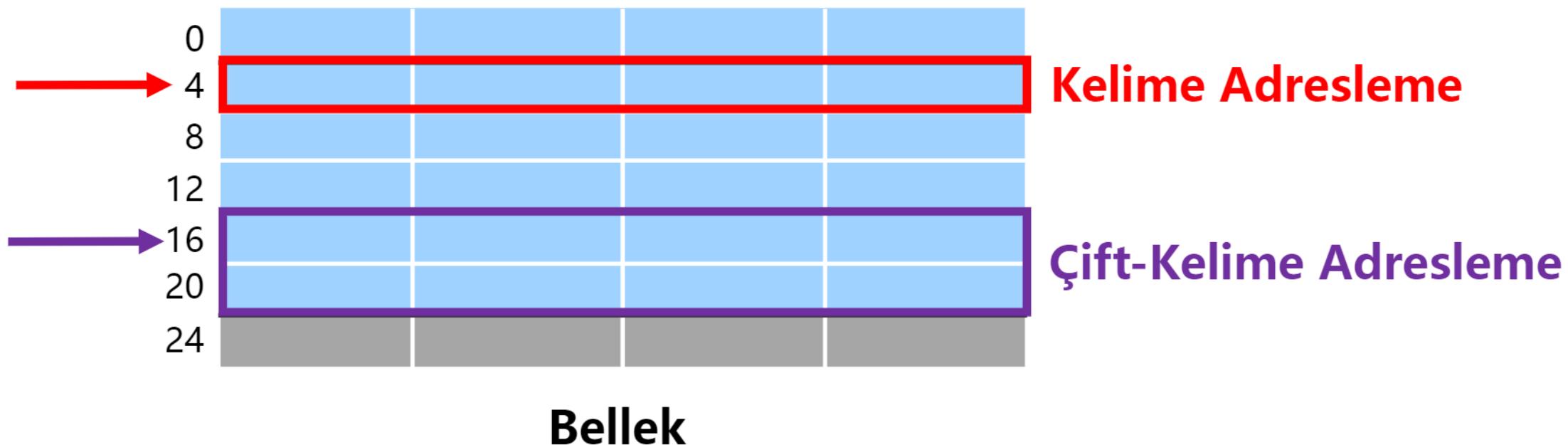
## Bellek Adreslerinin Hizalanması

Başka mimarilerde (MIPS) de olduğu gibi RISC-V **veri transfer buyruklarının** adreslerinde bazı **hizalama** kısıtları vardır.

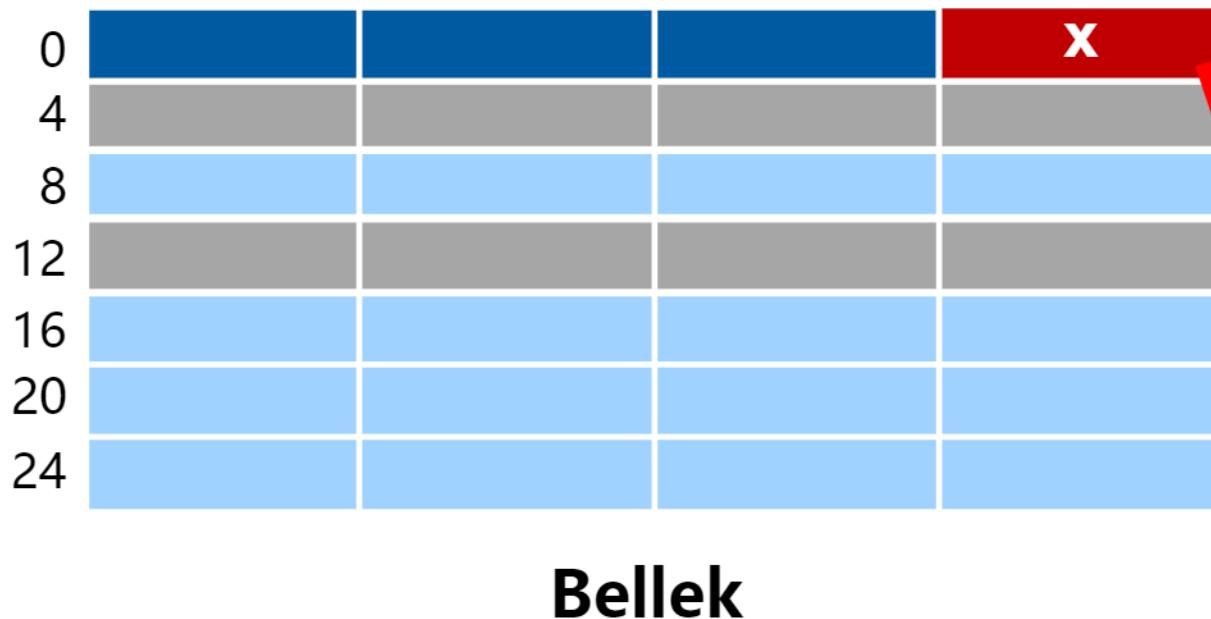
- Çift-kelime (doubleword) erişimlerinin adresleri **8'** in tam katı olmalıdır.
- Kelime (word) adresleri **4'** ün tam katı olmalıdır.

Bu kısıtlar **donanımın basit kalmasını** sağlar.

# Bellek Adreslerinin Hizalanması



# Bellek Adreslerinin Hizalanması

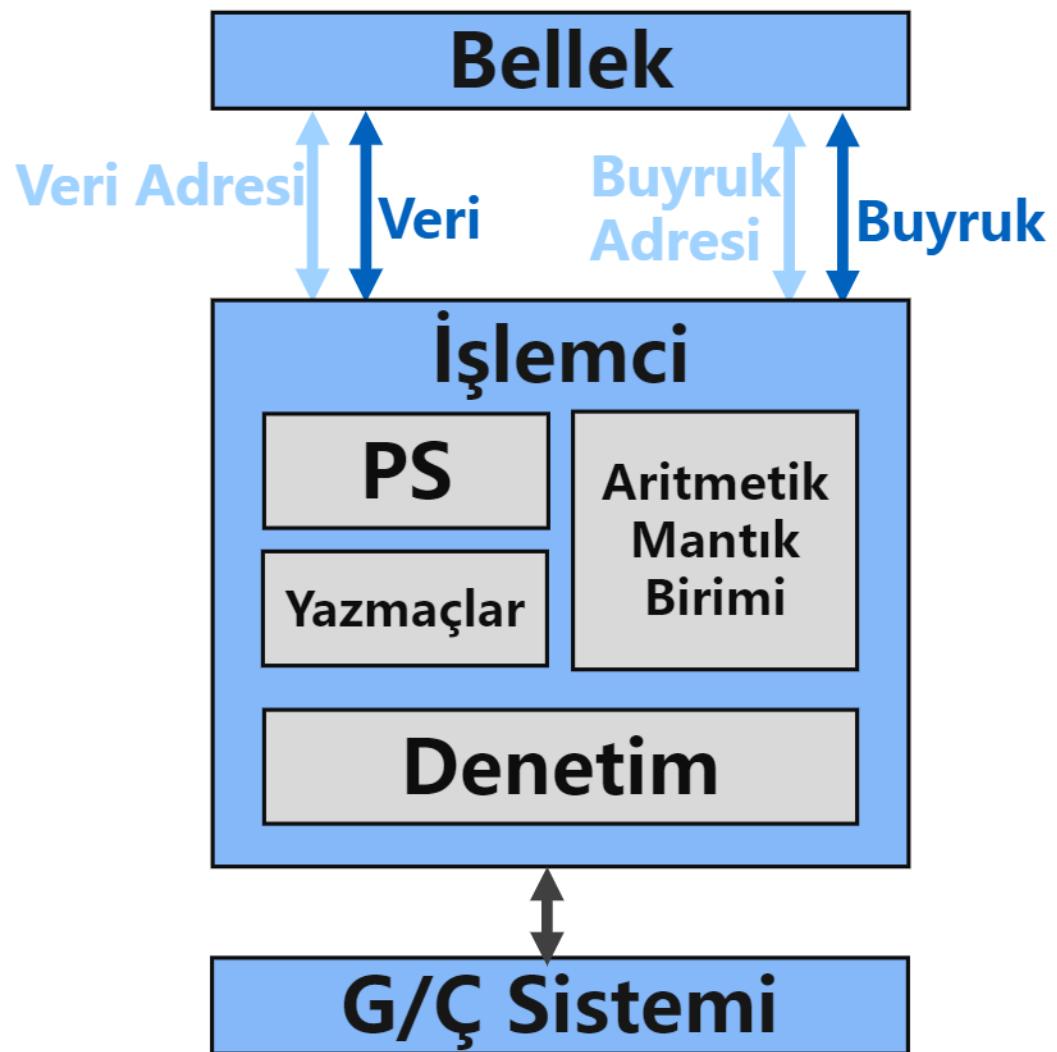
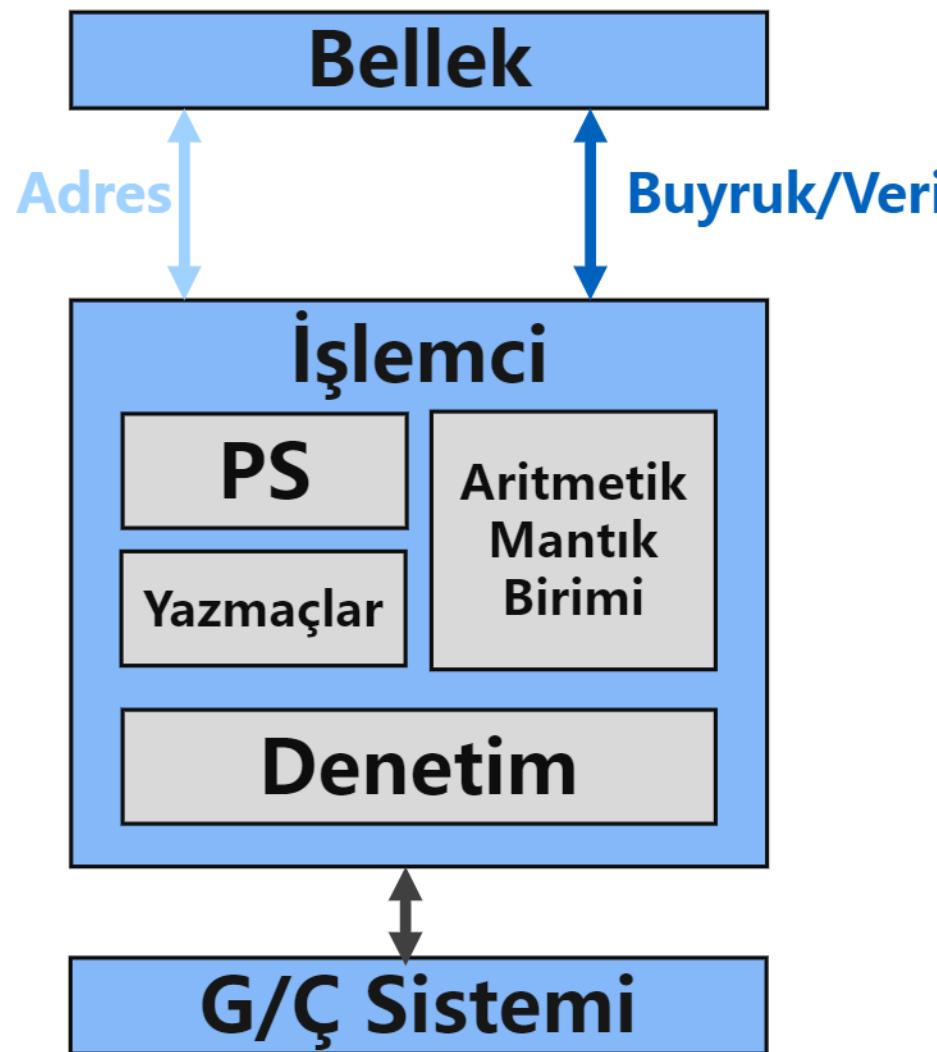


**Hizalanmamış Buyruk / Veri**

**Hizalanmış Buyruk / Veri**

**Kullanılmayan Bayt**

# Harvard ve Von Neumann Mimarileri



# Harvard ve Von Neumann Mimarileri

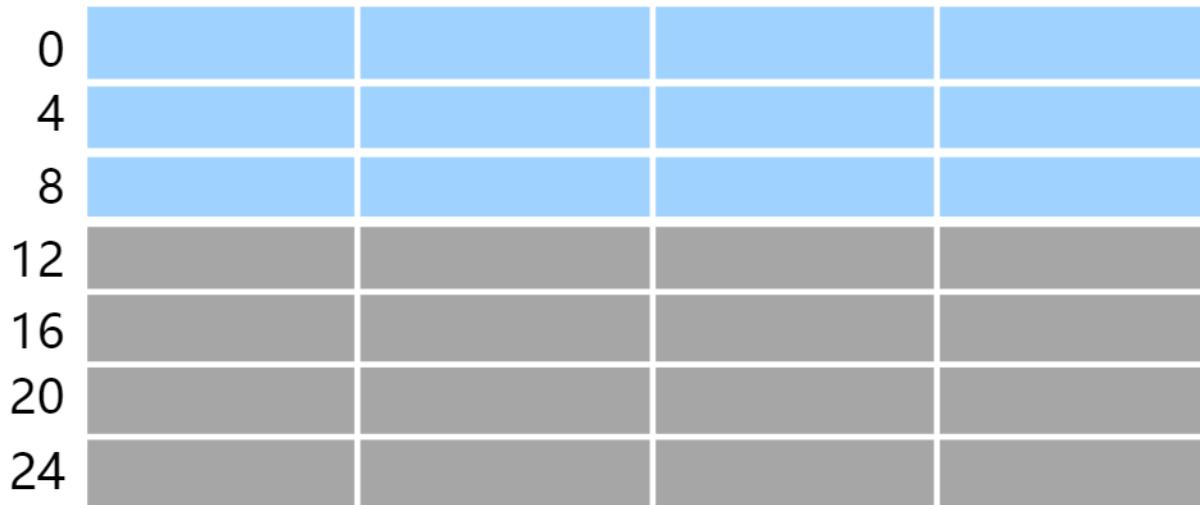
- Aynı fiziksel bellek uzayı hem buyruklar hem de veri için kullanılır.
- Buyruklar ve veriler aynı hattı kullanır.
- İşlemci buyruklara ve veriye aynı anda erişemez.
- Bir buyruğun tamamlanması iki çevrim sürer. İlk çevrimde buyruk, ikinci çevrimde veriye erişilir.
- Denetim biriminin maliyeti daha düşüktür.

**Von Neumann  
Mimarisi**

- Veri ve buyruklar farklı fiziksel bellek uzaylarında kullanır.
- Veri hattı ve buyruk hattı ayrıdır.
- İşlemci buyruklara ve veriye aynı anda erişebilir.
- Bir buyruğun tamamlanması bir çevrim sürer.
- Denetim biriminin maliyeti daha yüksektir.

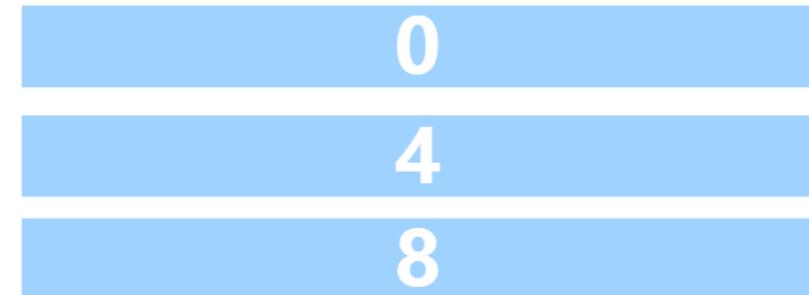
**Harvard  
Mimarisi**

# Sabit Boyutlu Buyruk Genişliği



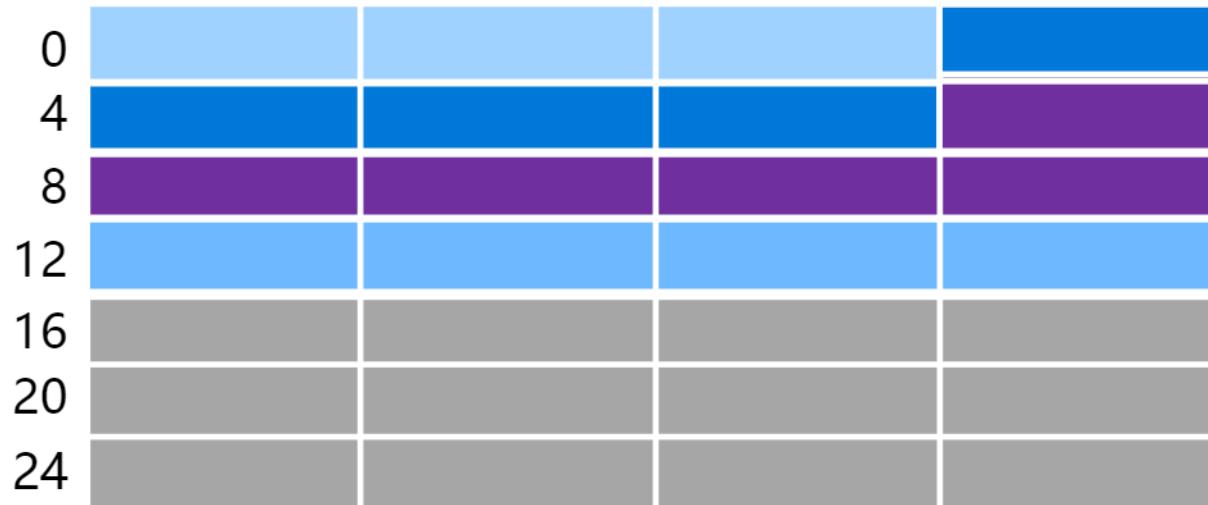
**Buyruk Belleği**

**Program Sayacı**



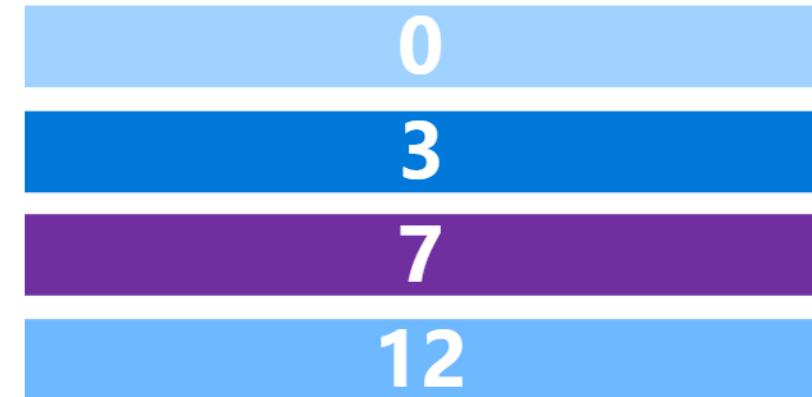
**Her buyruk aynı uzunlukta olduğu için PS her zaman aynı miktarda artar.**

# Değişken Boyutlu Buyruk Genişliği



**Buyruk Belleği**

**Program Sayacı +3+4+5**

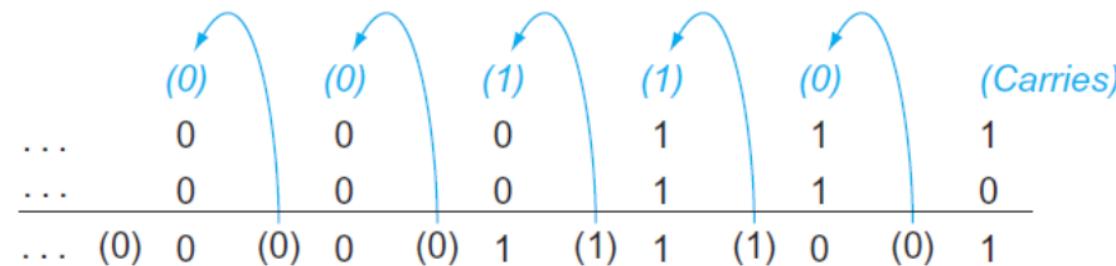


**Buyruklar farklı uzunluklarda olduğu için PS her zaman aynı miktarda artmaz.**

# Toplama ve Çıkarma

## Toplama

$$\begin{array}{r} 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0111_{\text{two}} = 7_{\text{ten}} \\ + \quad 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0110_{\text{two}} = 6_{\text{ten}} \\ \hline = \quad 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 1101_{\text{two}} = 13_{\text{ten}} \end{array}$$



# Çıkarma , Komplementi ile Toplama

$$\begin{array}{r} 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0111_{\text{two}} = 7_{\text{ten}} \\ - 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0110_{\text{two}} = 6_{\text{ten}} \\ \hline = 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0001_{\text{two}} = 1_{\text{ten}} \end{array}$$

$$\begin{array}{r} 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0111_{\text{two}} = 7_{\text{ten}} \\ + 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1111 \ 1010_{\text{two}} = -6_{\text{ten}} \\ \hline = 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0000 \ 0001_{\text{two}} = 1_{\text{ten}} \end{array}$$

# Kural Dışı Durum

$$-10 + 4 = -6 \quad ?$$

32 bitlik bir işlemcide 32 bitlik iki sayıyı topladığımızda toplam 33 bit ifade çıkıyorsa ne olacak?

İşlem 33. bite taşı işaret bitinin durumu ne olur? Yapılan toplama işlemi sonucu doğru mudur?

# Kural Dışı Durum

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	$\geq 0$	$\geq 0$	$< 0$
$A + B$	$< 0$	$< 0$	$\geq 0$
$A - B$	$\geq 0$	$< 0$	$< 0$
$A - B$	$< 0$	$\geq 0$	$\geq 0$



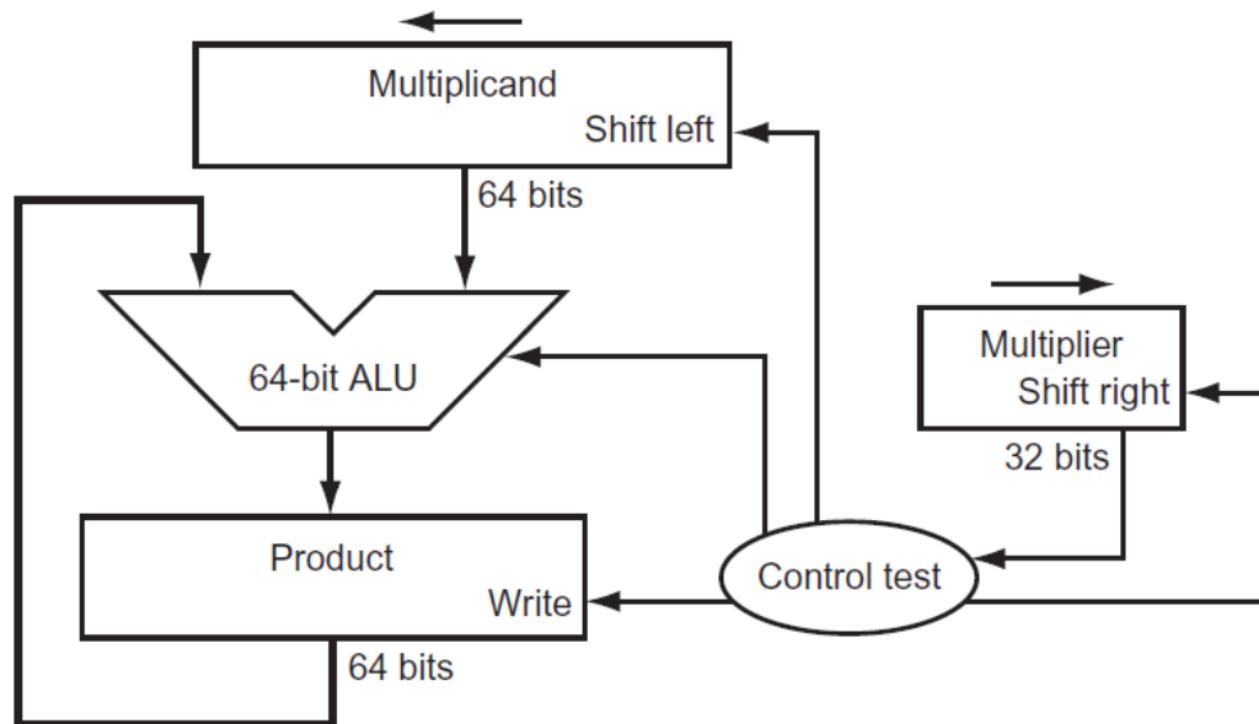
# Çarpma

Multiplicand	x	1000 <sub>ten</sub>
Multiplier	x	1001 <sub>ten</sub>
		1000
		0000
		0000
		1000
Product		1001000 <sub>ten</sub>

N ve m bit uzunluğunda iki sayı çarpılıyorsa sonuç m+n bit ile bütün muhtemel sonuçları temsil eder.

Toplamada olduğu gibi çarpmada da taşıma problemi ile karşı karşıyayız.

# Çarpma Algoritması



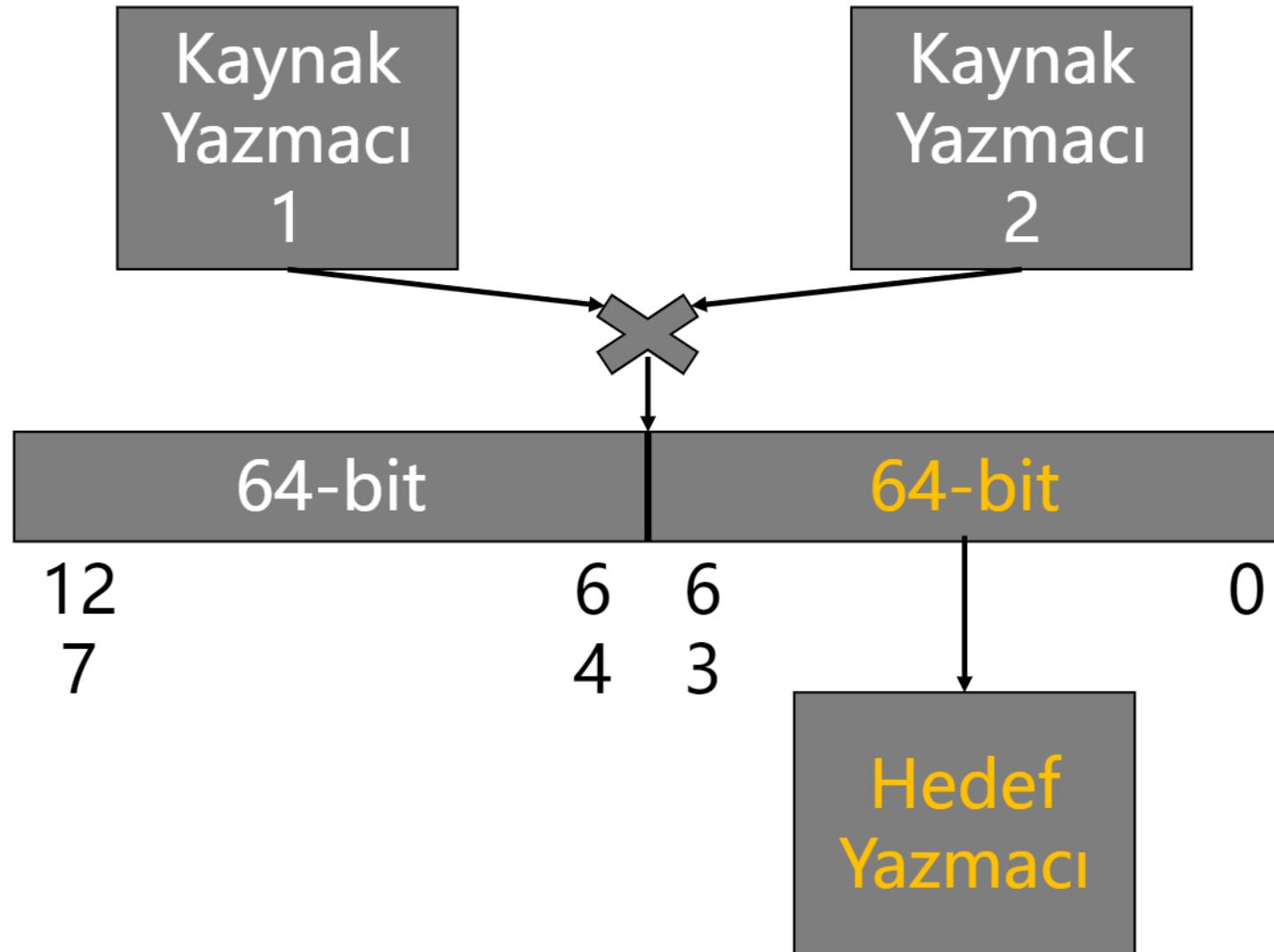
# Çarpma Örnek

$$0010_{\text{two}} \times 0011_{\text{two}}.$$

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 $\Rightarrow$ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 $\Rightarrow$ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 $\Rightarrow$ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 $\Rightarrow$ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

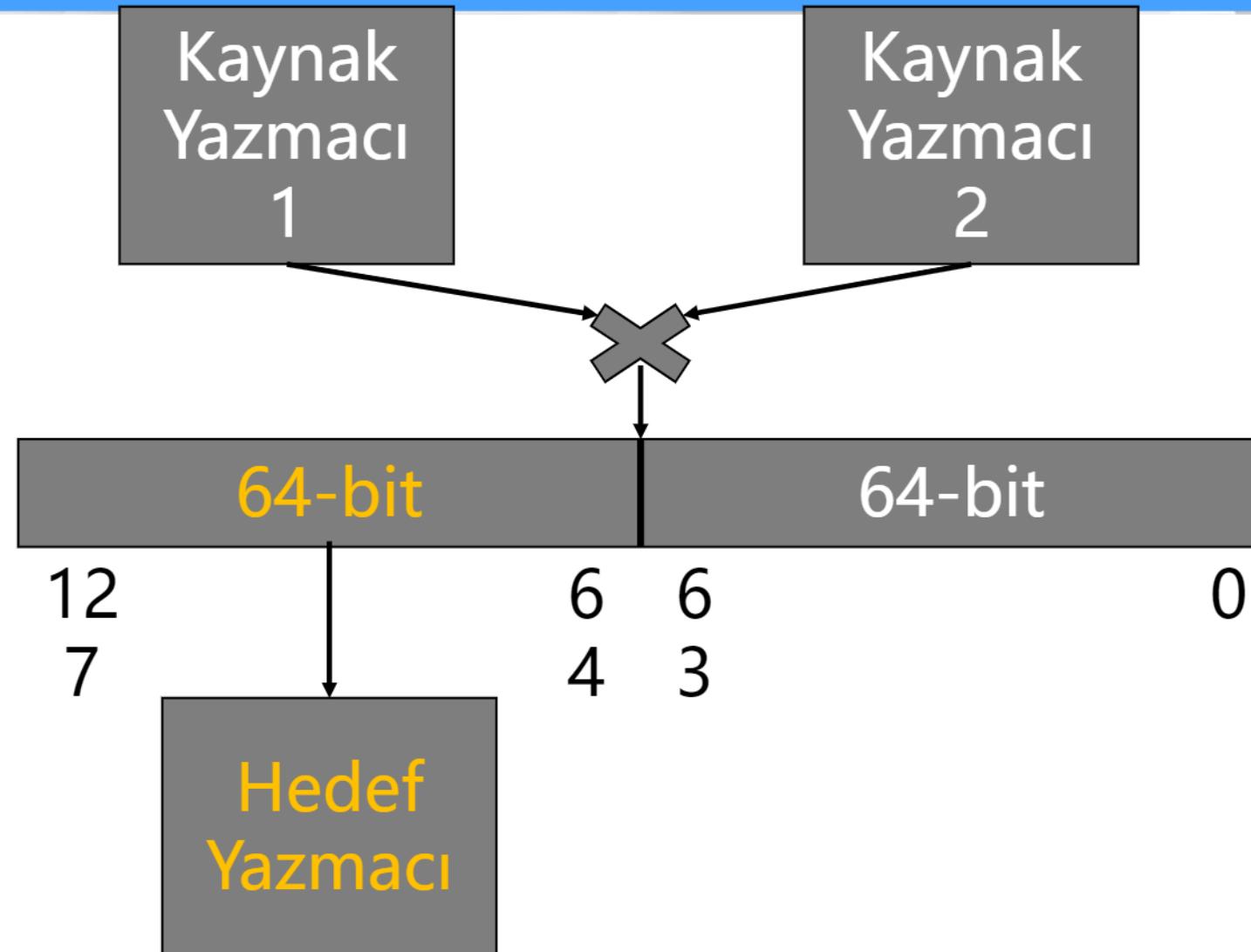
**MUL**

**Çarpım  
Sonucu**



**MULH**

**Çarpım  
Sonucu**



$1,001,010_{\text{ten}}$  by  $1000_{\text{ten}}$

Divisor	$1000_{\text{ten}}$	Quotient
	$\overline{1001010}_{\text{ten}}$	Dividend
	$-1000$	
	<hr/>	
	10	
	101	
	1010	
	$-1000$	
	<hr/>	
	$10_{\text{ten}}$	Remainder



$0000\ 0111_{\text{two}}$  by  $0010_{\text{two}}$

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	①110 0111
	2b: Rem < 0 $\Rightarrow$ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	①111 0111
	2b: Rem < 0 $\Rightarrow$ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	①111 1111
	2b: Rem < 0 $\Rightarrow$ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	②000 0011
	2a: Rem $\geq$ 0 $\Rightarrow$ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	②000 0001
	2a: Rem $\geq$ 0 $\Rightarrow$ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

div x5, x6, x7 -----→  $x5 = x6 / x7$  bölüm

rem x5, x6, x7-----→  $x5 = x6 \% x7$  bölümünden kalan

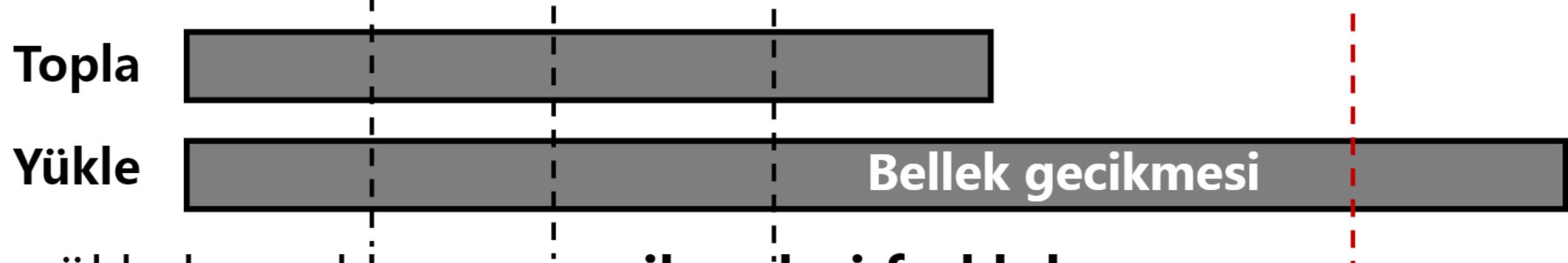
# İşlemci Tasarımı



Buyrukların çalışma döngüsü:

Topla Buyruğu: Getir → Çöz → Yürüt → Yaz

Yükle Buyruğu: Getir → Çöz → Yürüt → **Bellek** → Yaz



Topla ve yükle buyruklarının **gecikmeleri farklıdır.**

$$\text{Yürütme Zamanı} = \text{Buyruk Sayısı} \times \text{BBC} \times \text{Çevrim Zamanı}$$

# İşlemci Tasarımı Aşamaları

1. Buyruk Kümesi Mimarisi Belirlenmesi
2. BKM Gereksinimleri Belirlenmesi
3. Veriyolu Oluşturulması
4. Denetim Birimlerinin Oluşturulması

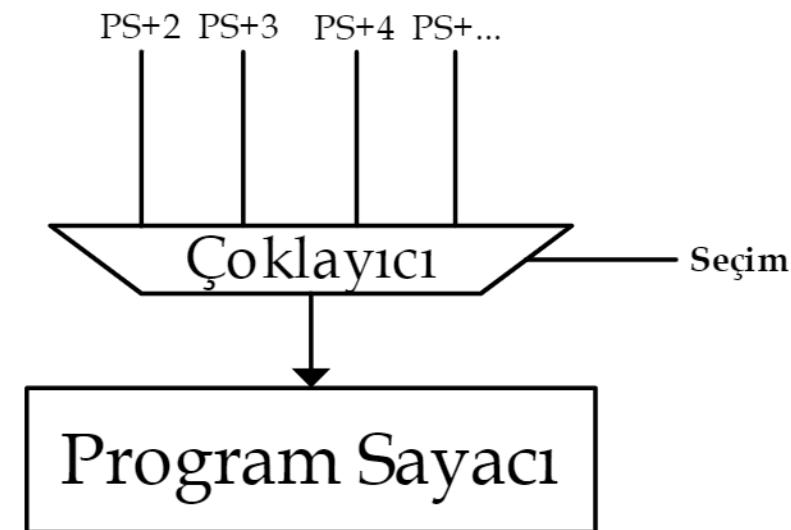
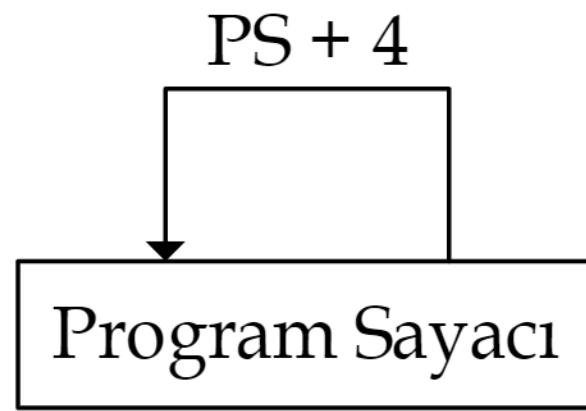


- 1. Buyruk Kümesi Mimarisi Belirlenmesi**
2. BKM Gereksinimleri Belirlenmesi
3. Veriyolu Oluşturulması
  - Gereksinimleri karşılayan veri yolu bileşenlerinin belirlenmesi
  - Veri yolu bileşenlerinin birleştirilmesi
4. Denetim Birimlerinin Oluşturulması
  - Denetim noktalarının belirlenmesi
  - Denetim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

# Buyruk Kümesi Mimarisi Belirlenmesi

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 Belirlenmesi

**İşlemciler seçilen buyruk kümelerindeki tüm buyrukları yürütebilmelidir.**

Tasarlayacağımız işlemcinin buyruk kümeleri (RV32-I' nın alt kümeleri):

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

1. Buyruk Kümesi Mimarisi Belirlenmesi → RISC-V
- 2. BKM Gereksinimleri Belirlenmesi**
3. Veriyolu Oluşturulması
4. Denetim Birimlerinin Oluşturulması



# BKM Gereksinimlerinin Belirlenmesi

Tasarıma başlanmadan önce cevaplanması gereken sorular:

- Yazmaç öbeğinde kaç yazmaç var?
  - RV32-I: 32 yazmaç.
- Yazmaçlar kaç bit genişliğinde?
  - RV32-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), DIŞLAYAN VEYA (xor).
  - Karşılaştırma: Eşittir (beq).



# İşlemci Tasarımı Aşamaları

1. Buyruk Kümesi Mimarisi Belirlenmesi → RISC-V
2. BKM Gereksinimleri Belirlenmesi
- 3. Veriyolu Oluşturulması**
4. Denetim Birimlerinin Oluşturulması



# Veriyolu Oluşturulması

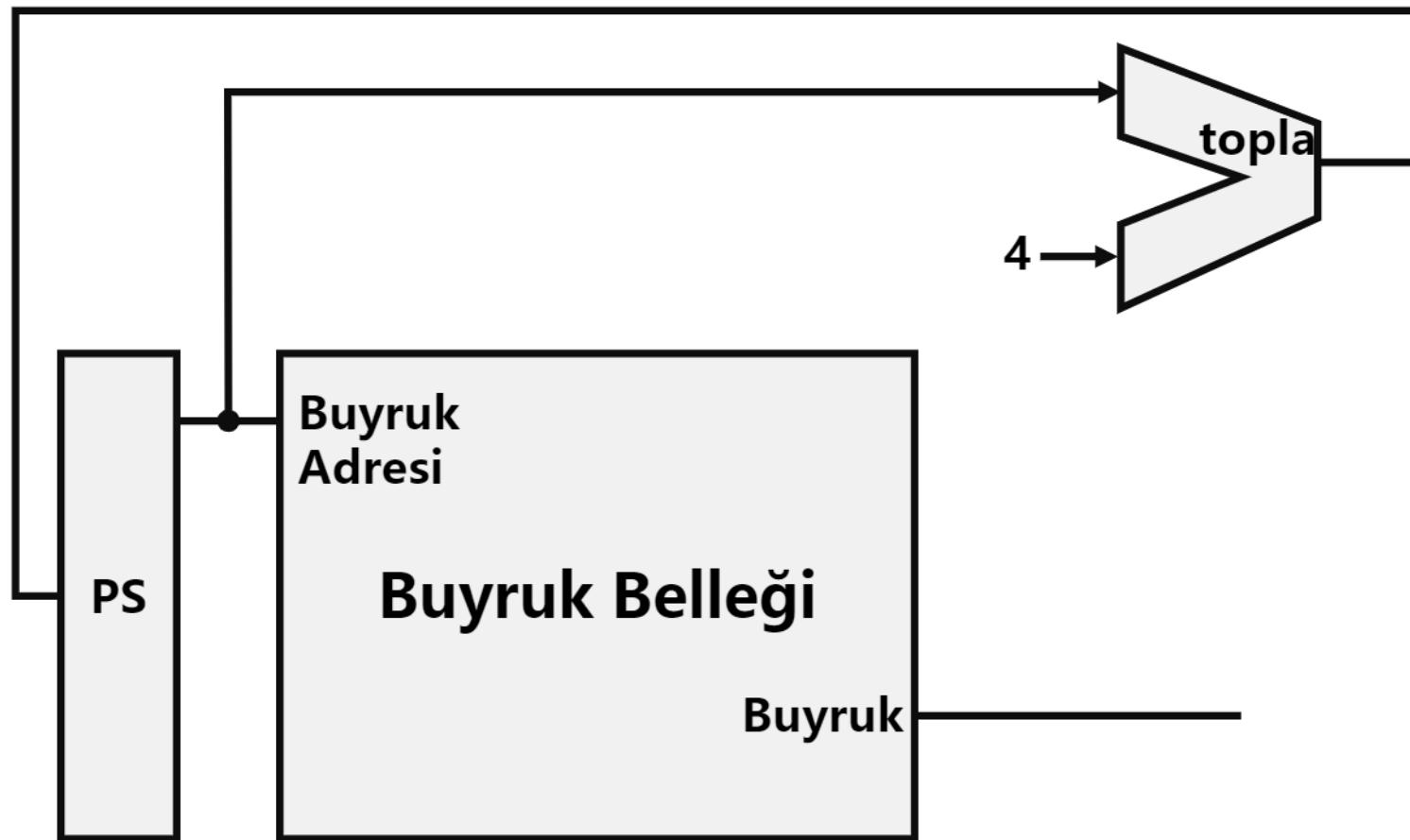
İşlemcide veri saklamak veya veri üzerinde işlem yapmak için kullanılan birimlere **veriyolu birimleri** adı verilir. İşlemci **veriyolunu** bu birimler oluşturur.

RV32-I işlemcisinin veriyolu birimleri:

- Program Sayacı ve Buyruk Belleği
- Yazmaç Öbeği
- Aritmetik mantık birimi
- Anlık genişletme birimi
- Veri Belleği
- Veri Seçici (Multiplexer)

# 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.

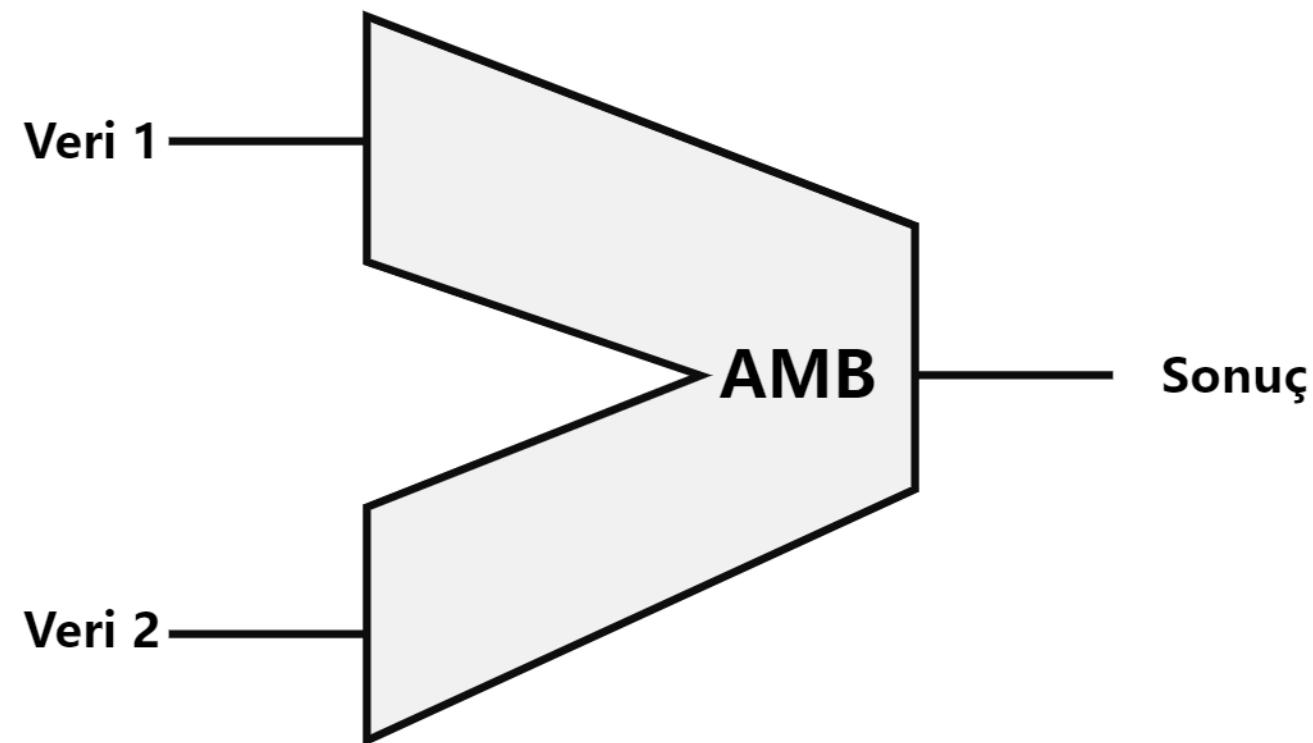
# Yazmaç Öbeği

**Yazmaç Öbeği**, yazmaç numaraları verilen kaynak yazmaçlarındaki verileri çıktı olarak verir veya yazmaç numarası verilmiş hedef yazmaca verilen veriyi yazar.



# Aritmetik Mantık Birimi

Aritmetik – mantık işlemlerinin yapıldığı birimdir.



# Anlık Oluşturma Birimi

Buyruk kümesi mimarisine göre farklı buyrukların anlık değerleri farklı bit uzunluklarında olabilir ve Aritmetik Mantık Birimi 32 bitlik verilerle işlem yapmaktadır. Bu anlık değerlerin işleme girebilmesi için genişletilmesi gerekmektedir.

**Anlık Oluşturma Birimi**, işleme girecek olan bu anlık değerleri buyruktan seper işaretle genişletir.

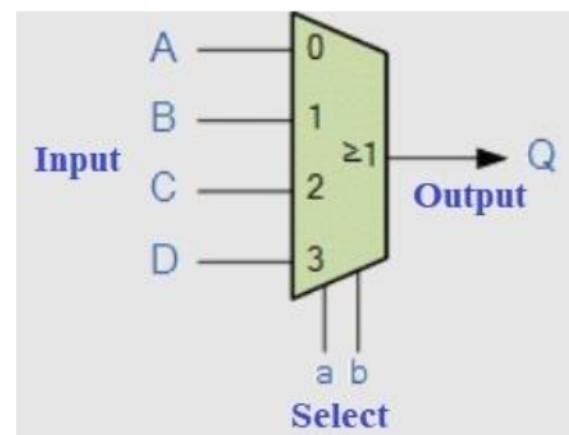


**Veri Belleği**, çalışan programların bilgilerini tutar ve bu adreslere erişimi sağlayan birimdir. Verilen adrese okuma ya da yazma yapabilir.

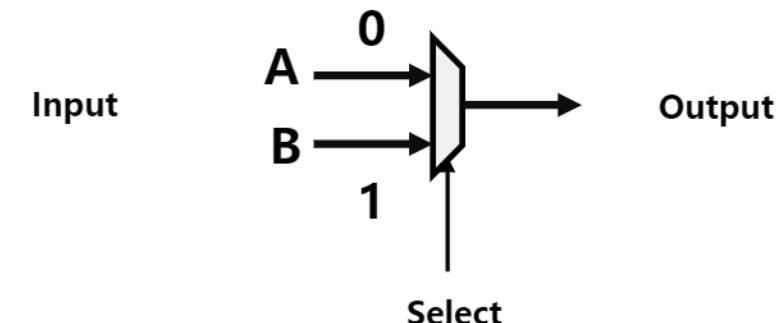


# Seçici Multiplexer

Multiplexer girişindeki verilerden hangisinin çıkışa aktarılacağının seçiminin yapıldığı devredir. Buyruk mimarisinde farklı buyruklar aynı veri yolu birimlerini kullanmak istediklerinde bu istekleri ayırt edebilmek için seçiciler kullanılmaktadır.



4x1 MUX



2x1 MUX