

Bilgisayar Mimarisi

Komut Kümesi Mimarisi - 2

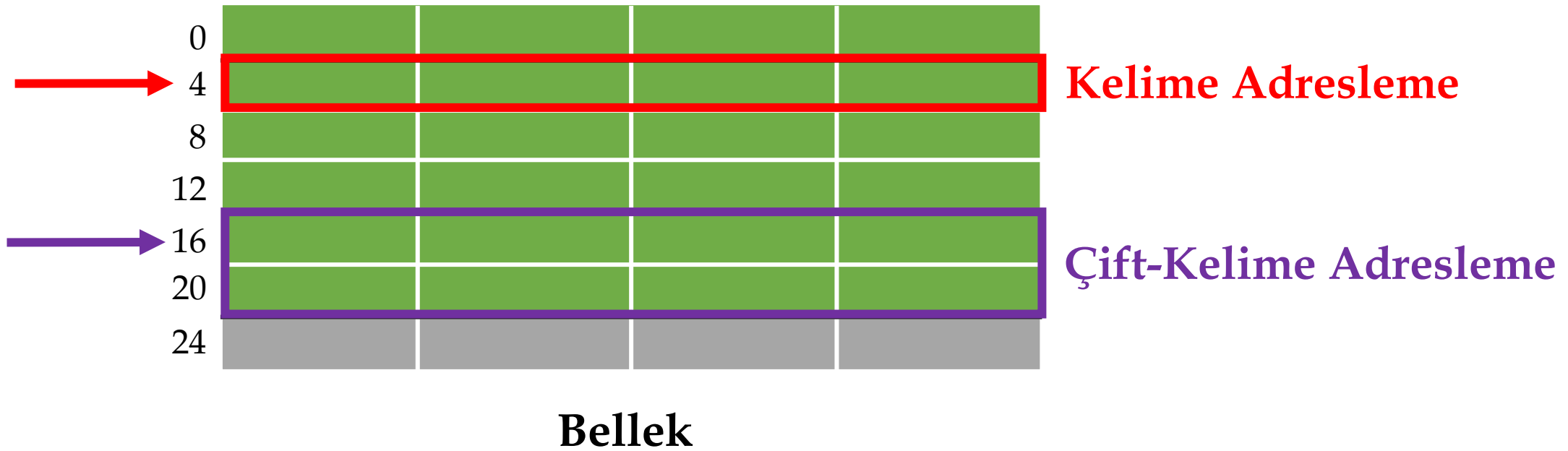
(Instruction Set Architecture (ISA))

Bellek Adreslerinin Hizalanması

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

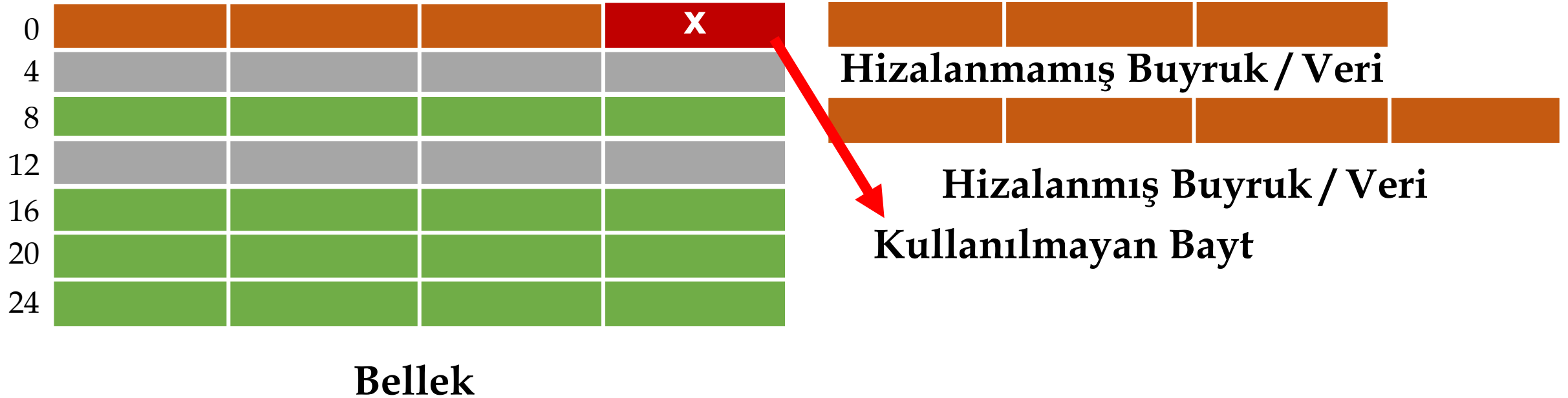
- Bellekteki her **bayt bir adrese** sahiptir.
- Çift-kelime (doubleword)(8 Bayt / 32 Bit) erişimlerinin adresleri **8'in** tam katı olmalıdır.
- Kelime (word) (4 Bayt / 16 bit) adresleri **4'ün** tam katı olmalıdır.
- *Bellek verileri Neden bit bit veya 4 bayt 4 bayt adreslenmez de bayt bayt adreslenir?*
- INTEL'de bayt bayt okuma yazma yapılabilirken MIPS ve RISC V'de word veya doubleword (32 bit veya 64 bitlik işlemciler için) olarak okuma yazma yapılır.
- Burada arada boş baytların kalmaması için sıralı olarak kaydedilir buna **bellek hizalama veya hizalı erişim** denir.
- Bu kısıtlar **donanımın basit kalmasını** sağlar.

Bellek Adreslerinin Hizalanması

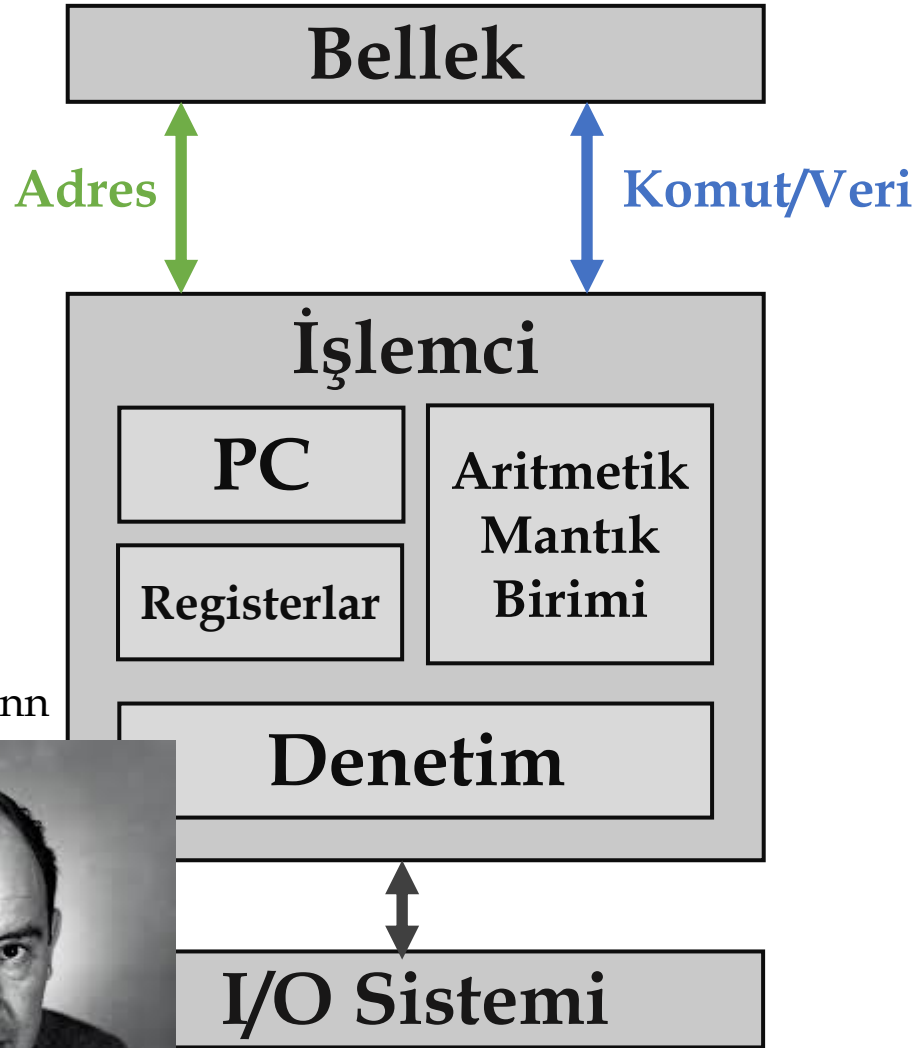


Bellek Adreslerinin Hizalanması

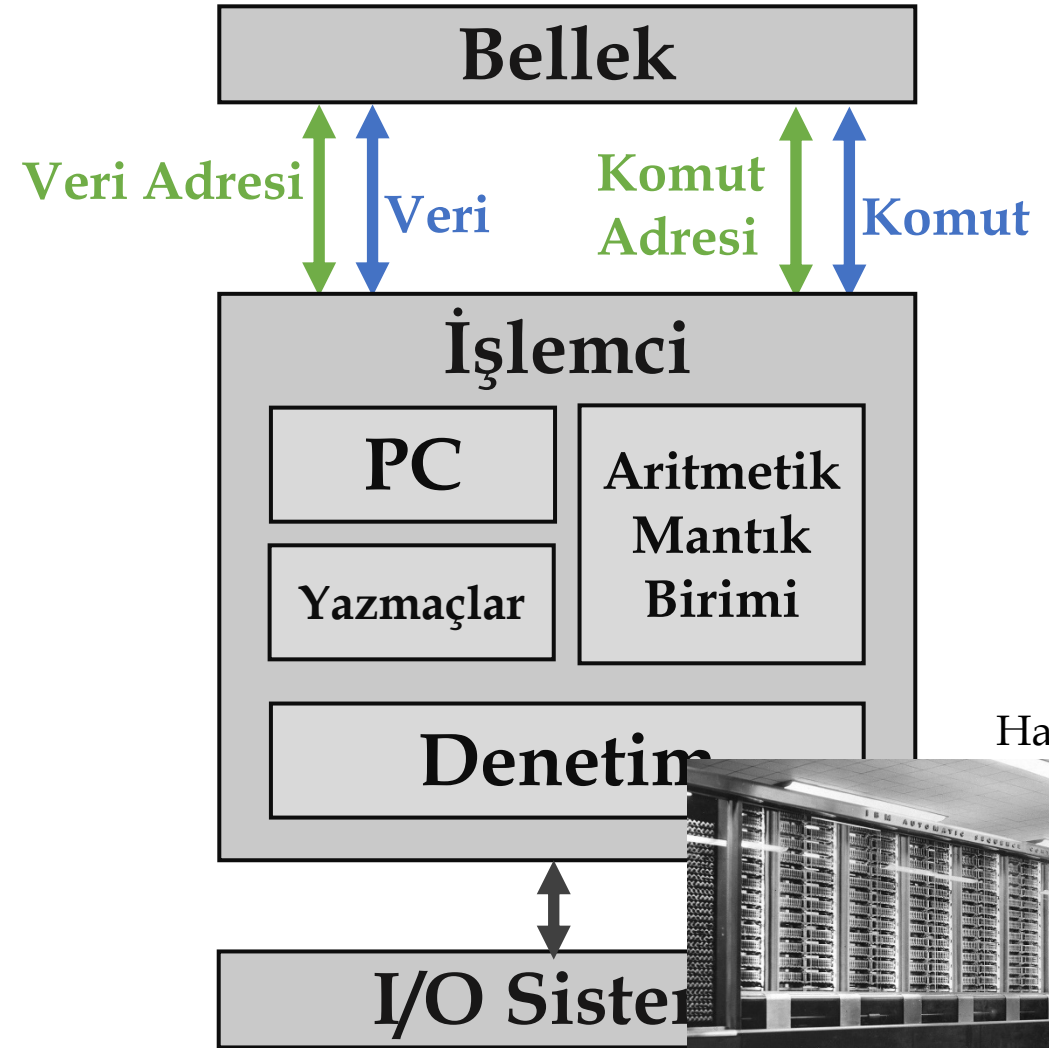
MIPS, RISC V gibi Komut Kümeleri daha hızlı komut işlemek için sabit boyutta komut kümeleri ile çalışır.



Harvard ve Von Neumann Mimarileri



Von Neumann
Mimarisi
CISC



Harvard
Mimarisi
RISC

Harvard Mark 1



John
von Neumann



Harvard ve Von Neumann Mimarileri



- Aynı fiziksel bellek uzayı hem komutlar hem de veri için kullanılır.
- Komutlar ve veriler aynı hattı kullanır.
- İşlemci komutlara ve veriye aynı anda erişemez.
- **Bir komutun tamamlanması iki çevrim sürer.** İlk çevrimde komut, ikinci çevrimde veriye erişilir.
- Denetim biriminin maliyeti daha düşüktür.

G/Ç Sistemi

**Von Neumann
Mimarisi**



- Veri ve komutlar farklı fiziksel bellek uzaylarında kullanılır.
- Veri hattı ve komut hattı ayrıdır.
- İşlemci komutlara ve veriye aynı anda erişebilir.
- **Bir komutun tamamlanması bir çevrim sürer.**
- Denetim biriminin maliyeti daha yüksektir.

G/Ç Sistemi

**Harvard
Mimarisi**

Sabit Boyutlu Komut Genişliği

MIPS ve RISC V gibi mimarilerde

0				
4				
8				
12				
16				
20				
24				

Komut Belleği

Program Sayacı +4

8

Her komut aynı uzunlukta olduğu için
PC her zaman aynı miktarda artar.

Değişken Boyutlu Komut Genişliği

INTEL gibi CISC mimarisinde



Komut Belleği

Komutlar farklı uzunluklarda olduğu için PC her zaman aynı miktarda **artmaz**.

Peki INTEL neden değişken boyutlu komut genişliği kullanıyor?

Komutların Kodlanması

Donanımda komutlar elektrik sinyalleri ile iletilir.

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

Basitçe, bir komut sayı olarak gösterilen küçük parçalardan oluşur.

Alan: Komutun bilgi içeren bir parçasıdır. Her Komut türünün belirli alanları olması gerekir.

add x9, x20, x21 →

Onluk Taban:

İkilik Taban:

0	21	20	0	9	51
0000000	10101	10100	000	01001	0110011
7 bit	5 bit	5 bit	3 bit	5 bit	7 bit

Toplama (add)

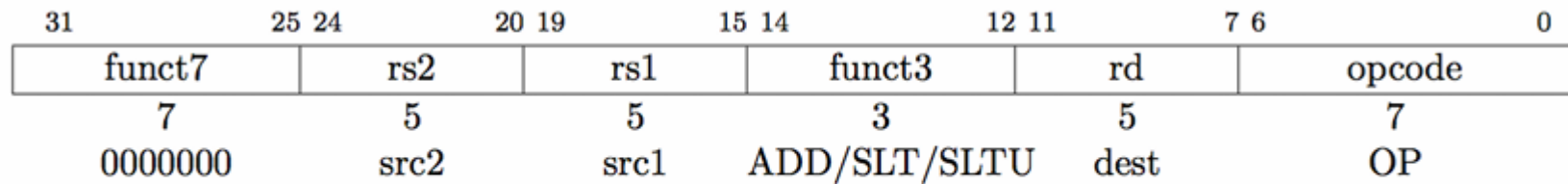
Kaynak Register (x21)

Hedef Register (x9)

Kaynak Register (x20)

*funct7 ve funct3 alanları
işlem türünü seçer.

*funct3 opcode'nin(işlemin)
işlevi ile ilgili bitler.



Komutların Kodlanması



Yukarıdaki bir RISC-V **komut formatıdır**.

- Bütün RISC-V buyrukları **32-bit** genişliğindedir.
 - Basit tasarım.

Makine dili: Komutların sayısal gösterimi.

Makine kodu: Sayısal olarak gösterilen komutlar dizisi.

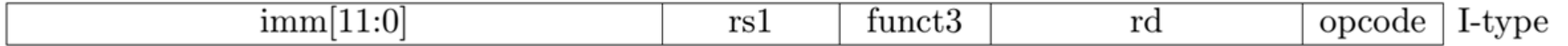
Tüm RISC-V Komut Tipleri

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0			
funct7				rs2			rs1		funct3		rd			opcode		R-type	
imm[11:0]							rs1		funct3		rd			opcode		I-type	
imm[11:5]				rs2			rs1		funct3		imm[4:0]			opcode		S-type	
imm[12]		imm[10:5]		rs2			rs1		funct3		imm[4:1]		imm[11]		opcode		B-type
imm[31:12]										rd			opcode		U-type		
imm[20]		imm[10:1]			imm[11]		imm[19:12]			rd			opcode		J-type		

- MIPS’de sadece R, I ve J türü komutlar vardır ve Opcode’ları 6 bit ve en soldadır.

Tüm RISC-V Komut Tipleri

- I türü Komut



- Amacı 12 bitlik sabit değeri bir yere atamak.
- $a=12$
- $a=a+1$

Tüm RISC-V Komut Tipleri

- S ve B türü Komut

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
-----------	-----	-----	--------	----------	--------	--------

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
---------	-----------	-----	-----	--------	----------	---------	--------	--------

- S = Store: Bir bilgiyi belleğe göndermek için
- B = Branch

Tüm RISC-V Komut Tipleri

- U türü Komut



- 20 bitlik sabit değeri registıra atamak için

RISC-V Komut Türleri Genel Bakış

RISC-V BKM	Kategori	Buyruklar
RV64-I (Taban)	Aritmetik	add, sub, addi
	Mantıksal	sll, srl, sra, and, or, xor, not
	Koşul (Dallanma / Branch)	beq, bne, bge, blt, bltu, bgeu, jal, jalr,
	Bellek	ld, sd, lw, sw, lh, sh, lui
A Eklentisi	Atomik Bellek	sc, lr
M Eklentisi	Tamsayı çarpma bölme	mul, mulh, mulhu, mulhsu, div, divu, rem, remu
D / F Eklentileri	Kayan virgül aritmetik işlemler (Double-precision / Single-precision)	fadd, fsub, fmul, fdiv, fmin, fmax, fsqrt, fcmp

Çarpma İşlemi

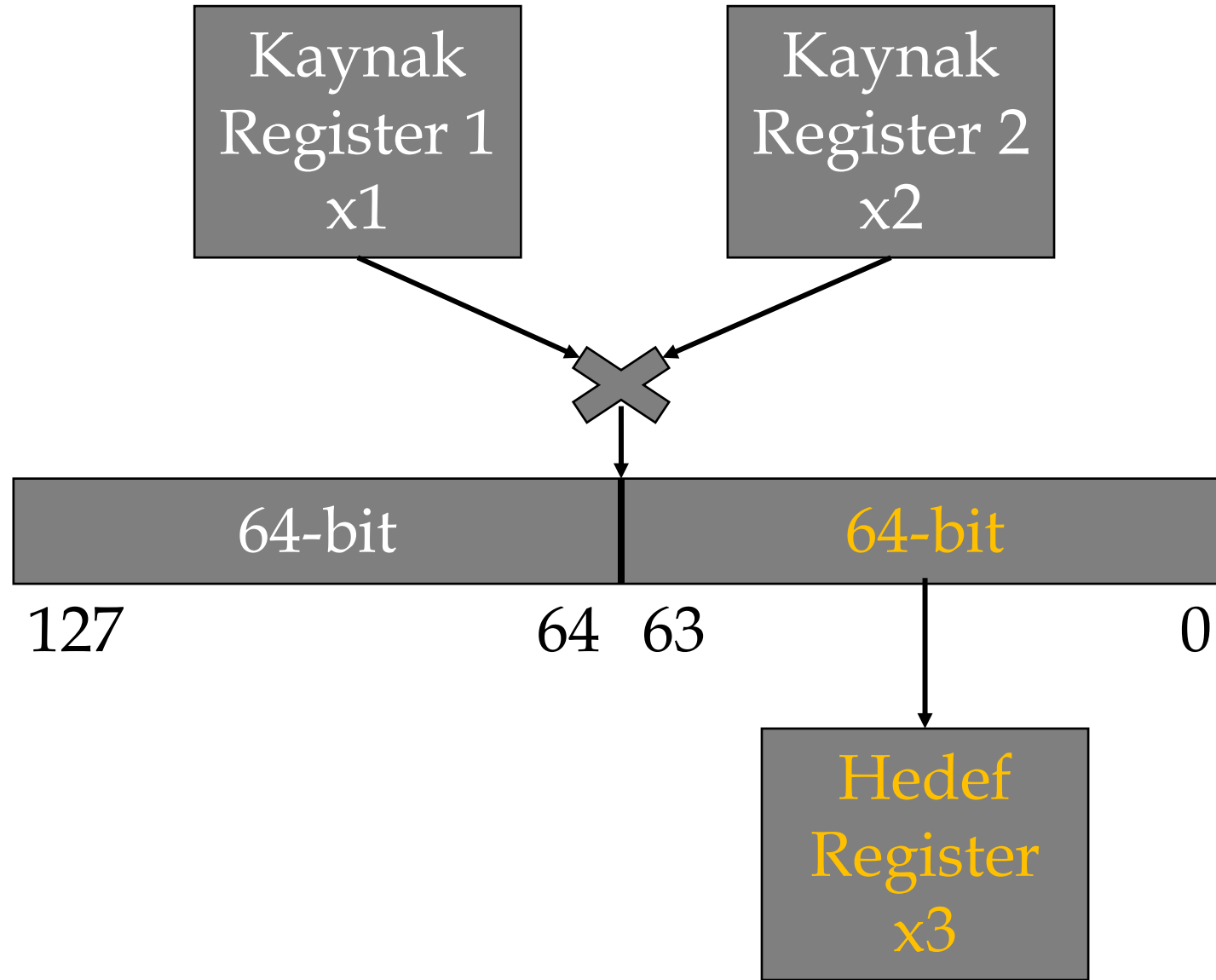
- X-bitlik iki değerin çarpım sonucu kaç bittir?
 - **2X-bit.**
- Çarpma işleminin iki register işleneni vardır.
 - **64-bitlik** iki değerin **çarpım** sonucu **128-bit** olur.
 - Sonucu kaydedebilmek için **iki hedef register** gerekiyor.

RISC-V Çarpma Komutu [MUL]

MUL X1, X2, X3

MUL

Çarpım Sonucu

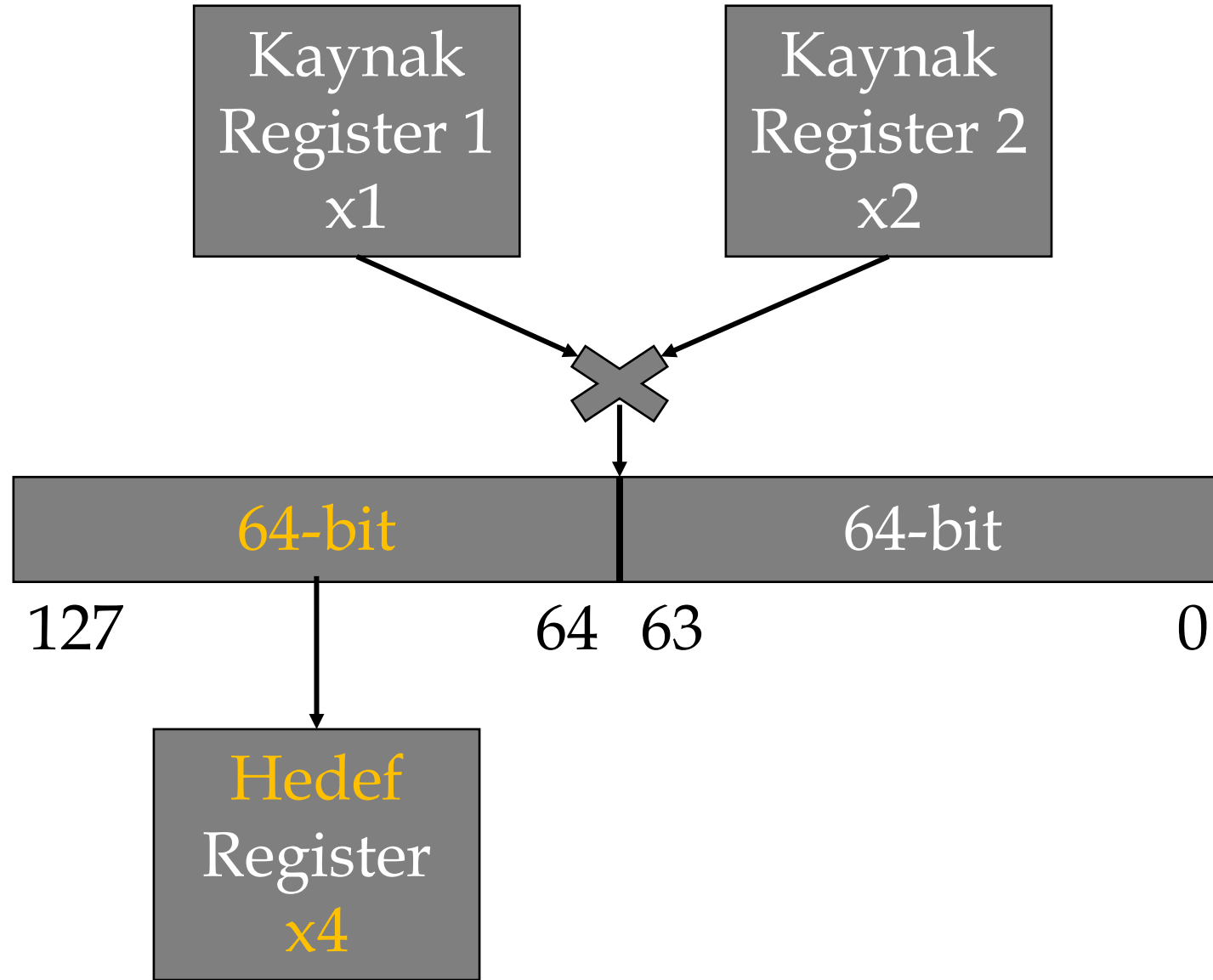


RISC-V Çarpma Komutu [MULH]

MULH X1, X2, X4

MULH

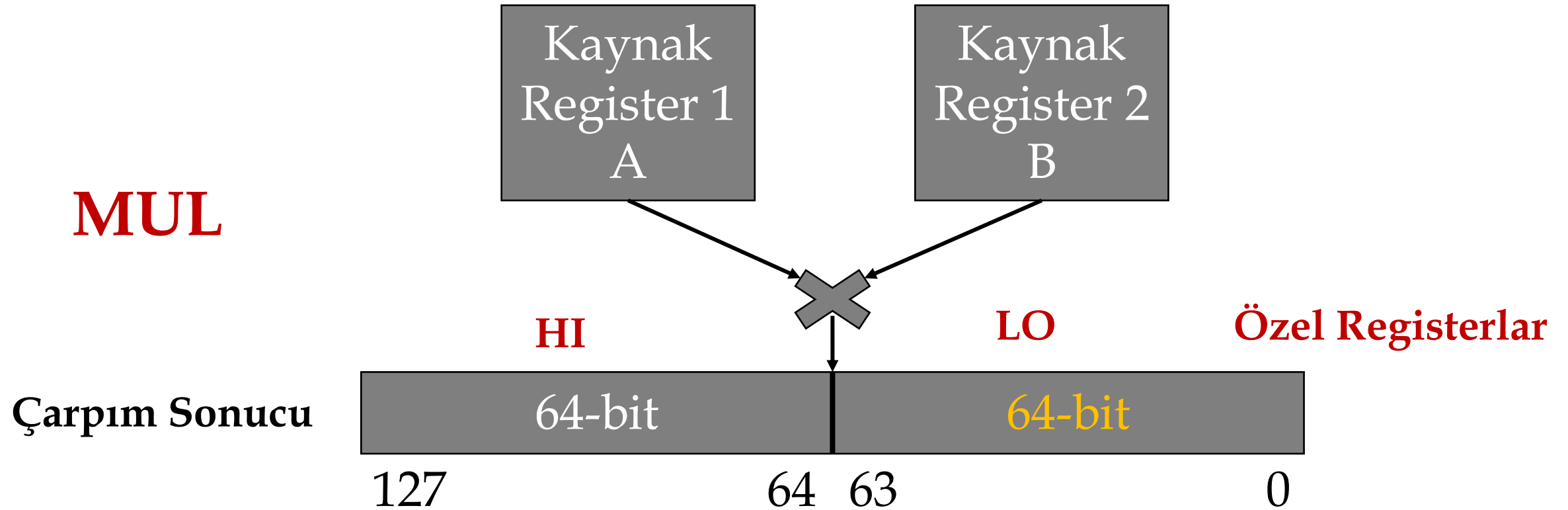
Çarpım Sonucu



MIPS'de Çarpma Komutu [MUL]

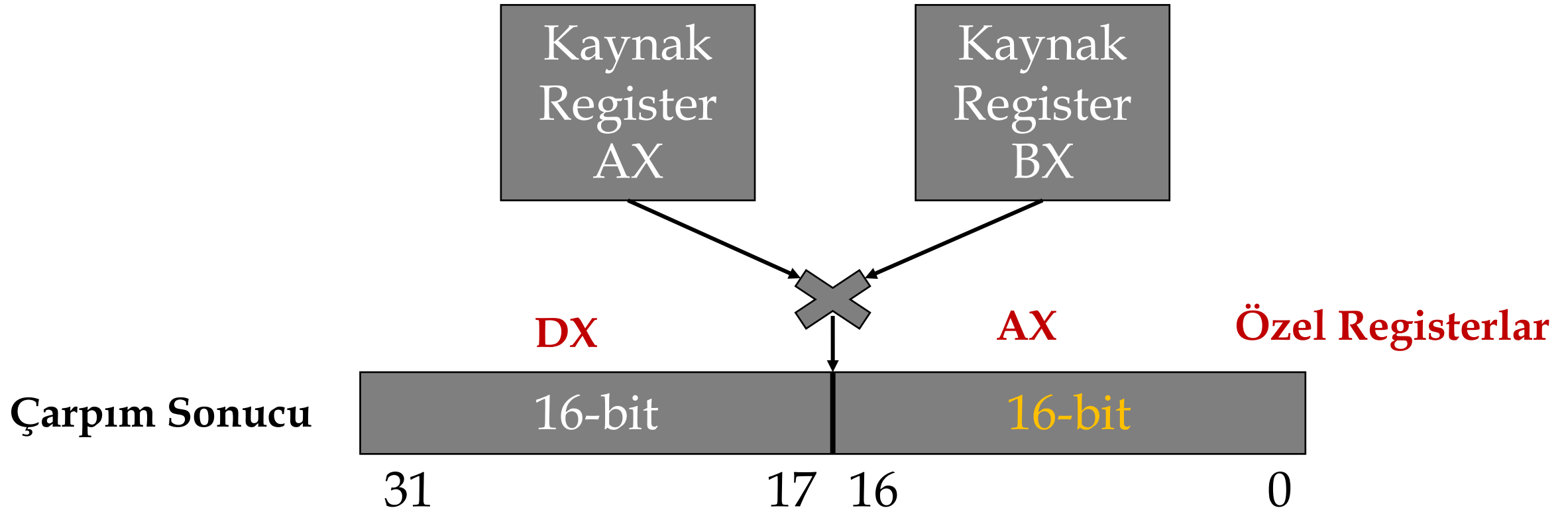
MUL A, B

MUL



INTEL'de Çarpma Komutu [MUL]

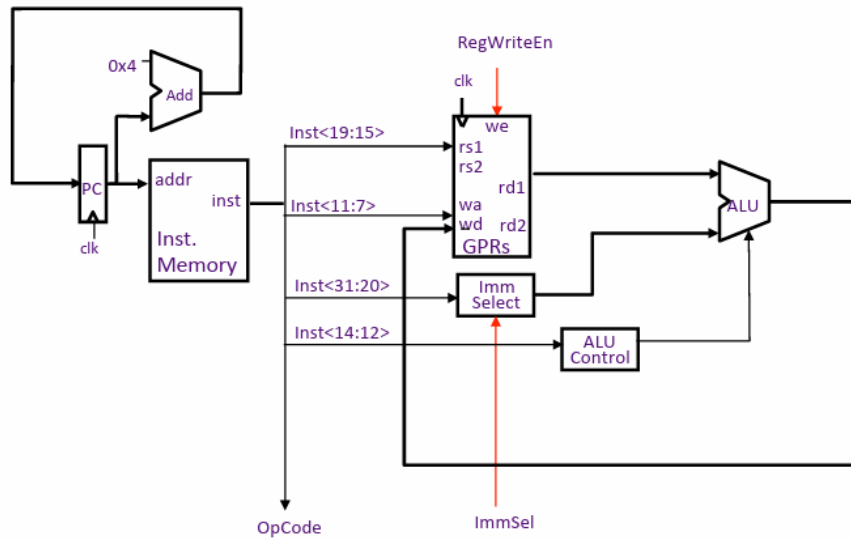
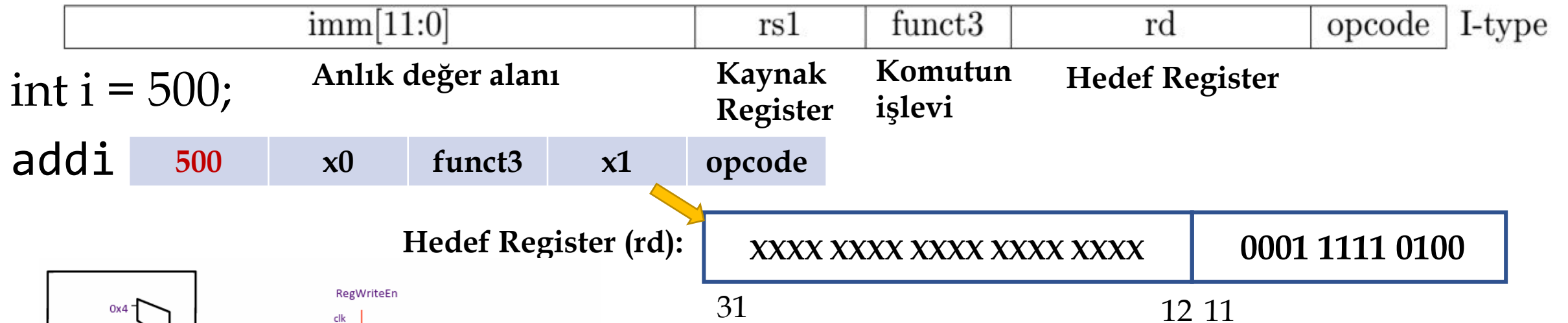
MUL BX



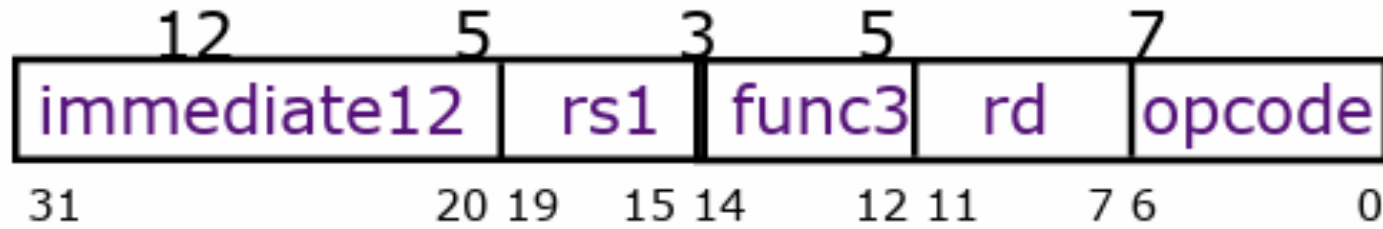
AX: Acumulator

Anlık Değerler

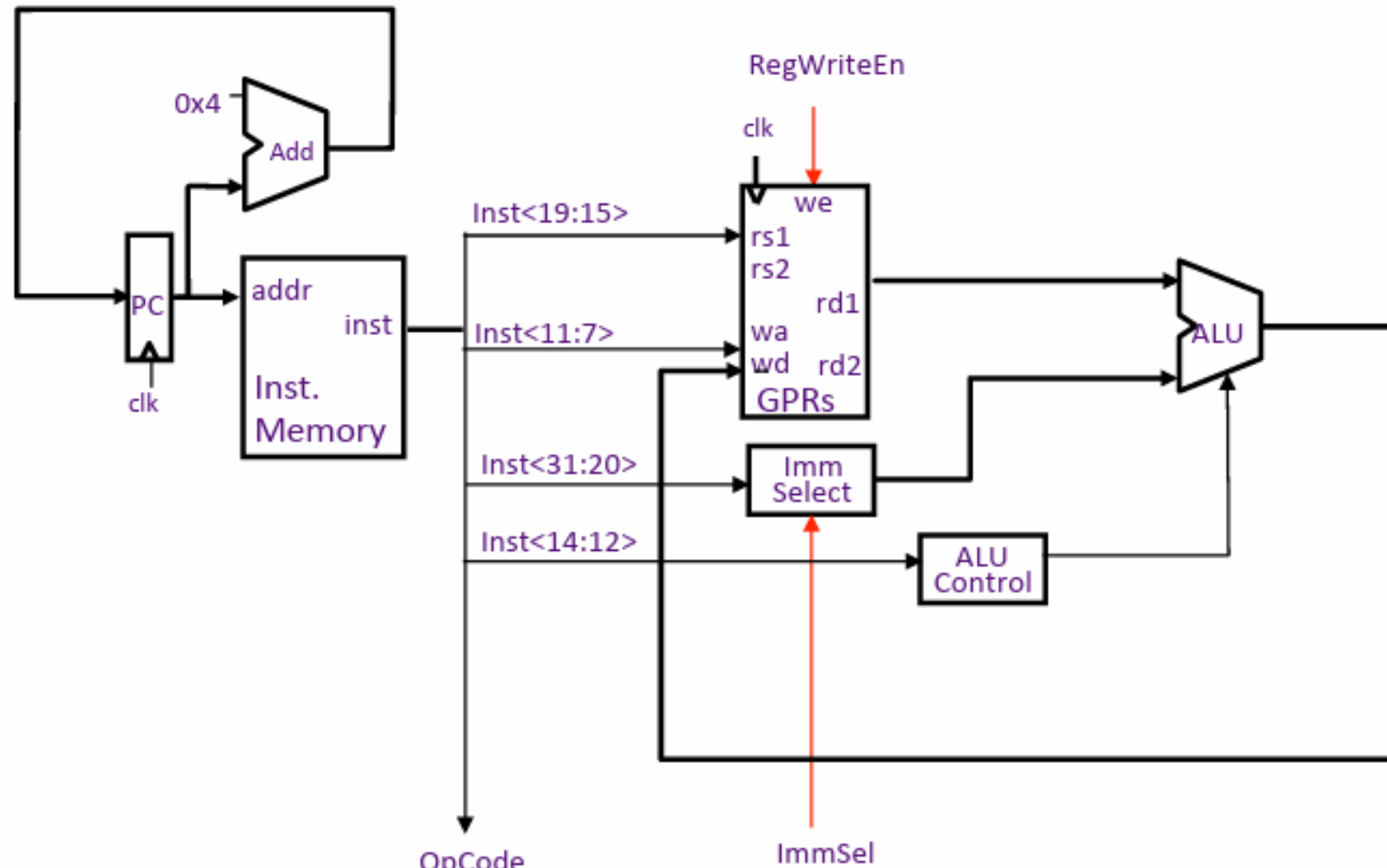
Programlarda sabit değerler sıklıkla kullanılır. Sabit değerler kullanılacakları zaman **bellekten** load komutları ile yüklenebilir ya da **anlık değer** olarak komutlar ile iletilebilir.



Anlık Değerler



$rd \leftarrow (rs1) \text{ op immediate}$



Anlık Değerler

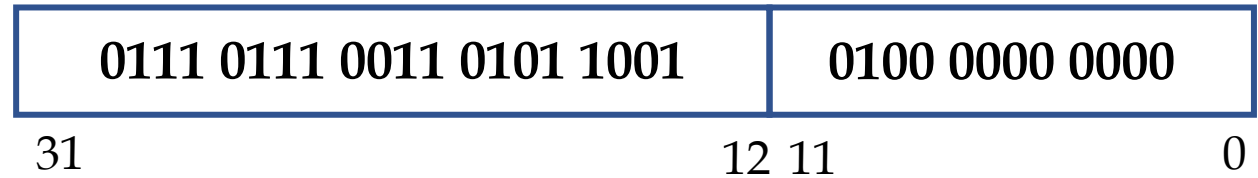


```
int i = 2000000000;
```

```
lui 488281 rd opcode --> registerın [31:12] bitlerine yazar
```

```
addi 1024 x0 funct3 rd opcode
```

Hedef Register:



lui: Load Upper Immediate

opcode: iş kodu

rd: hedef register

C Kodu Derleme Örneği

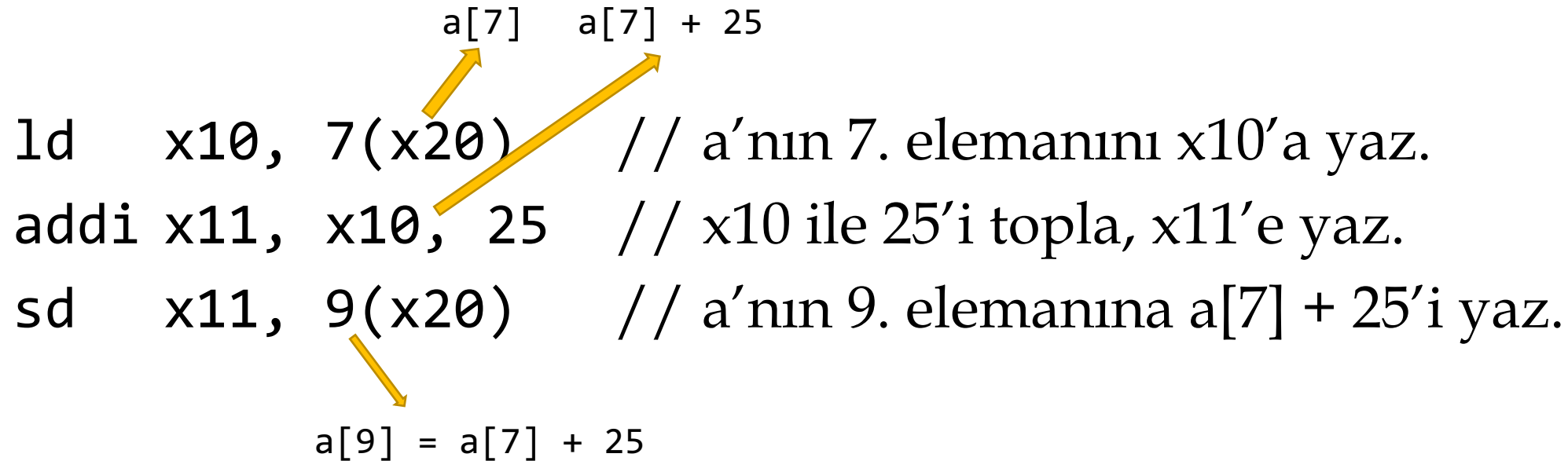
`uint64_t *a;` `// a dizisine işaret eden işaretçi`

`a[9] = a[7] + 25;` **Derleyici Değişken Eşleştirmesi**
 $a \rightarrow x20$

Komut	Örnek	Anlamı	Açıklama
Yükle	<code>ld x5, 8(x7)</code> (Load Doubleword)	$x5 = x7[8]$	x7 başlangıç adresli dizinin 8. elemanını x5'e yaz. (bellekten okur)
Sakla	<code>sd x5, 8(x7)</code> (Store Doubleword)	$x7[8] = x5$	x5'i x7 başlangıç adresli dizinin 8. elemanına yaz. (belleğe yazar)

C Kodu Derleme Örneği

```
uint64_t *a;           // a dizisine işaret eden işaretçi  
a[9] = a[7] + 25;      Derleyici Değişken Eşleştirmesi  
                        a → x20
```



```
ld    x10, 7(x20)      // a'nın 7. elemanını x10'a yaz.  
addi  x11, x10, 25     // x10 ile 25'i topla, x11'e yaz.  
sd    x11, 9(x20)      // a'nın 9. elemanına a[7] + 25'i yaz.
```

a[7] a[7] + 25

a[9] = a[7] + 25

Kodun Belleğe Saçılması

Çoğu programda yazmaçlardan fazla sayıda değişken vardır.

- Tüm değişkenler her zaman yazmaçlarda bulunamaz.

Derleyici **yakın zamanda kullanılmayacak** değişkenleri **bellekte** tutmaya çalışır. Buna kodun belleğe saçılması denir.

Yazmaçlara erişim belleğe erişimden hızlıdır. Programların **verimli** çalışması için:

- Donanımda **yeterince yazmaç** olması,
- **Derleyicilerin yazmaçları verimli kullanması** gerekir.