Bölüm: 5
İşlemci: Datapath ve Kontrol

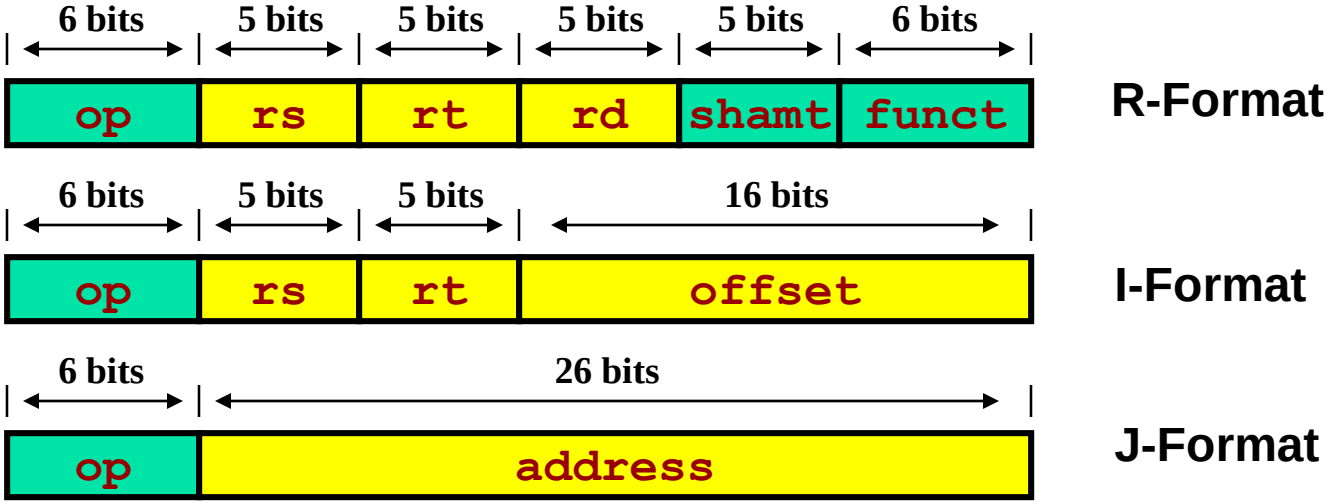
Slide Kaynağı: Patterson & Hennessy COD book website (copyright Morgan Kaufmann) adapted and supplemented

Önemli bir problemin çözümünde, hiçbir detay önemsiz değildir – **French Proverb**

Adam haklı **– Hayrettin Can**

MIPS'in gerçekleştirilmesi

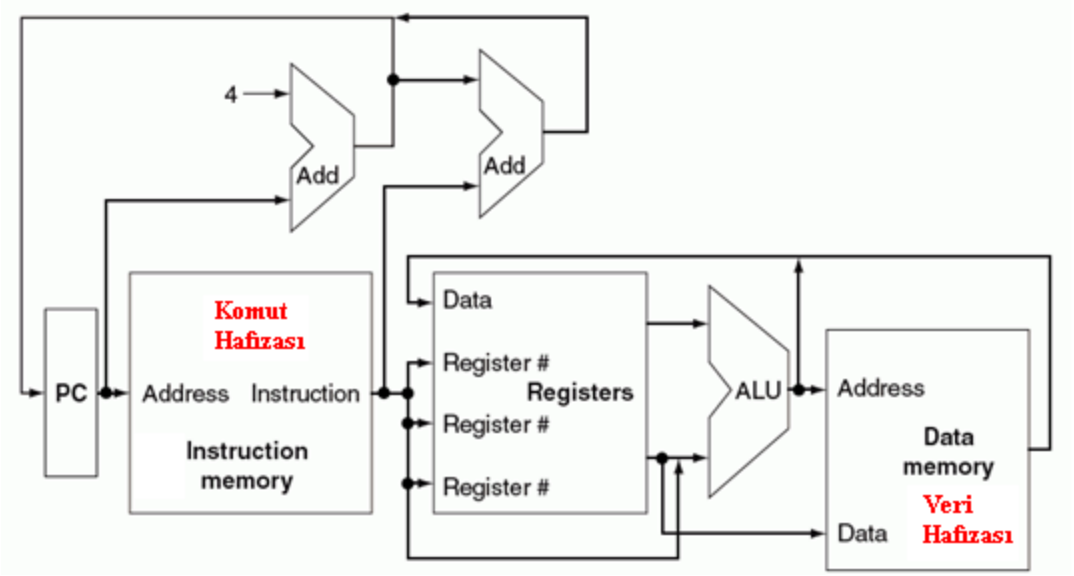
- Biz şimdi bir MIPS komut setinin gerçekleştirilmesini incelemeye hazırız.
- Komut yapılarını ana hatlarıyla hatırlayalım.
  - Aritmetik-Lojik komutlar: `add, sub, and, or, slt`
  - Hafıza-tabanlı komutlar: `lw, sw`
  - Kontrol-Akış komutları: `beq, j`



MIPS'in gerçekleştirilmesi: Fetch/Execute Çevrimi

- Fetch/execute gerçekleştirme işlemine yüksek seviye ayrıştırmayla bakarsak:
  - Komut adresini okumak için program Counter (PC) kullanılır.
  - Komutun hafızadan alınması ve PC'nin arttırılması.
  - Okunacak registerleri seçmek için için komut yapısındaki alanları kullan. Örneğin transfer komutu (Load, store) sadece 1 registerin okunmasına gerek duyar fakat diğer komutlar 2 reg'in okunmasına gerek duyar.
  - Komut sınıfına bağlı yürütme *(execute*) işlemi.
  - Tekrar…….

Dikkat ediniz ki çoğu komutlar doğrudan icra edilir ve bir sonraki komutun adresi PC+4 ile yazılır. Fakat dallanma veya atlama komutlarında ise ALU dallanılacak yeni komut adresini hesap edebilir. Veya şartsız dallanma komutlarında ise PC+4 ikinci bir toplama ile dallanılacak komutun adresi elde edilir.



Komut getirildikten sonra, formatına göre operandlar ilgili reg'lere getirilip işlenir. Veya ana hafıza ya load/store edilebilir.

İşlemcilerde gerçekleştirme stilleri

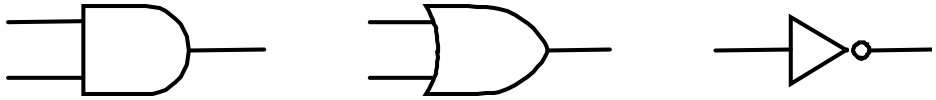
- Tek Cycle
  - Herbir komut 1 clock cycle'da icra edilir.
  - Clock en yavaş komutlar için yeterince uzun olmalıdır.; Bu yüzden,
  - Dez avantaj: sistem en yavaş komut kadar hızlıdır.
- Çoklu-Cycle
  - Çoklu-cycle adımlarında fetch (Alma)/execute(Yürütme) yapılır
  - Her bir clock cycle'da 1 adım icra edilir.
  - Avantajı: Her bir komut ihtiyacı kadar birkaç cycle kullanır.
- Pipeline
  - Herbir komut çoklu adımlarda yürütülür.
  - Her bir clock cycle'da 1 adım/komut icra edilir.
  - Paralel olarak çoklu komutlar işlenir – assembly line

Functional Elemanlar

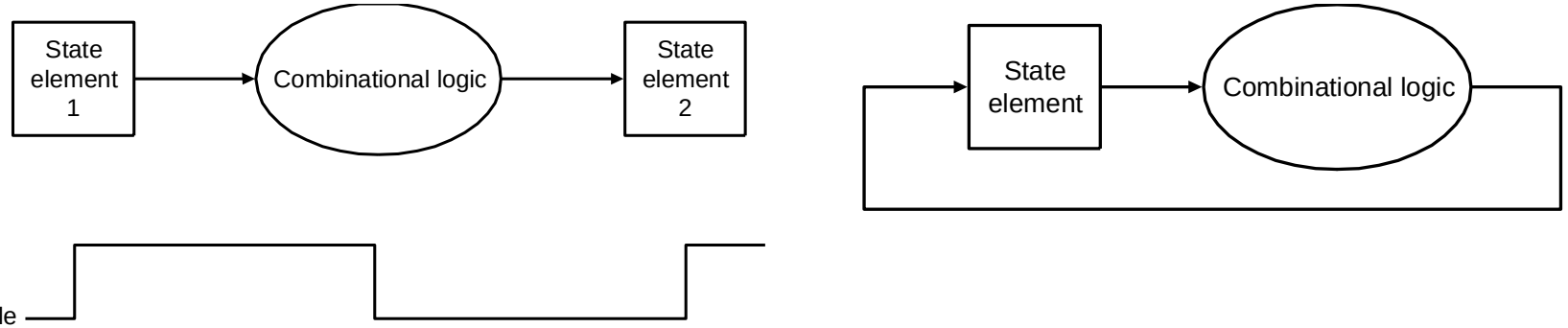
- Donanımda 2 tip fonksiyonel eleman vardır.
  - Veri üzerinde işlem yapan elemanları : (*Kombinasyonal elemanlar olarak adlandırılır – Toplayıcılar v.b.*)
  - Veri içeren elemanlar: (*state* veya *ardışıl elemanlar olarak adlandırılır. Hafıza elemanları v.b*)

Kombinasyonal Elemanlar

- giriş $\Rightarrow$ *çıkış fonksiyonları gibi çalışır* . örneğin; ALU
- Kombinasyonal lojik, bir registerden okur veya bir registere veya okuduğu registere yazar.
  - *read/write bir cycle'da meydana gelir.* – kombinasyonal eleman gelecekte kullanabilmek için veri depolayamaz



Combinational logic donanım birimleri



State (Durum) Elemanlar

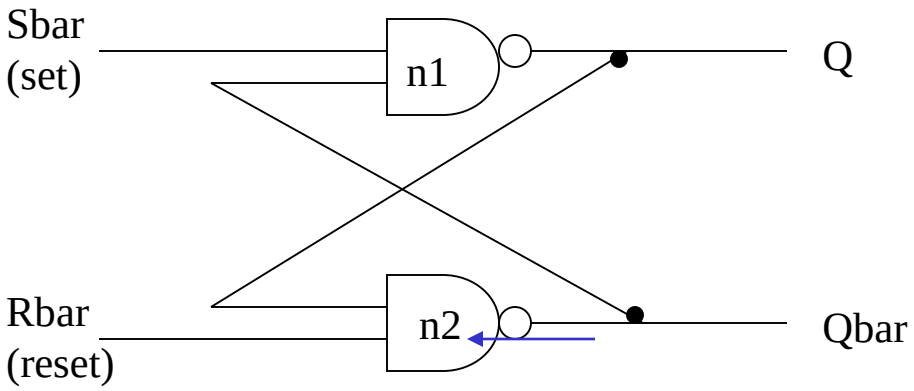
- State elemanlar dahili hafızaları içerir. Ör: *registers* ve hafıza.
- Bütün state elemanlar makinanın durumunu tanımlar
  - *Bunun anlamı nedir?* Tekrar açmayı ve kapatmayı düşün.

- *Flipflops* ve *latch'ler* 1 bitlik state(Durum) elemanlardır, Eşdeğeri 1 bitlik hafızalardır.

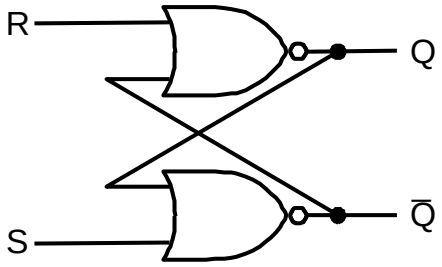
- Flip-flop veya Latch'in çıkış(ları) daima saklanan bit değerine bağlıdır. Onların durumu 0/1 veya yüksek/düşük veya doğru/yanlış olarak adlandırılır.

- Bir flip-flop veya latch'in girişi onun durumuna bağlı olarak clock olup olmadığına göre değişebilir.

Set-Reset (SR-) latch (unclocked)

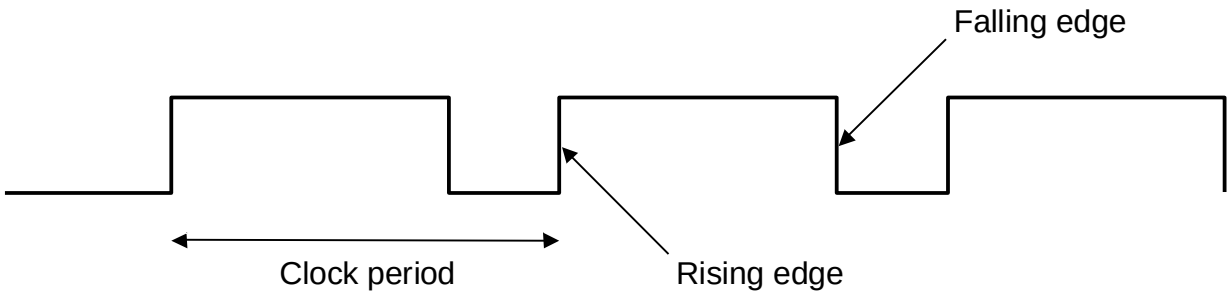


NOR latch



Senkronize Lojik:
Clock girişli FF'lar veya LATCH'ler

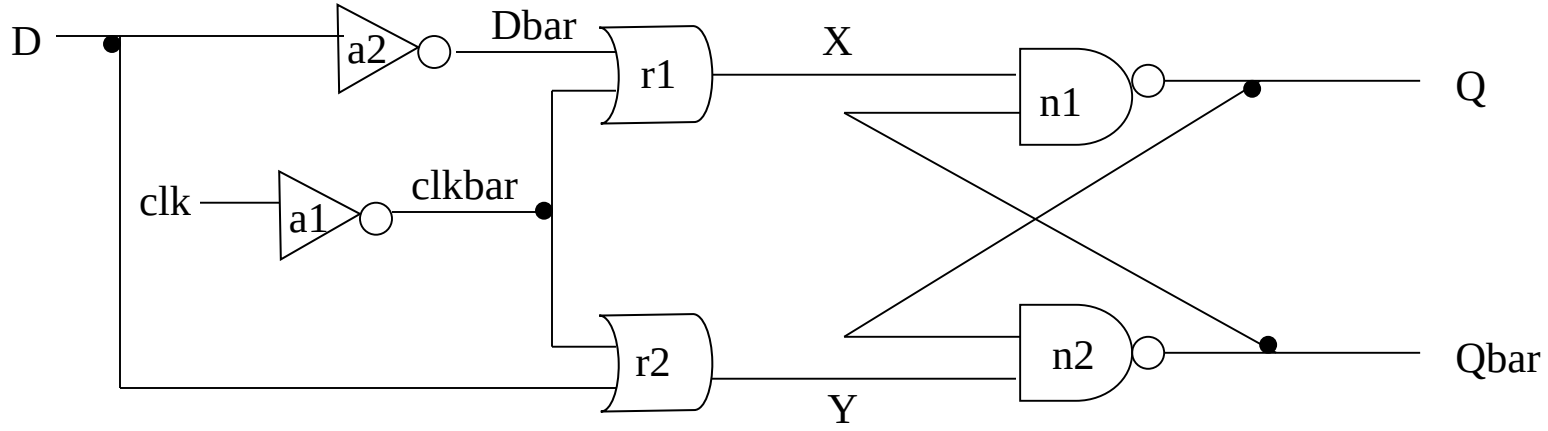
- Clock işareti; bir state elemanını (FF'lar barındıran) senkronlamak veya ona yaz/oku yapmayı denetlemek için kullanılır.
  - Genellikle kenar tetiklemeli (Yükselen/Düşen) state elemanları kullanılır.

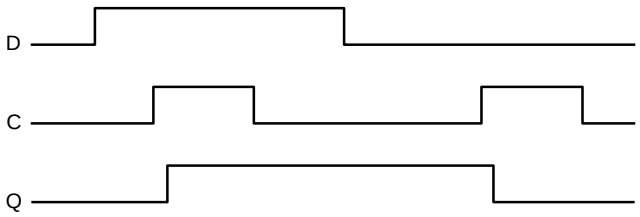


- Latch'ler seviye tetiklemelidir.
- Flip-flops are kenar tetiklemelidir.

Clock girişli D-latch

- Durum (Q) sadece clk'nın 1 olduğu zamanda değişebilir.
- Tek bir data (D) girişi vardır. (SR-latch ile karşılaştırıldığında)
- *Tanımlanmamaış davranışı yoktur.*

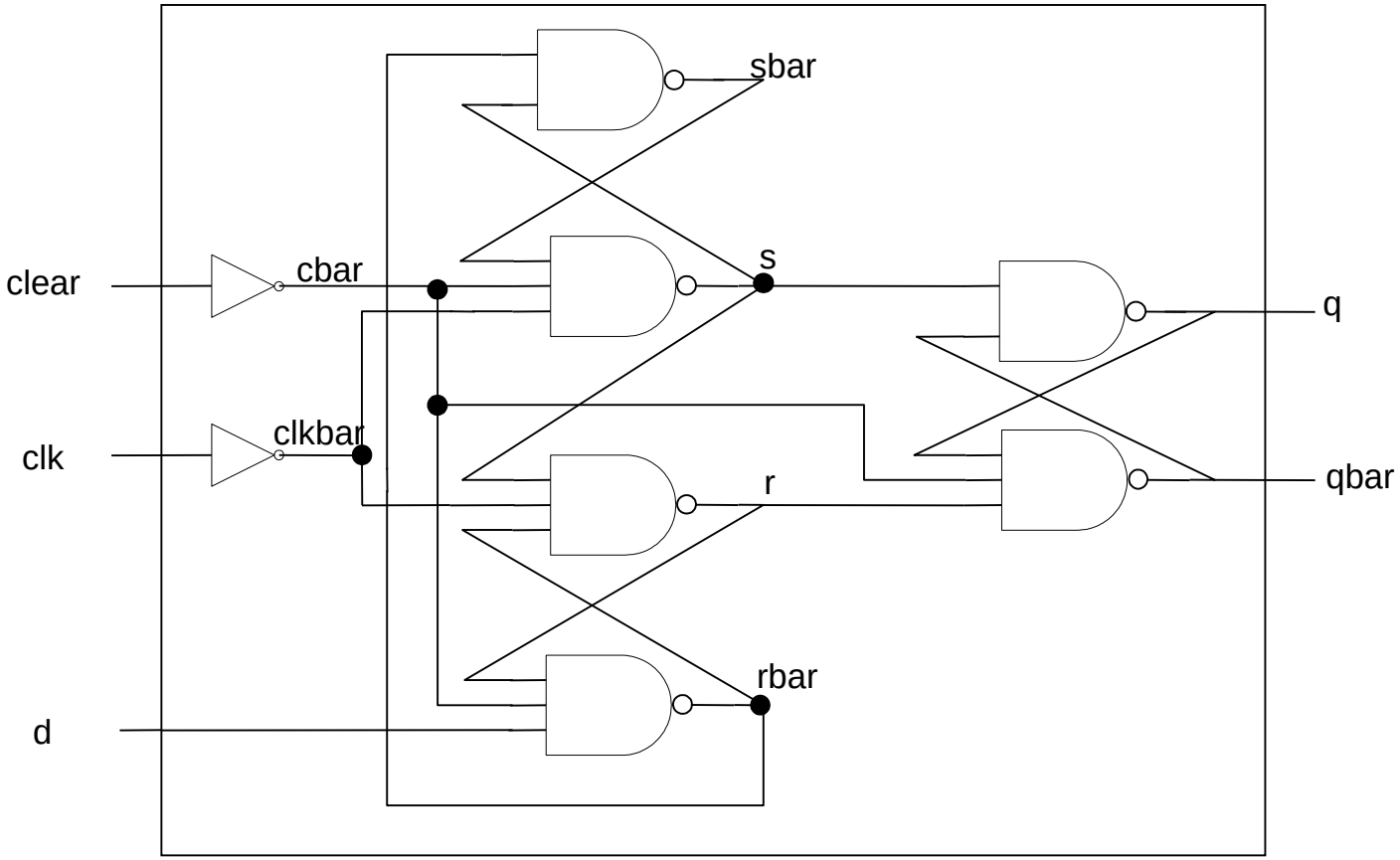




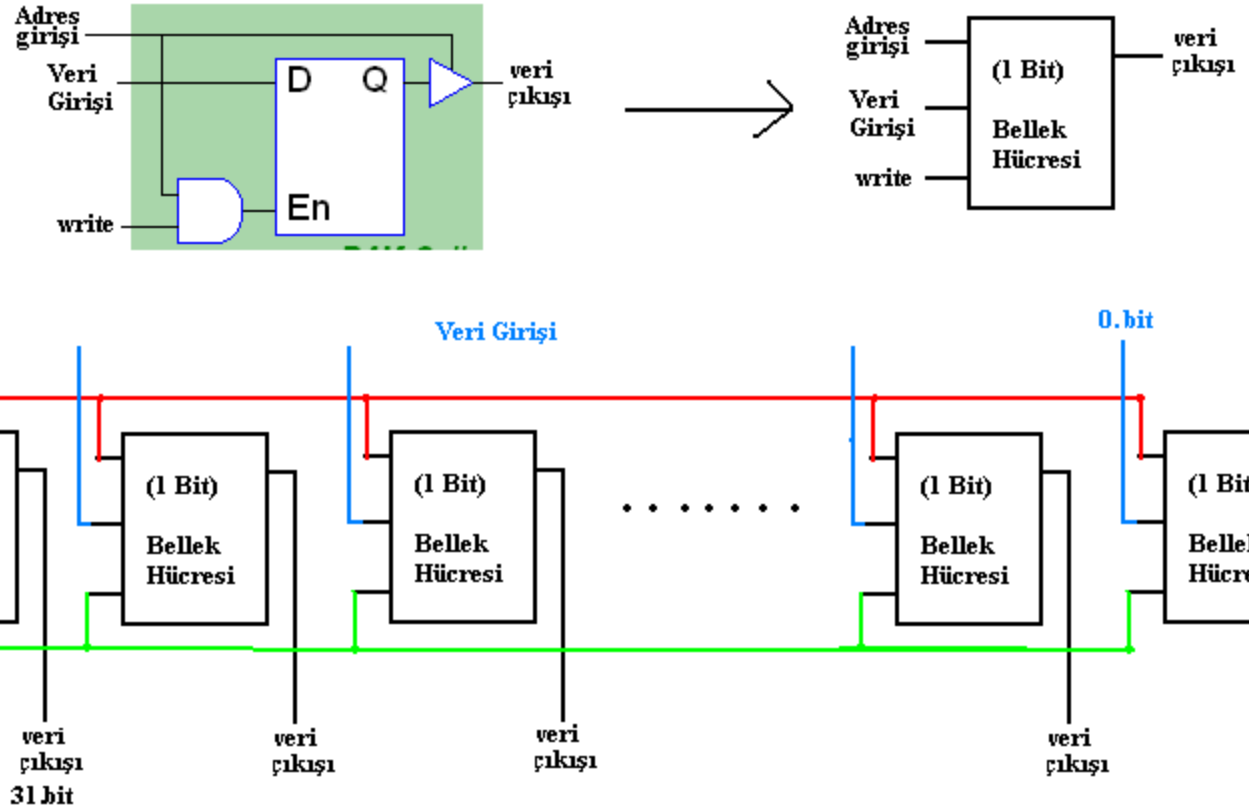
D-LATCH'in zamanlama diyagramı

Clock girişli D-flipflop

- *Negatif kenar tetiklemeli FF'dır.*
- 3 tane S-R Mandal'la yapılır.

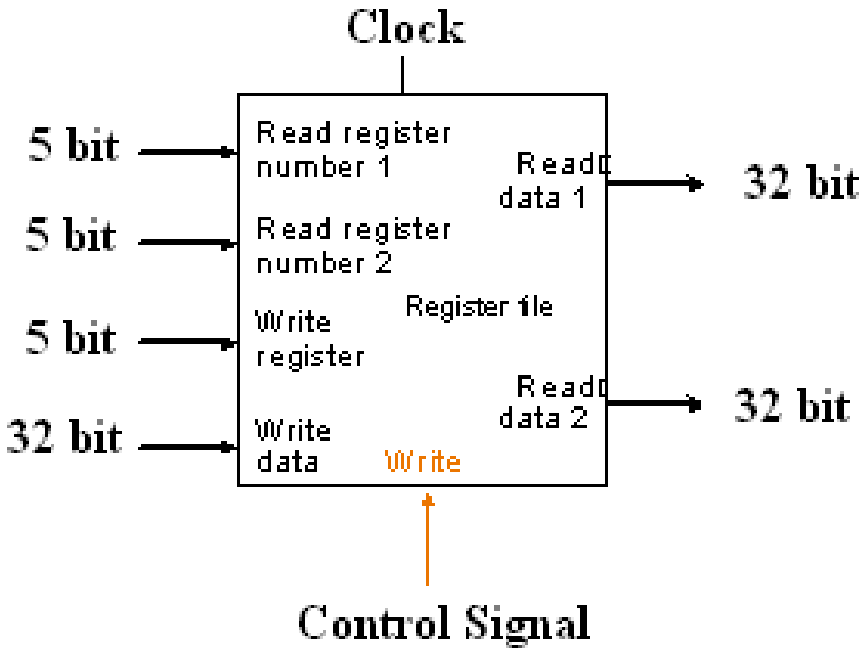


Register yapısı



Data path'de durum(State) elemanları: Register File

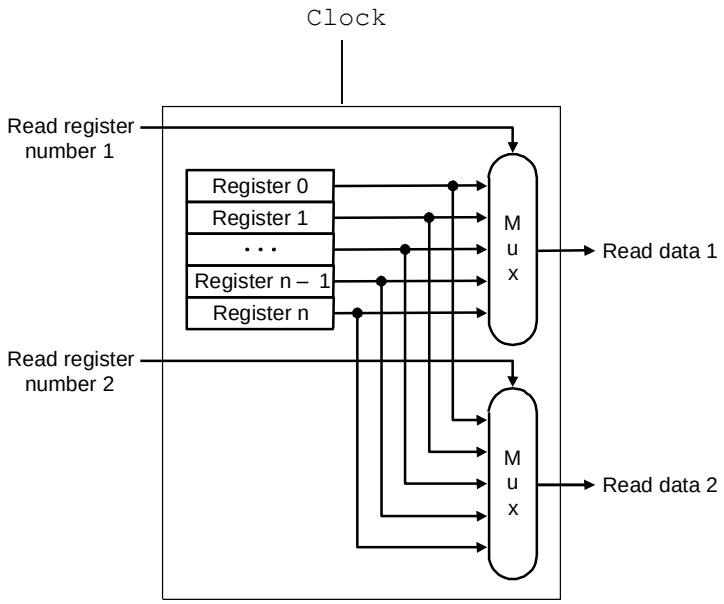
- Registerler D-flipflop'ların dizisi ile gerçekleştirilir.

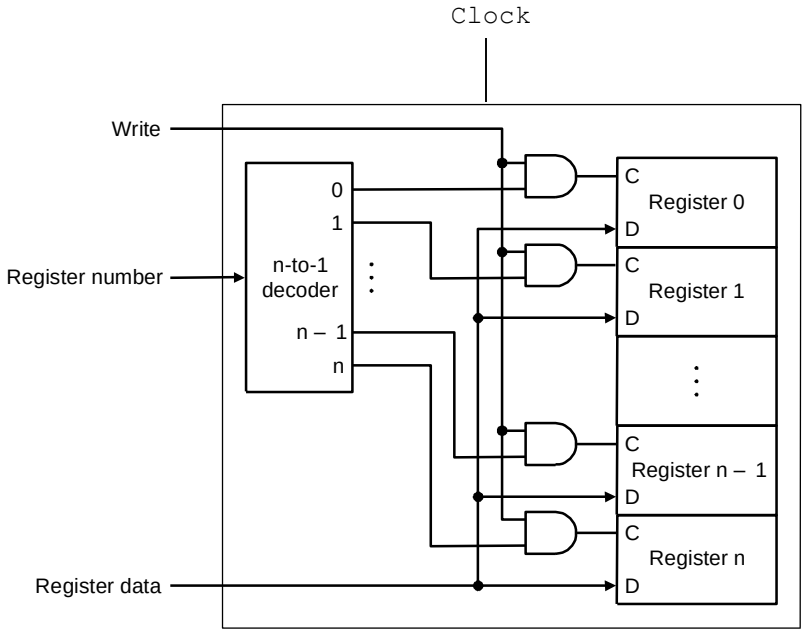


Register file iki okuma portu ve 1 yazma portuna sahiptir.

DATA PATH'de durum(state) elemanları: Register File

- Port gerçekleştirme:





32 register için oku portları 5 bit mux'ların çifti ile gerçekleştirilir.

Yazma portu bir decoder kullanılarak gerçekleştirilir. 32 register için 5'e 32'lik decoderdir. Yazma işlemi Clock ile ilişkilidir. (Yükselen veya düşen kenar)

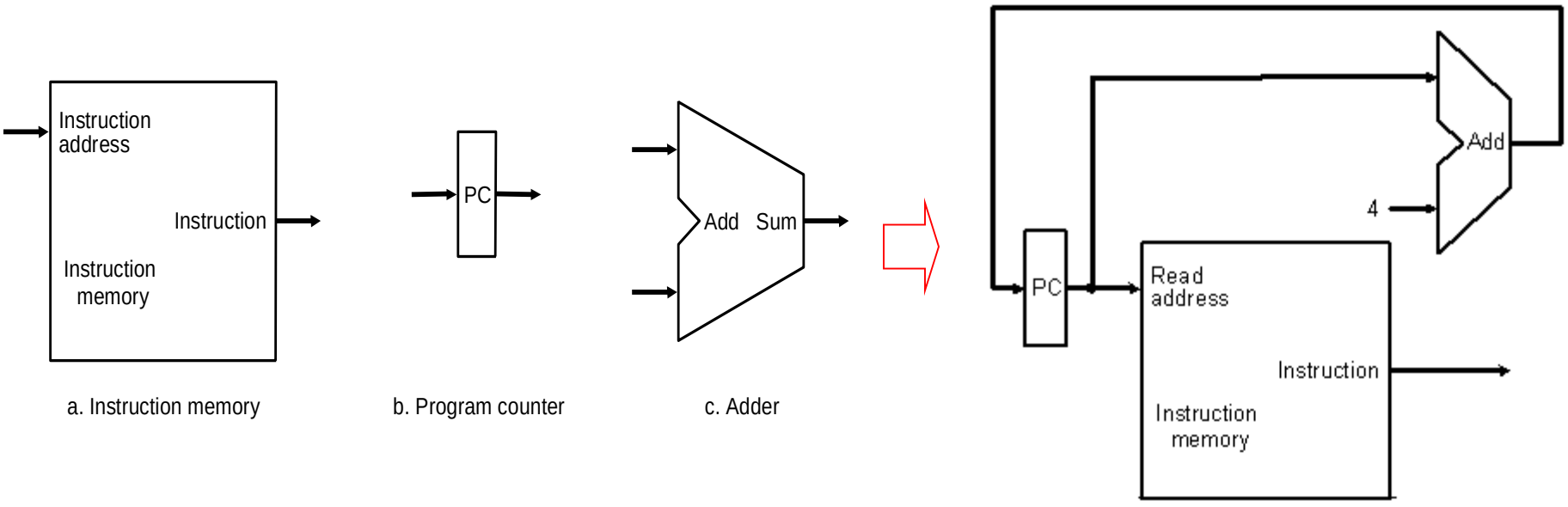
Verilog - VHDL

- Donanım tanımlama dili, Bütün donanımlar Verilog veya VHDL donanım tanımlama dili kullanılarak gerçekleştirilebilir.
- Detaylı bilgi ve örnekler için Verilog, VHDL slide'larına bakınız

MIPS'in tek Cycle'da gerçekleştirme

- MIPS komut seti mimarisinin ilk gerçekleştirilmesinde her bir komut için yeterince uzun tek bir clock cycle kullanacağız.
- Herbir komut yükselen veya düşen clock kenarı ile başlar ve yükselen veya düşen kenar ile sonlanır.
- Bu yaklaşım pratik değildir. Multicycle işlemeden daha yavaştır.Burada farklı komut sınıfları farklı cycle zamanları alabilir.
  - Herbir komut bir tek cycle gerçekleştirmede en yavaş komut kadar zaman alır.
  - multicycle gerçekleştirmede bir kaç cycle kullanılarak bu problemden kaçınılır.
- Tek cycle yaklaşımı pratik olmamasına rağmen konuyu anlamak için yararlı ve önemlidir.
- *Note* : we shall implement `jump` at the very end

Datapath: komut Store/Fetch & PC arttırma



a. Instruction memory

b. Program counter

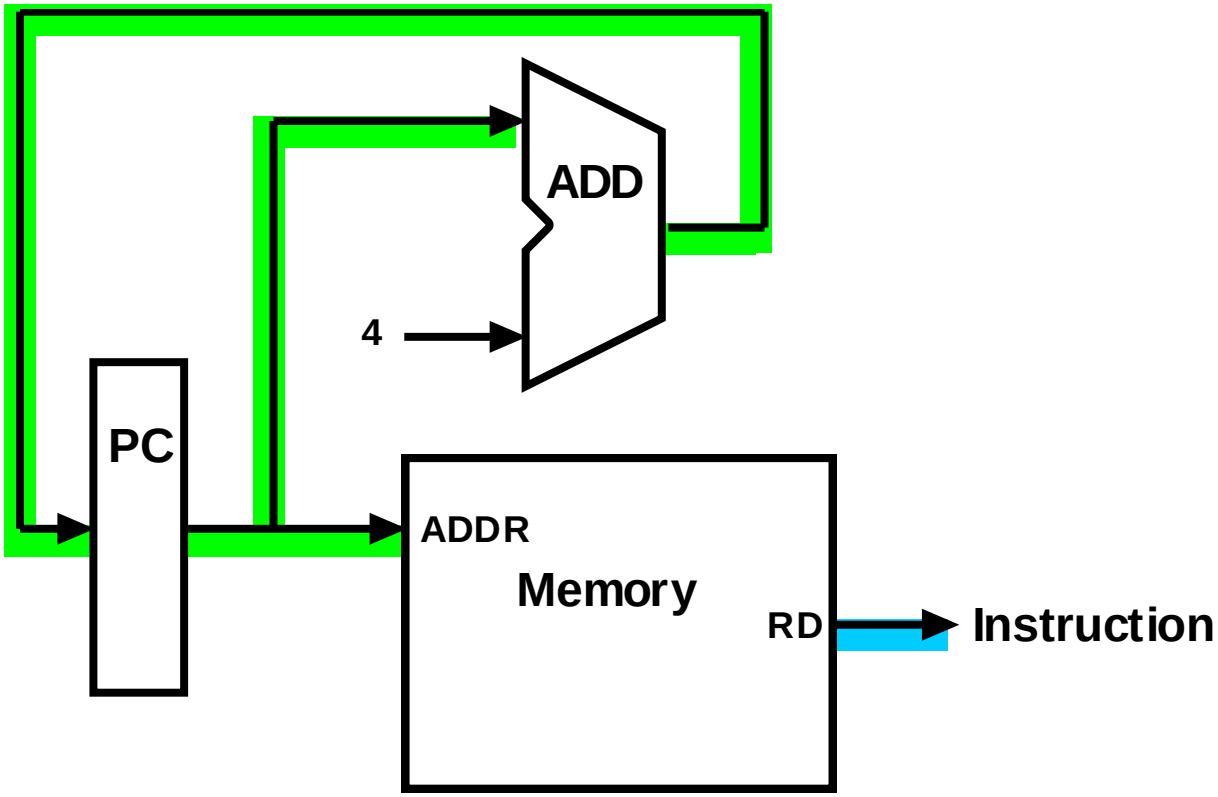
c. Adder

Fetch ve depolama ve PC' yi arttırmak için 3 elaman kullanılır

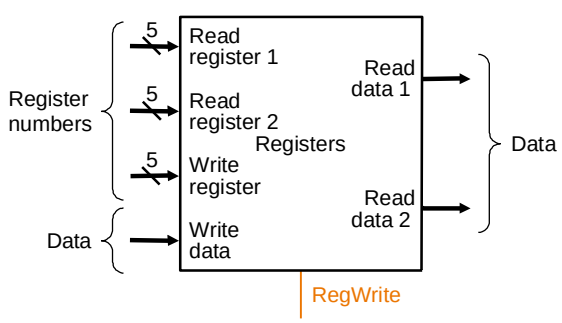
DATAPATH

Datapath Animasyonu

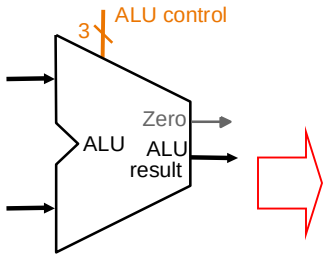
Instruction <- MEM[PC]
PC <- PC + 4



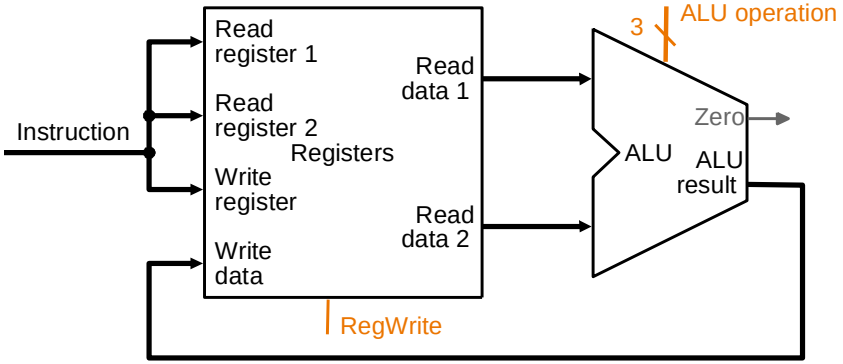
Datapath: R-Tipi Komut



a. Registers



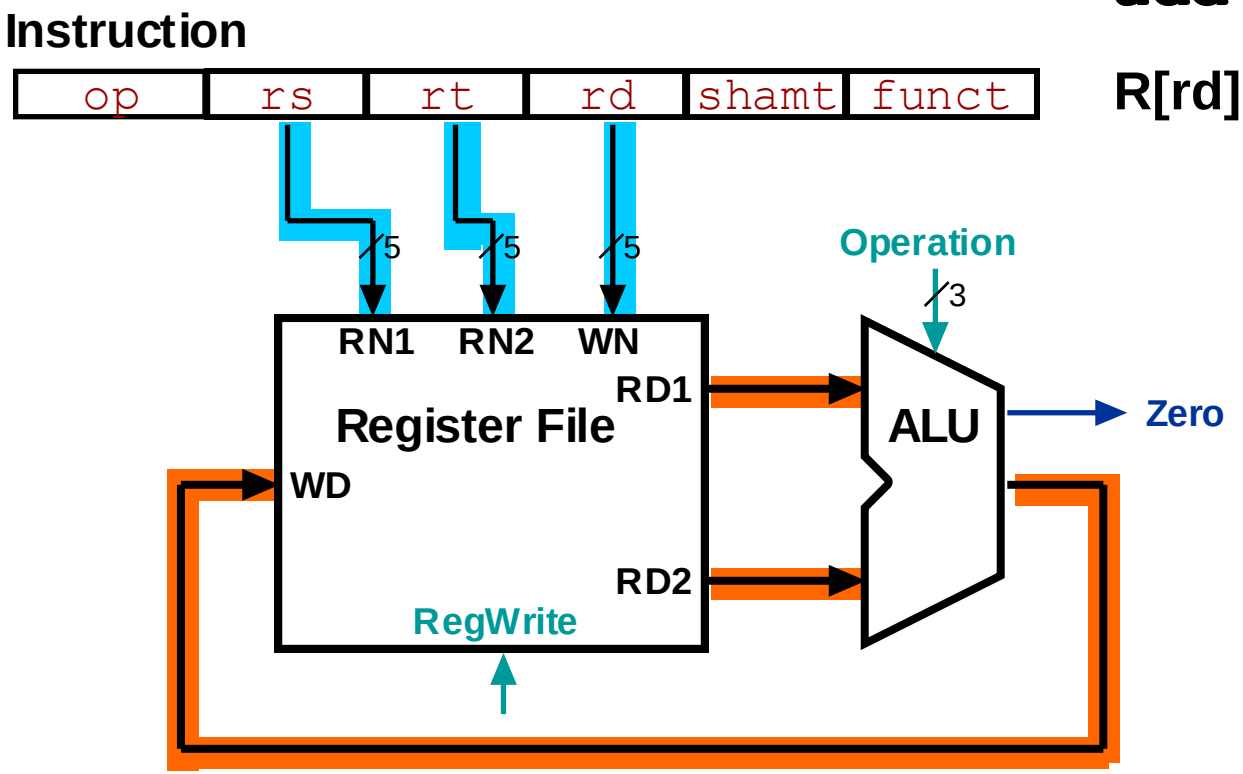
b. ALU



R-type komutları gerçekleştirmek için 2 eleman kullanıldı

Datapath

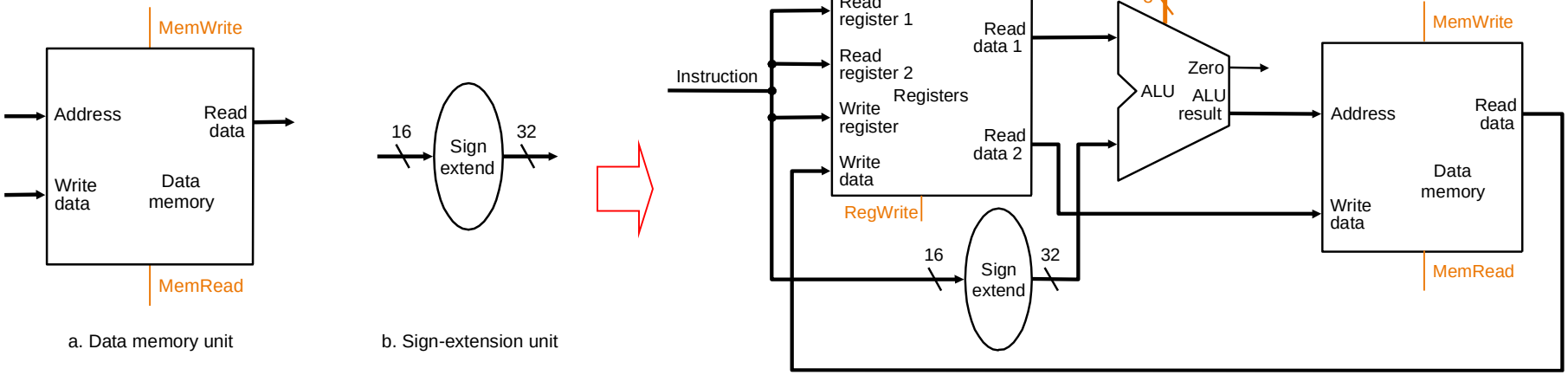
Datapath Animasyonu ...




```
add rd, rs, rt
```

R[rd] <- R[rs] + R[rt];

Datapath: Load/Store Komutu



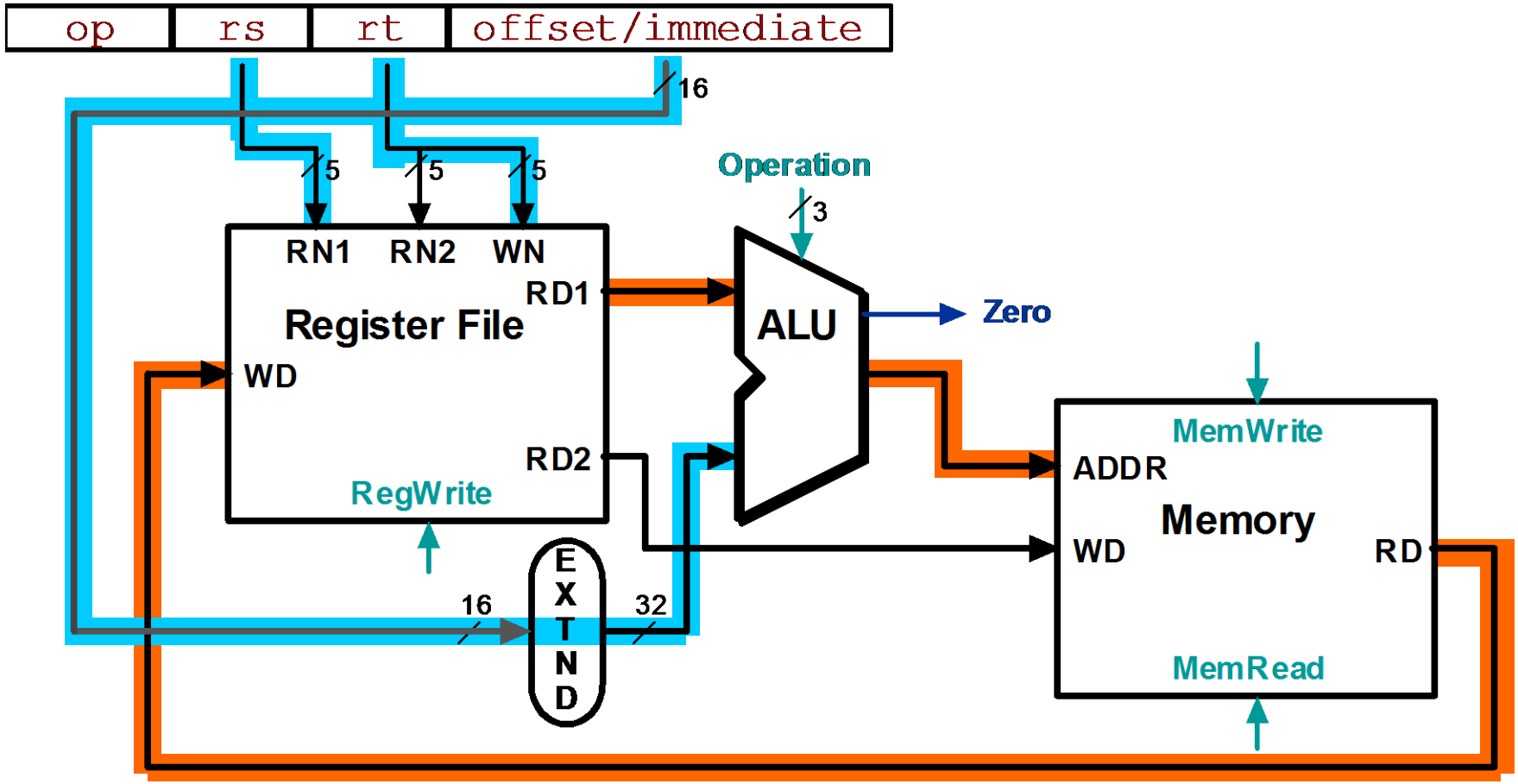
load/store komutunu gerçekleştirmek için İki ek eleman kullanıldı

Datapath

Datapath Animasyonu.....

```
lw rt, offset(rs)
```

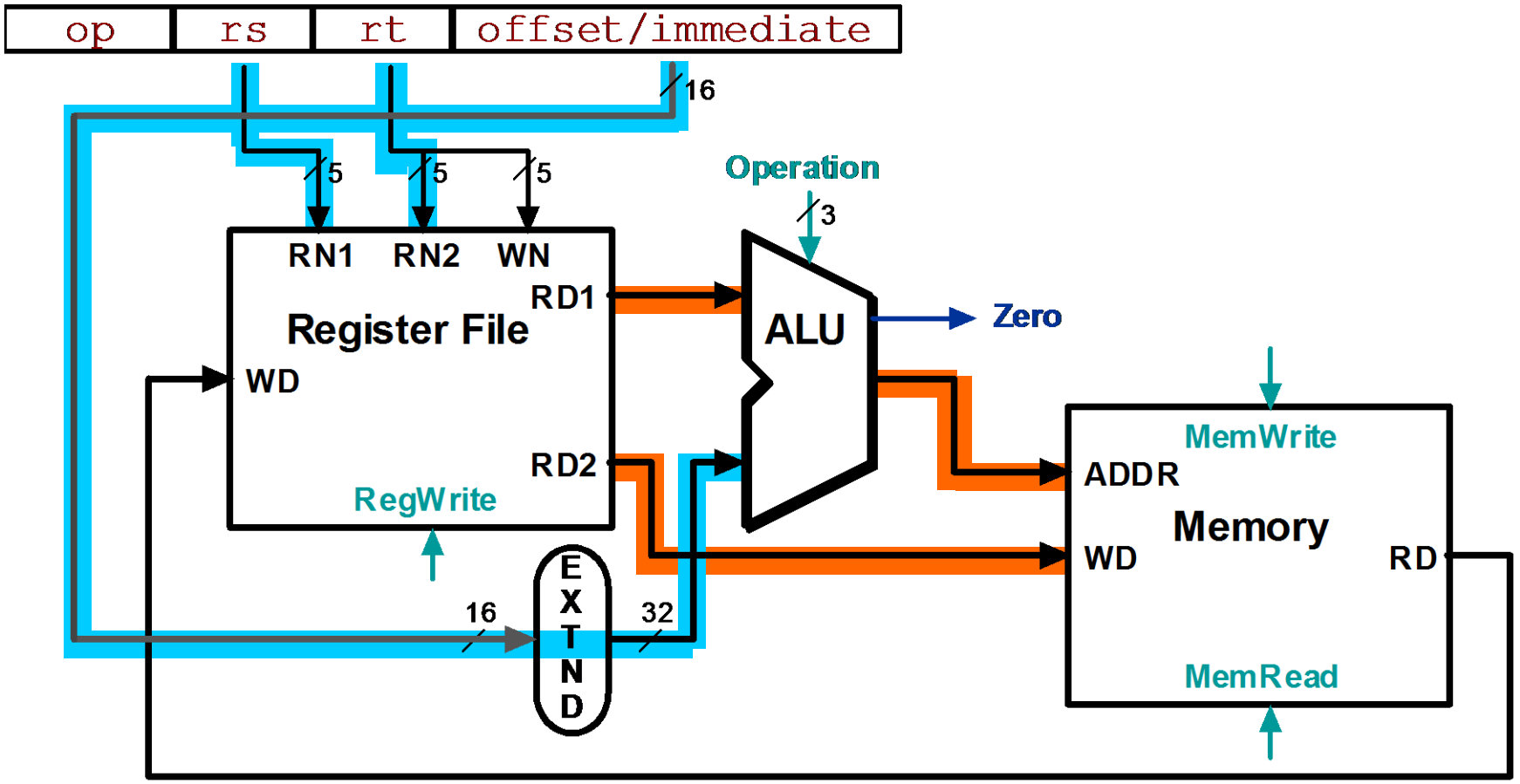
R[rt] <- MEM[R[rs] + s_extend(offset)];



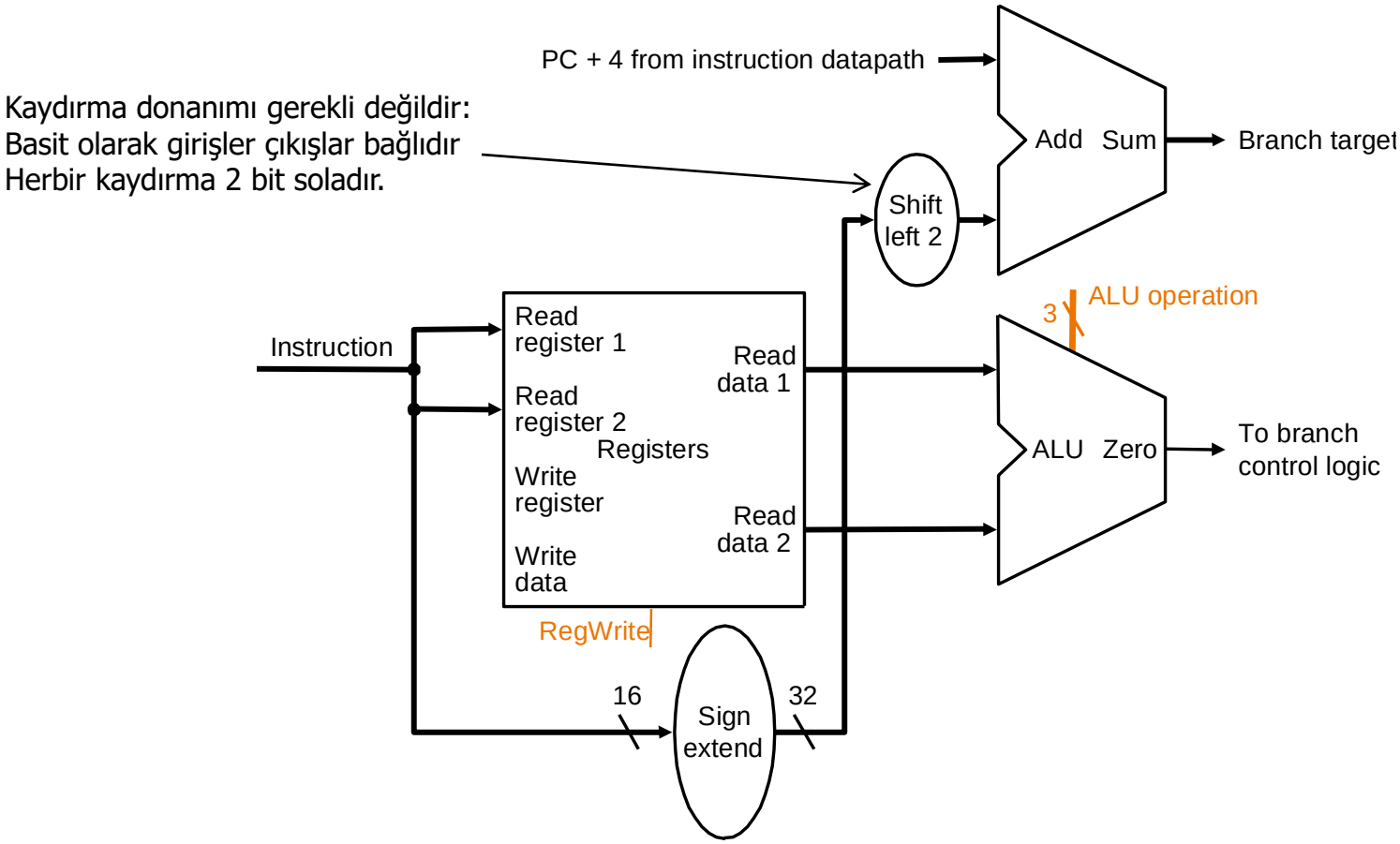
Datapath Animasyonu.....

```
sw rt, offset(rs)
```

MEM[R[rs] + sign_extend(offset)] <- R[rt]

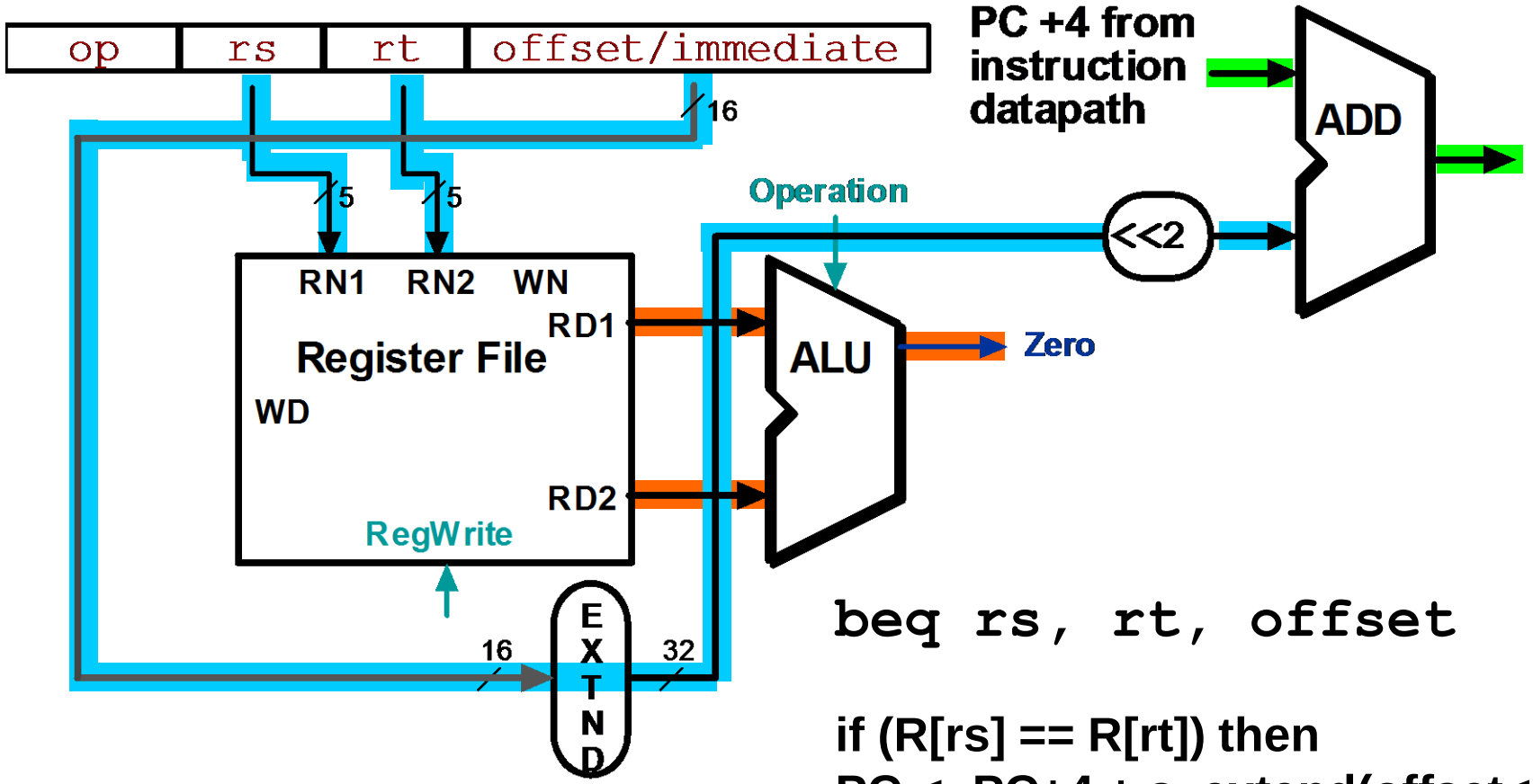


Datapath: Branch Instruction (Dallanma komutları)



Datapath

Animating the Datapath

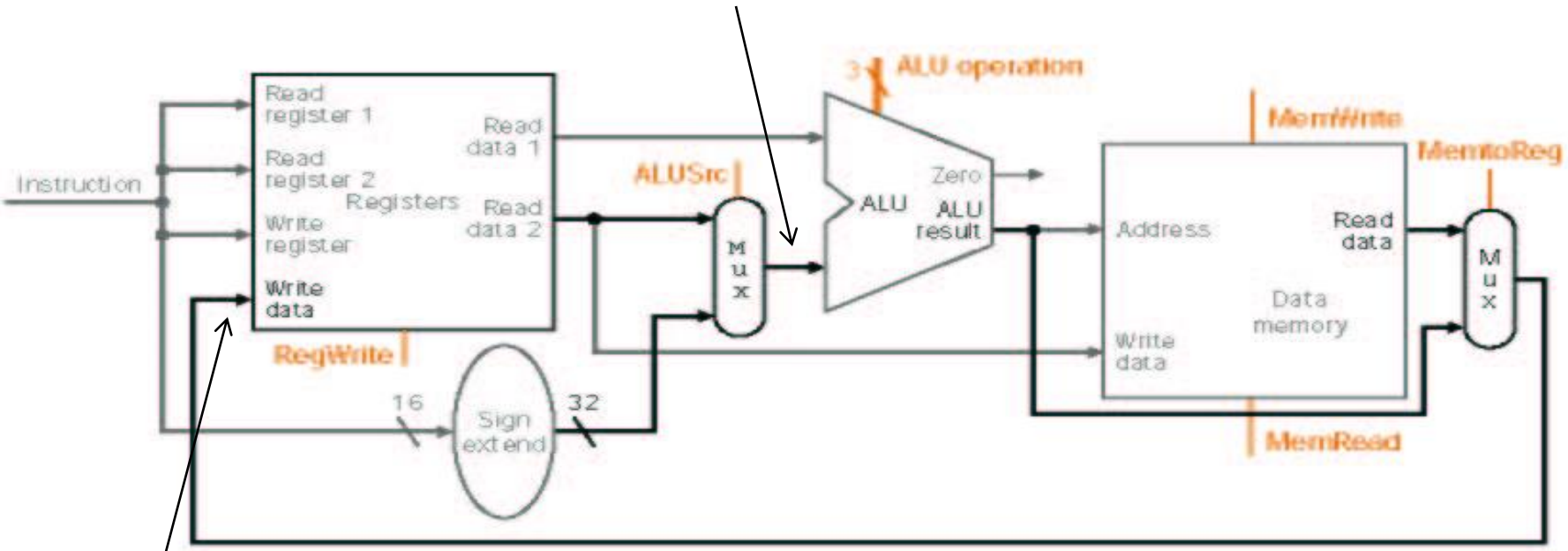



```
beq rs, rt, offset
```

if (R[rs] == R[rt]) then
PC <- PC+4 + s_extend(offset<<2)

MIPS Veriyolu I: Single-Cycle

Giriş ya register (R-type) yada komutun alt yarısı

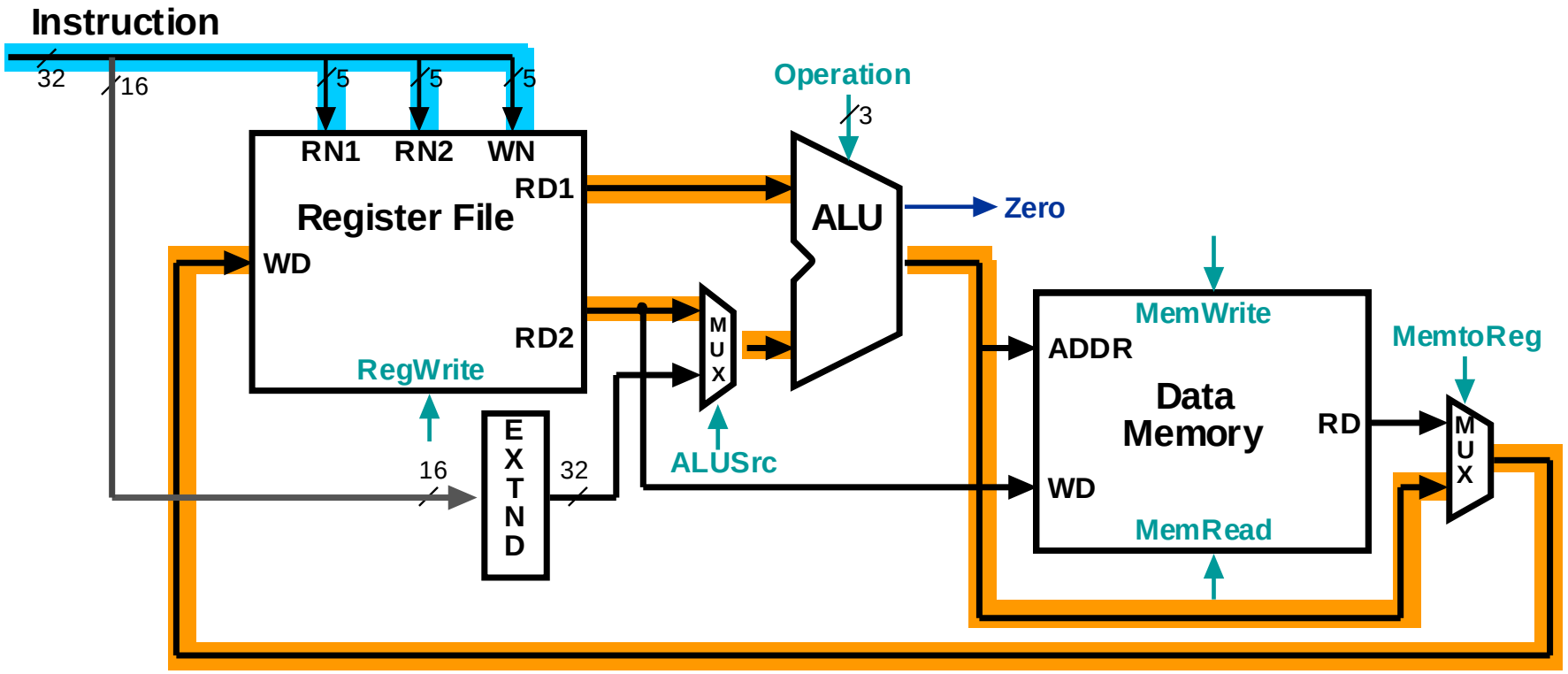


Veri ya ALU'dan (R-type)
Yada hafızadan (load)

R-tip komutlar ve load/store için veriyolu oluşturmak için 2 mux kullanılır

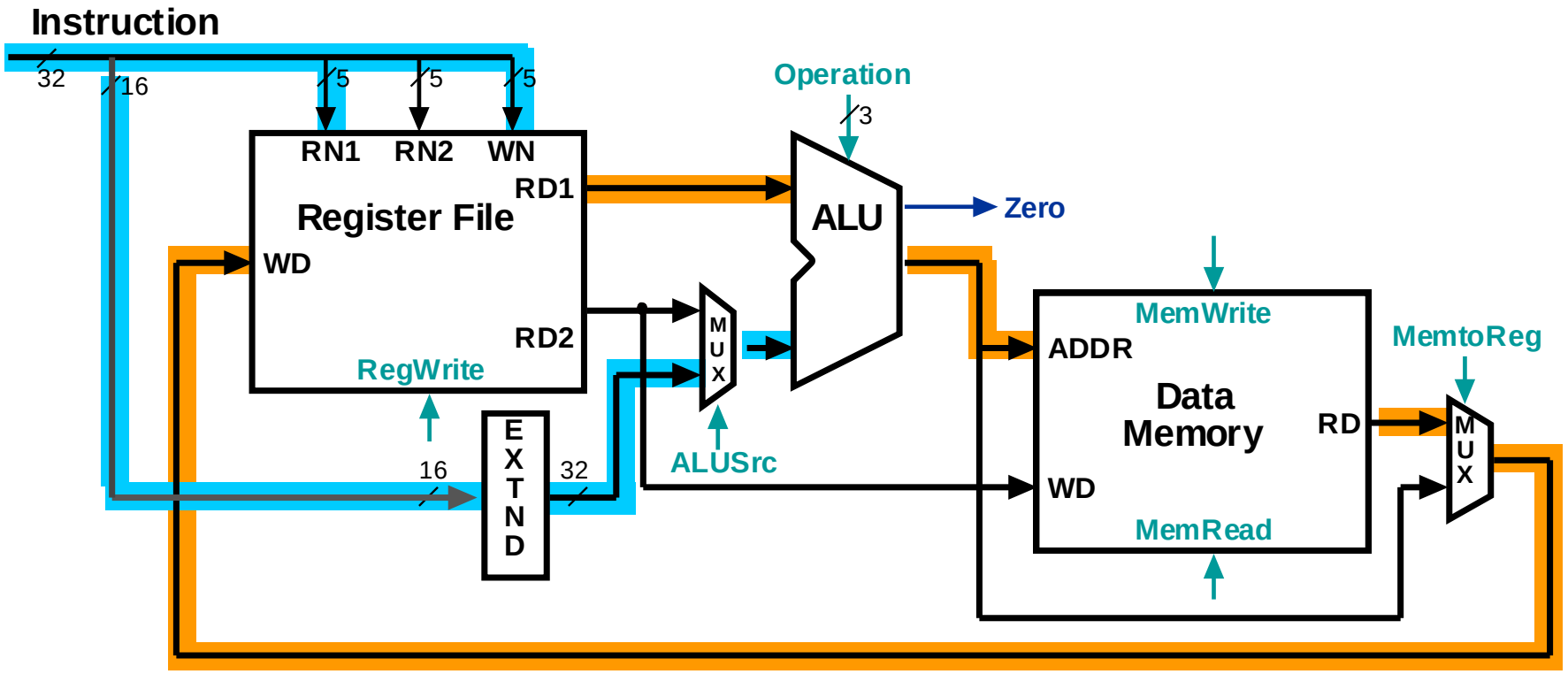
R-tipi komutun çalışması

```
add rd,rs,rt
```

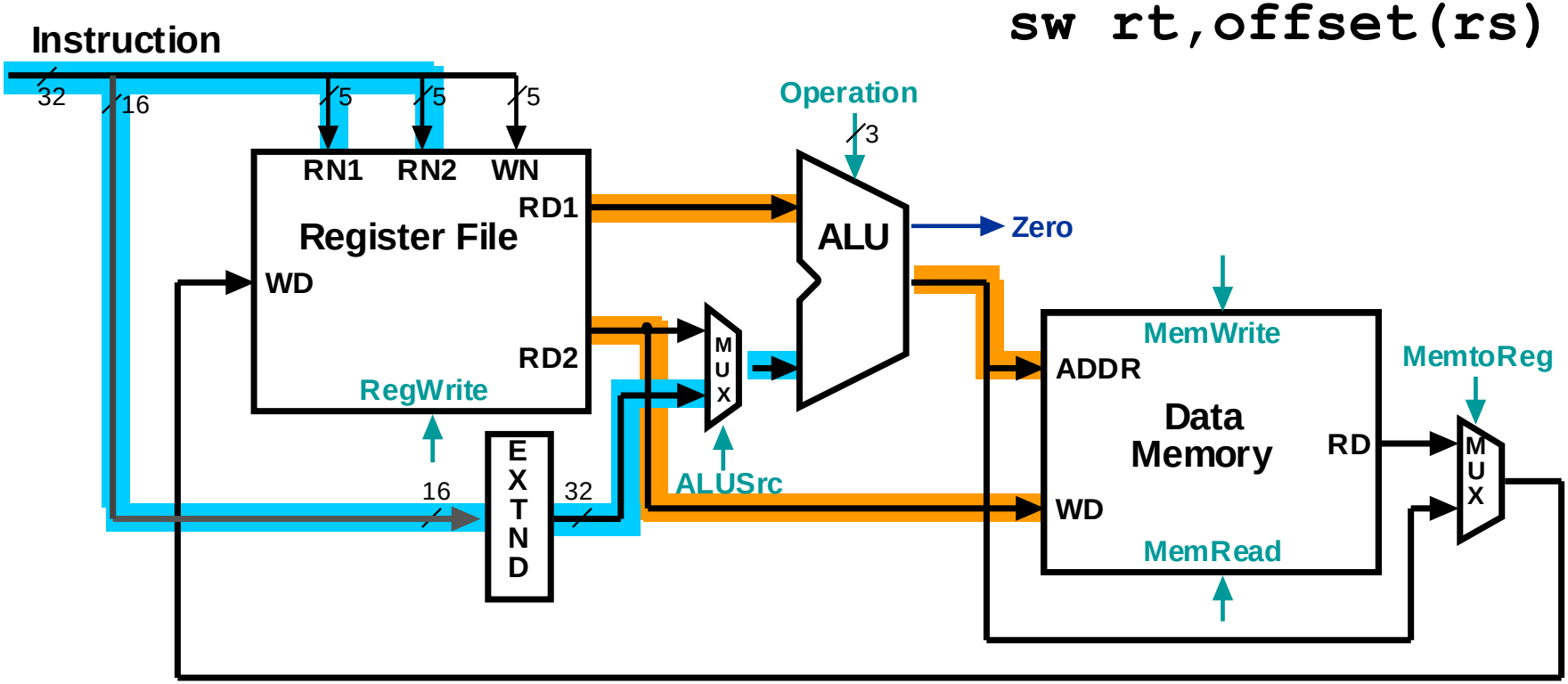


Load Komutunun Çalışması

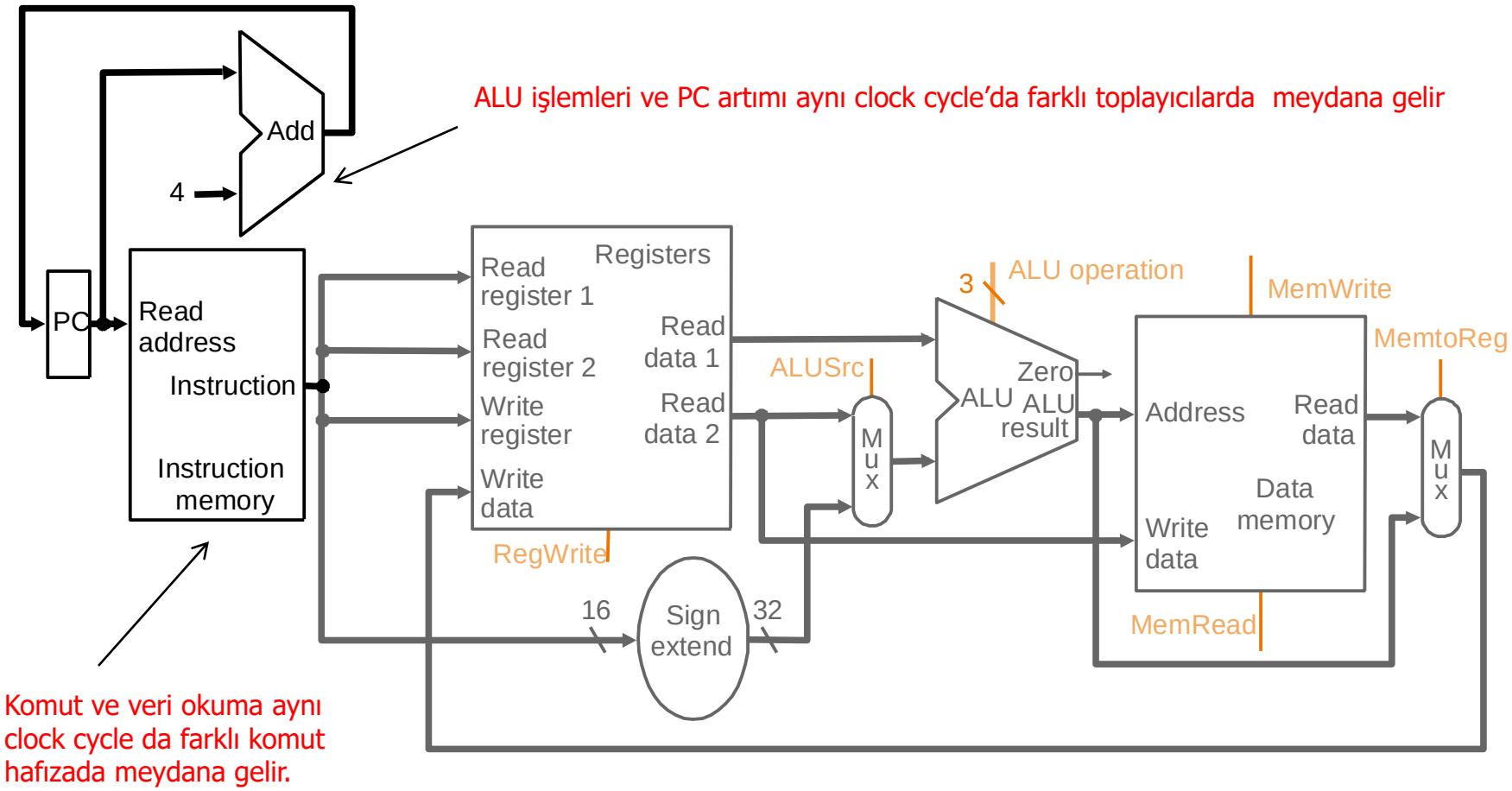
```
lw rt,offset(rs)
```



Store komutunun Çalışması

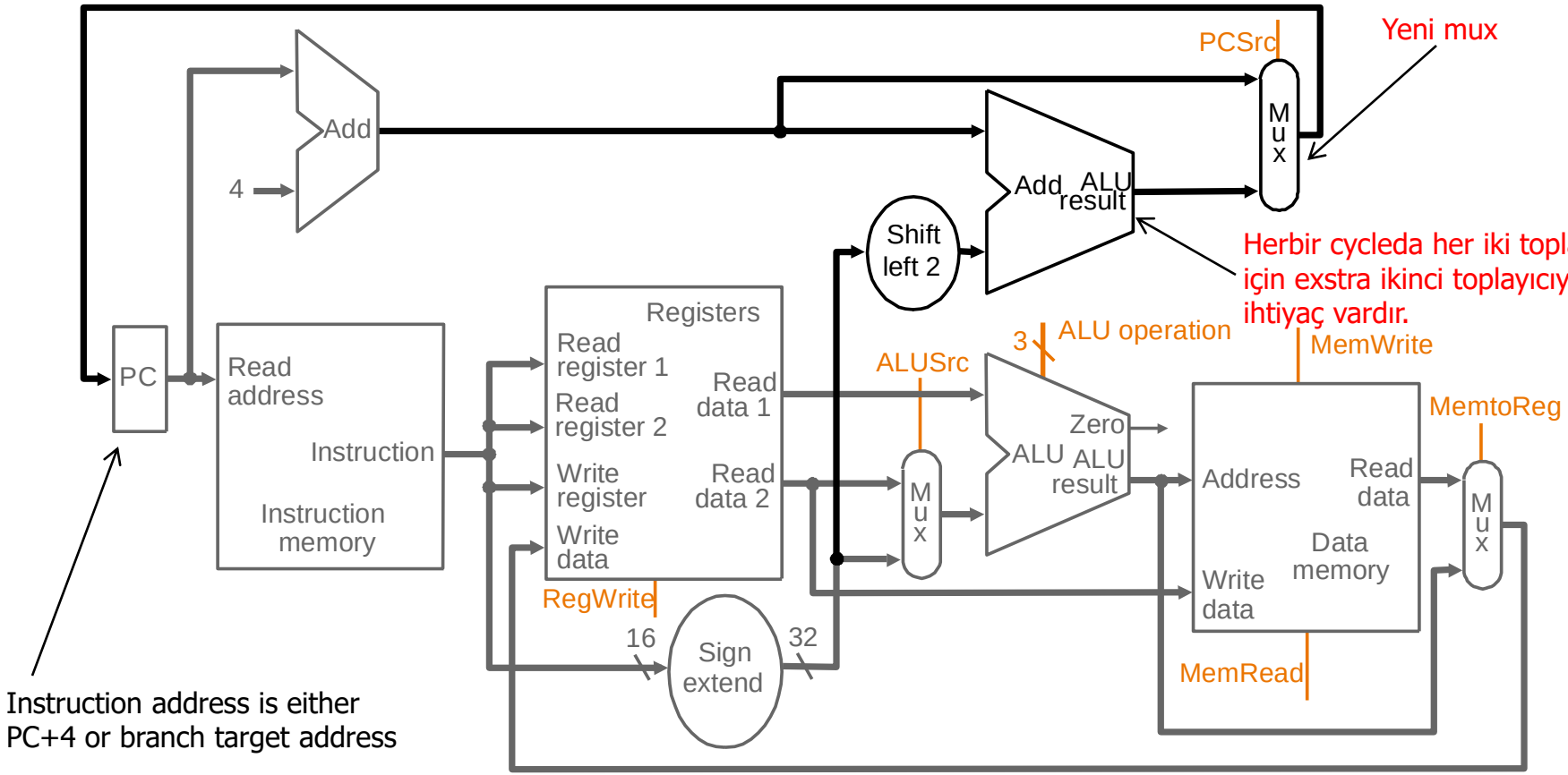


MIPS Datapath II: Single-Cycle



komut fetch'i ekleme

MIPS DATA PATH III: Single-Cycle

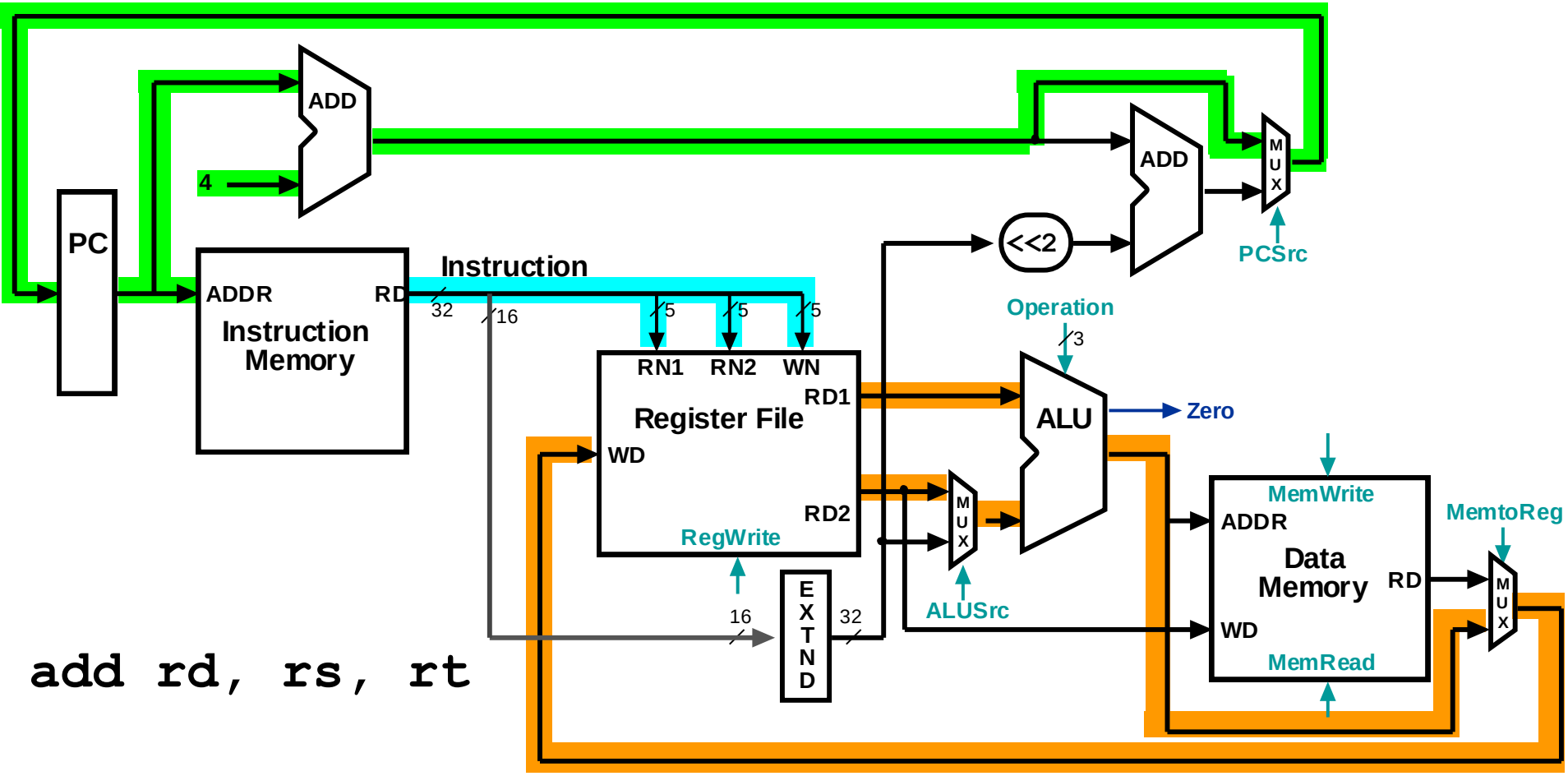


dallanma yeteneğinin ve diğer mux'un eklenmesi

Önemli not: single-cycle gerçekleştirmede veri komut boyunca depolanamaz. Veri yalnızca kombinasyonal lojik yoluyla taşınır

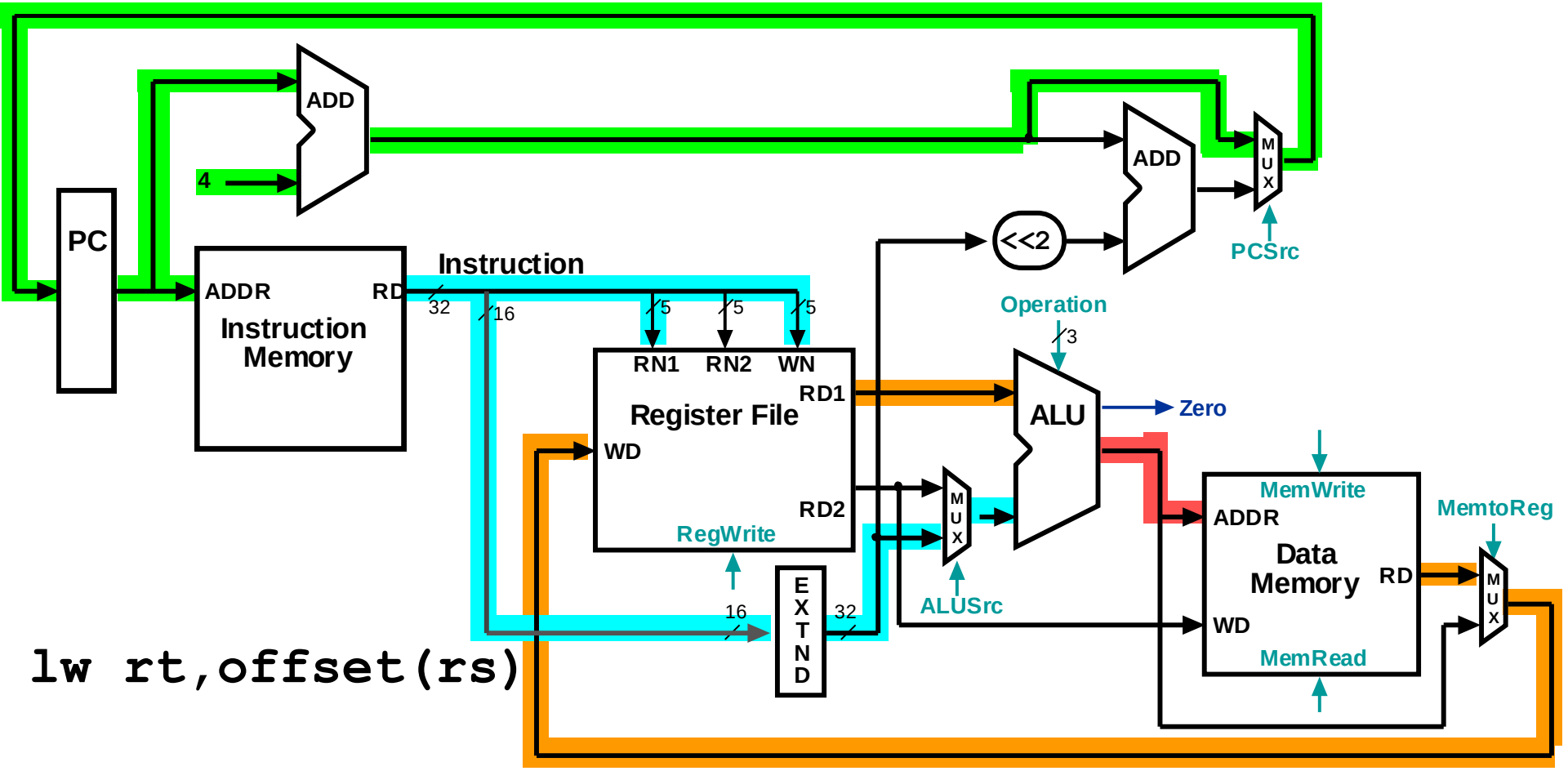
Soru: Memread sinyaline gerçekten ihtiyaç varmıdır. RegWrite' ı düşünün

DATAPATH: Executing `add`



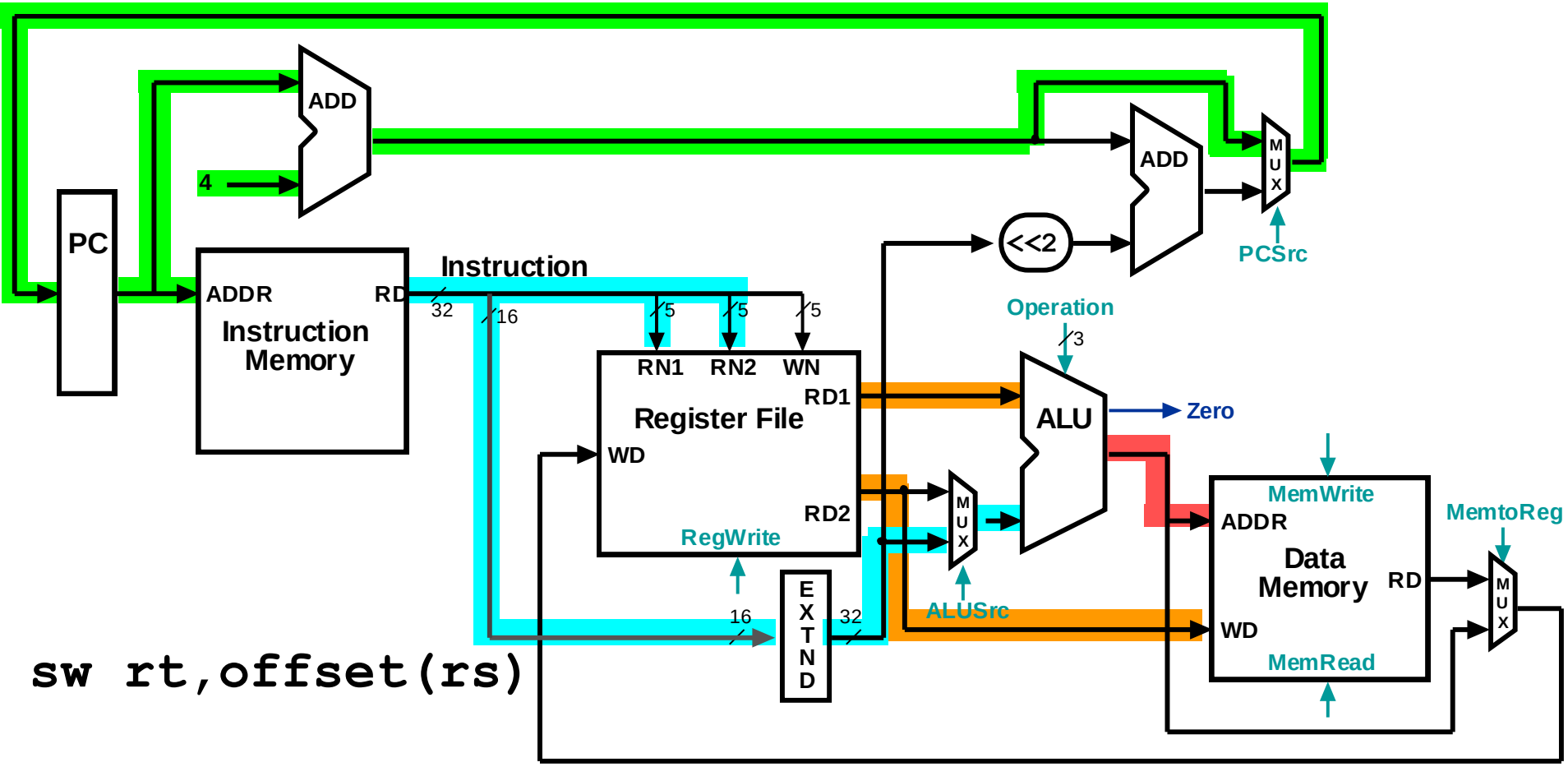

```
add rd, rs, rt
```

DATAPATH Executing `lw`



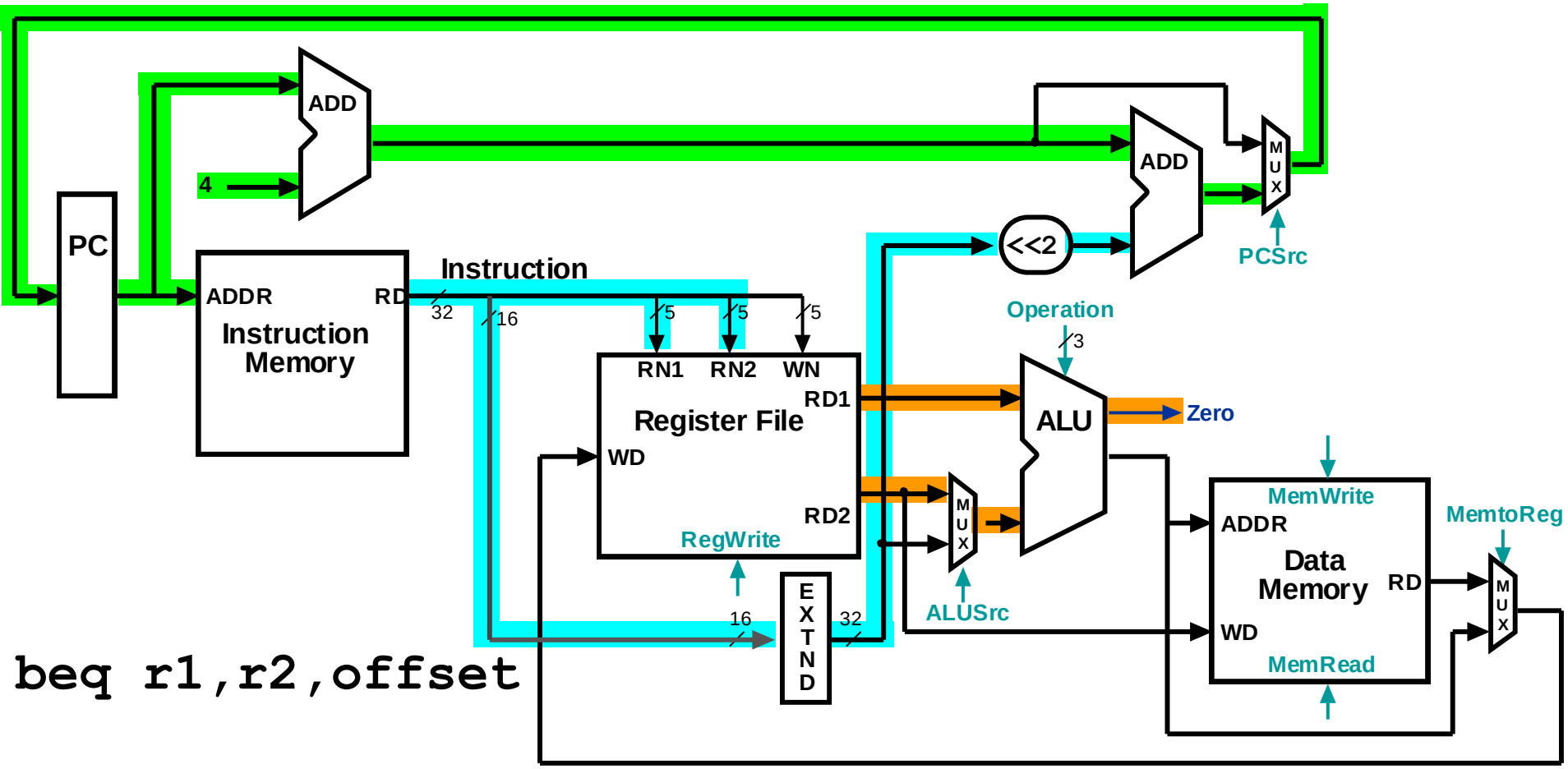

```
lw rt,offset(rs)
```

DATAPATH Executing `sw`




```
sw rt,offset(rs)
```

DATAPATH Executing beq



beq r1,r2,offset

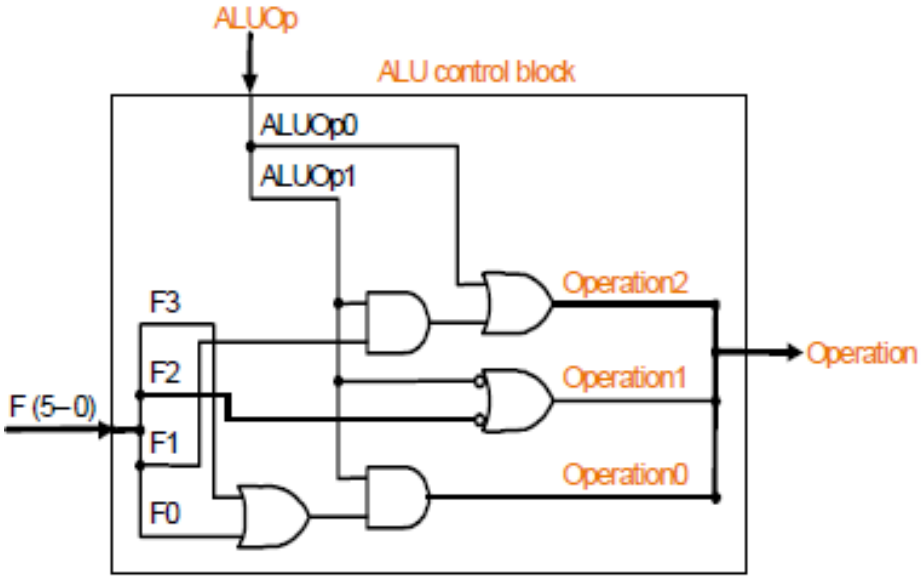
DATA PATH, Kontrol

- *Ana Kontrol biriminin girişleri;* Komut kodu formatındaki opcode bitleri (6 bit)'dir.

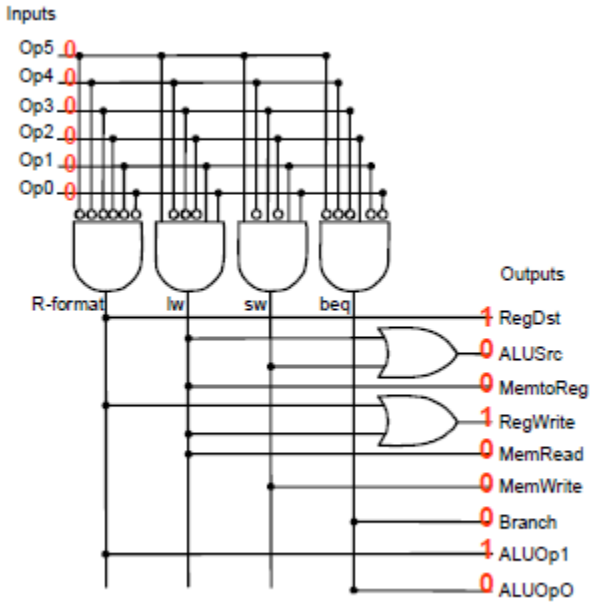
- *Ana Kontrol birimi aşağıdaki çıkışları üretir.*
  - ALU kontrol girişleri (2 bit input)
  - Yazma enable (olası olarak, okuma enable) her bir depolama elemanları için sinyal.
  - Herbir mux için seçme kontrolü

Kontrol Birimleri

ALU Kontrol Bloku



Ana Kontrol Birimi devre şeması

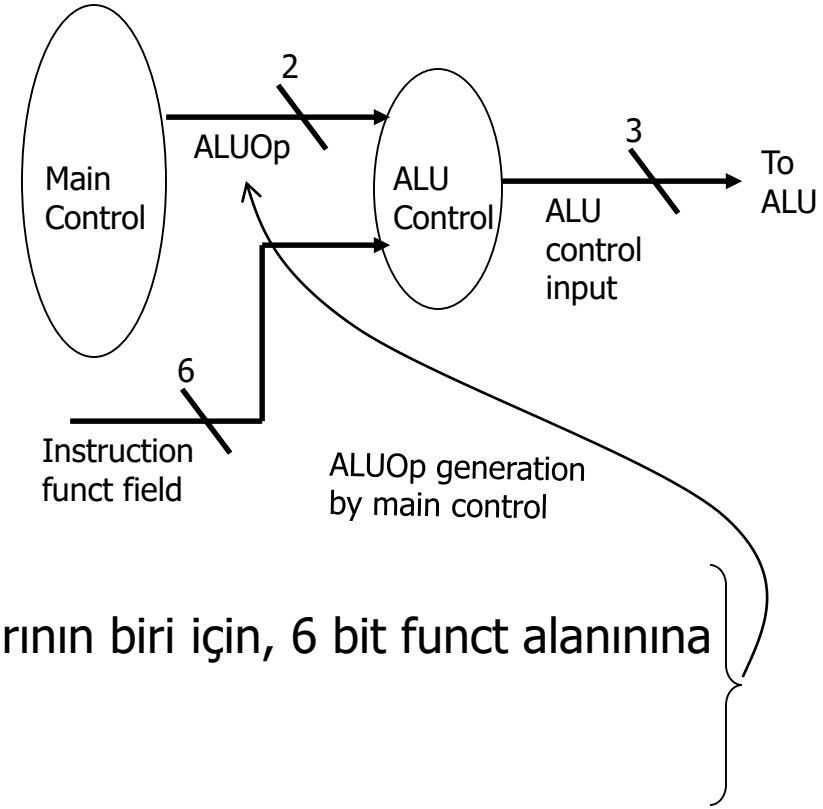


ALU Kontrol

- ALU kontrol planı: Ana kontrol birimi; ALU kontrol birimi girişine iki bit (ALUop kontrol) gönderir. ALU kontrol birimi girişine ayrıca komutun 6 bitlik *funct* bitleri de uygulanır. Çıkışta, ALU kontrolu için 3 bitli oluşur.

Recall from Ch. 4

ALU kontrol alanı	Fonksiyon
000	and
001	or
010	add
110	sub
111	slt



- ALU icra etmelidir.
  - load/stores için add (ALUOp 00)
  - *Dallanma için sub* (ALUOp 01)
  - *and*, *or*, *add*, *sub*, *slt* R-type komutlarının biri için, 6 bit funct alanınına bağlıdır. (ALUOp 10)

ALU kontrol bit ayarları

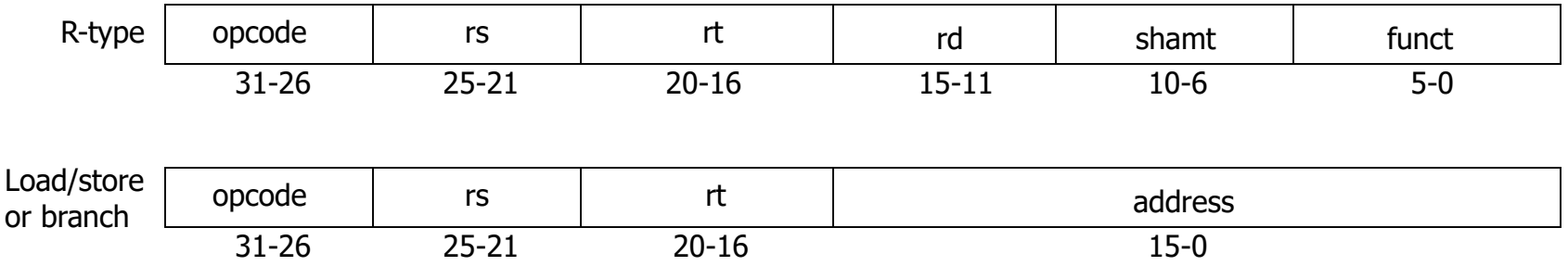
komut opcode	AluOp	komut işlemi	Funct Alanı	İstenen ALU etkisi	ALU girişi
LW	00	load word	xxxxxx	add	010
SW	00	store word	xxxxxx	add	010
Branch eq	01	branch eq	xxxxxx	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110
R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	set on less	101010	set on less	111

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0*	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

ALU kontrol bitleri için doğruluk tablosu

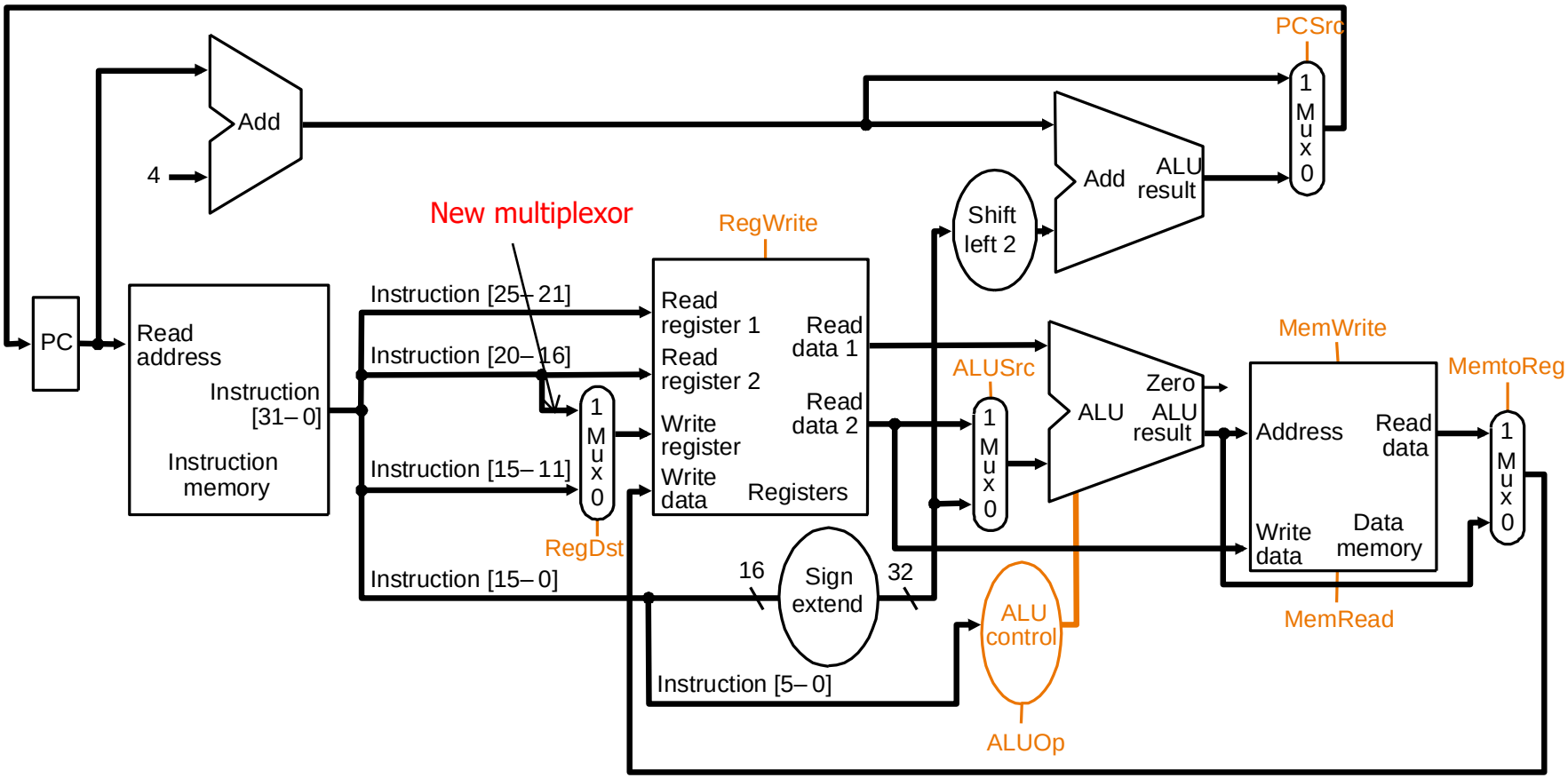
*, eğer X olursa, ozaman satır 2 ile 3-7 arası çatışma Vardır.

Ana kontrol Birimi tasarımı



- MIPS komut formatları hakkında gözlemler
  - opcode daima 31-26 bitlerindedir.
  - Okunabilen 2 register daima rs (bits 25-21) ve rt (bits 20-16) dir.
  - load/stores için base register daima rs (bits 25-21) dir.
  - Dallanma ve load/store için 16-bit offset daima 15-0 bitleridir.
  - Load için hedef register 20-16 (rt) bitlerindedir. R-type komutlarda 15-11 (rd) bitleridir. (bunun için mux ile seçmek gereklidir)

DATA PATH ve kontrol I



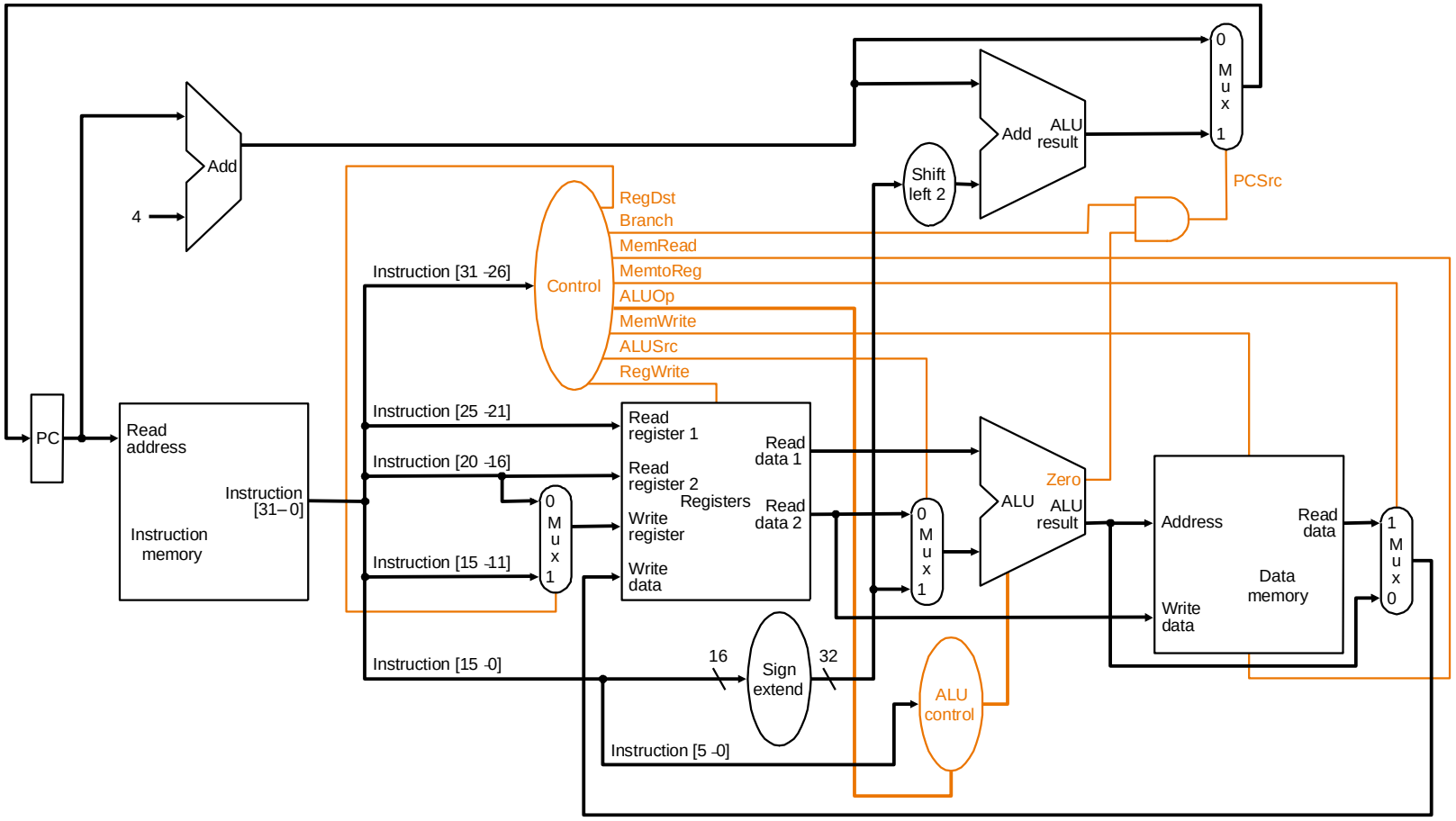
MIPS DATAPATH III'e kontrol ekleme (ve özel hedef register için alan seçmek için yeni bir mux): 9 kontrol sinyallerinin fonksiyonları nedir?

Kontrol sinyalleri

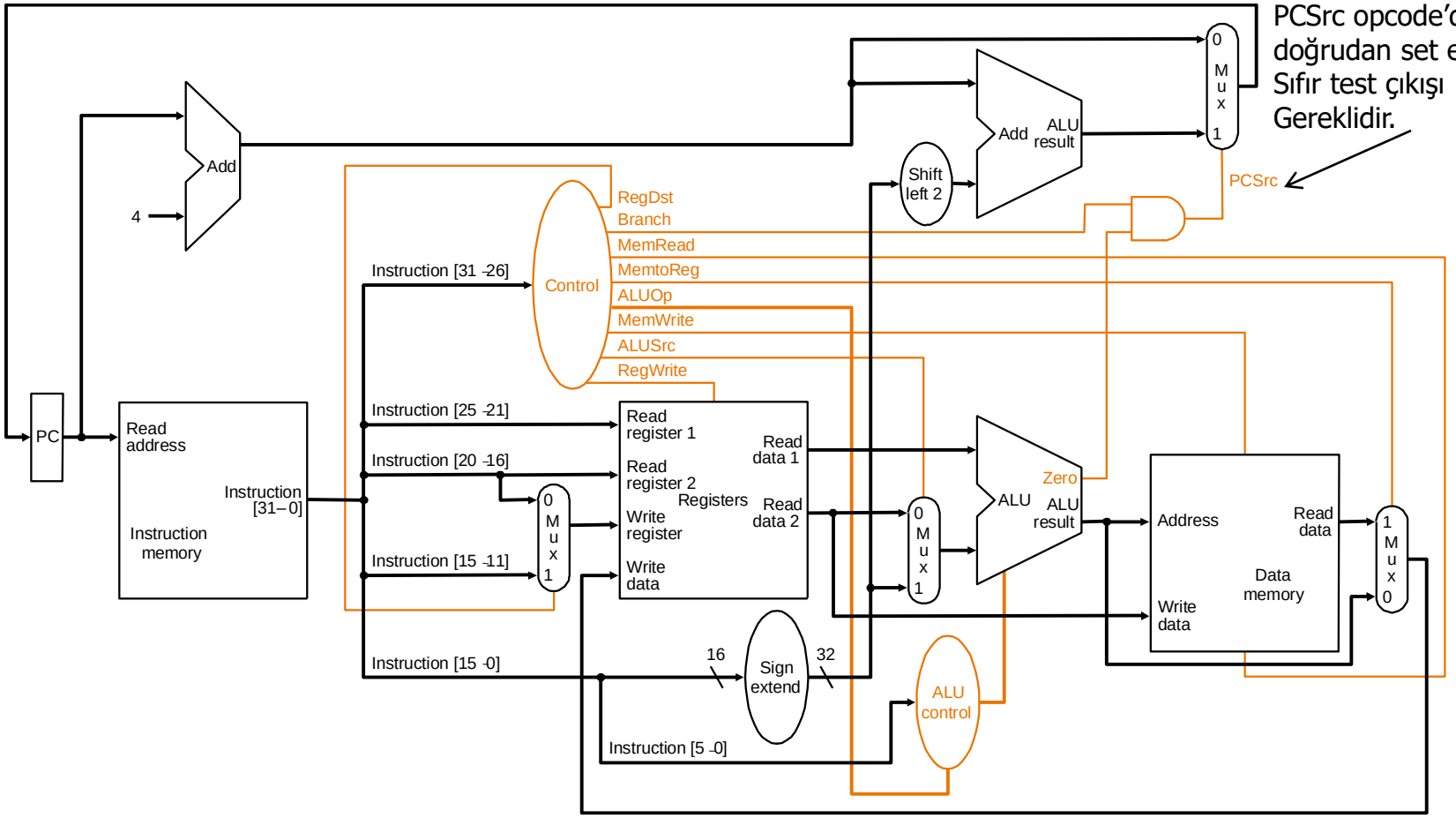
Signal Name	Effect when deasserted (Lojik 0)	Effect when asserted (Lojik 1)
RegDst	yazma registeri için hedef register sayısı rt alanı (bits 20-16)	yazma registeri için hedef register sayısı rd alanı (bits 15-11)
RegWrite	None	*Write register* girişindeki datayı, hedef reg'e yazar.
ALUSrc	ikinci ALU operandı ikinci register file çıkışından gelir(Read data 2)	İkinci ALU operandı is the sign-extended, komutun düşük 16 bitine sign-extended dir
PCSrc	PC toplayıcının çıkışı tarafından yer değiştirilir. bu PC + 4 ile hesaplanır	PC Toplatıcının çıkışı tarafından yer değiştirilir. bu hedef dallanmayla hesaplanır.
MemRead	None	Data memory içeriği giriş adresine göre belirlenir bu giriş ilk ilk Read data çıkışına konur
MemWrite	None	Data memory içeriği giriş adresine göre belirlenir bu giriş Write data ğirişinin değeri tarafından değiştirilir.
MemtoReg	ALU'dan gelen değeri Write data giriş registerine yaz	Data memory'den gelen değeri Write data giriş registerine yaz

Yedi kontrol sinyalinin etkileri

Datapath with Control II



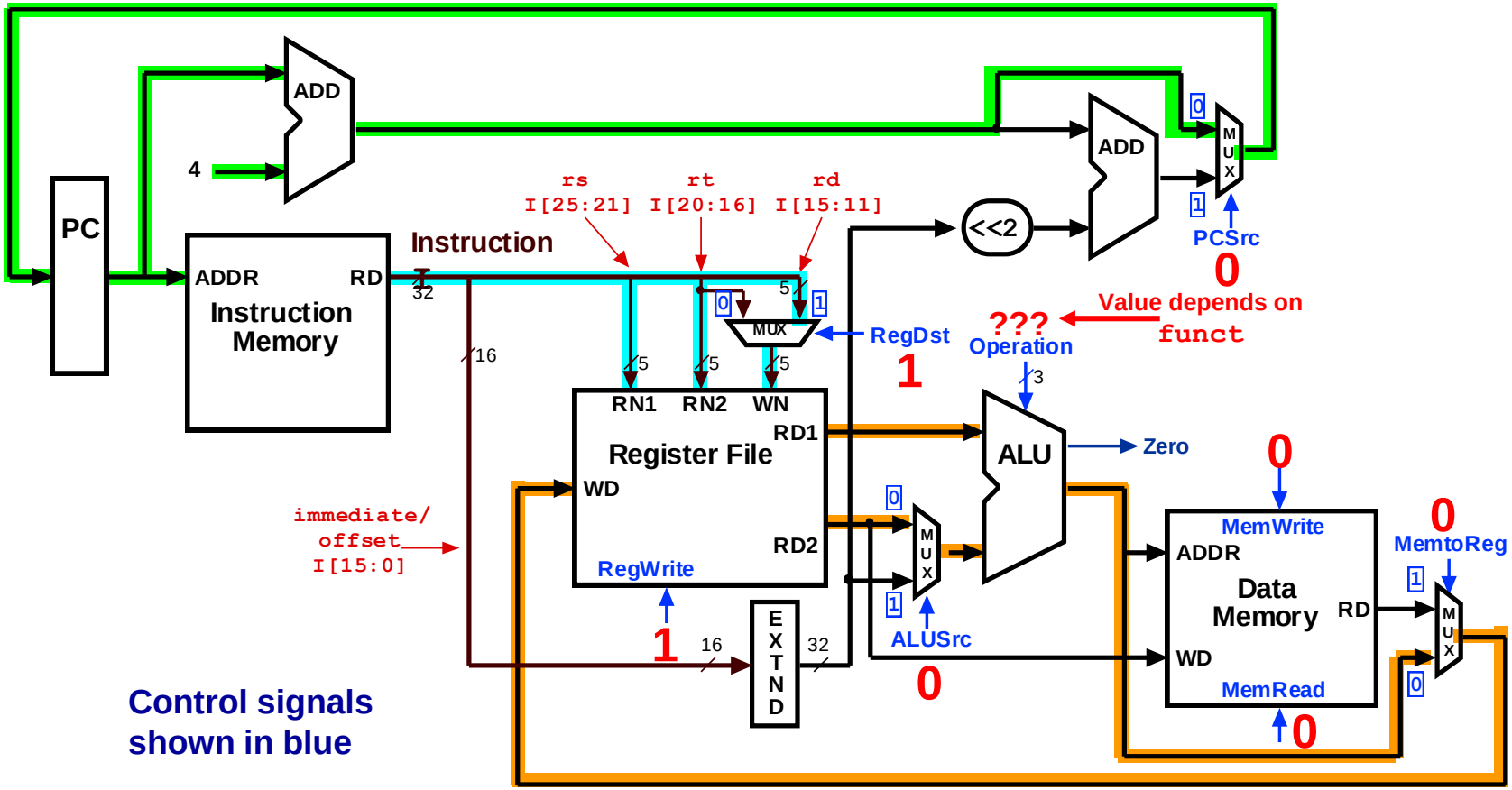
Kontrol birimi ile MIPS Veri yolu: Kontrol girişi 6-bit komut opcode alanıdır. Çıkış 7, 1-bit signal ve 2-bit ALUOp sinyalidir.



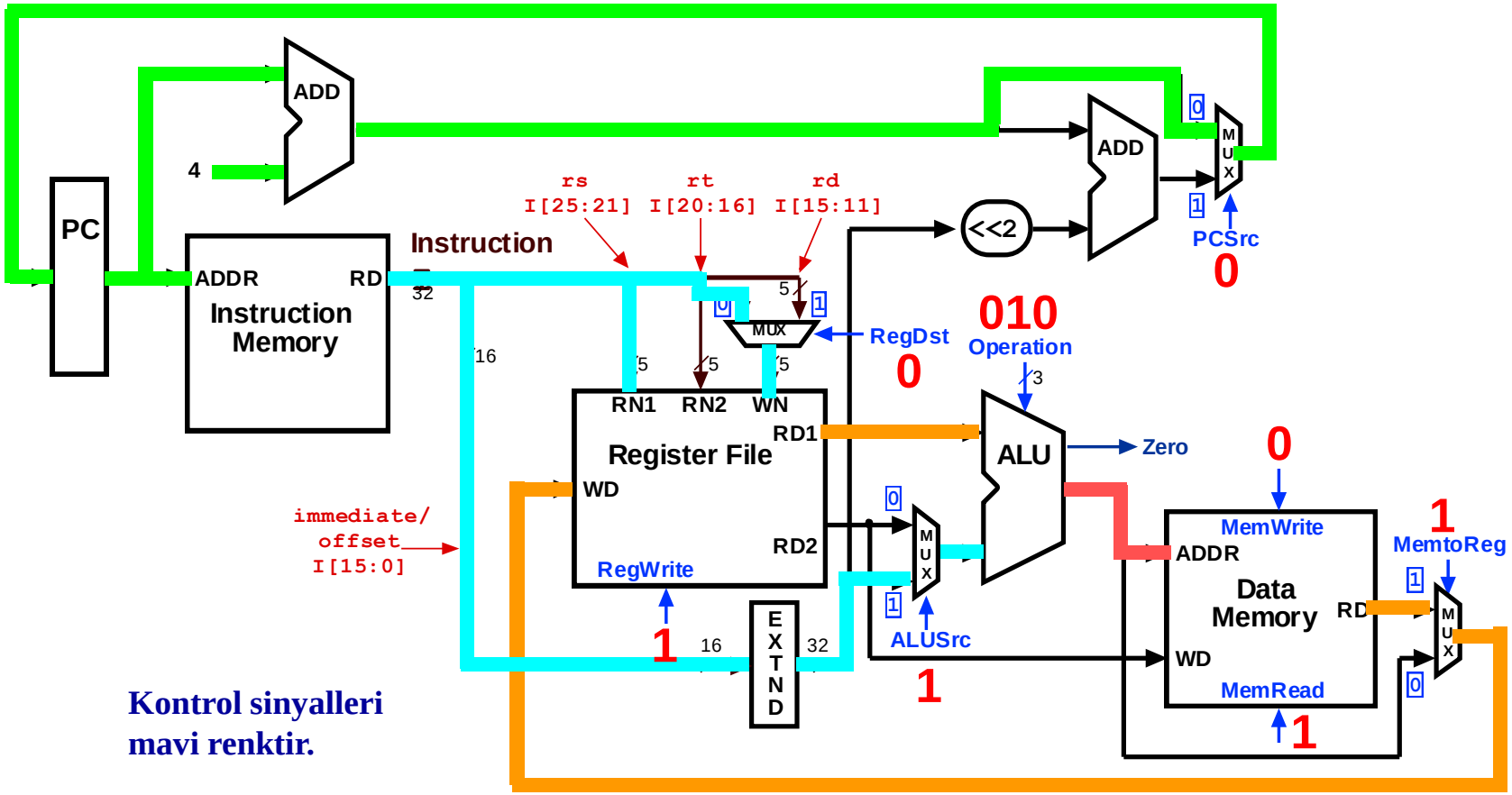
Komut opcode tabanlı MIPS veriyolu için kontrol sinyalleri hesaplama

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

Control Signals: R-Type Instruction

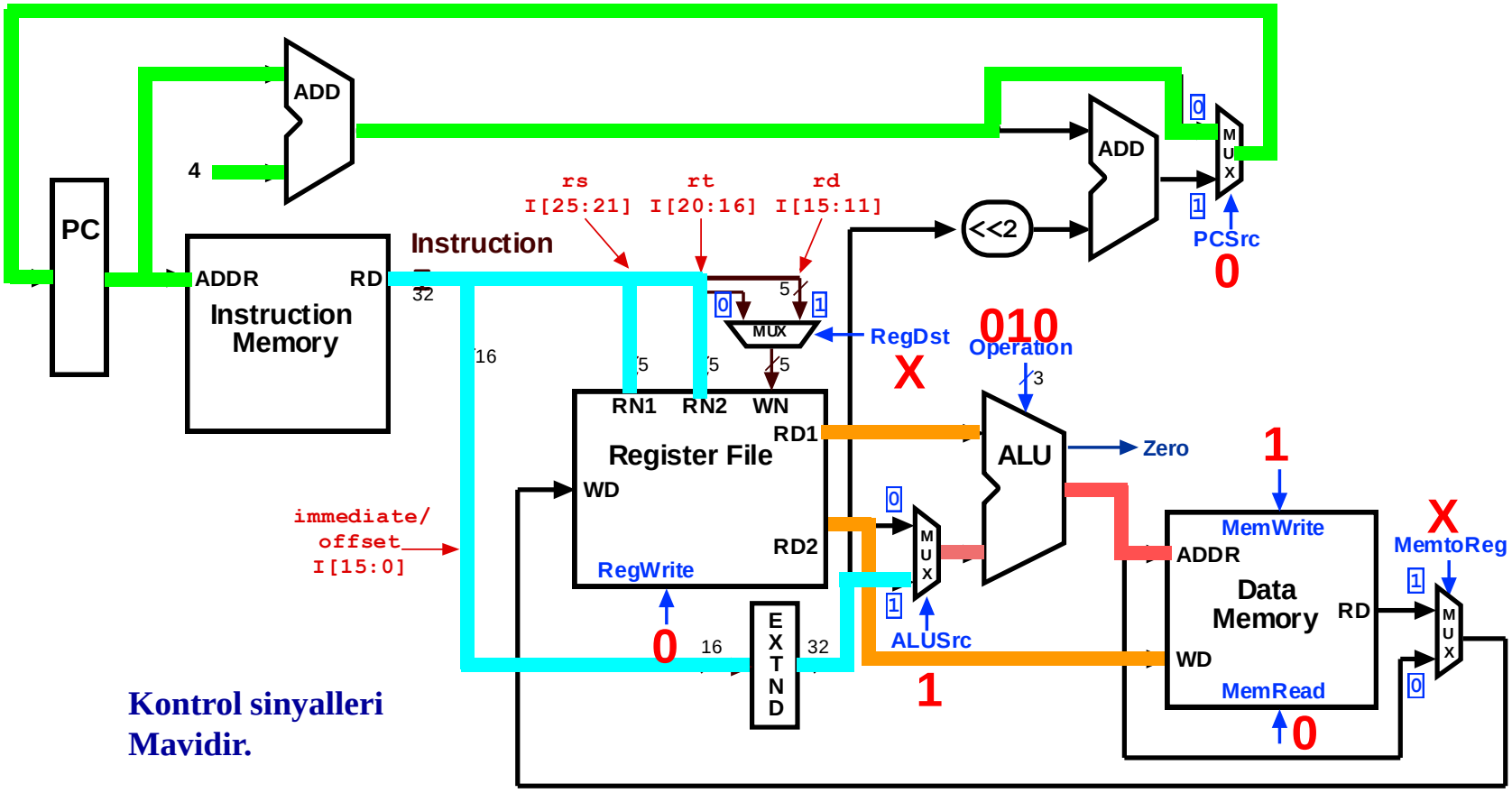


Kontrol sinyalleri: `lw` komutu



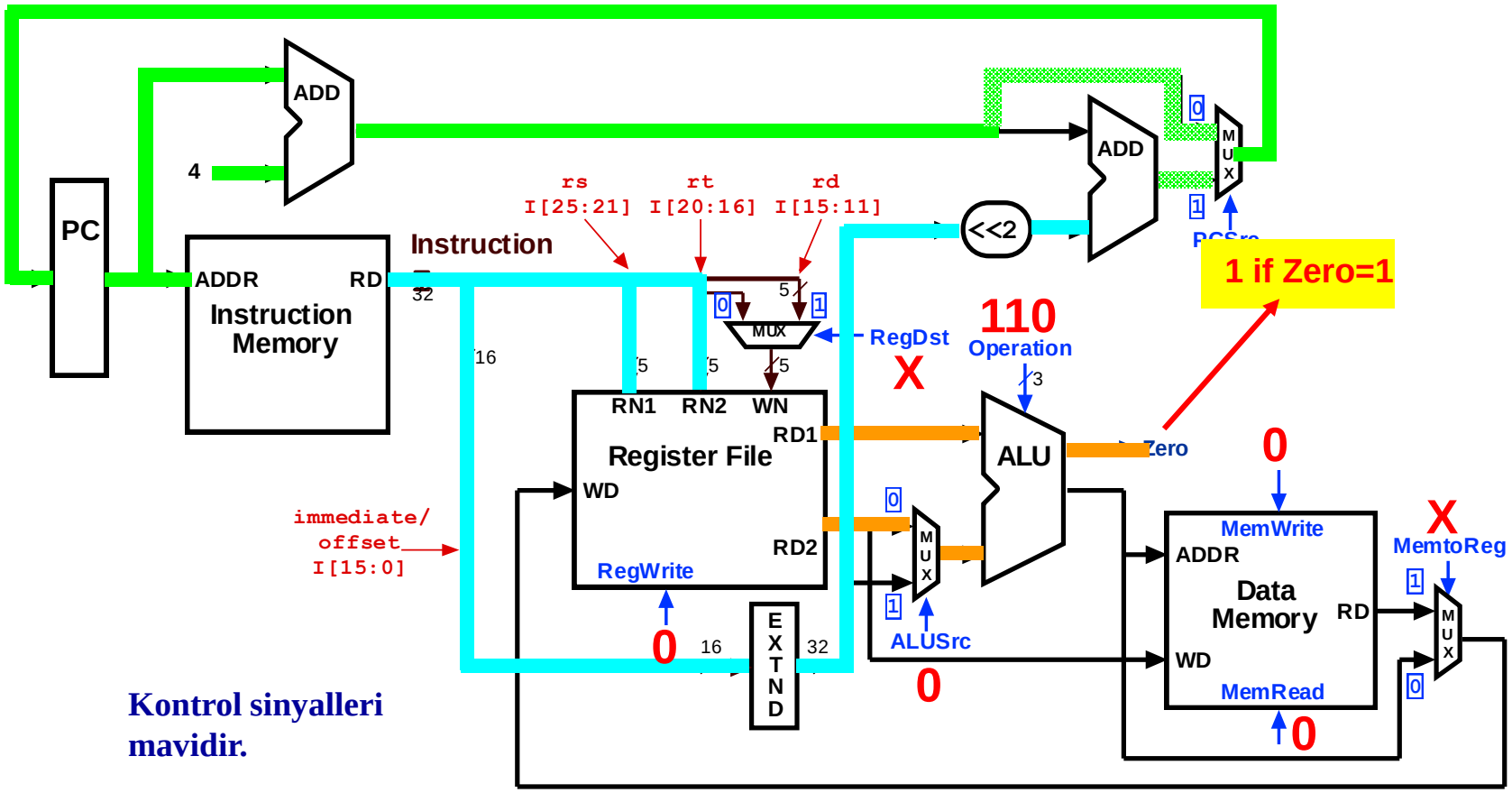
Kontrol sinyalleri mavi renktir.

Kontrol sinyalleri: `sw` komutu



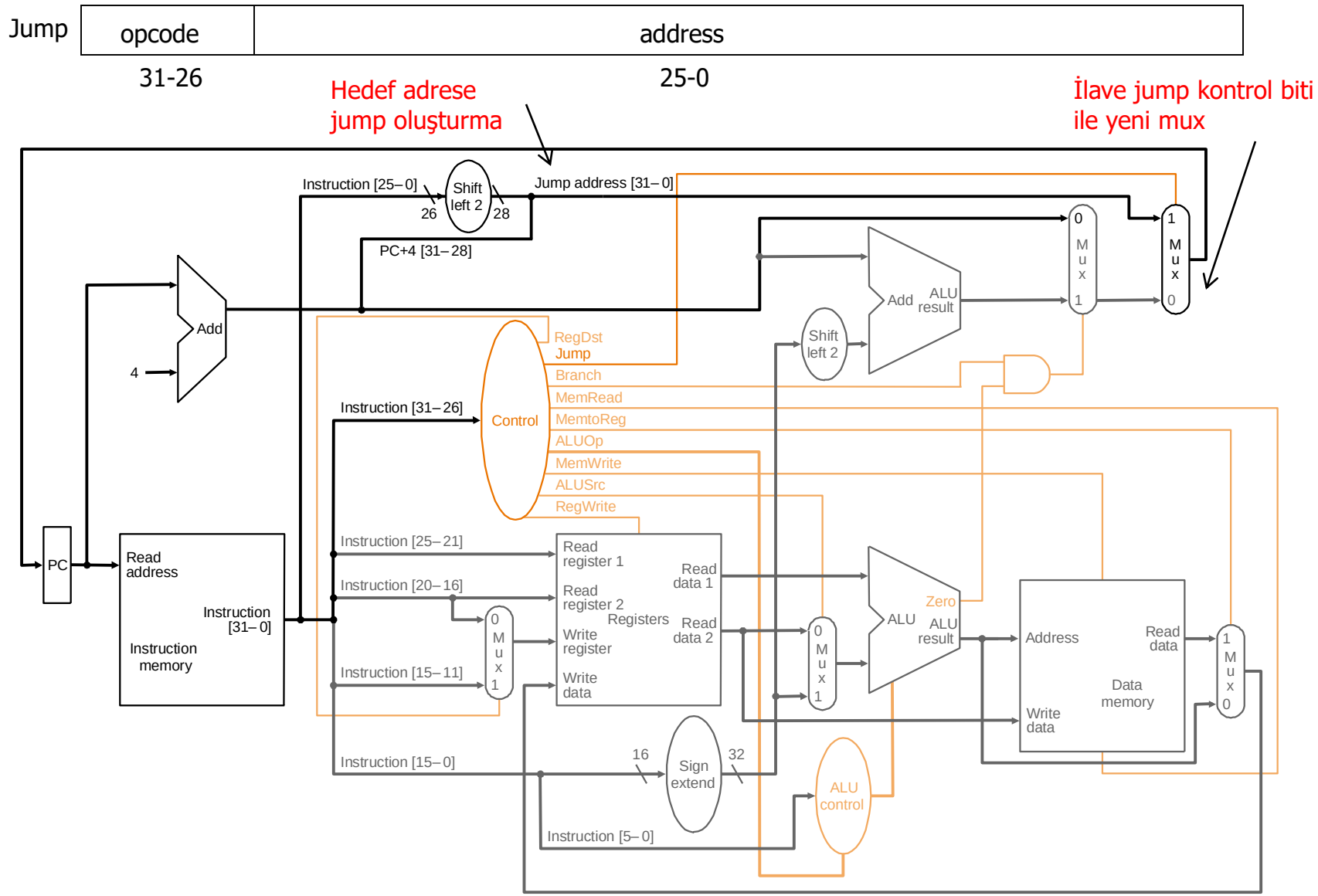
Kontrol sinyalleri Mavidir.

Kontrol Sinyalleri: `beq` komutu



Kontrol sinyalleri mavidir.

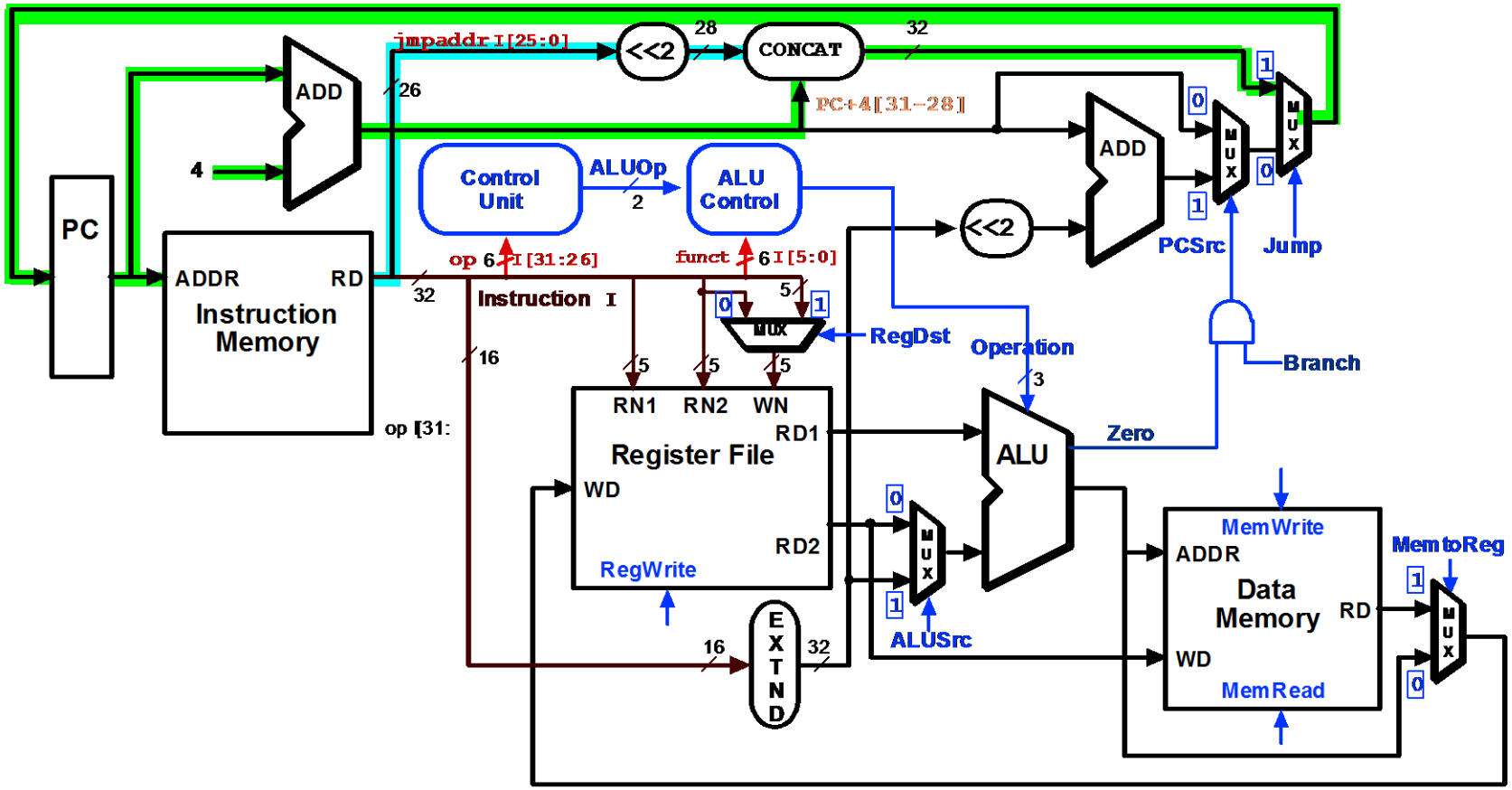
DATAPATH kontrol: III



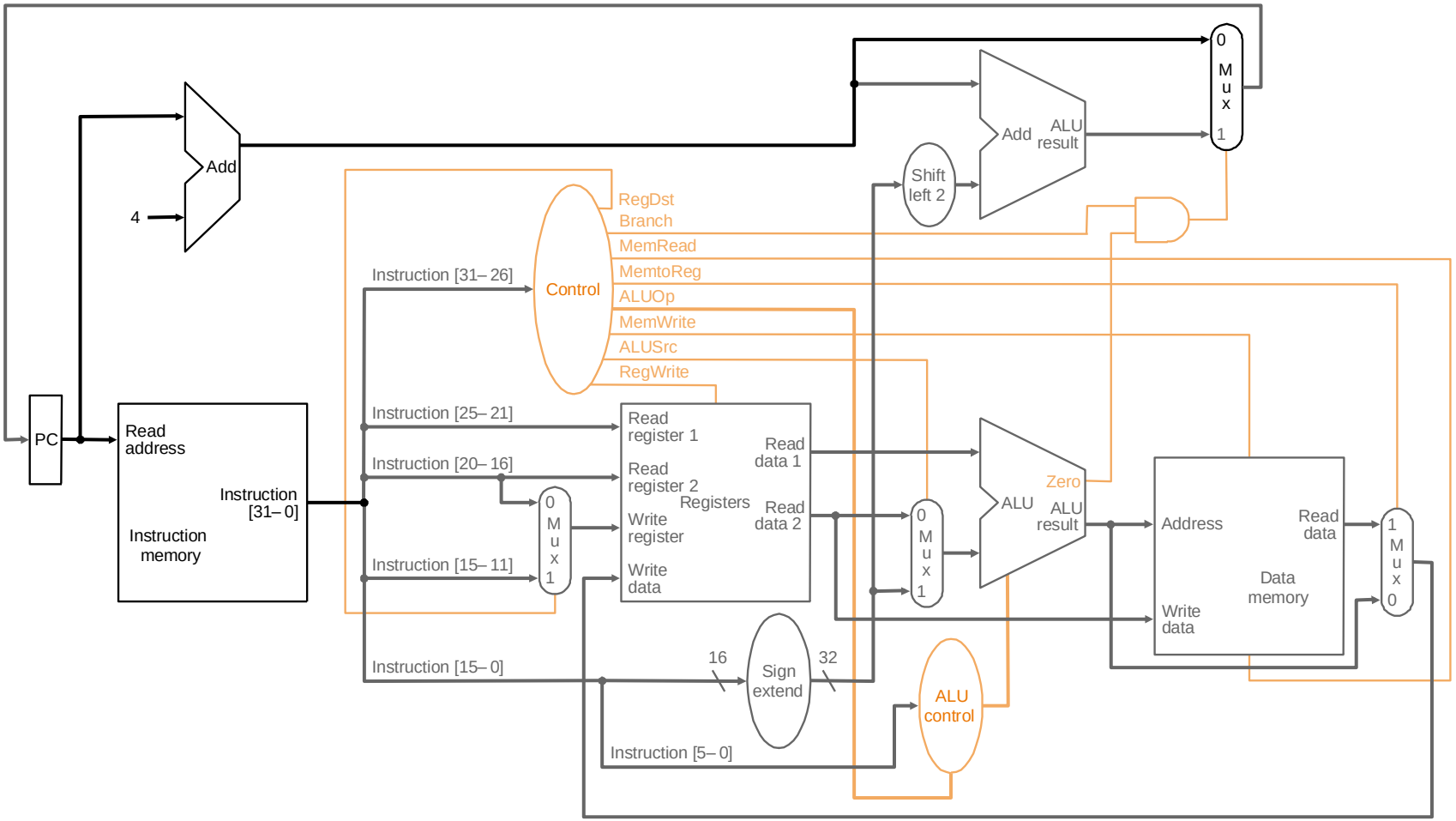
Jump'lar için MIPS veri yolu genişletilir: kontrol birimi yeni jump kontrol biti üretir.

DATAPATH: Executing `j`

`(j komutunun yürütülmesi)`

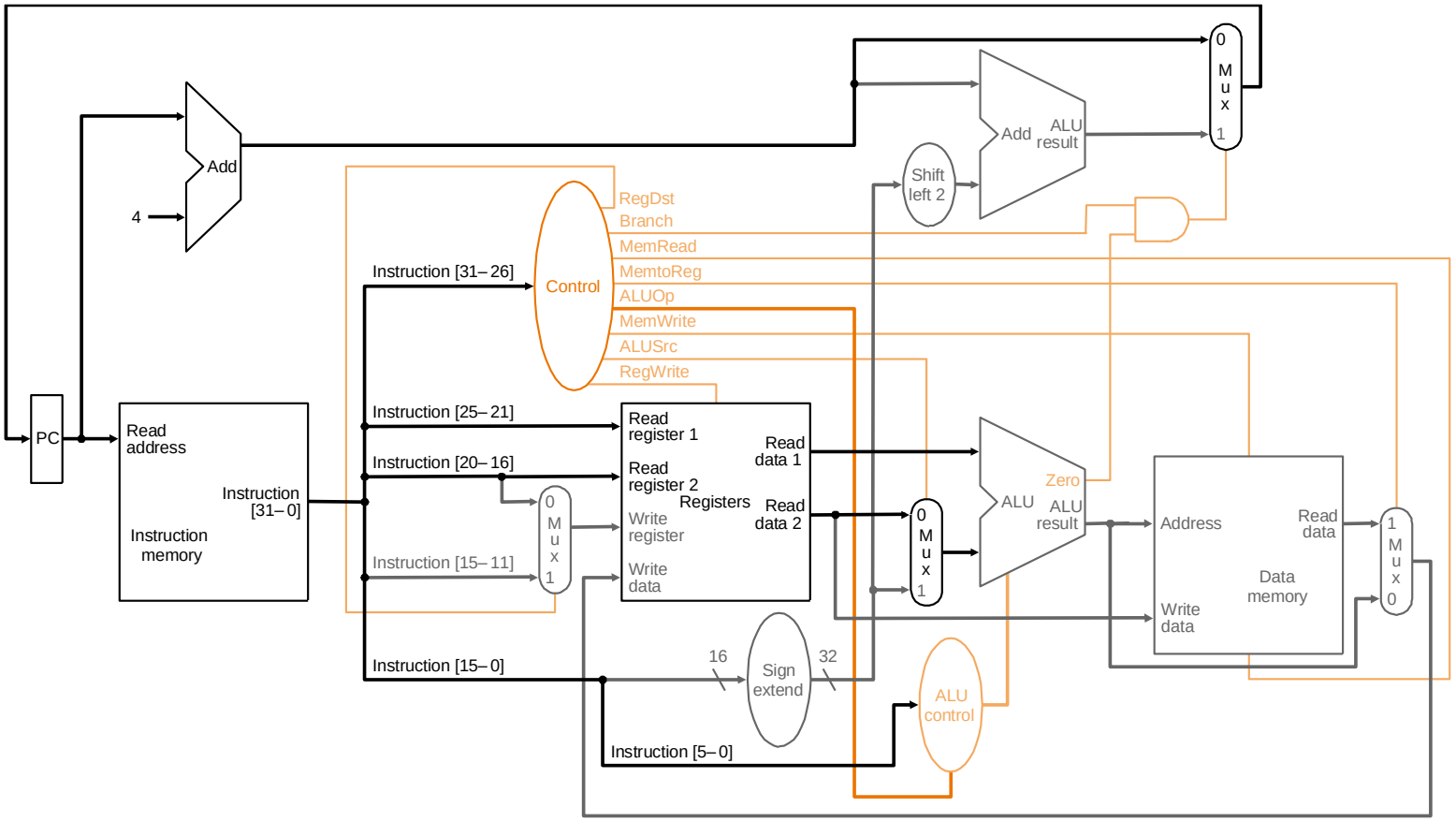


R-type Komutlar: Step 1
add $t1, $t2, $t3 (active = bold)



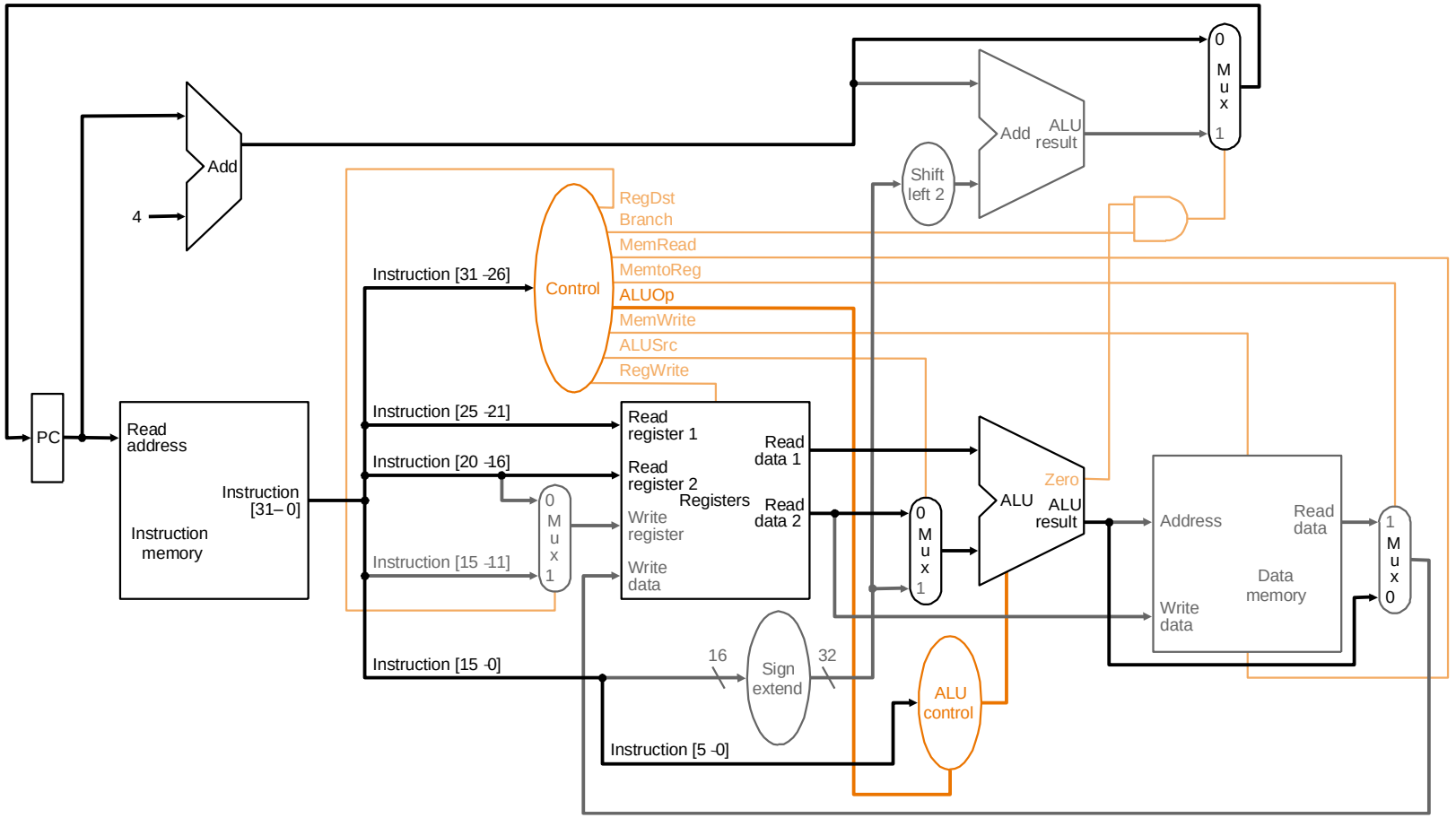
Fetch (komutun getirilmesi) PC'nin içeriğinin arttırılması

R-type Komut: Step 2
add $t1, $t2, $t3 (active = bold)



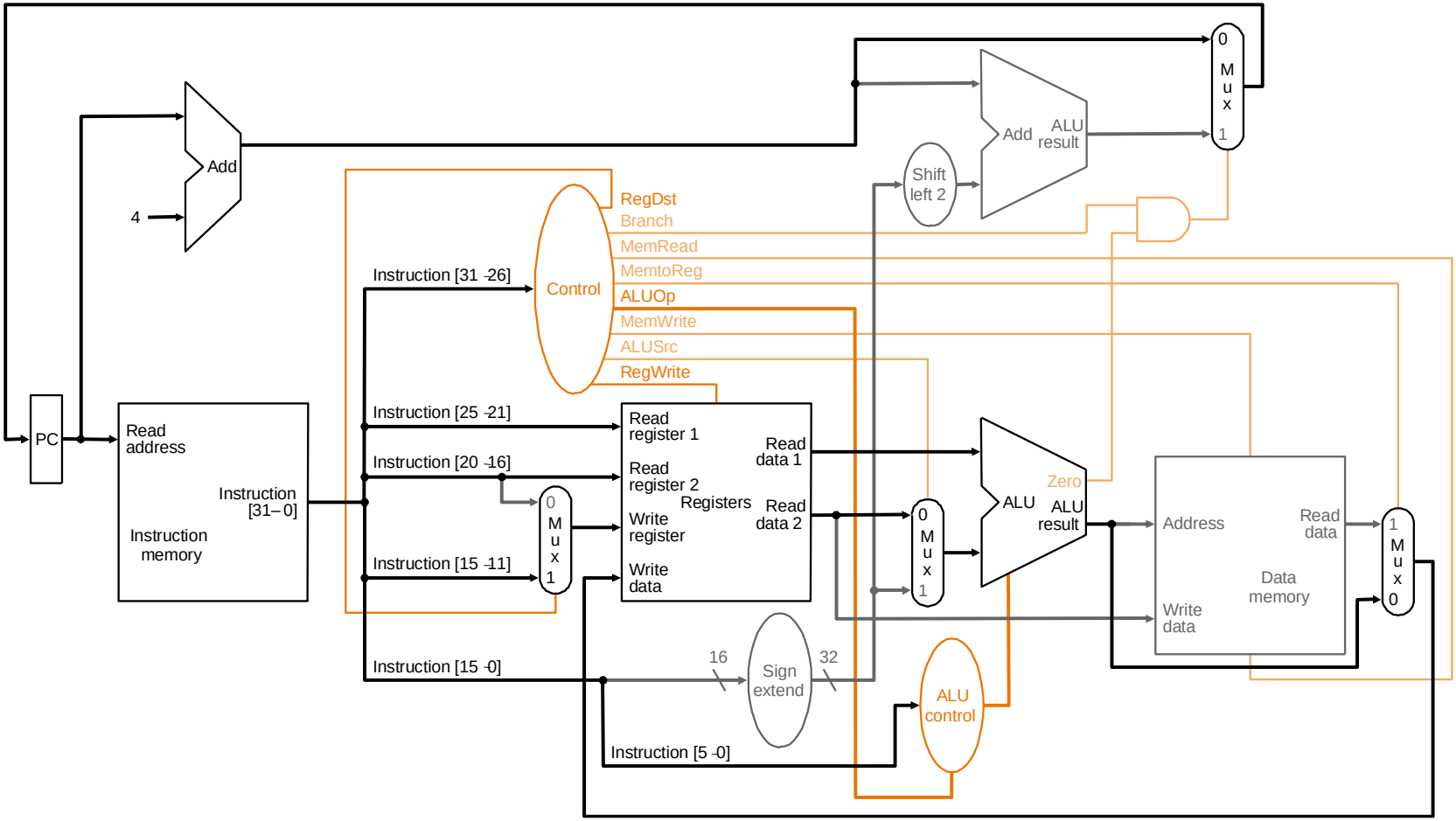
Register file'dan 2 kaynak register oku

R-type komut: Step 3
add $t1, $t2, $t3 (active = bold)



ALU 2 register operandı işler

R-type komut: Step 4
add $t1, $t2, $t3 (active = bold)



Sonuç registere yazılır.

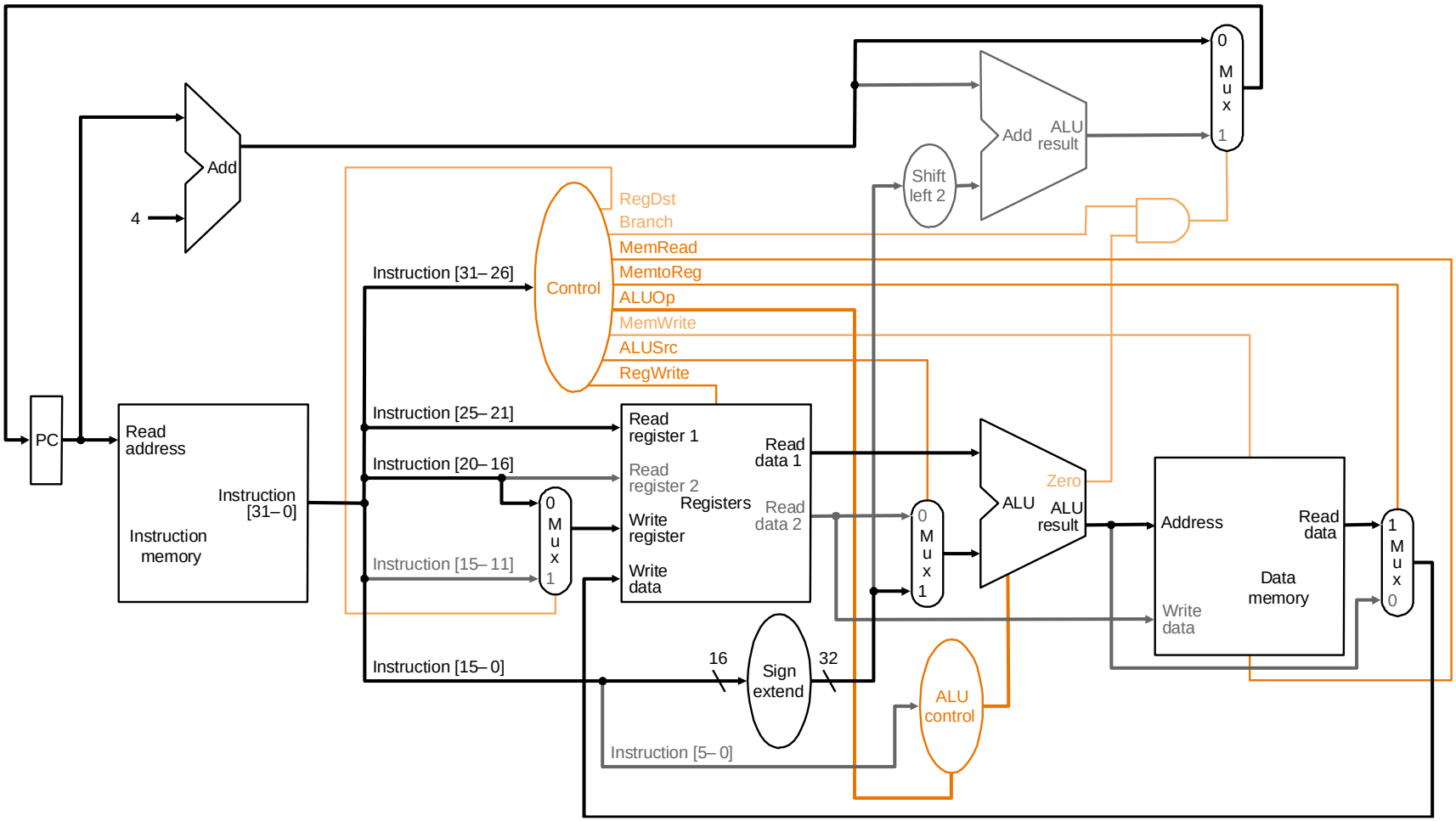
Single-cycle Gerçekleştirme Notları

- Adımlar gerçekten ayrı değildir. Tamamen bir *clock cycle*'da herbir komut tamamlanır.
- Bir cycle boyunca Datapath'in işletimi kombinasyoneldir. Hiçbir şey bir clock cycle boyunca depolanmaz.
- Bu yüzden, makine bir cycle başlangıcında stabildir. Ve cycle sonunda makine yeni bir stabil duruma geçer.
- *single-cycle işlemeyi anlamak için çok önemli*:
  bizim Verilogdaki SimpleSingleCycleComputer örneklerine bak.

Load komut adımları lw $t1,offset($t2)

1. Komutu Fetch et ve PC' yi arttır.
2. Register file'dan base registeri oku: base register ($t2) komutun 25-21 bitleri ile verilir.
3. ALU 16 bitlik lower sign-extended ve register file'dan değerlerin toplamını hesaplar
4. ALU'dan toplam data memory için adres olarak kullanılır.
5. Hafıza biriminden data register file'a yazılır. Hedef register ($t1) komutun 20-16 bitleriyle verilir

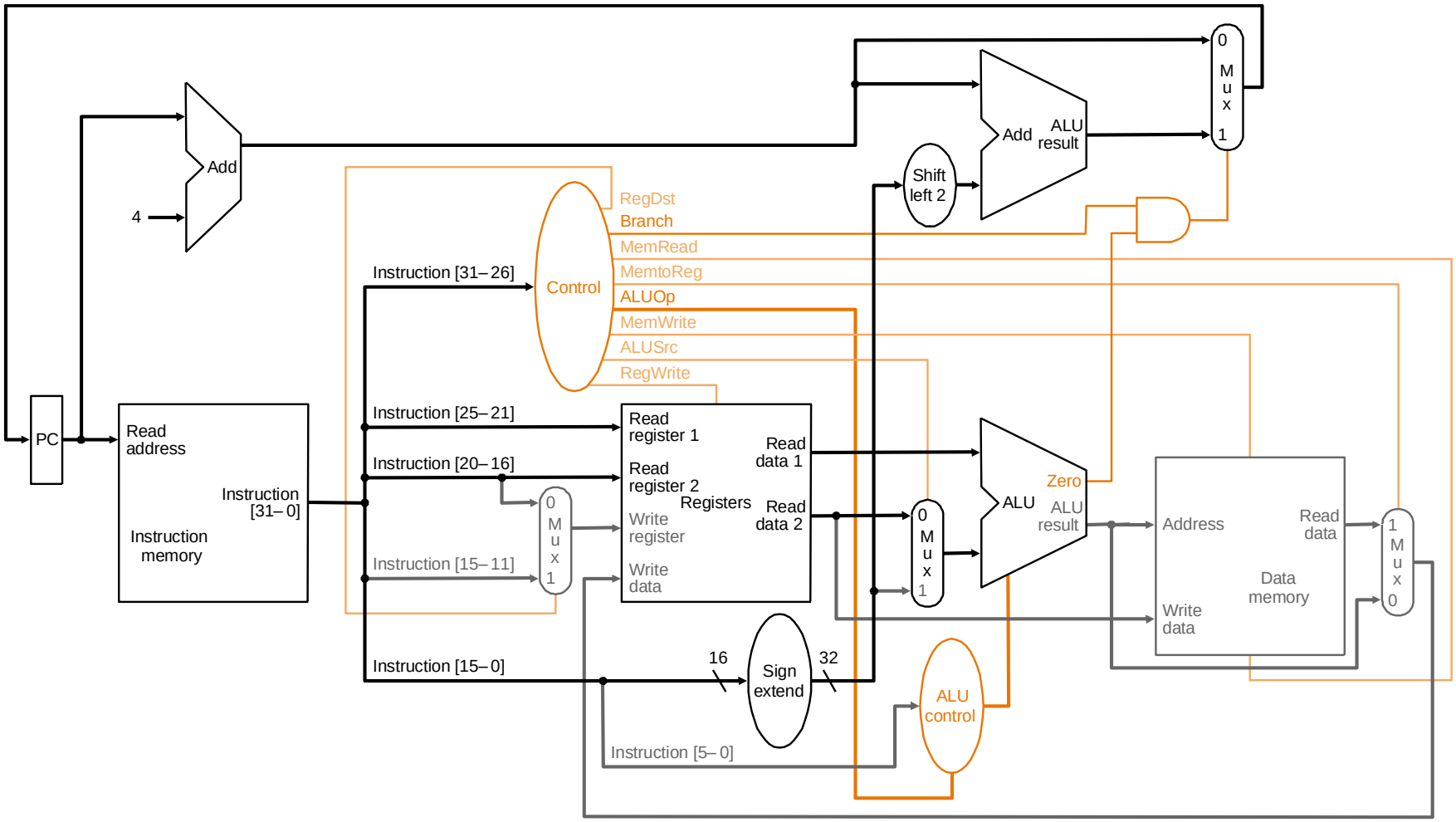
Load komutu : lw $t1 offset($t2)



Branch komut adımları: beq $t1, $t2, offset

1. Komutu Fetch et ve PC'yi arttır.
2. Register file'dan iki registeri ($t1 and $t2) oku
3. ALU register file'dan veri değerlerini çıkararak icra eder; PC+4' ün değeri Hedef dallanma adresini vermek için 2 sola kaydırılan komutun sign-extended lower 16 bit (offset)' eklenir.
4. ALU'dan Zero sonuçları karar vermek için kullanılır PC'de depolamak için sonuçlar toplanır. (1 veya 3. adımdan)

Branch komutu : beq $t1, $t2, offset

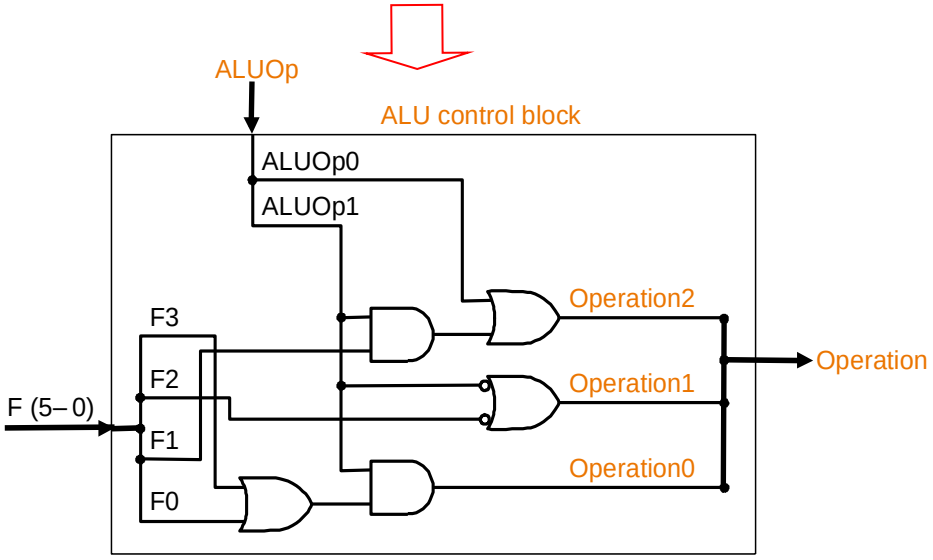


Gerçekleştirme: ALU kontrol Bloğu

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0 *	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Truth table for ALU control bits

•Eğer X olursa satır-2 ve satır3-7 arası karışıklık olur.

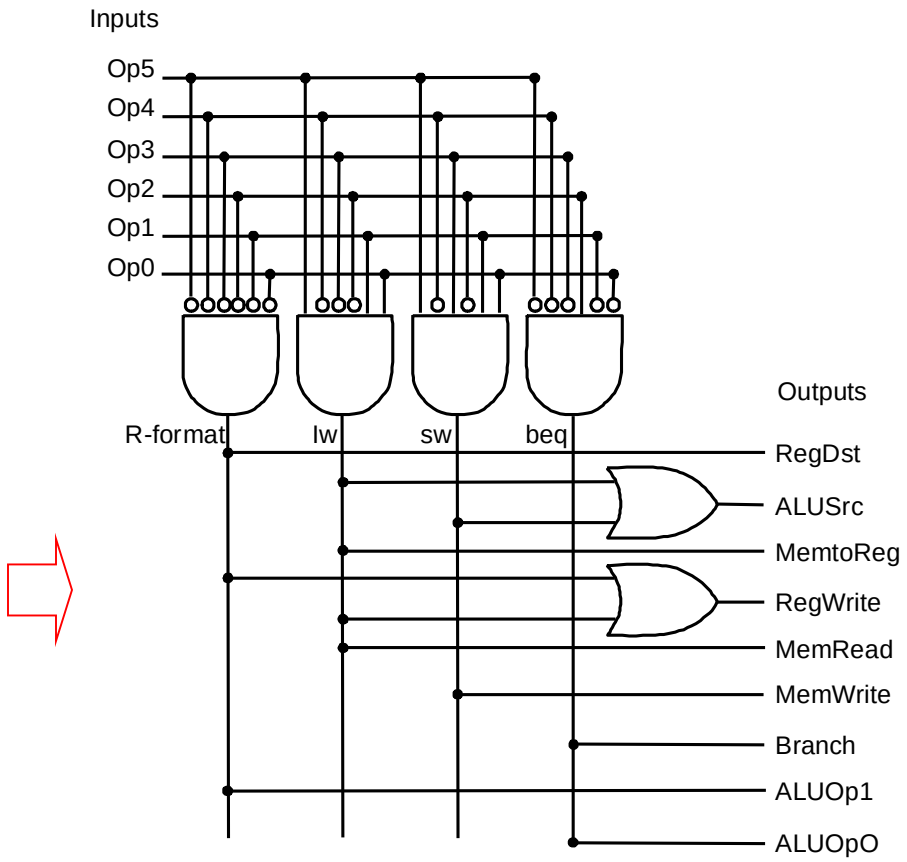


ALU Kontrol logic

Gerçekleştirme: Ana kontrol bloğu

	Signal name	R- Format	lw	sw	beq
Girişler	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Çıkışlar	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOP2	0	0	0	1

Ana kontrol sinyalleri için doğruluk tablosu



Ana kontrol PLA (programmable logic array): LA altında yatan prensip
Herhangi bir mantıksal gösterim Ürünlerin toplamı olarak yazılabilir.

Single-Cycle Tasarım Problemleri

- Farzedin ki *fixed-period clock* çalışmada; her komut datapath'de bir clock cycle kullanır.
  - CPI = 1
  - "cycle time" en uzun komutun uzunluğu tarafından belirlenir.
    - Ancak birkaç komut kısa clock cycle'da çalışabilir. Zaman israfı
    - Düşünün Daha komplike komutlara floating point gibi sahip olursak!
  - resources used more than once in the same cycle need to be duplicated
    - *Donanım ve chip alan israfi*

Örnek:Tek cycle gerçekleştirmede Fixed-period clock , variable-period clock kullanımı

- Kayan nokta ünitesi eklenmiş bir makine düşünün. Aşağıdaki gibi işlevsel(fonksiyonel) gecikmelerin olduğunu varsayalım:
  - *Memory erişimi*: 2 ns., *ALU* and *adders*: 2 ns., *FPU add*: 8 ns., *FPU multiply*: 16 ns., *register file access* (read or write): 1 ns.
  - *multiplexors*, *control unit*, *PC accesses*, *sign extension*, *wires*: gecikme yok.
- Farzedin ki komutlar aşağıdaki gibi karıştırılmıştır (Bir programda kullanılan komut sayısının % si olarak!!!!!!).
  - Yükleme komutu (Load) %31'lik benzer zaman almaktadır.
  - Tüm store'lar 21%'lik benzer zamanda oluşur.
  - R-format komutlar 27% 'lik benzer zamanda oluşmakta.
  - Dallanmalar 5% 'lik zamanda oluşmakta.
  - Şartsız dallanmalar 2%'lik zaman almakta.
  - FP toplama ve çıkarma benzer olarak toplam 7%'lik zamanda oluşmakta.
  - FP çarpma ve bölme benzer olarak toplam 7%'lik bir zaman almakta.

- ***(a) Bir sabit periyotlu Clock işareti ile single-cycle gerçekleştirme işlemi***
- ***(b) Her bir komutun yürütülmesi için yeteri kadar zaman uzunluğundaki değişken Clock periyoduna göre çalışan single-cycle işlemi (bu tip çalışma gerçekleştirilecek bir uygulama değildir.)***

  için performans karşılaştırması yapınız.

Çözüm

Instruction class	Instr. mem.	Register read	ALU oper.	Data mem.	Register write	FPU add/ sub	FPU mul/ div	Total time ns.
Load word	2	1	2	2	1			8
Store word	2	1	2	2				7
R-format	2	1	2	0	1			6
Branch	2	1	2					5
Jump	2							2
FP mul/div	2	1			1		16	20
FP add/sub	2	1			1	8		12

- Clock period for fixed-period clock = longest instruction time = 20 ns.
- Average clock period for variable-period clock =
  $8 \times 31\% + 7 \times 21\% + 6 \times 27\% + 5 \times 5\% + 2 \times 2\% + 20 \times 7\% + 12 \times 7\% = 8.1$ ns.
- Bu şekilde, $\text{performance}_{\text{var-period}} / \text{performance}_{\text{fixed-period}} = 20/7 = 2.46$