# BM-311 Bilgisayar Mimarisi

Hazırlayan: M.Ali Akcayol
Gazi Üniversitesi
Bilgisayar Mühendisliği Bölümü

## Konular

- Temel CPU Gerçekleştirimi
- Mantık Tasarım Gösterimleri
- Veriyolu Oluşturmak
- Basit Bir CPU Tasarımı
- Veriyolunda İşlem Yapmak

## Temel CPU Gerçekleştirimi

- Bir CPU'nun performansı clock cycle süresi ve her komut için gereken clock cycle sayısıyla belirlenir.
- Bu bölümde load, store, add, sub, and, or, shift left, branch if equal ve jump komutlarının donanımla oluşturulması anlatılacaktır.
- Her komut için PC değeri hafızaya gönderilir. Komutun operandlarına göre bir veya iki register'a hafızadan değer okunur.
- Komutların büyük bölümü benzer işlem birimlerini kullanır. Jump hariç tüm komutlar ALU'yu kullanır.
  - Hafıza erişimi yapanlar ALU ile adres hesaplar.
  - Aritmetik işlem yapanlar ALU'yu kullanır.
  - Branch komutları karşılaştırma için ALU'yu kullanır.

## Temel CPU Gerçekleştirimi - devam

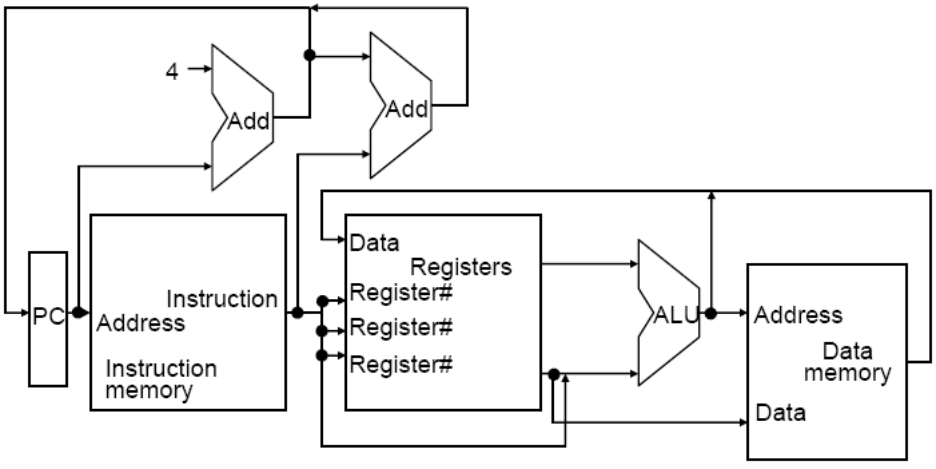
- ALU kullanıldıktan sonra komutlar farklı işlemler yaparlar.
- Hafıza erişimi yapanlar hafızaya bilgi yazarlar.
- Aritmetik-mantık işlem yapanlar sonucu bir register'a kaydederler.
- Branch yapanlar PC değerini değiştirirler.

## Temel CPU Gerçekleştirimi - devam

- Single Cycle
  - Her instruction 1 clock cycle'da tamamlanır
  - Clock cycle en yavaş instruction kadar uzun sürer
  - Dezavantajı, en yavaş instruction kadar hızlı olmasıdır
- Multi-Cycle
  - Fetch/execute cycle birden çok adıma bölünür
  - Her clock ile bir adım tamamlanır
  - Avantajı, her instruction ihtiyaç duyduğu kadar cycle kullanır
- Pipelined
  - Her instruction birden çok adımda çalıştırılır
  - Her cycle'da bir adım/instruction tamamlanır
  - Paralel olarak birden çok instruction işlenir

## Temel CPU Gerçekleştirimi - devam

- Şekilde temel CPU bileşenleri ve bağlantıları görülmektedir.

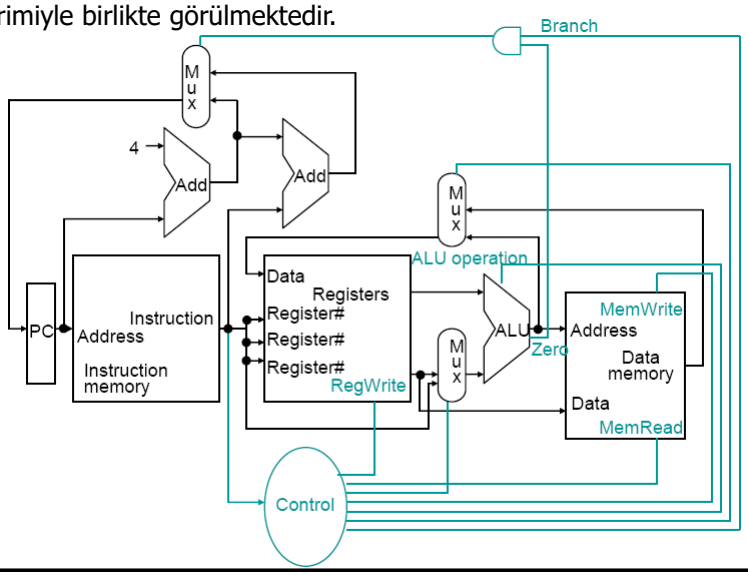


## Temel CPU Gerçekleştirimi - devam

- Şekilde PC değeri iki toplayıcıdan alınmaktadır.
- Register file içine yazılan veri ALU'dan veya memory'den alınmaktadır.
- Hangi birimin kullanılacağını multiplexer belirler.
- Hafızadan veri read komutuyla okunur, hafızaya write komutuyla yazılır.

## Temel CPU Gerçekleştirimi - devam

- Şekilde temel CPU bileşenleri ve bağlantıları, multiplexer ve kontrol birimiyle birlikte görülmektedir.



## Temel CPU Gerçekleştirimi - devam

- Şekilde PC değeri branch komutları için ALU çıkışıyla değiştirilmektedir.
- Multiplexer'ların tümü kontrol birimi tarafından denetlenmektedir.
- Kontrol birimi gereken kontrol işaretlerini instruction memory'den alınan ve decode edilen komutlarla belirler.

## Konular

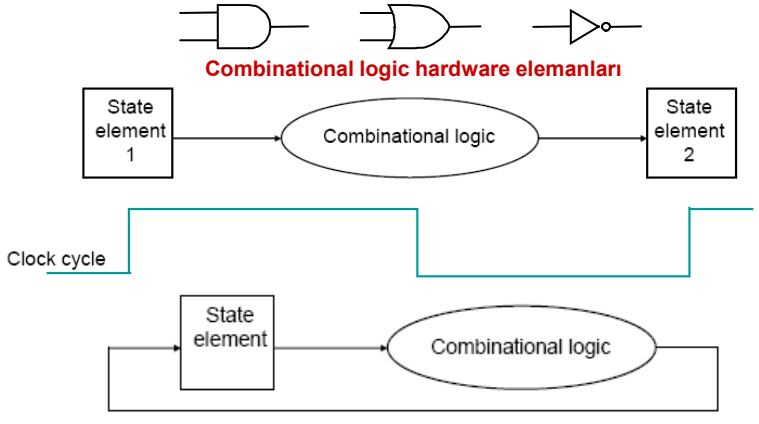
- Temel CPU Gerçekleştirimi
- Mantık Tasarım Gösterimleri
- Veriyolu Oluşturmak
- Basit Bir CPU Tasarımı
- Veriyolunda İşlem Yapmak

## Mantık Tasarım Gösterimleri

- CPU içindeki fonksiyonel birimler, veriler üzerinde işlem yapan ve durum saklayan iki tür mantık elemandan oluşur.
- İşlem yapan elemanlar *combinational* elemanlardır.
- Durum elemanları veri saklama birimine ihtiyaç duyar.
- D flip flop durum saklayan elemanı olarak kullanılabilir. Clock cycle ve girişe sahiptir. Her clock için durumunu yeniden oluşturur.
- Clock sinyal verinin ne zaman okunacağını veya yazılacağını belirler.

## Mantık Tasarım Gösterimleri

- Şekilde clock cycle ile durum elemanı 1'den durum elamanı 2'ye combinational logic ile sinyal aktarımı görülmektedir.



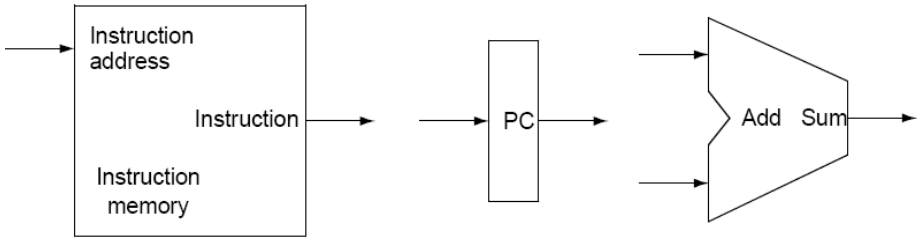

- Durumlararası geçiş bir clock cycle ile yapılmaktadır.
- Clock cycle'larda kenar-tetiklemeli (edge-triggered) yöntemi kullanılır.
- Yükselen veya düşen kenarda tüm işlemler başlatılır.

## Konular

- Temel CPU Gerçekleştirimi
- Mantık Tasarım Gösterimleri
- Veriyolu Oluşturmak
- Basit Bir CPU Tasarımı
- Veriyolunda İşlem Yapmak

## Veriyolu Oluşturma

- Şekilde komut hafızası, PC ve toplayıcı görülmektedir.

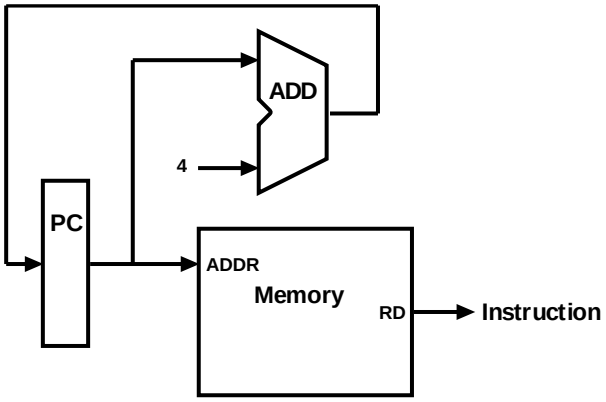



a. Instruction memory b. Program counter c. Adder

- Hafıza birimi komut sağlar.
- PC komut adresi saklar.
- Toplayıcı sonraki komutun adresini belirler.

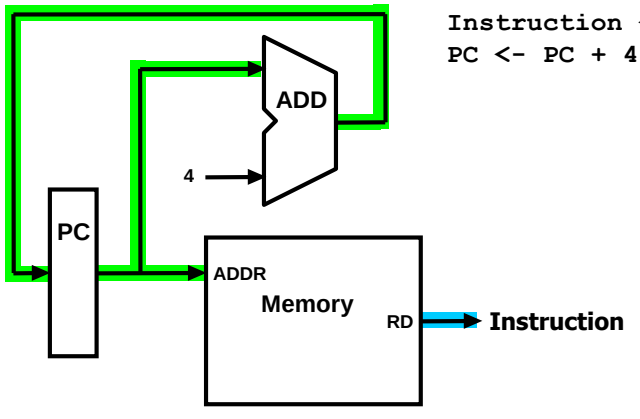
## Veriyolu Oluşturma

- Aşağıda bu üç birimin bağlanmış şekli görülmektedir.




- Şekilde instruction fetch ve PC değerinin artırılışı görülmektedir.

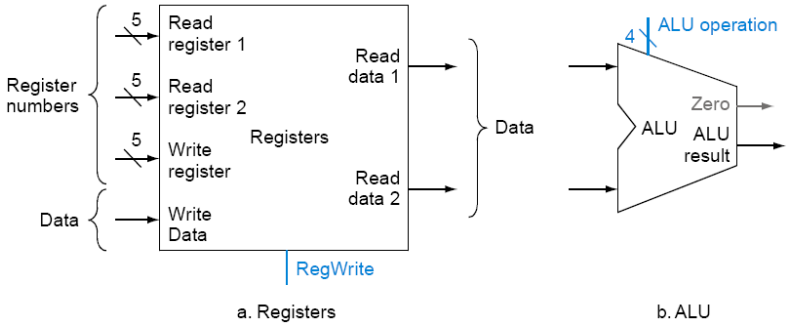
## Veriyolu Oluşturma




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

## Veriyolu Oluşturma

- R-format komutlar 3 register kullanır. İki giriş ve bir çıkış için kullanılır.
- Okuma için iki girişe ve yazma için bir girişe sahip register file kullanılabilir.
- Şekilde register file (32-bit genel amaçlı register'lar) ve ALU görülmektedir.

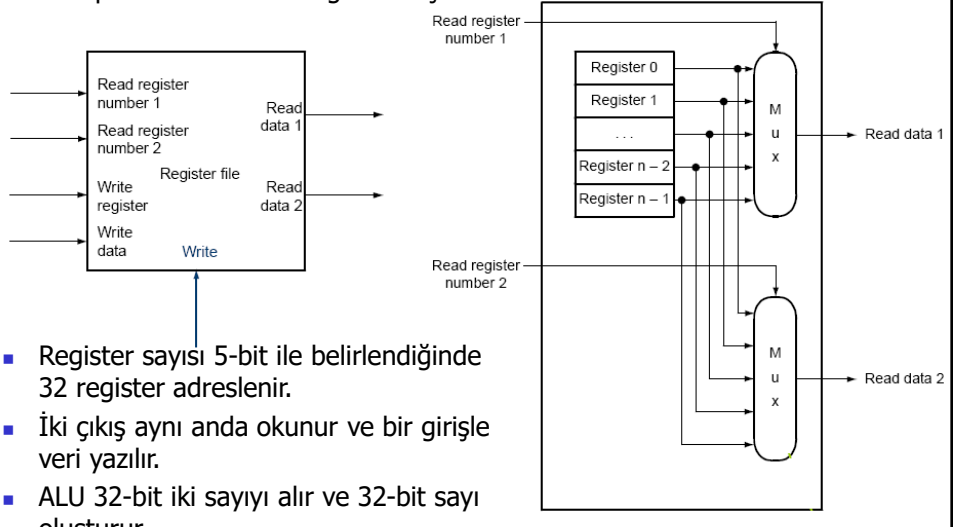



a. Registers b. ALU

- Register sayısı 5-bit ile belirlendiğinde 32 register adreslenir.
- İki çıkış aynı anda okunur ve bir girişle veri yazılır.
- ALU 32-bit iki sayıyı alır ve 32-bit sayı oluşturur.
- Sonuç 0 ise zero çıkışı 1 olur.

## Veriyolu Oluşturma

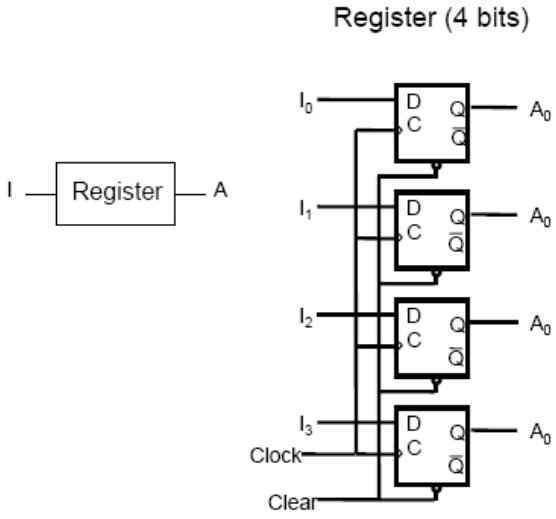
- Multiplexer kullanarak register seçimi




- Register sayısı 5-bit ile belirlendiğinde 32 register adreslenir.
- İki çıkış aynı anda okunur ve bir girişle veri yazılır.
- ALU 32-bit iki sayıyı alır ve 32-bit sayı oluşturur.
- Sonuç 0 ise zero çıkışı 1 olur.

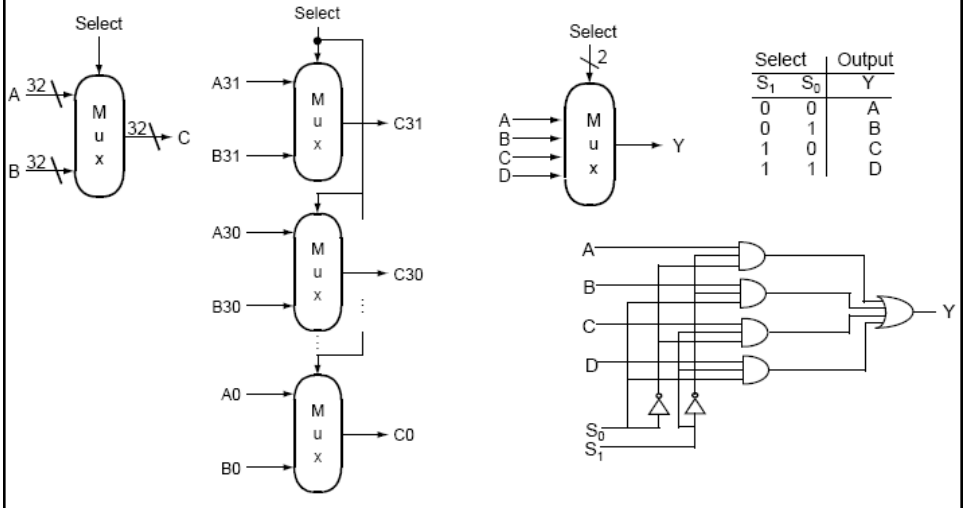
## Veriyolu Oluşturma

### Register



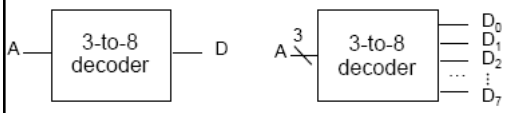

## Veriyolu Oluşturma

### Multiplexer



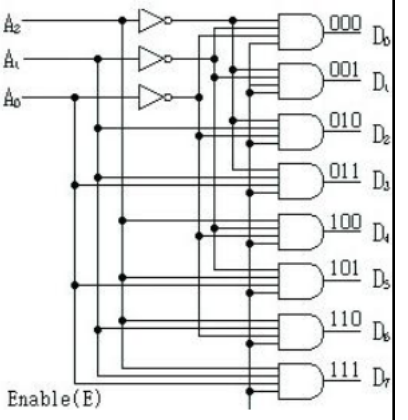
## Veriyolu Oluşturma

### n-to-m decoder



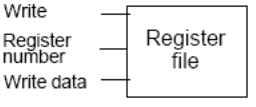

Truth table

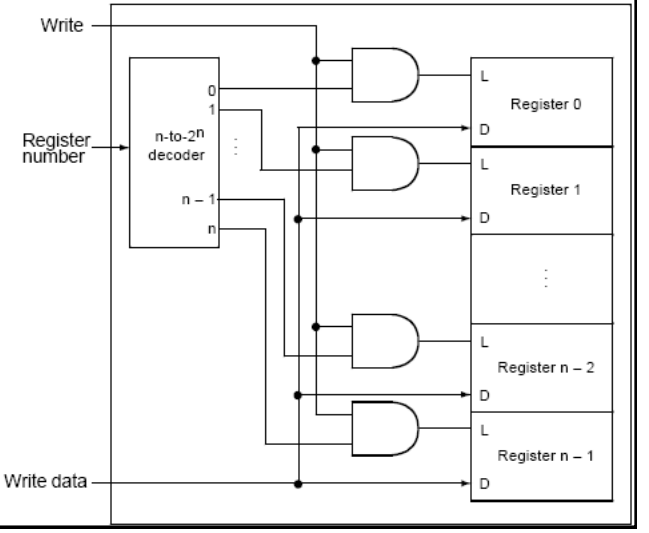
| Enable | Input | Outputs |
|---|---|---|
| E | $A_2$ $A_1$ $A_0$ | $D_7$ $D_6$ $D_5$ $D_4$ $D_3$ $D_2$ $D_1$ $D_0$ |
| 0 | x x x | 0 0 0 0 0 0 0 0 |
| 1 | 0 0 0 | 0 0 0 0 0 0 0 1 |
| 1 | 0 0 1 | 0 0 0 0 0 0 1 0 |
| 1 | 0 1 0 | 0 0 0 0 0 1 0 0 |
| 1 | 0 1 1 | 0 0 0 0 1 0 0 0 |
| 1 | 1 0 0 | 0 0 0 1 0 0 0 0 |
| 1 | 1 0 1 | 0 0 1 0 0 0 0 0 |
| 1 | 1 1 0 | 0 1 0 0 0 0 0 0 |
| 1 | 1 1 1 | 1 0 0 0 0 0 0 0 |




## Veriyolu Oluşturma

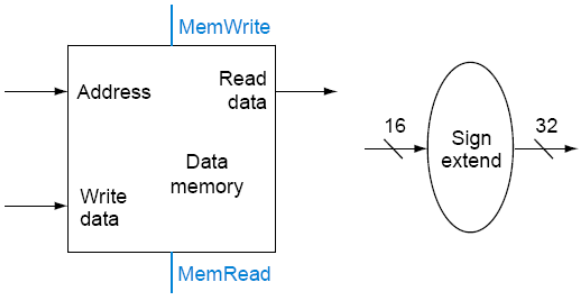
### Register file



## Veriyolu Oluşturma

- 16-bit işaretli sayının 32-bit işaretli sayıya dönüştürülmesi için sign-extend birimine ihtiyaç vardır.
- Şekilde load ve store komutları için gereken birimler görülmektedir.

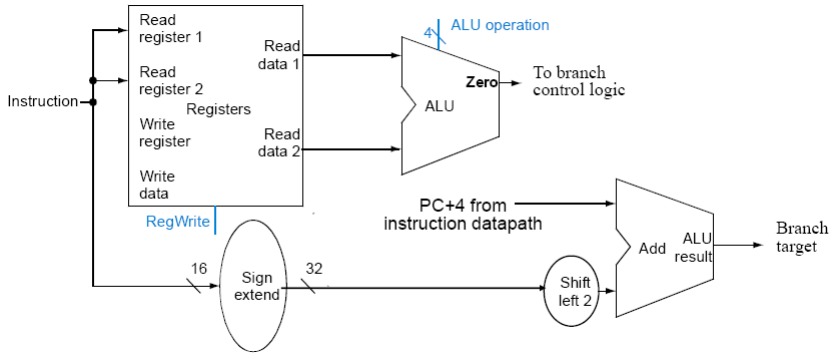



a. Data memory unit b. Sign-extension unit

- Hafıza birimi read / write komutlarını kontrol biriminden alır.
- Address ve write data girişleri adres bus ile kontrol bus'tan alınır.
- Sign extend birimi işaretli sayılarda bit sayısını artırır.

## Veriyolu Oluşturma

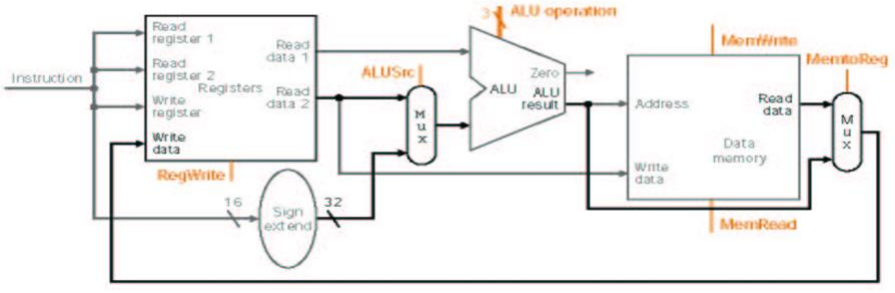
- Branch komutları iki girişi karşılaştırır ve sonuca göre PC değerini değiştirir.
- Şekilde branch equal komutunun çalışması için gerekli yapı görülmektedir.




- Üstteki ALU'da karşılaştırma sonucu eşit olursa, zero çıkışı kontrol birimini aktif yapar ve atlama gerçekleştirilir.
- Shift left 2 birimi offset değeri sola 2 kaydırır (4 ile çarpar).

## Veriyolu Oluşturma

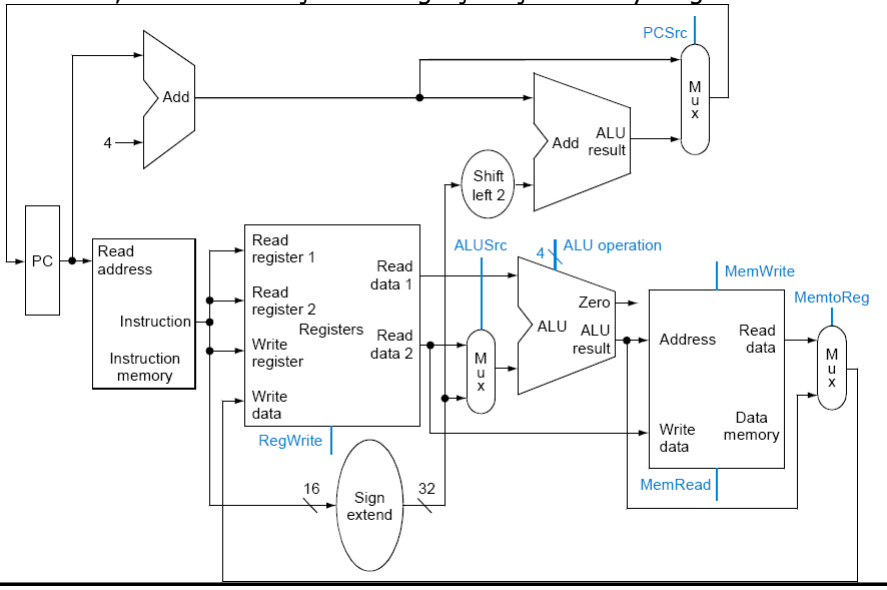
- Tek clock cycle ile komutları çalıştıran veriyolu aşağıda görülmektedir.




- Şekilde 2 multiplexer kullanılmıştır.
- MemToReg verinin hafızadan register'a aktarılması için 1 olmalıdır.
- ALU operation girişi ALU tarafından gerçekleştirilecek işlemi belirler.
- ALUSrc ALU'nun ikinci girişinin nerden alınacağını belirler.

## Veriyolu Oluşturma

- Branch, load ve store işlemlerini gerçekleştiren veriyolu görülmektedir.



## Konular

- Temel CPU Gerçekleştirimi
- Mantık Tasarım Gösterimleri
- Veriyolu Oluşturmak
- Basit Bir CPU Tasarımı
- Veriyolunda İşlem Yapmak

## Basit Bir CPU Tasarımı

- Bu kısımda lw (load word), sw (store word), beq (branch equal), add, sub, and, or ve sl (set on less than) komutlarının gerçekleştirilmesi yapılacaktır.
- ALU girişleri 4 tane alınmıştır. Toplam 16 komut kullanılabilir. Burda 6 tanesi kullanılacaktır.

| ALU Kontrol | Fonksiyon |
|---|---|
| 0000 | and |
| 0001 | or |
| 0010 | add |
| 0110 | sub |
| 0111 | sl |
| 1100 | nor |

- ld ve st için, ALU hafıza adresini hesaplar.
- Aritmetik işlemler ile branch işlemlerindede ALU kullanılır.

## Basit Bir CPU Tasarımı

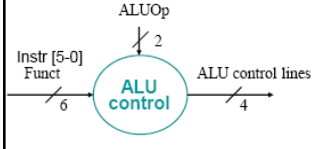
- ALU kontrol girişleri 2-bit ve fonksiyon kodu 6-bit'tir.

| Instruction | Instruction opcode | ALUOp | funct field | Desired ALU action | ALU control input |
|---|---|---|---|---|---|
| lw | lw | 00 | XXXXXX | add | 0010 |
| sw | sw | 00 | XXXXXX | add | 0010 |
| beq | beq | 01 | XXXXXX | subtract | 0110 |
| add | R-type | 10 | 100000 | add | 0010 |
| sub | R-type | 10 | 100010 | subtract | 0110 |
| and | R-type | 10 | 100100 | and | 0000 |
| or | R-type | 10 | 100101 | or | 0001 |
| slt | R-type | 10 | 101010 | set on less than | 0111 |

- Tabloda, ALUOp kodunun 00 ve 01 olduğu durumlarda, ALU'nun yapacağı iş komutun fonksiyon koduna bağlı değildir. O yüzden XXXXXX olarak verilmiştir.

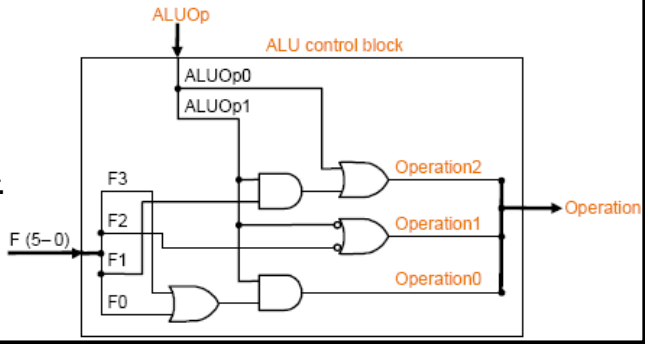
## Basit Bir CPU Tasarımı

- Tabloda ALU kontrol girişleri için doğruluk tablosu görülmektedir.



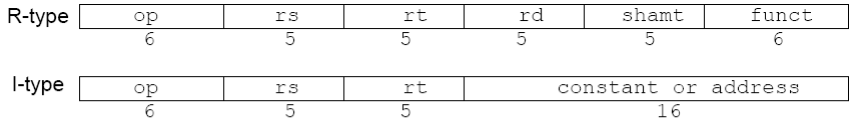

| ALUOp | | Funct field | | | | | | Operation |
|---|---|---|---|---|---|---|---|---|
| ALUOp1 | ALUOp0 | F5 | F4 | F3 | F2 | F1 | F0 | |
| 0 | 0 | X | X | X | X | X | X | 0010 |
| X | 1 | X | X | X | X | X | X | 0110 |
| 1 | X | X | X | 0 | 0 | 0 | 0 | 0010 |
| 1 | X | X | X | 0 | 0 | 1 | 0 | 0110 |
| 1 | X | X | X | 0 | 1 | 0 | 0 | 0000 |
| 1 | X | X | X | 0 | 1 | 0 | 1 | 0001 |
| 1 | X | X | X | 1 | 0 | 1 | 0 | 0111 |

- Tabloda, ALUOp kodunun 00 ve 01 olduğu durumlarda, ALU'nun yapacağı iş komutun fonksiyon koduna bağlı değildir. O yüzden XXXXXX olarak verilmiştir.

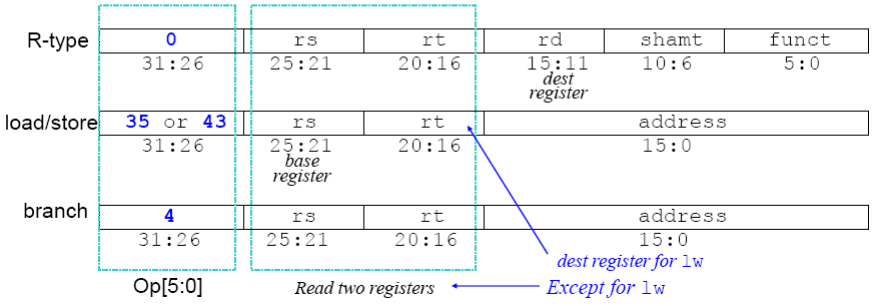


## Basit Bir CPU Tasarımı

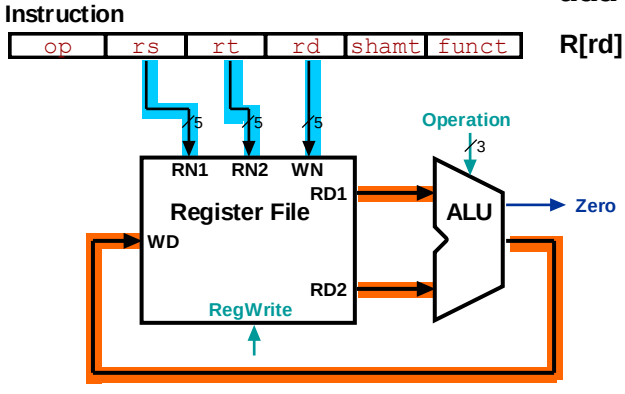
- Bir ALU tasarımında önce komut formatlarının ve alanlarının tanımlanması gerekir.




- The three instruction classes




## Basit Bir CPU Tasarımı

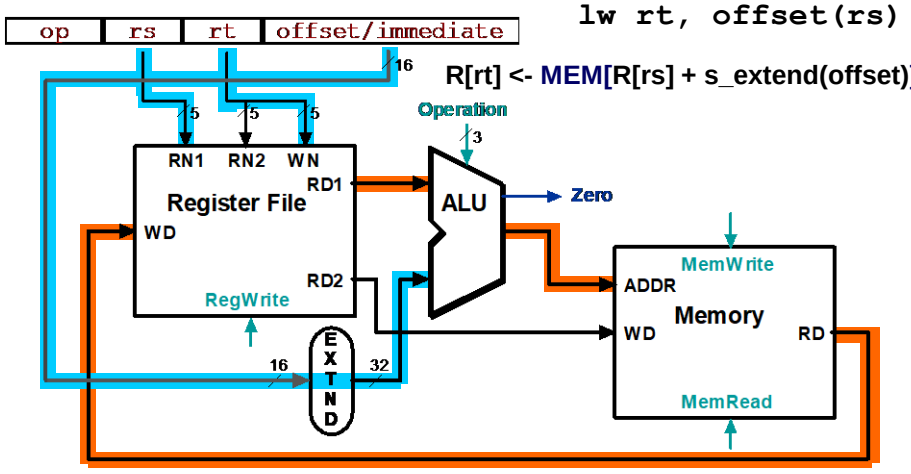



**`add rd, rs, rt`**

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

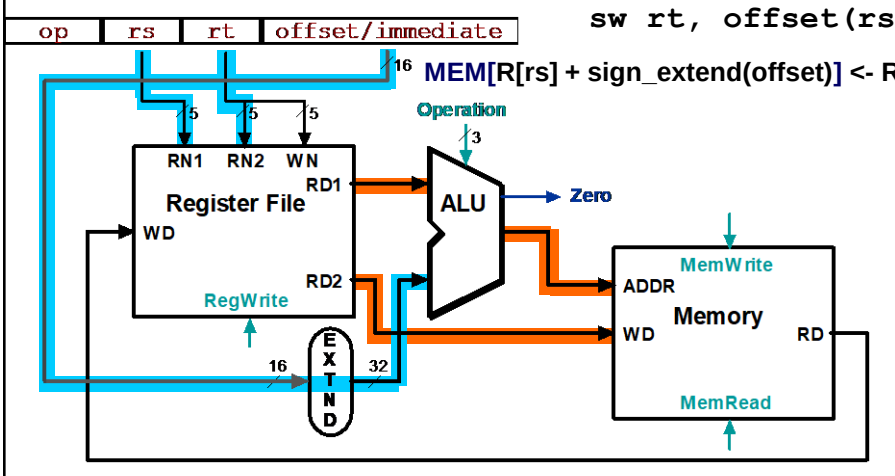
## Basit Bir CPU Tasarımı

`lw rt, offset(rs)`

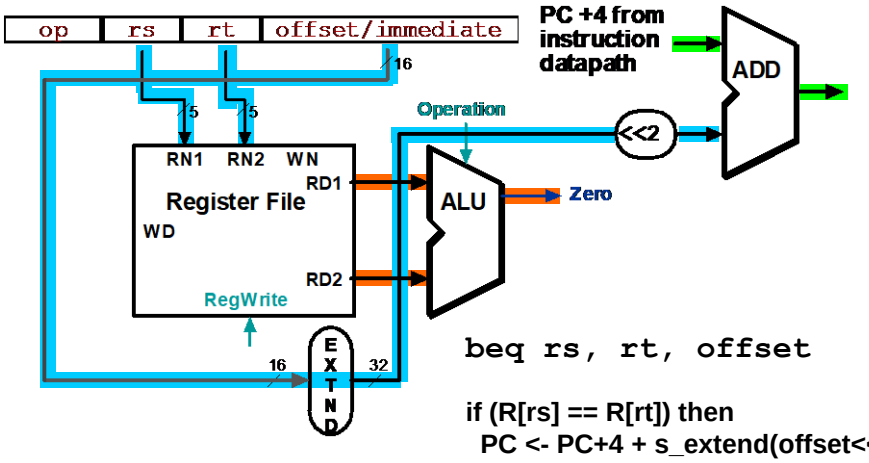



## Basit Bir CPU Tasarımı

`sw rt, offset(rs)`



## Basit Bir CPU Tasarımı




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

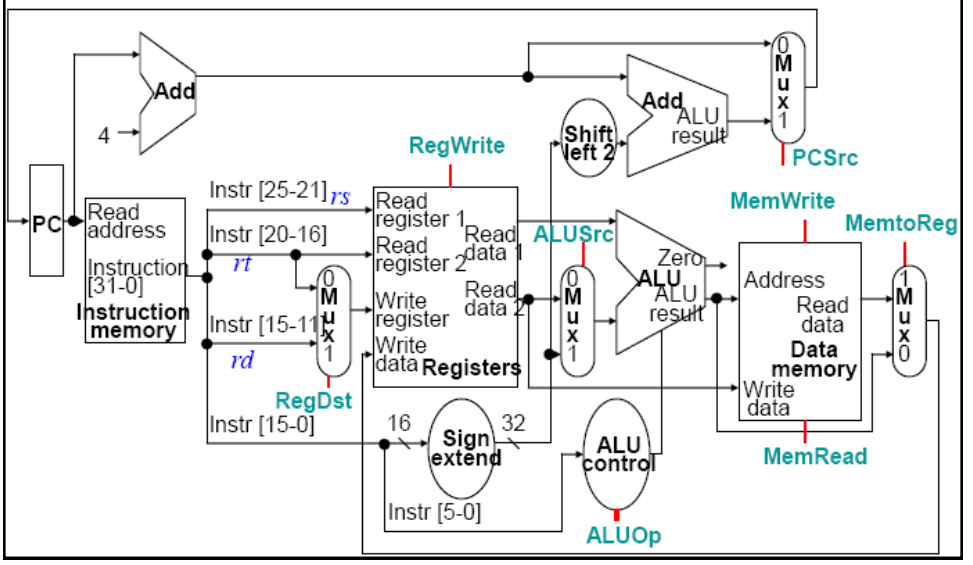
**if (R[rs] == R[rt]) then**
**PC <- PC+4 + s_extend(offset<<2)**

## Basit Bir CPU Tasarımı

- Opcode alanı tüm komutlarda 31:26 bitlerle gösterilir.
- R-type komutlar, load, store ve branch komutlarında giriş olan iki register rs ve rt olarak gösterilir.
- Load ve store için default register rs (25:21) olarak verilir.
- 16-bit offset adres (load, store ve branch) 15:0 arasındadır.
- Hedef register load için rt (20:16) ve r-type komutlar için rd (15:11) olur. Burda bir multiplexer ile seçme yapılacaktır.

## Basit Bir CPU Tasarımı

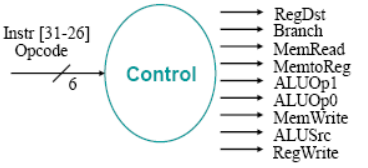
- Multiplexer ve kontrol işaretleri



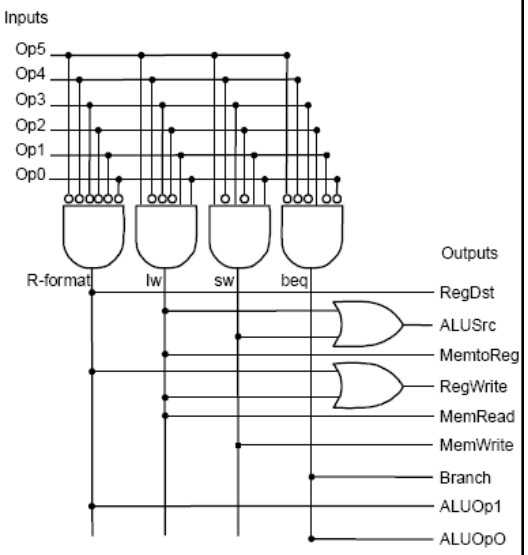

## Basit Bir CPU Tasarımı

- Aşağıda veriyolu tasarımında verilen kontrol işaretleri listelenmiştir.
  - RegDst: rt / rd alanı belirler (20:16) hedef register'dır.
  - RegWrite: Sounucun yazılacağı register'ı belirler.
  - ALUSrc: ALU'nun bir girişinin nerden alınacağını belirler.
  - MemRead: Hafızadan okuma yapar.
  - MemWrite: Hafızaya yazar.
  - MemtoReg: Hafızadan register'a veri aktarır.

# Basit Bir CPU Tasarımı



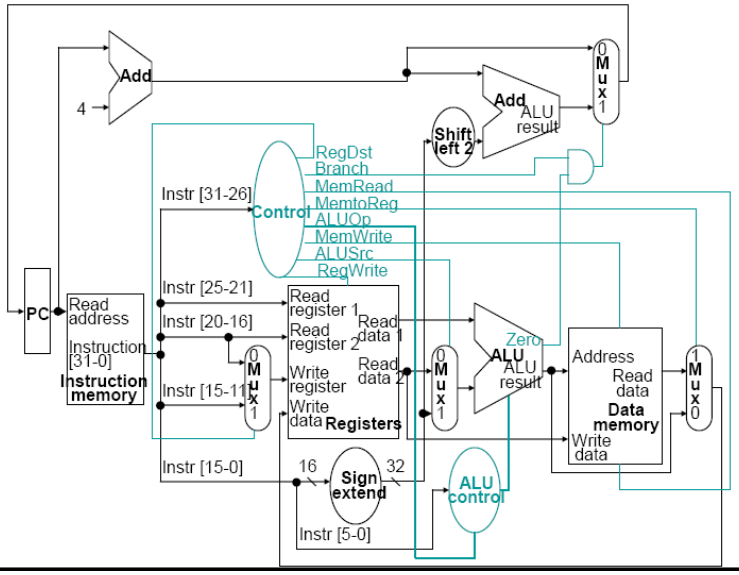

| | Sinyal adı | R-format | lw | sw | beq |
|---|---|---|---|---|---|
| Inputs | Op5 | 0 | 1 | 1 | 0 |
| | Op4 | 0 | 0 | 0 | 0 |
| | Op3 | 0 | 0 | 1 | 0 |
| | Op2 | 0 | 0 | 0 | 1 |
| | Op1 | 0 | 1 | 1 | 0 |
| | Op0 | 0 | 1 | 1 | 0 |
| Outputs | RegDst | 1 | 0 | x | x |
| | ALUSrc | 0 | 1 | 1 | 0 |
| | 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 |




Kontrol birimi

# Basit Bir CPU Tasarımı

- Aşağıdaki şekilde veriyolu ve kontrol birimi görülmektedir.



## Basit Bir CPU Tasarımı

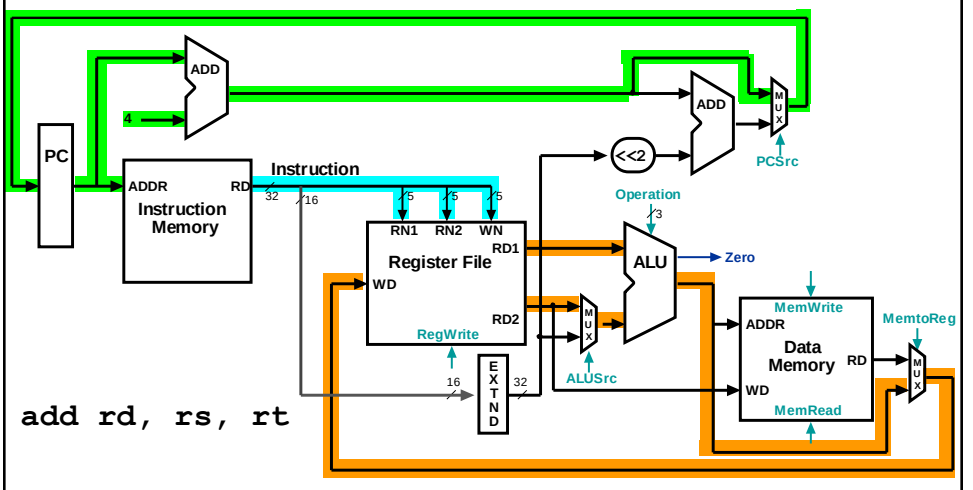
- Her opcode için üretilen kontrol işaretleri aşağıda verilmiştir.

| Instruction | RegDst | ALUSrc | MemtoReg | RegWrite | MemRead | MemWrite | Branch | ALUOp1 | ALUOp0 |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

## Konular

- Temel CPU Gerçekleştirimi
- Mantık Tasarım Gösterimleri
- Veriyolu Oluşturmak
- Basit Bir CPU Tasarımı
- Veriyolunda İşlem Yapmak

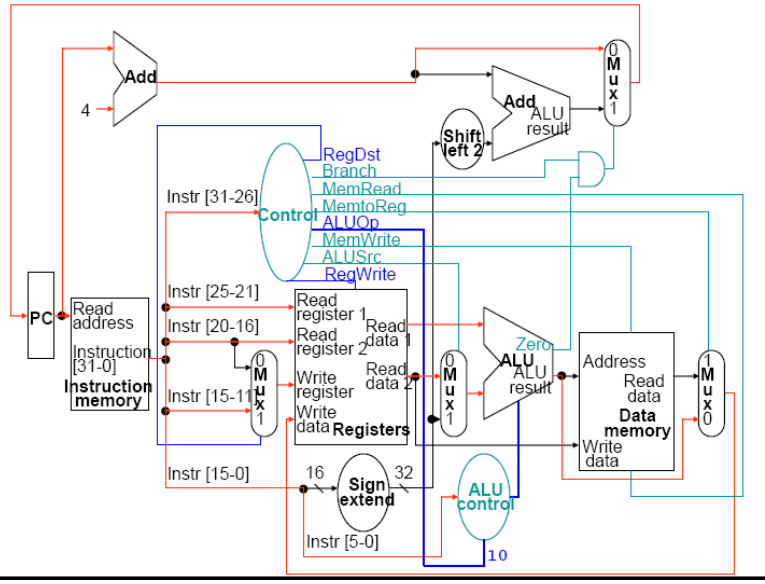
# Veriyolunda İşlem Yapmak




`add rd, rs, rt`

# Veriyolunda İşlem Yapmak

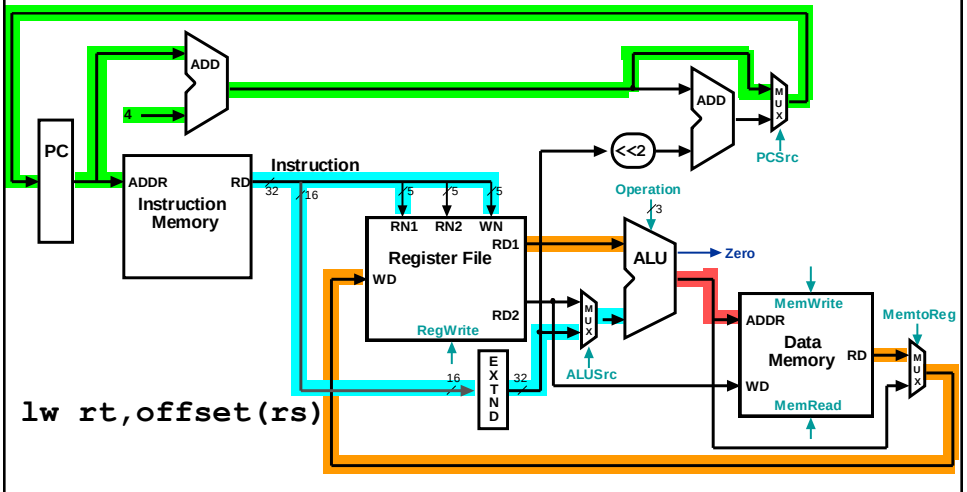
`add r1, r2, r3` komutunun çalışması



## Veriyolunda İşlem Yapmak

- Komut fetch edilir ve PC artırılır.
- r2 ve r3 register'ları okunur ve kontrol birimi kontrol işaretlerini belirler.
- ALU fonksiyon kodlarını (5:0) kullanarak gerekl işlemi yapmak için ALU kontrol işaretini oluşturur.
- ALU nun oluşturduğu sonuç hedef register r1'e (15:11) yazılır.

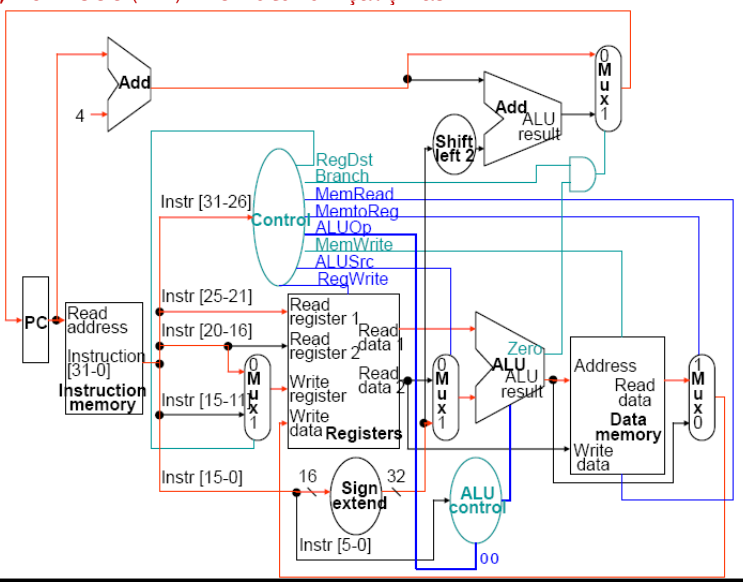
## Veriyolunda İşlem Yapmak




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

## Veriyolunda İşlem Yapmak

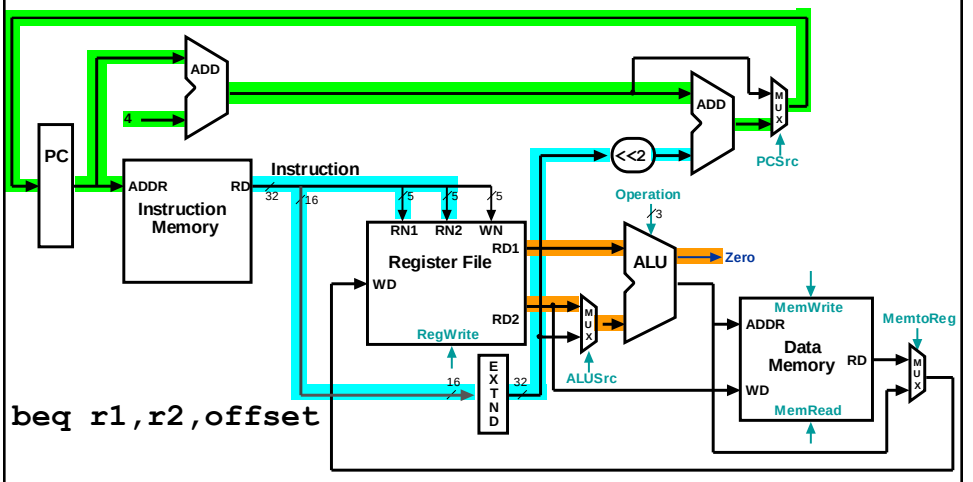
`lw r1, offset(r2)` komutunun çalışması




## Veriyolunda İşlem Yapmak

- Komut fetch edilir ve PC artırılır.
- r2 register'ı okunur.
- ALU sign-extend yaparak offset adresini hesaplar.
- ALU nun oluşturduğu sonuç hedef register r1'e (15:11) yazılır.
- ALU'nun elde ettiği değer hafızadan okunacak adrestir.
- Hafızadan alınan değer register'a (20:16) yazılır.

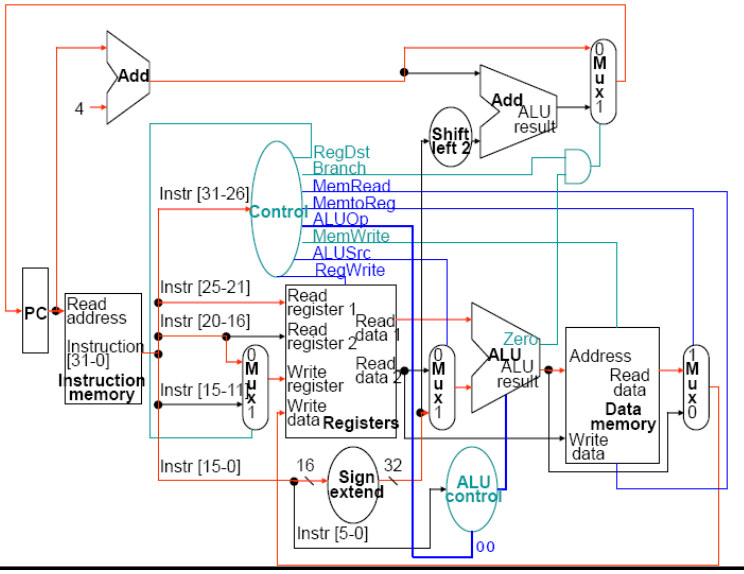
# Veriyolunda İşlem Yapmak




beq r1,r2,offset

# Veriyolunda İşlem Yapmak

beq r1, r2, offset komutunun çalışması



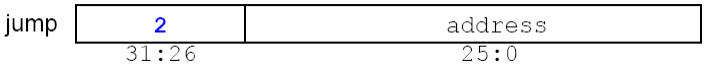
## Veriyolunda İşlem Yapmak

- Komut fetch edilir ve PC artırılır.
- r1 ve r2 register'ları okunur.
- ALU, PC+4 değeriyle komutun 15:0 offset adres bitlerini 2 shift yaparak toplar ve hedef adresi belirler.
- ALU nun zero çıkışı hangi toplayıcı sonucunun PC'ye aktarılacağını belirler.
- ALU'nun elde ettiği değer hafızadan okunacak adrestir.
- Hafızadan alınan değer register'a (20:16) yazılır.

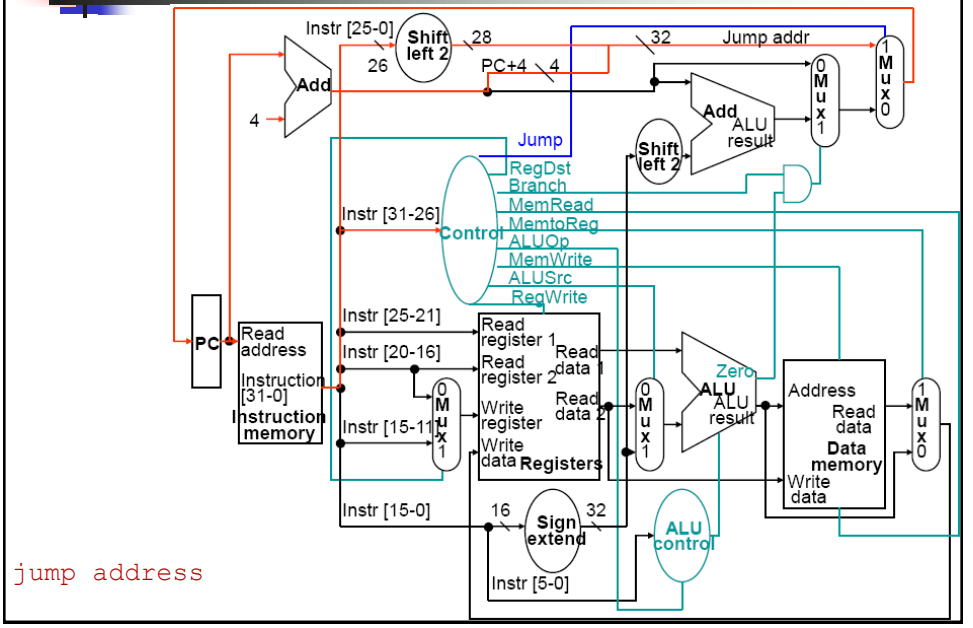
## Veriyolunda İşlem Yapmak

`jump address` – unconditional branch komutunun çalışması

- PC+4 değeriyle komutla immediate verilen 26 bit adres toplanır (0:25).
- Ek bir multiplexer kullanılmalıdır ve,
  - Jump target adres seçilebilir veya
  - PC+4 veya branch target adres'ten birisi seçilebilir.



## Veriyolunda İşlem Yapmak




jump address

## Veriyolunda İşlem Yapmak

- Komut fetch edilir ve PC artırılır.
- Alınan jump komutu ise 0:25 arası bitler alınır ve sola 2 shift edilir.
- Ardından, PC+4 değeriyle toplanır.
- Eklenen multiplexer hesaplanan yeni adresi PC register'ına aktarır.

## Ödev

- ld, st, add, sub, bre komutlarını çalıştıran bir mikroişlemcinin kontrol birimi ve veriyolunu kapı devreleri kullanarak tasarlayınız.
- Kullanılabilecek elemanlar, mux, ve, veya, not, xor.
- ALU içindeki işlem birimlerini detaylı tasarlamaya gerek yoktur.
- Register'ların tümünü D flip-floplarla bit bazında tasarlayınız.
- Hafızadaki saklama birimlerini detaylı çizmeye gerek yoktur.