E-KİTAP

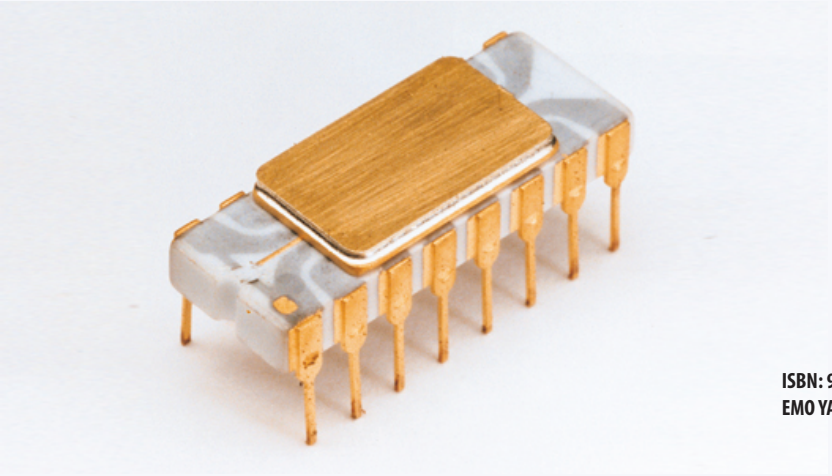
.pdf

ePUB

Mikroişlemcilere Giriş

Assembler ile Yazılım ve Arayüz

Mehmet Bodur



ISBN: 978-605-01-0635-2
EMO YAYIN NO: EK/2014/586

TMMOB
Elektrik Mühendisleri Odası
1954

Eylül 2014
Ankara

TMMOB
ELEKTRİK MÜHENDİSLERİ ODASI
1954

Mikroişlemcilere Giriş
Assembler ile Yazılım ve Arayüz

Mehmet Bodur

ISBN: 978-605-01-0635-2
EMO YAYIN NO: EK/2014/586

TMMOB ELEKTRİK MÜHENDİSLERİ ODASI
Ihlamur Sokak No: 10 Kat: 2 • 06420, Kızılay-Ankara
Tel: (0.312) 425 32 72-73 • Faks: (0.312) 417 38 18
e-posta: emo@emo.org.tr http://www.emo.org.tr

Bodur, Mehmet
Mikroişlemcilere Giriş: Assemler ile Yazılım ve Arayüz
EMO-1.bs--Ankara MEO Yayınları, 2014, 212 s.; 24 cm (EK/2014/586)
ISBN: 978-605-01-0635-2

Dizgi ve Tasarım
TMMOB ELEKTRİK MÜHENDİSLERİ ODASI

Mikroişlemcilere Giriş: Assembler ile Yazılım ve Arayüz

Mehmet Bodur

Eylül-2014

TMMOB ELEKTRİK MÜHENDİSLERİ ODASI

İçindekiler

Şekil Listesi

Yazarın Önsözü

Mikroişlemci kavramı 70'li yıllardan başlayarak mühendislik dünyamıza girdi. Önceleri ancak haberleşme, uzay ve havacılık gibi alanlarda kullanım bulan bu kavram hızla bilgisayar donanımı, endüstriyel kontrol sistemleri gibi alanlarda geniş uygulama alanı buldu. İşlemci üretim teknolojisinin gelişmesi ve ucuzlamasıyla günümüzde mikroişlemci kullanılmayan alan kalmadı.

Elinizdeki kitap DAÜ Bilgisayar Mühendisliği Türkçe Programına yönelik olarak, bilgisayar mühendisi adaylarına mikro işlemcilerin temel yapısı, devreleri, ve çevirici diliyle programlanmasını vermek üzere yazıldı.

Kitaptaki konuların takip edilebilmesi için öğrenci temel mantık tasarımının yanısıra ardışık sayısal devre analizini, ve yazmaç kavramını biliyor olmalıdır. Kitap kısa bir genel giriş bölümünün ardından doğrudan çevirim dili programlamaya giriyor ve mikroişlemci devrelerindeki bellek sistemi ile giriş çıkış sistemi, ve bunlara bağlı analog ya da sayısal dönüştürücü, zamanlama, ya da haberleşme donanımı gibi donanıma yönelik kavramlar için gerekli yazılım tabanını oluşturmayı amaçlıyor. Bu kitap akademik amaçla dağıtılan pdfLaTeX ile dizildi ve içindeki şekillerin büyük bölümü TiKz ortamında çizildi.

Son olarak kitabın yazılışında gerekli zamanı ve enerjiyi ayırabilmemi sağlayan, çeşitli konularda fikirleriyle destek olan sevgili ailem ve meslek arkadaşlarıma teşekkür eder kitabın okuyuculara yararlı olmasını dilerim.

Dr. Mehmet Bodur
Eylül 2012

Bölüm 1

Ön Bilgilenme

1.1 Sayı sistemleri

On parmağa sahip olduğumuz, ve beynimizin bir kümede bulunan on civarındaki nesneyi bir bakışta sayma kapasitesi bulunduğu için çok büyük sayıları bile ondalık sayı sisteminde ifade etmek bize çok kolay gelir. Doğduğumuzdan bu yana ondalık sayı sisteminde işlem yapmaya eğitim aldığımızdan ondalık sayı sistemini doğal sayı sistemimiz olarak kabul ederiz.

Bir anda ancak iki durumdan birinde kararlı olabilen ikidurumlu[1] devrelerden oluşturulan bilgisayarların doğal yapısı ise ikili[2] sayı sistemine uygundur.

Bilgisayar dünyasının anlayabilmemiz için ikili sayıları, ve onluk sayılarla ikili sayılar arasında dönüştürme yapmayı bilmeliyiz. İkili sayılarda aritmetik yapabilmek için eksi sayıları nasıl ifade edeceğimizi ve aritmetik işlemin sonucunun ne zaman bozulabileceğini de öğrenmeliyiz.

1.1.1 Onlu-İkili Dönüşüm

İkili sayı sisteminde her basamak ya sıfır 0_2 ya da bir 1_2 olabilir. Dolayısıyla birler basamağı sadece sıfırı ya da biri temsil edebilir. İki yazmak için birler basamağının soluna ikiler basamağı olarak adlandırdığımız yeni bir basamak eklenir. Böylece iki basamağı kullanarak
00_2 (=0), 01_2 (=1), 10_2 (=2) ve 11_2(=2+1=3)
yazabiliriz. Üçten sonraki sayıyı yazmak için bu iki basamağın da soluna değeri 4 olan yeni bir basamak kullanmamız gerekir. Sayılar giderek büyüdüğünde 8 'ler 16 'lar 32 ler ... basamakları da gerekir. Kullandığımız ikili basamaklara kısaca *bit*[3] denir. Örneğin birler basamağı *bit-0*, ikiler basamağı *bit-1*, dörtler basamağı *bit-2* olarak adlandırılır. Böylece 2^n ler basamağı *bit-*n ile gösterilmiş olur.

[1] flip-flop
[2] binary
[3] **Bi**nary Digi**t**

İkili sayıyı onlu sayıya dönüştürmek için her basamağın değerini rakamıyla çarpıp toplamlarını alırız.

Örnek 1.1.

00101101_2 sayısını onlu sistemde yazalım.

Çözüm Sıfırlı basamakların değerleriyle çarpımı sıfır vereceğinden en sağdan başlayarak yalnızca birli basamakların değerlerini birle çarpıp toplamını alacağız.

1+4+8+32=45

Onlu sayıyı ikiliye dönüştürmenin iki kolay yolu vardır. Birincisi ardışık olarak onlu sayı içindeki en yüksek ikili basamak değerini bulur ve bunu onlu sayıdan çıkarıp ikili sayıya ekleriz. Sonunda onlu sayı sıfırlandığında ikili karşılığını elde ederiz.

Örnek 1.2.

25_{10} sayısını ikili sistemde yazalım.

Çözüm

25 ten küçük ikili basamak değerleri $16=10000_2$, $8=1000_2$, $4=100_2$, $2=10_2$, ve $1=1_2$ dir.

25 -**16**=9 ; $16=10000_2$
9 - **8**=1 ; $8=1000_2$
1 - **1**=0 ; $1=1_2$
Toplam 25= 11001_2

$25= 16+8+1=10000_2+1000_2+1_2= 11001_2$

İkinci yöntemde sayıyı ardarda ikiye bölerken kalan birleri kullanırız.

Örnek 1.3.

25_2 sayısını ikili sistemde yazalım.

Çözüm

```
25/2 =  12, kalan 1 En sağ basamak (LSB)
12/2 =   6, kalan 0
 6/2 =   3, kalan 0
 3/2 =   1, kalan 1
 1/2 =   0, kalan 1 En sol basamak (MSB)
```

Sonuç $25_{10} =11001_2$

Kesirli sayıların kesirlerini dönüştürürken ardışık çarpımlar kullanılır.

Örnek 1.4.

0.625_{10} sayısını ikiliğe dönüştürelim.

Çözüm

Sayının kesirli bölümü ardışık $\times 2$ işlemleriyle ikili sayıya dönüşür. İşleme istenen sayıda basamak elde edilinceye kadar, veya sayı sıfır oluncaya kadar devam edilir.

$0.625 \times 2 = 1.25 \rightarrow .\mathbf{1}$
$0.25 \times 2 = 0.5 \rightarrow .1\mathbf{0}$
$0.5 \times 2 = 1.0 \rightarrow .10\mathbf{1}$
$0 \times 2 = 0 \rightarrow .101\mathbf{0}$
...

Sonuç: 0.625_{10} =0.1010....

Hem kesir hem de tamsayısı olan onluk sayının kesir ve tamsayısı ayrı ayrı dönüştürülür.

Örnek 1.5.

25.625 sayısını ikili sistemde yazalım

Çözüm:

$25.625_{10} = 25 + 0.625$
$11001_2 + 0.101_2 = 11001.101_2$

1.1.2 İşaretsiz ve işaretli sayılar

İkili sayılarla yalnızca sıfır ve pozitif sayıları yazmak istersek sayıyı yazmakta kullanacağımız bütün bitleri ikinin katları olarak artan basamak değerleriyle kullanırız. Bütün bitleri pozitif basamak değerleri için kullanarak yazdığımız sayı biçimine *işaretsiz*[4] *ikili sayı* denir. İşaretsiz sayı sisteminde $\{$bit-$(n-1)$, bit-$(n-2)$, ... , bit-1, bit-0 $\}$ olarak adlandırdığımız toplam n bit ile yazabilen en küçük sayı bütün bitlerin sıfır olmasıyla elde edilen sıfır sayısıdır. n-bitlik işaretsiz sayı sisteminde en büyük sayıyı bütün bitler bir olduğunda
$2^{n-1} + 2^{n-2} + ... + 2^1 + 2^0 = 2^n - 1$
olarak elde ederiz.

İkili sayı sisteminde eksi sayıları da gösterebilmek için en azından bir basamağın eksi değerli olması gerekir. En yaygın yöntem değeri

[4] unsigned

en yüksek olan en soldaki bitin eksi değerli *kullanıldığı işaretli*[5] *ikili sayı* sistemidir. Gene $\{$ bit-$(n-1)$, bit-$(n-2)$, ... , bit-1, bit-0$\}$ olarak adlandırdığımız toplam n bit kullanacak olursak, en soldaki bit-$(n-1)$ eksi değerli kullanılınca değeri -2^{n-1} olacaktır. Böylece yazabileceğimiz en küçük sayı yalnızca eksi değerli biti bir yapınca oluşan -2^{n-1} sayısı olur. En büyük sayı, eksi değerli bitin sıfır, diğer bütün bitlerin bir olduğu $2^{n-2}+...+2^1+2^0=2^{n-1}-1$ sayısıdır.

İşaretli sayılardaki eksi değerli bite *işaret biti* [6] denir. İşaret biti bir ise sayı sıfırdan küçük değerlidir. İşaret biti sıfır ise sayı sıfır ya da sıfırdan büyük demektir. İşaretli sayılarla toplama veya çıkarma işlemlerinde işaret bitinden daha değerli bir bit olamayacağından son eldenin değeri yoktur. Diğer bitlerden gelen elde yüzünden işaret biti bozulup sayının işareti değişirse istenmeyen bu duruma *taşma*[7] denir.

Örnek 1.6.

$N=-120$ sayısını sekiz bitle işaretli ikili sistemde yazalım.

Çözüm:

bitlerimizin değerleri şöyle olacaktır:

bit-7	bit-6	bit-5	bit-4	bit-3	bit-2	bit-1	bit-0
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-128	64	32	16	8	4	2	1

Sayıyı sıfırdan küçük yapan tek etmen bit-7'nin değerinin -128 olmasıdır. Sıfırdan küçük bir sayı yazmak için işaret biti olan bit-7 muhakkak bir olmalıdır. Diğer bitlerle yazacağımız sayıya p dersek

$-128+p=-120$ olacağından

$p=128-120=8=00001000_2$

olarak buluruz. Böylece sayının tümü $-128+8=-120$ olduğundan $-120=-128+8=10000000_2+00001000_2=10001000_2$ olarak yazılır.

Örnek 1.7.

$N=-6.75$ sayısını beş bit tamsayı, üç bit kesir kullanarak işaretli ikili sistemde yazalım.

Çözüm:

bitlerimizin basamak değerleri şöyle olacaktır:

bit-4	bit-3	bit-2	bit-1	bit-0	bit(-1)	bit(-2)	bit(-3)
-2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
-16	8	4	2	1	1/2	1/4	1/8

[5] signed
[6] sign bit
[7] overflow

Sayı sıfırdan küçük olduğundan işaret biti olan -16 değerli bit-4 bir olmalı. Sayının diğer bitlerine karar vermek üzere sayıya 16 ekleyelim. $p = 16 + N = 9.25$.
Demek ki geri kalan bitlerle 9.25 yazdığımızda işaretli sayı
$-16+9.25=6.75$
değerini taşıyacaktır.
$(N)_2$=11001.010_2

1.1.3 İkili tümleyen ve işaret değiştirme

İşaret biti dışındaki bütün bitlerin basamak değeri işaretsiz basamak değerine eşit olduğundan sıfırdan büyük sayılar için işaretli sayı ile işaretsiz sayının ikili kodları arasında fark olmaz. Ancak sıfırdan küçük sayılar işaretsiz sayı sisteminde gösterilemez. n bitlik işaretli ikili sayı sisteminde verilmiş $-2^{n-1} \leq P < 2^{n1}$ aralığındaki bir P sayısının toplama işlemine göre tersi olan $-P$ sayısını bulmak için $-P = 2^n - P$ işlemini kullanırız.

$$-P = 2^n - P$$

işlemine *ikiye tümlemek*[8], ya da *ikiye tümler bulmak* diyoruz. $2^n - P$ ye de P nin *ikiye tümleyeni*[9] ya da *ikiye tümleri* diyoruz.

Donanım açısından verilen tanımı kullanarak ikiye tümlemek için ikili çıkarma devresi gerekir. Çoğu sistemde malzeme tasarrufu açısından sadece ikili toplayıcı devreler kullanılır. İkiye tümleme işlemi

$$-P = 2^n - P = (2^n - 1) - P + 1 = \overline{P} + 1$$

biçiminde bire tümlerin bir fazlası olarak yazılabilir.

$\overline{P} = (2^n - 1) - P$ işlemi bire tümlemek[10] olarak adlandırılır ve bütün bitlerin tersine çevrilmesi işlemidir.

Kağıt üzerindeki çözümlerde $-P$ yi bulmanın en kolay yöntemi tersine çevirip bir toplama işlemini kestirmeden yapmamızı sağlayan kopyalı tersleme yöntemidir. Bu yöntemde ikili sayının ikiye tümlerini bulmak için sağdaki ilk bire kadarki sıfırlardan oluşan bölümünü ve ilk biri kopyalar, diğer bitlerin tümünü tersine çeviririz.

Örnek 1.8.

4 bit kullanarak -6 yazalım.

Çözüm: $n = 4$ olduğundan ikili tümleme işlemi
$2^4 - 6 = 16 - 6 = 10$
sonucunu verir;

[8] two's complementing
[9] two's complement
[10] one's complementing

Sonuç: $-0110_2 = 1010_2$

En soldaki basamak olan bit-3 ün basamak değeri -8 dir ve sayıyı ondalık sisteme dönüştürürken

$1010 = 1 \times (-8) + 0 \times 4 + 1 \times 2 + 0 \times 1 = -6$

olarak hesaplarız.

Örnek 1.9.

$N=-26$ sayısını n =8 bit kullanarak yazmak istiyoruz. $26 = 00011010_2$ sayısını kolayca yazdık. Işaretli ikili sistemde -26 sayısını bulalım.

Çözüm:

Sayının en solundaki sıfır ve biri kopyalayacağız. Geri kalanları tersine çevireceğiz.

$-26 = \overline{0}\,\overline{0}\,\overline{0}\,\overline{1}\,\overline{1}\,\overline{0}\,\mathbf{1}\,\mathbf{0} = 11100110_2$

Elde ettiğimiz sayının sağlamasını yapalım.

$-128 + 64 + 32 + 4 + 2 = -26$

Örnek 1.10.

$N=-6.75$ sayısını beş bit tamsayı, üç bit kesir kullanarak en kolay yoldan işaretli ikili sistemde yazalım.

Çözüm:

Bu soruyu daha önce çözmüştük. Daha kolay yoldan çözmek için 6.75 yazıp ikili tümlerini bulabiliriz. Bitlerimizin değerleri şöyle olacaktır:

bit-4	bit-3	bit-2	bit-1	bit-0	bit(-1)	bit(-2)	bit(-3)
-2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
-16	8	4	2	1	1/2	1/4	1/8

$-N = 6.75 = 4 + 2 + 1/2 + 1/4 = (00110.110)_2$.

İkiye tümlemek için en sağdaki sıfırları ve ilk biri kopyalayıp diğer bitleri tersine çevirelim.

$N = -(-N) = -6.75 = \overline{0}\,\overline{0}\,\overline{1}\,\overline{1}\,\overline{0}\,.\,\overline{1}\,\mathbf{1}\,\mathbf{0} = 11001.010_2$

Önce tamsayıya çevirerek te yapabiliriz. Üç bit kesir kullanacağımıza göre sayıyı $2^3 = 8$ ile çarparak tamsayıya çevirip tamsayı olarak ikili karşılığını bulalım.

$-6.75 \cdot 8 = -54 = \texttt{-00110110}_2 = \overline{00110110}_2 = \texttt{11001010}_2.$

$-6.75 = -54/3 = \texttt{11001.010}_2.$

1.1.4 Eksileme ve Tersine çevirme

İkiye tümleme işleminin diğer adı *Eksileme*[11] dir. n basamaklı olan P sayısının n basamaklı ikiye tümleri $2^n - P$ dir.

n-bit ikili sayının tüm bitleri bir olduğunda işaretsiz değeri $2^n - 1$ ettiğinden $2^n - 1 - M$ işlemi M sayısının bitlerini tersine çevirme[12] işlemine denktir ve yazmaçtaki M nin tersine çevrilmesi de denen bu işlem $\overline{M}$ ile gösterilir.

Tersine çevirme aritmetiksel olarak $\overline{M} = 2^n - 1 - M$ ile ifade edilse de donanımda tersine çevirici devreleri ile gerçekleştirilir. Eksileme ya da ikili tümleme işlemi ise genellikle sayının bitlerini tersine çevirip bir toplayarak yapılır.

$-M = 2^n - M = 2^n - 1 - M + 1 = \overline{M} + 1$

Böylece işlemi bir tersine çevirici[13] ve bir toplayıcı[14] devre ile gerçekleştirmek mümkün olur.

1.1.5 İkiliden Onaltılığa dönüşüm

İkili sayı sistemindeki sayılar ve kodlar akılda kalmayacak kadar uzun 1 ve 0 zincirleridir. İşaretli ya da işaretsiz ikili sayıları ve bit dizileri biçimindeki diğer kodları kolay anlaşılır ve akılda kalır biçimde ifade edebilmek için 16 lı[15] sayı sistemi kullanılır. Onaltılı sayı sistemi kısaca hex olarak ta adlandırılır.

İkili sayıyı onaltılığa dönüştürmek için kesir noktasından başlayarak dörtlü basamak gruplarına ayrılması ve her grubun hex karşılığının bulunması. Grupları dörde tamamlamak için 0 eklemek gerekebilir. 9 dan büyük dörtüler

10=1010_2=A, 11=1011_2=B, 12=1100_2=C,
13=1101_2=D, 14=1110_2=E, 15=1111_2=F

ile gösterilir.

Örnek 1.11.

11001.101_2 sayısını onaltılık sistemde yazalım.

Çözüm: Grupları 4 bite tamamlamak için gerektiği kadar sıfır koyacağız

*000*11001.101*0*$_2$ -> 19.A$_{16}$

[11] negation
[12] invert
[13] inverter
[14] adder
[15] hexadecimal

Eğer bu sayının 8-bit olması önem taşıyorsa (örneğin bit-4 işaret bitiyse) sayıya sıfır eklemeksizin onaltılığa çevirir kesir noktasının yerini ise /8 ile belirtiriz.

1100 1.101 = 1100 1101 /8 =$CD_{16}/8$

Örnek 1.12.

110010.111010_2 sayısını onaltılık sistemde yazalım.

Çözüm:

00110010.11101000_2 -> $32.E8_{16}$=32.E8H

Sayının onaltılık olduğunu bu örnekteki gibi sonuna eklenen bir H" ya da h" harfi ile de belirtebiliriz.

1.1.6 Onaltılıktan onluğa dönüşüm

Birler basamağı $16^0 = 1$ ile çarpılacağından aynen alınır. Solundaki basamak $16^1 = 16$, onun yanındaki 16^2 =16×16=256, bir sonraki basamak 16^3=256×16= 4096 ve dördüncü basamak 16^4 =65536 sayısıyla çarpılır. Kesir basamakları ise 1/16, 1/256, 1/4096 ... ile çarpılır.

Örnek 1.13.

110010.111010_2 sayısını onlu sayıya dönüştürelim.

Çözüm:

oo11 0010.1110 $1ooo_2$ -> $32.E8_{16}$

3×16 + 2 + 14/16 + 8/256 = 50.90625

1.2 İkili Sayılarla Aritmetik

İkili sayıların her bir basamağını bit olarak adlandırdık. Bitler birler basamağı sıfırdan başlatılarak bit-0, bit-1, ... biçiminde sayılır. bit-k basamağının değeri 2^k kadardır. Kesir basamaklarının negatif değerli adları basamağın değerine uygundur. Noktadan sonraki ilk kesir basamağının adı bit(-1), ağırlığı $2^{-1} = \frac{1}{2^1} = \frac{1}{2}$ olur.

İkilik sayının en soldaki bitinin basamak değeri en yüksektir ve bu bite en yüksek değerli bit[16] denir ve MSB ile gösterilir. Sayının en sağdaki biti ise en düşük değerli bit[17] olarak adlandırılır ve LSB ile gösterilir.

[16] Most significant bit (MSB)
[17] Least significant bit (LSB)

Örnek 1.14.

010.11_2 sayısında her basamağın adını ve değerini yazalım.

Çözüm:

sayı:	0	1	0	.	1	1
bitleri:	bit-2	bit-1	bit-0	.	bit-(-1)	bit-(-2)
değeri:	4	2	1		$1/2$	$1/4$
özelliği	MSB					LSB

Bit-2 MSB, bit(-2) LSB bitleridir. Noktaya kadarki bölüm tamsayı, noktadan sonrası kesirli sayı bölümüdür.

1.2.1 Toplama

Toplama yaparken elde[18] biti bir sonraki basamağa ilerler.

`0+0 = 0; 0+1 = 1; 1+0 = 1; 1+1 =10.`

Elde bitini de işleme koyabilmek için her bitin toplama devresinin üç girişli ve iki çıkışlı olması gerekir.

```
0+0+0 =00; 0+0+1 =01; 0+1+0 =01; 0+1+1 =10;
1+0+0 =01; 1+0+1 =10; 1+1+0 =10; 1+1+1 =11;
```

Örnek 1.15.

$01001011 + 11000001$

Çözüm:

```
 1 1     11   <- elde bitleri
   01001011
+  11000001
-----------
 1 00001100 <- sonuç (son elde 1 oldu)
```

1.2.2 Çıkarma

Sayısal devreler açısından çıkarma işlemi için yeni bir devre kullanmak yerine çıkarılacak sayının ikili tümleyeniyle toplama işlemi yapmak daha az donanım gerektirir.

$N - M = N + (-M)$

Örnek 1.16.

n=4 bit yazmaçlarla $N = 6$, $M = 4$ ile $N - M$ işlemini yapalım.

[18] carry

Çözüm:
N =0110, M =0100, $N + (-M)$ =?
$-M$=1100; N+($-M$)= 0110+1100= 1 0010
n=4 bit olduğundan elde bitinin anlamı yoktur.
Sonuç $N - M = 0010_2$ olur.

1.2.3 İşaretli uzatma

İşaretli sayılarla toplama yaparken iki sayının basamak sayısı birbirine eşit olmazsa işaret bitleri aynı hizaya gelmez ve sonuç yanlış çıkar. Bu nedenle işaretli sayıların kısa olanını işaretli uzatma[19] işlemiyle uzun sayı kadar uzatırız.

İkili işaretli tamsayıların işaretli uzatma işlemi en soldaki işaret bitinin gereken miktardaki kopyasını sayının soluna ekleyerek gerçekleştirilir.

Örnek 1.17.

B=101_2 (=-3) işaretli sayısını 3 bitten 5 bite uzatalım.

Çözüm: B nin işaret biti olan bit-2 =1 olduğuna göre B nin soluna iki tane daha 1 ekleyeceğiz. Uzatılmış sayıya $\dot{B}$ dersek:
$\dot{B}$= 11101_2.
İki sayıyı da negatifleyerek sağlamasını yapalım.
$-B$=-101_2=011_2=3; $-\dot{B}$=-11101_2= 00011_2=3.
Uzatma işlemi sayının değerini değiştirmedi.

Örnek 1.18.

A=01001_2 ile B=101_2 işaretli sayılarını toplayalım.

Çözüm: A ile B eşit uzunlukta değiller. A 5 bit, B ise 3 bit olduğundan toplamayı B yi 2 bit işaretli uzatarak toplamalıyız.

```
  01001
+ 11101
-------
1 00110
```

Sonuç olarak işaretli sayıları toplarken elde bitinin anlamı ve değeri yoktur. Örneğin 9-3 işleminin sonucunda elde bitini saymadan 6 elde ettik.

[19] sign extension

1.2.4 İşaretli Toplamanın Taşması

Elde bitleri işaret bitini bozarsa islemin sonucu ters işaretli olacağından yanlış çıkar. Bu olaya taşma denir.

Örnek 1.19.

6+3 işlemi 4-bitli işaretli sayılarda yapılırsa sonuç doğru çıkar mı?

Çözüm: 4-bitli işaretli sayılarda işaret biti olan bit3 kullanmadan yazılabilecek en büyük sayı 0111 =7 dir. 6+3=9>7 olduğundan sonucu 4-bit ile yazmak mümkün olmaz. Sonuç olarak bulduğumuz 1001_2 negatiftir. İki pozitif sayının toplamı işaret bitine taşınca sonucu negatife döndürerek bozdu.

İşaret bitine giren elde biti çıkan elde bitiyle dengelenmezse toplamada taşma olduğunu anlarız. Taşma ancak iki negatif sayı ya da iki pozitif sayı toplandığında oluşur. Bir negatif ile bir pozitif sayının toplamı asla taşamaz.

Örnek 1.20.

8-bit sayılarla $(-96)+(-106)$ işleminde taşma var mı?

Çözüm:

```
 10100000
+ 10010110
-----------
=1 00110110
```

Sayıların ikisi de negatif ama sonuç pozitif. Demek ki taşma olmuş.

1.2.5 Sayılar ve Yazıların Kodlanması

Bilgisayarlarda veri genellikle 8-bit ve katları uzunluklarda saklanır. 8-bit bit bayt olarak adlandırılır ve B ile gösterilir. Ondalık sayıları ve yazıları baytlara kodlamak için çeşitli yöntemler kullanılır.

Ondalık sayılar genellikle ya sıkışık-BCD[20] veya gevşek-BCD[21] olarak kodlanır. İkili işaretli sayılar genellikle 8, 16 veya 32 bitlik genişliklerde kullanılır. Yazıları kodlamak için ASCII kodlama en yaygın yöntemdir. Bunların yanısıra haberleşme ve duyaçlardan bilgi aktarımında kullanılan pek çok kodlama yöntemi vardır.

[20] packed-BCD
[21] unpacked-BCD

Ondalık sayıların BCD kodlanması

Sıkışık-BCD (pBCD) kodlamada her bayta iki ondalık basamak sığdırılır. Bu amaçla bayt 4-bitlik iki yarıma bölünür ve her yarımda bir ondalık basamak saklanır.

Örnek 1.21.

Ondalı 1208 sayısını sıkışık ve gevşek BCD kodunda yazıp ikili sistemde gösterelim.

Çözüm

sıkışık BCD gösterimi: $1208_{10} => 1208_{pBCD} = 1208_{16}$
$= 0001\,0010\;\;0000\,1000_2$

Her ondalık basamak dört ikilik basamaktan oluşan bir lokmaya[22] yazılmaktadır.

Gevşek-BCD (uBCD) kodlamada ise her ondalık basamak bir bayt alana yazılır.

$1208_{10} => 1208_{uBCD} = 01020008_{16}$
$= 00000001\,00000010\,00000000\,00001000_2$

her ondalık basamak bir bayt alan kaplamaktadır. Bu uzun sayı bellekte 08h 00h 02h 01h sırasıyla duruyorsa *küçük uçtan yerleştirilmiş*[23], tam tersine, 01h 02h 00h 08h sırasıyla durursa *büyük uçtan*[24]yerleştirilmiş olur.

Yazı karakterlerinin kodlanması

Yazı karakterlerini kodlamak için çeşitli yöntemler arasında ASCII kodlama en yaygınıdır.

Tablo 1.1: Hex ASCII kodlar tablosu

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
2-		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3-	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4-	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5-	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6-	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7-	p	q	r	s	t	u	v	w	x	y	z	{	|	}	→	←

[22] nibble
[23] little endian
[24] big endian

ASCII kodlarda büyük harf karakterlerle küçük harfler arasındaki tek fark bit-5 tir. Tüm büyük harflerin kodlarındaki bit-5 sıfırdır, küçük harflerin bit-5 leri ise birdir. Büyük ve küçük harfleri birbirine dönüştürmekte bu özellikten yararlanılır.

ASCII kodlanmış rakamların sol dört biti herzaman '0011'=3 tür. Tablo 1.2'da görüldüğü üzere ASCII kodların arasında kontrol karakteri olarak adlandırılan 07h (zil), 08h (geri al), 09h (sekme[25]), 0Ah (yeni satır), 0Ch (sayfabaşı), 0Dh (satırbaşı) gibi kodlar da bulunur. Bunlara ek olarak 7Fh (geri) kodu son karakteri silerek o karakterin bulunduğu yere gider.

Tablo 1.2: ASCII kontrol karakterleri

Kod	Adı	Görevi	Kod	Adı	Görevi
0	NUL	boş	16	DLE	veri bağlantısından çık
1	SOH	başlık başlangıcı	17	DC1	aygıt denetimi 1
2	STX	metin başlangıcı	18	DC2	aygıt denetimi 2
3	ETX	metin sonu	19	DC3	aygıt denetimi 3
4	EOT	aktarım sonu	20	DC4	aygıt denetimi 4
5	ENQ	sorgu	21	NAK	olumsuz bildirim
6	ACK	bildirim	22	SYN	zaman uyumlu boşta kalma
7	BEL	zil	23	ETB	aktarım bloğu sonu
8	BS	geri al	24	CAN	iptal
9	HT	yatay sekme	25	EM	ortam sonu
10	LF	satır besleme/yeni satır	26	SUB	değiştir
11	VT	dikey sekme	27	ESC	çık
12	FF	form besleme/yeni sayfa	28	FS	dosya ayırıcısı
13	CR	satır başı	29	GS	grup ayırıcısı
14	SO	dışarı kaydır	30	RS	kayıt ayırıcısı
15	SI	içeri kaydır	31	US	birim ayırıcısı

Örnek 1.22.

Aşağıdaki harflerin hexadecimal ASCII kodunu bulun.
'4' ? , 'a' ? , 'A' ? , '[' ? , satırbaşı ?, yeni satır ?, gerisil ?, zil ?;

Çözüm:

Tabloda harfi bulup sol dörtbiti soldan, sağ dörtbiti yukarıdan okuyoruz.
'4' $=34_{16}$. 'a'$=61_{16}$. 'A'$=41_{16}$. '[' $=5B_{16}$. gerisil $=7F_{16}$.

Kontrol karakterlerini ise 16'lık sayıya çevirmeliyiz.
yenisatır$=10=0A_{16}$. satırbaşı$=13=0D_{16}$. zil$=7=07_{16}$.

[25] tab

Yazı içindeki sayılar ASCII kodlanır.

Sayıları doğrudan ikili biçimiyle monitörde görmek veya yazıcıdan yazdırmak mümkün değildir. Monitör veya yazıcıya yollanacak sayılar ASCII text biçimine kodlanarak yollanır.

Örnek 1.23.

50 sayısını ondalık ve ikilik ASCII dizine dönüştürelim.

Çözüm:

Ondalık dizin 50= 35h 30h

İkilik dizin 110010= 31h 31h 30h 30h 31h 30h

Gray kodlama

Gray kodun özelliği ardışık sayılar arasında yalnızca bir bir değişim olmasıdır. Böylece ardışık sayılar arasında temiz bir geçiş olur. Örneğin ikili sayılardaki 1 den 2 ye geçerkenki gibi iki bit değişimi olsa, çok kısa bir süre için bile olsa bitler birer birer değişeceğinden ya 1 => 3 => 2 ya da 1=> 0=> 2 geçişleri gözlenecektir. Oysa 4-bit Gray kodunda 1=0001 ve 2=0011 ile kodlandığından böyle beklenmedik sayı değişimi gözlenmez.

0 = 0000, 1 = 0001, 2 = 0011, 3 =0010,
4 = 0110, 5 = 0100, 6 =0101, 7 =0111,
8 =1111, 9 =1110, 10 =1100, 11 =1101,
12 =1001, 13 =1011, 14 =1010, 15 =1000,
Gray kodlama genellikle pozisyon sensörlerinde kullanılır.

1.3 Mikroişlemcinin İçindekiler

8086 mikroişlemcisinin içi bir bilgisayarın işlemcisi gibi düzenlenmiştir. Bilgisayarlarda özel ve genel amaçlı yazmaçlar, aritmetik mantık birimi[26] ve denetim birimiyle, denetim biriminin yolladığı sinyallerle değişebilen veri yolları bulunur.

Veriler *bit*, *dörtbit*[27], *bayt*, *sözcük* ve sözcüğün katları olarak düzenlenirler ve işlenip belleğe aktarılıncaya dek merkezi işlem birimi[28] (CPU) içinde yazmaçlarda saklanırlar.

[26] Arithmetic Logic Unit (ALU)
[27] nibble
[28] Central Processing Unit (CPU)

Bilgisayarların iç düzeni CPU (merkezi işlem birimi) içinde yer alan bir komut yazmacı[29], genel amaçlı yazmaçlar, aritmetik-mantık-işlem-birimi (ALU), iç-veri/adres yolları ve denetim biriminden[30] (CU) oluşur.

CPU'ya bellek ve giriş/çıkış araçları gibi dışarıdan bağlanan birimlere *Veri yolu*, *Adres-yolu*, ve *Denetim-yolu* olarak adlandırılan sinyal giriş/çıkış birimleri üzerinden bağlanır.

1.3.1 Aritmetik Mantık Birimi

Yazmaçlardaki bit grupları üzerinde çeşitli aritmetik ve mantık işlemleri gerçekleştirmek üzere geliştirilmiş belleksiz [31] bir birimdir. Başlıca işlemler arasında ve, veya, yada, çevirme ve kaydırma, toplama çıkarma çarpma ve bölme, ve çeşitli bit işlemleri sayılabilir.

1.3.2 Ana Bellek

Bilgisayarlarda geçici olarak veri saklamak için her biri bir bit saklayabilen çok fazla sayıda ikidurumlu[32] devreleri bulunur. Veri ve işlenecek program kodunu saklayan ikidurumlular bayt adı verilen 8 lik hücreler halinde düzenlenerek adreslenirler. Bilgisayarlardaki merkez işlem biriminin doğrudan adresleyebildiği veri depolarına ana bellek denir.

Ana bellekteki veri depoları veri yolu genişliğine ve daha büyük veri tiplerine uygun gruplar halinde düzenlenir. Örneğin 8086 veri yolu 16 bitlik kelimeleri iletebileceği için ana bellek birer baytlık iki bank yapısında düzenlenmiştir. Ancak ana bellekte adresleme her bayta ayrı ayrı erişilmesine olanak sağlayacak biçimdedir. Bellekteki hücrelerin dizilme sırasının erişime bir etkisi yoktur (en baştaki hücreye erişmekle en sondaki hücreye erişme işlemi birbirinin aynıdır). Bu yüzden ana belleğe Rastgele Erişimli Bellek[33] (RAM) denir.

Bilgisayarın ana belleğindeki hücrelerin toplam sayısı, bilgisayarın bellek kapasitesini gösterir (örneğin, 1 MB$=2^{20}$ bayt, 1 GB$=2^{30}$ bayt)

[29] Instruction Register
[30] Control unit (CU)
[31] combinatorial
[32] flip-flop
[33] Random Access Memory (RAM)

1.3.3 Makina Dilleri

Merkezi işlem biriminde (CPU) çalıştırılabilecek her komutun bir sayısal kodu vardır. Makina dilleri 0 ve 1'lerden oluşan en alt seviyeli dillerdir. Bu dilde, makina komutları bir sayısal değer olarak kodlanır.

Makina dili işlemci ve donanıma bağlı olarak değişmektedir. Her işlemci mimarisinin kendine özgü bir makina dili vardır. Günümüzde programlar yüksek seviye dil derleyicileri ve çeviriciler[34] ile yazılmakta, doğrudan makina dili düzeyinde programlama yapılmamaktadır. Makina dili yalnızca bilgisayarın komutlarının tasarımı aşamasında önem kazanmaktadır.

1.3.4 Çevirici Diller

Çevirici diller makina dili komutlarıyla bire bir denk düşen komutçuklarla kurulmuş dillerdir. İşlemcinin çalıştırabileceği her makina komutu için bir çevirici komutu tanımlanır. Her işlemci platformu için farklı çevirici dilleri mevcuttur. Örneğin, bir Motorola işlemci ile Intel işlemcinin çevirici dilleri birbirinden farklıdır.

Örnek olarak 8086 çevirici dilinde bir kod parçası verelim.

```
1  ; İşletim sistemi gerektirmeyen merhaba
2    MOV AH,03H
3    INT 10H
4    MOV AL,01H
5    MOV BH,00H
6    MOV BL,01001111B
7    MOV CX,MESAJ_SON-MESAJ_BAS
8    PUSH CS
9    POP ES      ; ES<--CS
10   MOV BP, MESAJ_BAS
11   MOV AH,13H ; Ekrana yazdıran
12   INT 10H    ; BIOS fonksiyonu
13   JMP SON
14 MESAJ_BAS DB "Merhaba Dünya!"
15 MESAJ_SON DB 00H
16 SON:
17   RET
```

Eğer bir işletim sistemindeki servisleri çağırabiliyorsak aynı program daha kısa yazılabilir.

```
1    MOV AX,CS
2    MOV DS,AX
```

[34] assembler

```
3    MOV AH,09H
4    MOV  DX,OFFSET MESAJ
5    INT 21H
6    MOV AH,4Ch
7    INT 21H
8 MESAJ:  DB "Merhaba Dünya!", 13,10, '$'
```

1.3.5 Yüksek Düzey Programlama Dilleri

Büyük programları çevirici dillerle gerçekleştirmek ve hatalarından arındırmak C, C++, Java, C#, ADA, Fortran, Pascal, COBOL, BASIC gibi daha üst düzey programlama dillerine göre kat kat daha zordur. Çevirici dili bir işlemcinin temel dili olmasına karşın karmaşık programların yazılmasında yüksek düzey programlama dilleri kullanılır.

1.3.6 Bellek yazmaçlarının adreslenmesi

Bir bilgisayardaki adres ve veri yollarıyla erişilen bütün iç ve dış yazmaçlarının birbirleriyle karışmayacak bir adresleme sisteminde birer adresi vardır. Bellek yazmaçları adreslerine göre sıralanır. Yazmaç adresi bir arttırılınca elde edilen yeni adres bir sonraki yazmacın adresidir.

1.3.7 Komut seti

Her CPU nun denetim biriminin çözümleyip yorumlayabileceği bir komut seti vardır. Komut setinde genellikle ALU işlemlerinin hangi iç yazmaç ve bellek yazmacında uygulanacağı belirtilir.

1.3.8 Program sayacı

Bilgisayar bir sonraki komutun adresini *Program Sayacı* [35] denilen özel amaçlı bir iç yazmaçta tutar. Denetim birimi, CPU nun komutları belli bir başlangıç adresinden[36] başlayarak sırayla yorumlamasını sağlar.

1.3.9 Saat çevrimi

CPU ancak bir saat çevrimi ile durum değiştirip bir sonraki duruma geçer. CPU nun bir komutu yorumlayabilmesi için birden çok saat çevrimi gerekebilir. Denetim birimi bütün işleri CPU saat döngüsüyle adım adım yapar. CPU bir komutu yorumladıktan sonra bir sonraki adreste

[35] Instruction Counter
[36] reset address

yer alan komutu yorumlamaya başlar. Ancak, programın içinde kullanılan *program akış denetimi komutları*[37] bu olağan yorumlama sırasını değiştirilebilir.

1.3.10 Program akışı ve bayrak yazmacı

Program akış denetim komutları, koşullu atlama ve dallanmaları bir önceki ALU işleminin sonucuna göre durumu saklayan bayrak bitlerinden yararlanarak gerçekleştirir. Örneğin elde bayrağı `CF`, toplama işleminde son basamaktan bit taşarsa 1, taşmazsa 0 yapılır. Bunun gibi sıfır bayrağı `ZF`, işaret bayrağı `SF`, taşma bayrağı `OF` de işlem sonundaki durumu saklar. Dallanma komutları bu bayrak bitlerine bağlı olarak yeni adrese sapar ya da bir sonraki adrese devam eder. Örneğin `DEC CX` komutu `CX` yazmacını bir azalttığında `CX` sıfır olmazsa `ZF` bayrağını sıfır, `CX` sıfıra düşerse `ZF` bayrağını bir yapar. Bu komutun ardından kullanılacak `JZ L1` komutu `ZF=1` ise `L1` ile etiketlenmiş adrese atlar.

1.3.11 Belleğe yazılı program kavramı

Von Neumann tarafından tanımlanmış bu kavrama göre bilgisayarın denetim birimi bellekteki komutları belli bir adresten başlayarak adres sırasıyla işler. Bu sıra ancak program sayacındaki bir sonraki komut adresi işlenen bir program akış komutuyla değiştirilirse bozulur. Bu durumda bir sonra işlenecek adres te değişmiş olur.

1.3.12 CPU nasıl çalışır

CPU aşağıdaki işlemleri düzenli olarak tekrarlayarak çalışır.

1 - Denetim birimi *program sayacı*na göre sırası gelen komutu bellekten okur ve *komut yazmacı*na yerleştirir. Bu adıma *komut kapma*[38] denir. Bu adımın sonunda denetim birimi program sayacını bir sonraki komutu gösterecek şekilde arttırır.

2 - Denetim birimi komut yazmacındaki komutu çözümleyerek kaç periyodluk olduğunu ve her periyodunda veriyollarının nasıl değişmesi gerektiğini belirler. Komut
a-yazmaçtan yazmaca veri yollama,
b-yazmaçla bellek arasında veri yollama

[37] program flow control instruction
[38] instruction fetch

c-yazmaç ya da bellek üzerinde aritmetik mantık işlemi
d-bit veya durum değiştirme komutu
e-koşullu dallanma
gibi bir program akış denetimi komutu olabilir. Bu komut türlerinden (a, b) veri aktarma[39], aritmetik-mantık, ve (e) kontrol türü komut olarak sınıflandırılır.

3 - Denetim birimi, komutun işini bitirince sırası gelen komutu işlemek üzere ilk basamağa döner.

1.3.13 Kesme servisi

Bu işlem akışının yanısıra denetim birimi *kesme* [40] sinyallerini de takip eder. Herhangi bir anda kesme sinyali gelirse sırasıyla i) işlenmekte olan komutu tamamlar; ii) program sayacındaki kesme dönüş adresini yığıta iter; iii) program sayacına *kesme vektörü*nü yerleştirir. Böylece bir sonraki adresteki komut yerine kesme vektörü adresindeki ilk komut işletilir. Bu komut genellikle kesme servis programına atlayacak bir `JMP` komutudur. Kesme servis programı `IRET` *kesmeden dönüş* komutuyla biter. Bu komut, yığıttan çektiği kesme dönüş adresini program sayacına geri koyarak kesme servisi bitiminde program akışının ana programın kaldığı yere dönmesini sağlar.

1.3.14 Program çalıştırma hızı

Günümüz teknolojisinde program işleme hızını belirleyen etmen program komutlarının bulunduğu belleğe erişim hızıdır. CPU saati, iç veri yollarındaki veri akışını belirler. Bellekten komut veya veri okuma ise birden çok adımda gerçekleşir. Bellek yavaş çalışıyorsa işlenecek komutların bellekten yazmaca aktarırken CPU pek çok saat döngüsünü verinin hazır olmasını beklerken boşta geçirir. Boşa giden döngülere kayıp[41] ya da bekleme[42] döngüsü denir.

[39] data transfer
[40] interrupt request signal
[41] waisted clock cycle
[42] wait clock cycle

Bölüm 2

Adres ve Yazmaç Mimarisi

Bu bölümde 8086 işlemciyi oluşturan yazmaçları, bu yazmaçların birbiriyle ilişkisini ve bellekteki yazmaçlara erişme mekanizmalarını göreceğiz.

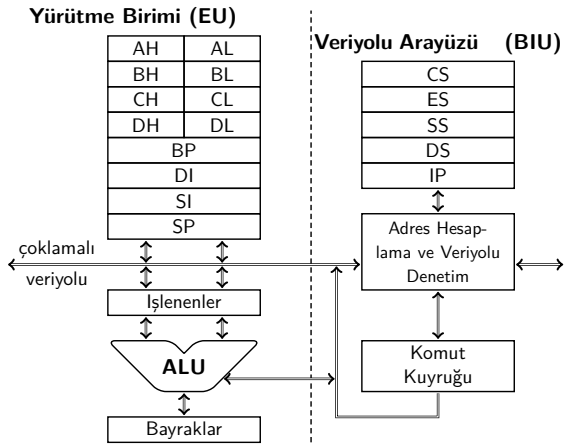
2.1 Yazmaç Mimarisi

8086 mimarisinin kullanıcıyı ilgilendiren yazmaçları şunlardır:
AX, BX, CX, DX : Genel amaçlılar.
Bunlar AX=AH:AL gibi 8-bitlik iki bölümlüdür
SP (yığıt) and BP (taban) 16-bit, İmleçler:
SI (kaynak) and DI (hedef) 16-bit, Indeks:
CS, DS, SS and ES 16-bit bölütler ,
Program İmleci: IP (instruction-ptr) 16-bit,
Durum Bayrakları Sayacı: FR (flag) yazmacında tutulur.
(CF (carry: elde),
PF (parity: eşlik),
AF (auxiliary carry: yardımcı elde),
ZF (zero: sıfır),
SF (sign: işaret),
TF (trap: yakalama),
IF (interrupt: kesme),
DF (direction: yön),
OF (overflow: taşma))
Adres bölütleme 8086 mimarisinde önemli bir yer tutar.

Genel amaçlı olarak adlandırılan AX, BX, CX ve DX yazmaçları iki 8-bitlik parça olarak erişilecekse AH ve AL, BH ve BL, CH ve CL, DH ve DL olarak çağırılır.

Örnek 2.1.

AX yazmacına 1A28H koyduktan sonra AL yazmacına 0D8H toplarsak AX yazmacındaki sayı kaç olur?



Şekil 2.1: 8086 yazmaç mimarisi

Çözüm: AX yazmacına 1A28H koymakla AH yazmacına 1AH, ve AL yazmacına da 28H konmuş olur. AL yazmacındaki 28H sayısına 0D8H toplandığında

$$\begin{array}{r} 28\,\mathrm{H} \\ +\ \mathrm{D8\,H} \\ \hline =1\,00\,\mathrm{H} \end{array}$$

eder. AL yazmacına 00 girer. AL yazmacındaki toplama AH yazmacını etkilemez. Dolayısıyla AH de gene 1AH kalır. İşlem sonunda AX yazmacında 1A00H kalır. Üçüncü basamaktaki 1 toplama sonucu oluşan son eldedir ve elde bayrağına gider.

Örnek 2.2.

AX yazmacına 1A28H koyduktan sonra AX yazmacına 0D9H toplarsak AX yazmacındaki sayı kaç olur?

Çözüm: AX yazmacında 1A28H varken 0D9H daha eklersek

$$\begin{array}{r} \mathrm{1A28\,H} \\ +\ \mathrm{00D9\,H} \\ \hline =\ \mathrm{1B01\,H} \end{array}$$

elde ederiz. İşlemin sonunda AH=1BH, AL= 01H, AX= 1B01H, elde bayrağı 0 olur.

Örnek 2.3.

AX yazmacına 1A28H koyduktan sonra AH yazmacına 0F2H toplarsak AX yazmacındaki sayı kaç olur?

Çözüm: 0F2H ile AH=1AH toplanır ve

```
   1AH
 + F2H
------
 =10CH
```

eder. Böylece AH=0CH ve AX=0C28H olur. Toplamada oluşan elde biti ise elde bayrağına gider.

2.2 8086 Adres Bölütleme

8088 yongasının tasarlandığı yıllarda yonga paketlerinde bacak sayısının kısıtlı olması nedeniyle 8088 ve 8086 adres yolu 20 hattır. Üretici firma İntel bu yongaların 8080 ve 8085 ile yazılım ve donanım açısından uyumlu olması için adres yolunu genişletmek üzere adres bölütlemeyi[1] tercih etmiştir.

8086 öncesi CPU lardaki 16-bit adres yazmaçlarıyla erişilebilecek adres uzayı 2^{16} =64k = 65536 yazmaçtır. Bu sayı uygulama programlarıyla verilerinde önemli bir sınır oluşturur. Bölütlü adreslemede adres yazmacı gene 16 bittir ve bölüt yazmacıyla gösterilen paragraf adresine olan farkını tutar. Bu adres farkı *ofset*[2] olarak adlandırılır. Böylece 20 bitlik fiziksel bellek adresi FA, hem adres yazmacındaki ofset, hem bölüt yazmacındaki paragraf adresi değerinden hesaplanır. Her paragraf 16 bayttan oluştuğu için

Fiziksel adres $= FA =$ bölüt $\times 16 +$ ofset

olur. Adresler bu yolla 20-bit e çıkınca adres uzayı[3] da 2^{20} =1 Mbayt a ulaşır. 8086 da program kodu için kod bölütü CS[4], veri adresleme için veri bölütü DS[5], ve ekstra bölüt ES[6], yığıt adresleme için de yığıt bölütü SS[7] kullanılmaktadır.

2.2.1 Veri adresi bölütleme

Örnek 2.4.

Veri adresi bölütlemede DS yazmacı paragraf adresini taşır. DS de 0100H varken DATA1 ofset adresi 0123H ise fiziksel adresi bulalım

Çözüm:

[1] Segmentation
[2] offset
[3] address space
[4] code segment
[5] Data segment
[6] Extra segment
[7] Stack segment

16=10H olduğundan
0100 H×16=0100 H×10 H=0100*0* H eder.
FA_{DATA1} =DATA1 + 16×DS
=0123 H + 0100*0*H =01123 H

8086 da *fiziksel adres* dışında bölütte tutulan *paragraf adresi* komutta kullanılan *ofset adresi* ve bu ikisinin toplanmadan biraraya getirilmesiyle oluşan *mantıksal adres* kavramları vardır.

Örneğimizde DATA1 için ofset adresi 0123 H, paragraf adresi 0100 H, ve mantıksal adres 0100:0123 H olarak yazılabilir.

2.2.2 Kod adresi bölütleme

8086 da program sayacı da bölütlenmiş adres sistemi kullanır. Bir sonraki komutun adresinin ofseti komut imleci[8], IP, olarak adlandırılan özel amaçlı bir yazmaçta tutulur. Paragraf adresi ise CS yazmacında durur. Böylece bir sonraki komutun mantıksal adresi CS:IP olarak gösterilir ve fiziksel adresi FA= CS×10H + IP olarak hesaplanır.

Örnek 2.5.

Kod adres bölütleme

CS ve IP yazmaçlarında 0100 H ve 4321 H varsa bir sonra çalışacak komutun fiziksel adresi nedir?

Çözüm:

Mantıksal adres CS:IP=0100:4321 H
Fiziksel adres FA= 0100*0* H + *0*4321 H =05321 H.

2.2.3 Yığıt adresi bölütleme

8086 işlemcisi kesmeler, bazı komutlar, ve programlama için gereken geçici verileri saklamak üzere göstergesi pek çok komutla güncellenen bir yığıt[9] yapısı kullanır. *Yığıt imleci*[10] SP 16 bitlik bir yazmaçtır. Yığıtın 20 bitlik adres uzayındaki herhangi bir adresi belirleyebilmesi için *yığıt bölütü*[11] yazmacıyla birlikte kullanılır. Yığıtı kullanan komutları ileride ayrıntılı olarak göreceğiz.

Örnek 2.6.

Yığıt adresi bölütleme

[8] Instruction Pointer (IP)
[9] stack
[10] stack pointer SP
[11] stack segment

SS ve SP yazmaçlarında 3000 H ve FFF5 H varsa yığıt tepesinin fiziksel adresi nedir?

Çözüm:

Mantıksal adres SS:SP =3000:FFF5 H

Fiziksel adres FA= 3000o H + FFF5 H =3FFF5 H.

Bölütü bidirilmemiş adreslerde ofset yazmacına göre çeviricide önceden tanımlı bölütler[12] aşağıdaki tabloda belirtilmiştir.

Ofset yazmacı	IP	BX SI DI	BP SP
Ön tanımlı bölüt	CS	DS	SS

2.3 Çevirici (assembler) satır yapısı

Çevirici dilinde yazılmış program kaynağı satırlar halinde düzenlenir. Tipik bir çevirici satırının genel biçimi şöyledir.

```
[etiket:] [komutadı [işlenenler] ] [;yorumlar]
```

Bir satırda bir etiket, yorum, direktif ya da program komutu yazılabilir. Direktifler sembolleri, makroları, kod yerleşimini tanımlamak ya da koşullu çeviri yapabilmek için kullanılır. Program komutları ve direktifler komutçuk ve işlenenlerden oluşur. Yorum alanı noktalı virgül ';' ile başlar ve satırın bitimine kadar devam eder. Yorum alanı kişilere bilgi aktarmak içindir, çeviride etkisi olmaz. Programda sabit değerler, yazmaçlar, ve bellek değişkenleri için semboller kullanmak programa okunurluk katar.

Örnek çevirici programı

```
1   .MODEL SMALL
2   .STACK 64
3   .DATA
4 DATA1  DB 52H
5 DATA2  DB 29H
6 SUM    DB ?
7   .CODE
8 MAIN PROC FAR  ;Program başlıyor
9   MOV AX,@DATA ;Data bölütü adresi ax'e kopyalandı
10  MOV DS,AX    ;  ve ax DS yazmacına aktarıldı
11  MOV AL,DATA1 ;İlk sayı AL yazmacına kopyalandı
12  MOV BL,DATA2 ; ikinci sayı BL ye kopyalandı
13  ADD AL,BL    ; AL  <- AL+BL
14  MOV SUM,AL   ; SUM <- AL+BL
15  MOV AH,4CH   ; DOS işletim sistemine dönülüyor
```

[12] default segment

```
16   INT 21H      ;    dönüş gerçekleşti
17 MAIN ENDP
18   END MAIN
```

Satırlardan 1 ve 2 deki .MODEL ve .STACK amaçlanan programın düzeni, yığıtı ve verinin başlangıç noktasına ilişkin direktiflerdir. Satır 3 teki .DATA bellekte tutulacak değişkenlerin başlangıcını belirten direktiftir. Satır 4 teki DATA1 bellekte tutulacak değişkenin adıdır ve DB bu değişkenden başlanarak birer bayt yer ayırılacağını, 52H bu bir bayt yere programla birlikte 52H değerinin yükleneceğini bildirir. Satır 7 deki .CODE program kodunun başlayacağını belirten direktiftir. Satır 8 ve 17 de 9 ile 16 satırları arasında MAIN adlı bir programın tanımlanacağı bildiriliyor. Satır 9 dan 16 ya kadar MOV, ADD ve INT işlemcinin işleyeceği komutçuklardır. Örneğin satır 11 deki MOV komutuyla, DATA1 ile etiketlenmiş bellek yazmacındaki değer AL yazmacına kopyalanıyor.

2.3.1 MOV ve ADD komutları

Adresleme modlarını anlatmadan önce bu modları gösterebileceğimiz MOV ve ADD komutlarını görelim. MOV komutu bir yerdeki verinin başka bir yere kopyalanmasını sağlar. 8086 da en fazla kullanılan komuttur ve 8 ya da 16 bit yazmaçlar için beş değişik adres moduyla kullanılabilir.

MOV *hedef*, *kaynak*

komutundaki *hedef* işleneni daima nereye kopyalanacağını, kaynak işleneni ise neyin kopyalanacağını gösterir.

8 bitlik MOV örnekleriyle başlayalım.

```
1    MOV CL,55H
2    MOV DL, CL
3    MOV AH, DL
4    MOV CH, BH
5    MOV AL,'5'
```

Birinci satırdaki 55H bir baytlık bir hex değişmezdir ve bu değer CL yazmacına kopyalanacaktır. Satır 2, 3, 4 dekiler kullanılan 8-bit yazmaçların tümü AL, AH, BL, BH, CL, CH, DL, DH yazmaçlarıdır. Satır 5 teki '5' ise değişmezin 35H (=ASCII karakter 5) olduğunu belirtir.

8-bit veri bayrak yazmacı dışındaki bütün 8-bit yazmaçlar arasında MOV komutçuğu ile birinden diğerine kopyalanabilir.

16-bit yazmaçlı ve değişmezli MOV örnekleri

```
1    MOV CX, 468FH
2    MOV AX, 2
3    MOV DX, AX
4    MOV DI,BX
5    MOV DS,SI
6    MOV BP,DI
```

ilk iki satırdaki 468FH ile 2, 16-bit değişmez değerlerdir. Satır-2 deki 2 nin 16-bit olduğunu kopyalanacağı hedef yazmaç olan AX e bakarak anlıyoruz. Diğer satırlarda hem kaynak hem hedef işlenenleri 16 bitlik yazmaç isimleridir.

ADD komutunun yapısı MOV komutuna çok benzer

```
ADD hedef, kaynak
```

komutunda hedef ile gösterilen yerdeki değer kaynak ile gösterilen değere toplanınca çıkan sonuç hedefle gösterilen yere yazılır. Aynı MOV gibi, ADD da 8 ya da 16 bitlik olabilir.

2.4 Adresleme modları

Adresleme modlarını yazmaç ve bellek adresleme modu olmak üzere iki bölümde inceleyebiliriz.

2.5 Yazmaç Adresleme modları

Daha önce MOV üzerinde gördüğümüz gibi 8 bitlik ve 16 bitlik olmak üzere iki çeşit yazmaç adı vardır. Uzunluğu farklı olan yazmaçlar bir MOV komutunun hedef ve kaynak işleneni olarak birlikte kullanılamaz.

2.6 Bellek Adresleme modları

8086 da *rastgele erişimli belleği* kullanabilen altı çeşit *bellek adresleme modu* vardır.

– *Anlık Değer* Modu (0123 H, 100, 10010100 B gibi)
– *Doğrudan Adresleme* Modu ([0123 H])
– *Yazmaç Değeriyle Dolaylı Adresleme* Modu ([BX])
– *Eklemeli Dizinle Adresleme* Modu ([DI] + 12)
– *Tabanlı Dizinle Adresleme* Modu ([BX] + 12)
– *Tabanlı ve Eklemeli Dizinle Adresleme* ([BX] [DI] + 4)

Adresleme modlarını anlatırken bir baytlık rastgele erişimli bellek yazmacını RAM_8(adr) ile, iki baytlık rastgele erişimli bellek yazmaçlarını

RAM_{16}(adr) ile göstereceğiz. Intel 8086 da *küçük uçtan yerleştirme*[13] kullandığı için RAM_{16}(adr) ile gösterdiğimiz 16 bitlik değerin soldaki 8-biti RAM_8(adr+1), sağdaki 8-biti ise RAM_8(adr) bellek yazmacıdır.

$$RAM_{16}(\text{adr}) = 256 \cdot RAM_8(\text{adr}+1) + RAM_8(\text{adr})$$
$$= (RAM_8(\text{adr}+1); RAM_8(\text{adr}))$$

2.6.1 Anlık (Literal) adresleme modu

Komutun kaynak işleneni komutun işlendiği anda literal değeriyle kullanılırsa işlenen anlık adreslemeyle iletilmiş olur.

Örnekler

```
MOV AL, 12H
MOV AX, 123H
MOV BL, 1
```

Literal değer gerçekte komutla birlikte bellekte durduğundan *anlık* değerler de bir çeşit bellek adresi sayılabilir. Çeviricide literal değerleri etiketlemek için EQU çevirici talimatı kullanılır.

Örnek

```
1 BIRSAYI EQU 2
2       ...
3       MOV AX,BIRSAYI
4       ADD AX,2+8*BIRSAYI
```

Burada satır-1 de BIRSAYI nın sabit olarak 2 değerini taşıdığı belirtiliyor. Satır-3 bu değeri anlık değer olarak kullanıyor. Satır-4 ise içinde BIRSAYI sabit değerini bulunduran sabit bir aritmetik ifadenin sonucunu anlık değer olarak kullanıyor.

2.6.2 Doğrudan Adresleme Modu

Doğrudan adresleme modunda, parametere olarak verilen 16 bitlik sabit bir değer adres olarak kullanılarak belleğe erişilir.

Örnek

```
MOV AL, DS:[1234H]
```

ile RAM_8(DS:1234H) değeri AL yazmacına kopyalanır.

```
MOV [CS:1234H], AX
```

ile AL deki değer RAM_8(CS:1234 H) bellek yazmacına yazılır.

[13] little endian

2.6.3 Yazmaçlı Dolaylı Adresleme Modu

Bu modda, bir yazmacın değeri adres olarak kullanılarak herhangi bir bellek adresindeki verilere erişilir. Bu modda, işlenende belirtilen yazmaç uygun bölüt adresi yazmacıyla dolaylı olarak kullanılır. Yazmaçlı dolaylı adreslemede yalnızca BX, DI, SI yazmaçları kullanılabilir.

Örnekler

`MOV AL, [BX]`

komutunda RAM_8(DS:BX) değeri AL ye kopyalanır.

`MOV AX, [SI]`

komutunda RAM_{16}(DS:SI) deki iki bayt AX e kopyalanır.

`MOV [SI], DX`

komutunda 16 bitlik DX değeri RAM_{16}(DS:SI) bellek yazmacına kopyalanır.

Bölüt yazmacını değiştirmek için komutun başına ya da yazmacın başına istenen bölüt yazılır

`CS MOV AL, [BX]` FASM çeviricisi için

`MOV AL, [CS:BX]` MASM çeviricisi için

2.6.4 Eklemeli Dizinli Adresleme

Bu modda, yazmaçla dolaylı adreslemeye ek olarak, parametre olarak verilen bir sayıyla birlikte, bölüt içi göreli adres oluşturulur ve yazmaçla ilişkilendirilmiş bölüte erişilir

Örnekler

`MOV AL, [DI+10]` ; RAM_8(DS:DI+10) -> AL

`MOV AX, [SI]+10` ; RAM_{16}(DS:SI+10) -> AX (böyle de olur)

Kullanılacak yazmaçlar DI, SI, BP, SP olabilir. Bölüt belirtilmemişse DI ve SI için DS bölütü, BP ve SP için ise SS bölütü kullanılır.

2.6.5 Tabanlı Eklemeli Adresleme

BX ve BP yazmaclarının ana işlevi tek boyutlu dizilerin *taban adresi*'ni oluşturmaktır. Tabanlı adresleme modunda, yazmaç içeriği dolaylı adres olarak kullanılır. Eklemeli adreslemede, yazmaç içeriğine ek olarak verilen değişmez sayı da dolaylı adrese eklenir. Fiziksel adrese yazmaçla ilişkilendirilmiş bölüt ile erişilir.

```
MOV AL, [BX+10] ; RAM8(DS:BX+10) -> AL
MOV AL, [BX]+10 ; RAM8(DS:BX+10) -> AL (diğer biçimi)
MOV CX, [BP]+4 ; RAM16(SS:BP+ 4) -> CX
```

BP imleci özellikle altyordamlara yığıtta aktarılan dizi yapılarına erişim için kullanılır.

2.6.6 Eklemeli Tabanlı Dizinli Adresleme

İki ve daha fazla boyutlu matris yapılarına erişmek için BX tabanı yanısıra DI ve SI indis yazmaçları ya da BP SP imleç yazmaçlarından biri daha adrese eklenebilir. Bu modda, yazmaçla dolaylı adreslemeye ek olarak, parametre olarak verilen bir sayıyla birlikte, bölüt içi göreli adres oluşturulur ve indis ya da imleç yazmacıyla ilişkilendirilmiş bölüt ile erişilir.

```
MOV CX, [BX] [SI]+4 ; RAM16(DS:BX+SI+4) -> CX
MOV CX, [BX] [SP]+2 ; RAM16(SS:BX+SP+4) -> CX
```

2.7 Bölüt Çakışması ve Katlanması

Bölütün adres uzayı imleç yazmaçlarının 16-bit olması nedeniyle 64kBayttır. Bölüt yazmacı bu 64kBayt pencerenin başlangıç adresini gösterir. DS, SS ve CS bölütleri duruma göre hiç çakışmayan[14] 64 kBayt pencereler, kısmen çakışan[15] pencereler, ya da tümüyle çakışan[16] 64kBaytlık bir pencere olarak oluşturulabilir.

Bölüt penceresi gerçek modun 1MBayt bellek uzayından taşacak kadar yüksek bir adresten başlıyorsa FFFFFH adresinden taşan bölüm 00000 adresinden başlayarak devam eder. Bu olaya *bölüt katlanması*[17] denir.

2.8 Bayrak yazmacı

Bayrak yazmacı doğrudan ALU da yürütülen komutun işlem sonucuna göre güncellenir. Bayrak yazmacının bitleri şu şekilde düzenlenmiştir.

b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
OF	DF	IF	TF	SF	ZF		AF		PF		CF

`CF`: Elde bayrağı
`PF`: Eşlik bayrağı
`AF`: Yardımcı artık

[14] non-overlapping
[15] partially overlapping
[16] fully overlapping
[17] segment wrap-around

ZF: Sıfır bayrağı
SF: İşaret bayrağı
TF: Yakalama bayrağı
IF: Kesme bayrağı
OF: Taşma bayrağı

Bu bayrakları güncelleyen komutları ve koşullu dallanmaları gerçekleştirmek üzere dallanma komutlarıyla kullanımını ilerideki bölümlerde ayrıntılarıyla göreceğiz.

Bölüm 3

Çevirici (Assembly) Dilinin Temel Bileşenleri

3.1 Sabitler ve ifadeler

3.1.1 Sabitler

Sabit sayı ve kodları değişik taban ve kodlama yöntemleriyle yazabilmek mümkündür. Bir sabit değerin genel biçimi

[{ + | - }] rakamlar [taban]

olarak gösterilebilir. İşareti yazılmamış değer pozitiftir. Taban seçimi değerin ikili, onlu, onaltılı veya sekizli rakamlarla yazılabilmesini sağlar. Eğer taban belirtilmemişse öntanımlı taban etkili olur. Öntanımlı tabanı direktifler ile değiştirmek mümkündür. Sabit değerler yalnızca rakam ve tırnak karakteriyle başlayabilir. onaltılı rakamlar harfle başlıyorsa en başa 0 eklenir.

Taban karakterleri:

h – onaltılı ("hexadecimal")
d – onlu ("decimal")
b – ikili ("binary")
o – sekizli ("octal")
r – kodlanmış gerçel ("encoded real")

Örnek 3.1.

Tabanı veya Kodu	Taban Karakteri	Yazılışı	Değeri
Onluk tabanda	d	30d	30
Onaltılık tabanda	h	6Ah	106
		0A5h	165
Sekizli tabanda	o	22o	18
İkili tabanda	b	1101b	13
ASCII kodlanmış	' '	'A'	65
		'52'	35,32

,

3.1.2 İfadeler:

Tamsayı ifadelerde işlem öncelikleri (), + - işaretleri, *, /, MOD, + - toplama ve çıkarma sırasındadır.

Örnek 3.2.

a	16/5	=3
b	−(3+4)*(6−1)	=−35
c	−3+4*6−1	=20
d	25 mod 3	=1

Türkçede reel yada gerçek sayı olarak ta bilinen gerçel sayılar [1] IEEE 754 standardında hex kodlanarak verilebilir. En soldaki bitten başlarsak bit-31 sayının işareti s_f, bit-30~bit-23 sayının kaydırılmış üsteli q_f, bit-22~bit-0 sayının eksik birli kesiri c_f dir. Sayının işareti $s = (-1)^{s_f}$, işaretli üsteli q=q_f−127, ve birli kesiri c=$1 + c_f$ olarak hesaplanır ve sayı

r =$s \cdot c \cdot 2^q$

olarak elde edilir.

Örnek 3.3.

0011 1111 1000 0000 0000 0000 0000 0000_2 sayısı

=3F800000 H olarak kodlanır.

s_f=0, s=1; c_f =0, c=1; q_f=01111111_2=127 , q=0;

r= $1 \cdot 1 \cdot 2^0 = 1$ bulunur.

Bu sayı bellekte bayt dizisi olarak saklanırken küçük adresten büyüğe doğru 00h 00h 08h 3Fh sırasında durur. 8086'nın yan işlemcisi 8087 gerçel sayılar üzerindeki aritmetik, trigonometrik, ve üstel işlemleri gerçekleştirebilir.

Örnek 3.4.

R=25.5 sayısını IEEE 754 biçiminde 32-bit olarak yazalım.
Sayı pozitif olduğundan s_f=0 olur.
$2^5 < 25.5 \leq 2^4$ olduğundan üsteli $q = 4$, ve birli kesiri
$c = 25.5/16 = 1.59375$, eksik birli kesiri $c_f = c - 1 = 0.59375$ olur.
İkili sayıya çevrilince c_f=0.100011_2 çıkar. Kaydırılmış üsteli
$q_f = 4 + 127 = 128 + 3$=10000011_2 olarak yazılır. Birleştirince
R=0**100 0001 1**100 0110 0000 0000 0000 0000_2 =41C60000Holur.
Kolay ayırdedilmesi için sayının üsteli koyu rakamlarla yazılmıştır.

Karakter ve dizgi sabitleri

Daha önce ASCII kodun tek veya çift tırnak içinde ifade edilebileceğini gördük. Daha uzun karakter dizgileri tek veya çift tırnak içinde yazılabilir. Her ASCII karakter tek bayt kaplar. en soldaki karakter bellekte en düşük adrese yerleşir.

[1] real numbers

3.2 Açıklamalar

Bir satıra noktalı virgül konursa noktalı virgülden sonrası çevirici program tarafından değerlendirilmez. Bu bölgeye programla ilgili açıklamalar[2] yazılabilir.

Başka açıklama sistemi kullanan assembly yazılımları da vardır. Ancak biz açıklamalarımızı noktalı virgülle sınırlayacağız.

3.3 Ayrılmış (Anahtar) sözcükler

Şunlar ayrılmış[3] sözcüklerdir.

İşlemci komut anımsatıcıları[4] MOV, ADD, ...
Derleyici talimatları .CODE .END PROC .SHORT, ...
Özellikler BYTE, WORD, ...
İşleçler FAR, SIZE, ...
Öntanımlı Semboller @DATA, @MODEL, ...

3.4 Belirteçler

Belirteçler[5] etiket veya isim sembolü oluşturmak ve bunlara gönderi[6] oluşturmak için kullanılır. 1-32 karakter uzunlukta olabilir. Büyük ve küçük harf ayrımı yoktur. İlk karakter dahil heryerde kullanılabilecek karakterler 'A'-'Z', 'a'-'z', '$', '?', '_', '@'; ilk karakter dışında diğer her yerde kullanılabilecek karakterler ise '0' - '9' ve '.' (nokta) karakterleridir. Türkçe özel karakterler gibi İngiliz alfabesinde olmayan karakterlerde büyük-küçük harf ayrımı oluştuğundan özel dikkat ve itina ister.

Örnek 3.5.

```
1 BEKLE: ADD CX,1
2   JNZ BEKLE
```

ya da

[2] comment
[3] reserved words
[4] mnemonics
[5] identifiers
[6] reference

```
1 BEKLE:
2   ADD CX,1
3   JNZ BEKLE
```

satırlarında ilk satırdaki bekle bir adres belirteci, üçüncü satıraki bekle ise bu adrese yönelik bir gönderi veya anmadır.

.

3.5 Deyimler

Deyimler[7] ya komut anımsatıcılardan[8] ya da talimatlardan[9] oluşur. Bir satıra ancak bir deyim yazılabilir. Derleyici, MOV ve ADD gibi komut anımsatıcıları işlenenleriyle birlikte makine koduna dönüştürür. Talimatlar kodda yer almayan ama derleyiciye kodun nasıl oluşturulacağını bildiren DB, PROC, ENDP gibi derleyiciye has anımsatıcılarla oluşturulur.

Örnek 3.6.

```
1  JMP START
2 SAYI1 DB 2
3 SAYI2 DB 5
4 TOPLAM DB ?
5 START:
6  MOV AL,CS:SAYI1
7  ADD AL,CS:SAYI2
8  MOV CS:TOPLAM,AL
9  END
```

programında 1, 6, 7, 8. satırlar komut deyimleri; 2, 3, 4. satırlar etiketli veri talimatı, 9. satır bitirme talimatı; 5. satır kod etiketi deyimidir.

3.6 Komutlar anımsatıcılar ve işlenenler

Tipik bir komut satırı aşağıdaki biçim ve bileşenlerden oluşur.

```
1 [etiket:] [anımsatıcı [işlenenler] ] [; açıklamalar]
```

Komut satırları derleyici tarafından makine koduna dönüştürülerek işlenebilir bir dosya içine yazılır. Oluşan makine kodunun adresi ileride gerekecekse etiketle adlandırılır.

[7] statement
[8] instruction mnemonics
[9] directives

Örnek 3.7.

```
1 ADDLP: ADD AL,BL ; AL<-AL+BL
2   JZ ADDLP       ; toplam sıfırsa etikete atla
```

İlk satırda komut etiketi `ADDLP`, komut anımsatıcı `ADD`, işlenenler `AL` ile `BL` dir. İkinci satırda ise etiket kullanılmamış, komut anımsatıcı ve işleneni sırasıyla `JZ` ve `ADDLP` dir. İkinci satırın sonundaki kısım açıklamadır.

Anımsatıcı ve işlenenler komutun makine kodunu oluşturmak için gereken bileşenlerdir. İstenirse komut ya da programla ilgili açıklamalar noktalı virgül karakterinden sonra yazılır ve bu kısım kodlamaya dahil olmaz.

Etiket adres belirtecidir ve belirteç kurallarına uygun olmalıdır. Çeviricilerin özellikle eski sürümleri 31 karakterden uzun etiketlerin sadece ilk 31 karakterini kullandığından hatalara neden olabilir. İlk karakteri rakam ya da nokta olamaz. Ayrılmış sözcükler etiket belirteci olamaz.

Komut anımsatıcıları ve işlenenleri işlemcinin komutlarına bağlı olarak işlemciye özeldir. 8086 Assembly komut anımsatıcılarını ileriki bölümlerde sınıflayarak ayrıntılarıyla anlatacağız. Genel hatlarıyla işlenenler bir sabit değer, ifade, yazmaç belirteci, bellek adresi ifadesi, ya da bellek belirteci (veri etiketi) olabilir.

Örnek 3.8.

Çeşitli komut-işlenen biçimlerine örnekler

Komut biçimi	anımsatıcı	işlenen	açıklama
işlenensiz	stc		; elde bayrağını 1 yap
tek işlenenli	inc	AL	; AL yazmacını arttır
iki işlenenli	mov	AL, 2	; AL ye 2 koy

Örnek 3.9.

Çeşitli talimat-işlenen biçimlerine örnekler

Talimat Örneği	açıklama	işlenen1 tipi
sayi1 EQU 2	sayı2 yi sabit 2 tanımla	sabit
sayi2 DB 0	sayı1 i bayt değişken tanımla	sabit
sayi3 DW sayi1	sayı3 ü sözcük değişken tanımla	sabit belirteç

Örnek 3.10.

Çeşitli komut-işlenen biçimlerine örnekler

Komut Örneği	İşlem	işlenen1	işlenen2
mov cx,5	cx←5	yazmaç	sabit
mov cx,sayı1+5	cx←sayı1+5	yazmaç	sabit ifade
mov [3],cx	RAM_{16}(DS:0003)←cx	RAM	yazmaç
mov ax, [bx]	ax←RAM_{16}(DS:BX)	yazmaç	dolaylı yazmaç
mov ax, [bx+4]	ax←RAM_{16}(DS:BX+4)	yazmaç	yazmaçlı ifade

3.7 Derleyici Talimatları

Derleyici talimatları kaynak programın derlenmesi ve listelenmesini biçimlendirir. Sadece derleme sırasında dikkate alınırlar ve sadece derleyiciyi etkilerler. Karşılığı bir makine kodu yoktur ancak dolaylı yollarla bir kısım komutların makine koduna nasıl dönüşeceğini belirlerler. Farklı derleyiciler farklı derleyici talimatlarına ve farklı bildirimlere sahip olabilir.

Yazacağımız programlarda kullanmamız gereken belleği düzenleyici derleyici talimatlarını tanıtalım.

3.7.1 Program düzeni talimatları

END: Programın sonunu belirler.
SEGMENT ... ENDS : Bölüt tanımlamak içindir
PROC ... ENDP: Yordam (prosedür) tanımlamak içindir.
ASSUME: Derleyiciye hangi bölütün hangi amaçla kullanılacağını belirtir. -.EXE program derlerken veri bölütü DS yazmacına atanmalıdır.
.MODEL bellektipi : Bellek modelini kolayca istenen standard tiplerden birine uygun biçime getirir. Bellektipi aşağıdaki seçeneklerden biri olabilir:

bellektipi	kod	veri
tiny	tek-çakışık	tek-çakışık
small	tek	tek
medium	sınırsız	tek
large	sınırsız	veri yapısı sınırı 64k
huge	sınırsız	sınırsız

.STACK [yığıtbüyüklüğü] talimatı yığıt için bölüt hazırlar.

.DATA talimatı, bir veri bölütü başlatır. Bölütü sonlandırmak için ENDS gerekmez yeni bir bölüt başlatılınca kendiliğinden sonlanır.

.CODE talimatı, kodlama bölütünü başlatır. Yeni bir bölüt başlatılınca ya da END talimatı ile program sonlanınca kodlama bölütü de kendiliğinden sonlanır.

3.7.2 Bölüt yazmaçlarının ilkdeğerlenmesi

Veri etiketi verinin yalnızca ofset değerini saklar. Verinin fiziksel adresini oluşturmak için veri bölüt paragrafı değerinin DS yazmacına doğru olarak aktarılması gerekir. Bölüt paragraflarının fiziksel adresi ancak işletim sistemi programın nereye yükleneceğini belirleyince ortaya çıkar. Program yükleyici[10], işletim sisteminin programa ayırdığı bellek alanına bağlı olarak kaynakta `@DATA` anahtar kelimesinin kullanıldığı yere bölüt paragrafının değerini anlık işlenen olarak yerleştirir. Ancak, 8086 işlemcisinde MOV komutu anlık işleneni doğrudan bölüt yazmacına koyamadığından bölüt yazmacına konulacak değer önce AX yazmacına konur, ve AX ten DS yazmacına aktarılır.

Örnek 3.11.

Aşağıdaki programda 1, 5. satırlar bölüt talimatları, 2, 3, 4. satırlar etiketli veri talimatı, 6, 7, 9, 10, 11. satırlar komut deyimleri; 13 ve 14. satırlar COMMAND penceresine geri dönmek için gereken DOS işletim sistemi servisidir. 15. satır bitirme talimatı; 8 ve 12. satırlar açıklamalardır.

```
1   .DATA
2 SAYI1 DB 52h
3 SAYI2 DB 25h
4 TOPLAM DB ?
5   .CODE
6   MOV AX,@DATA ; veri paragrafı -> AX
7   MOV DS,AX    ;              AX -> DS
8 ; artık değişkene etiketiyle gönderi yapabiliriz
9   MOV AL, SAYI1
10  ADD AL, SAYI2
11  MOV TOPLAM,AL
12 ; işletim sistemine dönelim.
13  MOV AH,4CH
14  INT 21H
15  END
```

[10] loader

Satır 6 ve 7, veri paragraf adresinin DS yazmacına konulduğu satırlardır. Bu iki satır olmadan DS yazmacında eskiden kalma değer ile erişilen RAM_8(DS:SAYI1) bu programdaki SAYI1 adresine erişemez.

Bu programı eğer C ile yazsaydık kaynak kodu şöyle olacaktı

```
1 void main(void)
2   {
3    char SAYI1=0x52, SAYI2=0x25;
4    char TOPLAM;
5    TOPLAM = VERI1+SAYI2;
6    }
```

Programda ekrana yazdırılan ya da dönen hiçbir sonuç olmadığından kodun ne yaptığını anlamak için yürütülebilir kodu emulator veya debug programlarıyla takip etmek gerekecektir.

3.7.3 Veri Tipleri ve Talimatları

Bellekte değişken yeri ayırmak için gereken talimatlardan önce assemblerde öntanımlı veri tiplerini görelim

real4 32-bit IEEE754 kısa gerçel sayı tipi
byte, **sbyte** 8-bit işaretsiz ve işaretli veri tipi
word, **sword** 16-bit işaretsiz ve işaretli veri tipi
dword, **sdword** 32-bit işaretsiz ve işaretli veri tipi
qword, **sqword** 64-bit işaretsiz ve işaretli veri tipi

Aşağıdaki talimatlarlar bellekte istenen veri tipinde yer açmak için kullanılır.

DB bellekte 8-bit tipler için işlenen listesine uygun yer açar. İşlenenler virgülle ayrılarak listelenir. Böylece bir tek bayt ya da virgüllerle ayrılmış bir dizi bayt için kullanılabilir.

DB ? ; etiketlenmemiş ve ilkdeğer verilmemiş bir baytlık yer ayırır.

sayı1 DB 'a' ; sayı1 etiketiyle tek karakterlik yer açıp ilkdeğer olarak 'a' nın ASCII kodunu koyar

msg1 DB 'kolay gelsin' ; msg1 etiketiyle adreslenmiş 12 karakterlik yer açıp içine ilkdeğer olarak 'kolay gelsin' yerleştirir.

sayılar DB 10, 12h, 0Ah, '0Ah' ; sayılar etiketiyle adreslenen 6 baytlık yer açıp içine sırasıyla 0Ah, 12h, 0Ah, 30h, 41h ve 68h yerleştirir.

tampon100 DB 100 DUP(20h): tampon100 etiketiyle adresli 100 bayt yer ayırıp içine 100 tane 20h koyar. 20h boşluk karakterinin ASCII kodudur.

DW bellekte 16-bit veri tipleri için işlenen listesine uygun yer açar. İşlenenler virgülle ayrılarak listelenir.
DW ? : etiketlenmemiş 16-bit veri yeri açar. İlkdeğer vermez.
sayı2 DW 10 ; sayı2 etiketli 16 bit yer açar. İlkdeğer olarak adreslenen belleğe küçük uçtan[11] 000Ah yerleştirir. Böylece eğer data bölütünde kullanılmışsa
RAM_8(DS:sayı2) adresine 0Ah,
RAM_8(DS:sayı2+1) adresine 00h
yerleşmiş olur.
sayı3 DW 1234H, -2,1 : sayı3 etiketiyle adresli 16 bitlik üç yer açar ve ilkdeğer olarak küçük uçtan 1234H, 0FFFEH, 0001H yerleştirir. Sonuçta belleğe sayı3 ofsetinden başlayarak 34h, 12h, 0FEh, 0FFh, 01h, 00h yerleşmiş olur.
sayılar2 DW 5 dup(0), 15 dup(1): bellekte sayılar2 adresli 16 bitlik 20 yer açıp içine ilkdeğer olarak 5 tane 0000h, ve 15 tane 0001h yerleştirir

DD bellekte 32 bit veri tipi için işlenen listesine yer açar. İşlenenler virgülle ayrılarak listelenir. DD de aynı DB ve DW gibi liste veya dup() ifadesiyle kullanılabilir.
DD 123H : bellekte 23H 01H 00H 00H olarak ilkdeğerlenen 4 baytlık yer açar.

DT bellekte ASCII veya BCD kodlanmış 10 baytlık yer açar.

3.7.4 Sabit Değer ve Adres Başı Talimatı

EQU bir etikete sabit bir değer atar. Atama ancak bir kere yapılır ve nerede yapılırsa yapılsın bütün kaynak metnini etkiler.
sabitsayı EQU 3 : sabitsayı etiketini 3 yapar. Bu etiket nerede ve nasıl kullanılırsa kullanılsın sanki 3 yazılmış etkisi gösterir.
EQU ile DB, DW, DQ gibi bellek talimatları arasındaki en büyük fark EQU ile değer atamasının bellekte yer almamasıdır. Birden çok yerde kullanılan bir sabitin değeri program başında EQU ile atanır. Programdaki sabit değerli parametrelerin program başında tanımlanması EQU ile gerçekleşir.

ORG kaynakta yazıldığı yere karşılık gelen bellek ofsetini belirler.
ORG 0010h : bölütte o satırdan başlayarak derlenecek kod veya veriler 0010h ofsetinden başlar

```
.data
```

[11] little endian

```
sayı1 DB 01h
   ORG 0010h
sayı2 DW 1235h
```

örneğinde, `sayı1` data bölütünün 0000h ofsetli adresine yerleşir. Bu bellek yazmacını RAM_8(DS:0000) olarak gösterdiğimizi hatırlatırız. ORG 0010h kodlama imlecine 0010h koyar. Böylece bir sonraki satırdaki 16-bit sayı2 değişkeni RAM_{16}(DS:0010) adresine yerleşir.

3.8 Veri Bellek Haritası

İyi programlama pratiği programda verilerle kodların ayrı yerlerde ve bağımsız düzenlenmesini gerektirir. 8086 nın bölüt mimarisi veri ve kodun bağımsız düzenlenmesine kolaylık getirmektedir. Veri bölütü DS:0000 adresinden başlarken kod bölütü CS:0000 adresinden başlar. Böylece program fiziksel bellekte hangi paragraftan başlatılırsa başlatılsın program yükleyici yalnızca segment yazmaçlarını ayarlayarak kodun ve verinin her durumda 0000h ofsetle başlamasını sağlar.

Verileri ve kodun bilgisayar ana belleğindeki görüntüsü için standartlaşmış gösterim biçimleri oluşmuştur. Bunlar arasında en yaygın kullanılanı hex bellek haritasıdır. Bu biçimde en bellek içeriğini göstermek üzere önce başlangıç adresi ardından bir dizi bellek içeriği yazılır.

Örnek 3.12.

Veri talimatları

```
1       .DATA
2 SAYI1 DB 03h
3 METIN DB 'Selam'
4 SAYI2 DB 24h
5 SAYI3 DW 03h, 124h
```

verilmiş ise veri bölütünün bellek haritası

```
DS:0000: 03 53 65 6C 61 6D 20 24
DS:0008: 03 00 24 01
```

olarak yazılır.

Bu örnekte satır-2 deki 03h veri bölütünün başında olduğundan ofseti 0000h dir. Satır-3 deki 'Selam' beş karakterlik bir dizindir ve her karaktere karşılık belleğe konulan kod Şekil 1.1deki ASCII kod tablosu yardımıyla 53h 65h 6Ch 61h ve 6Dh olarak bulunur. Satır-4 teki 20h ve 24h daha sonraki bellek konumlarında yer alır. Örneğin 20h nin adresi 0000h den başlayıp sayılarak DS:0006 olarak elde edilir.

Örnek 3.13.

Aşağıdaki veri talimatları için bellek haritasını çıkarın ve SAYI2 değişkeninin ofset adresini bulun.

```
1     .DATA
2 SAYI1 DW  3, 5
3 METIN DB  '3,5',13,10
4       DB  32 dup(?)
5 SAYI2 DB  15,20
```

Çözüm:

```
0000: 03 00 05 00 33 2C 35 0D
0008: 0A __ __ __ __ __ __ __
0010: __ __ __ __ __ __ __ __
0018: __ __ __ __ __ __ __ __
0020: __ __ __ __ __ __ __ __
0028: __ 0F 14 __ __ __ __ __
```

Burada '3,5' ile tanımlanan karakterler için belleğe ASCII karşılıkları konuyor. 32 dup(?) ile değiştirilmeden atlanan 32 baytlık alan açıkça görülüyor. SAYI2 nin ofset adresi ise bellek tablosundan 0029h olarak elde ediliyor.

3.9 Çeviriciler ve Emulatorler

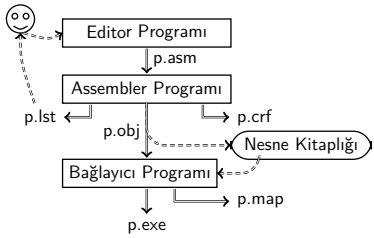
8086, 1980 lerden bu yana kullanılan bir işlemcidir ve 80x86 ailesinin ilklerindendir. Bugün kullanılmayan TURBO ASSEMBLERin (TASM) yanısıra Microsoft ASM lerin (MASM) de ilk sürümleri 8086 yı kullanmayı amaçlıyordu. Kaynak programı açık olmayan bu kod çeviricilerin yanısıra FLAT ASM açık kaynak GNU çeviricisi LINUX ekolünün ürünü olarak yaygınlaşmıştır. TASM, MASM ve Debug, FLAT ASM ve EMU8086 eğitsel amaçlı kullanımı kolay ve işlemcinin yazmaç seviyesinde işleyişini ayrıntılarıyla gösterebilen araçlardır.

Saydığımız çeviriciler endüstride kullanım yerleri olan güçlü araçlar olmalarına karşın birbirlerinden ciddi farklılıklar da gösterebilir. Örneğin FLATta bayt word ve dword tanıtma için DB, DW, DQ dışında başka bazı talimatları vardır, ancak ORG talimatı veri bölütü içimde kullanılınca kod başlangıç adresinde beklenmedik karışıklıklar oluşabilir. Biz bu üç çeviricinin de kullanabildiği talimat ve deyim biçimleriyle sınırlı kalacağız.

Başlangıçta yazdığımız programlarda giriş ve çıkış portu ya da klavye ve ekran kullanımıyla ilgili işletim sistemi servislerini kullanmadığımız için etkileşimli veri girişi yapamayacağız. Bu süre boyunca TASM veya EMU8086 gibi bir emulator/simulator kullanılması işlemcinin her bir komutunu nasıl çalıştırdığını izleyebilmek için önemli olacaktır. Daha ileride verilecek bütün örnekler EMU8086 ile FLAT kullanılarak çevrilmiş ve denenmiş kodlardır.

3.9.1 Çevirici dilinden yürütülebilir dosyaya

Şimdiye kadar verdiğimiz örnek programların bilgisayarda istendiği zaman nasıl yürütülebileceğini görelim. Örneğin Örnek 3.11 daki program EMU8086 ile kolayca yürütülebilir dosyaya dönüştürülebilir.



Şekil 3.1: Editör, Çevirici (Assembler) ve Bağlayıcı (Linker) ile yürütülebilir dosya oluşturma işlemi

Yürütülebilir dosya elde etmek için ilk basamak kaynak program dosyası oluşturmak üzere bu programı `-.ASM` uzantılı bir ASCII metin dosyasına yazıp diske kaydetmektir. Ardından TASM, MASM ya da EMU8086 nın kullandığı FLAT çevirici (assembler) ile kaynak programı nesne (obje)[12] koduna dönüştürürüz. Oluşan nesne kodunu bağlayıcı[13] programla uzantısı `-.exe` ya da `-.com` olan yürütülebilir dosyaya dönüştürürüz. Yürütülebilir programın adını 'command' penceresinde çağırdığımızda işletim sisteminin içindeki yükleyici[14] servisi -.exe dosyayı bellekte uygun bir yere yerleştirip bölütleri için gereken ilk değerleri tamamlar ve böylece program çalışmaya başlar.

Bazı program geliştirme ortamları ara basamakları gizleyerek sadece `-.exe` veya `-.com` dosyasını da oluşturabilir. Ek A'da ve B de TASM,

[12] object
[13] linker
[14] loader

MASM ve EMU8086 ortamlarıyla ilgili daha geniş bilgi bulabilirsiniz.

Bölüm 4

Komutlar ve Çevirici Kodları

4.1 Veri Aktarım Komutları

Daha önce adresleme biçimleriyle tanıdığımız MOV komutunu daha ayrıntılı inceleyelim. MOV komutunun işlenenlerini üç tipe ayırırız.

Anlık veri işleneni olarak yalnızca sabit bir sayı ya da sabit değerli ifadeler kullanılabilir. Uzunlukları hedefin uzunluğuna bağlı olarak bayt (8-bit), sözcük (16-bit) ya da çiftsözcük (32-bit) olabilir.

`MOV AX,5`

komut satırında birinci işlenen sözcük genişliğinde olduğundan ikinci işlenen 16-bitlik anlık veridir.

Yazmaç işleneni işlemci içindeki yazmaçlara erişmek üzere yazmaç adı yazılarak kullanılır. 16 bit yazmaçların bir bölümünün üst ve alt yarıları ayrı ayrı 8-bit yazmaç olarak kullanılabilir.

`MOV AX,5`

komut satırında birinci işlenen `AH` ve `AL` den oluşan 16-bitlik `AX` yazmacıdır.

Bellek işleneni kullanılarak bellekteki bir konum adreslenir.

4.2 Doğrudan Bellek İşleneni

Bellekteki bir veriyi adresleyen etiketler doğrudan bellek işleneni olarak kullanılabilir. İşlenen veri etiketine gönderi yaparsa derleyici tarafından adrese dönüştürülür.

Örnek 4.1.

```
1   .data
2 VAR1 DB 10h      ; adresi DS:0000 oldu
3   .code
4   mov AX, @DATA
5   mov DS, AX
6   mov AL,VAR1    ; AL <- 10h
7   mov AL,[VAR1] ; AL <- 10h
```

```
8  mov AL,0000h  ; AL <- 0  (anlık işlenen)
9  mov AL,[0000h]; AL <- 10h (var1in adresi DS:0000 )
```

kodunda satır-2 veri bölütünün en başında DS:0000 adresinde var1 etiketiyle ilkdeğeri 10h olan bir baytlık yer ayırıyor.

Satır-4 ve 5 DS e veri bölüt paragrafını koyuyor.

Satır-6 daki VAR1 gönderisi bu adresteki değerin kullanılmasını sağlıyor.

Satır-7 deki köşeli parantezli gönderi etiketlenen adresteki değeri ifade ediyor.

Satır-8 de 0000h anlık işlenen olarak kullanıldığından AL yazmacına 0 konuluyor.

Satır-9 da [0000h] bellek adresini belirtiyor ve VAR1 in bellek adresine rastladığından AL ye VAR1 in değerini kopyalıyor.

4.2.1 Veri adresi ve offset niteleyicisi

Veri etiketinin verinin kendisi yerine adresini bildirmesi için *offset* niteleyicisi kullanılır.

Örnek 4.2.

```
1   .data
2 var1 DB 10h  ; adresi DS:0000 oldu
3 var2 DB 20h  ; adresi DS:0001 oldu
4    org 001Ah
5 var3 DB 43h  ; adresi DS:0010 oldu
6   .code
7   mov AX, @DATA
8   mov DS, AX
9   mov ax, offset var2   ; AX <- 0001
10  mov ax, offset var3   ; AX <- 10h
```

Satır 9 ve 10 da offset var2 ve offset var3 işlenenleri ile ax yazmacına var3 değişkeninin ofset adresini koyuyoruz.

Örnek 4.3.

Veri bölütünde ofset 0010h te içinde 'kolay gelsin' yazılı metin1 etiketli bir dizin değişkenini takip eden 123 baytlık 1 dolu alan, ve ardından adres1 etiketli içi boş olan, ve, veri1 etiketli içinde 234h ile

5 bulunan 16-bitlik iki değişken oluşturun. Kod bölütünde ise ds yazmacını verilerin başına ayarladıktan sonra veri1 değişkeninin adresini adres1 değişkenine yerleştirin.

Çözüm: Data bölütünü başlatmak için .DATA kullanmalıyız. Ofseti 0010h adresine ulaştırmak için ORG 0010h kullanmamız gerekir. İstenilen değişkenlerden metin1 için DB, 16 bitlik veri1 ve veri2 için DW kullanmalıyız. Kod bölütünü başlatmak için .CODE talimatını kullanacağız.

```
1   .MODEL SMALL
2   .DATA
3 metin1 db 'kolay gelsin'
4   db 123 dup(1)
5 adres1 dw ?
6 veri1  dw 234h, 5
7   .CODE
8 MAIN PROC FAR
9   mov ax,@data
10  mov ds,ax
11  mov ax,offset veri1
12  mov adres1,ax
13 MAIN  ENDP
14  END MAIN
```

4.2.2 Verinin bir parçasına erişmek

Bir veri bayt olarak betimlenmiş olsa bile gerektiğinde word ptr niteleyicisi kullanılarak sözcük olarak kullanılabilir. Benzer şekilde, sözcük, çiftsözcük ya da dörtsözcük olarak betimlenmiş verilerin içinden istenilen bayta erişmek gerektiğinde byte ptr niteleyicisi kullanılır.

Örnek 4.4.

Aşağıdaki programda qword olarak tanımlı VERIQ1 ve VERIQ2 etiketli iki data var. Kod bölütünde VERIQ1 in baytlarını ters sırayla VERIQ2 ye yerleştiren bir program yazalım

```
1   .MODEL SMALL
2   .DATA
3 VERIQ1 dq 0123456789ABCDEFh
4 VERIQ2 dq ?
```

Çözüm: VERIQ1 değişkeni içinde VERİQ1 ofset adresinden başlayan toplam 8 bayt var. En sağdaki LSB (en küçük değerli) olanının adresi doğrudan VERIQ1 ile etiketlidir. Ancak VERIQ1 qword tipinde olduğundan tipini bayta dönüştürmek için byte ptr niteleyicisini kullanacağız.

Bir sonraki baytın adresi için bu adrese bir eklemeliyiz. Bu şekilde giderek en soldaki baytı adreslemek için `byte ptr VERIQ1+7` kullanmalıyız. `MOV` komutu bellekten belleğe kullanılamaz. Bu yüzden baytı önce `AL` yazmacına alıp sonra belleğe naklediyoruz.

```
1   .CODE
2   MOV AX,@DATA
3   MOV DS,AX
4   MOV AX,word ptr VERIQ1 ; AL:LSB AH:MSB
5   MOV byte ptr VERIQ2+7,AL
6   MOV byte ptr VERIQ2+6,AH
7   MOV AX,byte ptr VERIQ1+2
8   MOV byte ptr VERIQ2+5,AL
9   MOV byte ptr VERIQ2+4,AH
10  MOV AX,byte ptr VERIQ1+4
11  MOV byte ptr VERIQ2+3,AL
12  MOV byte ptr VERIQ2+2,AH
13  MOV AL,byte ptr VERIQ1+6
14  MOV byte ptr VERIQ2+1,AH
15  MOV byte ptr VERIQ2,AL
```

Kod çalışırsa `VERIQ1` nin içinde `0123456789ABCDEFh` oluşacaktır. Bellek haritasında gösterirsek

```
0000: EF CD AB 89 67 45 23 01
```

Döngünün ilk çevriminde bellekteki `EF` ve `CD` değerleri `AX` yazmacına alınıyor. Küçük uçtan yerleşim nedeniyle `AL`=`EF`, `AH`=`CD` oluyor. Bu yüzden `AL`'yi `VERIQ2+7`'ye, `AH`'yi ise `VERIQ2+6`'ya koyuyoruz. Böylece döngü sonunda bellekte

```
0008: 01 23 45 67 89 AB CD EF
```

ve `VERIQ2` nin içinde `EFCDAB8967452301h` oluşuyor.

4.3 Indeks ve Taban Adresli İşlenenler

Adresleme modları zengin olan 8086 da taban ve indeks adresleme modları iki boyutlu dizi elemanlarına adres hesaplamaksızın erişmeyi sağlar. Uzun tekrarlı kodların kısalmasına indeks ve taban adresli işlenenleri kadar LOOP komutunun da önemli bir yeri vardır.

4.3.1 LOOP komutu

Belirli sayıda tekrarlanması gereken kod parçalarını kolayca istenilen sayıda tekrarlayabilmek için tekrarlanacak bölümden önce `CX` yazmacına tekrarlama sayısı konur. Tekrarlanacak kodun başına da `L1` gibi

bir döngü girişi etiketi konur. Tekrarlancak kodun sonuna ise `LOOP L1` deyimi yerleştirilir.

LOOP komutu her çalışışında `CX` teki sayıyı bir azaltır ve daha sıfır olmadıysa `L1` e sapar, bir azaltılan `CX` sıfır olduysa `LOOP` komutundan sonraki komuta geçer.

Örnek 4.5.

Örnek 4.4 deki programda `qword` olarak tanımlı `VERIQ1` ve `VERIQ2` etiketli datalardan `VERIQ1` in baytlarını ters sırayla `VERIQ2` ye yerleştiren programı indeks adresli işlenenle yazalım.

Çözüm: İndeks adreslemede hedef için `DI`, kaynak için `SI` yazmacını kullanacağız. Döngü 8 kez tekrarlayacak, `DI` nın ilkdeğeri `VERIQ1`, ve `SI` nin ilkdeğeri `VERIQ2+7` olacak. Döngünün içinde bir bayt veriyi aktardıktan sonra `DI` yazmacını arttırıp `SI` yazmacını azaltacağız.

```
1   .MODEL SMALL
2   .DATA
3 VERIQ1 dq 0123456789ABCDEFh
4 VERIQ2 dq ?
5   .CODE
6   MOV AX,@DATA
7   MOV DS,AX
8   MOV CX, 8 ; 8 bayt taşıyoruz
9   MOV DI, offset VERIQ1
10  MOV SI, offset VERIQ2+7
11 L1:
12  MOV AL,[SI]
13  MOV [DI],AL
14  ADD DI,1
15  ADD SI,-1
16  LOOP L1
17  END
```

Toplama ve çıkarma işlemleri için henüz arttırma komutu `INC` i öğrenmediğimizden `ADD` komutu kullanmayı tercih ettik. Yazdığımız programları EMU8086 emulatör programını kullanarak deneyebiliriz.

4.4 Adresleme tiplerinin kodlamaya katkısı

Adresleme yönteminin program koduna katkısını sergilemek üzere veri bölütünde tanımlı bir dizi baytın toplamını bulmaya çalışalım.

Örnek 4.6.

Elimizde beş sayı var ve bunların toplamlarını TOPLAM değişkenine yazmak istiyoruz. Bu işi en hızlı biçimde yapmayı amaçlıyoruz.

Çözüm 8086 daki anlık işlenenleri kullanarak toplama ve saklama işlemini sadece 6 komutta yapabiliriz. Yürütmenin hızlı olması için AL yazmacını kullanırız ve TOPLAM değişkenini veri bölütü yerine kod bölütünde kullanırsak kodun başında DS yazmacına ilkdeğer vermek te gerekmez.

```
1 ; 5 sayının anlık toplamı
2  .model small
3   .code
4   mov  al,25h
5   add  al,12h
6   add  al,15h
7   add  al,1Fh
8   add  al,2Bh
9   mov  cs:toplam, al
10  mov ah,4dh ; DOS a donus
11  int 21h
12 toplam db ?
13  end
```

Program istediğimiz işi en hızlı biçimde yapıyor, amd ya ileride sayıları değiştirmek gerekse, sayılar programa dağılmış olduğundan bu işi bir programcıdan başkası yapamaz. Programımızı hiç olmazsa verileri veri bölütüne yerleştirecek şekilde yeniden yazalım.

Örnek 4.7.

Elimizdeki beş sayının toplamlarını TOPLAM değişkenine yazmak istiyoruz. Ancak programı ileride veri degişiklikleri kolay olacak biçimde yazmamız gerekiyor.

Çözüm 8086 daki doğrudan bellek adreslemeyi kullanarak bu işi kolayca yaparız. Gene AL yazmacını kullanırız. Bu kez TOPLAM ı da veri bölütüne koyacağız.

```
1 ; 5 sayının dogrudan toplamı
2  .model small
3   .data
4 sayı1 db 25h
5 sayı2 db 12h
6 sayı3 db 15h
```

```
7 sayı4 db 1Fh
8 sayı5 db 2Bh
9 toplam   db ?
10   .code
11   mov ax,@data
12   mov ds,ax
13   mov  al,sayı1
14   add  al,sayı2
15   add  al,sayı3
16   add  al,sayı4
17   add  al,sayı5
18   mov  toplam, al
19   mov ah,4dh ; DOSa dön
20   int 21h
21   end
```

Programın veri ve kodu ayırması verinin değişmesi durumunda iyi oldu. Ama kodu değiştirmedikçe sadece beş sayı toplayabiliriz. Programı daha genel amaçlı yazmak istersek bir döngünün başında toplamı ve kaç veri olduğunu ilkdeğerlendirip toplamları bir döngüde aldırmalıyız. Gene toplamı AL içinde almalıyız ki programımız hızlı çalışsın.

Örnek 4.8.

Elimizdeki beş sayının toplamlarını `toplam` değişkenine yazmak istiyoruz. Ancak programı ileride veri sayısı da değişebilecek biçimde yazalım.

Çözüm 8086 daki tabanlı bellek adreslemeyi kullanarak bu işi kolayca yaparız. Hızlı olsun diye gene `AL` yazmacını kullanırız.

```
1 ; 5 sayının tabanlı adresle toplamı
2  .model small
3   .data
4 sayısayısı equ 5
5 sayılar db 25h,12h,15h,1Fh,2Bh
6 toplam   db ?
7   .code
8   mov ax,@data
9   mov ds,ax
10 ; ilkdeger atama
11   mov  bx,offset sayılar
12   mov  al, 0     ; toplamı sıfırladık
13   mov  cx, sayısayısı
14 döngübaşı:
15   add  al,[bx]   ; gösterilen veriyi topla
```

```
16   inc  bx        ; sonraki veriyi göstersin
17   loop döngübaşı ; sayısayısı kez tekrarla
18   mov  toplam, al
19   mov ah,4dh     ; DOSa dön
20   int 21h
21   end
```

Programdaki döngü 5 kere çalışacak ve her dönüşte `inc bx` komutu nedeniyle `bx`i bir arttıracak. Böylece toplanacak adres bellekteki bir sonraki yer olacak.

Satır-3 te `equ` ile tanımlanan `sayısayısı` bir değişken değil, bir sabit değerdir. Bu yüzden

```
mov cx,sayısayısı
```

komutunda işlenen2 anlık işlenendir.

Kod geliştirmenin sonu yoktur. Örneğin daha da büyük sayıları toplayabilmek için toplamayı 16-bitlik yazmaçta yapabiliriz.

Örnek 4.9.

Dört büyük sayının toplamlarını indeksli adreslemeyle 16-bitlik `toplam` değişkenine yazmak istiyoruz.

Çözüm

```
1  ; 4 sayının dizinli adresle toplamı
2   .model small
3   .data
4  sayısayısı equ 4
5  sayılar dw 2512h,151Fh,-10h,-2
6  toplam   dw ?
7   .code
8   mov ax,@data
9   mov ds,ax
10 ; ilkdeger atama
11  mov  si,offset sayılar
12  mov  cx, sayısayısı
13  mov  ax, 0  ; toplam sifir
14 döngübaşı:
15  add  ax,[di]
16  inc  di
17  inc  di
18  loop döngübaşı
19  mov  toplam, ax
20  mov ah,4dh
21  int 21h
22  end
```

Sözcük (word) olarak tanımlı `sayılar` dizisinde her sayı iki bayt yer kapladığından bir sonraki sayıya adresi iki arttırarak ulaşabiliriz. Döngüde `inc di` komutunu iki kere kullanarak `di` yazmacını iki arttırmış olduk. Aynı iş `add di,2` ile de yapılır, ama `inc` komutu anlık işlenenli `add` komutuna göre daha hızlı olduğundan `inc` komutunu tercih ettik.

4.5 Sıçrama, Dallanma ve Adres Etiketleri

Örneğin bir döngü başı gibi ileride tekrar işleyeceğimiz kod bölümünün başlangıç adresine etiketleyerek erişebiliyoruz. Adres etiketleri koşulsuz sıçrama[1] yordam çağırma[2] ya da koşullu sıçrayarak dallanma gibi kullanım amacına bağlı olarak farklı tiplerde olabilir.

FAR (uzak) tipindeki adresler kod bölütünün dışındaki yerlerden gönderi alabilir. Modüller ve programlar arası çağrı ve sıçramalarda kullanılır. Hem bölüt hem de ofseti tutar.

NEAR (yakın) adresleme kod bölütü içindeki yerlerden gönderiye uygundur. Yalnızca ofseti tutar.

SHORT (kısa) adresleme 8-bitlik IP-göreceli adresleme yöntemiyle 127 bayt geriden 127 bayt ileriye kadar bir alanda erişim sağlar.

Etiketle gönderi arasındaki uzaklığı geriye olan adreslemeler için negatif, ileriye olan adreslemeler için pozitif olarak işaretliyoruz. Böylece SHORT tipi adres -127 ile 127 bayt kod aralığında geçerli oluyor.

4.5.1 Koşulsuz sıçrama JUMP, JMP

JMP her durumda program akışını gönderide kullanılan etikete yöneltir.

JMP short adresetiketi : *kısa* adres aralığına sıçrama yapar.

JMP adresetiketi : *dolaysız yakın* (direct near) adres aralığına sıçrar. Etiket hem kod bölütünde kalmalı hem de -32768 ile 32767 baytlık kod aralığında olmalıdır.

JMP [BX] : yazmaç dolaylı sıçramadır ve BX teki sayı IP göreceli adres olarak kullanılır.

JMP [DI] : Bellek dolaylı sıçramadır ve DI daki ofsete sıçrar.

JMP far ptr adresetiketi : Gönderide kullanılan etiket hem bölüt hem ofset içerir. Belleğin herhangi bir noktasına erişebilir.

[1] jump
[2] call

4.5.2 Koşullu Uzun Sıçrama

Koşullu sıçramalar bir karşılaştırma komutunun sonucuna, ya da doğrudan bayrak bitlerinin durumuna bağlı olarak verilen adrese sapar ya da sonraki komuta devam eder.

Bütün koşullu sıçramalar kısa adres tipini kullanır. 127 baytın ötesindeki etiketlere koşullu dallanma sağlamak için koşul ters çevrilip komutun devamına koşulsuz sıçrama komutu konur.

Diyelim ki sıfır bayrağı yükselmişse çok uzaktaki uzakadres etiketine sapmak istedik. Bu durumda

```
JZ uzakadres
```

komutu adres uzak olduğundan hata verecektir. Onun yerine

```
1    JNZ atla1
2    JMP uzakadres
3 atla1:
```

yazarak istenilen adrese aynı koşullu sapma sağlanmış olur.

4.5.3 Tek Bayrak Koşullu Sıçramalar

Tablo 4.1, koşulu tek bayrak olan sıçrama komutlarını özetler.

Tablo 4.1: Tek bayrak koşullu sıçramalar

Komut	Açıklama	Koşul	Tersi
JZ , JE	Sıfırsa sıçra	ZF = 1	JNZ, JNE
JC, JB, JNAE	Elde varsa	CF = 1	JNC, JNB, JAE
JS	işaret eksiyse	SF = 1	JNS
JO	Taşma varsa	OF = 1	JNO
JPE, JP	Eşlik varsa	PF = 1	JPO
JNZ , JNE	sıfır değilse	ZF = 0	JZ, JE
JNC , JNB, JAE	elde yoksa	CF = 0	JC, JB, JNAE
JNS	işaret artıysa	SF = 0	JS
JNO	taşma yoksa	OF = 0	JO
JPO, JNP	eşlik yoksa	PF = 0	JPE, JP

Örnek 4.10.

```
1   mov cx, döngüsayısı
2 döngübaşı:
```

```
3   ...(döngü gövdesi)
4   loop döngübaşı
5   ...
```

kodunda LOOP komutunun CX i bir azaltıp CX= 0 ise gönderi yapılan etikete sıçradığını biliyoruz. DEC CX komutu CX 'i bir azaltır ve sonuca göre sıfır bayrağını (Z'yi) tazeler. DEC CX komutunu kullanarak LOOP komutunun benzerini oluşturalım.

LOOP komutu sıçramayı CX sıfır olunca yaptığına göre DEC CX ten sonra JNZ adres kullanmamız gerekir.

```
1   mov cx, döngüsayısı
2 döngübaşı:
3   ...
4   dec cx
5   jnz döngübaşı
6   ...
```

4.5.4 İşaretli İşlem Koşullu Sıçramalar

İşaretli ikili sayıların karşılaştırma sonuçlarından koşul oluşturan bu sıçramaları kullanmak için gerekli bayrakları hazırlayabilmek için önce CMP komutuyla sayıları karşılaştırmak gerekir.
CMP işlenen1,işlenen2 aynı bir çıkarma komutu gibi çalışır, ancak çıkarmanın sonucunu bayraklardan başka hiçbir yazmaca kopyalamaz. İşaretli karşılaştırma koşulları Tablo 4.2 da verilmiştir.

Tablo 4.2: İşaretli karşılaştırma koşullu sıçramalar

Komut	Sıçrama için koşul	Bayrak koşulu	Tersi
JE, JZ	Eşitse (=). Sıfırsa	ZF = 1	JNE, JNZ
JNE, JNZ	Eşit değilse (≠). Sıfır değilse	ZF = 0	JE, JZ
JG, JNLE	Büyükse (>). Küçük eşit değilse (not ≤).	ZF = 0 ya da SF = OF	JNG, JLE
JL, JNGE	Küçükse (<) Büyük eşit değilse (not ≥)	SF ≠ OF	JNL, JGE
JGE, JNL	Büyük eşitse (≥). Küçük değilse (not <)	SF = OF	JNGE, JL
JLE, JNG	Küçük eşitse (≤) Büyük değilse	ZF = 1 ya da SF ≠ OF	JNLE, JG

Örnek 4.11.

AL yazmacındaki işaretli ikili sayı AH yazmacındakinden küçükse AH yi AL ile toplayıp bir ekleyelim.

Çözüm

```
1   ...
2   cmp AL,AH
3   jnl gerekmiyor
4   add AH,AL
5   inc AH
6 gerekmiyor:
7   ...
```

Görüldüğü gibi AH, AL den küçük değilse gerekmiyor etiketine sıçradık. Böylece tanımlanan işlemin yalnızca AH<AL ise yapılmasını sağladık.

4.5.5 İşaretli İşlem Koşullu Sıçramalar

İşaretli ve işaretsiz sayılarda büyüklük testi farklı bayraklara bakılarak yapılır. 8086 işaretsiz sayıları karşılaştırmak üzere farklı koşullu sıçrama komutları kullanır. Eğer sayılar işaretsiz sayıysa büyük küçük yerine Tablo da görüldüğü gibi komutlarda altında üstünde terimlerini kullanıyoruz.

Tablo 4.3: İşaretsiz karşılaştırmalı koşullu sıçramalar

Komut	Sıçrama için koşul	Bayrak	Tersi
JE, JZ	Eşitse (=). Sıfırsa	ZF = 1	JNE, JNZ
JNE, JNZ	Eşit değilse (<>). Sıfır değilse	ZF = 0	JE, JZ
JA, JNBE	Üstündeyse (>). Altında ya da eşit değilse (not <=).	CF = 0 ve ZF = 0	JNA, JBE
JB, JC JNAE,	Altındaysa (<); Üstünde yada eşit değilse (not >=)	CF = 1	JNB, JAE,
JAE, JNB,	Üstünde eşitse (>=). Altında değilse (not <)	CF = 0	JNAE, JB, JC
JBE, JNA	Altında eşitse (<=) Üstünde değilse	CF = 1 ya da ZF = 1	JNBE, JA

Örnek 4.12.

AL yazmacındaki işaretli ikili sayıyı AX yazmacına sıkışık BCD koduna dönüştürelim. Önce AH yi sıfırlayacağız. Döngüye girip AL>10 ise AH yi bir arttırıp döngüye devam edeceğiz. AL<=10 kalınca döngüden çıkacağız.

```
1   ...
2   mov ah, 0
3 döngübaşı:
4   cmp al,10
5   jle döngüçıkışı
6   inc ah
7   jmp döngübaşı
8 döngüçıkışı:
9   ...
```

4.5.6 Döngü Komutları ve DEC+JNZ den farkı

8086 zengin döngü komutlarına sahiptir. LOOP komutlarının en önemli özelliği döngü içinde güncellenmiş olan bayrak yazmacını etkilememesidir. Oysa dec cx komutu sıfır bayrağını değiştirir. Döngü sonundan güncellenen bayraklar döngü başında koşul olarak kullanılacaksa döngüyü loop kullanarak kurmak avantajlıdır.

Loop adresetiketi : her işlendiğinde cx i eksiltip eğer cx≠0 ise adresetiketi ile etiketlen koda sıçrar.

Loope, loopz : cx i eksiltir ve hem cx≠0 hem de ZF=1 ise gönderiye sıçrar.

Loopne, loopnz : cx i eksiltir ve hem cx≠0 hem de ZF=0 ise gönderiye sıçrar.

Loope ve loopne komutları ZF bayrağına bağlı döngülere zaman aşımı koymak için en uygun komutlardır.

4.5.7 Yordam Çağrı Komutları

Yordam[3] çağrı[4] komutları (CALL) yalnızca prosedür etiketlerine gönderi yapabilir. Yordam çağrı yürütülürken bir sonraki komutun adresi yığına itilir. Yordam, çağrıdan dönüş komutuyla bitince çağrıdan dönüş komutu yığındaki dönüş adresini yığından indirip çağrının yapıldığı programda kaldığı yere sıçrar.

CALL adresetiketi çeviricide yakın adres tipinde öntanımlıdır.

[3] procedure
[4] call

CALL far adresetiketi yürütülen bölütün dışında kalan yordamları çağırmak için kullanılır.

Örnek 4.13.

4.9 deki programı döngü ilkdeğerleme, DI ile toplama ve DOS a dönme yordamlarıyla yazacağız. Programın ilk hali

```
1 ; 4 sayının dizinli adresle toplamı
2  .model small
3   .data
4 sayısayısı equ 4
5 sayılar dw 2512h,151Fh,-10h,-2
6 toplam   dw ?
7   .code
8   mov ax,@data
9   mov ds,ax
10 ; ilkdeger atama
11   mov  si,offset sayılar
12   mov  cx, sayısayısı
13   mov  ax, 0  ; toplam sıfır
14 DonguBasi:
15   add  ax,[di]
16   inc  di
17   inc  di
18   loop DonguBasi
19   mov  toplam, ax
20   mov ah,4dh
21   int 21h
22   end
```

biçimindeydi. Satır 8 den 13 e kadarki deyimleri BASLAT adlı yordamda toplayacağız. Döngüyü ana programda bırakacağız, ama Satır 15 ten 17 ye kadarki deyimler ADDNDI adında bir yordamda duracak. Satır 20 ve 21 için de DOSADÖN yordamı yazacağız.

```
1 ;Yordam Cağırarak toplama
2     .model small
3     .stack 64
4     .data
5 sayılar dw 2340h,1de6h,3bc7h,566ah
6 toplam  dw  ?
7     .code
8 MAIN proc far
9     call BAŞLAT
10 topladöngü:
11     call ADDNDI
12     dec cx
```

```
13      jnz topladöngü
14      mov si,offset toplam
15      mov [si],bx
16      call DOSADÖN
17  main endp
18
19  BAŞLAT proc
20      mov ax,@data
21      mov ds,ax
22      mov cx,4
23      mov bx,0
24      mov di, offset sayılar
25      ret
26  BAŞLAT endp
27
28  ADDNDI proc
29      add bx,[di]
30      inc di
31      inc di
32      ret
33  ADDNDI endp
34
35  DOSADÖN proc
36      mov ah,4ch
37      int 21h
38      ret
39  DOSADÖN endp
40      end
```

Aynı programı yordamlardan başlayarak yazmak istersek `.code` girişine yordamları atlatmak üzere bir `jmp kodbaşı` komutu koymalıyız.

```
1  ;Yordamlar başta toplama
2      .model small
3      .stack 64
4      .data
5  sayılar dw 2340h,1de6h,3bc7h,566ah
6  toplam  dw  ?
7      .code
8  MAIN proc far
9      jmp kodbaşı
10 BAŞLAT proc
11     mov ax,@data
12     mov ds,ax
13     mov cx,4
14     mov bx,0
15     mov di, offset sayılar
16     ret
17 BAŞLAT endp
```

```
18
19 ADDNDI proc
20     add bx,[di]
21     inc di
22     inc di
23     ret
24 ADDNDI endp
25
26 DOSADÖN proc
27     mov ah,4ch
28     int 21h
29     ret
30 DOSADÖN endp
31
32 kodbaşı:
33    call BASLAT
34 topladongu:
35     call ADDNDI
36     dec cx
37     jnz topladongu
38     mov si,offset toplam
39     mov [si],bx
40     call DOSADÖN
41 main endp
42     end
```

4.6 EXE ve COM dosyası oluşturma

4.6.1 COM dosyaları

COM uzantılı dosyalar DOS un ilk yıllarında programları daima belleğin aynı bölgesine yükleyerek çalıştırma esasına dayanan yürütülebilir dosyalardır. En fazla 64kBayt olabilir. Kod bölütü, veri bölütüyle çakışık olan bu tip programlar DOS un yığıtını kullanır. Bu dosyalarda kod ve veri daima 0100 ofsetten başlar ve aynı programın COM derlemesi EXE sine göre daha kısadır.

TASM ile COM dosyası oluşturma örneği Ek A.7 de verilmiştir.

4.6.2 EXE dosyaları

EXE uzantılı çalıştırılabilir dosya biçimi daha geniş ana bellek olanaklarıyla birlikte 64kByte kod sınırlamasını kaldırmak üzere oluşturulmuştur. Kod büyüklüğünde sınır yoktur, ve kendi yığıtını tanımlamak zorundadır. Kod ve veri bölütleri istenen yerden başlayabilir. COM dosyasına sığabilen kodlar için EXE dosyası COM dan daha büyüktür.

4.7 Çok kullanılan komutlar

8086 zengin bir komut setine sahiptir. Ancak, amacımız bütün komutları detayına incelemek yerine 8086 donanım ve yazılımlarını tanıtmak olduğundan komutları tek tek incelemeyeceğiz. Mikroişlemcileri yeni öğrenmeye başlayanlar için bütün komutları ezberleyerek başlamak yerine bir süre için program yazarken komut özeti sayfalarını kullanmalarını öneriyoruz.

Ek de verilen komutlar pek çok çeşit program yazabilmek için yeterlidir. Burada komut anımsatıcılarını çok kısa açıklamalarla listeleyeceğiz. Listedeki kısaltmalar şunlardır.

İşlenenler için kısaltmalar: ai: anlık işlenen**; adr**: adres**; hi**: hedef**;**
ki: kaynak**;**
İşlenen tip ekleri: -8: 8-bitlik; **-16**: 16-bitlik;

4.7.1 İki işlenenli komutlar:

MOV hi, ki: hi $\leftarrow$ ki ( )
ADD hi, ki: hi $\leftarrow$ hi+ki, (CZSOPA)
ADC hi, ki: hi $\leftarrow$ hi+ki+CF, (CZSOPA)
AND hi, ki: hi $\leftarrow$ hi$\wedge$ki, (ZSP, CO$\leftarrow$ 0)
OR hi, ki: hi $\leftarrow$ hi$\vee$ki, (ZSP, CO$\leftarrow$ 0)
XOR hi, ki: hi $\leftarrow$ hi$\oplus$ki, (ZSP, CO$\leftarrow$ 0)
CMP ki$_1$, ki$_2$: ? $\leftarrow$ ki$_1$$-ki_2$, (CZSOPA)
TEST ki$_1$, ki$_2$: ? $\leftarrow$ki$_1$$\wedge$ki$_2$, (ZSP, CO$\leftarrow$ 0)

4.7.2 Adres İşlenenli Komutlar

CALL adr: yığıt$\leftarrow$(CS:IP); (CS:IP)$\leftarrow$adr, ( )
JMP adr: (CS:IP)$\leftarrow$adr, ( ).
Jxx adr: xx koşulu gerçekleşirse IP $\leftarrow$adr, ( )
xx yerini alacak bayraklı koşullar
C, NC, Z, NZ, S, NS, O, NO, P, NP
xx yerini alacak işaretsiz sayı koşulları
(yalnızca **CMP** sonrası kullanılır.)
A, B, E, AE, BE, NA, NB, NAE, NBE
xx yerini alacak işaretli sayı koşulları
(yalnızca **CMP** sonrası kullanılır.)
L, G, E, LE, GE, NL, NG, NBE, NGE
Koşul anahtarları: N:değil, **C**:elde, **Z**:sıfır, **S**:negatif**, O**:taşma,

P:eşlikli **E**:eşit, **A**:yukarı, **B**:aşağı, **L**:küçük, **G**:büyük anlamınadır.

LOOP adr: CX←CX−1, (CX≠ 0)⇒IP←adr, ( )
LOOPE adr: CX←CX−1, (CX≠ 0)∧(ZF≠ 0)⇒IP←adr, ( )

4.7.3 Tek işlenenli komutlar

INT a8: yazılım kesmesi[5]
yığıt←(CS:IP); (CS:IP)←M_{32}(4×a8), (I=1)
MUL ki8: işaretsiz AX←AL× ki8 (CO, ZSPA=?)
MUL ki16: işaretsiz DX;AX←AX× ki16 (CO, ZSPA=?)
IMUL ki8: işaretli AX←AL× ki8 (CO, ZSPA=?).
IMUL ki16: işaretli DX;AX←AX× ki16 (CO, ZSPA=?)..
DIV ki8: işaretsiz AL←AX/ki8 ; (COZSPA=?)
DIV ki16: işaretsiz AX←DX;AX/ki16 ;
DX←mod(DX;AX, ki16) (COZSPA=?)
IDIV ki8: işaretli AL←AX/ki8
AH←mod(AX, ki8) (COZSPA=?)
IDIV ki16: işaretli AX←DX;AX/ki16 ;
DX←mod(DX;AX, ki16) (COZSPA=?)
DEC hi: hi←hi−1 , (ZSOPA)
NOT hi: hi← $\overline{\texttt{hi}}$; ()
NEG hi: hi← $\overline{\texttt{hi}}$+1; (CZSOPA)

4.7.4 İşleneni olmayan komutlar

NOP: İş yapmaz, sadece zaman geçirir; ().
DAA: BCD toplama düzeltmesi; (CA).
DAS: BCD çıkarma düzeltmesi; (CA).
AAA: ASCII toplama düzeltmesi; (CA).
AAS: ASCII çıkarma düzeltmesi; (CA).
AAM: ASCII çarpma düzeltmesi; (ZSA=?).
AAD: ASCII bölme düzeltmesi; (ZSA=?).
CBW: AL deki işaretli sayıyı AX e uzatır; ().
CLC: CF ←0 , (C=0)
STC: CF ←1 , (C=1)
IRET: kesmeden geri dönüş.
(CS:IP)←yığıt, Bayrak yazmacı←yığıt.
RET: altyordamdan geri dönüş. (CS:IP) ←yığıt ; ( )

[5] interrupt

4.7.5 I/O Port Komutları

IN AL, a8: AL←port(pa8) ; ()
IN AL, DX: AL←port([DX]) ; ()
OUT a8, AL: port(pa8)←AL ; ()
OUT DX, AL: port([DX])←AL ; ()
I/O port komutlarının kullanılış biçimini ileriki bölümde göreceğiz.

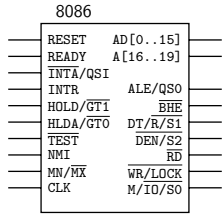
Bölüm 5

Bellek ve Giriş/Çıkış Arayüzü

8086 komut mimarisindeki giriş/çıkış (io) komutlarını anlayabilmemiz için 8086 bellek ve io yapısının genel hatlarından başlayalım.

5.1 ISIS 8086 Modeli

PROSIS® genel amaçlı bir devre tasarım ortamıdır, ve içinde yer alan ISIS® mikroişlemcileri çevresindeki program barındıran belleklerle ve analog ya da sayısal devre elemanlarıyla simule edebilir. 8086 modeli eğitim amaçlı kullanıma uygun oldukça gerçekçi bir görsel modeldir. Kullanım kolaylığı sağlamak ve devrenin aşırı karışmasını önlemek için modelin 8086 sı içinde 0x00800 adresinden başlayan bellekte barındırır. Veri yolları ekstra bellek ve io bağlantısı kurmaya elverişlidir.

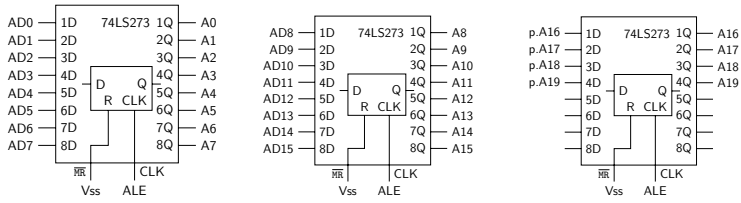


Şekil 5.1: ISIS 8086 Görsel Modeli Uç Bağlantıları

5.2 Adres ve Veri yolları

8086 Adres ve veri hatları AD0-AD15 uçlarından hem adres hem de veri için çoklanmış olarak kullanılır. AD0-AD15 hatları yalnızca ALE sinyalinin yüksek kaldığı süre boyunca geçerli adresi taşır. Bu sürede ya $\overline{RD}$ ya da $\overline{WR}$ hattı düşüktür. 74237 sekizli mandalları adresi bu anda mandallamak için uygundur. Şekil 5.2'de görüldüğü gibi birinci mandal yongası AD0-AD15 hatlarını ALE yüksekken geçirip düştüğü an mandallayarak sistemde kullanılan A0-A15'i üretiyor ikinci ve üçüncü

mandal yongalarının girişleri işlemciden gelen p.A16-p.A19 hatlarını alıyor ve çıkışları sistemde kullanılan A16-19 hatlarını üretiyor. Her üç yonganın da CLK girişi ALE ye bağlanarak adreslerin tam ALE'nin düşen kenarında mandallanması sağlanıyor. Mandalların sıfırlama girişi olan $\overline{\text{MR}}$ ise Vss=5V'a bağlanıyor. Böylece A0-A20 sistem adres yolu olarak kullanılırken AD0-AD15 ise veri okumak üzere belleklere ve io birimlerine bağlanıyor.



Şekil 5.2: 8086 için adres mandal devresi

ISIS'teki işlemci simulasyonunu 0x00800 adresinden başlayan 64kBayt belleğe sahip olarak düzenleyebiliyoruz. Bu durumda derlenecek makine kodunu veri, kod ve yığıt bölütleri tümüyle üstüste çakışacak ve kod 0x00800 den başlayacak biçimde düzenlemeliyiz. Örnek olarak elde ettiğimiz veri, adres ve denetim sinyallerini bir çıkış geçidi, iki giriş geçidi, ve bir USART evrensel seri asenkron yollayıcı alıcı[1] (USART) oluşturmakta kullanacağız.

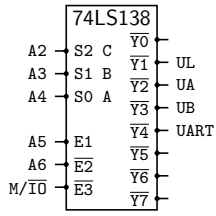
5.3 Sekiz Bitlik Adres Çözümleme

8086 nın 8-bit io adreslemeli komutlarını kullanacak en basit devre 8-bit io adres çözümlemesi yapan bir adres dekoder yongasından oluşur. 74HC138 3 - 8 dekoder yongası E1 girişi yüksek (H), $\overline{\text{E2}}$ ve $\overline{\text{E3}}$ girişleri düşük (L) ise C-B-A girişlerinin kodladığı çıkışı seçer. Y0-Y7 çıkışları tersleniktir ve seçilen çıkış L, seçilmeyenler H olur. Devrenin bağlantıları Şekil 5.3 te görülüyor.

Dekoderin çalışma biçimi Tablo 5.1 de, ve bu seçme sinyallerine karşılık gelen adreslerin özetini Tablo 5.4de veriyoruz.

Tabloda görüldüğü gibi örneğin UL çıkışını seçebilmek için işlemci io komutu yürütürken A6,A4 ve A3 bitlerini sıfır, A5 ve A3 bitlerini

[1] universal serial asynchronous receiver transmitter



Şekil 5.3: 8-bit io adres çözümleme devre bağlantıları

Tablo 5.1: 74HC138 dekoderin 8-bit için çalışma tablosu

M/IO	A6	A5	A4	A3	A2	⇐ veriyolu	
$\overline{E3}$	$\overline{E2}$	E1	C	B	A	$\overline{Y0}$...$\overline{Y7}$	Seçilen Çıkış
X	X	0	X	X	X	HHHHHHHH	hiçbiri
0	0	1	0	0	0	LHHHHHHH	$\overline{Y0}$ (boşta)
0	0	1	0	0	1	HLHHHHHH	$\overline{Y1}$ Çıkış Portu UL
0	0	1	0	1	0	HHLHHHHH	$\overline{Y2}$ Giriş Portu UA
0	0	1	0	1	1	HHHLHHHH	$\overline{Y3}$ Giriş Portu UB
0	0	1	1	0	0	HHHHLHHH	$\overline{Y4}$ USART
0	0	1	1	0	1	HHHHHLHH	$\overline{Y5}$ boşta
0	0	1	1	1	0	HHHHHHLH	$\overline{Y6}$ boşta
0	0	1	1	1	1	HHHHHHHL	$\overline{Y7}$ boşta
X	1	X	X	X	X	HHHHHHHH	hiçbiri
1	X	X	X	X	X	HHHHHHHH	hiçbiri

bir yollayacak bir adrese erişiyor olmalıdır. Bu koşulları 24h ile 27h arasındaki adres değerleri sağlamaktadır.

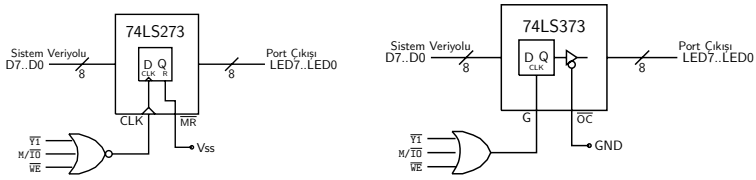
Tablo 5.2: 8-bit io adres çözümleme tablosu

M/IO	A6	A5	A4	A3	A2	⇐ veriyolu /⇓ Adres	Port
0	0	1	0	0	1	24h - 27h	$\overline{Y1}$ UL
0	0	1	0	1	0	28h - 2Bh	$\overline{Y2}$ UA
0	0	1	0	1	1	2Ch - 2Fh	$\overline{Y3}$ UB
0	0	1	1	0	0	30h - 33h	$\overline{Y4}$ USART

5.4 Çıkış Portu Devresi

Şekil 5.4 deki 74273 ya da 74373 mandal devresiyle komutun işlemciden yollattığı veriyi tutacak olsak bu veriyi mandal çıkışından kullanabiliriz. 74273 veriyi CLK sinyalinin yükselen kenarında tutar ve bu nedenle ($\overline{Y1}$ veya $\overline{WR}$)ın yükselen kenarında veri kararlıyken veriyi mandala alır. İkinci örnek devredeki 74373 ise $\overline{LE}$=1 olduğu sürece veriyi çıkışa geçirir, $\overline{LE}$ =0 iken çıkıştaki veriyi dondurur. ($\overline{Y1}$ veya $\overline{WR}$

veya M/$\overline{IO}$) iken $\overline{LE}$ =1 sağlayabilmek üzere ($\overline{Y1}$ veya $\overline{WR}$ veya M/$\overline{IO}$) iken kullanılıyor.



Şekil 5.4: Çıkış portu devreleri

İşlemci

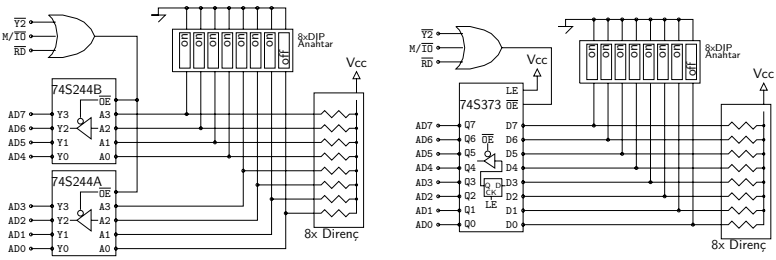
```
1    MOV AL,03h
2    OUT 24h,AL
```

komutlarını çalıştırırken OUT komutu sırasında işlemci adres hatlarına A=24h, veri hatlarına AL nin içeriği olan 03h, kontrol hatlarına ise M/$\overline{IO}$ ve $\overline{WR}$ sıfır yollar. Bütün bu sinyaller biraraya geldiğinde M/$\overline{IO}$=0 ve A=24h gidince 74138 adres çözücü $\overline{Y1}$:UL çıkışını sıfır yapar. Hem $\overline{Y1}$=0 hem de $\overline{WR}$=0 olunca çıkış portu mandalının CLK ya da $\overline{LE}$ girişi gereken sinyali alır ve veri yolundaki 03h değerini port çıkışına geçirir. Böylece Q7-Q0 uçlarından 03h çıkar. Uçlara bağlı LED dizisi ve akım sınırlayıcı direnç paketi her bir uç sıfır olduğunda uca bağlı LED yanacak biçimde düzenlendiğinden 03h ile üstteki iki LED söner, diğer altı LED yanar.

Çıkış portu mandalının görevi OUT komutuyla gelen ve veri yolunda bir mikrosaniyeden de az kalan veriyi OUT komutundan sonra da Q7-Q0 çıkışlarında saklamaktır. Çıkıştaki 03h, yeni bir OUT 24h,AL komutuna kadar orada kalır.

5.5 Giriş Portu Devresi

Adres 28h deki UA giriş portunun her bir sinyali porttan okunan sayının bir bitini oluşturur. Port girişindeki sinyallerden oluşan sayıyı okuyabilmek için girişleri tam gereken anda veri yoluna bağlayabileceğimiz üçdurumlu sekizli tampon gerekir. Sekizli tamponu değişik yollarla elde edebiliriz. Örneklerimizde (i) dual dörtlü üçdurumlu tampon yongası 74244 ile, ve (ii) sekizli mandal 74373 yongasını LE girişini H



Şekil 5.5: Giriş Portu devreleri

bağlayıp OE ucuyla girişleri çıkışlara bağlatan iki çeşit sekizli tampon kullandık. Her iki devre de şekil 5.5'de görülüyor

Giriş portundan okuma yapmak üzere

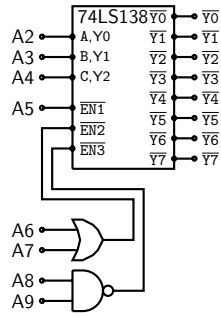
```
IN AL,28h
```

komutunu kullanırız. İşlemci bu komutu işlerken io komutu olduğundan adres yolundan A=28h yollar, kontrol yolundan ise M/$\overline{\text{IO}}$=0 ve $\overline{\text{RD}}$=0 verir ve AL yazmacına aktarmak üzere veri yolundan gelecek veriyi bekler. Adres çözümleyici A=28h ve M/$\overline{\text{IO}}$=0 olduğu için dekoderin $\overline{\text{Y2}}$ çıkışını sıfır yapar. Veya kapısı sayesinde hem $\overline{\text{RD}}$=0 hem de $\overline{\text{Y2}}$=0 olunca tamponun $\overline{\text{OE}}$ girişi sıfır olur ve girişe bağlı olan anahtarların oluşturduğu giriş bitleri AD0-AD7 veri yoluna iletilir. İşlemci bu veriyi okuyup AL yazmacına yazar. Böylece daha sonraki komutlarda girişteki anahtarlarını durumuyla ilgili işlemler yapabiliriz.

5.6 On Bitlik Adres Çözümleme

Anlık 8-bit adreslemeli port sayısı 256 yı geçemez. Oysa 8086 daha geniş bir io adres uzayında dolaylı adresleme kullanabilecek biçimde tasarlanmıştır. Örneğin IBM PC ilk tasarımında 10-bit io adres çözümlemesi kullanmıştır. 10 bitlik bir adres çözümleme devresinin şematik bağlantıları Şekil 5.6 te görülüyor.

E1 girişi A5 adres sinyaline bağlandığından devre ancak A6 H olduğunda bir çıkış seçecektir. $\overline{\text{E2}}$ girişi (A6 | A7) sonucu sıfırsa L olacağından bir çıkışın seçilmesi için hem A6 hem A7 L olmalıdır. $(\overline{\text{A8}\,\&\,\text{A9}})$ çıkışına bağlı $\overline{\text{E3}}$ girişi ise ancak hem A8 hem A9 H olduğunda L olarak dekoderin çıkışına izin verecektir. Sonuç olarak dekoderin çalışma biçimi Tablo 5.1 de, ve bu seçme sinyallerine karşılık gelen adreslerin



Şekil 5.6: 16-bit Adres Çözümleme devresi

Tablo 5.3: 74HC138 dekoderin 8-bit için çalışma tablosu

$\overline{E3}$		$\overline{E2}$		E1	C	B	A	⇐ Girişler	
A9	A8	A7	A6	A5	A4	A3	A2	⇐ veriyolu/⇓ $\overline{Y0}$...$\overline{Y7}$	Seçilen Birim
X	X	X	X	0	X	X	X	HHHHHHHH	hiçbiri
X	X	X	1	X	X	X	X	HHHHHHHH	hiçbiri
X	X	X	1	X	X	X	X	HHHHHHHH	hiçbiri
0	X	X	1	X	X	X	X	HHHHHHHH	hiçbiri
X	0	X	1	X	X	X	X	HHHHHHHH	hiçbiri
1	1	0	0	1	0	0	0	HLHHHHHH	$\overline{Y0}$ boşta
1	1	0	0	1	0	0	1	HLHHHHHH	$\overline{Y1}$ Çıkış Portu UL
1	1	0	0	1	0	1	0	HHLHHHHH	$\overline{Y2}$ Giriş Portu UA
1	1	0	0	1	0	1	1	HHHLHHHH	$\overline{Y3}$ Giriş Portu UB
1	1	0	0	1	1	0	0	HHHHLHHH	$\overline{Y4}$ USART
1	1	0	0	1	1	0	1	HHHHHLHH	$\overline{Y5}$ boşta
1	1	0	0	1	1	1	0	HHHHHHLH	$\overline{Y6}$ boşta
1	1	0	0	1	1	1	1	HHHHHHHL	$\overline{Y7}$ boşta
1	1	X	1	X	X	X	X	HHHHHHHH	hiçbiri

özetini Tablo 5.4de veriyoruz.

Bu adres çözümleme devresiyle örneğin `UL` çıkış portu `324H` de başlıyor. 16-bitlik adreslere ancak `DX` ile erişilir. Bu porta `03h` sayısını göndermek üzere işlemci

```
1  MOV DX,324h
2  MOV AL,03h
3  OUT DX,AL
```

komutlarını işleyebilir. İşlemcinin porta göndereceği 03h veri yolunda mikrosaniyeden daha kısa bir süre için kalır. Veri yoluna bağlayacağımız Şekil 5.4'deki 8-bitlik bir mandal devresiyle `03h` yi çıkış portu mandalında tutarsak bir sonraki `OUT` komutuna kadar mandal çıkışında `03h` görünmeye devam eder.

Adres çözümleyici `328h-32Bh` aralığındaki adreslerde $\overline{Y2}$ çıkışını `L` yapar. Adres `328h` deki giriş portunun her bir sinyalinden oluşan sayıyı

Tablo 5.4: 16-bit Adres çözümleme çizelgesi

A9	A8	A7	A6	A5	A4	A3	A2	⇐ veriyolu/⇓ Adres	Seçilen Birim
1	1	0	0	1	0	0	1	324h-327h	$\overline{Y1}$ Çıkış Portu UL
1	1	0	0	1	0	1	0	328h-32Bh	$\overline{Y2}$ Giriş Portu UA
1	1	0	0	1	0	1	1	32Ch-32Fh	$\overline{Y3}$ Giriş Portu UB
1	1	0	0	1	1	0	0	330h-333h	$\overline{Y4}$ USART

okuyabilmek için Şekil 5.5'deki gibi bir giriş portu tamponu gerekir. Giriş tamponunun girişlerindeki sayısal durumu `AL` yazmacına kopyalamak için işlemcinin

```
1    MOV DX,328h
2    IN AL,DX
```

komutlarını yürütmesi gerekir.

5.6.1 IBM-PC'nin giriş/çıkış adres tablosu

IBM-PC, giriş çikiş portlarının düzenlenişi açısından tipik bir örnektir. Aynı zamanda ISA-veriyoluna fazladan port bağlantısı yapmak üzere PC'deki giriş çıkış adresleri önem taşır. IBM-PC giriş/çıkış birimlerini 16-bitlik tam adres çözümlemesi ile etkinleştirse de portları yalnızca A9 ... A0 bitlerini kullanacak biçimde düzenler. Önemli birimlerin port adresleri Tablo 5.5'de görülmektedir.

000-01F	DMA 8237A Doğrudan Bellek Erişimi
020-03F	8259A, Master Kesme Denetleci-1
040-05F	8254-2 Zaman sayacı (Timer)
060-06F	8042 klavye denetleci (keyboard contr)
070-07F	NMI Gerçek-Zaman saat maskesi
...	.
278-27F	Paralel Yazıcı Port-2
2E1	GPIB adaptörü 0
2E2,2E3	Veri Toplama adaptörü (Data Acquisition Ad.) 0
2F8-2FF	Seri Port 2
300-31F	Prototip Kart
360-363	PC Ağ, Düşük-Adres
364-367	Ayrılmış
368-36B	PC Ağ, Yüksek-Adres
36C-36F	Ayrılmış
378-37F	Paralel Yazıcı Port-1

Tablo 5.5: IBM-PC giriş çıkış port adresleri.

Örneğin, port 61h deki klavye denetlecinin ürettiği saat vurularını `b4` bitinden okumak mümkündür. Bu saat vuruları sistemin işlemci frekansından bağımsız olarak her zaman 15.085 mikrosaniyede bir evrilir

ve böylece tam 30.17 μs periyodlu bir kare dalga oluşturur. Sistem frekansından bağımsız bir süre bekleme elde etmek için klavye saat vurularını saymak aşağıdaki altyordam ile mümkündür.

```
1  W15us proc
2  ; cx kere 15 mikrosan. bekle
3          pushf
4          push ax
5          push dx
6          mov dx,61h
7          in al,dx
8          and al,10h
9          jnz W15B1
10 W15B0:
11         in al,dx
12         and al,10h
13         jz W15B0
14         loop W15B1
15         jmp W15C0
16 W15B1:
17         in al,dx
18         and al,10h
19         jnz W15B1
20         loop W15B0
21 W15C0:
22         pop dx
23         pop ax
24         popf
25         ret
26 W15us endp
```

Bu altyordamı diyelim ki `CX=3` ile çağırdık. Satır 3-5 `dx`, `ax` ve bayrakları yığıta saklar. Satır 6-8 port-61h'in bit-4'ündeki olan saat bitini test eder. Satır-9 saat biti `1` ise `W15B1`'e sapar. Satır 10-13 saat biti `0` olduğu sürece döner, bit `1` olunca döngüden çıkar. Satır 14'deki loop, `CX`'i bir azaltır ve eğer `CX` sıfır olmamışsa saat biti bir olduğu sürece dönmek üzere `W15B1`'e sapar. Satır-14'teki loop komutunda `CX`=0 olduysa işlem sırası `W15C0`'a atlayarak çağıran programa dönüşü başlatır. Program, `W15B1`'den başlayan 16-19. satırlarda saat biti `1` olduğu sürece döner. Saat 0 olup döngüden çıkınca 20. satırdaki loop komutu `CX`'i bir azaltıp `CX` sıfır değilse başa, `W15B0` bölümüne saatin `0` olduğu süreyi beklemeye sapar. Başlangıçta `CX=3` ile çağırdığımız için saat üç kere sayılıp `CX` sıfırlanınca satır 21-24'teki pop komutlarıyla yığıta saklanan yazmaç değerleri dx ax ve bayrak yazmaçlarına geri konulur ve satır-25'teki `ret` ile geri çağiran koda dönülür.

Örneğin 100ms beklemek için 100ms= 100000μs olduğuna göre `CX`'e 100000 /15.085=6629 koyarak `W15us`'yi çağırabiliriz. Bir çağırışta en uzun 65535 *15.085μs=0.988 saniye bekleyebildiğinden örneğin 20 saniye için aşağıdaki altprogramı `CX`'e 200 koyarak çağırabiliriz.

```
1 W100ms proc
2 ; cx kere 100 milisan. bekle
3 W100msA
4    push CX
5    mov CX,6629
```

```
6     call W15us
7     loop W100msA
8     pop CX
9     ret
10 W100ms endp
```

İlerideki program kodlarımızda işlemciyi istediğimiz süre bekletmek üzere bu iki altyordamdan birini kullanacağız.

5.7 Bellek yapısı ve düzeni

20 bitlik adres uzayına sahip olan 8086 toplam 1MBayt bellek kullanabilir. Veri yolu 8-bit olan 8088, yalnızca tek bir bellek bankından oluşan 1MBayt belleği kullanabilir. Veri yolu 16-bit olan 8086'nın bellek düzeni sıfır dahil çift adreslere erişen bank-0 ile tek adreslere erişen bank-1 olmak üzere iki banktan oluşur.

5.7.1 Reset olayı

İşlemci güç kaynağına ilk bağlandığında bir *power-on* reset devresi işlemciyi başlangıç durumuna getirir. Başlangıç durumunda `CS=FFFFh`, ve `IP=0000h` olduğundan ilk işleyeceği komutun fiziksel adresi `FFFF0h` olur. Bu adres reset vektörü olarak adlandırılır ve genellikle buraya BOOT ROM'un başındaki komutları işlemeye gitmesi için bir uzak sıçrama komutu (`jump`) konur.

IBM-PC tasarımında *bios* ve *boot-rom* için adres `C0000h` dan `FFFFFh` e kadarki bellek bölümü ayrılır. `A0000` ile `BFFFF` arası grafik ekrana ayrılan video bellek bölgesidir. `00000h` ile `9FFFFh` arasındaki 640 kBayt işletim sistemiyle kullanıcıya ayrılan bellektir.

5.7.2 Uzatılmış ve Genişletilmiş Bellek Bölgeleri

Bunun dışında uzatılmış bellek[2] sistemi (EMS) denilen yöntemle 8 MBayt'a varan geniş bir belleğin 16 kBayt bloklar halinde `A0000h-F0000h` bellek aralığına anahtarlanmasına dayanan bir yöntem geliştirilmiştir.

IBM-AT den başlayarak 80386 ve 80486 işlemciye dayanan tasarımlarda bu işlemcilerin korunmalı kiplerinde 4 GBayt adres uzayı kullanabilmesi sayesinde `100000h` ve ötesindeki bölgeye genişletilmiş bellek[3] olarak adlandırılmıştır.

[2] Expanded Memory
[3] Extended Memory

5.7.3 Kesme Yapısı

Küçük bellek adreslerindeki en önemli sınır kesme vektörlerine ayrılmış olan 00000h ile 003FFh aralığıdır. DOS bu bölgeden sonraki alana yerleşir. Kesmelerin bir bölümü işlemciden kaynaklanır. Örneğin INT 00h kesmesi sıfıra bölme, INT 01h tek adım yürütme, INT 02h maskelenemeyen kesme, INT 03h durma noktası, INT 04h işaretli sayı taşmasına ayrılmıştır. Örneğin INT 02h oluşunca işlemci önce yürütülen komutu tamamlar. Ardından IR, CS ve IP yazmaçlarını yığıta yerleştirir ve 02h×4= 00008h adresindeki ilk iki sözcüğü IP ve CS e yükleyerek kesme servisine başlar. Bir kesme daima IRET (kesmeden dönüş[4]) komutuyla sonlanır. IRET komutu yığıta yüklenmiş duran IP, CS ve FR yazmaçlarının eski değerlerini yığıttan indirir ve IP, CS, FR yazmaçlarına koyar. Böylece işlemci kesme geldiğinde kaldığı yerden programını yürütmeye devam eder.

5.7.4 BIOS, DOS, ve Kullanıcı Kodları

00400h ile 007FFh aralığı DOS ve BIOS için ayrılmış özel değişkenlere aittir ve program kodu için kullanılamaz. 00800h adresinden başlayarak önce en temel DOS programları olmak üzere gerekli yürütülebilir programlar belleğe yerleşmiştir. Bunların ardından *Bitince Kod Bırakan*[5] (TSR) programlar gelir. DOS bellek işleticisi bellekteki programların yerlerini ve boş bölgeleri kayıt altında tutar. Gerektiğinde yüklenecek programlar ve veriler için kullanıcılara bellek bölgesi ayırır.

[4] interrupt return, IRET
[5] Terminate-Stay-Resident

Bölüm 6

DOS Servisleri, Aritmetik ve Metin İşleme

8086 mimarisini ve üzerine geliştirilen IBM-PC BIOS sistemini en verimli kullanan işletim sistemlerinden biri olan MS-DOS kullanıcıya sunduğu geniş bir servis yelpazesi ile özellikle kullanıcının giriş/çıkış, disk erişimi ve haberleşme sorunlarına çeşitli çözümler sağlar. Bu bölümde aritmetik mantık ve metin işleme komutlarının yanı sıra BIOS ve DOS servislerini çevirici programların içinde kullanır düzeye gelmeyi hedefliyoruz

6.1 BIOS ve DOS servisleri

Yazılım kesmesi komutu olan `INT anlıkdeğer` komutu işletim sistemine özel hızlı ve esnek yapılı bir çağrı sistemi oluşturarak kullanıcının giriş/çıkış eylemlerini kolaylaştırmayı amaçlar. IBM-PC tasarımında iki önemli servis seviyesi bulunur.

En altta olan servis donanıma ilişkin giriş/çıkış parametre ve işleçlerini içeren BIOS düzeyidir. Başlama rom'unda yer alan bu servis kodları işletim sistemi yüklenmese bile çağırılabilir. BIOS servisleri PC tasarımlarındaki donanım farklarına rağmen işletim sisteminin PC yi standart bir ortam gibi görmesini sağlar.

PC nin işletim sistemi olan DOS içinde daha üst düzeyde giriş/çıkış ve disk erişim servisleri vardır. Bunlar çeviricilerin yüksek düzey dilleri makine koduna dönüştürürken giriş/çıkış işlemlerini standartlaştırır ve kolaylaştırır. Biz bu altbölüm içinde hem BIOS hem de DOS servislerinin en önemli birkaç tanesini göstereceğiz.

6.1.1 INT 10 BIOS Servisleri

BIOS servislerinden örnekleyeceğimiz grup monitör ve klavye kullanımına ilişkin olacak. Servisler 10h kesmesi ile çağırılıyor, ve AH yazmacındaki servis numarasına bağlı olarak çeşitli io işleri yapabiliyor. Servisleri topluca Tablo 6.1'de görüyoruz.

Tablo 6.1: Kesme 10h ye bağlı io servisleri

AH=00h	**Video modu kurma**	
	AL=03h	CGA metin modu
	AL=04h	320x200 4 renk grafik modu
	AL=06h	640x200 Hi-res siyah beyaz
	AL=07	monokrom metin modu
AH=02h	**Yazı yeri kurma**	
	çağrıda:	DH=satır, DL=kolon, BH=sayfa
AH=03h	**Yazı yeri okuma**	
	dönüşte:	DH=satır, DL=kolon, BH=sayfa
		CX= yazıyeri tipi
AH=06h	**Ekran temizleme**	
	AL	sayfa numarası:
	BH	temizleme niteliği
	CH;CL	sol-üst satır ve sütun numarası
	DH;DL	sağ-alt satır ve sütün numarası
AH=0Ch	**Siyah-Beyaz grafik ekrana piksel koyma**	
	AL	0:Siyah, 1:Beyaz
	CX;DX	Kolon ve Satır koordinatı

Örnek 6.1.

Sayfa-0 görünürken ekranın ikinci satır dördüncü sütunundan sekizinci satır yirminci sütununa kadarki alanı beyazlatmak istiyoruz. Gerekli parametreleri yükleyerek bios servis kesmesini çalıştıralım

Çözüm: Karakter niteliklerini boyayarak ekran temizleyen 10h kesmesindeki AH=06h servisini çağıracağız. Servisin parametreleri: sayfa numarası AL=0; sol üst köşe satır ve sütun numaraları CH=2, CL=4; sağ alt köşe satır ve sütun numaraları DH=8, DL=20; ve bir de doldurulacak karakter niteliği BH= 07h kullanmalıyız.

```
1  ...
2  MOV AX,0600h; AL=00h,AH=06h
3  MOV CH,2
4  MOV CL,4
5  MOV DH,8
6  MOV DL,20
7  MOV BH,07h
8  INT 10h
9  ...
```

Örnek 6.2.

Yazdıracağımız sayıların ekranda ikinci satır dördüncü sütundan başlamasını istiyoruz. İlgili BIOS servisiyle gereken kodu yazalım.

Çözüm: 10h kesmesindeki AH=02 servisi yazı yerini istenen satır ve sütuna ayarlar.

```
1  ; yazı yerini 2.satır 4.sütuna ayarlama
2  MOV AH,02h ; servis numarası
3  MOV BH,0   ; sayfa-0 da çalışıyoruz
4  MOV DH,2   ; satır
5  MOV DL,4   ; sütun
6  INT 10h
7  ...
```

6.1.2 INT 21 DOS Servisleri

INT21 servisleri DOS işletim sisteminin ekran, klavye ve disk erişimini kolaylaştıran pek çok servisini barındırır. Bunlar arasından bizim örneklediğimiz bir kaç tanesi Tablo da görülüyor. DOS seviyesinde klavyeden alınan ve ekrana gönderilen karakter ve metinler Şekil 1.1 deki gibi ASCII kodludur. Boşluk karakteri kodu 20h, rakamlar 30h ... 39h, harfler 41h ... 5Ah ve 61h ... 7Ah aralıklarındadır. Satır başı 0Ah, yeni satır 0Dh ile kodlanır.

Tablo 6.2: Kesme 21h ye bağlı klavye/ekran servisleri

AH=01h	**Klavyeden bir tuş bekle ve ekranda göster.**
	Dönüşte: AL girilen karakteri getirir.
AH=02h	**Bir karakteri ekranda göster**
	Çağrıda: DL: yazılacak ASCII karakter.
AH=09h	**Bir metni ekranda göster**
	Çağrıda: DX: '$' karakteriyle sonlanan metnin ofset adresi
AH=0Ah	**Klavyeden bir metin getir.**
	Çağrıda: DX: metin tamponu ofseti Dönüşte Satır-sonu ile sonlanan metin tampona yazılmış olarak döner. Tampon: {tampon uzunluğu, metin uzunluğu, metin içerik}
AH=4Ch	Program kodunu bellekten sil ve DOS a dön

Bunlar arasında AH=0Ah klavyeden metin getirme servisi veri bölütünde bir tampon oluşturlmasını gerektirir. Bu tampon üç öğeden

oluşur. En baştaki baytta tamponun toplam karakter sayısı uzunluğu, ikinci baytta tamponda saklı metinin karakter sayısı uzunluğu, ardındaki baytlarda ise tamponun gövdesini oluşturan karakterler zinciri olmalıdır. Örneğin

```
06 00 20 20 20 20 20 20
```

6 karakterlik boş bir tampondur. Bu 6 karakterlik tamponun içine `X=1<cr>` konulduğunda içeriği

```
06 03 58 3D 31 0D 20 20
```

olur. Burada `<cr>` sembolü, ASCII kodu `0Dh` olan satır-gir (ya da satırbaşı) tuşunu gösteriyor.

Örnek 6.3.

`'E'` tuşuna basınca ekranda 2.satır 8. kolona `Hello World.`, diğer bütün tuşlar için `Selam Millet.` yazan bir program yazalım.

Çözüm:

```
1    .model small
2    .data
3  HW db ' Hello World.$'
4  SM db ' Selam Millet.$'
5    .code
6  main proc far
7    mov AX,@data
8    mov DS,AX
9  ;ekran imlecini 2.satır 8.kolon a taşıyalım
10   mov AH,02 ; bu servis int.10 dadır
11   mov BH,0  ; 0. sayfada
12   mov DH,2  ; 2. satır
13   mov DL,8  ; 8. kolon
14   int 10h   ; Cursor istenen yere gitti
15 ;klavyeden bir karakter bekleyelim
16   mov AH,01h
17   int 21h
18 ; tuşa basıldığında dönen karakter AL de olacak
19   cmp AL,'E' ; acaba karakter 'E' mi?
20   je  Hello  ; E ise Hello yazmaya git
21 ;E değilse Selam yazacağız
22   mov AH,09h ; bir metin yazdırma servisi
23   mov DX,offset SM ; selamın ofseti
24   int 21h
25 ; dosa doneceğiz
26   jmp DOSa
27 Hello:
28   mov AH,09h ; metin yazdırma servisi
29   mov DX,offset HW ; hellonun ofseti
30   int 21h
```

```
31 DOSa:
32    mov AH,4Ch
33    int 21h
34 main endp
35    end main
```

Bu son örnekte yer almayan (INT 21h, AH=02) servisinin işlevini ilerideki örneklerde kullanacağız.

6.1.3 INT 16h DOS Klavye Servisleri

Daha önce `INT 21h` içinde klavyeden bir tuş basılıncaya dek bekleyen servisi tanımış ve bir örnekte de kullanmıştık. DOS Klavye işletiminde içine 16 karakter sığan bir çembersel tampon bulunur. İşletim sistemi herhangi bir anda basılan klavye tuşlarının kodlarını bu tampona koyar. 16 hanesi de dolarsa DOS tampona sığmayan her karakteri bip sesiyle kullanıcıya duyurur. `INT 16h` klavye tamponunda basılı tuşları kullanabilmek üzere oluşturulmuştur.

Tablo 6.3: Kesme 16h ye bağlı io servisleri

AH=01h	**Klavye durumunu döndür**	
	Dönüşte:	ZF: 0- bekleyen tuş var 1- bekleyen tuş yok AH: Son tuşun tarama kodu AL: tuşun ASCII kodu Tampon imlecini güncellemez.
AH=00h	**Klavye tamponundaki en eski tuşu tampondan al ve döndür**	
	Dönüşte:	AH: tuşun tarama kodu AL: tuşun ASCII kodu Tampondan bir tuş indirir.

Örnek 6.4.

ASCII karakter kodları arasında `07h` = <BEL> bip sesi; `08h` =<BS> geriye git; `09h` =<TAB> ; `0Ah` =<LF> Yeni satıra geç; `0Ch` =<FF> Yeni sayfaya geç;; `0Dh`=<cr> satır gir (ya da satır başı), gibi kontrol kodları da yer alır. Bu örnekte yazacağımız program başlayınca ekrana 'durdurmak için boşluğa basın!!' yazacak 500 tekrarlık bir döngüde ekrana '-' koyup bip sesi çıkaracak. Her bipten sonra klavyeden basılan son karaktere bakacak, eğer boşluk karakteri basıldıysa DOS a dönerek sona erecek.

Çözüm:

```
    .model small
    .stack 100h
    .data
msg  db ' durdurmak için'
     db ' boşluğa basın.$'
dashbell db '-',07h,'$'
    .code
main proc far
   mov ax,@data
   mov ds, ax
;önce mesajı yazalım
   mov ah,09h
   mov dx,offset msg
   int 21h
; beşyüzlük döngüye hazırlan
   mov cx,500
tekrarla:
   dec cx  ; en çok 500 kez
   jz  DOSadön
; bip sesi ve tire işareti
   mov ah,09h
   mov dx,offset dashbell
   int 21h
; tampon boş mu
   mov ah,01h
   int 16h
   jz tekrarla
; tamponda boşluk varmı?
   mov ah,00h
   int 16h
   cmp AL,20h
   jnz tekrarla
; boşluk varmış, DOSa dön
DOSadön:
   mov ah,4Ch
   int 21h
main endp
   end main
```

6.2 Aritmetik Mantık Komutları

8086'nın ALU komutları yalnızca 8 ve 16 bitlik olduğundan daha uzun sayıların aritmetik işleminde eldenin bir sonraki işleme taşınması gerekir.

6.2.1 Uzun sayıları elde bayraklı toplama

Aşağıdaki örnekte `00789654h` ile `00678543h` sayılarının toplamını `00E01B97h` olarak elde edeceğiz. Bu sayılar 16-bit yazmaçlara sığamadığından toplamayı iki parçada yapacağız.

Örnek 6.5.

```
    .data
uzunbirsayı    DD 789654h
ikinciuzunsayı DD 678543h
toplamuzunsayı DD ?
    .code
MAIN:
    MOV AX,@data
    MOV DS,AX
```

```
;uzunbirsayının en sağdaki wordunu
;  ikinciuzunsayının en sağ worduyle
;     toplayıp toplamuzunsayıya koyalım.
   MOV AX, word ptr uzunbirsayı
   ADD AX, word ptr ikinciuzunsayı
   MOV word ptr toplamuzunsayı, AX
;geri kalan wordleri elde bitiyle toplayalım
   MOV AX, word ptr uzunbirsayı+2
   ADC AX, word ptr ikinciuzunsayı+2
   MOV word ptr toplamuzunsayı+2, AX
;Bu sayıları debug kullanmadan görebilsek ne iyi olurdu.
   MOV AH,4Ch
   INT 21h
   END MAIN
```

Programda satır-12 deki `word ptr` niteleyicisi `uzunbirsayı` etiketini word tipine dönüştürür. `word ptr` kullanmasaydık çevirici `AX` e 32-bitlik double-word değerin sığmadığı hatasıyla duracaktı.
`uzunbirsayı` etiketi dört bayt (=2 word) uzunlukta bir alanı temsil ediyor. Ancak etiket gerçekte bu alanın yalnızca ilk baytının adresini gösterir. Bu yüzden `word ptr uzunbirsayı` double-word sayının içinde `7654h` bulunan sağ word'üdür. Satır-13 teki `add` işlemiyle `AX`=`7654h`+`6543h` =`1B97h` ve `CF`=1 olur. Elde bayrağının yüksek değerli wordlerle toplanması gerekir. Bu amaçla satır-17 de toplamayı `ADD` yerine `ADC` kullanarak yaparız.

Uzunsayı toplamayı dört-wordlük sayılarla yapacak olsak aynı kodu dört kez tekrarlayacağız. Dört kez tekrarlamayı indeksli adresleme kullanarak döngü içinde yapabiliriz.

```
   TITLE uzunsayı toplamı
   .MODEL SMALL
   .STACK 64
   .DATA
DATA1 DQ 548B9963CE7H
   ORG  0010H
DATA2 DQ 3FCD4FA23B8DH
   ORG  0020H
DATA3 DQ        ?
   .CODE
MAIN PROC FAR
   MOV AX,@DATA
   MOV DS,AX
   CLC  ; elde bayrağını temizle
   MOV SI,OFFSET DATA1  ;SI data1 i göstersin
   MOV DI,OFFSET DATA2  ;DI data2 yi göstersin
   MOV bx,OFFSET DATA3  ;BX toplamı göstersin
```

```
18    MOV CX,04             ; dört kere tekrarla
19 GENE:
20    MOV AX,[SI]   ;data1 den bir wordu AX e koy
21    ADC AX,[DI]   ;data2 den bir wordu üstüne topla
22    MOV [BX],AX   ;sonucu SUM a koy
23    INC SI        ;data1 in sonraki wordünü göstersin
24    INC SI
25    INC DI        ;data2 in sonraki wordünü göstersin
26    INC DI
27    INC BX        ;data3 ün sonraki wordünü göstersin
28    INC BX
29    LOOP GENE     ; dört tur döngü çalıştır
30    MOV AH,4CH
31    INT 21H           ; DOSa dön
32 MAIN ENDP
33    END MAIN
```

Alıştırma: Bir program yazarak { -37, 126, 112, 15, 35, 41, 90 } işaretli sayıların toplamını bulup 16-bitlik `SUM` değişkenine yazınız.

Alıştırma: İşaretsiz 27354, 28521, 29553, 41280, 60606 sayılarının 4-bayt tutabilecek toplamını 32-bitlik `SUM` değişkenine yazınız.

6.2.2 32-bitlik uzun çıkarmalar

Elde bayrağı, uzun sayılarla çıkarma yaparken borç bayrağı olarak kullanılır.

```
1    TITLE uzunsayı çıkarma
2    .MODEL SMALL
3    .STACK 64
4    .DATA
5 data_a dd 62562FAh
6 data_b dd 412963Bh
7 result   dd ?
8      . . .
9   mov ax, word ptr data_a   ; ax <- LSW of data_a
10  sub ax, word ptr data_b   ; subtract LSW of data_b
11  mov word ptr result, ax   ;  ax <- LSW of result.
12  mov ax, word ptr data_a+2 ; ax <- MSW of data_a
13  sbb ax, word ptr data_b+2 ;  subtract with borrow
14  mov word ptr result+2, ax ;  ax <- MSW of result.      .     . . .
```

İlk 16-basamaktaki 62FAh−963Bh işleminde alınan borç basamak, satır-13 teki `sbb` komutu ile bir sonraki 16-basamaklık çıkarmada 625h−412h−(eldeki borç) işlemi biçiminde hesaba dahil edilir.

6.2.3 Farklı genişlikteki sayılarla toplama

Toplanacak ya da çıkarılacak sayılardan biri 16-bit diğeri 8-bit olursa önce 8-bit sayının 16-bite genişletilmesi gerekir. Sayının işaretli ya da işaretsiz olmasına bağlı olarak farklı genişletme işlemi gerekir. 8086 nın bu amaçla kullanılan `CBW` ve `CWD` komutları genişletme işleminde kolaylık sağlar.

Örnek 6.6.

```
1 ; işaretsiz küçük tamsayıları toplayacağız
2 ;  ancak sonuç uzun bir tamsayı olacak.
3   .model small
4   .data
5 sayısayısı equ 4
6 küçüksayılar   DB 180, 225, 211, 192
7 toplam  DW ?
8    .code
9 MAIN:
10   MOV AX,@data
11   MOV DS,AX
12 ;döngüye hazırlık yapalım
13   XOR DX,DX ; toplamı burada tutuyoruz
14   XOR AH,AH ; işaretsiz genişletme için gerekir
15   MOV CX,sayısayısı
16   MOV BX,offset küçüksayılar
17 toplamadöngüsü
18   MOV AL, [BX] ; küçük sayı tek bayt uzunlukta
19   ADD DX,AX    ; AH=0 AL yi 16 bite genişletti
20   inc BX       ; Bir sonraki küçüksayının ofseti
21   loop toplamadöngüsü
22 ; sonucu toplam a koymalıyız
23   MOV toplam, DX
24 ; DOSa dönüş
25   MOV AH,4Ch
26   INT 21h
27   END MAIN
```

Örnek 6.7.

```
1 ; Şimdi de işaretli küçük tamsayıları toplayalım
2 ;  ancak sonuç uzun bir tamsayı olacak.
3   .model small
4   .data
5 sayısayısı equ 4
6 küçüksayılar   DB -180, 225, 211, 192
7 toplam  DW ?
```

```
8     .code
9  MAIN:
10    MOV AX,@data
11    MOV DS,AX
12 ;döngüye hazırlık
13    XOR DX,DX ; toplamı burada tutuyoruz
14    MOV CX,sayısayısı
15    MOV BX,offset küçüksayılar
16 toplamadöngüsü
17    MOV AL, [BX] ; küçük sayı tek bayt uzunlukta
18    CBW          ; işaretli AL yi AX e genişlettik
19    ADD DX,AX
20    inc BX       ; Bir sonraki küçüksayının ofseti
21    loop toplamadöngüsü
22 ; sonucu toplam a koymalıyız
23    MOV toplam, DX
24 ; DOSa dönüş
25    MOV AH,4Ch
26    INT 21h
27    END MAIN
```

Son olarak bir de 16-bit işaretli sayıları 32-bit (dword) sayı olarak toplayalım.

Örnek 6.8.

```
1  ; Şimdi de 16-bit işaretli tamsayıları topluyoruz
2  ;  Sonuç 32 bit işaretli tamsayı olacak.
3    .model small
4    .data
5  sayısayısı equ 4
6  sayılar   DW -1881, 22055
7    DW 21012, 19291
8  toplam  DD ?
9     .code
10 MAIN:
11    MOV AX,@data
12    MOV DS,AX
13 ;döngüye hazırlık
14    XOR DI,DI ; toplam için
15          XOR SI,SI ; toplam-sol tarafı
16    MOV CX,sayısayısı
17    MOV BX,offset sayılar
18 toplamadöngüsü
19    MOV AX, [BX] ; sayı iki bayt
20 ; işaretli AX i DX:AX e genişlet
21    CWD
22 ; Toplama işlemi iki bölüm
```

```
23    ADD AX, [DI]
24          ADC DX,[SI]
25 ; iki bayt ileri
26    inc BX
27    inc BX
28    loop toplamadöngüsü
29 ; sonucu toplam'a yaz
30          MOV word ptr toplam, AX
31    MOV word ptr toplam+2,DX
32 ; DOSa dönüş
33    MOV AH,4Ch
34    INT 21h
35    END MAIN
```

6.2.4 Çarpma ve Bölme

İşaretli ve işaretsiz tamsayı çarpma ve bölme komutları hem işlemcinin hesap hızını arttırır hem de kodlamanın da kısalmasını sağlar. Toplama ve çıkarmada sonuç işlenenlerin uzunluğunda olmasına karşın çarpma ve bölmede kurallar oldukça farklıdır. 8bit x 8bit tamsayı çarpımın sonucu 16 bite uzanır. 16bit x 16 bit çarpımda ise sonuç 32 bit olur. Bu nedenle çarpım işlemine giren bellek ve yazmaçları iki işlenenle tanımlamak kullanışsızlaşır. Intel bu durumda sıradışı uzunluktaki işlenenler için sabit yazmaç grupları kullanmayı seçmiştir. Örneğin 8 bit çarpım daima `AL` ile bir yazmaç ya da bellek arasında olur, ve sonuç daima `AX` e konur. Böylece çarpım komutu tek işlenenli komutlar arasında yer alır.

Örnek 6.9.

Şekil 5.3, 5.5, ve 5.4 de gördüğümüz io portlarını hatırlayalım. Devremizde io adresi `A=24h` de bir çıkış portuyla io adresi `A=28h` de bir giriş portu bulunmaktaydı. Bu örnekte giriş portuna bağlı bir sayısal seviye ölçme cihazı bulunduğunu, ve kalibrasyonu sırasında 0 metre seviyede 30, 10 m seviyede 100 okunduğunu varsayalım. Öyle bir program yazacağız ki çıkış portundan metreler doğru olarak okunabilsin.

Çözüm:

```
1     .model small
2 ; çıkış portu 24h; giriş portu 28h;
3 ;  girişteki sensor 0m de 30, 10m de 100 yolluyor.
4 ;  çıkıştan metreler doğru okunacak.
5 ;   çıkış= (sensorokuması-30)*7/100.
6 ;
7 ; data gerekmiyor. DS'i tanımlamayacağız.
```

```
8     .code
9  MAIN:
10    IN AL,24h   ; sensörü AL ye okuduk
11    SUB AL,30   ; 30 eksilttik. negatif istemiyoruz
12    JL  negatifdeğil
13 ; negatifse AL=0 olsun
14    MOV AL,0
15 negatifdeğil:
16    MOV CL,10   ; 10 ile çarpacağız
17    MUL CL      ; çarpım sonucu AX te duruyor
18    MOV CL,70   ; yetmişe böleceğiz
19    DIV CL      ; sonuç AL yazmacında
20 ; AL deki sonucu çıkış portuna aktaralım
21    OUT 24h,AL
22 ; bu işi tekrar tekrar yaptıracağız.
23 ; böylece seviye değişince metreyi hemen
24 ; gösterecek
25    JMP MAIN
26 MAIN ENDP
27   END MAIN
```

Görüldüğü gibi `mul` komutu anlık işlenen kabul etmediğinden sayıyı `CL` ye koyarak kullandık.

Örnek 6.10.

Klavyeden bir tuş bekleyelim. Gelen tuşun ASCII kodunu onluk olarak ekrana yazalım. Bunu boşluk karakteri gelene dek sürdürelim.

Çözüm:

Tuş bekleme servisi kullanırsak tuş ekrana çıkar. INT 16 servisi ile tuş basılıncaya dek bir döngüde kalıp basılan tuşun ASCII kodunu alacağız. ASCII kodu onluk olarak yazabilmek için kodu 100 e bölüp sonuca '0' toplayarak ekrana yollayacağız. Sonra kalanı 10 a bölüp sonucuna '0' toplayarak ekrana yollayacağız. Son olarak kalana '0' ekleyerek ekrana yollayacağız.

Örneğin 'a' harfi basılırsa ASCII kodu 61h=97 dir.
97/100=0, kalanı 97 dir. 0+'0' = '0' ekrana yaz.
Kalan 97 yi 10 a böl. 97/10=9, kalan 7. 9+'0'='9' ekrana.
Kalana '0' ekleyince 7+'0'='7' ekrana.
Böylece ekranda '097' göreceğiz.

```
1     .model small
2  ; klavyeden basılan tuşun ASCII değerini gösterir
3    .code
4  MAIN PROC FAR
5  ; tampona karakter gelene dek bekle
```

```
6 bekle:
7    MOV AH,01h
8    INT 16h      ;
9    JZ  MAIN
10   MOV AH,00h   ; karakteri oku
11   INT 16h
12 ; karakterin ASCII kodu AL de duruyor
13   MOV AH,0     ; kodu 16-bite genişlettik
14   MOV CL,100   ; yüze böleceğiz
15   DIV CL       ; kalan AH de, sonuç AL de
16 ; AX teki sonucu bir yere saklayalım
17   PUSH AX      ; AX i yığıta koyuyoruz.
18 ; AL deki karakteri ekrana gönder.
19   MOV AH,02h   ; INT21h karakter gösterme servisi
20   ADD AL,'0'   ; ASCII kodlama
21   MOV DX,AL    ; basılacak karakter
22   INT 21h      ; ekrana bas.
23 ; AX teki sonucu geri yükleyelim
24   POP AX
25 ; Kalanı AL ye aktarıp AH yi sıfırlayalım
26   MOV AL,AH
27   XOR AH,AH
28 ; Ona bölelim
29   MOV CL,10
30   DIV CL      ; AH kalan, AL sonuç
31   ADD AH,'0'
32   ADD AL,'0'
33 ; AH karakter bastırırken gerekiyor.
34 ;  içindeki veriyi DH ye yere aktaralım
35   MOV DX,AX   ; DL <-AL ve DH <-AH oldu
36   MOV AH,02h ; karakteri bas
37   INT 21h
38   MOV DL,DH   ; kalan+'0' DH teydi
39   MOV AH,02h ; aslında AH de zaten 02h vardı
40   INT 21h
41 ; Bir de sayıları ayıran boşluk basalım
42   MOV DL,' '
43   MOV AH,02h ; gerekmezdi ama yazdık
44   INT 21h
45 ; şimdi herşeyi tekrarlayabiliriz
46   JMP bekle
47 MAIN ENDP
48   END MAIN
```

6.3 Mantık komutları

Mantık komutları bit bite AND, OR, XOR işlemlerini gerçekleştirir. Hedef işlenenin her bir biti kaynak işlenenin aynı hizadaki bitiyle işleme girer ve sonuç hedef işlenene yazılır. işlemin sonucuna göre Z,S,P bayraklarını güncellerken O ve C bayraklarını sıfırlar.

AND hedef, kaynak komutu kaynağın sıfır bitlerine karşılık gelen hedef bitlerini sıfırlamak için kullanılır. Örneğin AL yazmacındaki bit-3 ve bit-5 i sıfırlamak için AND AL,11010111b komutunu kullanırız.

OR hedef, kaynak komutu kaynağın bir olan bitlerine karşılık gelen hedef bitlerini bir yapar. Örneğin AL yazmacının en sağdaki dört-bitini bir yapmak üzere OR AL,0Fh komutunu kullanırız.

XOR hedefi, kaynak komutuyla kaynağın bir olan bitleri karşılık gelen hedef bitlerini tersler. Örneğin AL yazmacının en soldaki üç bitini terslemek için XOR AL,11100000b} komutunu kullanırız.

Örnek 6.11.

ASCII tabloda büyük harflerin koduyla ile küçük harflerin kodu arasındaki tek fark bütün büyük harflerin bit-5 inin sıfır,karşılık gelen küçük harfin bit-5 inin ise bir olmasıdır. Örneğin
'A' = 41h = 01**0**00001b = 65,
'a' = 61h = 01**1**00001b = 97
gibi. Küçük harfleri büyük harfe dönüştürmek için bit-5 i sıfırlamak yeterlidir

Yazacağımız program klavyeden satırsonuyla sonlanan en fazla 20 karakterlik bir metin bekleyecek, ve gelen metini büyük harfe dönüştürüp ekranda gösterecektir.

```
1     .model small
2  ; klavyeden gelen satırı
3  ;  büyük harfe dönüştürüp
4  ; ekranda göstereceğiz.
5     .data
6  ; klavye metini tamponları
7  tampon db 20
8  metinboyu  db 0
9  gelenmetin db 20 dup(20h)
10 bharfmetin db 20 dup(20h)
11    .code
12 MAIN PROC FAR
13    mov AX,@DATA
14    mov DX,AX
15 ; klavyeden metin getir
16    mov AH,0Ah
17    mov DX,offset tampon
18    int 21h
19 ; metin uzunluğu kadar dön
20    mov cx,metinboyu
21 ; hedef
22    mov di,offset bharfmetin
23 ; kaynak
24    mov si,offset gelenmetin
25 tekrarla:
26    mov al,[si]
27 ; al<'a' ise gerek yok
28    cmp al,'a'
```

```
29    jb gerekyok
30 ; al>'z' ise de gerek yok
31    cmp al,'z'
32    ja gerekyok
33 ; bit-5'i sıfırla
34    and al,11011111b
35 gerekyok:
36    mov [di],al
37 ; indeksleri bir sonraki
38 ;    harflere hazırla.
39    inc di
40    inc si
41    loop tekrarla
42 ; satırın sonuna '$' koy
43    mov [di],'$'
44 ; satırı ekrana yolla
45    mov al,09h
46    mov dx,offset bharfmetin
47    int 21h
48 ; dos a dönüş
49    mov al,4ch
50    int 21h
51 MAIN ENDP
52   END MAIN
```

6.4 BCD ve ASCII işlemler

BCD sayılar her 4 ya da her 8 bite bir ondalık basamak yazılarak elde edilir. en küçük basamak en başta[1] yazılır. ASCII metindeki sayılar küçük basamakla başlamaz. Bu yüzden bayt sırasının değiştirilmesi gerekir. Burada örnek olarak sıkışık BCD sayılarla toplama ve sayma, bir de ASCII sayılarla toplama ve sayma yapacağız.

Ascii sayıları sağa dayayarak toplama

```
1  ; ASCII sayılarla işlem
2     .model small
3  ; klavyeden gelen ilk sayıyı
4  ; ikinciye ekle
5  ; ve sonucu ekranda göster
6     .data
7  ; klavye metini tamponları
8  tb equ 10   ;tamponboyu
9  satır1 db tb, 0, tb dup(20h)
10 satır2 db tb, 0, tb dup(20h)
11 sonuç  db tb dup(20h),'$'
12    .code
13 MAIN PROC FAR
14    mov AX,@DATA
15    mov DX,AX
16 ; birinci sayıyı getir
17    mov AH,0Ah
18    mov DX, offset satır1
19    int 21h
20 ; ikinci sayıyı getir
21    mov AH,0Ah
22    mov DX, offset satır2
23    int 21h
24 ; sayıları sağa daya
25 ;1234------ ==>  ------1234
26 ;  s1+10 <- s1+n
27    mov bx, offset satır1
28    call sağadaya
29    mov bx, offset satır2
30    call sağadaya
31 ; toplama döngüsü
32    mov bx,offset sonuç+tb+1
33    mov di,offset sayı1+tb+1
34    mov si,offset sayı2+tb+1
35    mov cx,tboyu-1
36    clc   ; elde bayrağı sıfır
37 toplamadöngüsü:
38    mov al,[si]
39    adc al,[di]
40    aaa
41    pushf        ; eldeyi sakla
42    or al,30h    ; ascii kod
```

[1] little endian

```
43    mov [bx],al ; sonucu yaz
44    dec si
45    dec di
46    dec bx
47    popf  ; eldeyi geri yükle
48    loop toplamadöngüsü
49 ; sonucu basalım
50    mov ah,09h
51    mov dx,sonuç
52    int 21h
53 ; dos a dönelim
54    mov al,4ch
55    int 21h
56 MAIN ENDP
```

Yukarıdaki koda aşağıdaki `sağadaya` altyordamını da eklemeliyiz.

```
57 sağadaya proc
58    mov si, bx
59    mov di, bx
60    xor ah, ah
61    mov al, [bx]
62    add di, ax
63    inc bx
64    mov al, [bx]
65    add si, ax
66    inc bx
67 tekrarkopya:
68    cmp di,bx
69    jb kopyabitti
70    mov al,20h
71    cmp si,bx
72    jb boşlukkalsın
73    mov al,[si]
74 boşlukkalsın:
75    mov [di],al
76    dec di
77    dec si
78    jmp tekrarkopya
79 kopyabitti:
80    ret
81 sağadaya endp
82
83  END MAIN
```

Örnek 6.12.

ASCII sayıları dizine bakarak toplama

```
1  ; ASCII sayı toplama işlemi
2     .model small
3  ; klavyeden gelen ilk sayıyı
4  ; ikinciye ekleyip  ekrana yaz
5     .data
6  ; klavye metin tamponları
7  satır1 db 10, 0, 10 dup(20h)
8  satır2 db 10, 0, 10 dup(20h)
9  sonuç   db 10 dup(20h),'$'
10    .code
11 MAIN PROC FAR
12    mov AX,@DATA
13    mov DX,AX
14 ; birinci sayıyı getir
15    mov AH,0Ah
16    mov DX, offset satır1
17    int 21h
18 ; ikinci sayıyı getir
19    mov AH,0Ah
20    mov DX, offset satır2
21    int 21h
22 ; sayıların son basamağını
23 ;   dizine koy.
24    mov ah,0
25    mov al,satır1+1
26    mov di,offset satır1
27    add di,ax
28    mov al,satır2+1
29    mov si,offset satır2
30    add si,ax
31    mov bx,offset sonuç+10
32    clc      ; eldeyi sıfırla
33    pushf    ; ve yığıta koy
34 toplama.döngüsü:
```

```
; sayılar biterse 0 kullan
   mov ax,0
;  cmp di,offset satır1+2
   jb di.bitti
   mov al,[di]
di.bitti:
   cmp si,offset satır2+2
   jb si.bitti
   mov ah,[si]
si.bitti:
;toplamayı yap ve sakla
   popf
   add al,ah
   aaa
   pushf
   cmp ax,0
   je toplama.bitti
;sayıyı yerine koyalım
   mov [bx],al
;dizinleri güncelleyelim
   dec bx
   dec di
   dec si
;bir sonraki basamağı topla
   jmp toplama.döngüsü
toplama.bitti:
; satırı ekrana yolla
   mov al,09h
   mov dx,offset sonuç
   int 21h
; dos a dönüş
   mov al,4ch
   int 21h
MAIN ENDP
  END MAIN
```

Örnek 6.13.

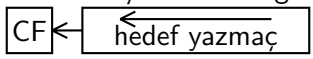
ASCII sayıyı arttırma

```
; ASCII sayıları arttırma
   .model small
; girilen metindeki 'a' ları sayalım.
satır  db 40, 0, 40 dup(20h)
sayı   db 5 dup(20h),'$'\ignore{$}
   .code
MAIN PROC FAR
   mov AX,@DATA
   mov DX,AX
; bir satır metin getir
   mov AH,0Ah
   mov DX, offset satır
   int 21h
; döngü hazırlığı
   mov cl,satır+1 ; satırın uzunluğu
   xor ch,ch      ; 16 bite genişlet
   mov di,offset satır+2 ; metnin başı
tekrarla
   cmp [di],'a'
   jne işlemegerekyok
   ; sayıyı arttır
      push di
      mov bx, 10
      mov di,offset sayı
sonrakibasamaklar:
         mov al,[di+bx]
```

```
27          adc al,0
28          aaa
29          mov [di+bx],al
30          dec bx
31          jnc artmabitti
32          cmp bx,0
33          jg  sonrakibasamaklar
34        artmabitti:
35        pop di
36  işlemegerekyok:
37        loop tekrarla
38  ; sayıyı yazdır
39     mov ah,09h
40     mov dx, offset sayı
41     int 21h
42     mov ah,4ch
43     int 21h
44  MAIN ENDP
45    END MAIN
```

6.5 Dönme ve Kaydırma Komutları

SHL hedef, 1 komutu hedef bellek ya da yazmacı bir bit sola kaydırır. Boşalan bite 0 girer.



Diğer bütün dönme ve kaydırma komutları gibi SHL ikinci işleneni işlemin kaç kez tekrarlanacağını belirtir ve çevirici tarafından yeterince **SHL hedef,1** komutu kullanılarak gerçekleştirilir. İkinci işlenen CL yazmacı da olabilir. Bu durumda kaydırma (CL MOD 32) kez tekrarlanır. Sola bir bit kaydırılan sayı 2 ile çarpılmış olur. Sayı işaretli ise işaret biti değiştiğinde işlemci O taşma bayrağına bir koyar.

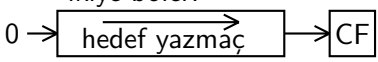
Örnek 6.14.

AL deki işaretsiz sayıyı 4 ile çarpalım. Bunun için 2-bit kaydıracağız

```
1    ...
2    shl al,2
3    ...
```

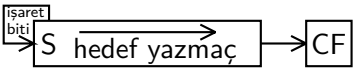
işlemi `MUL` ile yapsak `cl` ye 4 koyacak ve `mul` diyecektik `AL`=00010110b=22_{10} olsa bu kaydırma işleminden sonra `AL`=01011000=88_{10} olur.

SHR hedef; 1 komutu bitleri bir bit sağa kaydırarak işaretsiz sayıları ikiye böler.



Sayı işaretliyse ancak pozitifse sonuç doğru çıkar. Negatif sayının işaret biti bozulur ve bu durumda O bayrağı 1 olur. İşaretli sayıları ikiye bölmek için SAR aritmetik sağa kaydırma komutu kullanılır.

SAR hedef, 1 komutu işareti bozmadan bitleri sağa kaydırarak işaretli sayıları ikiye böler.



İşaretli sayıları bölmekte kullanılan SAR gibi işaretli sayıları çarpmak üzere de SAL tanımlanmıştır. Ancak işaretli sayı ile işaretsiz sayı çarpımı tümüyle aynı olduğundan SAL ile SHL de aynı komutlardır. Gene de işaretli çarpmalarda SAL kullanmak dökümantasyonu kolaylaştırır.

Örnek 6.15.

`AL` deki işaretli sayıyı 4 e bölelim. Bunun için `AL`yi 2-bit sağa kaydıracağız.

```
1    ...
2    sar al,2
3    ...
```

Aynı işlemi `IDIV` ile yapsak 'CBW' ile `AL`yi `AH` ye uzatacak, `CL` ye 4 koyacak ve `IDIV` kullanacaktık.

Bölüm 7

Makro ve moduler programlama

İnsan tabiatı sonucu zorlanmadan akılda tutulabilecek birim sayısı aşağı yukarı on civarındadır. Aynı zamanda, uzun ve karmaşık birimleri gerektiğinde içeriğini değiştirebilecek derecede anlayabilmek için mümkün olduğunca birbirinden bağımsız alt birimlerine ayrıştırmak zorundadır. İyi program yazma pratiği bize bu altbirimlendirmeyi daha programı yazarken yapmayı önerir. Oluşturulan altbirimler programı kolay anlamanın ve yazmanın yanısıra düzensiz tekrarlı işlerde yazılan kodların tekrar kullanılmasına da olanak sağlar.

Bu bölümde bir assembler makrosunun tanımlanışını öğrenip uygulayacak ve makroların doğru kullanımıyla avantajlarını göreceğiz. Ayrıca `include` direktifi aracılığıyla programımıza bir makro kolleksiyonu eklemeyi öğreneceğiz.

7.1 Makronun Avantajları

Çevirici dillerin çoğu gibi bizim örnek olarak kullandığımız Turbo Assembler ve EMU8085 te kullanılan Flat Assembler çeviricilerinde de makro yazma olanağı vardır.

Makrolar özellikle ekran temizleme, ekrana mesaj yazma, klavyeden bir satır yazı alma, uzun sayıları toplama, sayıları ikili onlu BCD veya ASCII gibi çeşitli biçimlere çevirme gibi çok sık ve defalarca kullanılan kod bloklarını kısa ve kolayca anlaşılır biçimde program koduna eklemek için kullanılır.

Makro bir kod bloğuna uygun bir sembolik isim atanmasını sağlar. Böylece makronun sembolik adını yazınca karşılığı olan kod da programın o noktasına kopyalanmış olur.

Makronun adı olarak kullanılan sembolik isim makronun yaptığı işleri kolayca hatırlamamızı sağlayarak programı anlamakta da yardımcı olur.

7.2 Makronun tanımlanması

Makro tanımı bir etiket ve `macro` anahtar sözcüğünün ardına yazılan bir dizi sembolik argüman ile başlar ve `endm` ile sonlanır.

```
makro_adı macro <arguman1><,arguman2> ...
   ... <makro içindeki komutlar>
   endm
```

Tanımlanan makroyu kullanmak için makronun adını sanki bir komutmuş gibi, parametrelere geçecek etiketleri ise işlenenler olarak yazarız.

```
   makro_adı arguman1,arguman2
```

Aşağıdaki örnekte ekrana bir metin yazdıracak ve program sonunda bizi DOS a döndürecek iki makro tanımlıyoruz. İlk makronun adı **EkranaYaz**, parametresi ise **veri** olsun. İkincinin adı **DOSaDön** ve parametresi yok.

Örnek 7.1.

```
   model small
   stack 64

EkranaYaz macro veri
   mov dx,offset veri
   mov ah,09h
   int 21h
   endm

DOSaDön macro
   mov ah,4ch
   int 21h
   endm

   .data
merhaba db 'Merhaba Millet..$'
   .code
main proc far
   mov ax,@data
   mov ds,ax
   ;makroları kullanabiliriz
   EkranaYaz merhaba
   DOSaDön
main endp
   end
```

Bu makrolar programın neresinde tanımlanırsa tanımlansın program aynı makine kodunu üretir ve aynı şekilde çalışır. Makrolar genellikle ya program başında ya da sonunda tanımlanır. Yalnızca kullanılan makrolar bir tek kullanıldığı zaman koda girdiğinden çok uzun makro kolleksiyonları içeren dosyalar programın başında include komutu ile program dosyasına dahil edilerek kullanılır. Bu çeşit kullanımı ileride örneklerle göreceğiz.

7.3 Makro ile Altprogramın farkı

Prosedür, altyordam ya da altprogram olarak adlandırılan yapı dönüş adresini yığıtta saklar ve kod bölütünde tanımlanır. Daha önce de öğrendiğimiz gibi bir altprogram

```
1 program_adı proc
2    ... <program kodu>
3    ret
4 program_adı endp
```

biçiminde tanımlanır ve

```
1    call programadı
```

ile çağırılarak kullanılır. Yukarıdaki örneği altprogramlar biçiminde yazarsak programımız aşağıdaki gibi olacaktır.

Örnek 7.2.

Çrnek 7.1'de makro kullanarak yazdığımız 'merhaba' kodunu bu kez altyordamlarla yazalım. Prosedürlerin parametresi ya yazmaçta, ya da yığıtta taşınmak zorundadır. Buradaki örnekte argumanların yazmaçta nasıl taşındığını görüyoruz.

```
1    model small
2    stack 64
3    .data
4 merhaba db 'Merhaba Millet..$'
5    .code
6 main proc far
7    mov ax,@data
8    mov ds,ax
9    ; EkranaYaz prosedürünü çağıracağız
10   ;  dx <-- metin adresi
11   mov dx,offset merhaba
12   call EkranaYaz
```

```
13    ; DOSaDön altyordamını çağıralım
14    call DOSaDön
15 main endp
16
17 EkranaYaz proc
18 ; giriş: dx: yazılacak metin adresi
19 ; çıkış: metin ekrana yazılır
20    push ax
21    mov ah,09h
22    int 21h
23    pop ax
24    ret
25 EkranaYaz  endp
26
27 DOSaDön proc
28 ; giriş yok, çıkış: dosa dönüş
29    mov ah,4ch
30    int 21h
31    ret
32 DOSaDön endp
33
34    end
```

Altyordamların mutlaka kod sektöründe tanımlanması gerekir ve tanımlanan altprogram tanımlandığı yerde kod oluşturur. Call ile çağırıldığı yerde ise önce bir sonraki komutun adresini dönüş adresi olarak yığıta koyar ve ardından call ile verilen adrese zıplar. Altprogramdan programa dönüş ret komutuyla yığıttaki dönüş adresinin komut imlecine indirilmesiyle gerçekleşir. Böylece işlemci dönüş adresini yığıttan indirmiş ve bir sonraki saat vuruşunda dönüş adresindeki komuta zıplamış olur.

Makro ile altprogram arasındaki temel farkları şöyle sıralayabiliriz. Makro tanımı kaynak metnin neresinde olursa olsun çevirici kaynak üzerindeki ön işlemi aşamasında makroları belleğine alır. Çevirici, belleğindeki bu tanımları her makronun komut gibi kullanıldığı yerlere gerekli arguman değişimini yaparak kopyalar. Böylece bir makro kaç defa komut olarak kullanılırsa her defası için derlenen kodun içinde bir kopya oluşturur. Oysa altprogram kod bölütü içinde tanımlanır, ve kodu tanımladığı yerde bir kere oluşur. Bu kod call komutuyla her çağırılışında program akışı kodu işleyip geri dönecek şekilde değiştirilmiş olur. makro ile altprogram arasındaki çeşitli açılardan görülen temel farklar Tablo da özetlenmiştir.

Özellik	Makro	Altprogram
Tanımlandığı yer	herhangi bir yer	kod bölütü
Tanımın durduğu yer	Çevirici Belleği	kod bölütü
Çağırma biçimi	makro adı	call komutu
Çağırınca eklenen kod	Tanımlı kodun tümü	call komutu
Birden fazla çağrıda	kod her çağrı için tekrarlanır	tekrarlanan yalnız call komutudur
Çağırılmazsa	kodlanmaz	gene de kodlanır
Parametre taşıma	dummy argumanla	yığıtta ya da yazmaçta

Bu özellikleri nedeniyle makrolar genellikle çok kısa ve sık sık tekrarlanan, ya da uzun olsa da yalnızca bir kere kullanılan işlemleri modülerleştirmekte kullanılırken alt programlar sadece uzun ve çok tekrarlanan kodları modülerleştirmekte kullanılırlar.

Makro tanımının parametre taşıma yeteneği, çok tekrarlanan ama uzun olan altprogramlara gereken parametre taşıma işlemlerini yapmak için çok elverişlidir. Bir makro tanımı bu çeşit uzun altprogram kodu için gereken parametreleri kolay anlaşılır biçimde yazmaçlara ve yığıta taşıyıp altprogramı çağırdıktan sonra yazmaçlarda geri dönen sonuçları da veri bölütünde tanımlı değişkenlere kolayca aktarabilir.

7.4 Makroda Lokal Etiketleme

Bir makro tanımlarken kullanılacak etiket önişleme aşamasında makronun kaynak metinde her kullanıldığı yere kopyalanacağından aynı etiket defalarca tekrarlanarak sorun çıkarır. Bu sorun makroda kullanılan tüm etiketleri lokal tanımlayarak çözülür. Etiketleri lokal tanımlamak için

```
1    local <etiket1><,etiket2>...
```

tanımına uygun bir derleyici komutu gerekir.

Örnek 7.3.

İki sayıyı çarpmak üzere `bx` yazmacındaki sayıyı `cx` kere toplayacak bir makro yazıp bu makroyu `wsayı1`×`wsayı2`/`wsayı3` işlemini hesaplamakta kullanalım. Bölme işlemini de ardışık çıkarmaları sayarak gerçekleştireceğiz.

```
1 carpım macro sayı1,sayı2,sonuç
2    local gene
3 ;bu makro sonuç=sayı1*sayı2 yi hesaplar
4 ;sayı1 ve sayı2 word, sonuç: doubleword
5     mov bx,sayı1
```

```
6     mov cx,sayı2
7     sub ax,ax ; ax sıfırlandı
8     mov dx,ax ; dx te sıfırlandı
9  gene:
10      add ax,bx
11      adc dx,0  ; elde varsa dx arttı
12      loop gene
13    mov result,ax   ; sonucun sağ wordu
14    mov result+2,dx ; sol wordu
15    endm
16
17 ; Bu makroyu kullanan programda
18 ;   'gene' adlı etiket olsa bile
19 ;   sorun çıkmaz
20   .model small
21   .data
22 wsayı1  dw 512
23 wsayı2  dw  4
24 wsayı3  dw  8
25 dcarp12 dd  ?
26 bölüm3  dw  ?
27    .code
28 main proc far
29 ; iki sayının çarpımını üçüncü sayıya bölelim.
30 ; ancak bölmeyi ardışık çıkarmalarla yapacağız.
31    mov ax,@data
32    mov ds,ax
33    çarpım wsayı1,wsayı2,dcarp12
34 ; şimdi bölme yapalım
35    sub bx,bx     ; bx sıfırlandı
36    mov dx,bx     ; dx te sıfırlandı
37    mov ax,wsayı3 ; ax çıkarılacak
38 gene:
39    sub dcarp12,ax
40    sbb dcarp12+2,dx ; elde borçluysa
41    js  bitti      ; işaret negatifse bitir
42    inc bx         ; bx bölmenin sonucu olacak
43    jmp gene
44 bitti:
45    mov bölüm3,bx
46 ; DOSa dönebiliriz
47    mov ah,ach
48    int 21h
49 main endp
50    end main
51 ; end komutunun işleneni olarak
52 ;  programın giriş noktasını yazıyoruz.
```

programın 9 ve 38 inci satırlarındaki `gene` etiketleri aynı olmasına karşın 9 daki `gene` lokal tanımlandığından program hata vermez.

7.5 Kaynak Metine Dosya Ekleme

Bu bölümde modüler programlama araçlarından makro ve altyordam yapılarını gördük. Büyük bir yazılımı bu araçları kullanmadan yazabilmek olağanüstü zordur. Moduler olmayan kısa yazılımları bile daha sonra değiştirebilecek kadar anlamak çok zordur. Ancak diğer yandan, kullandığımız makro ve altyordamlar bir kere test sağlama aşamasından geçmişse çalışmasını tekrar tekrar takip etmek gerekmediğinden kaynak metnin içinde fazladan yer kaplar. Bu çeşit sağlaması gerçekleştirilmiş makrolar konularına uygun isimlendirilmiş metin dosyalarına yerleştirilerek kaynak metin içine `include` komutu ile dahil edilirse kaynak metinde programın gövdesi ön plana çıkar.

`include` komutu, bir dosyadaki makro öbeğinin birden fazla kaynak metinde kolayca kullanımını sağlar. Bu şekilde makrolar geliştirildiğinde dosyayı kullanan kaynak metinler ilk derlenişlerinde kendiliğinden güncellenmiş olur. Komutun yapısı

```
1  include <dosyaadı.uzantısı>
```

biçimindedir.

Örnek 7.4.

Örnek 7.1deki kaynak metini makroları bir dosyaya koyarak yazalım. Önce makroların dosyası olan dosmakroları.txt dosyasını oluşturacağız.

—-dosya adı: dosmakroları.txt—-

```
1 EkranaYaz macro veri
2    mov dx,offset veri
3    mov ah,09h
4    int 21h
5    endm
6
7 DOSaDön macro
8    mov ah,4ch
9    int 21h
10   endm
```

—-dosya adı: merhaba.asm—-

```
1    model small
2    stack 64
3    include dosmakroları.txt
4    .data
5  merhaba db 'Merhaba Millet..$'
6    .code
7  main proc far
8    mov ax,@data
9    mov ds,ax
10   ;makroları kullanabiliriz
11   EkranaYaz merhaba
12   DOSaDön
13 main endp
14   end
```

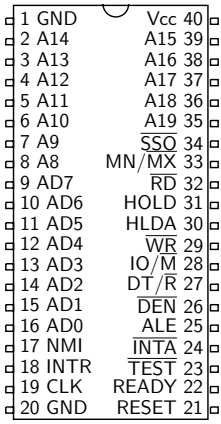
Üçüncü satırdaki `include` komutu `merhaba.asm` dosyasına `dosmakroları.txt` dosyasını kopyaladığından 11 ve 12 nci satırlarda `dosmakroları.txt` dosyasındaki makroları kullanabiliyoruz.

Bölüm 8

8086-8088 işlemcisi

8.1 Gelişimi ve uç tanımları

INTEL tarafından geliştirilmiş olan 8080 ile başlayıp Pentium dahil geniş bir mikroişlemci ailesinin 8-bit veriyolundan 16-bit veriyoluna geçiş noktasında 8088 ve 8086 olarak sunulmuş iki işlemci, 70'li yılların sonlarında yeni geliştirilmekte olan kişisel bilgisayarlarda kullanılmaya başlandı. Bu teknolojinin bilgisayar endüstrisinin devi IBM tarafından kabul görmesiyle birlikte 80x86 işlemciler kişisel bilgisayar endüstrisinde özel bir yere sahip oldular.



Şekil 8.1: 40 uçlu 8088 yongasının minimal mod uçları

8088-8086 işlemciler henüz entegre paketleme teknolojisinin en fazla 40-uçlu paketlere olanak sağlayabildiği dönemde geliştirildiğinden 16-bit veri genişliği için özel çoklamalı uç donanımı oluşturmak gerekti. İşlemci yongasının başta gelen giriş ve çıkış uçları şunlardır:

`AD0-AD7` hem adres hem de veri iletmek amacıyla kullanılır. Veri iletirken giriş ya da çıkış olabilmesine karşın adres iletirken daima çıkıştır.

A8-A19 çıkış uçları AD0-AD7'ye ilaveten 20 bit adres genişliğini oluşturur.

ALE çıkış ucu AD0-AD7 uçlarının adres olarak kullanıldığı sürede yüksek (H) olarak bu uçların adres verdiği süreyi belirler.

RESET giriş ucu işlemciyi belli bir adresten başlatır.

READY girişi, erişilecek adresin veri alma yada vermede yeterince hızlı olmaması durumunda işlemcinin veri aktarımı tamamlanıncaya dek bekleyebilmesi içindir.

INTR girişi önlenebilir kesme girşidir.

$\overline{\text{INTA}}$ çıkışı kesmenin kabul edildiğini onaylar.

NMI girişi önlenemez kesme girişidir.

CLK girişi saat girişidir. İşlemci saatsiz çalışmaz, her saat döngüsünde bir sonraki adıma geçer.

MN/$\overline{\text{MX}}$ girişi yükseğe bağlanırsa minimal donanım modu veya alçağa bağlanırsa maksimal donanım modu olmak üzere donanım modunu belirler. Maksimal modda denetim sinyallerini 8288 yongası üretir.

$\overline{\text{WR}}$ verinin belleğe ya da porta yazmak üzere yollandığını belirtir.

IO/M çıkışı yüksek (=1) durumunda adresin porta, düşük (=0) durumunda ise belleğe yönelik olduğunu belirtir.

DT/$\overline{\text{R}}$ çıkışı veri için kullanılan AD0-AD7 uçlarının DT/$\overline{\text{R}}$ =1 durumunda veri çıkışı olarak, DT/$\overline{\text{R}}$ =0 durumunda veri girişi olarak kullanıldığını gösterir.

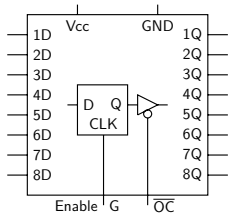
8.2 Adres ve Veri Yolları Yapısı

8088 sistemi en basit bellek arayüz devresini minimal modda gerektirir. Yalnızca bir bellek ve bir porta erişmek üzere tasarlanan bu sistemde 74LS373 8-bit sekizli mandal[1] kullanılarak ALE çıkışına bağlı olarak AD0-AD7 uçlarından gereken adres mandallanır.

Tablo 8.1: 74LS373 İşlev Tablosu

$\overline{\text{OC}}$	G	D	Q (çıkış)
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	hi-Z

[1] octal latch



Şekil 8.2: 74LS373 mandal yongası

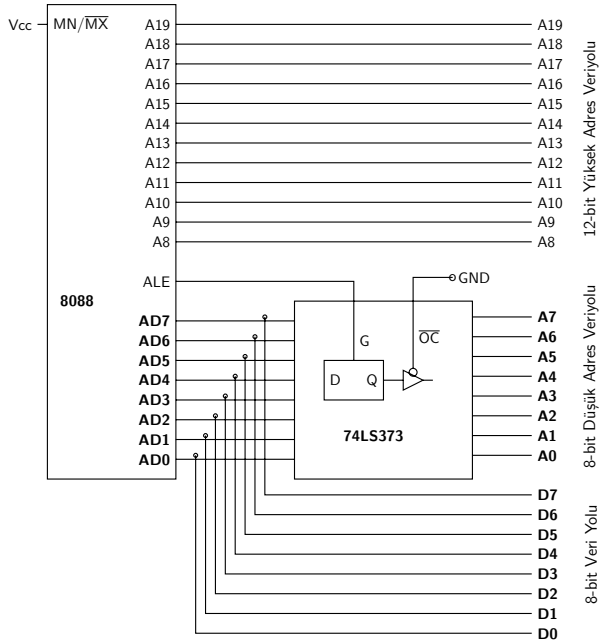
74LS373 yongasında sekiz veri mandalı ve her mandalın çıkışında birer üçdurumlu tampon devresi bulunur. Bütün mandalların veri geçirme kontrolu olan G kontrol girişi yüksekken (H veya 0 iken) mandalın durumunu D girişi belirler. Yonganın üçdurumlu çıkışlarının denetim girişi olan $\overline{OC}$ düşükken (L veya 0 iken) Q çıkışı mandal durumu değerindedir. $\overline{OC}$ H olduğunda Q çıkışı yüksek empedans (hi-Z) durumuna girerek bağlı olduğu devreden kopmuş gibi davranır.

ALE çıkışı 74LS373 mandal devresinin G girişine bağlıdır. İşlemcinin ALE çıkışı yüksek (H, ya da 1) ise mandal girişine bağlı AD0-AD7 adres sinyallerini mandal çıkışına iletir. ALE çıkışı düşerken (L ya da 0 olurken) mandaldaki son değer mandal çıkışında sabitlenir. Böylece ALE H iken AD0-AD7 uçlarından çıkan adres ALE L olunca da A0-A7 mandal çıkışlarında donakalır, ve AD0-AD7 uçlarının veri taşımayla görevli olduğu sürede A0-A7,A8-A19 adres yolu gene de doğru adresi vermeyi sürdürür.

8.3 8086-8088 Adres Uzayı

ALE düşerken mandallanarak elde edilen A0-A19 sinyalleri 8088'in toplam 20 adres biti genişliğindeki adres yolunu oluşturur. Bu sinyallerden A0-A7 hatları veri yolunun D0-D7 hatlarıyla çoklanarak elde edilmiştir. 20 hattan oluşan adres yolu 0'dan 1 048 575'e kadar toplam 2^{20} adresi belirtebilir. Her bir adres bir bayt veriye eriştiğinden 8088'in adres uzayı $2^{20} \times 8$ bit, ya da 2^{20} bayt, veya 1 Mega bayt biçiminde gösterilebilir.

8088'in adres çıkışları ancak bir TTL yük kaldırabildiğinden minimal olmayan 8088 devrelerinde bütün adres ve veri yolu uçları mandallanarak kullanılır.



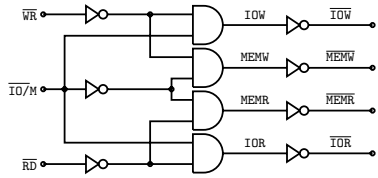
Şekil 8.3: 8088 çoklanmış adres-veri yolu yapısı

8.4 8086-88 veriyolu denetimi

8088 veriyolu denetimini altı çıkış ucundan gerçekleştirir.

- ALE çıkışı AD0-AD7 de geçerli bir adres varken yükselir.
- $\overline{\text{DEN}}$ çıkışı AD0-AD7 ye bağlanacak 74LS245 üçdurumlu ikiyönlü veriyolu tamponunun çıkış etkinleştirme sinyalidir.
- DT/$\overline{\text{R}}$ iki yönlü veriyolu tamponunun yönünü belirler.
- IO/$\overline{\text{M}}$ çıkışı yüksekse veriyolu çevriminin i/o adresine yöneliktir. Aksine düşükse çevrimde gönderilen adres bellektedir.
- $\overline{\text{RD}}$ yüksekse veriyolu çevrimi veri okumak için kullanılmıyor demektir. Aksine, düşükse çevrim veri okumaya yöneliktir.
- $\overline{\text{WR}}$ yüksekse veriyolu çevrimi veri yazmayı amaçlamıyor demektir. Aksine, düşükse çevrim veri yazmaya yöneliktir.

İşlemciden çıkan bu altı sinyalden ilk üçü doğrudan adres ve veri yollarının tamponlanması için kullanılır. Bunun dışında adres çözme ve bellek yada i/o erişiminde görev almaz. Son üç sinyal ise daha önce de belirttiğimiz gibi IBM-PC'de $\overline{\text{IOW}}$, $\overline{\text{IOR}}$, $\overline{\text{MEMW}}$, ve $\overline{\text{MEMR}}$ kontrol sinyallerini üretmekte kullanır. Standartlaşmış olan bu sinyaller daha ileride ISA veriyolu standardına dönüşmüştür.



Şekil 8.4: IBM-PC veriyolu denetim sinyalleri

8088 işlemcideki diğer başlıca denetim çıkışları şunlardır

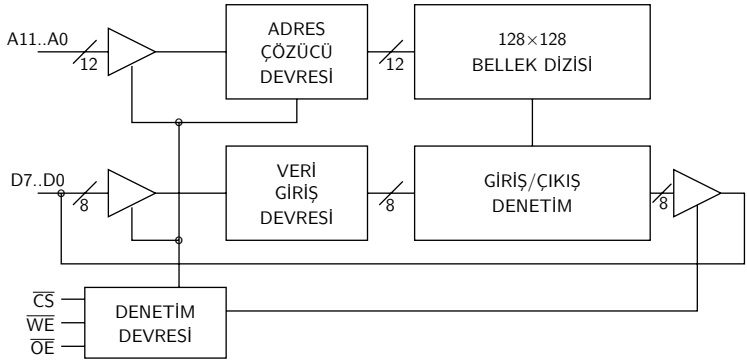
- CLOCK veriyolu durumunu eşzamanlamak için kullanılır.
- READY girişine işlemci T3 sonunda bakar ve etkin değilse T3 çevrimini tekrarlar. Böylece sistemdeki yavaş belleklere sorunsuz erişim sağlanır.
- NMI Maskelenemeyen kesme girişidir.
- INTR Maskelenebilir kesme girişi.
- RESET işlemci yazmaçlarını CS =FFFFh, IP =SS =DS =ES= 0000h ilkdeğerlerinden başlatır. Böylece işlemci bir sonraki veriyolu çevriminde FFFF0h fiziksel adresindeki komutu işler.

8.4.1 IBM-PC bellek ve port denetim sinyalleri

Minimal 8088 sisteminde donanım açısından adres verilme anını belirten işlemci çıkışı ALE, giriş/çıkış (i/o) adreslerinin bellek adreslerinden ayrışmasını sağlayan işlemci çıkışı IO/$\overline{\text{M}}$ ucudur. IBM-PC'de IO/$\overline{\text{M}}$, $\overline{\text{WR}}$ ve $\overline{\text{RD}}$ çıkışları, Şekil 8.4'deki basit çözücü devresiyle $\overline{\text{IOW}}$, $\overline{\text{IOR}}$, $\overline{\text{MEMW}}$ ve $\overline{\text{MEMR}}$ veriyolu kontrol çıkışlarını oluşturur. Bu çıkışların yanı sıra çiftyönlü veri yolunun tamponla güçlendirilmesi amacıyla DT/$\overline{\text{R}}$ ve $\overline{\text{DEN}}$ çıkışları gerekir.

8088 denetim girişleri

Bu çıkışların yanısıra 8088 yongasında saat çevrimini belirleyen clock, sistemi CS=FFFFh, IP=SS=DS=ES=0000h yazmaç değerleriyle ilk komut kodunu 0FFFF0h adresinden çalıştırmaya başlayan RESET girişi, maskelenemez ve maskelenebilir seviyede kesmeler için INTR ve NMI kesme girişleri, ve yavaş yongalardan okuma yapabilmek üzere T3 çevrimini tekrarlamayı denetleyen $\overline{\text{READY}}$ girişi önemli veri yolu kontrol sinyalleri arasında sayabiliriz. Bu sinyallar IBM-PC tasarımında kullanılarak yaygınlaşınca gerekli adres ve veri yollarıyla birlikte ISA stan-



Şekil 8.5: Statik belleğin yapısı ve denetim uçları

dart veriyolunu oluşturdu.

8.4.2 Işlemcinin başlama durumu

RESET işlemcinin yazmaçlarını CS=FFFFh, IP=SS =DS= ES=0000h ilkdeğerlerinden başlatınca, işlemci bir sonraki veriyolu çevriminde FFFF0h fiziksel adresindeki komutu işlemeye başlar. Işlemcinin FFFF0h adresindeki komutu okuyabilmesi için sistem besleme gerilimi kesildiğinde içindeki verilerin silinmeyeceği bir bellegin bu adreste etkinleşmesi gerekir.

Besleme gerilimi kesilse bile verilerin silinmediği bu çeşit belleklere uçucu olmayan (kalıcı) bellek denir. Günümüzde flaş-bellek olarak adlandırılan bellekler bu amaçla kullanılır. Flaş-belleklerin henüz geliştirilmediği yıllarda yalnız okunur bellek (ROM) bu amaçla kullanılmaktaydı. Sistemin başlaması açısından temel fark olmadığından biz devre örneklerimizi ROM bellekle sunacağız.

8.5 Bellek yongasının yapısı ve denetim uçları

Mikroişlemcilerle kullanılan statik rastgele erişimli (S-RAM) bellekler gene paketlemedeki bacak sayısı sınırlaması ve işlemciye uygunluk gibi nedenler yüzünden veri giriş ve veri çıkışı için aynı ucu kullanır.

Belleği etkinleştirmek üzere $\overline{\text{CS}}$ veya $\overline{\text{CE}}$ olarak adlandırılan bir giriş denetim ucu sistemde etkinleştirilmeyen belleklerin daha düşük güç harcayarak hazır beklemesini sağlar.

Belleğin adres uçlarından belirlenen adresindeki veriyi okumak üzere, $\overline{\texttt{OE}}$ çıkış etkinleştirme denetim girişi L yapılarak yonganın veri uçları çıkış olarak etkinleştirilir.

Belleğin adres uçlarından belirlenen adresine yeni veri değeri yazmak için $\overline{\texttt{WE}}$ yazma etkinleştirme denetim girişi L yapılarak yonganın veri uçlarını giriş olarak kullanması ve gelen değeri seçilen adrese yazması sağlanır.

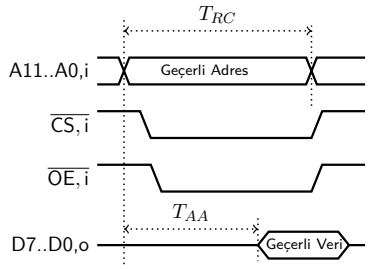
8.5.1 Belleğe erişim hızı

Hem okunabilip hem de yazılabilir bellek yongalarında üç denetim girişi bulunur. $\overline{\texttt{CS}}$ yongayı seçip okuma ya da yazma işlemine hazırlar. $\overline{\texttt{RD}}$ veri hatlarını çıkış yaparak adres hatlarından girilen yazmaç adresindeki yazmaç içeriğini bu çıkışlara bağlar. $\overline{\texttt{WR}}$ etkin olduğunda belleğin veri hatları giriş olur ve işlemciden gelen veriyi adreslenen yazmaç girişine bağlayıp bu yazmaca yazar. Tipik bir bellek okuma işleminde zamanlama çok önemlidir. Şekilde görüldüğü gibi $\overline{\texttt{CS}}$ ve adres verildikten sonra $\overline{\texttt{RD}}$ etkinleşse bile henüz adres çözümlenmeden adreslenen yazmacın içeriği okunamaz. Bu gecikme nedeniyle T_{aa} adres erişim[2] süresi ve T_{rc} bellek okuma çevrim süresi önen kazanır. Bazı bellek çeşitlerinde okuma ile yazmaçevrim süreleri birbirine eşit olmayabilir. Adres ve veri verilip $\overline{\texttt{WR}}$ etkinleştikten sonra bile verinin yazmaca yazılması için fazladan gereken süre T_{wc} süresinin uzamasına neden olur.

8.5.2 Bellek okuma çevrimi

Bellekte istenen adreste saklı olan veri değerini okumak için şu işlemlerin sırasıyla uygulanması gerekir:

[2] address access



Şekil 8.6: Bellek okuma çevrimi zamanlama parametreleri, T_{RC}: Okuma çevrimi süresi, T_{AA}: Adres erişim süresi

BELLEK OKUMA ÇEVRİMİ

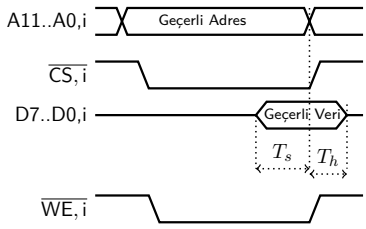
- İşlemci
  - ▷ A19-A0 üzerinden okunacak adresi yollar
    - – Adres çözücü belleğin $\overline{CS}$ ucunu etkinleştirir.
  - ▷ IO/$\overline{M}$ ile $\overline{RD}$ ucunu düşürür. Böylece belleğin OE girişi düşer ve veri çıkışı etkinleşir.
- Bellek
  - ▷ yongaya gelen adresi çözümler
  - ▷ adresin gösterdiği yazmacı seçer
  - ▷ seçilen yazmaç çıkışını veri çıkışına bağlar
- İşlemci
  - ▷ veri çıkışındaki bellek içeriğini komutun belirlediği işlemci yazmacına aktarır.
  - ▷ $\overline{RD}$ çıkışını yükseltince belleğin $\overline{OE}$ girişi yükselir. Veri çıkışı durur, ve okuma işlemi sonlanır.

8.5.3 Bellek yazma denetimi

Belleğin istenen adresine istenen veriyi yazmak üzere aşağıdaki işlem dizisinin sırayla uygulanması gerekir:

BELLEK YAZMA ÇEVRİMİ

- İşlemci
  - ▷ A0-A19 uçlarından gereken bellek adresini yollar
    - – Adres çözücü belleğe $\overline{CS}$ yollar. Bellek etkinleşir ve belleğin adres çözücüsü çalışmaya başlar.
  - ▷ IO/$\overline{M}$ ile $\overline{WR}$ ucunu düşürür. Böylece belleğin $\overline{WE}$ girişi düşer ve veri girişi etkinleşir.



Şekil 8.7: Bellek yazma çevrimi zamanlama parametreleri. T_s: Veri kurma süresi, T_h:Veri tutma süresi

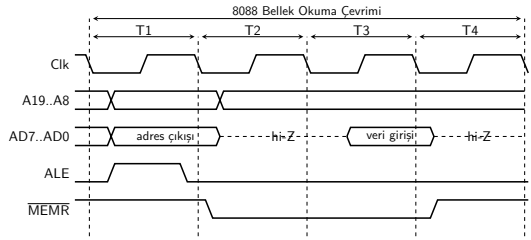
- ▷ AD0-AD7 uçlarından adrese yazılacak veriyi yollar.
- ○ Bellek
  - ▷ uygulanan adresi çözümleyip gösterdiği bellek yazmacını seçer.
  - ▷ veri girişini seçilen yazmaca yönlendirir.
- ○ İşlemci
  - ▷ $\overline{\texttt{WR}}$ çıkışını yükseltince belleğin $\overline{\texttt{WE}}$ girişi yükselir. Belleğe ulaşan veri seçilen adrese mandallanır ve yazma işlemi sonlanır.

8.6 8088 Bellek zamanlaması

8088 işlemcisinin her bellek çevrimi Şekil 8.8'da gösterilen en az dört saat çevriminden oluşur. Okuma çevrimini oluşturan saat çevrimleri `T1`, `T2`, `T3`, ve `T4` olarak adlandırılır. Bir bellek çevriminin bütün saat çevrimlerinde `A7-A19` adresin ikinci ve üçüncü baytlarını çıkarmaya devam eder. Belleğe erişim yapılacaksa bellek çevrimi sonlanıncaya dek `IO/`$\overline{\texttt{M}}$ çıkışını da düşürür. Beklemesiz bellek okuma çevriminde sadece bir `T3` olmasına karşılık, beklemeli çevrimde `T3`'ler `READY` girişi yükselene kadar tekrarlanır.

8.6.1 Beklemesiz bellek okuma çevrimi

`T1` çevrimi `ALE`'nin yüksek olduğu saat çevrimidir. Bu çevrimde `AD0-AD7` erişilecek belleğin adresini çıkarır. Bu uçlara bağlı 8-li mandal yongası bu adresi mandallar ve ileride `AD0-AD7` veri iletmekte kullanılıyorken bile mandal çıkışlarında `A0-A7` daha önce `ALE` yüksekken mandallanmış olan adresi vermeye devam eder. İşlemci `ALE`'yi `T1` sonunda düşürür.



Şekil 8.8: 8088 beklemesiz bellek okuma çevrimi

T2 başında işlemci $\overline{\text{RD}}$ çıkışını düşürünce IO/$\overline{\text{M}}$=0, $\overline{\text{WR}}$=1, ve $\overline{\text{RD}}$=0 olduğundan veriyolu kontrol çözücü $\overline{\text{MEMR}}$ çıkışını düşürür. Böylece bellek adres çözücü adresi verilen bellek yongası için $\overline{\text{CS}}$ üretir ve bellek te adresi çözümleyip erişilecek bellek yazmacını belirleme aşamasına geçer.

Bellek yongasının adresi çözmesi ve seçilen yazmacı veriyoluna bağlaması için belli bir süre gerekir. T2 ve T3 saat çevrimleri bu gecikme için gerekli süreyi sağlar.

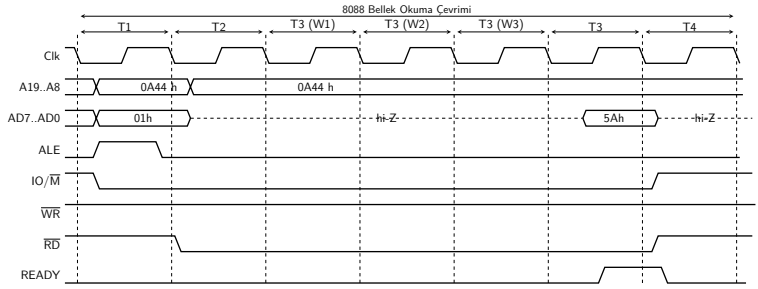
İşlemci T3 sonunda önce READY girişine bakar. Bellek yeterince hızlıysa READY hep yüksek durur. İşlemci READY yüksek ise belleğin veri yoluna çıkardığı veriyi yüklemesi gereken yazmaca aktarır. Böylece T3 sonunda belleği okuma ve okunan değeri gereken yazmaca aktarma işlemi tamamlanır.

T4 çevrimi başında işlemci $\overline{\text{RD}}$ çıkışını yükselterek etkinsizleştirir. T4 boyunca işlemci içeride bir sonraki veri çevrimine hazırlanırken veri yolu boşta kalır.

8.6.2 Beklemeli bellek okuma

İşlemci bellek okuma çevrimine kıyasla bazısı hızlı bazısı oldukça yavaş çeşitli tipte belleklerin bulunduğu bir sistemde çalışırken adres çözücü her belleğin etkinleştirme sinyaliyle birlikte bellek bekleme devresini de başlatır. Bu devre belleğin henüz hazır olmadığı süre boyunca işlemcinin READY sinyalini etkinleştirmez.

Şekil 8.9'de READY girişi kullanılarak gerçekleştirilen beklemeli bellek okuma çevrimi zamanlama diyagramı görülüyor. Burada işlemci saat çevrim süresi T_p=50ns olan bir 8088 bellek okuma çevrim süresi T_{RC}=280 ns olan bir belleğin 0A4401h adresindeki 5Ah değerini okuyor.



Şekil 8.9: 8088 beklemeli bellek okuma çevrimi

Beklemeli okuma çevriminde T1, T2 ve T3 aynen beklemesiz çevrimdeki gibi gelişir. Ancak, belleğin hazır olamayacağı 280ns süreyi belleğin $\overline{CS}$ sinyalinden başlayarak tutan bellek bekleme devresi bu süre boyuncaREADY girişini etkinsizleştirir.

Birinci T3 sonunda işlemci READY sinyalini etkin bulamayınca veriyi okumaz ve T3 çevrimini bir kez daha tekrarlar. İkinci ve üçüncü T3 çevrimlerinde de READY belleğin hazır olmadığını bildirdiğinden tekrar T3 çevrimine başlar. Ancak dördüncü T3 çevrimi sırasında belleğin 280 ns süresi dolunca bellek bekleme devresi READY girişini etkinleştirir. Dördüncü T3 sonunda işlemci READY girişini etkin bulunca veriyi okuyup gereken yazmaca aktarır. Böylece dört T3 çevriminin ardından okunan veri yazmaca aktarılır.

Sonunda READY'nin etkin olmadığı ilk üç T3 çevrimi bekleme çevrimleri olarak, READY'nin etkin olduğu T3 çevrimi ise okuma çevrimi olarak adlandırılır. bellek okuma çevrimi işlemcinin iç işlerini tamamladığı T4 boş çevrimi bitince sonlanır.

8.6.3 i/o (giriş/çıkış) port işlemleri

Minimal 8088 devresi üzerinde giriş çıkış komutlarına açıklık getirmek üzere daha önce giriş çıkış portlarını işlemcinin veriyolu denetim sinyalleriyle nasıl kurduğumuzu görmüştük. Burada IBM-PC veriyolu denetim sinyalleri kullanarak, basit giriş çıkış portları oluşturmayı hedefliyoruz.

8088 ve 8086'nın izole giriş/çıkiş komutları dört çeşittir.

- `in al, p8`
  Buradaki p8, 8-bit anlık port adresidir. Adreslenen giriş portunun girişindeki veri al yazmacına aktarılır.
- `mov dx, p16`

```
in al,dx
```

16-bit port adresi p16, dx yazmacına konarak kullanılır. in komutu, dx'in gösterdiği giriş portundan okunan veriyi al yazmacına yazar.

- ```
  out p8,al
  ```

  al deki veriyi anlık 8-bit adresi p8 olan çıkış portu yazmacına mandallar.

- ```
  mov dx,p16
  out dx,al
  ```

  Bu komut, al yazmacındaki veriyi dx'in gösterdiği, adresi p16 olan çıkış portu yazmacına yazar.

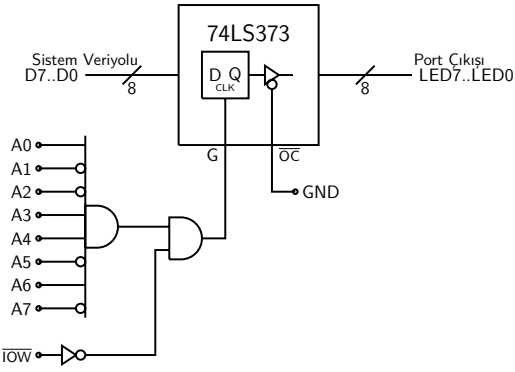
Her iki giriş komutunda da M/$\overline{\text{IO}}$ ve $\overline{\text{RD}}$ çıkışları etkinleştiğinden ISA sinyallerinden $\overline{\text{IOR}}$ etkinleşerek kullanılan port adresinin giriş portu olduğunu belirtir.Her iki çıkış komutunda da port adresiyle birlikte M/$\overline{\text{IO}}$ ve $\overline{\text{WR}}$ çıkışları etkinleşerek IBM-PC denetim sinyali$\overline{\text{IOR}}$'u etkinleştirir ve kullanılan port adresinin giriş portu olduğunu belirtir. Böylece bu dört giriş/çıkış (i/o) komutu sayesinde gerektiğinde tek komutla 8-bit adresli portlara, ya da önce 16-bit adresi dx yazmacına koyarak 16-bit adresli portlara kolayca erişmek mümkündür.

8.6.4 8088 basit çıkış portu

İşlemci doğru çıkış portu adresini kullandıysa, o adres ile seçilmiş bir 74LS373 8-bit veri mandalına, işlemcinin D0-D7 veri yolundan tam i/o çıkış komutu işlenirken gelen al içeriği mandallanırsa al'deki değer port yeniden adresleninceye kadar mandal çıkışında harici devrelerce kullanılabilir. Böyle bir basit adres çozücü ve mandal devresi bir basit çıkış portunu oluşturur.

Basit çıkış portunun mandalı veriyolundan gelen verileri tam doğru port adresi kullanıldığında ve aynı anda $\overline{\text{IOW}}$ veriyolu denetim sinyali etkinleştiğinde açılır. $\overline{\text{IOW}}$ sona erince veriyolundaki değer mandalda donar ve mandal çıkışına bağlı olan harici devrelere iletilir.

Şekil 8.10'teki çıkış portuna al deki 12h sayısını yazmak için CS:IP = 01A7: 0018 adresindeki out 59h,al komutunun her bir saat çevriminde nasıl çalıştığını görelim. Daha önce çalışmış olan mov al,12h komutunun al=12h yapmış olsun. out 59h,al komutunun makina kodu E6h 59h olduğundan 01A7:0018 (FA=01A88h) adresinde E6h, bir sonraki adreste de 99h duruyor. Komutun çalışabilmesi için önce işlemci FA=01A88h'dan E6h komut kodunu okuyacaktır. Bunun için iki saat



Şekil 8.10: 59h port adresinde 74LS373 mandallı basit çıkış portu devresi

çevrimi gerekir. Birinci çevrimde ALE yükseltilerek adres verilecek, bu sırada adres bellek içinse IO/$\overline{M}$=0 olacaktır. İkinci çevrimde ise ALE düşük dururken bellekten komutu okumak üzere $\overline{RD}$ düşürülüp bellekteki E6h değerinin AD[0..7] üzerinden komut yazmacına okunması sağlanır.

Tablo 8.2: `out 59h,al` komutunun işleme ayrıntısı

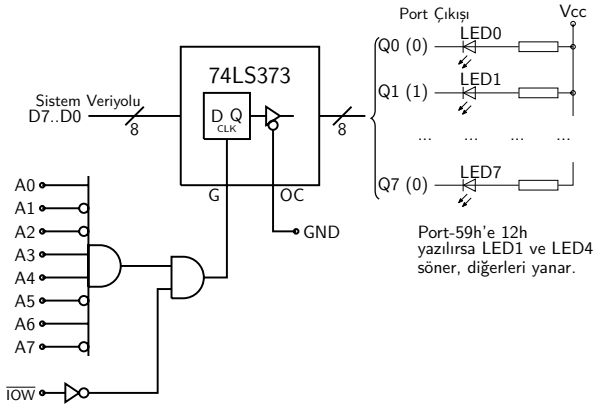
çevrim	A[19..8]	AD[0..7]	$\overline{WR}$	$\overline{RD}$	IO/$\overline{M}$	açıklama
ALE (T1)	**01Ah**	**88h**	1	1	0	**FA=01A88h bellekte**
bellek oku (T3)	**01Ah**	**E6h**	1	0	0	**kod=E6h okunuyor**
ALE (T1)	**01Ah**	**89h**	1	1	0	**FA=01A89h bellekte**
bellek oku (T3)	**01Ah**	**59h**	1	0	0	**kod=59h okunuyor**
ALE (T1)	**000h**	**59h**	1	1	1	**FA=00059h i/o port**
porta yaz (T3)	**000h**	**12h**	1	0	0	**veri=12h port 59h'e**
ALE (T1)	01Ah	8Ah	1	1	0	FA=01A8Ah bellekte
bellek oku (T3)	01Ah	8Bh	1	0	0	kod=8Bh okunuyor

Tabloda gördüğümüz koyu yazılmış ilk altı sıra OUT komutunun yürütülmesini açıklıyor. Son iki satırda ise, işlemci daha sonra sırası gelen `MOV AX,BX` komutunun kodundaki ilk bayt olan 8B kodunu okuyor.

İlk satırdaki T1 çevriminde işlemci 01A88h adresli bellekten out komutunu okumak üzere adres yolluyor. Sistemdeki 74LS344 mandallar bu adresi okuma sonlanıncaya kadar adres yollarında tutar. İkinci satırda aynı veri çevriminin T3 saat çevriminde işlemci bellekten gelen komut kodunu komut yazmacına aktarıp çözer. Bu komutun OUT olduğunu ve ikinci kod baytının port adresi olduğunu çözümleyince üçüncü satırdaki T1 saat çevriminde işlemci ALE 'yi yükselterek A19..A8

ve AD7..AD0 uçlarından bir sonraki adres olan 01A89h sayısını çıkarır. Bu adres gene adres mandallarında saklanıp bir sonraki veri okuma çevrimi boyunca adres mandal çıkışlarından SA19..SA0 olarak belleğe ve adres çözücüye ulaşır. Dördüncü satırdaki T3 veri okuma çevriminde veriyolundan işlemciye 59h ulaşıyor.

İşlemci, OUT 59h,AL komutunun son veriyolu çevriminin T1 saat çevriminde ALE'yi yükseltip adres yoluna port adresi olan 00099h adresini çıkarıyor. Son veriyolu çevriminin verinin iletileceği T3 saat çevriminde ise işlemci veri yoluna AL'deki 12h değerini iletip bunun çıkış portu tarafından mandallanması için i/o yazma denetim sinyalleri çıkarır. AL'deki 12h veri yolunda yalnızca bir saat döngüsü kalır, ama bu döngü sırasında çıkış portu mandalı 12h'yi çıkışa iletip saklamaya başlar. Mandal girişindeki 12h bir sonraki saat döngüsünde değişse bile mandalda saklanan 12h port çıkışında durmaya devam eder. Port çıkışı ancak işlemci porta başka bir değer yazan bir OUT komutu işletirse değişir.



Şekil 8.11: Çıkış portuna bağlı LED'ler

Diyelim ki Şekil 8.11'deki gibi portun b0 ve b1 biti çıkışları akımı sınırlayan birer direnç üzerinden anodu 5Volta bağlı LEDin katoduna bağlı olsun. Bu durumda çıkış ucu düşükse (=0V) LED ışıldar, yüksekse LED söner. Port çıkışında sürekli kalan 12h=00010010_2 değeri portun b0 bitine bağlı LED1'i ışıldatır ve b1 bitine bağlı LED2'yi söndürür.

8.6.5 8088 basit giriş portu

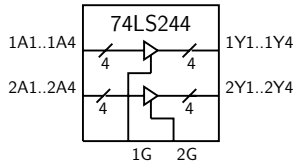
Basit giriş portunun amacı, port girişindeki sinyalleri tam portun adreslenerek okunduğu anda veri yoluna iletmektir. Bu iş için üç-durumlu çıkışı olan tampon devre kullanılması gerekir.

74LS373 ile giriş portu

74LS373 mandal devresinin çıkışındaki üç-durumlu tampon bu amaçla kullanılabilir. Tamponun önündeki mandal devresinin giriş ucunu tampon girişine iletebilmesi için `G` girişini sürekli yükseğe (+Vcc'ye) bağlamak gerekir.

74LS244 ile giriş portu

74LS244 sayısal sinyalleri güçlendirmek ve anahtarlamak üzere kullanılan dual dörtlü üç-durumlu tampon devresidir.



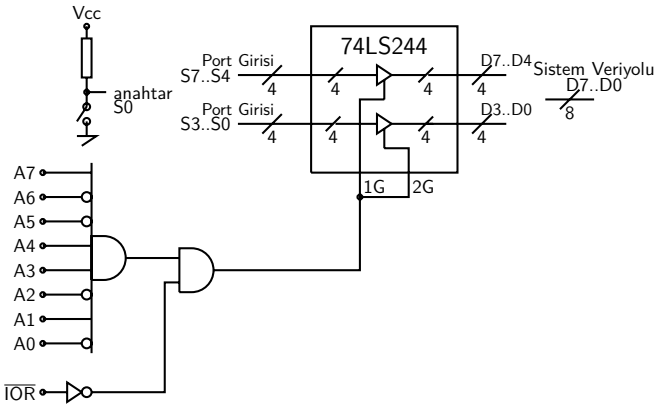
Şekil 8.12: 74LS244 üçdurumlu dual 4-bit tampon yongası

Tampon devre girişteki sinyalleri daha yüksek akımları sürebilecek şekilde güçlendirerek çıkışa iletir. İlk dört girişteki `1A1`, ... ,`1A4` sinyalleri `1Y1`, ... ,`1Y4` çıkışına ancak `1G` denetim girişi düşükken iletilir. `1G` yüksekken `1Y1`, ... ,`1Y4` çıkışları yüksek empedans (hi-Z) durumuna geçerek sanki yalıtkan gibi davranır. Benzer şekilde aynı yongadaki ikinci dörtlü tamponun `2A1`, ... ,`2A4` girişleri de `2Y1`, ... ,`2Y4` çıkışlarına ancak `2G` düşükken iletilir. `2G` yüksekken bu dört çıkış yüksek empedans (hi-Z) durumuna geçer. Yonga sekiz sinyalin birlikte tamponlanmasında kullanılırken `1G` ve `2G` birbirine bağlanır ve genellikle `G` olarak adlandırılır. Bu durumda giriş ve çıkışlar de `A0`, ... ,`A7` ve `Y0`, ... ,`Y7` olarak adlandırılabilir.

Giriş portu devresi

İşlemci IN komutunun son veriyolu çevriminde port adresini verip porttan gelen veriyi AL yazmacına aktarır. Port tamponu girişindeki sayısal

değerin tam doğru anda veri yoluna bağlanması gerekir. Bunun için Şekil 'deki gibi hem i/o adres çözücünün doğru adreste etkinleşmesi, hem de bu adresin i/o okuma amaçlı verildiğini gösteren $\overline{\texttt{IOR}}$ sinyalinin etkinleşmesi gerekir. Bir mantık kapısı bu iki sinyalin düşük olduğu süre boyunca düşük çıkış sağlarsa bu çıkışla çıkış tamponunun G girişi etkinleştirilerek tampon veri girişleri işlemcinin veri yoluna aktarılır.



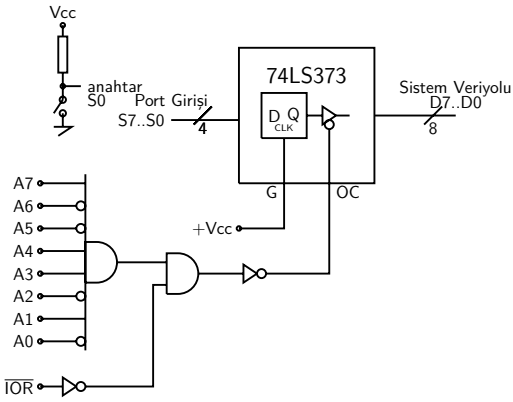
Şekil 8.13: 74LS244 ile 9Ah adresli giriş portu devresi

Şekil 8.13'de verilen adres çözücü A7..A0 adres uçları 1Ah olduğunda yüksek, diğer bütün adreslerde düşük çıkış veriyor. Adres i/o porta yazmak için verildiğinde denetim devresi işlemcinin IO/$\overline{\texttt{M}}$, $\overline{\texttt{WR}}$ ve $\overline{\texttt{RD}}$ çıkışlarına bakarak $\overline{\texttt{IOR}}$ sinyalini etkinleştirir (yükseltir). Bir mantık kapısı hem adres çözücü çıkışının hem de $\overline{\texttt{IOR}}$ çıkışının etkin olduğu anda giriş portu tamponunun G denetimini etkinleştirerek S7..S0 anahtarlarının durumunu işlemcinin veri yoluna aktarır. İşlemci T3 çevriminde bu değeri AL yazmacına aktararak program kodunun kullanımına açar.

Diyelim ki portumuzun D7..D0 bitlerine bir ucu 0V'a (toprağa) bağlı olan S7..S0 anahtarları (ya da butonları) bağlı olsun. Anahtarlar açıkken port girişi belirsiz kalmasın diye bütün port girişlerinden +5V'a örneğin 10kΩ gibi bir yükseltme direnci bağlanmış olsun. Böylece S0'a bağlı anahtar açık devreyse porttan okunan verinin b0 biti 1, anahtar kapalıysa b0 biti 0 olacaktır. Yükseltme dirençleri bağlı girişlerin anahtar bağlanmamış olanları da 1 olarak okunacaktır. Örneğin yalnızca S0 kapalı, diğerleri açık devreyken in AL,6Ah komutuyla anahtarların durumu AL yazmacına 'FEh' olarak geçer.

8.6.6 74LS373 ile giriş portu

74LS373 yongasındaki G girişi Vcc'ye bağlandığında (G=1) mandal devresi D girişini kesintisiz olarak Q çıkışına aktarır. Yongadaki çıkış tamponu $\overline{\texttt{OC}}$=0 ile çıkış sağlayıp $\overline{\texttt{OC}}$=1 olduğunda çıkışı hi-Z duruma getirir. $\overline{\texttt{OC}}$ denetiminin 74LS244'teki 1G ve 2G'ye göre ters mantıkta çalışması nedeniyle devrede bir de evirici kullanırız.



Şekil 8.14: 9Ah adresli Giriş portunun 74LS373 3-durum çıkışlı mandalla gerçekleştirilmesi

İşlemci, örneğin CS:IP=01A7:0018 (FA=01A88h) adresindeki `E4h 9Ah` ile kodlanmış `IN AL,9Ah` komutunu iki bellek veri çevrimi ve bir i/o veri çevrimi olmak üzere üç veri çevriminde gerçekleştirir. Tablo 8.3'de görüldüğü gibi ilk iki veri çevrimi bellekten komut koduna ve peşinden gelen port adresine erişmek içindir. Üçüncü veri çevriminin `T1` saat çevriminde işlemci port adresi olan `9Ah` adresini çıkarır ve aynı veri çevriminin `T3` saat çevriminde porttan gelen `FEh` sayısını `AL` yazmacına aktarır. Böylece `IN` komutu tamamlanmış olur. Ardından gelen veri çevrimi `IN`'den sonraki komutun okunduğu çevrimdir.

IN komutuyla `AL`'ye aktarılan FEh'deki bit-0'ın düşük, bit-1'in yüksek olması tam portun okunduğu anda S0 anahtarının kapalı devre, S1 anahtarının ise açık devre olduğunu gösterir. Böylece eğer S1 anahtarına bağlı olarak yapılacak bir işlem varsa `AL`'nin b1 bitine bakılarak gereken işleme karar verilebilir.

Tablo 8.3: `IN AL,9Ah` komutunun işleme ayrıntısı

çevrim	A[19..8]	AD[0..7]	$\overline{WR}$	$\overline{RD}$	IO/$\overline{M}$	açıklama
ALE (T1)	**01Ah**	**88h**	1	1	0	**FA=01A88h bellekte**
bellek oku (T3)	**01Ah**	**E4h**	1	0	0	**kod=E4h okunuyor**
ALE (T1)	**01Ah**	**89h**	1	1	0	**FA=01A89h bellekte**
bellek oku (T3)	**01Ah**	**9Ah**	1	0	0	**kod=9Ah okunuyor**
ALE (T1)	**000h**	**9Ah**	1	1	1	**FA=0009Ah i/o port**
portu oku (T3)	**000h**	**FEh**	1	0	0	**port-9Ah'den AL'ye FEh**
ALE (T1)	01Ah	8Ah	1	1	0	FA=01A8Ah bellekte
bellek oku (T3)	01Ah	8Bh	1	0	0	kod=8Bh okunuyor

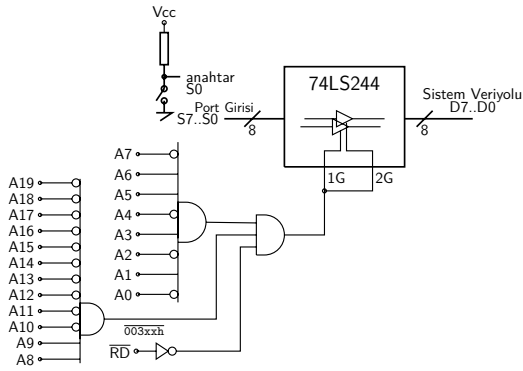
8.7 Bellek üzerinden giriş/çıkış

İntel'de bir ayrıcalık olarak bulunan izole giriş/çıkış sistemi pek çok mikroişlemci sisteminde gerçekleştirilmemiştir. Özellikle azaltılmış komut setine sahip RISC işlemcilerde komut setine diğer komutlara oranla daha az kullanılan giriş/çıkış komutlarını eklemek, verimini komut sayısını azaltarak yükselten RISC felsefesine aykırıdır. Genel olarak izole giriş/çıkışın Intel işlemcilere sağladığı avantajların başında i) bellek uzayının hiçbir bölümünün giriş/çıkışa ayrılmamasını, ii) giriş/çıkış adres çözücüsünün bellek adres çözücüsünden daha basit olmasını, ve iii) bu basitliği sayesinde daha hızlı veri aktarımı yapabilmesini sayabiliriz. Buna karşılık, sistemde izole giriş/çıkış bulunsa bile, bellek üzerinden yapılacak giriş/çıkış sisteminin, i) bellek üzerinde işlem yapabilen aritmetik mantık komutlarının giriş/çıkış için kullanılabilmesi, ii) giriş/çıkış adres uzayının daha genişlemesi, iii) daha az denetim sinyali kullanan sistem donanımının basitleşmesi, ve iv) giriş/çıkışların dolaylı ve doğrudan gibi daha çeşitli adresleme modlarıyla gerçekleştirilmesini sayarız.

8.7.1 Bellek üzerinden giriş portu

Bellek üzerinden giriş portunda üçdurumlu giriş tamponu $\overline{IOR}$ yerine $\overline{MEMR}$ ile etkinleştirilir. Örneğin Şekil 8.15'teki port adresi `0036Ah` olan bir giriş portunda adres seçme devresi bütün adres bitlerini işliyor, ve port bellek adresiyle kullanılacağından ötürü $\overline{IOR}$ yerine $\overline{MEMR}$ etkinken seçiliyor.

Şekil 8.15 devresindeki giriş portunu okurken `in al,dx` komutu yerine, 0036Ah adresinden okuma yapacak herhangi bir komut kullanmak olanağı vardır. Örneğin programın başlangıcında `ES`'ye sıfır yaza-



Şekil 8.15: Bellek üzerinden giriş devresi bütün adres bitlerine bakılarak etkinleştirilir ve $\overline{\texttt{IOR}}$ yerine $\overline{\texttt{MEMR}}$ kullanılır.

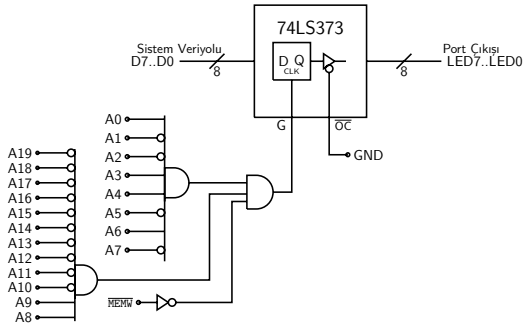
rak bu bellek adresine `036Ah` ofsetiyle erişebiliriz.

```
1   xor AX,AX
2   mov ES,AX ; ES=0 oldu.
3     ...
4   mov bl,ES:[036Ah]
```

Adres bellekte olduğundan, bu adresi doğrudan bellekten ikinci işlenen olarak aritmetik-mantık komutlarıyla bile kullanabiliriz.

8.7.2 Bellek üzerinden çıkış portu

Bellek adresinden çıkış portu için port mandalı etkinleştir sinyalini üretirken tüm adres bitlerinin kullanılması, ve etkinleştirmenin $\overline{\texttt{IOW}}$ yerine $\overline{\texttt{MEMW}}$ sinyaliyle gerçekleşmesi gerekir.



Şekil 8.16: Bellek üzerinden çıkışta $\overline{\texttt{MEMW}}$ ya da $\overline{\texttt{WR}}$ kullanılır.

Bellek üzerinden çıkış portuna yollanacak değerlerin al de olması gerekmez, ancak çıkış portuna erişmek için uygun bir bölüt adresi kullanmak gerekir. Örneğin şekildeki gibi adresi 0039Ah olan porta erişmek üzere ES = 0 iken ES:039Ah adresini kullanmak yeterlidir.

8.7.3 İzole giriş/çıkışı olmayan sistemler

Bellek üzerinden giriş/çıkış gerektiğinde 8086 gibi izole giriş çıkış olanakları bulunan bir işlemciyle uygulanırsa sistemde $\overline{\text{MEMR}}$ ve $\overline{\text{MEMW}}$'nin yanısıra$\overline{\text{IOR}}$ ve $\overline{\text{IOR}}$ sinyalleri de bulunacaktır. Oysa, izole giriş/çıkış olanağı bulunmayan işlemcilerde bütün okuma ve yazma işlemleri bellek üzerinde olduğundan bu tip sistemlerde $\overline{\text{IOR}}$ ve $\overline{\text{IOR}}$ bulunmaz. Bu durumda $\overline{\text{MEMR}}$ ve $\overline{\text{MEMW}}$ sinyallerine kısaca $\overline{\text{RD}}$ ve $\overline{\text{WR}}$ adı verilir. 8051 işlemcisiyle kurulan sistemler bu duruma tipik bir örnektir.

8.8 IBM-PC veriyolu denetim sinyalleri

8.8.1 8088 denetim çıkışları

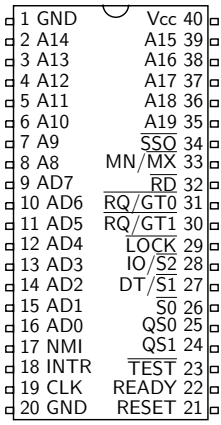
Minimal 8088 sisteminde donanım açısından adres verilme anını belirten işlemci çıkışı ALE, i/o adreslerinin bellek adreslerinden ayrışmasını sağlayan işlemci çıkışı IO/$\overline{\text{M}}$ ucudur. IBM-PC'de IO/$\overline{\text{M}}$, $\overline{\text{WR}}$ ve $\overline{\text{RD}}$ çıkışları, Şekil 8.4'deki basit çözücü devresiyle $\overline{\text{IOW}}$, $\overline{\text{IOR}}$, $\overline{\text{MEMW}}$ ve $\overline{\text{MEMR}}$ veriyolu kontrol çıkışlarını oluşturur. Bu çıkışların yanı sıra çiftyönlü veri yolunun tamponla güçlendirilmesi amacıyla DT/$\overline{\text{R}}$ ve $\overline{\text{DEN}}$ çıkışları gerekir.

8.8.2 8088 denetim girişleri

Bu çıkışların yanısıra 8088 yongasında saat çevrimini belirleyen clock, sistemi CS=0FFFFh, IP=SS=DS=ES=0000h yazmaç değerleriyle ilk komut kodunu 0FFFF0h adresinden çalıştırmaya başlayan RESET girişi, maskelenemez ve maskelenebilir seviyede kesmeler için INTR ve NMI kesme girişleri, ve yavaş yongalardan okuma yapabilmek üzere T3 çevrimini tekrarlamayı denetleyen $\overline{\text{READY}}$ girişi önemli veri yolu kontrol sinyalleri arasında sayabiliriz. Bu sinyallar IBM-PC tasarımında kullanılarak yaygınlaşınca gerekli adres ve veri yollarıyla birlikte ISA standart veriyolunu oluşturdu.

8.9 8088 maximal modu

İşlemcinin MN/$\overline{\text{MX}}$ girişi Vcc'ye bağlanarak yüksek mantık seviyeye sabitlenirse işlemci minimal sistem yapısında çalışır ve bazı uçları Şekil 8.17'deki gibi 8288 veriyolu denetlecine uyumlu çıkışlar üretir. MN/$\overline{\text{MX}}$=0 kullanıldığında $\overline{\text{DEN}}$ DT/$\overline{\text{R}}$ IO/$\overline{\text{M}}$ uçları S0, S1, S2 olarak davranır. INTEL'in 8288 veriyolu denetleci bu üç sinyali şu denetim sinyallerini üretmek üzere çözer:



Şekil 8.17: 40 uçlu 8088 yongasının maximal moddaki uçları

Tablo 8.4: 8288 veriyolu denetleci çıkışları

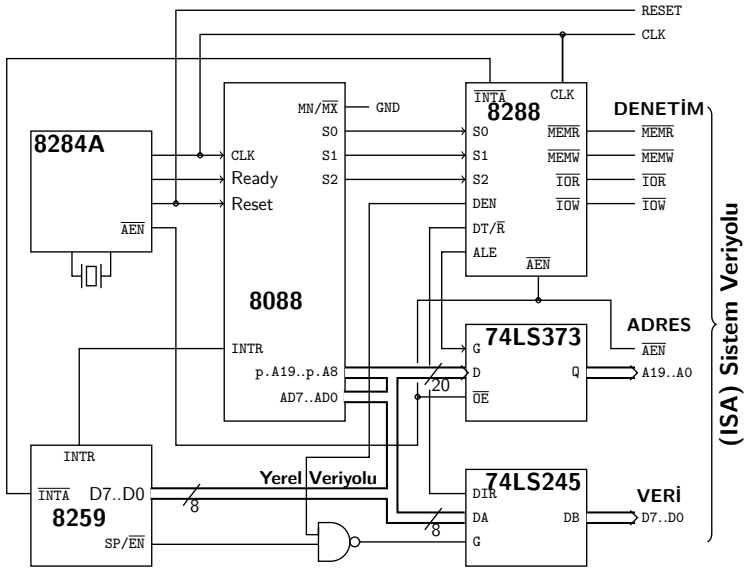
S2 S1 S0	İşlemci Durumu	8288 çıkışı	eski adı
0 0 0	kesme onayı	$\overline{\text{INTA}}$	$\overline{\text{INTA}}$
0 0 1	i/o port okuma	$\overline{\text{IORC}}$	$\overline{\text{IOR}}$
0 1 0	i/o port yazma	$\overline{\text{IOWC}}$	$\overline{\text{IOW}}$
0 1 1	halt (dur)	-yok-	.
1 0 0	bellek kod okuma	$\overline{\text{MRDC}}$	$\overline{\text{MEMR}}$
1 0 1	bellek veri okuma	$\overline{\text{MRDC}}$	$\overline{\text{MEMR}}$
1 1 0	bellek veri yazma	$\overline{\text{MWTC}}$	$\overline{\text{MEMW}}$
1 1 1	edilgin	-yok-	.

Maksimal modda 8288 tablodaki veri yolu denetim çıkışlarına ek olarak S0,S1,ve S2 'nin değişiminden işlemcinin minimal modda çıkardığı DT/$\overline{\text{R}}$, $\overline{\text{DEN}}$, ve ALE çıkışlarını da üretir. Başlangıçta güçlendirilmiş

adres ve veri hatlarıyla birarada IBM PC sistem veriyolu oluşturan bu sinyaller daha ileride ISA veriyolu adıyla standartlaşmıştır.

8.9.1 8088 IBM-PC ve 8-bit ISA veriyolu

8088 işlemcisi IBM-PC de kullanılmaya başlandığında veri yolu denetim sinyalleri Şekil 8.4'deki devreyle üretildi. IBM-PC ana kartında $\overline{\text{IOW}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$ ve $\overline{\text{MEMR}}$ denetim sinyalleriyle Adres ve Veri hatlarından oluşan bir dizi bağlantı çeşitli arayüz kartlarının baskılı devresinin kenarındaki iletken suyollarına uyan bir dizi sokette toplanıyordu. Bu yapı ilk 8-bit sistem veriyolunu oluşturdu. Ardından IBM-PC 8288 veriyolu denetlecini kullanmaya başlayınca Şekil 8.18'de görülen sistem veriyolu ISA standardı olarak benimsendi. Daha da geliştirilen 80286 işlemcili PC-AT lerde modellerinde de aynı denetim sinyalleri farklı devrelerle üretilmeye başlanınca daha farklı isimlerle kullanılmaya başlandı. ISA veriyolunun temel özellikleri arasında tüm adres ve veri hatlarının tamponlarla güçlendirilmesi, belleğe yazma ve okuma ile porta yazma ve okuma için ayrı ayrı düşük-etkin denetim sinyalleri kullanılması, veriyoluna bağlanacak arayüzlerin kesme gereksiniminin karşılanması gibi birçok yönünü sayabiliriz.



Şekil 8.18: 8088'in ISA standard veriyolu arayüzü

8.10 80286 ve 16-bit ISA veriyolu

Adres genişliğini 24-bite çıkaran 80286 işlemcisi gerçel modunda 8088 ve 8086 gibi 20 adres biti kullanır. Diğer adres bitleri ise Korunmuş modda kullanılır. Biz Gerçel moduyla ilgileneceğiz.

8086 gibi, 80286 işlemcisi de 16-bitlik veri yoluna sahip olmasına karşın belleğe bayt adresleriyle ulaşır. 16-bit bellek genişliğini bank-L ve bank-H olarak adlandırılan iki bellek grubu ile sağlar. Bank-L veri yolunun D7..D0 bitlerine veri sağlar ve 0, 2, 4 .. gibi A0 biti düşük sıfır olan adresledrdeki baytlar için kullanılır. işlemcinin D15...D8 veri bitlerine bağlanan Bank-H ise 1, 3, .. gibi A0 biti yüksek adreslerin bulunduğu baytlara erişir.

Işlemci bütün okumaları 16-bit yaparak adres-veri arayüzündeki tampona yerleştirir. Bunun için Bank-L adresini kullanır. Ancak yazılacak veri bir bayt ise ve örneğin Bank-L 'de ise 16-bit yazacak olsa Bank-H deki bayta istenmeyen veri yazılmasını önlemek üzere $\overline{\text{BHE}}$ olarak adlandırılan çıkışını kullanır.

$\overline{\text{BHE}}$	A0	Yazılan baytın adresi
0	0	16-bit. Bank-H ve Bank-L
0	1	8-bit. Bank-L, tek adres
1	0	16-bit. Bank-H, çift adres
1	1	veriyolu kullanılmıyor

16-bit ISA veriyolu hem veri ve adres uçlarının artması, hem de yeni denetim sinyalleri nedeniyle eski soketi yetmeyince soketin yanına konan ek soket ile eski 8-bit ISA veriyoluna uyumlu olarak tasarlandı. 16-bit ISA veriyoluna SD8..SD15 tamponlanmış sistem veri uçları, LA23..LA20 adres uçları, ve $\overline{\text{BHE}}$ sinyaline ek olarak çeşitli amaçlarla kullanmak üzere kesme girişleri de eklenmiştir.

8.11 80286 ve 16-bit ISA veriyolu

Adres genişliğini 24-bite çıkarırken 16 bit veri genişliğini koruyan 80286 işlemcisi gerçel modunda 8088 gibi 20 adres biti kullanır. Diğer adres bitleri ise Korunmuş modda kullanılır. Biz gerçel moduyla ilgileneceğiz.

80286 işlemcisi 16-bitlik veri yoluna sahip olmasına karşın belleğe bayt adresleriyle ulaşır. 16-bit bellek genişliğini Bank-L ve Bank-H olarak adlandırılan iki bellek grubu ile sağlar. Bank-L veri yolunun D7..D0 bitlerine veri sağlar ve 0, 2, 4 .. gibi A0 biti düşük sıfır olan adresledr-

deki baytlar için kullanılır. işlemcinin D15...D8 veri bitlerine bağlanan Bank-H ise 1, 3, .. gibi A0 biti yüksek adresteki bayta erişir.

Işlemci bütün okumaları 16-bit yaparak adres-veri arayüzündeki tampona yerleştirir. Bunun için Bank-L adresini kullanır. Ancak yazılacak veri bir bayt ise ve örneğin Bank-L 'de ise 16-bit yazacak olsa Bank-H deki bayta istenmeyen veri yazılmasını önlemek üzere $\overline{\text{BHE}}$ olarak adlandırılan çıkışını kullanır.

$\overline{\text{BHE}}$	A0	Yazılan baytın adresi
0	0	16-bit. Bank-H ve Bank-L
0	1	8-bit. Bank-L, tek adres
1	0	16-bit. Bank-H, çift adres
1	1	veriyolu kullanılmıyor

16-bit ISA veriyolu hem veri ve adres uçlarının artması, hem de yeni denetim sinyalleri nedeniyle eski soketi yetmeyince soketin yanına konan ek VESA soket ile eski 8-bit ISA veriyoluna uyumlu olarak tasarlandı.

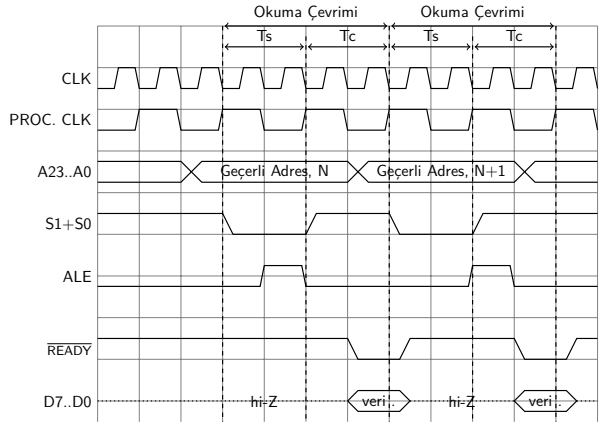
ISA veriyolu günümüzde 32-bit ve daha geniş bellek yapılarında kullanılan multimediaya uymaması ve 32-bit ISA (EISA) bu yapılar için yavaş kalması nedeniyle yerini 1990'lı yıllarda Peripheral Component Interconnect (PCI) veriyoluna bıraktı. Video araçların arayüzünde kullanılan VESA (VL-veriyolu) ile ses ve diğer giriş çıkışı sağlayan EISA standardını biraraya getiren PCI veriyolu grafik ve ses adaptörlerine erişim sağlar. Günümüzde işlemcinin yerel veriyolu yan-işlemci, merkezi işlem birimi, ön-bellek[3] ve ana-bellek erişiminde kullanılırken PCI köprüsü ve veriyolu üzerinden ISA/EISA veriyollarına, LAN, SCSI, ve Grafik adaptörlerine ve diğer giriş/çıkış elemanlarına erişir.

Günümüzdeki işlemcilerin yonga içindeki ön-belleğe daha yüksek hızlarda erişmesi için tasarlanan veri yolu arkabölge veri yolu olarak adlandırılırken ana belleğe erişimi amaçlayan yerel veri yoluna önbölge veriyolu denir. PCI veriyolu, önbölge veriyoluna 47 uçlu bir soket üzerinden bağlanan bir köprü yongası ile türetilir. 32 ve 64 bit genişlik sağlayan PCI saniyede bir Giga bayta kadar çeşitli hızlarda çalışabilir. Bu hızlara çıkabilmesi için PCI veriyolu aynı iletim çerçevesinde bir adresi takiben ard arda pek çok veriyi iletebilir yapıda geliştirilmiştir.

[3] cache memory

8.12 80286 okuma çevrimi

80286 işlemcisi sadece maksimum modda çalışır, ve 8259A kesme denetleci, 82C284 saat üreteci ve 82C288 veriyolu denetleci sistemin gerekli sinyallerini oluşturur. İşlemci bellek ve giriş/çıkış veriyolu çevrimlerinin evresini M/$\overline{\text{IO}}$S0 ve S1 çıkışlarıyla 82C288 veriyolu denetlecine bildirir. 82C288 ALE, $\overline{\text{RD}}$, $\overline{\text{WR}}$, DT/$\overline{\text{R}}$ ve DEN çıkışlarını sağlar. Buna ek olarak işlemcinin ve girişleri okuma/yazma çevrimlerini uzatmak üzere $\overline{\text{READY}}$, ya da çevrimler arasında veriyolunu kullanacak diğer devrelere zaman tanımak için CMDL (Komut Geciktirme) girişleri bulunur.



Şekil 8.19: 80286 Okuma çevrimi

80286'nın hiç beklemesiz okuma çevrimi Tc ve Ts olarak adlandırılan iki saat çevriminden oluşur. 80286, okunacak bellek adresini bir önceki okuma çevriminin ikinci yarısında vermeye başlar. Böylece adres mandallarının girişinde hazır duran adres Tc çevrimi başında $\overline{\text{MEMR}}$=0 olunca sistem adres yoluna geçer ve adres çözücü bu adresi barındıran yonganın $\overline{\text{CS}}$ girişini etkinleştirir. Tc içinde işlemci $\overline{\text{RD}}$=0 DEN=1 ve DT/$\overline{\text{R}}$=0 vererek belleğin $\overline{\text{OE}}$ girişine $\overline{\text{MEMR}}$=0 gitmesini sağlar. Bellekteki adres çözümleme süresi sonunda bellek adresteki yazmacın içeriğini veri yoluna yollar. İşlemci Ts çevriminin sonunda önce $\overline{\text{READY}}$ girişine bakar. Ts sonunda $\overline{\text{READY}}$=0 ise bellekten gelen veriyi gereken yazmaca aktararak okuma çevrimini tamamlar.

Bellek, veriyi Ts sonuna kadar hazır edemezse bellek sistemindeki bekletme denetim devresi, veri hazır oluncaya kadar işlemcinin $\overline{\text{READY}}$ girişine 1 vererek işlemciye belleğin henüz hazır olmadığını bildirir.

İşlemci Ts sonunda $\overline{\texttt{READY}}$=1 ise veriyi yazmaçlara aktarmaz ve yeni bir Ts çevrimine başlar. Verinin okunmadığı Ts çevrimlerine bekleme çevrimi denir ve Tw ile gösterilir.

İlerideki kısımlarda bellek ve giriş/çıkış devreleri üzerine tasarımlar ve çözümlemeler yaparken devrelerimizi standard ISA-veriyolu sinyallerine göre oluşturmayı tercih edeceğiz.

Bölüm 9

Bellek alt sistemi

Bir mikroişlemci sisteminde bellek hem program kodunu hem de verileri saklamak için gerekir. Bellek alt sistemi, elimizdeki mevcut bellek yongalarını mikroişlemci sistemine gereken genişlik ve çeşitte bir bellek yapısı oluşturmak üzere bir araya getirmeyi amaçlar.

Bellek yongaları veriyi bir dizi adreslenebilir yazmaçta saklar. Adresler sıfırdan başlayarak numaralanır ve yazmaç sayısı genellikle ikinin üstü kadardır. Adres girişleri adres numarasını ikili sistemde kabul eder. Örneğin 0 dan 255'e toplam $256 = 2^8$ yazmacı olan bir belleğin 255 sayısını ikili sistemde yazmaya yetecek sayıda (=8) adres giriş ucu bulunur. Her bir yazmaçta saklanan bit sayısına belleğin veri genişliği denir. Belleklerin veri genişliği genellikle bir ile 64 bit arasında, ikinin tam üssü, ya da bir fazlasıdır. Tipik veri genişlikleri arasında 1, 4, 8, 9, 16, 17 bit olanlarını sayabiliriz.

Ikili sistemde bilgi birimi bittir, ve bellek kapasitesi bit ile ölçülür. Örneğin 8 bitlik 256 yazmacı olan bir belleğin kapasitesi 256×8 bit olarak ifade edildiğinde belleğin hem yazmaç düzeni, hem de kapasitesi belirtilmiş olur. Aynı belleği 256 sözcüklü olarak betimlemek te mümkündür.

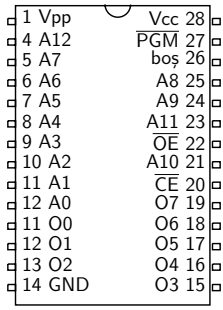
Örnek 9.1.

2764 yongasında toplam 8 veri ve 13 adres ucu bulunur. Bellek kapasitesini bulup yazmaç düzenini ve kapasitesini ayrı ayrı ifade edelim.

Çözüm: 13 adres ucu sayesinde $2^{13} = 8$k sözcük adreslenebilir. Sekiz veri ucu bulunduğuna göre her sözcük sekiz bit, ya da bir bayt genişliktedir. O halde yazmaç düzeni 8k$\times 8$ bit, ya da 8k bayt olarak yazılır. Kapasiteyi bit olarak yazmalıyız. 8k$\times 8$ bit $= 8 \times 8$k bit$=64$ k bit eder.

9.1 Bellek kapasitesi birimi

Ikili sistemde verinin en küçük kapasite ölçüsü bit'tir. Ancak bir bit ancak iki durumdan birini, örneğin evet ya da hayırı, veya örneğin



Şekil 9.1: 2764 EEPROM yongasının uçları

var ya da yoku gösterebilir. Çoğu durumda kapasite bit sayısıyla okunduğunda sayı aklımızda kolayca kalamayacak derecede yüksek olur. Bir kilo bit, ikinin onuncu üstü olan 1024 bit için kullanılır. Örneğin $2^{13} = 8192$ biti $2^3 \times 2^{10} = 8$kilo bit olarak ifade edilir. Daha da büyük birim gerekirse $2^{20} = 1024 \times 1024$ bit$= 1048576$ bit $= 1$ Mega bit, ve $2^30 = 1024 \times 1024 \times 1024 = 1$ Giga bit, ve $2^40 = 1024 \times 1024 \times 1024 \times 1024 = 1$ Tera bit birimleri kullanılır. Bu birimlerin yanısıra, 8-bit veri genişliğindeki belleklerin kapasitesi yaygın olarak bayt, kilo bayt, mega bayt ve tera bayt cinsinden verilir.

Adres ucu sayısına bağlı olarak belleğin yazmaç sayısı kolayca bulunabilir. Tablo 9.1'de 1 k sözcükten 1 M sözcüğe kadar gereken adres ucu düzeni verilmiştir. Adres uçları her zaman sıfırdan başlanarak numaralandığından adres uç sayısı en son adres ucunun numarasından bir büyüktür. Gene bu tablodan görüldüğü gibi adres ucu sayısı 10 ile 19 arasındaysa adreslenen sözcük sayısını kilo sözcük olarak göstermek daha kullanışlıdır.

Örnek 9.2.

Adres uçları A11..A0, veri uçları D7..D0 olarak markalanmış belleğin kapasitesini ve veri düzenini bulunuz.

Çözüm: Toplam 12 adres ucu, 8 veri ucu var. Kapasitesi bit olarak $2^{12} \times 8$=32kbit, bellek düzeni ise $2^{12} \times 8$=4k×8 bayttır. Bellek düzeni kısaca 4kbayt olarak ta ifade edilebilir.

Adres Uçlarının Adları	Uç sayısı	Sözcük Kapasitesi
A0 ... A9	10	1 k
A0 ... A10	11	2 k
A0 ... A11	12	4 k
A0 ... A12	13	8 k
A0 ... A13	14	16 k
A0 ... A14	15	32 k
A0 ... A15	16	64 k
A0 ... A16	17	128 k
A0 ... A17	18	256 k
A0 ... A18	19	512 k
A0 ... A19	20	1 M

Tablo 9.1: Adres ucu sayısı ve sözcük kapasitesi

9.2 Karakteristik bellek değişkenleri

Bellek tiplerine bağlı olarak belleklerin çeşitli karakteristik özellikleri ön plana çıkabilir. Genel olarak en önem kazanan özellik belleğin okuma ve yazma hızıdır.

Okuma hızı, adres çözümleme (erişim) süresiyle veri transfer süresinin bileşimidir. Yazma hızı ise, erişim süresi ve veri yazma süresinden oluşur. EEPROM gibi bazı tip belleklerin veri yazma süresi okuma süresinden milyon kat uzun olabilir. EEPROM bellek hücreleri kullanmasına karşın Flaşbellekler yazma hızını yazılacak veriyi önce geçici olarak tampon RAM belleğe yazıp daha sonra tamponun tümünü aynı anda EEPROM hücrelere geçirir. Böylece örneğin 1024 biti aynı anda yazınca bit başına yazma süresi bir bit yazma süresinin binde birine düşer.

Bellek erişim hızının yanısıra belleğin bit başına harcadığı güç te önemli bir parametredir. CMOS SRAM bellekler bit başına çok az enerji harcasa da çok büyük kapasitelerde erişim süresi ve maliyeti yükseldiği için tercih edilmez. Dinamik belleklerin özellikle yüksek kapasitelerde hem erişim süresi hem de bit başına harcadığı güç düşer. Bu nedenle ana bellek sistemi tasarımında tercih edilirler.

EEPROM belleklerde daha önce belirtildiği gibi yazma süresi önem kazanır. Tipik yazma süresi birkaç milisaniye civarı olan EEPROM bellekler tampon bellek kullanarak bit başına yazma süresini düşürse de EEPROM bellek hücrelerine uygulanacak yazma çevrimi sayısı sı-

nırlıdır. Bu parametre günümüzde Flaşbellek yazma çevrimi sayısı bir milyonu aşabilse de tasarımda flaşbellek kullanımında önemli bir sınırlamaya neden olur.

9.3 Bellek çeşitleri

Istenen adresine doğrudan okuma ve yazma amacıyla erişilebilen belleklere rastgele erişimli bellek denir ve RAM kısaltmasıyla gösterilir. Flipflop yazmaçlar ve adres çözümleme devresinden oluşan belleklere statik RAM (kısaca SRAM) adı verilir. Veriyi kapasitörde yük olarak tutan, bu yüzden her milisaniyede bir yükü zayıflayan kapasitörlerin yükünü yenilemeyi gerektiren belleklere de dinamik bellek (DRAM) denir.

Bilgisayar sistemlerinde hem okunur hem yazılır RAM belleklerin yanısıra yalnız okunur (ROM) belleler de gerekir. En hızlı okunabilen ROM bellekler bir adres çözümleyicinin gereken çıkışlarını diyotlarla veri çıkışına bağlayan maskeli ROM'lardır. Bunlara veri üretimde bağlantı maskesi vasıtasıyla yerleştirilir ve bu veri asla silinemez.

Gene maskeli ROM bellek gibi bir dekoder çıkışına bağlı diyotlardan oluşan, ancak zayıf metalize bağların özel bir programlayıcının sağladığı yüksekçe bir gerilimle kalıcı olarak kopartılabildiği bir sefer programlanabilen bellekler de vardır. Küçük kapasiteli bu bellekler OTP ROM, ya da kısaca PROM olarak adlandırılır ve genellikle donanımın gerektirdiği karakter tablosu, ya da adres çözücü gibi özel ikili işlevlerle programlanır. Örneğin 8k $\times$8 bit PROM yongasının adres erişim süresi T_{rc}=100ns civarıdır.

EEPROM olarak adlandırılan bellekler kapısına özel yük tutan bir izole bölge sayesinde ancak yüksekçe bir programlama gerilimiyle istenen veriyi saklar. Okumak için 200 ns gerektiren bir EEPROM'da veri silmek ve yeniden veri yazmak 5 ile 20 ms arasında zaman alabilir. Aynı adrese en fazla 1 000 000 kez yazılabilir.

Flaş ROM bellek bir EEPROM olmasına rağmen veri yazma hızını arttıran kapasitesi 256 bayt ile 8k bayt arasında olabilen yazma tampon belleği sayesinde yazma süresi mikrosaniyeler seviyesine indirir. Gigabaytlar seviyesindeki bellek kapasitelerine ulaşmak mümkündür ve adres çözümleyicinin özel düzeni sayesinde T_{rc}=200ns civarı erişim süresi elde edilir. Flaş ROM belleklere de üst üstr en fazla 1000000 kere yazılabilir.

Tablo 9.2: Çeşitli ROM yongaları

UV-EPROM	hız(ns)	kapasite(bit)	Düzen	uç sayısı	V_{pp}
2716	450	16k	2k×8	24	25
2732A-20	200	32k	4k×8	24	21
27C64-15	150	64k	8k×8	28	12.5
27128-20	200	128k	16k×8	28	12.5
27C256-20	200	256k	32k×8	28	12.5
27C512-25	250	512k	64k×8	28	12.5
27C010-12	120	1M	128k×8	32	12.5
EEPROM Adı	hız(ns)	kapasite(bit)	Düzen	uç sayısı	V_{pp}
28C16A-25	250	16k	2k×8	24	25
28C64A-15	150	64k	8k×8	28	12.5
28C256-15	150	128k	16k×8	28	12.5
Flaş ROM	hız(ns)	kapasite(bit)	Düzen	uç sayısı	V_{pp}
28F256	120	256k	32k×8	32	12
28F512	120	512k	64k×8	28	12
28F010	150	1M	128k×8	32	12
28F020	150	2M	256k×8	32	12

9.3.1 Sık kullanılan ROM bellek yongaları

Mikro işlemci devrelerinde yaygın kullanım alanı bulan bellek yongaları başlıcaları üç grupta toplanır. Örneklerimizde yer alacak ROM yongalarının çeşitli özellikleri Tablo 9.2'da verilmiştir.

EPROM yongaları: genellikle 27xxx olarak kodlanır. EPROM'lar 16k ile 4M bit aralığında kapasiteye sahiptir.

EEPROM yongaları: genellikle 28Cxxx olarak adlandırılır. Yalnızca 16k, 64k ve 128k kapasitelerde üretilir.

FlaşROM yongaları: 28Fxxx olarak kodlanır.

9.4 Bellek erişim ve denetim sinyalleri

9.4.1 Adres ve veri uçları

Belleklerin 2^n yazmaç adreslemek üzere A_0 ... A_{n-1} olarak adlandırılan toplam n adres ucu bulunur. Örneğin 2k yazmaç adresleyen 2716'da 2^{11}=2k yazmaç adreslemek üzere $n = 11$ olduğuna göre A0 ... A10 olarak adlandırılmış 11 adres ucu vardır.

ROM belleklerin veri uçları yalnızca çıkış için kullanıldığından genellikle O0 .. O7 gibi O harfiyle işaretlenir. RAM belleklerde veri uçları

okuma çevriminde veri çıkışı, yazma çevriminde ise veri girişi amacıyla kullanılır.

9.4.2 Bellek denetim sinyalleri

RAM ve ROM belleklerin denetim sinyalleri genellikle düşük-etkindir. Bellek yongası $\overline{\texttt{CS}}$ ya da $\overline{\texttt{CE}}$ girişi düşürülerek seçilir, ve seçilen yonga adres çözümlemeye hazır duruma girer, ancak veri uçları giriş olarak kalır. Seçilen belleğin güç harcaması seçilmeyen belleğe göre tipik olarak on kattan fazla artar. $\overline{\texttt{OE}}$ çıkış-etkinle girişi düşükse veri yolu çıkış durumuna geçer ve seçilen adresteki veriyi çıkarır. Ancak, adres çözücü T_{aa} süresi dolmadan henüz adresi seçemediğinden çıkışlara rastgele değerler gider. RAM yongalarda, $\overline{\texttt{CE}}$=0 ile seçtikten sonra, $\overline{\texttt{OE}}$'yi düşürmeden yazılacak adresi ve veriyi belleğin adres ve veri uçlarına yollayıp adres çözümleme süresi dolduktan sonra $\overline{\texttt{WE}}$'yi düşürüp yükselterek veri bellekte uygulanan adrese yazdırılır.

9.4.3 Statik RAM, Rasgele erişimli bellekler

Statik RAM bellekler veri hücresi her bit için bir flip-flop devresi kullanılarak oluşturulmuş büyük bir bellek matrisi ile adreslenen yazmacı seçen bir adres-çözücü birimden oluşur. Genellikle 8-bit veri genişliğine sahip olan bu bellekler küçük kapasitelerde yüksek hızlara çıkabildikleri için ön-bellekte kullanılırlar. Basit yapıları ve denetim birimi gerektirmemesi nedeniyle düşük-uç uygulamalarda veri belleği gereksinimi için de tercih edilirler.

Numarası	hızı(ns)	Organizasyon	uç sayısı
6116-1	100	2k×8-bit	24
6264-10	100	8k×8-bit	24
62256-10	100	32k×8-bit	28

Tablo 9.3: Çeşitli statik RAM yongalar

9.4.4 Dinamik bellek

Dinamik RAM (DRAM) yongalar, sakladıkları veriyi veri hücresindeki kondansatörün yükü olarak saklar. Statik RAM yongalarının bir

bit veri saklayabilen 10-transistörlü flip-flop devreleriyle karşılaştırıldığında DRAM hücre bit başına veriyi saklamakta beş kat daha az yonga alanı kullanır ve bu nedenle özellikle yüksek veri kapasitelerinde tercih edilir

DRAM veri hücreleri iki boyutlu bir dizi oluşturacak şekilde düzenlenerek satır ve sütun adresi ile seçilir. Seçilen biti okuyan veri duyacı devresi eğer bit bir ise hücre kapasitörünü tekrar yükler. Hücre düzeninin doğal matrix yapısı nedeniyle adres çözümleme, satır-adresi ve sütun-adresi olmak üzere iki adres çözücüye dağıtıldığından özellikle yüksek kapasitelerde aynı kapasiteli SRAM yongaya göre daha hızlı çalışır.

DRAM yonganın en büyük dezavantajı, hücrelerde bit değerini yük olarak tutan kapasitelerin 1ms gibi kısa sürede boşalmasıdır. Bilgi kaybını önlemek üzere bütün hücrelerdeki verinin 1 ms den önce okunarak yeniden yüklenmesi gerekir. Her kolon için bir duyaç devresi bulunduğundan bellegin tümündeki veriyi yenilemek için satır adreslerinin tümünü en azından bir ms içinde tarayıp her bir satır adresi için ya okuma ya da yenileme işlemi yapmak gerekir. Gerek işlemciden gelen adresi satır ve sütun adreslerine ayırmak, gerek süreyi aksatmadan veri yenileme işlemini yapabilmek üzere DRAM yongalar *DRAM bellek denetim birimi*nin denetiminde kullanılır ve sisteme denetim biriminin bu iş için oluşturduğu soketlere takılarak bağlanır.

DRAM yongaların uç sayısını ve böylece maliyetini azaltmak üzere satır ve sütun adresleri ortak çoklanır. Örneğin, 64k$\times$1 bit olarak düzenlenmiş 4464 yongasında satır ve sütun adreslerini girmek üzere yalnızca 8 adres ucu (A7..A0) bulunur. $\overline{\text{RAS}}$ (satır adres seçici) ve $\overline{\text{CAS}}$ (sütun adres seçici) olmak üzere iki denetim girişi bulunan bu yonganın veri giriş ve veri çıkış olmak üzere iki veri ucu vardır. $\overline{\text{RAS}}$ düştüğünde A7..A0 girişlerindeki adresi A7..A0 satır adres biti olarak mandallar. $\overline{\text{RAS}}$ düşükken $\overline{\text{CAS}}$'ın düştüğü anda gene A7..A0 girişlerinden gelen 8-biti A15..A8 olarak kolon adresi çözümleyiciye aktarır. Seçilen hücre belirlenince eğer $\overline{\text{WE}}$ düşükse veri girişindeki bitler adreslenen hücreye aktarılır. Eğer $\overline{\text{OE}}$ ucu düşükse hücredeki veri yonganın veri çıkışına aktarılır.

DRAM önce $\overline{\text{RAS}}$ yerine $\overline{\text{CAS}}$ düşürülerek çalıştırıldığında veri çıkışını etkinleştirmez ve peşpeşe verilen adres ve $\overline{\text{RAS}}$ darbeleri için veri tazeleme işlemi yapabilir. Bir satır adresi sayacında üretilen adresle belleğe erişim yapılmayan çevrimlerde birden fazla satır yenilemek mümkün olur.

Örnek 9.3.

64k×1-bit DRAM belleğin kaç adres, ve kaç veri ucu vardır.

Çözüm: 64k=2^{16} olduğuna göre 64k için 16 adres biti gerekir. Ancak DRAM bu adres bitlerinin yarısını satır, diğer yarısını sütun adresi olarak kullandığından $16/2 = 8$ adres ucuna sahiptir. Birer bitlik düzende olduğundan bir veri ucunu veri giriş ve veri çıkış ucu olarak kullanır.

Örnek 9.4.

128k×4-bit DRAM belleğin kaç adres, ve kaç veri ucu vardır.

Çözüm: 128k=2^{17} olduğuna göre 128k için 17 adres biti gerekir. Yarısını satır, yarısını sütun adresi olarak kullanan DRAM'ın $17/2 = 9$ (tamsayı bölme) adres ucu bulunur. Dörder bitlik düzende olduğundan dört veri ucunu hem veri giriş hem de veri çıkış için kullanır.

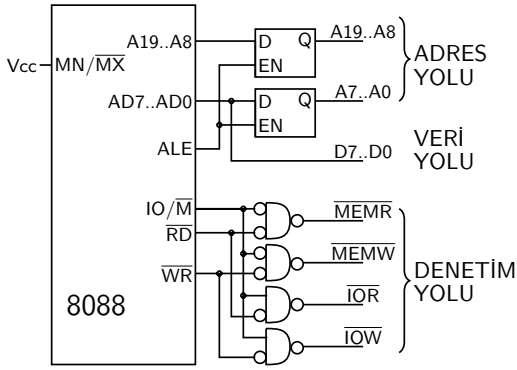
Numarası	hızı(ns)	Organizasyon	uç sayısı
4116-20	200	16k×1-bit	16
4164-15	150	64k×1-bit	16
41464-8	80	64k×4-bit	18
41264-6	60	256k×1-bit	16
414264-10	100	256k×4-bit	20

Tablo 9.4: Çeşitli DRAM yongalar

DRAM yongaların genellikle veri genişliği bir yada dört bit, adres genişliği ise 7 ile 12 bit arasındadır. Bir bitlik yongaların sekiz tanesi birlikte kullanılarak 8-bit bellek elde edilir. IBM-PC'de bu sekize ek olarak kullanılan dokuzuncu bite sekiz veri bitinin paritesi yazılır. Veri okunduğunda paritesi uyuşmazsa bir kesme ile işlemci durdurulur. Böylece kesmenin dönüş adresinden hangi belleğin bozulduğu kolayca anlaşılır.

9.5 Bellek Genişletme ve Adres Çözücü

8088 minimal sistemin adres çözücü devreleri ile sınırlı kalacağımızdan adres çözücü örneklerimizi aşağıdaki adres, veri ve denetim yolları üzerinde vereceğiz.



Bellek sistemi genellikle birden fazla bellek yongası, bunların bekleme çevrimlerine göre işlemciye READY üreten bekleme denetim devresi, ve bellek yongalarının birarada çalışmasını sağlayacak devrelerden oluşur. Bellek yongalarını

1. Bellek yongasının veri genişliği işlemci veri yoluna göre yetersiz ise veri genişliğini arttırmak üzere,
2. Belleğin adres genişliği sistemde istenen adres aralığını dolduramıyorsa adres genişliğini arttırmak üzere.

iki farklı amaçla birarada kullanabiliriz.

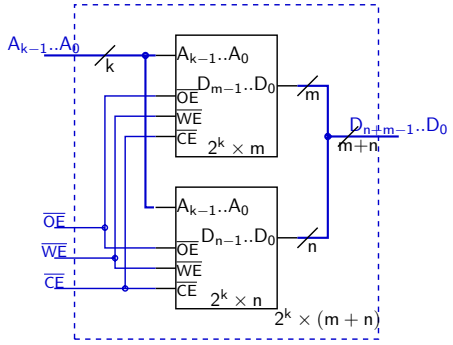
9.5.1 Veri genişletme

Adres genişliği birbirine eşit belleklerin veri genişliğini arttırmak üzere belleklerin bütün adres uçları a0'ı a0'a, a1'i a1'e bağlamak üzere birbirine bağlanır. Belleklerin $\overline{\text{CE}}$, $\overline{\text{OE}}$, ve $\overline{\text{WE}}$ denetim uçları da aynı biçimde birbirine bağlanarak yongaların aynı anda aynı adresten okuma ve yazması sağlanır. Veri hatları ise bir yonganınki alt bölümü, diğeri ise üst bölümü oluşturacak biçimde bağlanır. Bu şekilde bağlanan $2^k \times m$ bitlik ve $2^k \times n$ bitlik iki bellek, veri genişliği $m + n$ bitlik bir belleğe dönüşür.

Örnek 9.5.

4k$\times$ 4 bitlik 2 bellek kullanarak 4-bit yerine 8-bit veri genişliğinde bellek birimi oluşturalım.

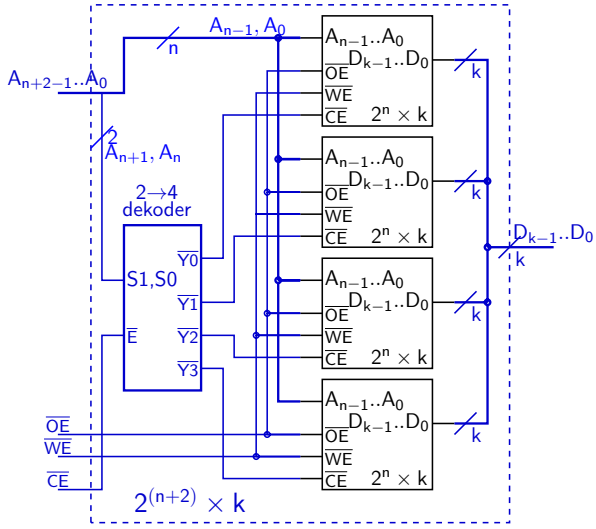
Çözüm 4k=2^{12} olduğundan k=12, m=4, n=4 oluyor. Belleklerin aynı anda aynı adresi okuma ya da yazma için kullanabilmesi için adres ve denetim uçlarını diğerinin aynı isimli adres ve denetim ucuna bağlamalıyız. Bellek veri uçlarını küçük harfle gösterirsek birinci belleğin b3..b0 uçlarını B3..B0'a bağlarken ikinci belleğin b3..b0 uçlarını B7..B4'e bağlarsak iki yonga aynı anda çalışarak 8-bit veriyi saklayıp okur.



Şekil 9.2: Belleklerin veri genişletmek üzere birleştirilmesi

9.5.2 Adres genişletme

Elimizdeki belleklerin veri genişliği işlemcimize uyumlu olsa da sistemdeki bellek kapasitesinin bellek yongasından daha geniş olması için bellekleri adres genişletmek üzere biraraya getirme yöntemine adres genişletme denir.



Şekil 9.3: Belleklerin adres genişletmek üzere birleştirilmesi, p=4, k=2 ve m=2 durumu.

Toplam n adres ucunu A_{n-1} .. A_0 olarak, ve k adet veri ucunu $D_{k-1}..D_0$ ile gösterdiğimiz p=2^m tane bellek yongamız varsa adres

genişletme yöntemiyle bu belleklerden A_{n+m-1} .. A_0 ile gösterdiğimiz $n+m$ adres ucuna ve gene k veri ucuna sahip bir bellek elde ederiz.

Örneğin elimizde dört adet 4k× 8-bit bellek yongası olsa, ama sistemde 16k× 8-bit bellek gerekse, 4=2^2 olduğuna göre p=4, m=2, ve bellekler 4k adreslediğine göre 4k=2^{12} olduğundan yonganın adres genişliği n=12, veri genişliği ise k=8 bittir. Belleklere A11..A0 olmak üzere toplam 12 adres hattı bağlanır. Adres genişletirken 4 yongadan birini etkinleştirmek üzere iki seçme-girişi ve dört etkinleştirme çıkış olan bir dekoder kullanırız. Dekoderin etkinleştirme girişi etkin değilse hiçbir çıkışı da etkinleşmediğine göre amaçlanan 16k baytlık belleğin $\overline{CE}$ sinyali olarak kullanabiliriz. Dekoderin seçme girişlerini sistemin A13,A12 adres hatlarına bağlayarak bu adreslerin durumuna göre dekoderin A13,A12 =00 olduğunda ilk belleği, A13,A12 = 0,1 olduğunda ikinciyi, ve böyle giderek A13,A12 =1,1 olduğunda dördüncüyü etkinleştirmesini sağlayalım. Bunun için dekoderin çıkışlarından $\overline{Y0}$ 'ı birinci belleğin $\overline{CE}$ girişine, $\overline{Y1}$'i ikinci belleğin $\overline{CS}$'sine ... $\overline{Y3}$'ü dördüncününkine bağlarız. Artık bir anda belleklerden yalnızca bir tanesi çalışır ve belleklerden hangisinin çalışacağını adresteki A13,A12 bitleri belirler. Böylece dört bellek adres uzayında peşpeşe 4k-baytlık 4 alanı doldurur.

Örnek devrede dekoder kullanmış olsak ta dekoderin işlevi
$\overline{OE}$, A13,A12=0,0,0 olduğunda Y3, Y2, Y1, Y0= 1110,
$\overline{OE}$, A13,A12=0,0,1 olduğunda Y3, Y2, Y1, Y0= 1101,
$\overline{OE}$, A13,A12=0,1,0 olduğunda Y3, Y2, Y1, Y0 = 1011,
$\overline{OE}$, A13,A12=0,1,1 olduğunda Y3, Y2, Y1, Y0= 0111, ve
$\overline{OE}$ =1 olduğunda ise Y3,Y2,Y1,Y0= 1111
çıkarmaktır. Bu iş bir VE-VEYA devresiyle kolayca gerçekleşir. Adres çözümleme amacıyla VE-VEYA devresi, dekoder yongaları, veya küçük kapasiteli hızlı ROM bellek yongalarından biri kullanılır. VE-VEYA devreleri özellikle gelişen teknolojiyle birlikte günümüzde yaygın kullanım alanı bulan PLC ve FPGA yongalarının dekoder olarak programlanmasında gerekir.

9.6 Adres Çözücü devreler

Adres çözümleme tasarımlarında en kullanışlı yöntemlerden biri adres tablosu, ya da diğer adıyla adres haritası kullanmaktır. Adres tablosunun en üst satırı çözümlemede gereken sistem adres bitleridir. Örnegin 13 adres uçlu 8k×8bit'lik (6264) dört belleği 20-bit adres yolu

olan bir sisteme 00000h adresinden başlayarak yerleştirmek istersek, ilaveten, 8k×8bitlik bir EEPROM'u da son adresini 0FFFFh adresine rastlayacak biçimde kullanmayı amaçlasak donanımdaki bağlantıları tanımlayan adres haritasını aşağıdaki gibi hazırlarız.

sistem	a19	a18	a17	a16	a15	a14	a13	a12	a11-a0	hex adres
RAM_a	0	0	0	0	0	0	0	0	000_h	00000
	0	0	0	0	0	0	0	1	FFF_h	01FFF
RAM_b	0	0	0	0	0	0	1	0	000_h	02000
	0	0	0	0	0	0	1	1	FFF_h	03FFF
RAM_c	0	0	0	0	0	1	0	0	000_h	04000
	0	0	0	0	0	1	0	1	FFF_h	05FFF
RAM_d	0	0	0	0	0	1	1	0	000_h	06000
	0	0	0	0	0	1	1	1	FFF_h	07FFF
ROM	1	1	1	1	1	1	1	0	000_h	FE000
	1	1	1	1	1	1	1	1	FFF_h	FFFFF

Tabloyo oluştururken RAM_a'yı 00000_h adresinden başlatırsak, 8k belleğin son adresi $01FFF_h$ olur. RAM_b'yi bir sonraki adres olan 02000_h'den başlatarak belleklerin sürekliliğini sağlarız. Bu şekilde yerleştirerek sonunda RAM_d'yi 06000_h'den başlatarak son adresinin $07FFF_h$'ye geldiğini buluruz. Bellek uzayının sonuna dayamak istediğimiz ROM yongası için $FFFFF_h$'den geriye doğru 8k giderek ROM başlama adresini $FE000_h$ buluruz. Tabloyu bu şekilde doldurarak her bir bellek yongasını etkinleştirmek üzere adres bitlerinin hangilerini kullanmak gerektiğini kolayca görebiliriz.

Tabloda her bir belleğin içinde çözümlenen adres bitlerini ayırarak işe başlarız. Bu tablodaki $a_{12}..a_0$ sütunları hem RAM'lara, hem de ROM'a doğrudan bağlanarak bu yongaların içinde çözümlendiğinden ayrıca yonga dışında çözümleme gerektirmez. Belleklerin kendi adres devresinde çözümlenecek adres bitlerine bellek-içi adres bitleri diyoruz.

Adres çözümlemenin amacı her bir yonga için etkinleştirme sinyali üretmektir. Tablonun en solundaki bir dizi adres biti ($a_{19}..a_{15}$) bütün RAM satırlarında aynı olduğundan RAM yongalarının tümü için $a_{19}..a_{15}$ bitleri yalnızca bir NAND (çıkışı $\overline{y}$ olan) ile değerlendirilir. Bu NAND'ın çıkışı ($\overline{y}$) bir sonraki VE kapısına evirilip bağlanarak değerlendirmiş olduğu adres bitlerinin adrese uygunluğunu iletmesi sağlanır.

Son olarak, adres tablosunu her yonga için iki satır yerine tek satırda kodlamak ta mümkündür. Zaten bütün yongalar için ilk satır ile ikinci satır yonga-içi adreslere kadar birbirinin aynı, yonga-içi adresler için farklıdır. İki satırı tek satıra düşürürken satırların aynı olan adres bitlerini değeri (0 ya da 1) ile, farklı olanlarını ise x ile gösterebiliriz. Ayrıca devrenin yonga içi çözümlenen adreslerle ilgisi olmadığından bu bölgeyi kısmen yazmamak ta sık kullanılan kısaltma yöntemlerine

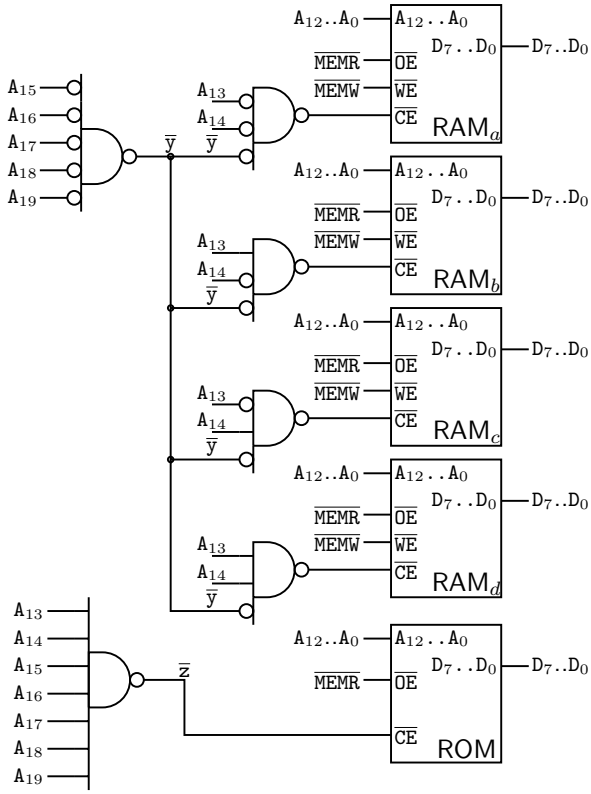
dahildir. Bu şekilde kısaltılan adres tablosu aşağıdaki biçime dönüşür.

sistem	a19	a18	a17	a16	a15	a14	a13	a12 ...	hex adres
RAM$_a$	0	0	0	0	0	0	0	x ...	00000 01FFF
RAM$_b$	0	0	0	0	0	0	1	x ...	02000 03FFF
RAM$_c$	0	0	0	0	0	1	0	x ...	04000 05FFF
RAM$_d$	0	0	0	0	0	1	1	x ...	06000 07FFF
ROM	1	1	1	1	1	1	1	x ...	FE000 FFFFF

9.6.1 NAND ile adres çözücü

Adres tablosuna bakınca $a_{19}..a_{15}$ sütunlarındaki mavi bölgede tüm bitlerin bütün bellek yongaları için aynı değerde, hepsinin 0 olduğunu görüyoruz. Bu kısmı gerçekleştirmeye tek bir VE-değil (NAND) kapısı yeter. Beş girişli NAND'ın girişlerine evirilmiş adres bitleri $\overline{a_{19}}..\overline{a_{15}}$'u verirsek bu NAND'ın çıkışı sadece ($\overline{a_{19}}$ $\overline{a_{18}}$ $\overline{a_{17}}$ $\overline{a_{16}}$ $\overline{a_{15}}$) =(11111), diğer deyişle, (a_{19} a_{18} a_{17} a_{16} a_{15}) =(00000) sağlandığında 0 çıkarır. RAM'ların, kırmızı yazılı a_{14} ve a_{13} bitlerinin farklı değerlerinde etkinleşmesi istendiğinden her bir yongaya bir NAND daha kullanırız ve tabloda yongayı sıfırda etkinleştiren her adres bitini NAND girişine bir evirici ile bağlarız. Önceki NAND çıkışı olan $\overline{y}$, ($a_{19}..a_{15}$) bitleri (00000) iken 0 olur. RAM$_a$'yı etkinleştirecek NAND'ın girişlerine $\overline{y}$, a_{14} ve a_{13} evirilerek verilir. Yalnızca ($\overline{y}$,a_{14}, a_{13})=(0,0,0) olduğunda NAND çıkışı RAM$_a$'nın $\overline{CE}$ girişini düşürerek bu RAM'ı etkinleştirir. Diğer RAM yongalarının $\overline{CE}$ sinyali için de benzer şekilde üç girişli NAND ve eviriciler kullanılır.

Devredeki tek EEPROM'un bütün harici ($a_{19}..a_{13}$) adres bitlerinin 1 olduğu durumda etkinleşmesi gerekiyor. Bunu sağlamak üzere harici adres bitlerini, $\overline{z}$ çıkışını veren NAND kapısının girişlerine doğrudan, evirici kullanmadan bağlarız. NAND çıkışını da doğrudan ROM'un düşük-etkin çalışan $\overline{CE}$ girişine vererek aşağidaki çözücü devresini elde ederiz.



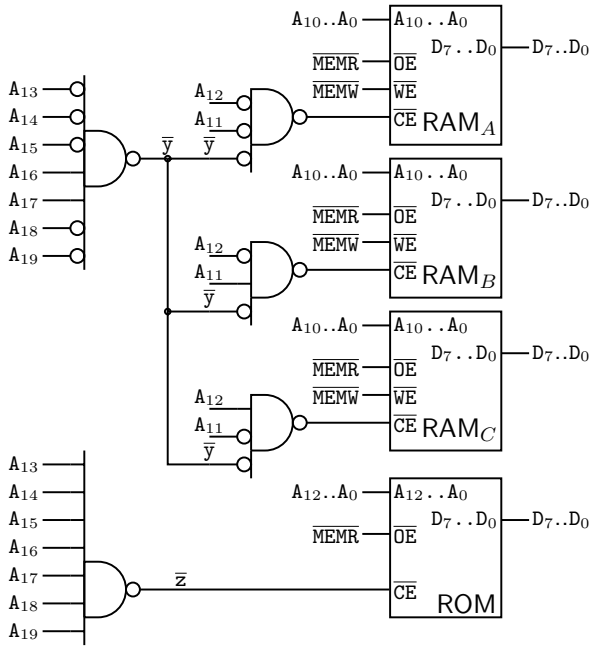
Örnek 9.6.

Son adresi $\texttt{FFFFF}_h$'de olan bir 2764 EEPROM (8k×8-bit) ve ilk adresi 30000_h'den başlayan üç 6116 RAM (2k×8-bit) için NAND kapılı adres çözümleme devresi tasarlayalım

Çözüm: Adres tablosunda $a_{10}..a_0$ bitleri RAM_A, RAM_B, ve RAM_C nin yonga-içi çözümleyicisiyle işlenecektir. ROM'un çözümleyicisi de $a_{12}..a_0$ bitlerini kullanır. RAM_A 30000_h'den başlayarak yerleştirilince aşağıdaki tablo çıkar.

sistem	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10	...	hex adres
RAM_A	0	0	1	1	0	0	0	0	0	x	...	30000 307FF
RAM_B	0	0	1	1	0	0	0	0	1	x	...	30800 30FFF
RAM_C	0	0	1	1	0	0	0	1	0	x	...	31000 317FF
ROM	1	1	1	1	1	1	1	x	x	x	...	FE000 FFFFF

Bu tablodaki RAM'ların mavi bölgesi 7-girişli bir NAND gerektirir. Her RAM için farklı olan her kırmızı satırdaki iki bit mavi bölgeyi özetleyen NAND çıkışıyla birlikte etkinleştirme NAND'ına gireceğinden üç girişli üç NAND kullanacağız. ROM tek yonga olduğundan ROM satırının mavi bölgesini değerlendirecek bir NAND doğrudan ROM'u etkinle çıkışı üretir. Devre aşağıdaki gibidir.



9.7 Dekoder ile adres çözümleme

Dekoder, sayısal elektronikte sık kullanılan birimlerden biridir. Yüksek-etkin m seçme girişi ($S_{m-1}..S_0$), ve $n = 2^m$ adet ($\overline{Y_{n-1}}..\overline{Y_0}$) düşük-etkin çıkışı bulunur. Düşük etkin çalışan çıkış-etkinle ($\overline{E}$) girişi düşük değilse çıkışların hiçbiri etkin olmaz, ve yüksekte kalır. $\overline{E}$ düşükse seçme girişlerine ikili kodlanmış olan sayıya karşılık gelen çıkış düşerek etkinleşir. Yaygın olarak kullanılan 74LS138 3'ten 8'e dekoderin üç seçme girişi, $2^3 = 8$ çıkışı ve biri yüksek etkin, diğer ikisi yüksek etkin olan üç çıkış-etkinle girişi vardır.

74LS139 yongası içinde yalnızca tek etkinle girişi ($\overline{E}$) bulunan 2'den 4'e iki dekoder devresi yer alır. Bu iki dekoder 74LS139A ve 74LS139B olarak adlandırılır.

Dekoderlerin seçme girişleri adres tablosundaki bazı satırlar arasında farklı olan adres bitlerine bağlanarak kullanılır. Gerekirse, fazladan kullanılan VE-VEYA kapılarıyla dekoder çıkışlarını daha da küçük adres aralıklarına çözümlemek, ya da çıkışları birleştirerek daha da geniş adres aralıklarını etkinlemek mümkündür.

E1	$\overline{E2}$	$\overline{E3}$	S2	S1	S0	$\overline{Y0}$	$\overline{Y1}$	$\overline{Y2}$	$\overline{Y3}$	$\overline{Y4}$	$\overline{Y5}$	$\overline{Y6}$	$\overline{Y7}$
0	x	x	x	x	x	1	1	1	1	1	1	1	1
x	1	x	x	x	x	1	1	1	1	1	1	1	1
x	x	1	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

Tablo 9.5: 74LS138 3'ten 8'e Dekoder Doğruluk Tablosu

$\overline{E}$	S1	S0	$\overline{Y0}$	$\overline{Y1}$	$\overline{Y2}$	$\overline{Y3}$
1	x	x	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Tablo 9.6: 74LS139 2'ten 4'e Dekoder Doğruluk Tablosu. Bir pakette A ve B olarak adlandırılan iki dekoder bulunur.

Örnek 9.7.

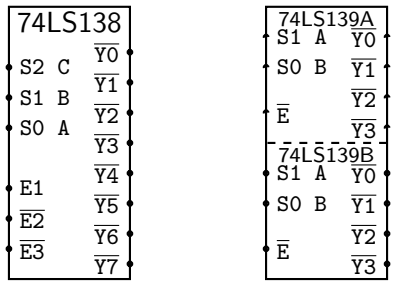
Aşağıdaki verilen adres tablosunu için bir 74LS138 ve NAND kapıları kullanarak adres çözücü devresini bulalım.

sistem	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10	...	hex adres
RAM_A	0	0	1	1	0	0	0	0	0	x	...	30000 307FF
RAM_B	0	0	1	1	0	0	0	0	1	x	...	30800 30FFF
RAM_C	0	0	1	1	0	0	0	1	0	x	...	31000 317FF
ROM	1	1	1	1	1	1	1	x	x	x	...	FE000 FFFFF

Çözüm: ROM'un $\overline{CE}$'si tek bir NAND ile üretilir. RAM'lardan biri $a_{19}..a_{13}$ =(0011000) olduğunda etkinleşirken, RAM_A, RAM_B ve RAM_C belleklerinin etkinleşmesi de a_{12} ve a_{11} adres bitlerinin 00, 01 ve 10 olmasına bağlı olarak belirlenir.

Öncelikle a_{12} ve a_{11}'i 138'in S1 ve S0 girişine bağlayarak başlamalıyız. RAM'ları düşükken etkinleştiren a_{13}'ü S2'ye bağlayarak RAM_A'nın $\overline{CE}$ sini (S_2,S_1,S_0)=(000)'a karşılık gelen $\overline{Y_1}$ çıkışından üretebiliriz. Ancak diğer adres bitlerinin de istenen adres aralığını sağlaması için devreyi değişik biçimlerde gerçekleştirebiliriz.

Birinci yolda, E_1'i tabloya göre yüksekte etkinleyen a_{16}'ya; $\overline{E_2}$'yi ise

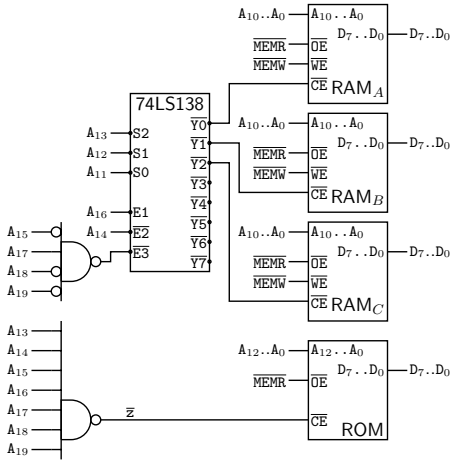


Şekil 9.4: 74LS138 ve 74LS139 Dekoder yongaları

düşükte etkinleyen a_{13}'e bağlarız. Geriye kalan adres bitlerini çıkışı $\overline{S_2}$'yi sürecek NAND kapısıyla işleriz. Yukarıdaki bağlantıları adres tablosunda şöyle gösteririz.

sistem	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10	...	hex adres
RAM_A	0	0	1	1	0	0	0	0	0	x	...	30000 307FF
RAM_B	0	0	1	1	0	0	0	0	1	x	...	30800 30FFF
RAM_C	0	0	1	1	0	0	0	1	0	x	...	31000 317FF
ROM	1	1	1	1	1	1	1	x	x	x	...	FE000 FFFFF
LS138	$\overline{E_3}$	$\overline{E_3}$	$\overline{E_3}$	E_1	$\overline{E_3}$	$\overline{E_2}$	S_2	S_1	S_0			$RAM_{A,B,C}$

Tablonun RAM'lar ve ROM için kullanılan adres çözücü devre şematiği şöyledir.

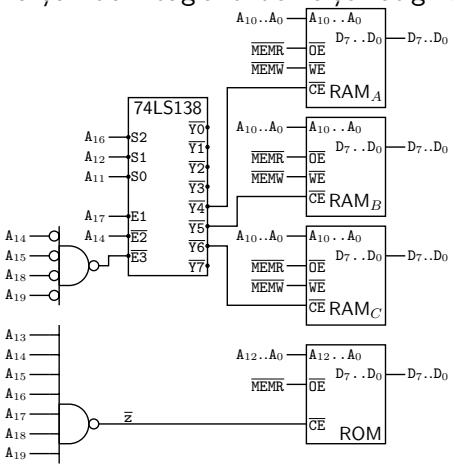


Aynı devreyi ikinci yolla gerçekleştirirken yüksekte etkinleştiren a_{17} ile a_{16}'yı E_1 ile S_2'ye bağlarsak, RAM_A yı $(S_2S_1S_0)$=100 etkinleştirdiğinden RAM_A nın $\overline{CE}$'sini $\overline{Y_4}$'e bağlarız. bu durumda RAM_B nin $\overline{CE}$'sini $\overline{Y_5}$'e RAM_C nın $\overline{CE}$'sini $\overline{Y_6}$'ya bağlarız. Böylece diğer bağlantılara yalnızca düşükte etkinleştiren adres bitleri kalır. Bu durumu aşağıdaki tabloda gösteriyoruz.

sistem	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10	...	hex adres
RAM_A	0	0	1	1	0	0	0	0	0	x	...	30000 307FF
RAM_B	0	0	1	1	0	0	0	0	1	x	...	30800 30FFF
RAM_C	0	0	1	1	0	0	0	1	0	x	...	31000 317FF
ROM	1	1	1	1	1	1	1	x	x	x	...	FE000 FFFFF

LS138	$\overline{E_3}$	$\overline{E_3}$	E_1	S_2	$\overline{E_3}$	$\overline{E_3}$	E_2	S_1	S_0			$RAM_{A,B,C}$

Bu şekildeki bağlantı devre şematiğinde aşağıdaki gibi ifade edilir.



Bu iki çözüm örneğinden görüldüğü gibi adres çözümleme devresi tasarımında çok geniş esneklik payı olduğundan çözümlerin pek çoğu aynı derecede iyi çözüm olarak kabul görür.

Örnek 9.8.

Aşağıdaki verilen adres tablosunu için bir 74LS139'daki iki dekoderi ve NAND kapıları kullanarak adres çözücü devresini bulalım.

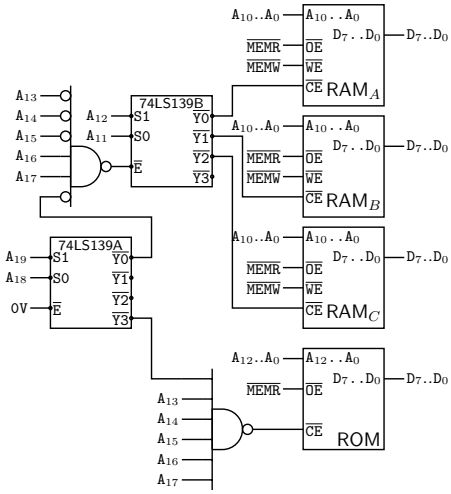
sistem	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10	...	hex adres
RAM_A	0	0	1	1	0	0	0	0	0	x	...	30000 307FF
RAM_B	0	0	1	1	0	0	0	0	1	x	...	30800 30FFF
RAM_C	0	0	1	1	0	0	0	1	0	x	...	31000 317FF
ROM	1	1	1	1	1	1	1	x	x	x	...	FE000 FFFFF

Çözüm: Elimizdeki iki dekoderden birini ROM ile RAM'ları ayırmakta, ve böylece bunların NAND devrelerini küçültmekte kullanabiliriz. Diğer dekoderi $a_1 2$ ve $a_1 1$ adres bitlerini çözmekte kullanırsak $\overline{Y0}$'ı RAM_A, $\overline{Y1}$'ı RAM_B, $\overline{Y2}$'yi RAM_C'nin $\overline{CE}$'lerine bağlarız. Bu durumu adres tablosunda aşağıdaki gibi ifade ederiz.

sistem	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10	...	hex adres
RAM_A	0	0	1	1	0	0	0	0	0	x	...	30000 307FF
RAM_B	0	0	1	1	0	0	0	0	1	x	...	30800 30FFF
RAM_C	0	0	1	1	0	0	0	1	0	x	...	31000 317FF
ROM	1	1	1	1	1	1	1	x	x	x	...	FE000 FFFFF

LS139A	S_1	S_0										RAM/ROM
LS139B			$\overline{E}$	$\overline{E}$	$\overline{E}$	$\overline{E}$	$\overline{E}$	S_1	S_0	x	...	$RAM_{A,B,C}$

Tabloyu devre şematiğine dönüştürürsek aşağıdaki devreyi elde ederiz.



Son olarak vereceğimiz bu örnekte dekoder çıkışlarından adres aralığı daraltma ve genişletme yöntemini göreceğiz.

Örnek 9.9.

Adres 62000h den başlayarak bir 6116 (RAM_A 2k $\times$8bit), adres 64000h den başlayarak iki 6264 (RAM_B, RAM_C, 8k $\times$8bit), ve son olarak 68000h'den başlayarak bir 62256 (RAM_D 32k $\times$8bit) bellek yongası yerleştirmek üzere

a) adres tablosunu

b) adres tablosunda tek LS138 ile çözümü

c) tek LS138'li devrenin şematik çizimini

hazırlayın.

Çözüm:

a) Adres tablosunu daima küçük adreslerden büyüğe giderek kurmalıyız.

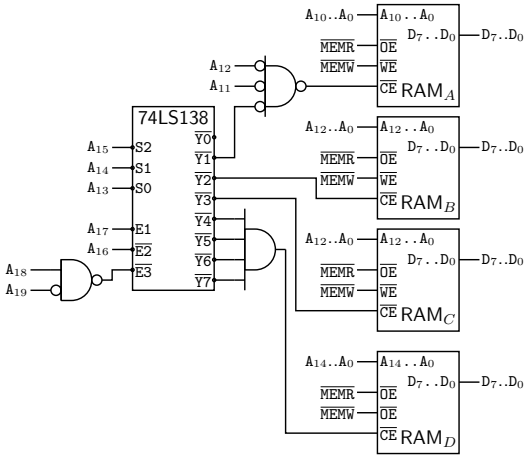
sistem	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10	...	hex adres
RAM_A	0	1	1	0	0	0	1	0	0	x	...	62000 627FF
RAM_B	0	1	1	0	0	1	0	x	x	x	...	64000 65FFF
RAM_C	0	1	1	0	0	1	1	x	x	x	...	66000 67FFF
RAM_D	0	1	1	0	1	x	x	x	x	x	...	68000 6FFFF

b) Tabloda yonga içi çözülen adreslerin çoğu (dörtte ikisi) a12 bitinden başlıyor. Dekoderin S2,S1,S0 girişlerine a15,a14,a13 bitlerini rastlatırsak iki RAM tam çözümlenirken diğer iki RAM de adres aralığı daraltma ve adres aralığı genişletme teknikleriyle etkinleyeceğiz. Dekoder seçme sütunlarını kırmızı renk ile gösterirsek tablo aşağıdakine dönüşür.

sistem	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10	...	hex adres
RAM_A	0	1	1	0	0	0	1	0	0	x	...	62000 627FF
RAM_B	0	1	1	0	0	1	0	x	x	x	...	64000 65FFF
RAM_C	0	1	1	0	0	1	1	x	x	x	...	66000 67FFF
RAM_D	0	1	1	0	1	x	x	x	x	x	...	68000 6FFFF
LS138	$\overline{E3}$	$\overline{E3}$	$\overline{E1}$	$\overline{E2}$	S2	S1	S0				...	$RAM_{A,B,C,D}$

Elimizdeki dekoderi a_{12} ve a_{11} adres bitlerini çözmekte kullanırsak $\overline{Y0}$'ı RAM_A, $\overline{Y1}$'ı RAM_B, $\overline{Y2}$'yi RAM_C'nin $\overline{CE}$'lerine bağlarız. Bu durumu adres tablosunda aşağıdaki gibi ifade ederiz.

c) Bu tabloyu devre şematiğine çevirdiğimizde aşağidaki devre oluşur.



9.8 PC'de bellek bütünlüğü ve güvenirliği

IBM-PC'de bellek bütünlüğü ve güvenliğini korumak üzere iki önemli tedbir bulunur. Hem bellek bütünlüğü hem de bellek güvenirliği amaçlı tedbirler bilgisayara güç beslemeyi başlatınca açılış işlemi sırasında gerçekleştirildiği gibi ayrıca bellek güvenirliği PC'nin çalışması boyunca bellekten yapılan tüm okumalarda donanım aracılığıyla sınanır.

Bellek bütünlüğü, bilgisayarın açılışını sağlayan boot ROM'undaki kodların değişerek bozulması ya da doğru okunamaması durumunda kullanıcıyı uyarmayı ve yürütmeyi durdurmayı amaçlar. Boot ROM içinde PC'nin çeşitli birimlerini test eden kodlardan biri de bütün ROM'u ilk adresinden son adresine kadar XOR ya da toplama işleminden geçirir. Sonucun son baytına CHECKSUM değeri denir, ve ROM'un son baytına bu değeri sıfır yapacak değer yazılır. Böylece CHECKSUM testinin sonucu sıfır vermezse hataya ilişkin bir mesaj yollayan işlemci testlerin sonunda hata oluştuğu için bir sonsuz döngüye takılır.

Bellek güvenirliği, özellikle DRAM belleklerin, okuma, veya tazeleme işlemlerindeki hatalar yüzünden düzgün çalışmaması durumunda kodu yürütmenin durdurulması ve kullanıcının uyarılmasına yöneliktir. Bu amaçla, 8-bit genişliğindeki bellek bankları donanımda 9 bit olarak gerçekleştirilir. Belleğe yazılan her baytın eşlik değeri[1] bir eşlik hesaplayıcı donanımla bulunup dokuzuncu bite yazılır. Okunan her baytın eşliği de aynı donanım ile hesaplanıp dokuzuncu bitten okunan eşlik ile karşılaştırılır. Okunan ve hesaplanan eşlikler farklıysa eşlik karşılaştırıcı XOR devresi işlemciyi bir durdurulamaz kesme ile uyarır. Kesme servisi, kesme dönüş adresinden bozuk bellek blokunu tespit ederek kullanıcıyı bir mesaj ile uyarır ve sonsuz döngüde bekler.

Bellek güvenirliğini doğru eşlik değerleriyle başlatabilmek üzere boot ROM'daki açılış kodu belleğin tümüne damatahtası dizisi adı verilen 0AAh ve 055h dizisi yazıp okur. Böylece bozuk RAM daha açılışta belirlenmiş olur. Bunun yanı sıra, bilgisayar açılış aşamasını hatasız geçse bile besleme gerilimi veya elektromanyetik gürültü nedeniyle oluşacak okuma-yazma hatalarını da anında tespit edip programın muhakkak doğru çalışmasını, ya da hata mesajıyla durmasını sağlar.

9.9 Bellek veri yolu genişliği ve veri yolu band genişliği

Veri yolundan birim zamanda iletilen veri miktarına *veri yolu bant genişliği* denir. f_{bw} ile gösterilen veri yolu bant genişliği , n_d veri yolu genişliğine ve T_d veri yolu çevrim süresine bağlıdır.

$$f_{bw} = \frac{n_d}{T_d}$$

Örneğin 5MHz'lik 8088 sisteminde veriyolu, 8-bit okuma ya da yazma bellek çevrimini 4 işlemci çevriminde tamamlıyor. Bellek veri yolu bant genişliğini hesaplarken n_d=1 bayt, ve T_d=4$\times\frac{1}{5\text{MHz}}$ olduğundan

$$f_{bw,88,0W} = \frac{n_d}{T_d} = 1\,\text{bayt} \times 5\text{MHz}/4 = 1.25\,\text{Mbayt/saniye}$$

buluruz.

[1] parity

80286 veriyolu 16-bit=2 bayt genişlikte olduğundan ve 16-bit veri aktarımını 2 işlemci çevriminde tamamladığından 2.5 MHz 8086 sisteminde veriyolu bant genişliği

$$f_{bw,286,0W} = \frac{n_d}{T_d} = 2\,\text{bayt} \times 2.5\text{MHz}/2 = 2.5\,\text{Mbayt/saniye}$$

olur. İşlemci saat hızı iki kat düşük olan 80286'nın bant genişliği iki kat daha yüksek olduğuna göre birim zamanda iki kat fazla komut çalıştırabilir.

8088'den iki kat düşük hızdaki 80286 işlemci, yavaş bellek ile çalıştırıldığında, örneğin her erişim çevriminde 3 bekleme çevrimi kullansa bant genişliği

$$f_{bw,286,3W} = \frac{n_d}{T_d} = 2\,\text{bayt} \times 2.5\text{MHz}/(2+3) = 1\,\text{Mbayt/saniye}$$

olacağından 8088'li sistemden yavaş çalışır. Yavaş bellek yüzünden 80286 sisteminin saat hızını arttırsak bile bekleme çevrim sayısı da aynı oranda artacağından sonuç pek fazla etkilenmez.

Bölüm 10

Giriş/Çıkış devreleri uygulamaları

8086 ve 8088 işlemcilerinde izole giriş/çıkış gerçekleştirmek üzere dört çeşit komut olduğunu, ve bu komutlara uygun basit giriş/çıkış devrelerinin nasıl çalıştığını bölüm 8.6.3, 8.6.4, ve 8.6.5'de öğrendik. Buna ek olarak, tıpkı izole giriş/çıkış komutu bulunmayan pek çok işlemci gibi 8088 ve 8086 sisteminde de bellek üzerinden giriş/çıkış portu gerçekleştirebiliriz. Bellek üzerinden giriş/çıkış portlarının en büyük avantajı her çeşit adreslemeyi kullanan bellek veri aktarım ya da işlem komutu ile giriş çıkış yapmaya olanak sağlamasıdır. Ancak, bunun yanında, başlıca sakıncaları, giriş/çıkış donanımının bellek erişim sistemini karmaşıklaştırması ve bellek adres uzayında yer kaplamasıdır. Bu bölümde önce bellek üzerinden giriş çıkış devrelerini izole giriş çıkış ile karşılaştırmalı olarak göreceğiz. Ardından, birden çok giriş/çıkış portu gerektiğinde sık sık kullanılan programlanabilir çevre arayüzü (PPI)[1] yongalarıyla çeşitli uygulamalar göreceğiz.

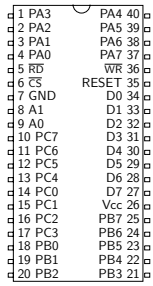
10.1 8255 programlanabilir çevre arayüzü yongası

8080 işlemcisine port olarak geliştirilmiş olan 8255 çevre arayüzü yongası PPI[2] olarak ta isimlendirilir. Yonga, hem basit giriş/çıkış, hem de tek ve çift yönlü 8-bit paralel veri aktarımı protokolünü sağlamak üzere tasarlanmıştır. Burada yonganın yalnızca basit/giriş çıkış amaçlı kullanımını sağlayan mod-0 uygulamalarıyla ilgileneceğiz.

10.1.1 8255'in uçları ve yazmaç adresleri

8-bit veriyoluna bağlanmak üzere geliştirilmiş 8255 PPI yongası mod-0 kullanımında PCH ve PCL olmak üzere dörder bitlik iki adet, ve PA ile PB olmak üzere sekizer bitlik iki adet giriş/çıkış devresi olarak tasarlanmıştır. PPI'daki yazmaçlara $\overline{\text{CS}}$ etkinleştirilmeden erişilemez.

[1] programmable peripheral interface (PPI)
[2] Programmable Peripheral Interface (PPI)



Şekil 10.1: 40 uçlu 8255 yongasının uçları

Dörder bitlik portlar aynı yazmacın sağ ve sol dört biti olduğundan bir arada kullanıldığında 8-bitlik üç adet giriş/çıkış portu oluşturur. Bu portların yazmaç adresleri 8255'in `A0` ve `A1` uçlarıyla seçilir. Seçilen porta yazmak için $\overline{\texttt{WR}}$, portu okumak için ise $\overline{\texttt{RD}}$ girişi etkinleştirilir. Üç port yazmacı dışında, 8255'in çalışma modlarını ve bu modlardaki durumunu belirleyen bir de yapılandırma[3] yazmacı bulunur. `RESET` girişi bu yazmaçları tüm portların giriş olarak çalıştığı başlangıç durumuna getirir. Şekil 10.1'deki 8255 yongasının uçları şunları amaçlar.

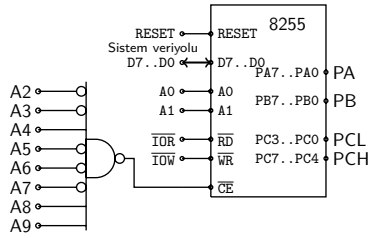
- `D0..D7`: Sistem veri yolu
- $\overline{\texttt{CS}}$, `A0`, `A1`: Yonga etkinleştirme ve yazmaç seçme girişleri
- $\overline{\texttt{RD}}$, $\overline{\texttt{WR}}$: Seçilen yazmaca okuma ve yazma girişleri
- `PA`, `PB`: Port-A ve Port-B giriş/çıkış uçları
- `PCL`, `PCH`: Port-C sağ-dört ve sol-dört giriş/çıkış uçları
- `RESET`: PPI başlatma ucu.

PPI yongasındaki port yazmaçlarına Tablo 10.1'daki gibi erişilir.

CS	A1	A0	Seçilen Yazmaç
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Yapılandırma Yazmacı.
1	x	x	Yonga Etkin değil

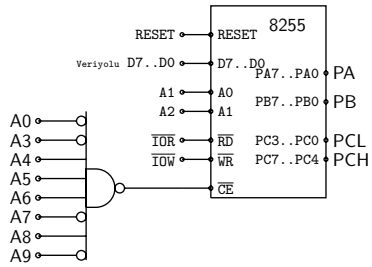
Tablo 10.1: 8255 PPI yongasında yazmaçlara erişim tablosu.

[3] configuration



Adres Yolu	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Adres
PORTA	1	1	0	0	0	1	0	0	0	0	310h
PORTB	1	1	0	0	0	1	0	0	0	1	311h
PORTC	1	1	0	0	0	1	0	0	1	0	312h
Yapılandırma Yazmacı	1	1	0	0	0	1	0	0	1	1	313h

PPI.A0 ile PPI.A1'in sistemin A0 ve A1 adres hatlarına bağlanma zorunluluğu yoktur. Örneğin 16-bitlik veri yolu kullanıldığında, çift adresler Bank-L'den (d15..d8 bitleri) okunurken tek adresler Bank-H'den (d7..d0 bitleri) erişildiğinden bütün yazmaç adreslerini çift adreslere rastlatmak için aşağıdaki gibi PPI.A0'ı sistemin A1'ine, PPI.A1'i ise sistemin A2'sine bağlamak gerekir. Bu durumda adres çözümlemenin eksikli kalmaması için sistemin A0'ını da adres çözücü kapıda değerlendiririz.



Adres Yolu	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Adres
PPI	I	1	1	0	0	0	1	A1	A0	0	310h-316h
PORTA	1	1	0	0	0	1	0	0	0	0	310h
PORTB	1	1	0	0	0	1	0	0	1	0	312h
PORTC	1	1	0	0	0	1	0	1	0	0	314h
Yap. Yazm.	1	1	0	0	0	1	0	1	1	0	316h

10.1.2 8255'in yapılandırma yazmacı ve mod-0 yapısı

8255 PPI yongasını mod-0 düzeninde yapılandırmak için yapılandırma yazmacının ilgili bitlerine gereken değerleri yazmalıyız. Yapılandırma yazmacının bitleri Tablo 10.2'daki işlevleri gerçekleştirir:

Örnek 10.1.

Bir sistemde PPI'nın PA'sından bazı röleleri sürüp, PB'sine, 6 anahtar-düğmeli bir tuş takımı, PCL'ye iki LED bağlayacağız. Boş

Bit	işlevi
D7	1: Basit giriş/çıkış modu / 0: BSR (bit-set-reset)
D6 D5	Mod seçimi, 00: Mod0 / 01: Mod1 / 1x: Mod2
D4	PA yapılandırma, 0: çıkış / 1: giriş.
D3	PCH yapılandırma, 0: çıkış / 1: giriş.
D2	Mod Seçimi, 0: Mod0 / 1: Mod1.
D1	PB yapılandırma, 0: çıkış / 1: giriş.
D0	PCL yapılandırma, 0: çıkış / 1: giriş.

Tablo 10.2: 8255 PPI yapılandırma yazmacı bitlerinin mod0 işlevleri

kalacak PCH'yi ise çıkış yapmak istiyoruz. Yapılandırma yazmacına ne yazmamız gerekir.

Çözüm: Rölelerin bobinlerini sürmek üzere PA'yı çıkış, anahtarları okumak üzere giriş, Hem LED'lere bağlanacak PCL'yi, hem de boş kalacak PCH'yi çıkış yapmak istiyoruz. Yapılandırma yazmacının D7-D5'ini 100 yaparak mod-0 basit giriş çıkış modunu seçeriz. İlaveten D4'ü ve D3'ü 00 yaparak hem PA'yı hem de PCH'yi çıkış olarak yapılandırırız. D2'yi 0 yaparak mod-0'ı seçeriz. PB'yi giriş, PCL'yi çıkış olarak kullanmak üzere de D1 ve D0'ı da 10 yaparız. Sonuç olarak yapılandırma yazmacına YY=$10000010b$=$82h$ yazmamız gerekir.

——=——

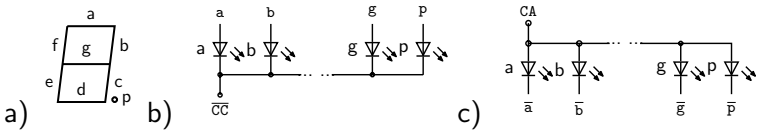
10.2 8255 ile 7-Bölütlü Göstergelerin kullanılışı

70'lı yıllarda üretilmeye başlanan ışık çıkaran diyotlar (LED'ler) iki üç voltu geçmeyen ileri eğilim gerilimleri nedeniyle 5V gerilim kaynaklı sayısal devrelere en uygun gösterge niteliği kazandılar. 5V ileri gerilim altında bir LED diyot TTL çıkış katlarının sürebileceğinden daha yüksek akımlar çekmesi nedeniyle portlara daima akımı sınırlayan bir seri direnç ile bağlanır. Üst üste iki karenin her bir kenarını birbirinden bağımsız olarak aydınlatan 7 adet LED'in özel biçimde bağlanması ile elde edilen yapıya 7-bölütlü LED gösterge[4] adı verilir. Kısaca 7seg olarak adlandırdığımız bu göstergelerin sayıyı gösterecek yedi bölütündeki 7 LED'in yanısıra ondalık noktası için de bir LED'i bulunur.

[4] 7-segment LED display

10.2.1 7-Bölütlü Göstergenin Yapısı

Yedi bölütlü göstergelerdeki sekiz LED'in her birini en az sayıda uç ile birbirinden bağımsız sürebilmek üzere sekiz akım kolu ve bir de akımın geri dönüşünü sağlayacak ortak uç gerekir. Örneğin ortak anotlu göstergede LED'lerin anotları ortak anot CA ucuna bağlanır. Her bir bölütün katotları da $\overline{a}$, $\overline{b}$, $\overline{c}$, $\overline{d}$, $\overline{e}$, $\overline{f}$, $\overline{g}$, ve $\overline{p}$ adıyla anılır. Üstü çizgili bölüt isimleri LED'i etkinleştirmek için ucun düşük mantık çıkışıyla sürülmesi gerektiğini gösterir. Katodları $\overline{CC}$ ucunda birleştirilen ortak katod göstergelerde ise, her bir bölütün anodu ayrı ayrı a, b, c, d, e, f, g, ve p olarak adlandıracağımız bölüt uçlarını oluşturur. Katodları birleşik olanlara ortak katodlu, anodları birleşik olanlara ise ortak anodlu 7seg denir. 7seg'lerin bölüt isimler Şekil 10.2'deki gibidir.



Şekil 10.2: 7-segment LED gösterge. a) bölütlerin ve noktanın markalanması (a, b, ... , g, p) b) Ortak Katodlu 7seg göstergenin yapısı, c) Ortak Anotlu 7seg yapısı.

10.2.2 Gösterge Kodu ve Kullanılışı

Ortak katodlu ($\overline{CC}$) göstergeye "1" yazmak istersek b ve c bölütlerini akım yollayarak etkinleştirmemiz, diğer tüm bölütleri durdurmamız gerekir. Örneğin $\overline{CC}$ ucu toprağa (0V'a) bağlı göstergenin bölüt uçları a← PA0, b← PA1, c← PA2, d← PA3, e← PA4, f← PA5, g← PA6, p← PA7 biçiminde bağlıysa, bölüt LED'leri port çıkışı 1 olduğunda yanacağına göre PA'ya $\overset{p\,g\,f\,e\,d\,c\,b\,a}{00000110_b}$=06h gönderirsek yalnızca b ve c bölütünü yakabiliriz. Göstergede "1" görünmesi için yolladığımız bu sayıya birin gösterge kodu deriz. Gösterge kodları, bölüt uçlarının port uçlarına bağlandığı sıraya göre değişir. Bölütleri aynı sırada bağlı *CA*[5] göstergenin gösterge kodu, *CC*[6] göstergenin gösterge kodunu eviriğidir. Örneğin burada birin aynı bağlantı sırası için "1"'in CA gösterge kodunu 06h bulduysak CC gösterge için $\overline{06h}$ =0F9h buluruz.

[5] Common Anode *ortak anotlu*
[6] Common Cathode *ortak katodlu*

Bağlantı değişikliklerinin program koduna etkisini azaltmak üzere gösterge kodlarını bir tablodan çağırarak kullanırız. Aşağıda iki değişik bağlantı sırası için, { a← PA0, b← PA1, c← PA2, d← PA3, e← PA4, f← PA5, g← PA6, p← PA7} bağlantılı bir CC göstegenin, ve { p← PA0, a← PA1, b← PA2, c← PA3, d← PA4, e← PA5, f← PA6, g← PA7} bağlantılı bir CC göstergenin veri bölütündeki kod tablosunu görüyoruz.

```
1    .data
2 ;        abcdefgp
3 CCGK db 11111100b, ;0
4         01100000b, ;1
5         11011010b, ;2
6         11110010b, ;3
7         01100110b, ;4
8         10110110b, ;5
9         10111110b, ;6
10        11100000b, ;7
11        11111110b, ;8
12        11110110b  ;9
13        ...
```

```
1    .data
2 ;        pabcdefg
3 CCGK db 01111110b, ;0
4         00110000b, ;1
5         01101101b, ;2
6         01111001b, ;3
7         00110011b, ;4
8         01011011b, ;5
9         01011111b, ;6
10        01110000b, ;7
11        01111111b, ;8
12        01111011b  ;9
13        ...
```

ya da onaltılık sayılarla

```
1    .data
2 ; bağlantı sırası abcdefgp
3 CCGK db 0FCh, 060h, 0DAh, 0F2h, 066h,
4         0B6h, 0BEh, 0E0h, 0FEh, 0F6h
5         ...
```

ve

```
1    .data
2 ; bağlantı sırası pabcdefg
3 CCGK db 07Eh, 030h, 06Dh, 079h, 033h,
4         05Bh, 05Fh, 070h, 07Fh, 07Bh
5         ...
```

Tabloyu kullanırken, örneğin bölüt uçları port-0310h'ye bağlanmışsa ortak katodlu göstergeye 3 yazmak üzere gösterge kodunu tablodan dolaylı adreslemeyle okuruz.

```
1    .data
2 ; bağlantı sırası abcdefgp
3 CCGK db 0FCh, 060h, 0DAh
4   db 0F2h, 066h, 0B6h, 0BEh
5   db 0E0h, 0FEh, 0F6h
6    ...
7    .code
8    ...
9    mov bx,3
```

```
10    mov al,[bx]+offset CCGK
11    mov dx,0310h
12    out dx,al
13    ...
```

Aynı tabloyu ortak anodlu devrede tablodan okunan gösterge kodlarını evirerek, ya da, `com` komutuna gerek kalmadan evirilmiş gösterge kodu tablosu ile kullanabiliriz.

```
1     .data
2  CCGK db 0FCh, 060h, 0DAh
3    db 0F2h, 066h, 0B6h, 0BEh
4    db 0E0h, 0FEh, 0F6h
5     .code
6     ...
7     mov bx,3
8     mov al,[bx]+offset CCGK
9     com al ; CA ise evir
10    mov dx,0310h
11    out dx,al
12    ...
```

```
1     .data
2  CCGK db ~0FCh, ~060h, ~0DAh
3   db ~0F2h, ~066h, ~0B6h, ~0BEh
4   db ~0E0h, ~0FEh, ~0F6h
5     .code
6     ...
7     mov bx,3
8     mov al,[bx]+offset CCGK
9     mov dx,0310h
10    out dx,al
11    ...
```

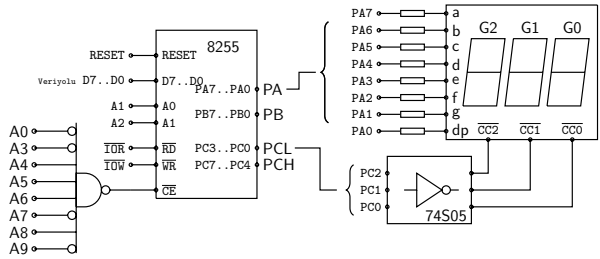
10.2.3 Çoklamalı 7-Bölütlü Gösterge

Sekiz taneye kadar 7seg göstergeyi bir arada bir porta çoklamalı yöntemle bağlamak mümkündür. Çoklamalı yöntemde aynı porta bağlı aygıtları bir anda yalnızca birini olmak üzere sırayla sürerek hepsinin çalışması sağlanır.

Etkin olan aygıtı belirlemek üzere ikinci bir port kullanılır. Örneğin ortak katodlu üç göstergeyi çoklamalı kullanmak için, her üç göstergenin bölüt uçları Şekil 10.3'deki gibi, akım sınırlayacak dirençler üzerinden `PA` portuna bağlanır. Bir anda yalnızca bir gösterge çalışacağından port çıkışı göstergeleri sürecek güçtedir.

En sağdan başlayarak göstergeleri `G0`, `G1` ve `G2` olarak numaralayıp ortak katodlarını sırasıyla $\overline{\texttt{CC0}}$, $\overline{\texttt{CC1}}$, $\overline{\texttt{CC2}}$ ile gösterelim. $\overline{\texttt{CC0}}$, $\overline{\texttt{CC1}}$, $\overline{\texttt{CC2}}$ uçları, sırasıyla `PC0`, `PC1` ve `PC2` uçlarını eviren 74S05 evirici-tamponun çıkışlarına bağlanır. Böylece `PC0` yüksekken evirilmiş çıkış $\overline{\texttt{CC0}}$'ı toprağa bağlar, düşükken toprak bağlantısı olmayan G0 çalışmaz.

74S05 veya ULN 2004 yongaları sekizli bir evirici tampondur. Girişleri 1 TTL yükü olmasına karşın açık kollektör tipi çıkışları 100 mA'e kadarki yükleri rahatça sürebilir. Bu iki yongayı motor ya da röle bobini gibi yüksek akımla sürülen aygıtları sürmek için de kullanırız. Evirici tampon yedi bölütten aynı anda geçebilecek akımın `PCL` uçlarını aşırı yüklemesini önler.



Şekil 10.3: Çoklamalı gösterge uygulaması

Örneğin göstergeye "123" yazmak için, sırasıyla

a) G0'a '3' yazmak için $\{$PA$\leftarrow \overset{a\,b\,c\,d\,e\,f\,g\,p}{11110010}b, PC\leftarrow \overset{2\,1\,0}{00000110}b\}$ yollanıp görülmesi için 2 milisaniye beklenir
b) G1'e '2' yazmak için $\{$PA$\leftarrow \overset{a\,b\,c\,d\,e\,f\,g\,p}{11011010}b, PC\leftarrow \overset{2\,1\,0}{00000101}b\}$ yollanıp görülmesi için 2 milisaniye beklenir
c) G2'ye '1' yazmak için $\{$PA$\leftarrow \overset{a\,b\,c\,d\,e\,f\,g\,p}{01100000}b, PC\leftarrow \overset{2\,1\,0}{00000011}b\}$ yollanıp bu gösterge için de 2 milisaniye beklenir.
d) Rakamların görülmesini sürdürmek üzere (a-c) işlemleri ara vermeksizin tekrarlanır.

$\{$PA$\leftarrow \overset{a\,b\,c\,d\,e\,f\,g\,p}{11110010}b, PC\leftarrow \overset{2\,1\,0}{00000110}b\}$ yollandığında, PC'nin kullanılan üç biti evirildiği için $\overline{\text{CC2}}$=0, $\overline{\text{CC1}}$=0, $\overline{\text{CC0}}$=0 olur ve yalnızca en sağdaki G0 göstergesinin ortak katodu topraklandığı için PA'ya yazılan 0F2h sadece o göstergede 3 yazılmasını sağlar. Birkaç milisaniye sonra portlara $\{$PA$\leftarrow \overset{a\,b\,c\,d\,e\,f\,g\,p}{11011010}b, PC\leftarrow \overset{2\,1\,0}{00000101}b\}$ yazıldığında bu kez PA'ya yollanan 0DAh, G1 göstergesinde 2 görülmesini, ve ondan birkaç milisaniye sonra yollanan $\{$PA$\leftarrow \overset{a\,b\,c\,d\,e\,f\,g\,p}{01100000}b, PC\leftarrow \overset{2\,1\,0}{00000011}b\}$, G2 göstergesinde 1 çıkmasını sağlar. Bu işlem sürekli tekrarlanırsa hızlı değişimlerin ortalamasını alan gözlerimiz sanki G2 göstergesinde sürekli 1, G1'de sürekli 2, G0'da sürekli 3 yazıyor gibi görür ve böylece göstergedeki sayıyı 123 olarak okuruz.

Örnek 10.2.

Giriş çıkış adresi 0310h-0313h aralığında olan PPI'ın PA'sına $\overline{\text{a}}\leftarrow$ PA0, $\overline{\text{b}}\leftarrow$ PA1, $\overline{\text{c}}\leftarrow$ PA2, $\overline{\text{d}}\leftarrow$ PA3, $\overline{\text{e}}\leftarrow$ PA4, $\overline{\text{f}}\leftarrow$ PA5, $\overline{\text{g}}\leftarrow$ PA6, $\overline{\text{p}}\leftarrow$ PA7 olacak biçimde, üç ortak anotlu yedi-bölütlü-gösterge bağlayalım ve PC2, PC1, PC0 uçlarını akımı evirmeden tamponlayan emiter-takipçisi transistörlerle CA2, CA1, CA0 uçlarına bağlayalım. 8086'mızın işlemci frekansı ilk IBM-PC'ler gibi 4.77MHz olsun. Bu donanım için göstergede saniyeleri yazacak bir kaynak kodu yazmaya çalışacağız.

Çözüm Programda bazı sabitleri isimlendirmek kodlamayı kolaylaştıracaktır. Örneğin PA 0310h, ve PB 0312h gibi. Zaman saymak için `loop` komutunun 17 saat çevrimi sürmesinden yararlanacağız. İşlemci frekansı 4.77MHz olduğuna göre boş loop döngüsü 17/4.77= 3.564 mikrosaniye sürer. Ya da 1ms (milisaniye) için 4770/17=280 loop döngüsü gerekir. CX'e 65535'ten büyük rakam koyamayacağımız için bu yolla en çok 65536/280=234 ms bekleyebiliriz.

```
1  ; M.Bodur / saniye göstergesi
2  ; işlemci hızı 4.77 MHz.
3  ; 0310-0313 port asresindeki
4  ;  PPI'ın PA7..PA0 ına
5  ;  ortak katodlu üç 7-seg
6  ;  abcdefgp sırasıyla bağlı.
7  ; PC2..PC0 sırasıyla
8  ;  ~CC2, ~CC1, ~CC0'a bağlı.
9
10    .model small
11    .stack 64
12
13 bekle macro süre
14     local bekleLP
15     mov cx, süre
16     beklelP: loop bekleLP
17     endm
18
19    .data
20 ; gösterge kod tablosu
21 CCGK db 0FCh, 060h, 0DAh
22   db 0F2h, 066h, 0B6h
23   db 0BEh, 0E0h, 0FEh, 0F6h
24 ; gevşek BCD, 3 rakamlı
25 sayı  db 0,0,0
26 ; gösterge indeksi, ve yeri
27 gi dw 2
28 gy db 00000100b
29 ;    zaman sayacı
30 zs db 100
31
32    .code
33 ;ilkdeğerler
34    mov ax,@data
35    mov ds,ax
36
37 ; PPI'nin ilkdeğeri
38 ;  PA ve PCL portları çıkış,
39     mov dx,313h
40 ;  bit-5 PA, bit-4 PCH,
41 ;  bit-1 PB, bit-0 PCL
42     mov al,10001010b
43     out dx,al
44
45 Anadöngü:
46     bekle 10*280 ; 10ms
47 ; zaman sayacını bir azalt
48     dec zs
49
50 ; zs =0 ise 1 saniye doldu
51     jnz zssıfırdeğil
52 ; zs=100 yap, sayı'yı arttır
53     mov al,100
54     mov zs,al
55
56    ;BCD sayıyı arttır
57     mov bx,0
58     mov cx,3
59    scf
60 LP:
61     mov al,0
62     adc al, [bx+offset sayı]
63     aaa
64     mov [bx+ offset sayı],al
65     inc bx
66     loop LP:
67
68 zssıfırdeğil:
69 ; gösterge imlecini kaydır.
70 ;  gy=0 olursa gi=2,
71 ;  ve gy=00000100b yap
72     dec gi
73     shr gy
74     jne gibitmedi
75     mov ax, 2
76     mov gi, al
77     mov al, 00000100b
78     mov gy, ax
```

```
gibitmedi:

; indekslenen sayıyı göster
    mov bx,gi
    mov bl,[bx+sayı]
    mov al,[bx]+offset CCGK
    com al ; CA ise al'yi evir
    mov dx,0310h
    out dx,al
; gy'nin gösterdiğini seç
    mov al,gy
    mov dx,0312h
    out dx,al

; 10ms periyodlu döngü başına
    jmp Anadöngü
    end main
```

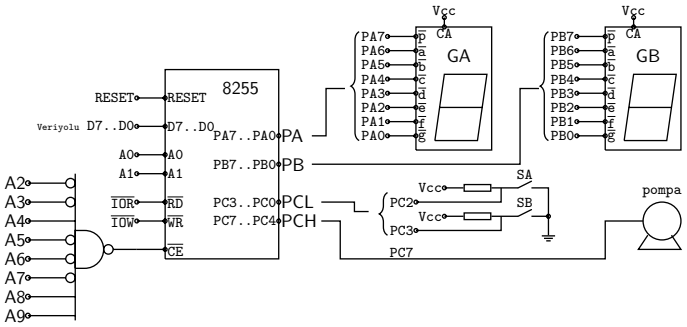
Örnek 10.3.

8088'in 4.77 MHz'de işlediği aşağıdaki devredeki 8255'in CS girişi 0318h - 031Fh aralığında etkinleşiyor. PPI'nin A0 ucu veri yolunun A1'ine, A1 ucu ise veri yolunun A2'sine bağlıdır. 8255'in PA ve PB çıkışlarına GA ve GB ile gösterilen birer ortak-anotlu 7-bölütlü-gösterge bağlıdır. Göstergelerin $\overline{p}$,$\overline{a}$,$\overline{b}$,$\overline{c}$,$\overline{d}$,$\overline{e}$,$\overline{f}$,$\overline{g}$ uçları sırasıyla b7..b0 bitlerine, CA uçları da Vcc'ye bağlıdır. PCL'nin PC2 ucu SA seviye detektör anahtarına, PC3 ise SB seviye detektör anahtarına bağlıdır. Anahtarlar kapalı devreyken port girişini toprağa bağlamaktadır. PC7 ise ilk tanktan ikinci tanka su basan pompa motoruna kumanda etmektedir.

Her on milisaniyede bir,

- GA'da eğer SA açık devreyse 'E', değilse 'F' görünmesini.
- GB'de eğer SB açık devreyse 'E', değilse 'F' görünmesini.
- PC7 çıkışının yalnızca SA kapalı ama SB açıksa 1, değilse 0 olmasını

sağlayan bir çevirici kodu yazın.



Şekil 10.4: İki tanklı su deposu sisteminin donanımı.

Çözüm:

Bu kodlamayı önce mantık işlemlerini dallanmalarla kodlayarak, daha sonra da istenenlerin bir işlev tablosunu çıkararak iki değişik yol ile çözeceğiz.

Kodlama-1: Program kodlamaya başlamadan belirlememiz gereken maddeler şunlardır:

- Port adresleri PA: 0310h, PB: 0311h, PC:0312h, Yapılandırma Yazmacı YY: 0313h'dir.
- PA, PB ve PCH çıkış, PCL ise giriş olarak yapılanmalıdır.
- 4.77 MHz saat hızındaki 8088 bir milisaniyede (ms) loop komutunu 1000*4.77/17=280 kere işler. 10 ms bekleme için boş loop 2800 kere dönmelidir.
- pabcdefg sırasıyla bağlı olan GA ve GB'ye 'E' yazan gösterge kodu $\overline{01001111b}$, 'F' yazan gösterge kodu $\overline{01000111b}$'dir .
- SA ve SB anahtarlarını sınamak üzere SA= 00001000b ve SB=00000100b olmak üzere iki etiket kullanacağız. Anahtar toprağa bağlı olduğundan, açıkken 1, kapalıyken 0 okuruz. TEST komutu ile sınadığımızda sonuç sıfırsa anahtarın kapalı, birse açık olduğunu anlarız.
- hem SA açık, hem SB kapalı durumda PC7'yi 1 yapabilmek üzere BL=10000000b ile başlayacağız. SA kapalıysa veya SB açıksa BL'yi 00000000b yapıp BL'yi PC'ye yazacağız.

```
1  ; iki tanklı su sisteminde
2  ; pompa ve gösterge kodu
3
4  ;sabitler
5  PA     equ 310h
6  PB     equ 311h
7  PC     equ 312h
8  YY     equ 313h
9  GkE    equ ~4Fh
10 GkF    equ ~47h
11 SA     equ 00001000b
12 SB     equ 00000100b
13
14 ;makrolar
15 portyaz macro port,sayı
16    mov dx, port
17    mov al, sayı
18    out dx, al
19    endm
20 portoku macro port
21    mov dx, port
22    in dx, al
23    endm
24
25    .model small
26    .code
27
28 ;PPI yapılandırma
29    portyaz YY, 10000001b
30
31 anadöngü:
32    ; PC7 için BL=80h yap
33    mov bl=80h
34    ;anahtarları oku
35    portoku PC
36    ; SB için ah'ye sakla
37    mov ah, al
38    test al, SA
39    jz SAaçık
40    ;SA kapalıysa
41    mov bl,0
42    portyaz PA, GkF
43    jmp SBtest
44 SAaçık:
45    portyaz PA, GkE
46 SBtest:
47    test ah, SB
48    jz SBaçık
```

```
49    ;SB kapalıysa
50    portyaz PA, GkF
51    jmp pompa
52 SBaçık:
53    mov bl,0
54    portyaz PA, GkE
55 pompa:
56    portyaz PC, bl
57 ; 10 ms bekleme
58    mov cx, 2800
59 LP: loop LP
60   jmp mainloop
61   end
62
```

Kodlama-2: Bu çözümde problemdeki eylemi tablo olarak yazıp bütün işlemleri tabloya bakarak yapacağız. Bunun için gerekli ön hazırlık:

- Problemde durum bilgisi yoksa işlem sonucu yalnızca girişlere bağlıdır. Bu soruda istenenler de girişlerin eski değerlerine değil, yalnızca son değerine bağlı olduğundan sorudaki eylem girişlere kombinatoryal olarak bağlıdır ve giriş-çıkış tablosu ile ifade edilebilir.
- Soruda SA ve SB olmak üzere iki ikili giriş, PA, PB ve PC olmak üzere üç 8-bitlik çıkış var. Tablonun girişleri SA ve SB'nin 4 değişik kombinasyonu, çıkışları PA, PB ve PC olacak.

SB	SA	GA	GB	pompa
PC3	PC2	PA	PB	PC
kapalı 0	kapalı 0	47h F	47h F	00h dur
kapalı 0	açık 1	4Fh E	47h F	00h dur
açık 1	kapalı 0	47h F	4Fh E	80h pompala
açık 1	açık 1	4Fh E	4Fh E	00h dur

```
1    .model small
2    .data
3 TPA  ~47h,~4Fh,~47h,~4Fh
4 TPB  ~47h,~47h,~4Fh,~4Fh
5 TPC   0,0, 80h,0
6    .code
7 main proc far
8    mov ax,@data
9    mov ds,ax
10 ; PPI yapılandır
11   mov dx, 313h ; YY
12    mov al, 10000001b
13    out dx,al    ; PPI
14
15 anadöngü:
16    mov dx, 312h ; PC
17    in al, dx   ; SA,SB
18    and ax,000Ch; maske
19 ; SB,SA'yı en sağa kaydır
20    shl al,1
21    shl al,1
22 ; index yap
23    mov bx,ax ;
24 ; tablodan oku, porta yaz
25    mov al, [bx]+offset TPA
26    mov dx, 0318h ; PA
27    out dx,al
28    mov al, [bx]+offset TPB
29    mov dx, 031Ah ; PB
30    out dx,al
31    mov al, [bx]+offset TPC
32    mov dx, 031Ch ; PC
33    out dx,al
34 ;10 ms bekle
35    mov cx,2800
36 LP: loop LP
37    jmp anadöngü
38 main endp
39    end main
```

Kodlama-3: C dilinde çözüm dallanmalarla şöyle ifade edilir.

```
// PA, PB, PC, YY port adresi: 0x318h, 0x31Ah, 0x31Ch, 0x31Eh.
// "E" nin kodu  ~01001111b = ~4Fh
// "F" in kodu   ~01000111b = ~47h
#include <dos.h>
#include <conio.h>
#include <stdio.h>
main() {
  char PC;
  outp(0x313h, 0b10000001); //PPI yapılandır
do{
  PC=inp(0x312h);
  if (PC&0x04) {outp(0x310, ~0x47)};
  else outp(0x310, ~0x4F);
  if (PC&0x08) outp(0x311, ~0x47);
  else outp(0x311, ~0x4Fh);
  if ((PC&0x0C)==0x04) outp(0x312,0x80);
  else outp(0x312,0x00);
  } while(1); }
```

10.3 LCD gösterge modülü

Hitachi firmasının Optrex adıyla geliştirdiği çok satırlı dot-matris karakter üreteçli LCD sürücü, doğrudan sürülmesi son derece zor olan LCD-cam gösterlere göre arayüzü son derece kolay bir LCD modül tipinin standartlaşmasını sağladı.

Tipik bir LCD modülde 8 veri ucu D7..D0, ve üç denetim ucu bulunur. Denetim uçlarından

- E düşen kenarda çalışan etkinleştirme girişidir.
- RS yazmaç seçici girişidir. metin için 0, komut için 1 kullanılır.
- R/$\overline{W}$ okuma/yazma seçici giriştir. Okuma 1, yazma 0 iledir.

Modülün daha kısıtlı ve yavaş çalışan 4-bit modunda yalnızca D7..D4 uçlarını kullanılarak veri uçları azaltılır. Böylece portların uç sayısı yetersiz kaldığında, R/$\overline{W}$ ucu toprağa bağlanan modül yalnızca 6 çıkış ucuyla sürülebilir. Tipik olarak 32x5x7 karakterlik modül yaklaşık 4mA harcayarak pikselleri saniyede 50 kez sürer. Sönük ışıkta modülü görünür yapan arka-aydınlatması için de yaklaşık 15 mA gerekir.

LCD modül güç uygulandıktan sonra yaklaşık 80 ms başlama süresi gerektirir. Çoğu uygulamada modülü düzenlemeye başlamadan önce 200ms beklenir. Modülü düzenlemek için kullanılabilecek komutlar Tablo 10.3'da verilmiştir.

Hex Code	Command
28	2 satır 5x7 nokta 4-bit modu (4.8ms bekle)
38	2 satır 5x7 nokta 8-bit modu (4.8ms bekle)
01	Göstergeyi sil (1ms bekle)
02	Ilk karaktere dön
04	Ekran imini arttır (sola kaydır)
06	Ekran imini azalt (sağa kaydır)
05	Ekranı sağa kaydır
07	Ekranı sola kaydır
08	Ekranı ve ekran imlecini kapat
0A	Ekranı kapat ekran imlecini aç
0C	Ekranı aç ekran imlecini kapat
0E	Ekranı ve ekran imlecini aç
0F	Ekranı aç ekran imlecini kırpala
10	Ekran imlecini sola kaydır
14	Ekran imlecini sağa kaydır
18	Bütün ekranı sola kaydır
1C	Bütün ekranı sağa kaydır
80	Ekran imlecini ilk satır ilk sütuna koy
C0	Ekran imlecini ikinci satır ilk sütuna koy

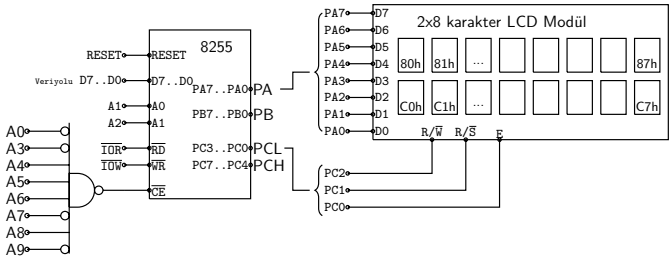
Tablo 10.3: LCD modül düzenleme komutları

10.3.1 LCD Modülün port bağlantısı ve kullanılışı

LCD Module "Selam" yazmak için yolladığımız komut ya da karakterlerin kodlarını sırayla modülün `D7..D0` girişlerine verir, veri bir komutsa `RS`=1 yapar, `E` girişini yükseltir ve düşürürüz. Modülü yeni başlatıyorsak 8-bit veri iletimi için yapılandırmak üzere şu verileri yollamamız gerekir:

- Modüle gerilim uygulayıp, 200ms bekleriz.
- Modülün `RS` girişi yüksek mantık değerindeyken şu komutları yollamalıyız
  - ▷ 38h yollayarak modülü 2-satır ve 5x7 noktalı düzende başlatırız. Bu komutun ardından en az 4.8ms beklememiz gerekir.
  - ▷ 0Eh yollayarak, ekranı ve ekran imlecini açarız.
  - ▷ 01h yollayarak, ekrandaki bütün karakterleri sileriz. Bu işlem 1.2 ms alır.
  - ▷ 06h yollayarak, ekran imlecinin her harfle otomatik sağa kaymasını sağlarız
  - ▷ 80h yollayarak, ekran imlecini ilk satırın başına getiririz.
- Yazılacak metini yollarken `RS` girişini düşükte tutarız. Ekran imi harfler yollandıkça kendiliğinden sağa kayar. Ardarda yollanan 'S', 'e', 'l', 'a', 'm' ascii karakterleri ekranda birinci satır birinci sütundan başlayarak 'Selam' yazar.

Örnek 10.4.



Şekil 10.5: 8-bit verili LCD modülü 8255 arayüzü

Şekil 10.5'deki LCD modülün arayüzünün bağlandığı 8088 yongasının işlemci saati 4.77 MHz olsun. Kullanılmayan port uçları giriş olarak yapılandırılacak olsun. Modülün ilk satırında geçen saniyeleri yazan bir 8088 programı yazalım.

Çözüm: Önce gerekli bilgileri toplayarak başlayalım.

- Port adresleri PA-0170h, PB-012Ah, PC-014Ch, Yapılandırma Yazmacı YY-0176h adresindedir.
- `PA` ve `PCL` çıkış `PB` ve `PCH` ise giriş olmalıdır.
- 8088 loop komutu 17 saat döngüsü tuttuğundan 4.77 MHz saat hızında bir milisaniye (ms) için `CX`=1000*4.77/17=280 lik `loop` döngüsü gerekir. `N` ms bekleme için `CX`=280* `N` gerekir. Bu yolla tek döngüde en çok 65535/280= 234 ms bekleme süresi elde edilir.
- Modüle veri yazarken PC2 sıfır yapılmalı, veri komut ise PC1'deki RS=1, metin ise RS=0 olmalı, PC0 a bağlı E, veri varken 1 yapılıp, 1 mikrosaniye sonra 0'a düşürülmelidir. Bunun için Metin=00000000b, ve Komut=00000010b adında sabitler tanımlayacağız

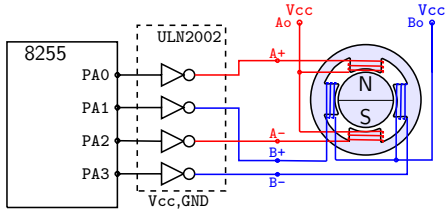
```
1  ; (c) M.Bodur.
2  ; LCD modülde saniye sayma
3  ;  İşlemci saati 4.77 MHz.
4    .model small
5    .stack 64
6
7    .data
8  sayı db 0,0,0
9
10 ; sabitler
11 msan  equ 280
12 Metin equ 0
13 Komut equ 2
14 PA    equ 0170h
15 PB    equ 0172h
16 PC    equ 0174h
17 YY    equ 0176h
18
19 bekle macro süre
20     local bekleLP
21     mov cx, süre
22 bekleLP: loop bekleLP
23     endm
24
25 LCDYolla macro veri
26     mov al,veri
27     call LCDChr
28     endm
29
30 ; artık koda başlayabiliriz.
31     .code
32 main:
33     mov ax,@data
34     mov ds,ax
```

```
35
36 ; PPI yapılandırma
37     mov dx, YY
38     mov al, 10001010b
39     out dx,al
40
41 ; LCD başlatma
42     mov ah,0
43     LCDYolla 0
44     mov ah, Komut
45     LCDYolla 38h
46     bekle  5*msan
47     LCDYolla 0eh
48     LCDYolla 01h
49     bekle  2*msan
50     LCDYolla 06h
51     LCDYolla 80h
52     LCDYolla 06h
53
54 Anadöngü:
55     ;Ekranı sil, başa git
56     mov ah, Komut
57     LCDYolla 01h
58     bekle  2*msan
59     LCDYolla 80h
60
61    ;Sayıyı göster
62     mov ah, Metin
63     mov al, sayı+2
64     or  al,30h
65     call LCDChr
66     mov al, sayı+1
67     or al,30h
68     call LCDChr
69     mov al, sayı
70     or al,30h
71     call LCDChr
72
73    ;BCD sayıyı arttır
74    mov bx,0
75    mov cx,3
76    scf
77 LP:
78    mov al,0
79    adc al, [bx+offset sayı]
80    aaa
81    mov [bx+ offset sayı],al
82    inc bx
83    loop LP:
84
85 ; 5*200 msan= 1saniye bekle
86 LP1s: mov al,5
87    bekle 200*msan
88    dec al
89    jnz LP1s
90
91 ;  döngü başına git
92    jmp anadöngü
93
94 LCDChr proc
95 ; LCD'ye AL'dekini yollar.
96 ; RS değerini AH belirler.
97    push ax
98    push dx
99    push cx
100   mov dx, PA
101   out dx, al
102   mov dx, PC
103   mov al, ah
104   out dx, al
105   inc al
106   out dx, al
107   ;1 mikrosan bekle
108   bekle 1
109   dec al
110   out dx, al
111   mov dx, PA
112   mov al, 0
113   out dx, al
114   pop cx
115   pop dx
116   pop ax
117   ret
118 LCDChr endp
119   end main
```

10.4 Adım Motoru Arayüzü

En basit yapılı 6 uçlu can-stack tipi sabit miknatıs rotorlu adım motorunun statorunda orta uçlu iki bobin çifti bulunur. Her bir bobin çifti statorda farklı bir açıyla yerleştirildiğinden sürülen bobine bağlı olarak miknatıslı rotorun N kutubu sürülen bobinin kutup-başlarına yönelir. Adım motorlarının rotorundaki sabit miknatısın N-kutup sayısı 4, 8, 12, 24, 36, 48, 64, 96 gibi çeşitli sayılarda olabilir ve örneğin 24 N-kutuplu bir can-stack motor her tam adımda $1/24$ tur ilerler.

Motorun A sarımının orta ucu A0, diğer iki ucu A^+ ve A^- olarak, B sarımının da orta ucu B0, diğer iki ucu B^+ ve B^- olarak adlandırılır. Böyle bir motoru sayısal devrelerle sürmek için en kolay yol A0 ve B0 uçlarını +Vcc gerilim kaynağına bağlamak, A^+, B^+, A^- ve B^- uçlarını ise portun b0,b1,b2,b3 bitleriyle değiştirmektir. Motor sarımlarının nominal akımı, port çıkışının sürebileceği maksimum akımdan yüksek olduğundan araya 78S05 veya ULN2002 gibi bir açık-kollektörlü sürücü, ve bobinin akımını keserken oluşacak gerilim darbelerini önlemek ve manyetik alanında biriken enerjiyi gerilim kaynağına geri yüklemek üzere bobinlerin ucundan gerilim kaynağına birer diyot bağlamak gerekir. 500mA'e kadarki endüktif yüklere kullanılabilen ULN2002 gereken diyotlarla da donatılmıştır.



Şekil 10.6: Sabit miknatıs rotorlu altı uçlu adım motorunun unipolar çalışması için 8255 arayüzü.

Sarımlara yalnızca bir yönde gerilim uygulayan Şekil 10.6'deki bağlantı tipine unipolar bağlantı denir. A^+ ile A^- aynı manyetik yolun üzerinde olduğu için ikisine aynı anda uygulanan akımların manyetik etkisi birbirini götürür. Uçlara uygulanan unipolar nominal gerilime karşılık rotorun açısı ve bu açıda tutunma torku aşağıdaki tablo 10.4'da verilmiştir. Tutma torku unipolar yerine bipolar bağlantı kullanılırsa dört sarımın da birbirini destekler yönde sürülmesiyle %100 olur. Birinci satırda görüldüğü üzere rotordaki kalıcı miknatıs nede-

niyle gerilim uygulanmasa bile nisbeten zayıf ta olsa motorun adımını kolayca kaybetmeyeceği derecede bir tutma torku vardır.

Tablo 10.4: Unipolar bağlanan k N-kutuplu motorda uygulanan uç gerilimlerine karşılık rotor açısı θ ve tutma torku oranı (F/F_{max})

B^-	A^-	B^+	A^+	Açı: θ	tork: (F/F_{max})%	adım
0	0	0	0	-	5%	-
0	0	0	1	0	30%	yarım
0	0	1	1	45/k	50%	tam
0	0	1	0	90/k	30%	yarım
0	1	1	0	135/k	50%	tam
0	1	0	0	180/k	30%	yarım
1	1	0	0	225/k	50%	tam
1	0	0	0	270/k	30%	yarım
1	0	0	1	315/k	50%	tam
0	0	0	1	360/k	30%	yarım

10.4.1 Motor Kodu Tablosu ve Kullanılması

Tablodan da anlaşılacağı üzere, adım motorunun sürekli dönebilmesi için sarım uçlarına uygulanan akımların düzenli biçimde sürekli değiştirilmesi gerekir. Tablodaki sırada oluşan değişim dizisi rotoru her adımda saatin tersi yönde sekizde bir tur döndürür. Ancak motorun daha güçlü çalışması için yalnızca iki sarımın aynı anda sürüldüğü güçlü adımları kullanmak gerekir. Bu dört adımlık dizi ise rotoru her adımda dörtte bir tur döndürür. Arada zayıf adımları kullanan sekizli motor kodlarına yarım adımlı kod da denir.

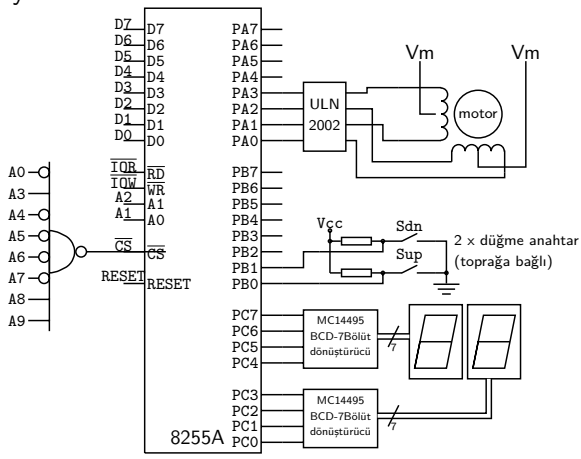
Port PA'ya bağlanmış olan motor, tablodaki 8 adımlık dizi için sırasıyla adım kodları PA=$\{$01h, 03h, 02h, 06h, 04h, 0Ch, 08h, 09h$\}$ dizisinde ileri dönerken dizinin herhangi bir noktasında dizi geriye doğru ilerlerse geriye döner. Daha güçlü çalışan 4 adımlık dizi ise PA=$\{$03h, 06h, 0Ch, 09h$\}$ adım kodlarından oluşur. Motorun bir adımdan bir sonrakine geçebilmesi zaman aldığı için her adımın arasında motorun hızına uygun bir süre beklemek gerekir. Bekleme sürelerini ayarlayarak motorun hızını son derece hassas olarak belirleyebiliriz.

Uygulamada, adım kodu dizisini bir tabloya koyar, motorun açısal durumunu adım sayısı olarak örneğin `Adım` tamsayısında tutarız. Motor kodunu tablodan okurken `Adım`'ın son üç yada iki bitini imleç olarak kullanırız. Adımları kolayca uygulayabilmek için makro ve prosedürlerden yararlanırız.

.

Örnek 10.5.

4.77 MHz'lik 8088 e bağlanmış bir Bir 8255A'nın portları PA'dan bir adım motorunu sürerken PB0'den Sup ve PB1'den Sdn düğmelerinin durumunu okuyor. Her 200ms 'de bir Sup 0 (basılı) ise motor yarım adım ilerletilsin, Sdn 0 ise yarım adım geriletilsin, ve adım motorunun açısal konumu basamak sayısı olarak ikili sistemde PC'ye yazılsın istiyoruz.



Çözüm Programı kodlamak üzere hazırlık yapalım.

- Devre şematiğinden PA: 0308h, PB: 030Ah, PC: 030Ch, ve yapılandırma YY: 0x030Eh adreslerinde olduğu görülüyor.
- PPI'da, PA ve PC çıkış PB giriş olarak yapılandırılmalıyız.
- 200 ms beklemek üzere 200*4770/17 =56117 loop komutu çalıştıracağız.
- Yarım adımlı sekizli motor kodları olan 01h, 03h, 02h, 06h, 04h, 0Ch, 08h, 09h dizisini MKT motor kodu tablosuna yerleştireceğiz.
- Motor adım sayacının son üç bitini motor kodunun imleci olarak kullanacağız.
- Sdn ve Sup basıldığında 0, serbest iken 1 okunur. Anahtarları mantıksal akış ile çözümleyerek motor konum sayacını güncelleyeceğiz.
- Adım sayacını Port-C'ye yazarken evirerek değeri '1' olan bitlerin porttan '0' olarak çıkıp LED'ini yakmasını sağlayacağız.

```
1 ; Motor denetim
2 ; (c) M.Bodur
3 .model small
4   .data
5 Adım  dw ?
6 MKT db 01h,03h,02h,06h
7   db 04h,0Ch,08h,09h
8   .code
9 main proc far
10   mov ax,@data
11   mov ds,ax
12 ;8255 yapılandırma
13   mov dx,30Eh
14   mov al, 10000010b
15   out dx,al
16 ;adım ilkdeğeri
```

```
17   mov ax, 0
18   mov Adım,ax
19
20 anadöngü:
21 ;anahtarları oku
22   mov dx,30Ah
23   in al,dx ;Sup-b0,Sdn-b1
24 ; ah'ye kopyala
25   mov ah,al
26 ; Sup birse adımı arttır
27   and al,01h ; Sup
28   jnz diğerdüğme
29   inc step
30 diğerdüğme:
31 ;ah'deki kopyayı kullan
32   and ah,02h ; Sdn
33   jnz adımgöster
34   dec Adım
35 adımgöster:
36   mov dx,30Ch
37   mov ax, Adım
38   out  dx,al
39 ;adımı indekse döndür
40   and ax,0007h ; maske
41   mov bx, ax
42   mov al,[bx]+offset MKT
43   mov dx, 308h
44   out dx,al
45   mov cx, 56117 ; onlusayı
46 L1: loop L1 ;200ms bekle
47 ..jmp anadöngü
48 main endp
```

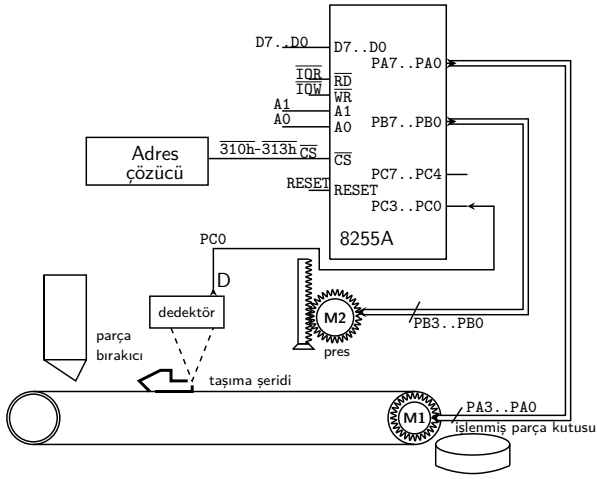
Bu programı C ile kodlamak oldukça kolaydır.

```
1 #include <conio.h>
2 #include <stdio.h>
3 main() {
4     const char
5     MKT[ ]= (0x01,0x03,0x02,0x06,0x04,0x0C,0x08,0x09);
6     outp(0x30B, 0x80); // 8255A yapılandır
7     char anahtar, Adım=0;
8     do{ // anadöngü
9     anahtar=inp(0x309);
10    if ( !(anahtar&0x01) ){ step++; }
11            if ( !(anahtar&0x02) ){ Adım--; }
12            outp(0x30A, Adım);
13            outp(0x308, MKT[ Adım&0x03 ] );
14            delay(200); // 200ms bekleme
15            } while(1);
16    }
```

Örnek 10.6.

Aşağıdaki 4.77MHz 8088 işlemcili sistemde 308h-30Fh port adreslerindeki 8255A PPI yongasına M1 ve M2 adım motorlarıyla D optik parça detektörü bağlanmıştır. PA portuna bağlı M1 bir konveyör şeridini döndürüyor. Şerit, parça bırakıcıdan inen parçaları taşırken, parça detektörün altındaysa, detektör PPI'nin PC2 ucuna +Vcc, altında değilse 0V veriyor. PB'ye bağlı olan M2 ise bir baskı presini aşağı yukarı

hareket ettiriyor. Yazılacak program kodunun sırasıyla şu işleri yapması isteniyor



- Şerit boş kaldığı sürece M1 motoru 50ms'de bir adım ilerleyerek döndür.
- Dönen şerit, üzerine parça bırakıcıdan düşen parçayı ilerletip D detektörünün altına getir.
- Program kodu, D'den Vcc gelmeye başlayınca M1'e tam 22 adım daha attırarak parçayı tam presin altına getir.
- Kod, M2 motorunu 30ms'de bir adım ilerleyerek toplam 45 adımda parçaya presi uygula.
- M1 motoru dururken, M2 motorunu 25ms'de bir adım atarak 30 adım ileriye döndürerek presi parçanın üzerine indirmeli, ve ardından aynı hızda 30 adım geriye döndürerek presi yukarıya çek.
- Parça işlenmişler kutusuna düşünceye kadar, ve D'den Vcc gelmediği sürece M1 motorunu 50ms de bir adım hızında döndürmeye devam et.

Çözüm Kodlamaya geçmeden önce planımızı yapalım.

- Motorların dört tam adım için 4'lü konum kodlarını tutan MKT adlı bir tablo oluşturup içine { 3,6,12,9} yazmalıyız.
- M1 ve M2 motorlarının açısal konumunu tutmak üzere 8-bitlik adım1 ve adım2 tanımlamalıyız.
- Öncelikle motorlardan herbirini değişik hız ve adımla çalıştırabilmek için bir makro tanımlamakta yarar var. `motor` makrosunda şu argumanlar gerekecek: i) `a` konum değişkeni, ii) `p` port adresi, iii) `d` dönüş yönü iv) `n` adım sayısı v) `w` bekleme süresi.
- Bekleme için daha öncekiler gibi 17 saat döngülük `loop` komutunun 4.77MHz'de 28 kere işlediğinde toplam 100 mikro saniye sürmesini kullanabiliriz.

Şimdi programın kodunu yazmaya başlayabiliriz.

```
;(c) M Bodur 2 motorlu örnek
; 4.77 MHz'de 100usan=28loop
yuzusan equ 28

motor macro a,p,d,n,w
;a: motor konum sözcügü
;n: uygulanacak adim sayisi
;d: dönüs yönü, (1 or -1)
;p: motor portu
;w: bekleme süresi(x100us)
;MKT: motor kodu tablosu
   local mn,mw
   mov cx,n
mn:
   mov bx,a
   add bx,d
   mov a,bx
   and bx, 0003h
   mov al,[BX+offset MKT]
   mov dx,p
   out dx,al
; wait w * 100mikrosaniye
   push cx
   mov cx, w*yüzusan
mw:
   loop mw
   pop cx
   loop mn
   endm

;Program Kodu
   .model small
   .stack 64
   .data
adim1 dw 0
adim2 dw 0
MKT db 3,6,12,9

   .code
   mov ax,@data
   mov ds,ax

;PPI yapilandirma
; PA,PB çikis, PCL giris
   mov dx,303h
   mov al,10000001b
   out dx,al

anadöngü:
;M2 bir adim ileri
;            a,   p,d,n, w
   motor adim1,300h,1,1,500
;detektöre bak
   mov dx,302h
   in al,dx
   ; D=0 ise basa dön
   and al,01h
   jz anadöngü,

;parça algilandi
;            a,   p, d, n, w
   motor adim1,300h, 1,22,500
   motor adim2,301h, 1,45,300
   motor adim2,301h,-1,45,300
   jmp anadöngü
   end
```

Aynı kodu, daha basit makrolar kullanarak, ve sadece kod bölütünde yazmaya çalışsak aşağıdaki programı elde ederiz

```
motor macro p,n
  mov dx,p
  mov bx,word ptr n
  and bx,0003h
  mov al,CS:[bx+offset MKT]
  out dx,al
  endm

bekle macro yüzmikrosaniye
  local bekle1,
  push cx
  mov cx, yüzmikrosaniye*28
```

```
13 bekle1:
14   loop bekle1
15   pop cx
16   endm
17
18   .model small
19   .code
20 main:
21   jmp kodbasi
22 MKT:
23   db 0Ch,06h
24   db 03h,09h
25 A1 dw 0
26 A2 dw 0
27
28 kodbasi:
29 ;PPI yapilandir
30   mov dx, 303h
31   mov al, 81h
32   out dx, al
33
34 anadöngü:
35   inc word ptr CS:A1
36   motor 301h, A1
37   bekle 500
38
39   ; detektörü oku
40   mov dx,302h
41   in al, dx
42   test al, 01h
43   jnz anadöngü
44
45 ;parça detektörün altinda
46   mov cx,22
47 geneA1:
48   inc word ptr CS:A1
49   motor 300h, A1
50   bekle 500
51   loop geneA1
52
53   mov cx,45
54 ileriA2:
55   inc word ptr CS:A2
56   motor 301, A2
57   bekle  300
58   loop ileriA2:
59
60   mov cx,45
61 geriA2:
62   dec word ptr CS:A2
63   motor 301, A2
64   bekle 300
65   loop geriA2
66 ..
67   jmp anadöngü
68   end
```

10.5 Sayısaldan-Örneksele Dönüştürücüler

Sayısal işlemciler yazmaç genişliğine bağlı olarak 8 ya da 16-bitlik tamsayıları doğrudan komutlarıyla işleyebilir. Işlemcinin doğrudan mantıksal ve aritmetik işlemlerde kullanabildiği sonlu tamsayıları gerilim, ya da akım gibi fiziksel varlıklı örneksel değişkenlere dönüştüren donanıma Digital/Analog (D/A) ya da Sayısal/Örneksel dönüştürücü, kısaca DAC diyoruz.

Önde gelen DAC teknolojileri arasında

i) R/2R merdiven devresini,

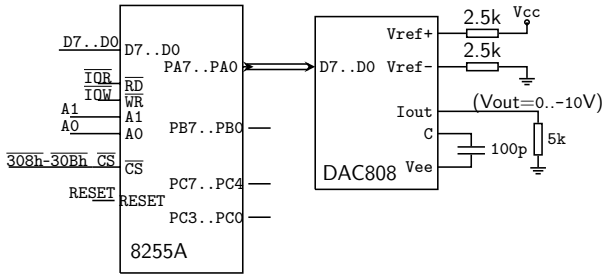
ii) darbe genişlik bindirme[7] tekniğini ve

iii) delta-sigma dönüştürücüleri

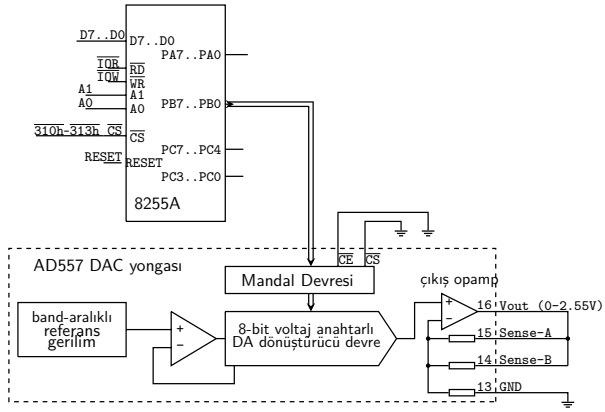
sayabiliriz. Telekomun sayısallaşmasıyla birlikte delta-sigma dönüştü-

[7] Pulse Width Modulation

rücülerin kullanımı baskın duruma gelse de R/2R devreleri özellikle endüstriyel kontrol alanında rolünü korumaktadır. Bu bölümde inceleyeceğimiz DAC808 ve AD557 yongaları 200 kHz'in üzerinde çalışabilen tipik 8-bit R-2R devreleridir. Şekil ve 'de DAC808 ve AD557 ile kurulan örnek DAC devreleri görülmektedir.



Şekil 10.7: DAC808 ile DAC arayüzü



Şekil 10.8: AD557 ile DAC arayüzü

Sayısal-Örneksel dönüştürmelerde örneksel değişkenin alt ve üst sınırlarına alt ve üst referans (ya da alt ve üst gönderim değeri) deriz. Sayısal-örneksel dönüştürmenin çözünürlüğünü genellikle anlamlı bit sayısı ile ifade ederiz. Dönüştürmedeki örneksel tolerans ise alt ve üst referansların farkının 2^n 'de biridir. Dönüştürme hesaplarında kolaylık açısından sayısal değerin bir değişmesi ile örneksel değerde oluşacak değişime örneksel basamak denir. Dönüştürücü sayıyı gerilime dönüştürüyorsa örneksel basamağa gerilim basamağı da denir. Örne-

ğin 8 bitlik bir DAC dönüştürücü 0 ile 255 arasındaki sayıları V_{ref-} ile V_{ref+} arasındaki 255 basamağa dönüştürür. V_{ref-} gerilimi genellikle 0V olarak kullanılır. V_{ref+} ise girişteki en büyük sayıyla elde edilecek çıkış gerilimini belirler.

Akım aynası devreler aracılığıyla $R/2R$ üzerinden analog çıkış akımı oluşturan DAC 808 yongasının (V_{ref+}, V_{ref-}) girişleri 2.5kΩ dirençlerle V_{ref}=5V'a bağlandığında, sayısal giriş N=255 iken çıkış akımı I_{out}=2 mA olur. Bu I_{out} çıkış akımı, bir direnç üzerinde V_{out} gerilimine dönüştürülerek kullanılırsa N=255 durumunda 5k direnç üzerinde -10V, 2.5k üzerinde ise -5V çıkış gerilimi oluşur. Çıkış gerilimi bir evirici opamp devresinin girişine uygulanırsa, 2.5k'lık geri besleme direnci ile, N=255 için opamp çıkışındaki Vout 5V olur. Geri besleme direncini uygun değerde seçerek maksimum çıkış gerilimini istediğimiz değere ayarlayabiliriz.

Özellikle ses sinyali üretmek üzere tasarlanmış olan AD557'nin içinde V_{ref+}=2.55 V referans kaynağı ve V_{ref-}=0V (=toprak) bağlantısı bulunur. Böylece 3V'tan 5.5V'ta kadarki besleme gerilimleri için girişi N=0 da çıkışı Vout=0V, girişi N=255 olduğunda ise çıkışı Vout = 2.55V üretir.

10.5.1 Basamak gerilimi

İdeal DAC girişine sırayla N=0'dan N=255'e kadarki sayıları yolladığımızda çıkış gerilimi $v_{out} = N \times V_{ref}/255$ olur. V_{out}'u hesaplamakta kullandığımız $\Delta V = V_{ref}/255$ çarpanı, birim giriş başına çıkış gerilimi değişimidir ve basamak gerilimi olarak adlandırılır. Basamak gerilimi ΔV sembolü ile kısaltılır.

Örnek 10.7.

8-bitlik DAC808 devresinin referans gerilimi +5V ve toprağa bağlı 2.5kΩ dirençlerle oluşturulunca çıkış akımı 0 ile -10V arasında değişiyor.

a) DAC'ın basamak gerilimi kaç volttur?

b) Girişe N=24 uygulandığında çıkış gerilimini bulun.

Çözüm:

a) DAC'ın girişi 8-bit olduğundan uygulanacak en büyük sayı N=255'tir. N=0'da çıkış 0V, N=255'te -10V olduğuna göre basamak gerilimi

$$\Delta V = -10V/255 = 0.0392\text{V}$$

bulunur.

b) N=20 uygulandığında çıkış gerilimi

$$V_{out} = N \times \Delta V = 20 \times 0.0392 = 0.784\text{V}$$

olur.

10.5.2 Periyodik Fonksiyon Üreteci

Testere dişi dalgası, bir periyodu içinde ilkdeğerinden sondeğerine doğrusal olarak yükselen ve periyodunu tekrarlayarak devam eden bir sinyaldir. DAC çıkışı, çözünürlüğünün yettiği ölçüde, basamak gerilimi kadar değişimler yaparak bir testere dişi dalgası oluşturabilir. Örnek bir testere dalgası oluştururken basamak süresi ve dalganın genliğini belirleyen faktörleri inceleyeceğiz.

Oluşacak testere dişi dalgasının en düşük ve en yüksek değerleri olan V_a ve V_b gerilimlerini üretecek N_a ve N_b girişleri basamak voltajından $N = V/\Delta V$ bağıntısıyla kolayca bulunur. Testere dişinin T_p sürelik doğrusal yükselen sırtını girişteki sayıyı eşit sürelerde birer birer arttırarak elde ederiz. N_a'dan N_b'ye kadar girişi toplam $N_b - N_a$ kez arttıracağımıza göre girişi her arttırmanın ardından

$$\Delta t = \frac{T_p}{N_b - N_a}$$

kadar bekleme süresi gerekir.

Örnek 10.8.

0..-10 V çıkış sağlayabilen 8-bitlik bir DAC ile periyodu T_p=500ms olan 0..-3V'luk testere dişi dalga üretmek üzere N_a, N_b, ve Δt değerlerini bulalım.

Çözüm:

$$N_a = 3000\text{mV}/39.0625\text{mV} = 76.8 \approx 77$$

$T_p = 500$ ms süreyi 77 basamağa dağıtınca her basamakta

$$\Delta t = 500\text{ms}/77 = 6.5\text{ms}$$

beklemek gerektiğini buluruz.

Örnek 10.9.

Beklemeleri kodlarken devredeki işlemcinin 4MHz'de çalıştığını ve `loop` komutunun 17 saat çevriminde tamamlandığını varsayarak. şekil 10.7'deki DAC808 devresinden, 500ms periyotlu, -1V 'tan -8V 'a düşerek tekrar eden bir testere dişi dalga üretecek program yazalım.

Çözüm: Programa başlamadan gerekecek hazırlıklar şunlardır:

- DAC'ın basamak gerilimi -10/255 =-0.0392 V 'tur.
- -1V için N_{-1V}= -1/-0.0392 = 25.5, tamsayı olarak 25'tir.
- -8V için N_{-8V}= -8/-0.0392 = 204 'tür.
- 500ms'lik periyotta 25'ten 204'e N toplam 179 basamak artıp, bir de 25'e dönüş için toplam 180 periyotluk zaman gerekir. Her bir adımın süresi T_N=500/180 = 2.778 ms olmalıdır.
- 2.778 ms=2778μs sürecek bir `loop` döngüsü cx=$\dfrac{2778\mu\text{saniye} \times 4M(\text{çevrim/saniye})}{17(\text{çevrim/tur})}$=653 tur ile başlamalıdır.
- PPI'da PA'yı giriş, diğer bütün portlar çıkış olarak yapılandıracağız.
- Programda altprogram veya veri bölütü kullanmayacağımızdan `DS` 'yi ve yığıtı başlatmaya gerek yoktur.

```
1    .model small
2    .code
3  main proc far
4  ; PPI yapılandırma
5     mov al, 10010000b
6  ...mov dx, 303h
7     out dx,al
8  ; ana döngü başlıyor
9  anadongu:
10 ; -1 V'tan başla
11    mov dx, 300h ; PA
12    mov al, 25 ; -1V
13    out dx, al
14 ; Azalarak devam et
15 L1:
16 ; 2.778 ms bekle
17    mov cx,653
18 bekle: loop bekle
19 ; 205'e dek arttır
20    inc al
21    cmp al,205
22    jb L1
23    jmp anadöngü
24 main endp
25    end main
```

Örnek 10.10.

Aşağıdaki programdaki `bekle10ms` makrosu, işlemci hızı 3.4MHz'lik bir 8088'de argümanı kadar 10 mikrosaniye bekleme sağlıyor. 3.4MHz'lik 8088'e bağlı olan Şekil 10.8'deki AD557 devresinin üreteceği örneksel çıkışın dalga şeklini ve periyodunu bulup (0, 3V)'a karşılık (0, 20 m saniye)'lik bir grafikte gösterin.

```
1 macro bekle10us nn
2 ; işlemci hızı 3.4MHz.
3   local L1
4   mov cx,nn*2
5 L1: loop L1
6   endm
```

```
 7
 8   .model small
 9   .code
10 main proc far
11 ;PPI PA-PB-PC çıkış
12   mov dx, 30Bh
13   mov al, 10000000b
14   out dx,al
15
16 anadongu:
17   mov dx, 309h
18   mov al,50h
19 art:
20   out dx,al
21 ..bekle10us 5
22   inc al
23   cmp al,0A0h
24   jb art:
25 ..bekle10us 100
26 ..mov al,50
27   out dx,al
28 ..bekle10us 200
29 azal:
30   out dx,al
31   bekle10us 2
32   dec al
33 ..cmp al,0
34   jne azal
35   jmp anadongu
36 main endp
37    end main
```

Çözüm: Önce `bekle10us` makrosunun gerçekten argümanın değeri başına ne kadar beklediğini görelim: Argümanın değerine k dersek `loop` komutu $k\times 2$ kere döndüğüne göre işlemci toplam $17 \times 2 \times k$ işlemci çevrimi bekler. Her işlemci çevrimi $T_p = \frac{1}{3.4\text{MHz}}$ olduğuna göre toplam süre

$$T(k) = 34k \times T_p = 10\,k\ \mu\text{saniye}$$

olur.

Devredeki DAC'ın basamak gerilimini belirleyelim: Şekildeki bağlantı için v_{out} gerilimi N=0 için 0V, N=255 için 2.55V üretiyor. Basamak gerilimi

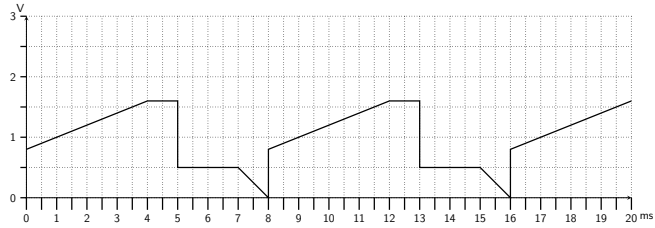
$$x\Delta V = 2.56\text{V}/256 = 0.01\text{V} = 10\text{mV}$$

bulunur.

Şimdi programı izleyerek çıkışı belirleyelim.

- satır-18'de `al`=50h=80'den başlayan anadöngü satır-19..24 arasında 50 μs aralıklarla artarak al=0A0h=160'a kadar çıkıyor ve her turda DAC'a yazılıyor. Böylece toplam 80×50=4000μs =4ms sürede DAC'ın v_{out} gerilimi 80×10mV=0.8V'tan başlayıp onar milivolt basamaklarla 160×10mV=1.6V'a çıkıyor.
- Ardından 1.6 Voltta satır-25 nedeniyle 100×10μs = 1ms sabit kalıyor.
- satır-26-27'de `al`=50 yapılıp bu değer DAC'a yazılınca V_{out} aniden 50× 10mV = 0.5V oluyor ve satır-28 nedeniyle 2ms boyunca sabit kalıyor.

- satır-29..34 arasındaki azaltma döngüsü 50 kere dönüyor ve toplam $50\times\ 20\ =\ 1000\ \mu$saniyede al'yi sıfıra indirerek çıkışı 0V'a düşürüyor.
- satır-35'te anadöngüye dönüşle bir periyod sonlanıyor. Toplam periyod süresi 4ms+1ms+2ms+1ms= 8ms oluyor. Çıkış 8 ms'de bir kendini tekrarlıyor.
- Gerilimleri grafiğe geçirirsek aşağidaki çizimi elde ederiz.



10.5.3 Örneksel-Sayısal Arayüz

Yalnızca sayısal değerleri işleyebilen bir bilgisayarın, sıcaklık, basınç, voltaj, akım gibi örneksel değişkenleri işleyebilmesi için örneksel değişken bir dönüştürücüyle [8] gerilime, gerilim de Analog/Digital dönüştürücü (ADC) devreler kullanılarak sayısal değere dönüştürülür. Diğer fiziksel değişkenleri akım ya da gerilim gibi elektriksel değişkene dönüştüren birimler duyaç[9] olarak da adlandırılır.

ADC tipleri

Örneksel sayısal dönüştürücü tipleri arasında en yaygın üçü şunlardır:

- Değişkeni süreye dönüştürüp süreyi sayarak sayısal değere dönüştürmek en basit ve en ucuz yoldur.
  - **RC zaman sabitinden:** Örneğin oyun çubukları[10] bir potansiyometrenin direncini bir kondansatörle oluşturduğu zaman sabitinden kondansatörün boşalma süresini sayarak bulur.
  - **Çift tümlev yöntemi:**[11] girişin ve referans geriliminin dual eğimli tümlevini kullanan dönüştürücüler, 16-bit yada daha yüksek yüksek çözünürlüğe, girişin yaklaşık bir saniye süreyle ortalamasını alarak erişirler.

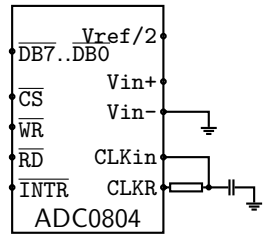
[8] transducer
[9] sensor
[10] Joysticks
[11] Dual Slope Integration

- ▷ **Delta-sigma yöntemi:** giriş sinyalinin ADC değerini, giriş geriliminin bir kondansatör üzerinde alınan tümlevini dengeleyecek dayanak gerilimi darbe sayısından bulur.

- ○ **Ardışık yaklaştırma yazmacı**[12] genellikle ses frekanslarından daha yüksek hızlarda dönüştürme yapmak üzere kullanılır. Bu yöntem en sol bitten başlayarak her çevrimde bir voltaj karşılaştırması yapıp bir bit ayrıştırarak istenen çözünürlüğe erişir.
- ○ **Anlık Dönüştürücü:**[13] Video frekansalarıyla videodan da yüksek frekanslarda, aynı anda mümkün olduğunca çok gerilim karşılaştırıcı kullanarak dönüşümü hızlandırır. Örneğin 5-bit anlık (flaş) dönüştürücüde toplam 31 örneksek karşılaştırıcı giriş gerilimini aynı anda 31 basamak değeriyle karşılaştırıp, 31 girişli bir enkoder devresiyle birkaç nanosaniyede beş bitlik çıkışa kodlar. Örneğin, iki anlık 5-bit dönüştürücü peşpeşe kullanılarak çevrim süresi 30-40 ns civarı olan 10-bit dönüştürücü oluşturulur.

ADC0804 yongası

ADC0804 8-bit çözünürlüklü tek kanallı ve 8080 veriyoluna doğrudan bağlanabilen bir SAR dönüştürücüdür. Uçlarının işlevleri şöyledir:



Şekil 10.9: ADC0804 yongasının uçları

- ○ $\overline{\texttt{CS}}$ düşük-etkin giriş: yongayı okuma ve yazma için etkinleştirir.
- ○ $\overline{\texttt{RD}}$ düşük-etkin giriş: yongadan dönüştürücü sonucunu veri yoluna aktarmayı etkinleştirir.
- ○ $\overline{\texttt{WR}}$ düşük-etkin giriş: dönüştürme işlemini başlatır. Dönüştürme işlemi toplam 16 saat çevrimi sürer.
- ○ `Vref/2` örneksel dayanak gerilimi girişidir. Dönüştürme tamamlanınca ADC0804 `Vin` örneksel girişinden

[12] Successive Approximation Register (SAR)
[13] Flash Converter

N_{ADC}= 255×(Vin+−Vin-)×Vref/2
sayısal dönüştürme sonucunu oluşturur.
- DB7..DB0 üçdurumlu veri çıkış uçları $\overline{\text{CS}}$ ve $\overline{\text{RD}}$ düşükken N_{ADC} sayısal dönüştürme sonucunu çıkarır. $\overline{\text{CS}}$ veya $\overline{\text{RD}}$ yüksek olduğu sürece yüksek empedans kalır.
- Vin+ ve Vin- örneksel giriş gerilimi uçlarıdır.
- DGND ve AGND sayısal ve örneksel toprak uçlarıdır.
- CLKIN ve CLKR uçları RC saat üreteci uçlarıdır. R=10k, C=150pF kullanıldığında saat hızı yaklaşık 500kHz olur ve ADC0804 8-bit dönüştürmeyi en çok 72 saat çevriminde tamamlar.
- $\overline{\text{INTR}}$ çıkışı örnekselden sayısala dönüştürme başlayınca yükselir, dönüştürme tamamlanınca düşer. $\overline{\text{INTR}}$ ucu düşer düşmez okuma yaparak sonuç en kısa gecikmeyle okunur.

ADC0804 Vin=Vin+−Vin- gerilimini sayıya dönüştürürken bir birim sayısal artışa karşılık gelen gerilim artışına basamak gerilimi denir. Basamak gerilimi ΔV ile gösterilir.

Örnek 10.11.

Vref/2=1.024 Volt için ADC0804'ün i) basamak gerilimi ne kadardır?, ii) okunan değer $N_{ADC} = 32$ ise Vin giriş gerilimi ne kadardır?

Çözüm: i) Basamak gerilimi

$$\Delta V = Vref/N_{ADCmax} = 2048/256 = 8\text{mV}$$

olur. ii) Bu ADC ile okunan N_{ADC}=32 değeri Vin= $\Delta V \times N_{ADC}$=0.256 V anlamına gelir.

Örnek 10.12.

ADC0804 yongasının dayanak gerilimini tam 10mV'a ayarlamak için Vref/2 ucuna kaç volt uygulamalıyız?

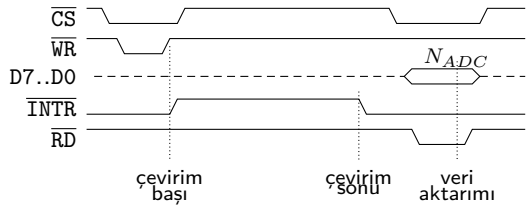
Çözüm: Vref/2 = 10mV × 256 /2 =2560/2 mV = 1.280 V.

ADC0804 dönüştürme çevrimi

ADC 0804'ün örnekselden sayısala dönüştürme çevrimi aşağıdaki basamaklardan oluşur.

- Dönüştürme çevrimi $\overline{\text{CS}}$ düşükken $\overline{\text{WR}}$ ucunun yükselen kenarında başlar. Bunun için işlemcinin ADC portuna herhangi birşey yazması, ya da eğer bu uç port üzerinden bağlıysa düşürüp yükseltmesi gerekir.

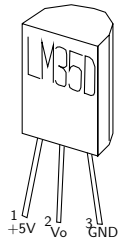
- Dönüştürme çevriminin başlamasıyla birlikte $\overline{\texttt{INTR}}$ ucu yukarı kalkar.
- Dönüştürmenin tamamlanmasıyla $\overline{\texttt{INTR}}$ aşağı iner. İşlemci cu-INTR ucunu sürekli test ederek, ya da bu ucu kesme kaynağı olarak kullanarak dönüştürmenin tamamlandığını anlayabilir. Eğer işlemcinin zaman kaybetmesi tolere edilebiliyorsa, $\overline{\texttt{INTR}}$ ucuna bakmayan işlemci, dönüştürme süresi olarak en az 150 μsaniye beklemelidir.
- Dönüştürme tamamlandıktan sonra $\overline{\texttt{CS}}$ düşükken $\overline{\texttt{RD}}$ düşük kaldığı süre boyunca ADC0804 N_{ADC}'yi veri çıkışlarına çıkarır. ADC0804 port olarak bağlanmışsa işlemci doğrudan okuma yaparak, bir port üzerinden bağlıysa $\overline{\texttt{CS}}$ ve $\overline{\texttt{RD}}$'yi düşürüp portu okuyarak N_{ADC} değerine erişir.



Şekil 10.10: ADC 0804 zamanlama diyagramı

LM35D ve ADC0804 ile sıcaklık ölçümü

Sıcaklık gibi fiziksel değişkenleri gerilime dönüştürmek için sıcaklık duyacı kullanırız. LM35D ortam sıcaklığını kolayca gerilime çevirmek üzere geliştirilmiş bir duyaç yongasıdır.

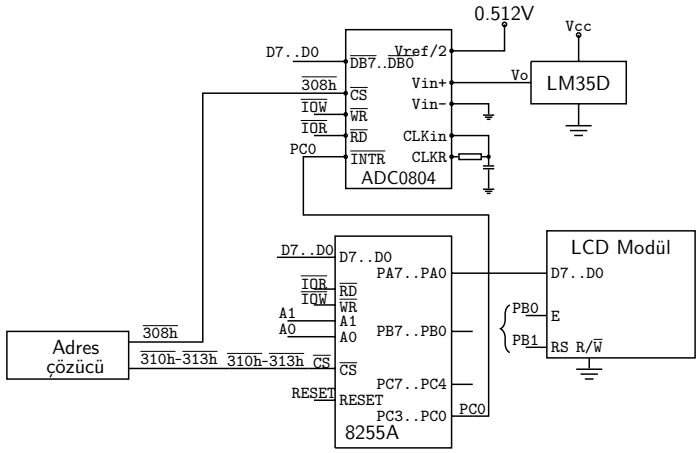


Şekil 10.11: LM 35D sıcaklık duyacının görünüşü

Santigrad Celcius sıcaklığı θ ile gösterirsek, LM 35, -55^{o}C ile 150^{o}C arasında en çok 0.5^{o}C hata ile $Vo = 10 \times \theta$mV çıkış verir. Eksi çıkış gerilimleri için Vo çıkışından -5V'a 100kΩ bağlamak yeterlidir. LM35D tipi olanlar ise, 0^{o}C ile 100^{o}C arasında $Vo = 10 \times \theta$mV çıkış sağlar. Fahrenheit ölçüm için ise 10mV/F kazanç ile LM34 (-50F .. 300F) ve LM34D (32F .. 200F) aralığını ölçer.

Örnek 10.13.

Bu örneğimizde `Vref/2`=0.512V dayanak gerilimli bir ADC0804'ü 0308h adresine bağlayacağız. Devremizdeki 0300h .. 0303h adreslerinde çalışan 8255'in PC0 ucunu ADC'nin $\overline{\texttt{INTR}}$ çıkışını okumak için kullanacağız. diğer portlardan PA'yı bir LCD modüle veri yollamakta, PB0 ve PB1'i ise LCD'nin `E` ve `RS` girişini sürmekte kullanacağız. Yazacağımız programın amacı ADC0804'ten okuduğumuz sayısal değerden sıcaklık değerini hesaplayıp LCD'de göstermektir.



Çözüm: Önce kodlama için gereken bazı bilgileri gözden geçireceğiz. Kodlamayı iki basamakta yapacağız. ilk basamakta ADC'den Nadc'yi okuyup sıcaklığı hesaplayacağız ve PA'ya yazacağız. İkinci basamakta sonucu PA'ya yazmak yerine LCD module aktarmak için gerekli LCDinit ve LCDnum prosedürlerini yazacağız.

Gereken bilgiler: Tc, ^{o}C biriminden sıcaklık olsun.

- ADC: ΔV= 512×2/256=4mV. Nadc= Vo / ΔV
- LM35D: Vo= 10 mV×Tc
- Tc(Nadc): Tc= Nadc/2.5 .
  Yalnızca tamsayı işlemleri yapabildiğimizden işlemi
  Tc =Nadc × 2 / 5 olarak hesaplamalıyız.

Port adresleri:

- Yapılandırma: 313h;
- PA: 310h; PB: 311h; PC: 312h;
- ADC başlatma ve okuma adresi: 308h;

LCD için:

- Saat hızı 4.77MHz olan işlemci 17 saat-çevrimlik loop komutunu CX=1000*4.77/17=280 kere çalıştırınca 1ms beklemiş olur. Yazacağımız dur1msan makrosu argumanı kadar msaniye bekleme sağlayacaktır.
- LCD'ye veri ve metin yollamak için yazdığımız prosedür ah=02h ile çağırılırsa al'yi denetim verisi olarak, ah=0 ile çağırılırsa al'yi metin verisi olarak yollayacak. Böylece örneğin mov ax,0280h den sonra çağırırsak LCD'ye denetim verisi olarak 80h gidecek. mov ax,'A' dan sonra çağırırsak LCD'ye 'A' yazacak.
- LCD'ye metin yollarken, veriyi PA'ya koyar. PC'den sırasıyla 0, 1, 0 ve 0 çıkararak RS=0 iken E darbesi üretiriz.
- LCD'ye denetim verisi yollarken, veriyi PA'ya koyar. PC'den sırasıyla 2, 3, 2, ve 0 çıkararak RS=1 iken E darbesi üretiriz.
- LCD'yi başlatırken 200ms bekleriz. Ardından sırasıyla denetim verisi olarak 38h yollar, 5ms bekler, 0Eh, ve 01h yollar, 2ms bekler, son olarak 06h yollarız.
- LCD'ye Tc'yi yazabilmek için Tc'yi 100'e ve 10'a bölerek onluk ascii sayıya çevirir Tca'ya koyarız. Sonra denetim verisi olarak 80h ile imleci başa taşır, metin verisi olarak ascii karakterleri sırayla yollayıp yazıyı oluştururuz.

Yapılacak işler:

- 8255'i yapılandır.
- LCD'yi başlat
- ana döngü içinde
  - ▷ ADC işlemini başlat
  - ▷ ADC'nin bitmesini bekle
  - ▷ ADC'yi Nadc'ye oku
  - ▷ Tc=Nadc×2/5 sayısını hesapla
  - ▷ Tc'yi LCD'ye yaz
  - ▷ Tc'yi PA portuna yaz.
  - ▷ Ana döngünün başına dön.

```
1  .model small
2  .stack 64
3  .data
4  Nadc db  ?
5  Tc   db  ?
6  Tca  db  ?,?,?
```

```
7
8    .code
9  main proc far
10   mov ax,@data
11   mov ds, ax
12
13 ;PPI yapılandırma
14   mov dx, 303h
15   mov al, 81h ; PCL giriş,
16   out dx,al
17
18 ;LCD'yi başlat
19   call LCDbaşlat
20
21 ; ana döngüye başla
22 anadongu:
23 ; ADC'yi başlat
24   mov dx, 308h
25   out dx,al
26 ; ADC'nin sonlanmasını bekle
27   mov dx,302h ; port PC
28 bekle:
29   in al,dx
30   and al,01h ; bit-0'a bak
31   jnz bekle  ; 1 ise bekle
32 ;ADC'yi oku
33   mov dx,308h ;ADC portu
34   in al,dx
35   mov Nadc,al
36 ;ikiyle çarpıp beşe böl
37   mov cl, 2
38   mul cl ; ax= Nadc . 5
39   mov cl, 5
40   div cl ; ax= Nadc . 2/5
41   mov Tc,al   ; sıcaklık
42   call LCDTc  ; LCD'ye yaz
43   mov al,Tc   ; sıcaklık
44   mov dx,300h ; port PA
45   out dx,al   ; PA'ya yaz
46 ;anadöngüye dönüş
47   jmp anadongu
48 main endp
49
50 proc LCDTc
51 ;Tc'yi ascii'ye dönüştür
52 ;ilk iki rakam boşluk olsun.
53   mov ax,2020h
54   mov word ptr Tca,2020h
55 ;Tc'yi ax'e yükle
56   mov al,Tc
57   xor ah,ah  ;ah=0
58 ;al<10 ise tekrakamlıdır
59   cmp al,10
60   jb rakam1
61   cmp al,100
62   jb rakam2
63 ;üçüncü rakamı bul
64   mov cl,100 ; al=ax/cl
65   div cl     ; ah=ax%cl
66   add al,30h ; ascii yap
67   mov byte ptr Tca,al
68 ;kalanı dönüştür
69   mov al,ah
70   xor ah,ah ;ah=0
71 rakam2:
72   mov cl,10
73   div cl
74   add al,30h
75   mov byte ptr Tca+1,ah
76   mov al,ah
77 rakam1:
78   add al,30h
79   mov byte ptr Tca+2,al
80
81 ;rakamları LCD'ye yaz
82 ; satır-1 sütun-1 den başla
83   mov ax,0280h ; denetim 80h
84   call LCDveer
85   xor ah,    ; metin yolla
86   mov al, Tca
87   call LCDveri
88   mov al, Tca
89   call LCDveri
90   mov al, Tca+1
91   call LCDveri
92   mov al, Tca+2
93   call LCDveri
94   ret
95 LCDTc endp
96
97 dur1msan macro msan
98   local L1
99   mov cx, 280*msan
100 L1:
101   loop L1
102   endm
103
104 LCDveri proc
```

```
105 ; AL  veri
106 ; AH  2:denetim, 0:metin
107   push ax
108   push dx
109   mov dx,300h ; veri portu
110   out dx,al   ; veri yollandı
111   mov dx,301h ; RS ve E portu
112   mov al,ah   ; E=0, RS=ah
113   out dx,al   ; çıkışa
114   inc al      ; E=1, RS=ah
115   out dx,al   ; E yükseldi
116   dec al      ; E=0
117   out dx,al   ; E düştü
118   mov al,0    ; RS düştü
119   out dx,al   ;
120   mov dx,300h ; veri portu
121   out dx,al   ; sıfırlandı
122   pop dx
123   pop ax
124   ret
125 LCD endp
126
127 LCDbaslat proc
128   dur1msan 200
129   mov ax,0238h
130   call LCDveri
131   dur1msan 5
132  mov ax,020Eh
133   call LCDveri
134   mov ax,0201h
135   call LCDveri
136   dur1msan 2
137   mov ax,0206h
138   call LCDveri
139   ret
140 LCDbaslat endp
141
142   end main
```

Yukarıdaki LCDveri prosedürünü makro olarak yazsaydık her bir karakter yazarken programa 18 komutluk kodu eklemiş olacaktık.

Ek A

Turbo ve MS Assembler ile çalışma

Turbo Assembler (TASM) 8086 nın yükseldiği dönemlerde en yaygın kullanılan çevirici yazılımıydı. TASM ile derlenecek yürütülebilir program dosyasının kaynak metnini örneğin ilk.asm gibi uzantısı -.asm olan bir dosyaya yazmak gerekir.

A.1 Kaynak Dosyası ve DOS Penceresi

TASM içindeki editör alıştığınız işlem kodlarını kullanmadığı için büyük ihtimalle size zorluk çıkarır. Kaynak metni dosyasını en kullanışlı bulduğunuz ASCII editör ile hazırlayabilirsiniz. Diyelim ki programınızı *ilk.asm* dosyasına yazdınız. TASM çeviriciyi kullanabilmek için start -> `run` -> `[command]` ile bir DOS işletim sistemi penceresi açmalısınız. TASM klasöründe başarılı açılmış bir DOS penceresinde şuna benzer bir mesaj göreceksiniz.

```
1 Microsoft Windows XP [Version ...]
2 (C) Copyright ....
3 C: ... \TASM>
```

A.2 TASM ile Obje ve Liste Oluşturma

TASM ın bulunduğu klasör içinde tüm TASM opsiyonlarını görmek için

```
>TASM /h
```

komutunu kullanabilirsiniz.
Bizim kullanacağımız opsiyonlar şunlardır

`/l`,`/la` : liste dosyası oluştur (`la` genişletilmiş liste).

`/z` : düzeltme hatası varsa ilgili satırı göster.

Örneğin `ilk.asm` dosyasının objesini oluşturmak için

```
... \TASM>TASM ilk.asm
```

ya da listeleme seçeneğini de çalıştırmak için

```
... \TASM>TASM ilk.asm \l
```

yazarız. TASM hatasız çeviri durumunda bize şu mesajları döndürür.

```
Turbo Assembler Version 1.0 ....

Assembling file: ilk.asm
error messages: None
Warning messages: None
Remaining memory: 455k

C: ... \TASM>
```

Tipik bir liste dosyasında şunları görürüz

```
Turbo Assembler  Version 1.0    02/25/08 17:26:10        Page 1
ILK.ASM
      1 0000                      .MODEL SMALL
      2 0000                      .STACK 64
      3 0000                      .DATA
      4 0000  52                  DATA1   DB 52H
      5 0001  25                  DATA2   DB 25H
      6 0002  ??                  SUM     DB ?
      7 0003                      .CODE
      8 0000                      MAIN    PROC FAR        ;program girisi
      9 0000  B8 0000s                    MOV AX,@DATA    ;data bolut par.
     10 0003  8E D8                       MOV DS,AX       ;  DS e konuldu
     11 0005  A0 0000r                    MOV AL,DATA1    ;ilk sayı
     12 0008  8A 1E 0001r                 MOV BL,DATA2    ;ikinci sayı
     13 000C  02 C3                       ADD AL,BL       ;ikisini topla
     14 000E  A2 0002r                    MOV SUM,AL      ;SUM a koy
     15 0011  B4 4C                       MOV AH,4CH      ;DOS a cık
     16 0013  CD 21                       INT 21H
     17 0015                      MAIN    ENDP
     18                                   END MAIN
```

Örneğin `.MODEL SMALL`, `.STACK` hiç kod oluşturmayan talimat örnekleridir. Veri bölütündeki değişkenlerden 4.satırda tanımlanan `DATA1`, 0000 ofsetinde bir baytlık yerdir. `DATA2`nin adresi ofset `0001` de, `SUM` ofset `0002`dedir. Kod bölütü ofseti `0000` dan başlar. 9.satırda `B8 0000s` kodu `MOV AX,anlık16` komutunun makine obje kodudur ve `0000s` ileriki aşamada bağlayıcı program tarafından gerçek değerine dönüştürülecektir. 10. satırda `0003` kod bölütündeki ofseti, `8E D8` bu ofsetteki

makine kodu, aynı satırdaki `MOV DS,AX` deyimi ise makine kodunun kaynak metnidir. Takip eden `; DS e konuldu` açıklaması kodlamaya girmemiştir. `MOV DS,AX` 8086'nın çoğu yazmaç komutları gibi iki baytlık bir komuttur. Buna karşılık örneğin `MOV AX,anlık16` bir baytlık komut ile 16-bitlik anlık işlenen olmak üzere üç baytla kodlanmıştır. Örneği incelersek 12.satırda iki bayt kod ve 16-bit adresle kodlanan `MOV BL,DATA2` komudunu görebiliriz. 11 ve 12. satırlarını karşılaştırırsak geçici yazmaç olarak `AL` yi kullandığımızda kodun kısaldığını da görürüz. İşlemcinin yürüteceği kodları önce komut yazmacına taşıması gerektiğinden kısa kod aynı zamanda hızlı koddur.

A.3 TLINK ile EXE dosyası oluşturma

Kaynak metinden `ilk.obj` dosyasını başarıyla oluşturduysak bir sonraki basamak TLINK bağlayıcısıyla bu obje dosyasını yürütülebilir dosyaya dönüştürmektir. Bunun için gene DOS penceresinde

```
1 ... \TASM>TLINK ilk
```

yazmamız yeterlidir. TLINK, obje içinde indekslenmiş adreslerin yerine objedeki etiket adreslerini yerleştirecektir. TLINK in daha önemli bir işi daha vardır. Eğer programımızda daha önceden objelenip kitaplığa yerleştirilmiş yordamlar çağırırsak TLINK bu yordamların kodunu programınızın sonuna ekleyip adresleri ona göre düzenler. TLINK yazdığımız DOS komutuyla bize `ilk.exe` dosyasını oluşturur. `ilk.exe` yi

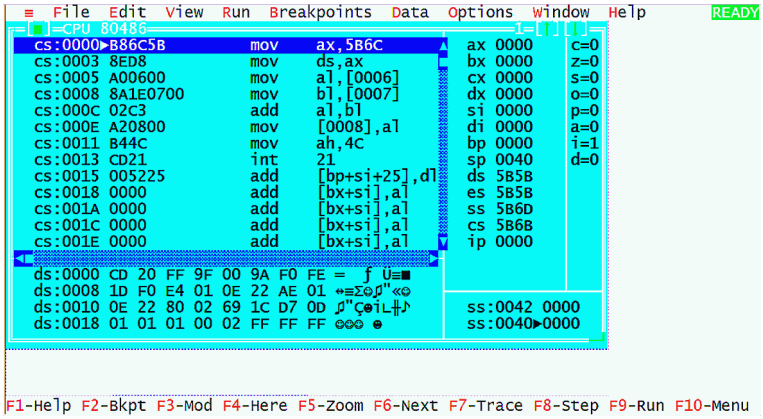
```
1 ... \TASM>ilk.exe
```

komuduyla çalıştırabiliriz. Ancak `ilk.exe` nin kullanıcıdan beklediği girişi ya da kullanıcıya ileteceği çıkışı olmadığından sanki hiçbir iş yapılmıyor gibi görünür. `ilk.exe` nin neler yaptığını ancak `debug` programlarıyla gözlemleyebiliriz.

A.4 TD ve EXE dosyasının takip edilmesi

TD içinde `ilk.exe` yi açabilmek için ya önce TD yi başlatıp sonra `File -> Load` ile `ilk.exe` dosyasını yükleriz, ya da

```
1 ... \TASM>TD ilk.exe
```



Şekil A.1: TD program kodunu, datasını ve işlemcinin yazmaçlarını gösterir.

yazarak TD yi başlatırken ilk.exe yi de yüklemesini bildiririz.

`ilk.exe` TD ye yüklendiğinde ekrana Şekil A.1 deki görüntü çıkar. `cs:0000` ile adreslenen alan kod bölütünde yer alan komutları gösterir. `ax, bx, ... ip` yazmaçların göründüğü penceredir. Aşağıdaki `ds:0000` ile başlayan bellek dökümü veri bölütündeki değişkenleri izleyebilmek içindir. Sağ alt köşedeki pencereden de yığıttaki olaylar gözlenir.

Şimdi `cs:0000` satırını incelersek `mov ax,5B6C` komutunun bellekte

`cs:0000 B86C5B`

olarak kodlandığını görürüz. `0000` ofsetten başlayarak işlem kodu `B8`, ve 16-bit anlık işleneni `5B6C` küçük uçtan dizilerek yer alıyor. Sağdaki küçük değerli bayt önce kodlandığından adres-`0001` e `6C` konuyor ve büyük değerli bayt onu izliyor. Böylece `B86C5B` ardışık üç bayt yer kaplıyor. Bir alttaki kodun adresi `0000+3` veya bölütüyle gösterirsek `cs:0003` oluyor. Küçük uçtan dizilmeyi `cs:0005` ve `cs:0008` den başlayan kodlarda da gözlüyoruz.

A.5 EXE nin TD de yürütülmesi

TD yürüteceği bir sonraki komutu küçük üçgen ile işaretler. F7, ve F8 tuşları kodu birer basamak yürütür ancak F7 yordam çağrılarını hızlı işler, F9 ise bir durma noktasına ulaşıncaya dek kodu tam hız çalıştırır. Durma noktasını koymak ve kaldırmak için F2 tuşu kullanılır.

A.6 MASM ile çalışma

MASM çevirici TASM dan çok az değişiklik gösterir. MASM bağlayıcı programına gereken etiketleri `CRF` dosyasına, kullanılmış bütün etiketlerin alfabetik dizinini de `MAP` dosyasına koyar. Bu dosyalar büyük kaynak programların modüller biçiminde bağlanması sırasında gerekir. Verdiğimiz örnek program hem TASM ile hem de MASM ile hiçbir değişikliğe gerek kalmadan çevrilir ve çalışılır.

A.7 TASM ile COM dosyası oluşturma

Aşağıdaki örnek TASM ile COM dosyası oluşturmak için bir şablon olarak kullanılabilir.

```
1  1 0000                csg  segment
2  2                       org 100h
3  3                       assume cs:csg,ds:csg,es:csg
4  4 0100                prgcode proc near
5  5 0100  EB 07 90        jmp start
6  6                     ;data area starts
7  7 0103  0956          data1 dw 2390
8  8 0105  3456          data2 dw 3456h
9  9 0107  ????          sum   dw ?
10 10                    start:
11 11                    ;code area starts
12 12 0109  A1 0103r       mov ax,data1
13 13 010C  03 06 0105r    add ax,data2
14 14 0110  A3 0107r       mov sum,ax
15 15 0113  B4 4C          mov ah,4ch
16 16 0115  CD 21          int 21h
17 17 0117              prgcode endp
18 18 0117              csg  ends
19 19                      end prgcode
```

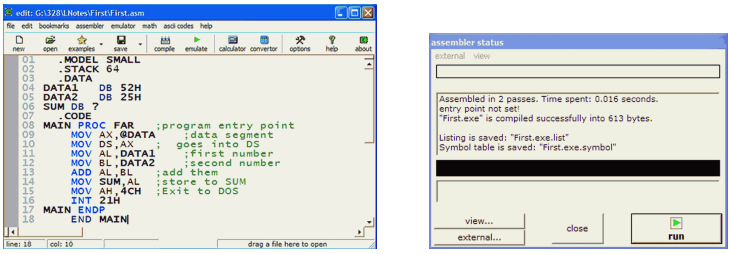
Ek B

EMU8086 Kod Geliştirme Ortamı

Kod geliştirme ortamları bir programı kaynak metinden başlayarak kullanıcıyı çevirme ve bağlama işlemleriyle yormadan COM veya EXE dosyası oluşturup oluşan makine kodunu aynı zamanda kaynak metin üzerinden takip ederek izleyebilmemizi sağlar. EMU8086 assembly program geliştirme amaçlı eğitsel bir araçtır.
`www.emu8086.com` sayfasından en son sürümünü indirip bilgisayarınıza kurabilirsiniz. Windows üzerinde geliştirilmiş olan bu yazılımı `ASM`, `OBJ`, `LST`, `EXE` veya `COM` dosyaları fareyle tutup EMU86 ikonunun içine bırakarak çalıştırabilirsiniz. ASM ve LST dosyalar metin editörünü, `OBJ`, `EXE`, ve `COM` dosyalar ise hata ayıklayıcı ünitelerini çalıştırmaya başlayacaktır.

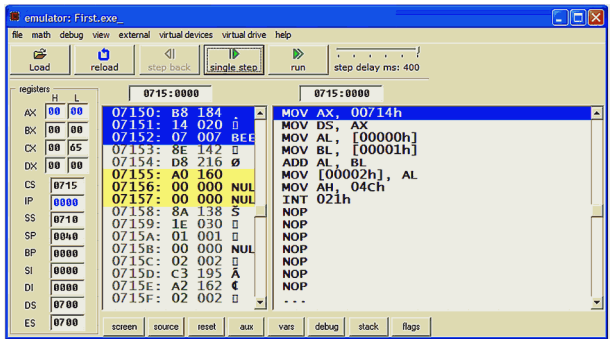
B.1 EMU8086 Metin Editörü

EMU86 Editörü komut anımsatıcılarını, hex sayıları ve etiketleri farklı renklendiren özel bir editordür. Metin hazırlandıktan sonra (compile) butonuna basarak kaynak metnin çevrilmesi ve bağlanmasını sağlayabiliriz. Yanındaki (emulate) butonu ise 8086 emulatorunu başlatacaktır.



Şekil B.1: EMU8086 Editörü ve çevirici bildirim penceresi

Emul8086 ortamı kullanışlı bir yardım donanımına sahiptir. COM ve EXE dosyası oluşturmak üzere şablonlar içerir. Önemli bir avantaji



Şekil B.2: Emulator penceresi

EMU8086 nın floppy disk, bellek, işlemci ve I/O portları dahil komple bir 8086 sistemini kendinize has bios ve boot programları dahil emule edebilmesidir. Windows ortamının kullanım kolaylığı sayesinde ASM programcılığına yeni başlayanlar için bile cazip bir ortamdır.

B.2 EMU8086 Çevirici Dilyapısı

EMU8086 dilyapısı[1] MASM ve TASM gibi yaygın çeviricilerin dilyapısına çok benzer. Ancak kendine has bazı ek talimatları vardır. Başka çevirici ile derlemek için EMU8086 kaynak metinlerindeki '#' ile başlayan talimatları açıklamaya dönüştürmek gerekir. En son `END` talimatının ardından giriş noktasını yazmak gerekir. Bunların dışında EMU8086 nın ürettiği kod TASM ve MASM koduyla 100% aynıdır. Ayrıca EMU8086 da `byte ptr` ve `word ptr` anahtar sözcükleri `b.` ve `w.` ile kısaltılabilir. Diğer çeviricileri kullanacak olanlar metinde gerekli değiştirmeleri yapmalıdır. `LEA` ile ilgili sorunlar çıkabilir. `LEA` ya alternatif komut olarak `MOV` tercih edilmelidir.

[1] syntax

Ek C

Sık Kullanılan Komutlar

8086 Assemblerinde yaygın olarak kullanacağımız komutları tanıtacağız.

C.1 İki işlenenli komutlar

Komut bir aritmetik-mantık işlemi yürütüyorsa genellikle *kaynak* işlemde kullanılan ama değeri değişmeyen anlık değer, bellek ya da yazmaç içeriğini, *hedef* ise sonuçların yazıldığı bellek ya da yazmacı gösterir. Özel durumlar haricinde kaynak ve hedef aynı uzunlukta (bayt ya da word) olmalıdır ve ikisi birden bellekte olamaz. Hedef8 ve kaynak8 yalnızca 8-bit verili komutlarda, *hedef16* ve *kaynak16* ise yalnızca 16 bit verili komutlarda kullanılmıştır.

MOV hedef, kaynak: Veriyi kaynaktan hedefe kopyalar.
Bayrakları etkilemez.

XCHG hedef$_1$, hedef$_2$: Hedef$_1$ ile hedef$_2$ işlenenlerinin değerlerini yerdeğiştirir.
Bayrakları etkilemez.

ADD hedef, kaynak: Hedef ve kaynak işlenenlerini toplayıp hedef işlenene koyar.
`CZSOPA` bayraklarını günceller.

ADC hedef, kaynak: Kaynak, hedef, ve daha önceki işlemlerden kalan elde (`CF`) bayrağını toplayıp hedefe koyar.
`CZSOPA` bayraklarını günceller.

SUB hedef, kaynak: Hedeften kaynağı çıkarıp sonucu hedefe koyar.
`CZSOPA` bayraklarını günceller.

SBB hedef, kaynak: Hedeften kaynağı ve elde bayrağını çıkarıp sonucu hedefe koyar..
`CZSOPA` bayraklarını günceller.

AND hedef, kaynak: İki işlenenin mantıksal 've' sonucunu hedef işlenene koyar.
`CO` bayraklarını sıfırlar, `ZSP` bayraklarını günceller.

OR hedef, kaynak: İki işlenenin mantıksal 'veya' sonucunu hedef işlenene koyar.
CO bayraklarını sıfırlar, ZSP bayraklarını günceller.

XOR hedef, kaynak: İki işlenenin mantıksal 'yada' sonucunu hedef işlenene koyar.
CO bayraklarını sıfırlar, ZSP bayraklarını günceller.

CMP kaynak$_1$, kaynak$_2$: Kaynak$_1$ i kaynak$_2$ den çıkarır. Çıkarma sonucunu hiçbiryere yazmaz ancak sonucuna göre CZSOPA bayraklarını günceller. Koşullu sıçramalardan önce kullanılır.

TEST kaynak$_1$, kaynak$_2$: İki işlenenin mantıksal 've' sonucuna göre ZSP bayraklarını günceller, CO bayraklarını herzaman sıfırlar. Koşullu sıçramalardan önce kullanılır.

C.2 Adres İşlenenli Komutlar

CALL adres: Altyordam çağırma komutudur. Dönüş adresini (CS:IP) yığıta koyup gönderideki adrese sapar. Dönüş için RET komutu gerekir.
Bayrakları etkilemez

JMP adres: Koşulsuz sapma komutudur. Adres kısa, yakın veya uzak tipte olabilir. Bayrakları etkilemez.

Jxx adres: Kısa adresli **xx** koşullu sıçramadır. Koşul gerçekleşirse adrese sapar, gerçekleşmezse bir sonraki adrese devam eder. Bayrakları etkilemez. **xx** şunlardan biri olabilir
Bayraklı koşullar TEST, CMP, ADD, SUB, AND vs. sonrası:
C, NC, Z, NZ, S, NS, O, NO, P, NP
N:değil, **C**:elde, **Z**:sıfır, **S**:negatif, **O**:taşma, **P**:eşlikli
işaretsiz sayılarla CMP sonrası
A, B, E, AE, BE, NA, NB, NAE, NBE
N:değil, **A**:yukarı, **B**:aşağı, **E**:eşit
işaretli sayılarla CMP sonrası
L, G, E, LE, GE, NL, NG, NBE, NGE
N:değil, **L**:küçük, **G**:büyük, **E**:eşit anlamınadır.

LOOP adres: CX i bir eksiltir. CX sıfır olmamışsa adrese sapar. Bayrakları etkilemez.

LOOPE adres: CX i bir eksilir. Hem CX sıfır değil hem de ZF sıfır değilse adrese sapar. Bayrakları etkilemez.

C.3 Tek işlenenli komutlar

INT anlık8: Yazılım kesmesi[1] komutudur. Bayrak yazmacını ve bir sonraki komutun adresini (CS:IP) yığıta yükleyip kesme vektörüne sapar. Dönüş için IRET gerekir.
CZSOPAI bayrakları etkilenmez, I bayrağını 1 yapar.

MUL kaynak8: AL ile bellek ya da yazmaçtaki işaretsiz sayıyı çarpar. Sonuç AX te durur.
CO bayrakları güncellenir, ZSPA etkilenir.

IMUL kaynak8: AL ile bellek ya da yazmaçtaki işaretli sayıyı çarpar. Sonucu AX te döndürür.
CO bayrakları güncellenir, ZSPA etkilenir.

MUL kaynak16: AX ile bellek ya da yazmaçtaki işaretsiz sayıyı çarpar. Sonuç DX;AX te durur.
CO bayrakları güncellenir, ZSPA etkilenir.

IMUL kaynak8: AX ile bellek ya da yazmaçtaki işaretli sayıyı çarpar. Sonucu DX;AX te döndürür.
CO bayrakları güncellenir, ZSPA etkilenir.

DEC hedef: bellekteki ya da yazmaçtaki değeri bir azaltır. ZSOPA bayraklarını günceller, C bayrağını etkilemez.

NOT kaynak: bellek ya da yazmaçtaki değerin birli tümleyenini ($\overline{M}$) alır. Bayrakları etkilemez.

NEG kaynak: bellek ya da yazmaçtaki değerin ikili tümleyenini $\overline{M}+1$ işlemiyle hesaplar. CZSOPA bayraklarını günceller.

DIV kaynak8: AX teki işaretsiz sayıyı bellek ya da yazmaçtaki işaretsiz sayıya böler. Sonuç AL de, kalan AH de durur.
Bayrakların tümü etkilenir.

DIV kaynak16: DX;AX teki işaretsiz sayıyı bellek ya da yazmaçtaki işaretsiz sayıya böler. Sonuç AX de, kalan DX te durur. Kaynak16 anlık olamaz.
Bayrakların tümü etkilenir.

IDIV kaynak8: AX teki işaretli sayıyı bellek ya da yazmaçtaki işaretsiz sayıya böler. Sonuç AL de, kalan AH de durur.
Bayrakların tümü etkilenir.

IDIV kaynak16: DX;AX teki işaretli sayıyı bellek ya da yazmaçtaki işaretli sayıya böler. Sonuç AX de, kalan DX te durur. Kaynak16 anlık olamaz.
Bayrakların tümü etkilenir.

[1] interrupt

PUSH kaynakyazmaç16: 16-bitlik kaynak yazmacı yığıta koyar.
Bayrakları etkilemez

POP hedefyazmaç16: Yığıttan 2-bayt indirip hedef yazmaca kopyalar.
Bayrakları etkilemez.

C.4 Bit Kaydırma ve Döndürmeler

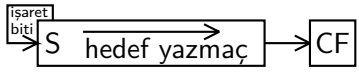
Bütün bit kaydırma ve döndürme komutlarında bir hedef ve bir bit sayısı işleneni bulunur. Hedef işleneni 8-bit ya da 16-bitlik bellek ya da yazmaç olabilir. Bit sayısı ya 1, ya da CL işleneniyle gösterilir. Bit sayısı 1 olduğunda kayma ya da döndürme yalnızca bir bit uygulanır. Bit sayısı CL olduğunda ise kaydırma ve döndürme CL MOD 16 kere uygulanır. Örneğin CL=00010110_2=38 durumunda 38 MOD 32 = 6 bit dönme ya da kayma olacaktır. Bütün kaydırma ve döndürmelerde CPZSO bayrakları güncellenir, A bayrağı etkilenir. İşaret biti değişirse O=1, değişmezse O=0 olur.

SHL hedef, 1 ; SAL hedef, 1 Hedef bellek ya da yazmacı sola bir bit kaydırır ve bit 0 a 0 koyar. En soldaki bit elde bayrağına kayar.

CPZSO bayrakları güncellenir.

SHL hedef, CL ; SAL hedef, CL Bir bitli kaydırma gibi, ancak yazmacı sola (CL MOD 32) bit kaydırır.

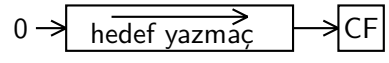
SAR hedef, 1 Hedef bellek ya da yazmacı sağa bir bit (CL deki sayının en az değerli üç biti kadar) kaydırır ancak işaret bitinin (en soldaki bit) değerini değiştirmez. bit-0 elde bayrağına kayar.



CPZSO bayrakları güncellenir.

SAR hedef, CL Bir bitli kaydırma gibi, ancak (CL MOD 32) bit kaydırır

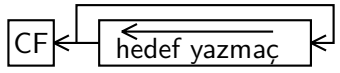
SHR hedef, 1 (SHR hedef, CL) Hedef bellek ya da yazmacı sağa bir bit (CL deki sayının en az değerli üç biti kadar) kaydırır ve en soldaki biti sıfırlar. Bit-0 elde bayrağına kayar.



CPZSO bayrakları güncellenir.

SHR hedef, CL Bir bit kaydırma gibi ancak (CL MOD 32) bit kaydırır.

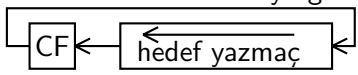
ROL hedef, 1 Hedef yazmaç ya da bellekteki bitleri sola doğru döndürür. En soldaki biti elde (C) bayrağına ve bit-0 a koyar.



CPZSO bayrakları güncellenir.

ROL hedef, CL Bir bit döndürme gibi ancak (CL MOD 32) bit döndürür.

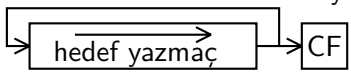
RCL hedef, 1 Hedef yazmaç ya da bellekteki bitleri elde bayrağıyla birlikte sola doğru döndürür. C bayrağındaki biti bit-0 a, ve en soldaki biti de elde bayrağına taşır.



CPZSO bayrakları güncellenir.

RCL hedef, CL Bir bit döndürme gibi ancak (CL MOD 32) bit döndürür.

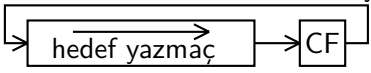
ROR hedef, 1 Hedef bellek ya da yazmacındaki bitleri sağa doğru döndürür. Bit-0 ı hem C bayrağına hem de en soldaki bite taşır.



CPZSO bayrakları güncellenir.

ROR hedef, CL Bir bit döndürme gibi ancak (CL MOD 32) bit döndürür.

RCR hedef, 1 Hedef yazmaç ya da bellekteki bitleri elde bayrağıyla birlikte sağa doğru sdöndürür. C bayrağındaki biti en soldaki bite, ve bit-0 bitini de elde bayrağına taşır.



CPZSO bayrakları güncellenir.

RCR hedef, CL Bir bit döndürme gibi ancak (CL MOD 32) bit döndürür.

C.5 İşleneni olmayan komutlar

NOP: İş yapmaz, sadece zaman geçmesine neden olur.

DAA: `DAS` gibi ancak `ADD` işlemini düzeltir.

DAS: `SUB` işlemini BCD kodlamaya göre düzeltir. Işlemin hemen ardından kullanılır.
`CA` bayraklarını etkiler.

AAA: `ADD` işlemlerini ASCII kodlamaya göre düzeltir. Işlemin hemen ardından kullanılır.
`CA` bayraklarını etkiler.

AAS: AAA gibi, ancak `SUB` komutunu düzeltir. `CA` bayraklarını etkiler.

AAM: `MUL` işlemini ASCII kodlamaya göre düzeltir.
`ZSM` bayraklarını etkiler.

AAD: `DIV` işleminden önce kullanıldığında bölme sonucunu ASCII kodlamaya göre düzeltir.
`ZSA` bayraklarını etkiler.

CBW: AL deki işaretli sayıyı AX e uzatır.

CLC: elde bayrağı C yi sıfırlar.

STC: elde bayrağı C yi sıfırlar.

IRET: kesmeden geri dönüş komutudur. IP, CS ve Bayrak yazmaçlarının yığıttan geri indirir ve böylece kesme servisinden programın kaldığı noktaya geri döner.

RET: CALL ile yığıta saklanmış dönüş adresini (CS:IP) yığıttan indirip dönüş adresine sapar.
Bayrakları etkilemez.

PUSHF: Bayrak yazmacını (FR) yığıta koyar.
Bayraklar etkilenmez.

POPF: Yığıttan bayrak yazmacına 2-bayt indirir.
Bütün bayraklar etkilenir.

C.6 I/O Port Komutları

IN AL, anlık8: 8-bitlik anlık port numarası verilen giriş portundaki 8-bit veriyi AL yazmacına kopyalar. Bayrakları etkilemez

IN AL, DX DX teki port numarası verilen giriş portundaki 8-bit veriyi AL yazmacına kopyalar. Bayrakları etkilemez.OUT anlık8, AL 8-bitlik anlık port numarası verilen çıkış portuna AL yazmacındaki 8-bit veriyi kopyalar. Bayrakları etkilemez

OUT anlık8, AL: anlık8 ile 8-bit port numarası verilen çıkış portuna AL yazmacındaki 8-bit veriyi kopyalar. Bayrakları etkilemez.

OUT DX, AL: DX te port numarası verilen çıkış portuna AL yazmacındaki 8-bit veriyi kopyalar. Bayrakları etkilemez.

Dizin