MC68000

Derste anlatılan konuları açıklamak için örnek olarak kullanılacaktır.

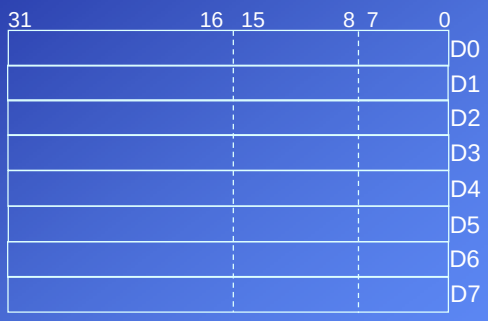
- Veri Yolu 16 bit (Gerektiğinde 8 bit olarak kullanılabilir)
- 16/32 bit mikroişlemci
- 16 adet 32 bitlik veri ve adres saklayıcısı (*Data and Address Registers*)
- Adres Yolu 24 bit: 16-MByte adresleme kapasitesi
- Beş farklı veri üzerinde işlem yapabilir:
  Bit, sekizli (*byte*), 16 bit (*word*), 32 bit (*long word*), BCD
- Bellek haritalı G/Ç (*Memory Mapped Input/Output* -I/O)
- 14 adet adresleme kipi (*Addressing Modes*)
- İki çalışma konumu (modu)
  - *Superviser* (yönetici)
  - *User* (kullanıcı)
    - Bazı komutlar çalıştırılamaz
    - Bellek kod çözücü uygun şekilde tasarlanarak bellek erişiminde kısıtlamalar getirilebilir.

Programlanabilir Saklayıcılar (*User Programmer's Model*)

Veri Saklayıcıları (*Data Registers*):

8 adet özdeş saklayıcı

8, 16, 32 bit olarak kullanılabilir



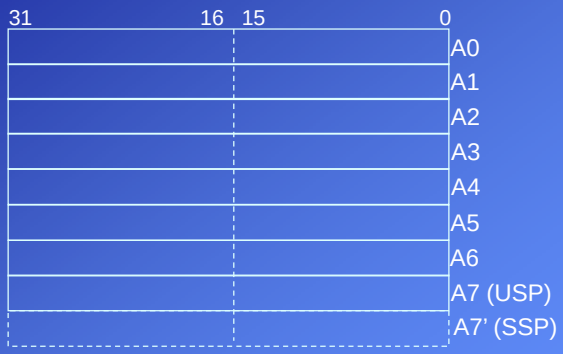
Adres Saklayıcıları (*Address Registers*):

8+1 adet. İşaretçi (*Pointer*) olarak kullanılır.

16, 32 bit olarak kullanılabilir.

A7 yığın işaretçisidir.

Supervisor konumunda ayrı bir yığın işaretçisi daha vardır: *System Stack Pointer*



Adres yolu 24 bit olduğu için saklayıcıdaki verinin ilk 24 biti kullanılır.

Saklayıcı 16 bit olarak kullanıldığında adres yoluna işaret uzatma ile 24 bit çıkarılır.

Durum Saklayıcısı (*Satus Register*):

- 16 bit
- İki kısımdan oluşur: Yönetici ve kullanıcı (*CCR Conditon Code Register*)

Overflow (V), Zero (Z), Negative (N), Carry (C), Extend (X).

Interrupt mask (I_0 I_1 I_2)

Trace (T) mode, Supervisor (S) state

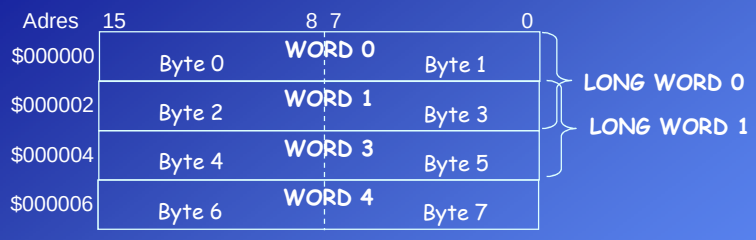
5, 6, 7, 11, 12, 14 numaralı bitlerin anlamları tanımlanmamıştır. Bu alanlar sonraki işlemciler için ayrılmıştır.

Program Sayacı (*Program Counter* - PC):

- 32 bit
- Adres saklayıcısı olarak da kullanılır.

Verilerin Bellekte Yerleşimi

Verilerin yüksek anlamlı kısımları bellekte küçük adresten başlayarak yer alırlar



Adresleme sekizli (*byte*) düzenine göre yapılır.
Word (16 bit) ve *long word* (32 bit) tipi veriler çift adreslerden başlar.

Adresleme Kipleri

Altı temel kip vardır. Bunların türevleri ile birlikte 14 adresleme kipi oluşur.

1. Saklayıcı doğrudan (*Register Direct*)
2. İvedi (*Immediate*)
3. Mutlak (*Absolute*)
4. Saklayıcı dolaylı (*Register Indirect*)
5. Program Sayacı dolaylı, bağıl (*Program Counter Relative*)
6. Doğal (*Implied*)

1a. Veri Saklayıcısı Doğrudan Adresleme

İşleme giren veri bir veri saklayıcısındadır.

MOVE.W D_n , D_m $D_n \rightarrow D_m$

B: *Byte*, W: *Word*, L: *Long*

1b. Adres Saklayıcısı Doğrudan Adresleme

İşleme giren veri bir adres saklayıcısındadır. Eğer hedef adres saklayıcısı ise komut "A" ile biter.

MOVEA.W D1 , A5 $D_1 \rightarrow A_5$ (Kaynak veri saklayıcısı, hedef ise adres saklayıcısıdır.)

Sadece W: *Word* veya L: *Long* olabilir.

2a. İvedi Adresleme

İşleme giren veri komutun içinde yer alır.

MOVE.L #$4A7F0000 , D0

2b. Hızlı İvedi Adresleme (*Quick*)

Sadece bazı komutlar ile birlikte kullanılır.

Komut daha az yer kaplar ve daha hızlı çalışır.

Örneğin MOVE komutunda 8 bitlik veriler için kullanılır.

MOVEQ #5 , D0 D0'ın 32 bitlik kısmı etkilenir.

3a. Mutlak Adresleme (Kısa)

Verinin 16 bitlik adresi komutta yer alır. 16 bitlik adres işaret uzatılarak 24 bit yapılır.

MOVE.B D0, ($58AA) $0058AA adresine yazılır

MOVE.B D0, ($B51A) $FFB51A adresine yazılır

3b. Mutlak Adresleme (Uzun)

24 bitlik adres komutta yer alır.

MOVE.W ($45C720),D7 $45C720 adresinden başlayan 16 bit D7'ye yazılır

4. Saklayıcı Dolaylı Adresleme

4a. Adres Saklayıcısı Dolaylı Adresleme

Bir adres saklayıcısı işleme girecek olan adresi taşır.

Örnek:

MOVE.L D0 , (A0) D0'ın içeriği A0'ın gösterdiği adrese yazılır.

A0: 00001000, D0: 4350A7C8

Komut yürütüldükten sonra:

Belleğin durumu:

001000: 43 MOVE.W olsa sadece A7C8 aktarılacaktı.

001001: 50 MOVE.B olsa sadece C8 aktarılacaktı.

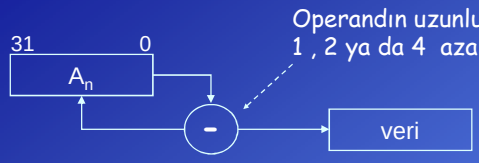
001002: A7

001003: C8

A0'ın içeriği değişmez.

Verinin yüksek anlamlı kısmı küçük adreslere yazılır.

4b. Adres Saklayıcısı Dolaylı Önceden Eksiltmeli Adresleme (*Predecrement*)



$A_n - (1, 2, 4) \rightarrow A_n$
B W L

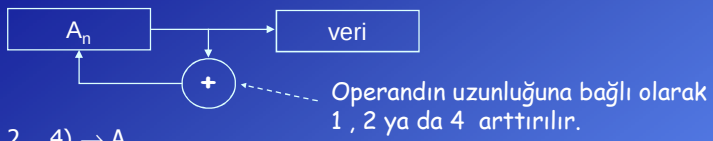
Örnek:

MOVE.W D0 , - (A0) Önce A0 2 azaltılır sonra D0'ın içeriği A0'ın gösterdiği yere yazılır.

A0: 00001002, D0: 3725A100
Bellek:
001000: A1
001001: 00
Komut yürütüldükten sonra A0=00001000 olur.

Azalan adreslere doğru yazma işlemi yığına yazma (PUSH) için kullanılabilir.

4c. Adres Saklayıcısı Dolaylı Sonradan Arttırmalı Adresleme (*Postincrement*)



$A_n + (1, 2, 4) \rightarrow A_n$
B W L

Örnek:

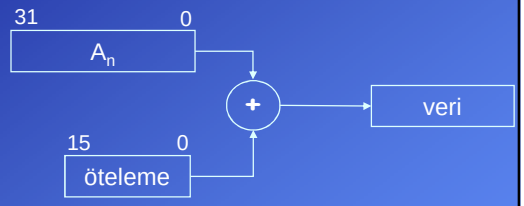
MOVE.W (A0)+ , D0
Önce A0'ın gösterdiği yerdeki 16 bitlik veri D0'a yazılır sonra A0 2 arttırılır.
A0: 00001000, D0: XXXXXXXX
Bellek:
001000: A1
001001: 00
Komut yürütüldükten sonra A0=00001002, D0: XXXXA100 olur.

Yığından okuma (PULL) için kullanılabilir.
Önceden eksiltme ve sonradan arttırma kipleri yığın ve kuyruk işlemleri için kullanılır. 68000'in ayrıca yığın komutları yoktur.

4d. Adres Saklayıcısı Dolaylı Ötelemeli (*with displacement*)

Öteleme 16 bitlik işaretli sayıdır.
A_n'in gösterdiği yerden 32K ileri, 32K geri adresleme yapılabilir.
Örnekler:

```
MOVE.B   - 4(A2) , D0
MOVE.W   $0C(A5) , D7
MOVE.L   -2(A3) , 12(A5)
```

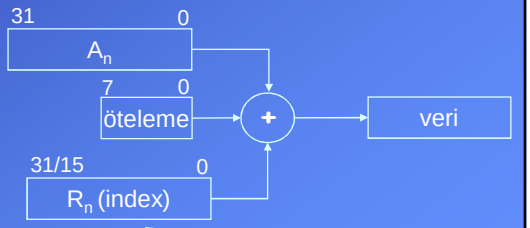


4e. Adres Saklayıcısı Dolaylı Ötelemeli ve Sıralı (*with displacement and index*)

Örnek:

```
MOVE.W   - 2(A3,D5.W) , 4(A5)
```

A3: Taban
D5: Sıra (*index*)
-2 : Öteleme

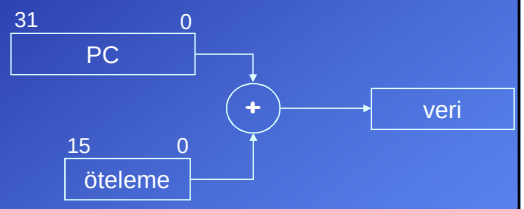


A_n veya D_n olabilir.
16 ya da 32 bitlik kısımlar kullanılabilir.

5a. Program Sayacı Dolaylı Ötelemeli , Bağıl Adresleme

Sabit adresler kullanılmaz.
Verilerin adresi yürütülmekte olan komutun adresine bağıl olarak belirtilir.
Program farklı adreslere yerleştiğinde de çalışabilir.
Örnek:

```
MOVE.B   50(PC) , D5
```

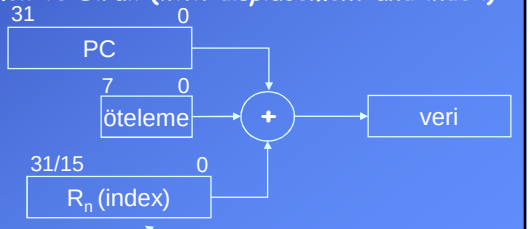


5b. Program Sayacı Dolaylı Ötelemeli ve Sıralı (*with displacement and index*)

Örnek:

```
MOVE.W   - 2(PC,D5.W) , 4(A5)
```

PC: Taban
D5: Sıra (*index*)
-2 : Öteleme



6. Doğal adresleme (*implied*)

Ayrıca operand verilmez. Operand komut kodunda içerilir.
RTS, TRAPV, NOP

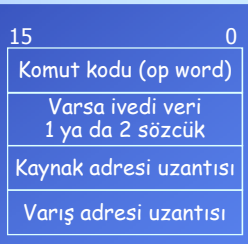
A_n veya D_n olabilir.
16 ya da 32 bitlik kısımlar kullanılabilir.

MC68000'de Komut Yapısı

Komutların uzunluğu 16 bit ve katları şeklindedir.

Bir komut en az 1 sözcük en fazla 5 sözcük uzunluğundadır.

Komut kodunda (ya da işlem sözcüğü) (*op word*) komutun işlevi ve operandların adresleme kipleri ile ilgili bilgiler yer alır.



Komut Yapılarına Örnekler

Bir Operandlı Komut Yapısı:

İşlem Sözcüğü:

00: B
01: W
10: L

Mode ve Register alanlarına ilişkin kodlar için kullanıcı kılavuzuna bakınız.

Örnek:

CLR.W D3 | 01000010 01 000 011

CLR W D 3

Örneklerin devamı:

CLR.L (A2)+ | 01000010 10 011 010

CLR L (An)+ 2

Adres saklayıcısı dolaylı sonradan arttırmalı

CLR.B ($3000) | 01000010 00 111 000

CLR B Mutlak kısa

0011 0000 0000 0000

Mutlak adresleme (kısa)

Adres ($3000) ikinci sözcükte yer alır.

CLR.B $4(A6) | 01000010 00 101 110

CLR B d(An) 6

0000 0000 0000 0100

Adr. Sakl. Dolaylı ötelemeli

Öteleme ($4) 16 bit olarak ikinci sözcükte

CLR.B -7(A6) | 01000010 00 101 110

CLR B d(An) 6

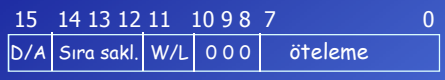
1111 1111 1111 1001

Adr. Sakl. Dolaylı ötelemeli (öteleme negatif)

Öteleme (-7) 16 bit olarak ikinci sözcükte

Sıralı ve Ötelemeli Adresleme için ek sözcük:

Sıralı ve ötelemeli adresleme kipinde aşağıdaki yapıda bir ek sözcük kullanılır.
Ek sözcükte sıra saklayıcısı ve öteleme ile ilgili bilgiler yer alır.



0:D 1:A — 0:W 1:L — Öteleme için 8 bitlik yer ayrılmıştır.

Örnek:

CLR.W $C2(A3, D7.L)

01000010 01 110 011
0 111 1 000 11000010
D 7 L $C2

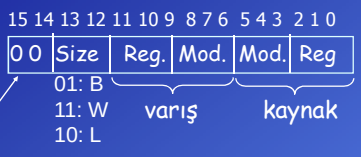
CLR.W d(A3,Rn.S)

Sıra sakl. ve öteleme ile ilgili bilgiler ek sözcükte yer alıyor.

Komut Yapılarına Örnekler

İki Operandlı Komut Yapısı:

Örnek: MOVE



Komut sadece bu alana göre çözülmez, tüm işlem sözcüğü kullanılır.

Örnek:

MOVE.W D2, (A5)+

00 11 101 011 000 010
W 5 (An)+ D 2

Örnek:

MOVE.W #$1234 , $25(A3)

W 3 d(An) ivedi	
00 11 011 101 111 100	
0001 0010 0011 0100	$1234 (ivedi veri)
0000 0000 0010 0101	$0025 (varış adersi uzantısı)

Quick Komut Yapısı:

- Sekiz bitlik ivedi veriler kullanılır.
- Varış bir veri saklayıcısıdır.
- Saklayıcının 32 biti etkilenir.
- Normal ivedi adreslemeden daha az yer kaplar.

MOVEQ

15 11	10 9	8	7 0
0111	Reg.	0	veri

Örnek:

MOVEQ #-5 , D2

0111	010	0	1111 1011
			-5

Aynı işlemi yapan normal MOVE komutu 3 sözcük (48 bit) yer kaplar.

MOVE.L #-5 , D2

L 2 D ivedi
00 10 010 000 111 100
1111 1111 1111 1111
1111 1111 1111 1011

32 bitlik -5

ADDQ komutu 3 bitlik veriler ile çalışır.

MC68000 Komutları

Bu bölümde MC68000 mikroişlemcisinin bazı komutları tanıtılacaktır.

Veri Aktarma Komutları:

MOVEM ***Move multiple registers***

Belirtilen tüm saklayıcıları belli bir adresten itibaren belleğe yazar ya da, belirtilen bir bellek adresinden verileri okuyarak istenilen saklayıcılara yerleştirir.

Kullanım şekli 1: MOVEM <register list>,<ea>
Kullanım şekli 2: MOVEM <ea>,<register list>

Örnekler:
```
MOVEM.L   D0-D7/A0-A6 , $1234
M0VEM.L   (A5) , D0-D2/D5-D7/A0-A3/A6
MOVEM.W   D0-D5/D7/A0-A6 , -(A7)
MOVEM.W   (A7)+, D0-D5/D7/A0-A6
```

Altprogramların başında ve sonunda saklayıcıları yığına yazmak/yığından okumak için kullanılabilir.

```
MOVEM.L D0-D5/A0-A3,-(A7)      Saklayıcılar yığına yazıldı
   ...
Alt programın gövdesi
   ...
MOVEM.L (A7)+,D0-D5/A0-A3      Saklayıcıların değerleri yığından geri alınıyor
RTS                            Çağıran programa dönüş
```

LEA Load effective address

[An] ← <ea>

Bir değişkenin adresini bir adres saklayıcısına almak için kullanılır.

Adres 32 bit olarak hesaplanır.

Örnekler: LEA Table,A0
LEA (Table,PC),A0
LEA (-6,A0,D0.L),A6
LEA (Table,PC,D0),A6

Örnek:

```
        LEA     DIZI , A0       Dizi adresi A0'a
        MOVE.B (A0)+, D1        Dizinin ilk elemanı D1'e...
        ...
DIZI    DS.B    100             Define Storage (direktif)
```


Akış Denetimi (*Flow control*) Komutları:

Bcc Branch on condition cc

cc koşulu belirtir.

If cc = 1 THEN [PC] ← [PC] + d

d: 8 ya da 16 bitlik işaretli bağıl adrestir.

Hatırlatma: Komut yürütülürken PC, Bcc'den sonraki komuta işaret eder.

Yazım: Bcc <label>

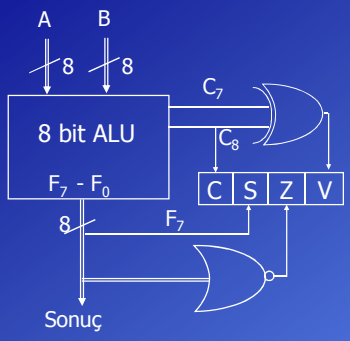
İstenirse bağıl adres boyutu verilebilir: BEQ.B ESIT ya da BNE.W FARKLI

Boyut verilmezse derleyici etiketin uzaklığına göre uygun boyutta bağıl adresi hesaplar.

Koşullar (cc):

BCC	branch on carry clear	branch if C = 0
BEQ	branch on equal	branch if Z=1
BGT	branch on greater than	branch if (Z + (N ⊕ V)) = 0
BHI	branch on higher than	branch if (C + Z) = 0
BGE	branch on greater than or equal	branch if (N ⊕ V) = 0
BLT	branch on less than	branch if (N ⊕ V) = 1
BLS	branch on lower than or same	branch if (C + Z) = 1

Bayrakların Oluşumu:



Taşma:
$V = C_7 \oplus C_8$
C_8: Elde
C_7: Bir önceki basamaktaki elde

Taşma diğer bir yolla da belirlenebilir:
poz + poz → neg poz – neg → neg
neg + neg → poz neg – poz → poz

Çıkarma ve karşılaştırma işlemlerinde **C** (elde) biti BORÇ bayrağı olarak görev yapar.

Hatırlatma:

Elde: İşaretsiz sayıların toplanmasında oluşabilir. Sonucun n bite sığmadığını (n+1). bitin gerekli olduğunu gösterir.

Borç: İşaretsiz sayıların çıkartılmasında oluşabilir. Birinci sayının ikinciden küçük olduğunu, sonucun negatif çıktığını gösterir.

2'ye tümleyen yöntemine göre yapılan çıkarmada n+1. bit oluşursa borç yoktur.

Taşma: Sadece işaretli sayılar üzerinde yapılan toplama ve çıkarma işlemlerinde oluşur. Sonucun, ayrılan bit sayısı ile ifade edilemediğini gösterir.

DBcc Test condition, decrement, and branch

Yazım: DBcc Dn,<etiket>

Burada etiket 16 bitlik bir bağıl adrestir.

Dn'in 16 bitlik kısmı sayaç olarak kullanılır.

İşlem:

IF(condition cc false)
  THEN [Dn] ← [Dn] - 1 (*decrement loop counter*)
    IF [Dn] = -1 THEN DBcc'den sonraki komut (PC alma çevriminde 2 arttırılmıştır.)
      ELSE [PC] ← [PC] + d (bağıl olarak dallan)
  ELSE DBcc'den sonraki komut (PC alma çevriminde 2 arttırılmıştır.)

Örnek: Döngü (10 defa)

```
        MOVEQ   #9 , D0      Başlangıç değeri 9, çünkü -1'de çıkılıyor.
L1      ......               Döngü içi
        ......
        DBF     D0,L1        Burada F : False koşul her zaman yanlış
```

Örnek: İki Dizinin Karşılaştırılması (Tüm elemanlar eşit mi?)

Birinci dizi DIZI1 adresinden, ikinci dizi ise DIZI2 adresinden başlıyor.
Dizilerde 50 adet 8 bitlik eleman bulunuyor.
Dizilerin içeriği program çalışmadan önce doldurulmuştur.

```
        LEA       DIZI1, A0        Dizilerin başlangıç adresleri
        LEA       DIZI2, A1
        MOVE.W    BOYUT, D0        Dizilerin boyu
        SUBQ.W    #1, D0           DBcc'den -1'de çıkılır
LOOP    CMPM.B    (A0)+, (A1)+     Dizi elemanları karşılaştırılıyor
        DBNE      D0, LOOP
        TST.W     D0               Döngüden neden çıkıldı? (D0?)
        BMI       ESIT             -1'de çıkıldıysa tüm elemanlar eşit
FARKLI  .......
        .......
ESIT    .......
        .......
DIZI1   DS.B      50               1nci dizinin elemanları için bellekte yer ayrılıyor 50B
DIZI2   DS.B      50               2nci dizinin elemanları için bellekte yer ayrılıyor 50B
BOYUT   DC.W      50               Dizilerde 50 tane eleman var
```