HAFTA 4

MİKROİŞLEMCİLER

Adresleme Modları

- Adresleme Modları CPU tasarımlarında komut seti mimarisinin önemli bir parçasını oluşturmaktadır.
- Herhangi bir komut seti mimarisinde tanımlı çeşitli adresleme modları, o mimarideki makine dili kodlarının her bir komutun operand veya operandlarını nasıl kullanacaklarını belirler.
- Adresleme modu, bir makine kodu dahilindeki veya herhangi bir noktadaki saklayıcılarda ve/veya sabitlerdeki bilgileri kullanarak bir operandın etkin bellek adresini nasıl hesaplayabileceğini belirler.
- Amaç, mikrodenetleyicinin saklayıcılarına erişmek, bellek alanlarına erişmek, program belleğinde istenen yere erişerek program yönlendirme işlemlerini gerçekleştirmektir.

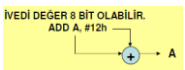
Adresleme Modları

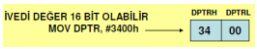
- İvedi Adresleme (Immediate Adressing)
- Saklayıcı Adresleme (Register Adressing)
- Direkt Adresleme (Direct Adressing)
- Dolaylı Adresleme (Indirect Adressing)
- Bağıl Adresleme (Relative Adressing)
- Mutlak Adresleme (Absolute Adressing)
- Uzun Adresleme (Long Adressing)
- Indeksli Adresleme (Index Adressing)

İvedi Adresleme

- İvedi adreslemede komutun kendisi kullanılacak değeri içermektedir. Bu daha yüksek seviye dillerde örneğin 7 veya 39 gibi sabitleri kullanmak gibidir.

```
#define PI 3.14
...
Alan = PI * r * r
```





Saklayıcı Adresleme

- Komutun operandları belirli saklayıcılara işaret etmektedir.
- Bu saklayıcılar
  - Mikrodenetleyicinin bankları içinde bulunan saklayıcılar (R0-R7)
  - Akümülatör gibi özel amaçlı saklayıcılar

```
ADD A, R7
MOV R1, #35H
```

Direkt Adresleme

- Komut doğrudan adres bilgisini içermektedir.

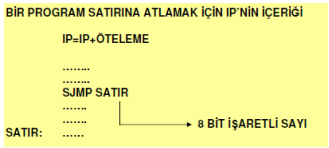
```
MOV 40H, 30H
ADD A, 40H
MOV A, 90H      ➔   MOV A, P1 ;(P1➔ 90H)
```

Dolaylı Adresleme

- Komut operand olarak saklayıcıların içeriklerini adres olarak almaktadır.
- Saklayıcının (R0-R1) içeriği işaretçi olarak kullanılır.

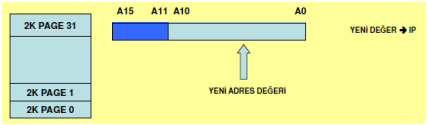
```
MOV 40H, @R0
ADD A, @R1
```

Bağıl Adresleme



Mutlak Adresleme

- ACALL ve AJMP komutları ile kullanılır.
- 2 KB'lik hareket alanı vardır.
- IP'nin içeriğindeki 11 bitlik adres, verilen operand ile değiştirilir.



Uzun Adresleme

- LCALL ve LJMP komutları kullanılır.
- Hareket alanı 64 KB'lık alandır.
- 64 KB alan içerisinde istediği herhangi bir noktaya gidebilir.

YENİ ADRES DEĞERİ → IP

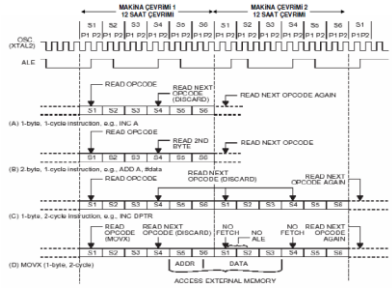
Indeksli Adresleme

- Komut baz saklayıcısı ile offseti içerir.

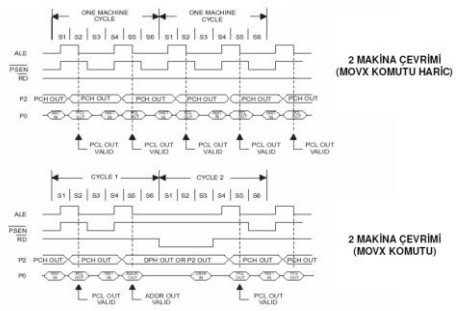
BASE SAKLAYICISI + OFSET SAYISI → YENİ ADRES

JMP @A +DPRT JMP @A + PC	→	(IP VEYA DPTR) + Acc → IP
MOVC A, @A+DPTR MOVC A, @A+PC		Acc← (Acc+DPTR) Acc← (Acc+PC)

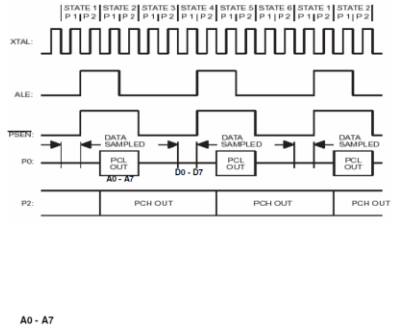
Makine çevrimi ve durum işaretleri



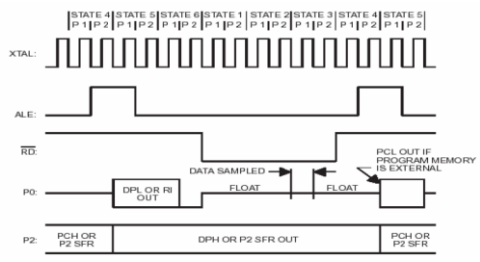
Harici Program Hafızadan Yürütme



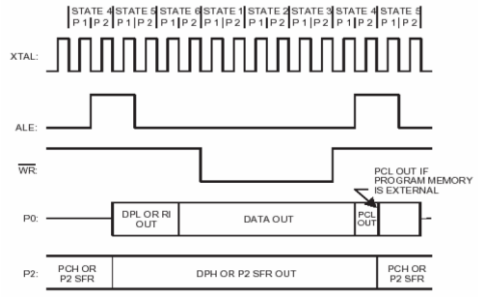
Harici bellek (program ve veri) okuma (FETCH)



Harici bellek (program ve veri) okuma (READ)



Harici bellek (program ve veri) okuma (WRITE)



8051 TEMELLİ MİKRODENETLEYİCİ KOMUT SETİ

- Veri Transfer Komutları
- Aritmetik işlem komutları
- Lojik işlem komutları
- Program Yönlendirme Komutları

VERİ TRANSFER KOMUTLARI

- MOV
- MOVC
- MOVX
- PUSH
- POP
- XCH
- XCHD
- SWAP

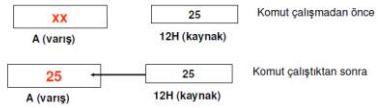
MOV

MOV varış, kaynak

Amaç: Kaynaktan Varışa veri transferi

Etkilenen bayrak: YOK

Örnek: *MOV A, 12H*



MOV

MOV varış, kaynak

KOMUT	ÇEVRİM	KODLAMA			İşlem
MOV A, Rn	1	11101 *rrr*			(A) ← (Rn)
MOV A, direct	1	11100101	direct_adr		(A) ← (dır)
MOV A, @Ri	1	1110011*i*			(A) ← ((Ri))
MOV A, #data	1	01110100	imm_data		(A) ← #data
MOV Rn, A	1	11111 *rrr*			(Rn)← (A)
MOV Rn, direct	2	10101 *rrr*	direct_adr		(Rn)← (dır)
MOV Rn, #data	1	01111 *rrr*	imm_data		(Rn)← #data
MOV direct, A	1	11110101	direct_adr		(dır)← (A)
MOV direct, Rn	1	10001 *rrr*	direct_adr		(dır)← (Rn)
MOV direct, direct	2	10000101	direct_adr	direct_adr	(dır)← (dır)
MOV direct, @Ri	2	1000011*i*	direct_adr		(dır)← ((Ri))
MOV direct, #data	2	01110101	direct_adr	imm_data	(dır)← #data
MOV @Ri, A	1	1111011*i*			((Ri))← (A)
MOV @Ri, direct	1	1010011*i*	direct_adr		((Ri))← (dır)
MOV @Ri, #data	1	0111011*i*	imm_data		((Ri))← #data

MOV - MOVC

MOV DPTR, #data16

Amaç: DPTR saklayıcısına 16 bit ivedi değer yüklemek
Etkilenen bayrak: YOK

MOVC A, @A+DPTR *MOVC A, @A+PC*

Amaç: program hafızada bulunan başvuru tablosuna indisli adresleme ile erişmek
Etkilenen bayrak: YOK

```
        MOV A, SIRA_NO
        CALL BAŞVURU
        ----------
        ----------
BAŞVURU:
        MOV DPTR, #TABLO
        MOVC A, @A+DPTR
        RET
TABLO:  DB 12, 24, .....
```

MOVX

MOVX varış, kaynak

Amaç: Harici veri hafıza erişimi (okuma ve yazma amacı ile)
Etkilenen bayrak: YOK

MOVX A, @Ri (Ri ile 0-255 byte arasına erişim)
MOVX A, @DPTR (DPRT ile 0-64 K arasına erişim)
MOVX @Ri, A
MOVX @DPTR, A

```
MOV DPTR, #0125H
MOVX A, @DPTR

MOV DPTR, #0125H
MOV A, R1
MOVX @DPTR, A
```

```
MOV R0, #25H
MOVX A, @R0

MOV R0, #25H
MOV A, #FH
MOVX @R0, A
```

PUSH - POP

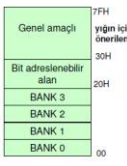
PUSH direct

Amaç: yığına veri atmak
Etkilenen bayrak: YOK

İşlem: SP ← SP+1
Yığına veri at

POP direct

Amaç: Yığından veri almak
Etkilenen bayrak: YOK
İşlem: Yığından veri al
SP ← SP-1



Yığın Kullanılan durumlar

1. POP komutu
2. PUSH komutu
3. CALL komutu
4. RET komutu
5. INT İşlemi
6. RETI komutu

ALT PROGRAM İÇİNDE YIĞIN İŞARETÇİNİN İÇERİĞİNİ BOZARAK RET KOMUTUNU ÇALIŞTIRMAK HATAYA NEDEN OLUR.

```
MOV SP, #70h
.........
.........
CALL PRG
.........

PRG:
        .........
        .........
        PUSH 80H
        .........
        RET
```

XCH – XCHD - SWAP

XCH A, <BYTE> *exchange Acc with byte variable*

Amaç: (A)↔(BYTE)
Etkilenen bayrak: YOK

XCH A, Rn (A) ↔ (Rn)
XCH A, direct (A) ↔ (dir)
XCH A, @Ri (A) ↔ ((Ri))

XCHD A, @Ri *exchange digit*

Amaç: (A3....A0)↔((Ri3.....Ri0)
Etkilenen bayrak: YOK

SWAP A

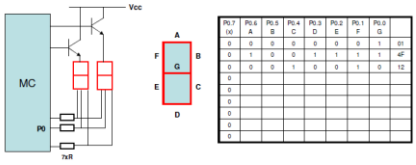
Amaç: (A3....A0)↔(A7....A4)
Etkilenen bayrak: YOK

10H Adresindeki BCD Değerin 7SD'de görüntülenmesi

```
DEĞERSİZ
SEGMENT İÇİN

MOV A, 10H
ANL A, #00001111b
CALL DISPLAY
MOV P0, A
.....
.....
```

```
DEĞERLİ
SEGMENT İÇİN

MOV A, 10H
SWAP A
ANL A, #00001111b
CALL DISPLAY
MOV P0, A
.....
```

```
DISPLAY:
  INC A
  MOVC A, @A+PC
  RET

DB: 01, 4FH, 12H,.....
```



P0.7 (x)	P0.6 A	P0.5 B	P0.4 C	P0.3 D	P0.2 E	P0.1 F	P0.0 G	
0	0	0	0	0	0	0	1	01
0	1	0	0	1	1	1	1	4F
0	0	0	1	0	0	1	0	12
0								
0								
0								
0								
0								
0								

MOV (Bit)

MOV C, BIT *MOV BIT, C*

Amaç: Veri biti kopyalama
Etkilenen bayrak: YOK

F (x, q) = x*q sonucunu bulunuz. x=p1.0 q=p1.1 F=p1.2 olsun

```
MOV C, P1.0
ANL  C, P1.1
MOV P1.2, C
```

Aritmetik İşlem Komutları

- ADD
- ADDC
- DA
- SUBB
- MUL
- DIV
- INC
- DEC

ADD - ADDC

ADD A, <byte> *ADDC A, <byte>*

Amaç: Toplama işlemi
Etkilenen bayrak: CY, AC, OV

ADD A, Rn	(A) ←(A)+(Rn)
ADD A, direct	(A) ←(A)+(dir)
ADD A, @Ri	(A) ←(A)+ ((Ri))
ADD A, #data	(A) ← (A)+ #data
ADDC A, Rn	(A) ←(A)+(Rn)+C
ADDC A, direct	(A) ←(A)+(dir) +C
ADDC A, @Ri	(A) ←(A)+ ((Ri)) +C
ADDC A, #data	(A) ← (A)+ #data+C

16 bitlik işaretsiz iki sayının toplama işlemini yapan program parçasını yazınız.
SAYI1 + SAYI2= SONUC

```
Tanımlar
Sayı1_L equ 10h
Sayı1_H equ 11h
Sayı2_L equ 12h
Sayı2_H equ 13h
Sonuc_L equ 14h
Sonuc_M equ 15h
Sonuc_H equ 16h
```

```
MOV A, SAYI1_L
ADD A, SAYI2_L
MOV SONUC_L, A

MOV A, SAYI1_H
ADDC A, SAYI2_H
MOV SONUC_M, A
```

```
MOV A, #00
ADDC A, #00
MOV SONUC_H, A
```

DA

Amaç: Decimal düzeltme işlemi
Etkilenen bayrak: CY

TOPLAMA İŞLEM KOMUTUNUN ARKASINDAN HEMEN ÇALIŞTIRILIR
IF (YARI ELDE BAYRAĞI = 1) VEYA (SONUC A-F ARASINDA) THEN
SONUÇ = SONUÇ + 6

BCD kodlanmış iki sayının toplama işlem sonucu

```
        23H         R0
        68H         R1
+   ----------
        8B
        DA A
        91H         R2
```

```
MOV A, R0
ADD A, R1
DAA
MOV R2, A
```

SUBB – MUL – DIV – INC -DEC

SUBB A, <byte>
Amaç: Eldeli çıkarma işlemi
Etkilenen bayrak: CY, OV

SUBB A, Rn	(A) ←(A)- (Rn)-C
SUBB A, direct	(A) ←(A)- (dır)-C
SUBB A, @Ri	(A) ←(A)- ((Ri))-C
SUBB A, #data	(A) ← (A)- #data-C

MUL AB
Amaç: ÇARPMA İŞLEMİ
Etkilenen bayrak: CY, OV
A * B → B A

DIV AB
Amaç: BÖLME İŞLEMİ
Etkilenen bayrak: CY, OV
A / B ==> (TAM→ A) (KALAN→ B)

INC byte
Amaç: +1 İŞLEMİ
Etkilenen bayrak: YOK

INC A	(A) ← (A) +1
INC Rn	(Rn) ← (Rn) +1
INC direct	(dir) ← (dir) +1
INC @Ri	((Ri)) ← ((Rn)) +1

DEC byte
Amaç: -1 İŞLEMİ
Etkilenen bayrak: YOK

DEC A	(A) ← (A) -1
DEC Rn	(Rn) ← (Rn) -1
DEC direct	(dir) ← (dir) -1
DEC @Ri	((Ri)) ← ((Rn)) -1

LOJIK İŞLEM KOMUTLARI

- ANL
- ORL
- XRL
- CLR
- SET
- CPL
- RL
- RR
- RLC
- RRC

ANL

ANL A, <byte>
Amaç: LOGIC AND İŞLEMİ
Etkilenen bayrak: YOK

```
10011110    A
01111011    R0

00011010    ANL A, R0
```

ANL A, Rn	(A) ←(A) ^ (Rn)
ANL A, direct	(A) ←(A) ^ (dır)
ANL A, @Ri	(A) ←(A) ^ ((Ri))
ANL A, #data	(A) ← (A) ^ #data
ANL direct, A	(dir) ← (dir) ^ A
ANL direct, #data	(dir) ← (dir) ^ #data

ANL C, bit **ANL C, /bit**
Amaç: Bit değişkenleri için AND İŞLEMİ
Etkilenen bayrak: CY

F (x, q) = x*q sonucunu bulunuz. x=p1.0 q=p1.1 F=p1.2 olsun

```
MOV C, P1.0
ANL  C, P1.1
MOV P1.2, C
```

ORL

ORL A, <byte>
Amaç: LOGIC OR İŞLEMİ
Etkilenen bayrak: YOK

```
00011010    A
01001011    R0

01011011    ORL A, R0
```

ORL A, Rn	(A) ←(A) v (Rn)
ORL A, direct	(A) ←(A) v (dir)
ORL A, @Ri	(A) ←(A) v ((Ri))
ORL A, #data	(A) ← (A) v #data
ORL direct, A	(dir) ← (dir) v A
ORL direct, #data	(dir) ← (dir) v #data

ORL C, bit — **ORL C, /bit**
Amaç: Bit değişkenleri için OR İŞLEMİ
Etkilenen bayrak: CY

F (x, q, y) = x*(q +y) sonucunu bulunuz. x=p1.0 q=p1.1 y=P1.2 F=p1.3 olsun

```
MOV C, P1.2
ORL  C, P1.1
ANL C,  P1.0
MOV P1.3, C
```

XRL – CLR – SET - CPL

XRL A, <byte>
Amaç: Exclusive OR İŞLEMİ
Etkilenen bayrak: YOK

```
00011010 ——→ A
01001011 ——→ R0

01010001 ←—— XRL A, R0
```

XRL A, Rn	(A) ←(A) EOR (Rn)
XRL A, direct	(A) ←(A) EOR (dir)
XRL A, @Ri	(A) ←(A) EOR ((Ri))
XRL A, #data	(A) ← (A) EOR #data
XRL direct, A	(dir) ← (dir) EOR A
XRL direct, #data	(dir) ← (dir) EOR #data

CLEAR

CLR A
Amaç: A ←0
Etkilenen bayrak: YOK

CLR BIT
Amaç: BİT ←0
Etkilenen bayrak: YOK

CLR C
Amaç: C ←0
Etkilenen bayrak: YOK

SET

SETB BIT
Amaç: BİT ←1
Etkilenen bayrak: YOK

SETB C
Amaç: C ←1
Etkilenen bayrak: YOK

COMPLEMENT

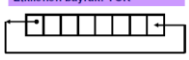
CPL A
Amaç: A ←A'
Etkilenen bayrak: YOK

CPL BIT
Amaç: BİT ←BIT'
Etkilenen bayrak: YOK

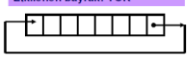
CPL C
Amaç: C ←C'
Etkilenen bayrak: YOK

ÖTELEME KOMUTLARI

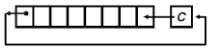
RL A
Amaç: Rotate Acc left
Etkilenen bayrak: YOK



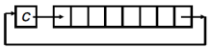
RR A
Amaç: Rotate Acc Right
Etkilenen bayrak: YOK



RLC A
Amaç: Rotate Acc left through Carry
Etkilenen bayrak: CY



RRC A
Amaç: Rotate Acc Right through Carry
Etkilenen bayrak: CY



PROGRAM YÖNLENDİRME KOMUTLARI

- LCALL
- ACALL
- SMP
- AJMP
- LJMP
- RET

YÖNLENDİRME

ŞARTSIZ YÖNLENDİRME

LCALL Adres	LJMP Adres
ACALL Adres	AJMP Adres
RET	SJMP Adres

DURUMA BAĞLI YÖNLENDİRME KOMUTLARI

JZ <rel_adres>
IF A=0 THEN PC=PC + rel_ad.

JNZ <rel_adres>
IF A<>0 THEN PC=PC + rel_ad.

10H, 11H ADRESLERİNDEKİ SAYILAR İÇİN
(10H) = (11H) ??

```
      CLR C
      MOV A, 10H
      SUBB A, 11H
      JZ KARAR
      ..........
      ..........
KARAR:
      ..........
      ..........
```

DALLANMA

JC <rel_adres>
IF CY=1 THEN PC=PC + rel_ad.

JNC <rel_adres>
IF CY=0 THEN PC=PC + rel_ad.

JB BIT, <rel_adres>
IF BIT=1 THEN PC=PC + rel_ad.

JNB BIT, <rel_adres>
IF BIT=0 THEN PC=PC + rel_ad.

JBC BIT, <rel_adres>
IF BIT=1 THEN BIT=0 PC=PC + rel_ad.

CJNE <des_byte>, <src_byte>, <rel_adres>
IF *des_byte* <> *src_byte* THEN PC=PC + rel_ad.
Etkilenen bayrak: CY

CJNE A, direct, rel
CJNE A, #data, rel
CJNE Rn, #data, rel
CJNE @Ri, #data, rel

DJNZ <byte>, <rel_adres>
Byte=byte-1
IF *byte* <> 0 THEN PC=PC + rel_ad.
Etkilenen bayrak: CY

DJNZ Rn, rel
DJNZ direct, rel

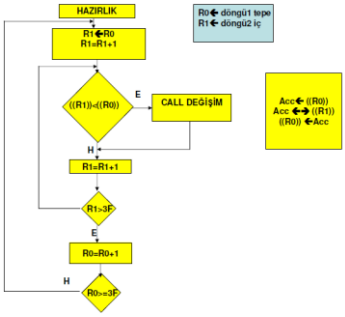
YORUM

- Komutlar kapladığı hafıza açısından değerlendirilmelidir.
  - 1 Byte, 2 Byte, 3 Byte
- Komutlar işlem süreleri açısından değerlendirilmelidir.
  - 1 MC, 2 MC, 4 MC
- Çalışma sonrasında durum saklayıcıların etkilenmesine göre değerlendirilmelidir.
- Adresleme yetenekleri açısından değerlendirilmelidir.
- İşlevleri açısından değerlendirilmelidir.

ÖRNEK

SORU: 30H - 3FH DAHİLİ HAFIZA ALANI İÇERİSİNDE BULUNAN 8 BİTLİK İŞARETSİZ SAYILARI KÜÇÜKTEN BÜYÜĞE DOĞRU SIRALAYAN PROGRAMIN
A. AKIŞ DİYAGRAMINI ÇİZİNİZ.
B. PROGRAMI YAZINIZ.

AKIŞ DİYAGRAMI



KOD

```
      MOV R0, #30H
;
DONGU1:
          MOV R1, 00
          INC R1
;
DONGU2:
          CLR C
          MOV A, @R1
          SUBB A, @R0
          JNC DALLAN
          LCALL DEGISIM
;
DALLAN:
          INC R1
          CJNE R1, #40H, DONGU2
;
          INC R0
          CJNE R0, #3FH, DONGU1

LOOP:
          SJMP LOOP
```

```
DEGISIM:
          MOV A, @R0
          XCH A, @R1
          MOV @R0, A
          RET
```