# BÖLÜM 5. KOMUT SETİ MİMARİSİ

**Adresleme Metotları**

İvedi (Immediate) Adresleme
Direkt (Direct) Adresleme
Dolaylı (Indirect) Adresleme
İndis (Index) Adresleme
Göreceli (Relative) Adresleme
Doğal (Inherent) Adresleme

**Register Transfer Dili**

# KOMUT SETİ MİMARİSİ

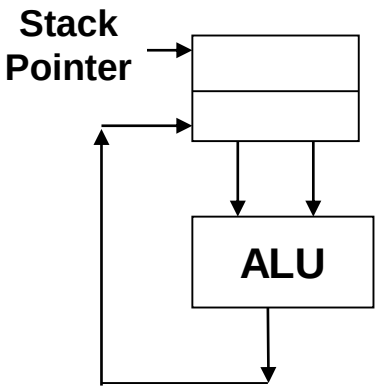
CPU, komutlar üzerinde işlem yapar. Komutlar ise CPU'nun yapması gereken işleri tanımlar. Bir komut, işlem bilgisini (opcode) ve adres bilgisini içerir. Belirtilen adresteki veri operand olarak anılır. Operand ise bellekte ya da kaydedicide olabilir.

Bir işlemcinin komut seti, operandın nereden alınacağına bağlı olarak tasarlanır. Genel olarak 3 tip mimari göze çarpmaktadır;

1. **Yığın (Stack) mimarisi**
2. **Akü mimarisi**
3. **Genel amaçlı register mimarisi**

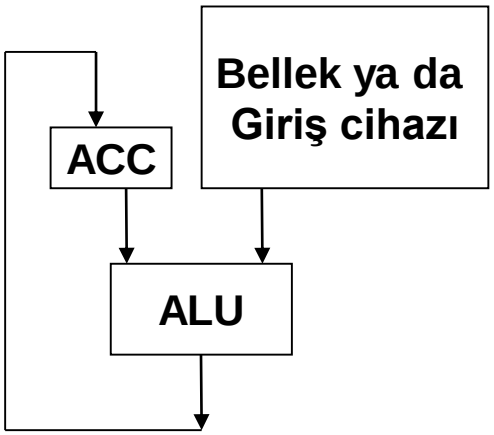
# Yığın (Stack) Mimarisi

İşlemler yığının üst elemanları üzerinde yapılır ve sonuç yine yığına atılır. Bu mimaride yığına bilgi aktarma işlemi *push* ve yığından bilgi alma işlemi de *pop* komutlarıyla olur.



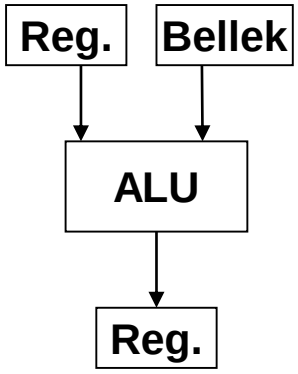
# Akü Mimarisi

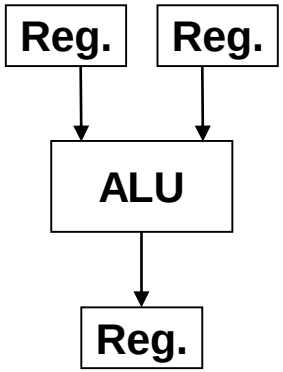
Operandlardan biri bellekte diğeri ise aküdedir (ACC). İşlem yapılır ve sonuç aküye transfer edilir. Akü, komutlarda ayrıca belirtilmez (imalıdır). İleride tasarlayacağımız temel bilgisayar sistemimizde bu mimariyi kullanacağız.



# Genel Amaçlı Register Mimarisi

Bu mimaride operandlar, register ya da bellek olarak belirtilir. Register-bellek ya da register-register tipinde olabilir. Operandlar üzerinde işlemler yapıldıktan sonra sonuçlar kaydedicilere aktarılır.



**Örnek:** Bellekteki iki sayıyı (sayi_1 ve sayi_2) toplayıp, sonucu (toplam) tekrar belleğe koymak istediğimizi düşünelim.

**Stack mimarisinde,**

*Push* sayi_1
*Push* sayi_2
*Add*
*Pop* toplam

**Akü mimarisinde,**

*load* sayi_1
*add* sayi_2
*store* toplam

**Register-bellek mimarisinde,**

*load* Reg_1, sayi_1
*Add* Reg_2, Reg_1, sayi_2
*Store* Reg_2,toplam

**Register-register mimarisinde,**

*load* Reg_1, sayi_1
*load* Reg_2, sayi_2
*Add* Reg_3, Reg_1, Reg_2
*store* Reg_3, toplam

# Adresleme Metotları

Bir komutun, opkod ve adres kısımlarından oluştuğundan bahsedilmişti. Komutun adres kısmı, üzerinde işlem yapılacak operandı gösterir. Operand bellekte ise operandın adresi, etkin adres (effective address) diye anılır.

**1. İvedi (Immediate) Adresleme:** Bu adresleme modunda operand, komutun bir parçasıdır. Yani operand, komutun adres kısmındaki veridir.

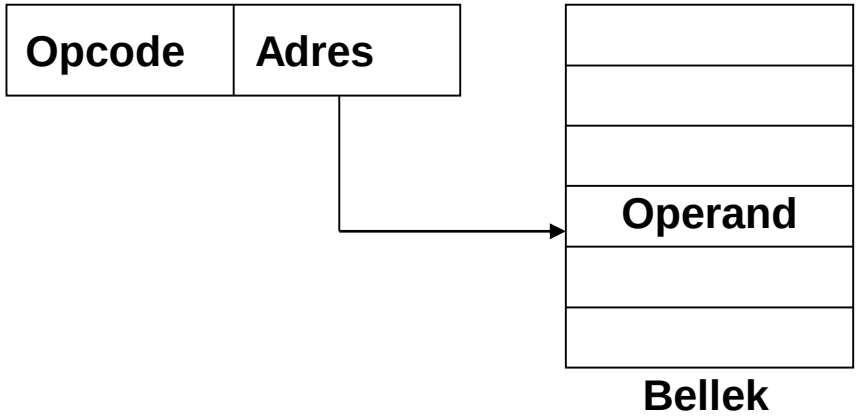
| Opcode | Operand |
|---|---|

**Örnek:** *Add #3* komutunda '3' operanttır. Bu komut akünün içeriğine 3 değerini ekler ve sonucu aküye yazar. Belleğe erişim gerektirmeyen hızlı icra edilen bir komuttur. # sembolü operandın kendisini göstermek için kullanılmıştır

Yüksek seviyeli dillerde değişkenlere sabit bir başlangıç değeri atamak için de bu adresleme metodu kullanılabilir. (x=3)

# Adresleme Metotları

**2. Direkt (Direct) Adresleme:** Bu adresleme metodunda komutun adres kısmı, operandın adresini gösterir.

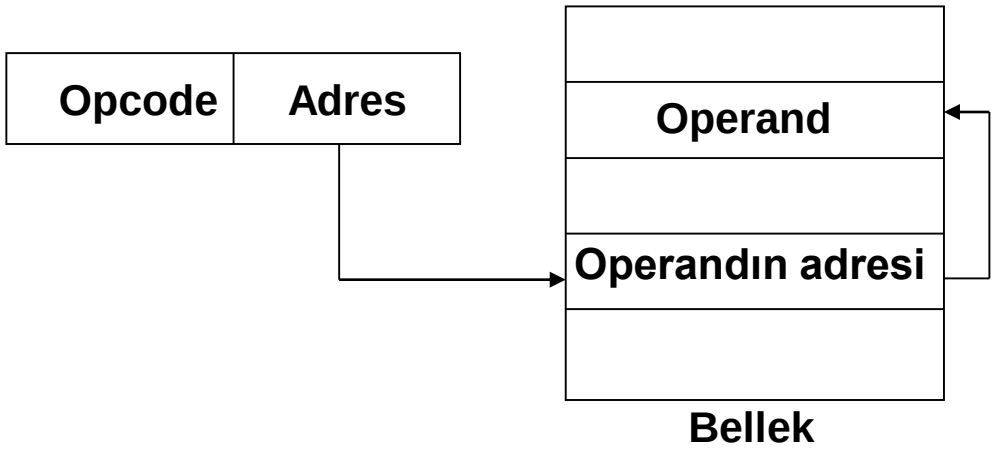



**Örnek:** *Add 100* komutu, akünün içeriğine 100 numaralı bellek bölgesinde tutulan değeri ekler ve sonucu aküye yazar.

- Bu adresleme metodunda bellek referans edildiğinden, komutların işletim zamanı ivedi moda göre daha uzundur
- Sınırlı adres uzayına erişim mümkündür (Adres kısmının içerdiği bit sayısı kadar).

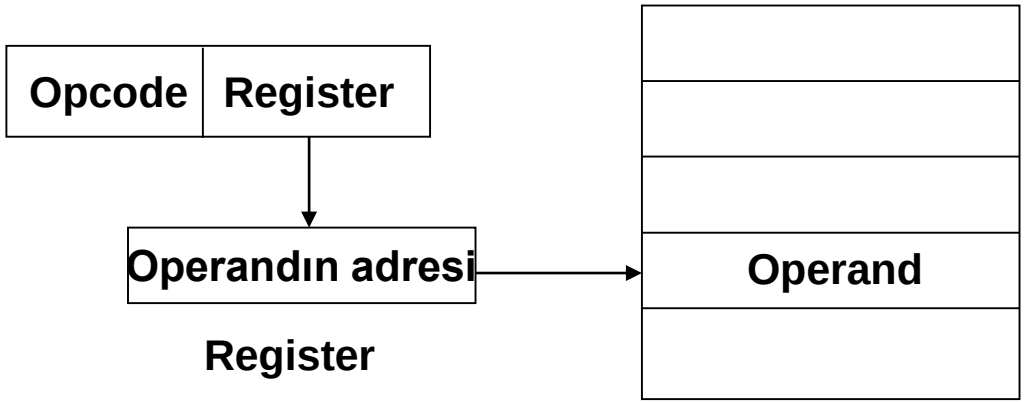
# Adresleme Metotları

**3. Dolaylı (Indirect) Adresleme:** Bu adresleme metodunda komutun adres kısmı, operandın adresini göstermez; operandın etkin adresinin tutulduğu bir register'i ya da bellek bölgesini gösterir.




**Örnek:** *Add (100)* komutu akünün içeriğine, adresi 100 numaralı bellek bölgesinde olan operandın değerini ekler ve sonucu aküye yazar. '( )' sembolleri dolaylı adreslemeyi belirtmek için kullanılmıştır.

Register - dolaylı adresleme modunda, operandın etkin adresi register'da tutulur.



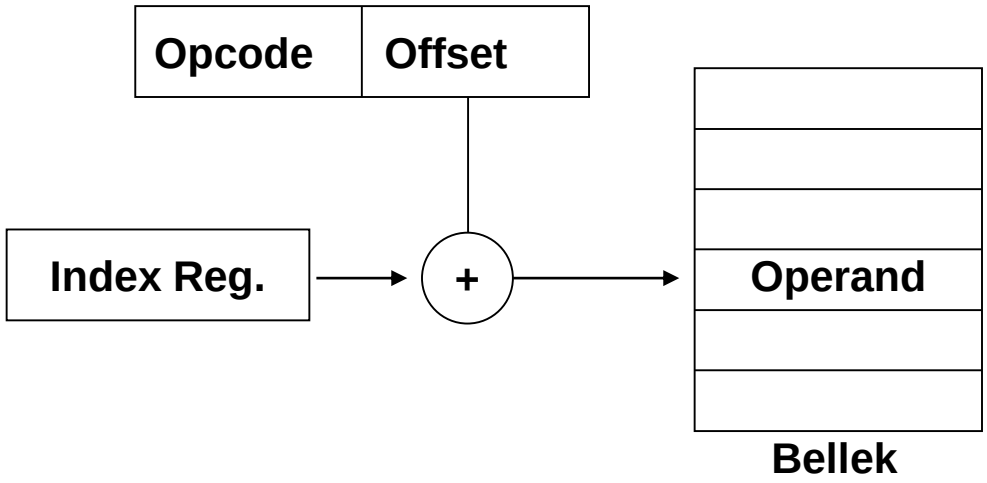

**Örnek:** *Add (R1)*

Özetle;

- Bu adresleme metodunu kullanan komutların işletim zamanı, direkt adreslemeye göre daha uzundur
- Operanda erişim için ekstra efektif adres hesabı gerektirir.
- Adres uzayını genişletmek mümkündür.

# Adresleme Metotları

**4. İndis (Index) Adresleme:** Bu adresleme modunda bir index registerinden faydalanılır. Özellikle bir dizinin elemanlarına sıralı erişim söz konusu ise kullanılabilir. Ulaşılmak istenen dizinin başlangıç adresi index registerine atanır. İstenen dizi elemanının ofset değeri, komutun operand kısmında yer alır.



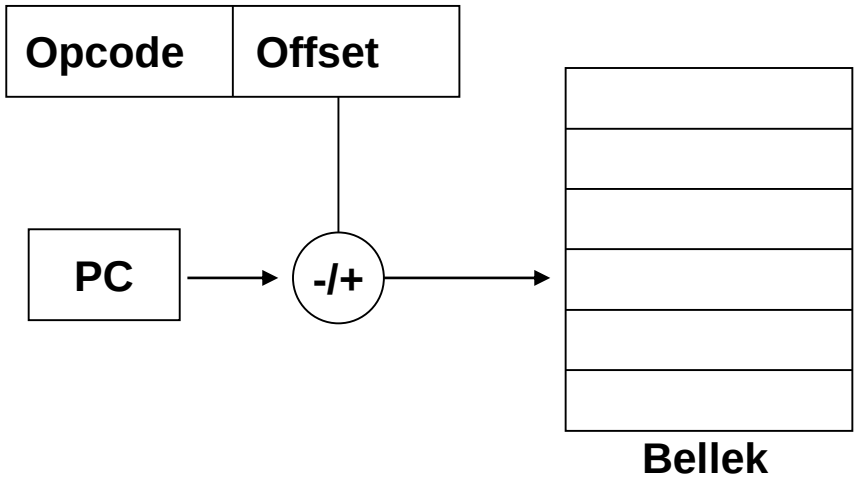

**Örnek:** main {

k=A[2];

}

# Adresleme Metotları

**5. Göreceli (Relative) Adresleme:** Program sayacının (PC) mevcut değerinin yakınlarında bir yere dallanma yapar.




**Örnek:**

```
main{
if (i==1) i++;
else i--;
}
```

# Adresleme Metotları

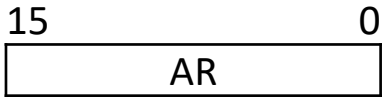
**6. Doğal (Inherent) Adresleme**: Operand bilgisi içermeyen adresleme modudur.

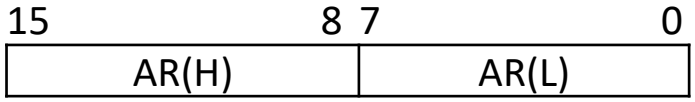
**Örnek:** Clr (Akünün içeriğini temizler)
Inc (Akünün içeriğini 1 arttırır)

# Register Transfer Dili

Bir bilgisayar sistemindeki en temel işlem, registerlar arası bilgi transferleridir. Bu dil, bilgisayar sisteminin donanımsal yapısını ve davranışını tam bir biçimde ifade eder. Donanım tasarım dili (Hardware Description Language-HDL) olarak da bilinir.

- Registerlar genellikle büyük harflerle ifade edilirler. Ayrıca harflerden sonra rakam bilgileri de içerebilirler. n bitlik bir kaydedici içerisindeki flip floplar 0'dan (n-1)'e kadar numaralandırılırlar. Temel bilgisayar sistemimizdeki kaydediciler 8 ya da 16 bitliktir.




- Yüksek anlamlı kısım AR(8-15) ya da AR(H)
- Düşük anlamlı kısım AR(0-7) ya da AR(L)
- Belli bir bit grubu, AR(0-3) ya da $AR_{0-3}$ şeklinde ifade edilir.

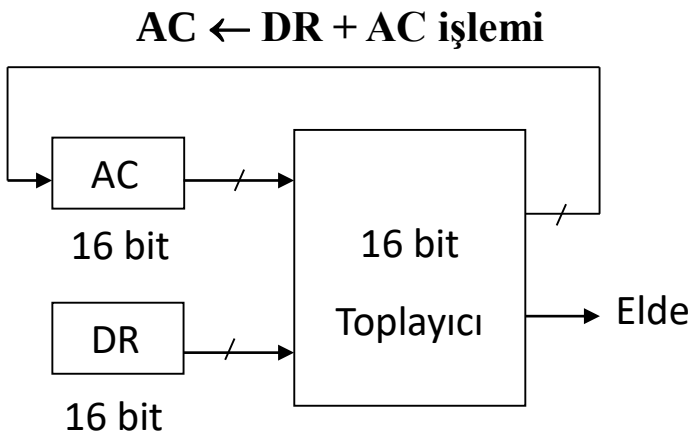
# Register Transfer Dili

- Register transfer işlemi '←' sembolü ile gösterilir ve aktarımın yönünü belirtir.

Örneğin AR ← PC ifadesi PC'nin içeriğinin AR'ye aktarılacağını belirtir (Bit sayılarının uyumlu olması gerektiği unutulmamalıdır). Bu aktarım sonucunda PC'nin içeriği korunur.

- Aritmetik ya da mantık işlemlerinin sonuçları da kaydedicilere aktarılabilir;

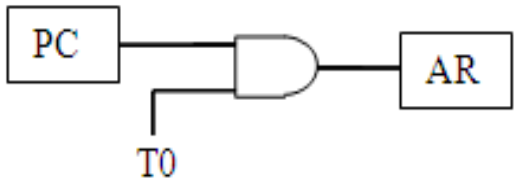
| | |
|---|---|
| AC ← DR + AC | Toplama |
| AC ← DR - AC | Çıkarma |
| AC ← AC' | Tümleyen |
| AC ← AC'+1 | 2'ye tümleyen |
| AC ← AC ∩ DR | VE işlemi |
| AC ← AC ∪ DR | VEYA işlemi |
| AC ← SHR(AC) | Sağa kaydırma |
| AC ← SHL(AC) | Sola kaydırma |



# Register Transfer Dili

- Transfer işlemi bir kontrol sinyali ile olacaksa, bu şart "Kontrol Sinyali :" şeklinde ifade edilir.

Örneğin T0 : AR ← PC




- Belli bir kontrol sinyaline bağlı olarak, birden fazla aktarım (mikro işlem) eş zamanlı olarak yerine getirilmek isteniyorsa "," ile belirtilir;

Örneğin X: AR ← PC, PC ← PC +1

# Register Transfer Dili

Şayet X kontrol sinyalinin durumuna göre değişik atamalar yapılacaksa (if-else yapısı) aşağıdaki notasyon kullanılır;

X : PC ← AR

X' : PC ← TR