İLERİ MİKRODENETLEYİCİLER VE UYGULAMALARI

"Mustafa Engin" <mustafa.engin@ege.edu.tr>

EGE ÜNIVERSITESI EGE MESLEK YÜKSEKOKULU

Mekatronik Programı

İçindekiler

C Dili İle Mikrodenetleyici Programlama

Giriş

Hangi yazılım dilinde yazılmış olursa olsun bir kodun işlemci tarafından algılanabilmesi için makine koduna çevrilmesi gerekir. Makine dili hiyerarşinin en alt kısmını oluşturur ve çok detay içerir. Bu yüzden tasarımcı tarafından oluşturulması çok zordur. Hatta program büyüdükçe bu imkansız hale gelir. Makine dili doğrudan işlemciyi yönlendiren bir dil olduğu için daha güvenilirdir. Hiyerarşide yukarı çıkıldıkça detay azalır buna paralel olarak tasarım giriş süresi azalır. Bununla birlikte kod güvenilirliği azalır.

Yüksek seviye dillerde yazılan programların makine diline çevrilmesi için derleyiciler kullanılır. Derleyiciler metin veya grafiksel arayüzler ile girilmiş kodu makine diline çevirirler. Derleyiciler genelde iki basamaklı işlem ile bu dönüşümü gerçekleştirirler. Bunların ilki derleme işlemidir. Bu işlem metin dosyası ile yazılmış bir dosyayı nesne koduna(object code) çevirir. Nesne kodu çalıştırılabilir bir kod değildir. Ama bütün programlama dilleri için aynıdır. Derleyicilerin ürettikleri nesne kodları genellikle .obj veya .o uzantılı dosyalarda saklanır.

İkinci işlem bağlama işlemidir. Bu işlem için bağlayıcılar(linker) kullanılır. Bağlayıcı oluşturulan nesne dosyalarını ile kütüphaneden İhtiyat duyulan modüllerini toplar, bütün adresleri çözer ve bunları indirilebilir çıktı dosyası içerisinde birleştirir.

Gömülü C (Embedded C)

Etrafımıza baktığımızda, kendimizi çok çeşitli gömülü sistemlerle sarıldığını görürüz. Bu sistem bir dijital kamera, ya da cep telefonu veya bulaşık makinesi olabilir. Bu araçların hepsi kendilerine özgü çeşitli işlemcilere sahiptirler. Tabi ki bu işlemcilerle birlikte bir gömülü yazılıma gereksinim vardır. Donanımın gömülü sistemin vücudunu oluşturduğunu varsayarsak, gömülü sisteminin işlemcisi vücudun beynini, gömülü yazılım ise ruhunu oluşturur. Gömülü yazılım gömülü sistemin işleyişini belirler.

Microişlemci tabanlı sistemlerin ilk yıllarına bir göz attığımızda programların assembler kullanılarak geliştirildiğine şahit oluruz. O yıllarda yapılan geliştirmelerde programların ne yaptığını bulmak için bir mekanizma söz konusu değildi. Programların doğru çalışmasını kontrol etmek için led ve anahtar gibi harici cihazlar kullanılıyordu. Yalnızca bazı şanslı geliştiriciler simülasyon araçları kullanıyorlardı. Fakat bu araçlar çok pahalıydı ve çok güvenilir değillerdi.

Zaman ilerledikçe assembly kullanımı azalmaya başladı ve gömülü programla dili seçimi yavaş yavaş C dili üzerine kaymaya başladı. C, günümüzde gömülü işlemci ve denetleyicilerde kullanılan en yaygın programlama dilidir. Bununla birlikte çok yüksek zamanlama doğruluğu, kod boyutu verimliliği gerektiren yerlerde kodun bazı bölümlerinde assembly dili hala kullanılmaya devam etmektedir. C dili Kernighan ve Ritchie tarafından 8Kbaytlık bir boşluğa sığabilecek ve taşınabilir olan bir işletim sistemi yazmak amacıyla geliştirilen bir dildir. O yıllarda UNIX işlemci sistemleri üzerinde uygulanıyordu ve işlemci sistemi geliştirme amacıyla kullanılıyordu. Programcılara oldukça kompakt kod yazma imkanı sunuyordu. Bu özelliği hackerlar arasında dil kullanım seçiminde C diline ün kazandırmıştı.

Assembly dilinde oluşturulan programların işlemci özgü olması, programların sistemler arasında taşınabilir olmamasını neden oluyordu. Bu dezavantajı aşmak için, tasarımcıların gömülü sistemlerde C dahil olmak üzere birçok üst düzey dile yönelmelerine neden olmuştur. Bu geçiş aşamasında PLM, Modula-2 , Pascal gibi diller de gömülü sistemlerde kullanılmasına rağmen geniş kabul bulmamıştır. Bu dillerin arasında C yalnızca gömülü sistemlerde değil masaüstü uygulamalarında(bilgisayar) da geniş kabul gördü. Günümüzde C genel amaçlı uygulamalar için ana dil olma özelliğini kaybetmiş olsa da gömülü programlamada hala popülaritesini korumaktadır. C'nin Gömülü sistemlerinde geniş kabul görmesi nedeniyle derleyiciler ve ICE gibi destek araçları geliştirilmiştir. Bu ise gömülü sistemlerde C yi kullanan programlayıcıların işlerini oldukçe kolaylaştırmaktadır.

Gömülü Sistem Programlama

Gömülü sistem programlama (Embedded System Programming) masaüstü bilgisayarlar üzerinde geliştirilen uygulamalardan faklıdırlar. Bu iki sistem arasındaki ana farkları aşağıdaki gibi sıralayabiliriz.

- Gömülü cihazlar sınırlı kaynağa sahiptirler.(sınırlı ROM, Sınırlı RAM, düşük işlemci gücü, sınırlı yığın alanı gibi)
- Gömülü sistemler ile bilgisayarda kullanılan bileşenler farklıdır.
- Gömülü sistemlerde tipik olarak daha küçük ve daha az güç tüketen bileşenler kullanılır.
- Gömülü sistemler daha fazla donanıma bağlıdırlar.

Gömülü Programlamada göze çarpan en önemli iki özellik kodun hızı ve büyüklüğüdür. Kodun hızı işlemci gücü, zamanlama kısıtlamaları ile yönetilirken, kodun boyutu mevcut program hafızası ve kullanılan programlama diline bağlıdır. Gömülü sistem programlamadaki hedef Minimum alan ve minimum zamanda maksimum özellikli program elde etmektir.

Gömülü Sistemde Kullanılan Diller

Gömülü sistemle farklı tür diller kullanılarak programlanırlar.

- Makine kodu (Machine Code)
- Düşük seviyeli diller Low level language (assembly)
- Yüksek seviyeli diller (High Level Language) (c, c++,java, ada)
- Uygulama Düzeyi Diller(Visual Basic, Access)

Assembly

Assembly dili, assembly dilene özgü ifadeler ile ikilerden oluşan (binary) makine kodu arasında eşleştirmeyi yapar. Assembly dili gömülü cihazları programlamak için bariz bir seçim gibi görünür. Fakat bu dil kullanımı hız ve boyut açısından verimli kodlar geliştirme konusunda sınırlıdır. Ayrıca assembly kodları yüksek yazılım geliştirme maliyetine yol açar ve taşınabilirliği yoktur. Küçük kodların tasarımlarda genelde herhangi bir sorunla karşılaşılmaz.. Fakat assembly dili ile yazılmış büyük programlarda programı yönetmek giderek zor hale gelir. Ayrıca iyi bir assembly derleyicisi bulmak da günümüzde zor hale gelmiştir. Dolayısıyla günümüzde gömülü işlemci sistemi programlaması için yüksek seviyeli diller tercih edimektedir.

Gömülü C

Gömülü sistemlerde C 'nin kullanılmasının avantajlarını aşağıdaki gibi sıralayabiliriz.

- Öğrenilmesi oldukça basittir, (programlama, anlama, hata ayıklama)
- C derleyicileri günümüzde kullanılan neredeyse bütün gömülü cihazlar için mevcuttur. Assembly'nin aksine C'nin herhangi bir özel işlemci/microkontrolcü bağımlığı yoktur. Bu ise çoğu sistemler üzerinde koşabilecek programlar geliştirilmesinde C dilini kullanıcı için uygun hale getirir.
- C assembly dilinin işlevselliği ile yüksek seviye dillerin özelliklerini birleştirdiği için C, 'orta düzey bilgisayar dili' veya 'yüksek seviye assembly dili' olarak kabul edilir.
- Oldukça verimlidir.
- I/O erişimini destekler ve büyük gömülü projelerin yönetim kolaylığı sağlar.

Diğer dillerde bu avantajların çoğunu sunmaktadırlar. Fakat C dilini bu programlardan (Pascal, Fortran) ayıran özelliği C'nin orta seyiye bir dil olması ve yüksek seviye dillerin sağlamış oldukları yararlardan ödün vermeden doğrudan donanım kontrolü sağlamasıdır. Diğer yüksek seviye dillerle karşılaştırıldığında C daha fazla esneklik sunar. Çünkü C nispeten küçük yapılı bir dildir ve düşük seviyeli bit temelinde veri işlemesini destekler. Assembly diliyle karşılaştırdığımız zaman, C de kod yazımı daha güvenilirfarklı platformlar arasında daha taşınır bir özelliğe sahiptir."(bazı küçük değişikliklerle) Ayrıca C de geliştirilen programların anlaşılması, programın sürdürülmesi ve program üzerindeki hata ayıklama daha kolaydır. Program C de daha hızlı geliştirilebilir. C de iyi bir kod yazılıp, yüksek kaliteli derleyiciler yardımıyla etkin bir assembly koduna dönüştürülebilir. Ayrıca çoğu gömülü C programları içerisinde assembly yazılmasını destekler. Eğer kritik görülen bir nokta varsa bu kısmın C programına assembly olarak dahil edilir.

C++

Nesne yönelimli bir dil olan C++ gömülü aygıtlar gibi kısıtlı kaynağa sahip ortamlarda etkin programların geliştirilmesi için uygun değildir. C++'ın, Sanal fonksiyonları ve istisna işlemleri (Virtual functions & exception handling) gömülü sistemler içerisindeki hız ve alan açısından verimli olmayan bazı specific özellikleridir. Bazen gömülü sistem yazılımlarında C++ yalnızca bazı özellikleri kullanılır.

ADA

Nesne yönemli bir başka dil olan Ada, C++ dan faklıdır. Başlangıçta ABD DOD tarafından tasarlanan ada, Ada83 ve Ada 95 olmak üzere iki defa uluslararası standart (Ada83 ve Ada95) olarak kabul edilmesine rağmen popular olamamıştır. Fakat ada dili gömülü yazılım geliştirme işlemini basitleren birçok özelliği vardır.

Java

Gömülü sistem programlama için kullanılan diğer bir dilde java dilidir. Sistemler arası taşınabilirlik sunması ve tarama(browsing) uygulamalarında kullanışlı olması nedeniyle özellikle üst seviye cep telefonlarında tercih edilen bir dildir. Fakat programın çok kaynak tüketen Java Virtual Machine (JVM) ihtiyaç duyması, bu dilin küçük gömülü cihazlar için kullanılmasına engel olmaktadır.

Diğer Diller

Ayrıca Dinamik C ve C # de gömülü uygulamalarda kullanılan bazı özel dillerdir.

C Ve Gömülü C Arasındaki Farklar

Her ne kadar C ile gömülü C farklı uygulamalarda kullanılsalarda, yapı açısından bir çok benzerlikleri vardır. Gömülü C mikrodenetleyici tabanlı uygulamalar için C ise masaüstü bilgisayarlar için kullanılır. C dili bellek ve işletim sistemi gibi masaüstü PC'nin sağlamış olduğu kayanakları kullanabilmektedir. Bu yüzden bilgisayardaki programlamada bellek konusunda herhangi bir sıkıntı yaşanmaz. Bunun yanında Gömülü C, gömülü işlemci üzerindeki sınırlı kaynakları kullanmak zorundadır.(RAM, ROM,I/Os) Bunun için program kodunun mevcut program belleğine sığdırılması gerekir. Kodun sınırı aşması durumunda ise sistemin çökmesi kaçınılmazdır.

C derleyicileri (ANSI C) genellikle işletim sistemi üzerinde yürütülebilir dosyalar üretir. Gömülü C derleyicileri ise doğrudan mikroişlemciler içerisine yüklenecek dosyayı oluştururlar. Masaüstü Bilgisayar uygulamalrında sağlanmayan bütün kaynaklara erişim gömülü derleyiciler tarafından sağlanır. Gömülü sistemler masa üstü bilgisayar uygulamalarının aksine genellikle geçek zamanlı kısıtlamalara sahirtirler. Gömülü sistemler genellikle masaüstü uygulamalarında bulunan bir konsola sahip değildirler. Özetlemek gerekirse gömülü C ile programlamada kaynakların en optimum şekilde kullanılması, yazılan kodun verimli hale getirilmesi, gerçek zamanlı kısıtlamalara uyulması gibi özellikler gömülü C'nin belirgin farklarıdır.

Genel C Dilinin Kurulları

C dilinde bir program oluşturmak bir anlamda ev yapmaya benzer. Ev yapımı için temel hazırlanır. Daha sonra çimento ve kum kullanılarak tuğlalar yapılır. Ardından bu tuğlalar bir düzen içerisinde birleştirilerek duvarlar inşa edilir. Bütün duvarlar bitirildikten sonra evi inşa etmiş oluruz. Gömülü C'de de bir takım komutlar birleştirilerek fonksiyonlar oluşturulur. Daha yüksek düzeyde işlem olarak kabul edilen bu fonksiyonlar birleştirilerek program elde edilir.

Bütün C dili programları **main()** adında en az bir fonksiyona sahip olmak zorundadır. Main fonksiyonu C programının temelini oluşturur ve program kodunun yürütülmesine bu fonksiyondan başlanır. Programda bulunan bütün fonksiyonlar doğrudan veya dolaylı olarak main fonksiyonu tarafından çağrılır. Main fonksiyonu program başladığı zaman sistem tarafından çağrılan ilk fonksiyon olduğu için, üzerine düşük seviye görev yüklenir. Çoğu durumda main fonksiyonun işlevi programı başlangıç durumuna getirmek ve fonksiyonlar arası ilişkileri düzenlemekten öteye geçmez.

Bir gömülü C'nin en basit şekli aşağıdaki gibidir.

```
#include <stdio.h>
int main()
{
        printf ("HELLO WORD"); /* klasik C test programı*/
        while(1); // sonsuza kadar işle
        return 0;
}
```

Bu program "HELLO WORD" ifadesini seri port olarak tanımlanmış çıkışa gönderecek ve işlemci resetlenene kadar program askıya alınacaktır.

Anlatımımıza yukarıdaki örneği incelerek devam edelim.

#include <stdio.h> ifadesi ile C'nin standart Giriş/Çıkış kütüphanesini (stdio.h) programa dahil edilmiş olur. Derleyici stdio.h dosyasını bu programın bir parçasıymış gibi algılar. Böylece bu kütüphanede tanımlanmış bütün tür, komut ve fonksiyonlar program içerisinde kullanılır hale getirilmiş olur. Bu sayede stdio kütüphanesinde tanımlanmış bir fonksiyon olan printf() fonksiyonunu program içerisinde kullanabildik.

return 0; ifadesi program sonlandığında işletim sistemine (eğer işletim sistemi kullanılıyorsa) 0 değerini gönderir.

Genel C Kuralları

1. Her bir main() adında en az bir fonksiyona sahip olmak zorundadır. Programın yürütülmesine main() fonksiyonundan başlanır.
2. Programda bir ifadenin sonunu göstermek için Noktalı virgü (;) kullanılır. Bir ifadenin en basit formu noktalı virgülün kendisidir.

```
init_platform();
printf("Hello World\n\r");
cleanup_platform();
```

3. Bir fonksiyon içeriğinin başlangıç ve bitiş noktalarını belirtmek için süslü parantez {} kullanılır.

```
int main()
{
}
```

4. " " çift tırnak bir metin dizesinin başlangıç ve bitiş noktasını göstermek için kullanılır.

```
char isim[]= “FPGA”;
```

5. Programa eklenecek başlık dosyaları için **#include** deyimi kullanılır. **#include** deyimi sonlandırılması için noktalı virgül (;) kullanılmaz. Eklenecek dosya ismi **<** **>** veya **“** **“** simgelerinin arasına yazılır. Genel kullanım ANSI C'deki standart başlıkları için < > simgesinin, kullanıcı tarafından oluşturulan başlık dosyalar için “ “ simgesinin kullanılması şeklindedir.

```
#include < stdio.h>
#include  “platform.h”
```

ANSI C'deki standart başlık dosyaları şunlardır:

assert.h	locale.h	stddef.h
ctype.h	math.h	stdio.h
errno.h	setjmp.h	stdlib.h
float.h	signal.h	string.h
limits.h	stdarg.h	time.h

6. **Tanımlayıcılar**: Değişken veya fonksiyon isimlerine tanımlayıcı adı verilir. Tanımlayıcı adları harf veya _ altçizgi ile başlamak zorundadır. Ardından harf, rakam veya alt çizgi ile devam ederler.

   Tanımlayıcı isimleri belirlenirken aşağıdaki yöntemler tavsiye edilir.

   - Tanımlayıcı isimleri anlamlı kelimelerden seçilmeli ve yeterince uzun olmalıdır. İsimler birkaç kelimeden oluşacak ise kelimler arasına _alt çizgi konulmayı yada kelimelerin baş harfleri büyük yazılmalıdır.

     ```
     int udp_gonderme_fonksiyonu();
     void UdpGondermeFonksiyonu();
     ```

   - Sabit tanımlayıcıların bütün harflerinin büyük yazılması, program içerisinde tanımlayıcının sabit olduğunu gösterecektir.

     ```
     #define ADRESS 0x0009
     const int BAUDRATE= 9600;
     ```

     Tanımlayıcılar küçük büyük harf duyarlılıkları vardır. Aşağıda ifadelerdeki temp tanımlayıcıları birbirinden faklıdır.

     ```
     int temp;
     int Temp;
     int TEMP;
     ```

   Tanımlayıcılar herhangi bir uzunlukta olabilir. Bu kullanılan derleyiciye bağlıdır. Bazı derleyiciler yalnızca sınırlı sayıda karakteri tanırlar. (ilk 25 karakter gibi) Bu yüzden tanımlayıcı tanımlarken dikkatli olunması gerekir.

7. C dili serbest biçimli bir dildir. Bu yüzden program içerisindeki tırnak içerisinde gösterilen boşluklar hariç diğer bütün boşluklar göz ardı edilir. Programdaki boşlukların düzenli olması programın anlaşılabilir olmasını sağlar.

   Düzensiz kod

```
int main(){printf ("Selam C"); while(1); /*sonsuza kadar işle*/ return 0;}
```

   Düzenli kod

```
int main()
{
    printf ("Selam C");
    while(1); /*sonsuza kadar işle*/
    return 0;
}
```

8. C'de açıklamalar için **//** veya **/*...*/** simgeleri kullanılır. Açıklamalar yazılan koda dahil olmayıp, programlayıcının not ve uyarılarını içerir. Açıklamalar programlar için kritik olup, programın okunabilirliği artırır. Programa eklenmiş doğru yorumlar programın diğer programlayıcılar tarafından anlaşılması sağlar yada daha sonra kendi programlayıcısının programda ne yaptığını hatırlamasına yardımcı olur.

   Geleneksel yorum sınırlayışı **/*…*/** 'dır. Slash-yıldız blok yorumları oluşturmak için kullanılır. Derleyici bir "/*" ifadesi ile karşılaştığında ""*/"" ifadesine kadar olan bütün metinleri açıklama olarak kabul eder.

```
/* bu ifadeler
daha sonra kullanılacak */ return 0;
```

   Diğer taraftan Slash-slash (//) sınırlayıcısı sadece önüne konulduğu satırın sonuna kadar olan metinleri yorum olarak algılattırır.

```
//bu ifadeler
//daha sonra kullanılacak
```

   Bu iki sınırlayıcı arasındaki fark **//** simgesi tek bir satırı kapsarken, /*..*/ simgesi ise bu iki ifade arasındaki metinleri (birden çok satır olsa bile) kapsaması.

9. C dilinde derleyici için özel anlama sahip sözcükler bulunur. Bu sözcüklere özel amaçlı sözcükler denir. Özel amaçlı sözcükler tanımlayıcı olarak kullanılamazlar ve program içerisinde küçük harflerle yazılmaları gerekir. Aşağıda C de kullanılan özel amaçlı sözcükleri bulabilirsiniz.

auto	default	extern	inline	signed	typedef
break	defined	flash	int	sizeof	union
bit	do	float	interrupt	sfrb	unsigned
case	double	for	long	sfrw	void
char	eeprom	funcused	register	static	volatile
const	else	goto	return	struct	while
continue	enum	if	short	switch	

Değişkenler ve Sabitler

Çok az bir program önceki bölümde verdiğimiz örnek kadar basit olabilir. Çoğu programda çıkış değerleri üretilmeden önce bir dizi hesaplamalar yapılır. Bu hesaplamalar esnasında verilerin geçici olarak saklanacağı alanların olması gerekir. C dilinde bu saklama alanlarına değişkenler denir. Veriler program içerisinde değişken ve sabit biçimlerinde kaydedilir. Değişkenler değerleri değiştirilebilen verilerdir. Sabitler ise değişkenlerin aksine atama yapıldıktan sonra değeri değiştirilemeyen verilerdir. Değişken ve sabitler faklı boyut ve biçimde program belleğine kaydedilirler.

Veri Türleri

Bütün değişkenlerin ne çeşit veri tutacağını belirten bir türünün olması gerekir. Değişkenler gömülü sistemin kısıtlı belleğine kaydedilir. Bu yüzden gömülü sistemde değişken tanımlanırken gereksiz yere bellek alanı israf edilmemesi gerekir. Bu yüzden tasarımcı değişken bildirimi esnasında değişkenin türünü tayin etme konusunda hassas davranması gerekir.

C programlama dilinde üç tane temel veri türü bulunur.

- char
- int
- float

Bu veri türlerin önünde *short, long, signed* ve *unsigned* özel niteleyicileri kullanılarak veri türlerinin türevleri oluşturulabilir.

İşaretsiz (unsigned) ön eki kullanılarak, verilerin değerlerinin sıfır ve sıfırdan büyük olması sağlanabilir. İşaretli ve işaretsiz verilerin bellekteki uzunlukları aynıdır. Fakat işaretsiz türdeki verilerin üst limiti, işaretli türündekinin iki katına eşittir.

Char Türü

Char türü "0-9" ve "A-Z" gibi yazdırılabilir yada "enter,tab,yeni satır" gibi yazdırılamaz karakterlerden oluşur.
char veri türünü ve bellekteki kapladığı alanı aşağıdaki tabloda görebilirsiniz.

Tür	Boyut (Bit türünde)	Değer Aralığı
char	8	-128 ile 127
unsigned char	8	0 ile 255
signed char	8	-128 ile 127

Int türü

Tamsayıları ifade eder. int veri türünü ve bellekteki kapladığı alanı aşağıdaki tabloda görebilirsiniz.

Tür	Boyut (Bit türünde)	Değer Aralığı
int	16	-32768 ile 32767
short int	16	-32768 ile 32767
unsigned int	16	0 ile 65535
signed int	16	-32768 ile 32767
long int	32	-2147483648 ile 2147483647
unsigned long int	32	0 ile 4294967295

Float Türü

Gerçek sayıları ifade eder. double ve float olmak iki şekilde tanımlanır. float veri türünü ve bellekteki kapladığı alanı aşağıdaki tabloda görebilirsiniz.

Tür	Boyut (Bit türünde)	Değer Aralığı
float	32	-3.4e +/- 38 ile +3.4e +/- 38
double	64	-1.7e +/- 308 ile +1.7e +/- 308

Değişken Bildirimi

Bir değişkenin programda kullanılabilmesi için program içerisinde bildiriminin yapılması gerekir. Değişkenin bildirimi, değişkenin türünü ve boyutunu gösteren özel amaçlı bir sözcük ve bir tanımlayıcı ile yapılır.

Özel Amaçlı Sözcük	Tanımlayıcı
unsigned int	adres

```
long int numara;
```

Değişken Kapsamı

Değişkenlerin programda kullanılabilmesi için programın başında bildirimlerinin yapılması gerekir. Bir değişkenin kapsamı program içerisindeki erişilebilirliğidir. Değişken program içerisinde yerel yada genel olarak tanımlanabilir.

- **Yerel Değişkenler:** fonksiyon tanımlandığı zaman, fonksiyon tarafından ayrılan bellek alanıdır. Bu değişkenler sadece tanımlandığı fonksiyon içerisinde anlamlıdır ve diğer fonksiyonlar tarafından kullanılamaz. Derleyici yerel değişkeni yalnızca tanımlandığı fonksiyonun bir parçası olarak gördüğü için, farklı fonksiyonlarda aynı isme sahip yerel değişkenler tanımlanabilir.
- **Genel Değişkenler:** Derleyici tarafından ayrılan bellek alanları olup, bütün fonksiyonlar tarafından kullanılabilir.

Örnek

```
#include <stdio.h>
int sayi;//Genel değişken
int main()
{
    int sayi_mod_2;//Yerel değişken
    int sayi_mod_4;// Yerel değişken
    sayi=1000;
    sayi_mod_2 = sayi >> 1;
```

```
        sayi_mod_4= sayi >>2;
        printf("mod_2   = %d\n\r mod_4    = %d",sayi_mod_2,sayi_mod_4);
return 0;
}
```

Sabitler

Sabitler program boyunca değeri değiştirilemeyen nesnelerdir. Sabitler birçok durumda derleyici programın parçası gibi davranırlar ve RAM'den ziyade ROM da saklanırlar.

X+5 atamasında *"5"* bir sabittir ve derleyici tarafından doğrudan toplama işlemine sokulur.

printf ("selam"); ifadesinde *"selam"* metni program belleğine yerleştirilir ve değeri değiştirilemez.

Programlayıcıda const özel sözcüğünü kullanıp sabit bildirimi yapabilir. Sabit bildiriminde const özel sözcüğü ile birlikte tanımlayıcı ismi, türü ve değeri belirtilmelidir.

```
constant  unsigned char k =80;
```

Bir nesneyi sabit olarak tanımlamak nesnenin kısıtlı depolama alanı bulunan RAM yerine program kodunda saklanmasını neden olur. Bu ise sınırlı RAM alanının korunmasına yardımcı olur.

Sayısal Sabitler

Sayısal sabitler, sayısal tabanlarını göstermek ve programı daha okunabilir hale getirmek için farklı şekillerde bildirimleri yapılabilir. Örneğin sabitleri aşağıdaki gibi bildrimlerini yapabilirz.

124	Ön Eksiz Ondalık gösterim
0b10001	**0b** ön ekli ikili gösterim
0xAAFF	**0x** ön ekli hexadecimal gösterim
898**U**	**U** son ekli unsigned integer
5698**L**	L son eki almış Long integer
569**UL**	**UL** son eki almış unsigned long integer
0.256F	**F** son eki almış floating point

float türünde olan 32 sabitinin gösterimi aşağıdakilerden biri olabilir.

32.0	32.	32.0e0	32E0	3.2e1	3.2e+1	.57e2	570.e-1

Karakter Sabitleri

Karakter sabitleri "0-9" ve "A-Z"gibi yazdırılabilir yada "enter,tab,yeni satır" gibi yazdırılamaz karakterlerden oluşur. Yazdırılabilir karakter sabitleri tek tırnak içerisinde gösterilir('C'). Ayrıca karakter gösterimi ters slash ile birlikte temsil edilen sekizlik veya onaltılık değerlerini yazılarak da yapılabilir.

K karakterinin sekizli gösterimi '\113' , onaltılık gösterimi '\x4B' dir.

Ters slash ile tırnak karakterlerinin gösteriminde ise bu karakterlerin önüne ters slash konulur. Örneğin tırnak için gösterim '\'', ters slash için gösterim'\\' şeklindedir.

Karakterlerin kodlarının bulunduğu ASCII tablosuna buradan ulaşabilirsiniz.

Örnek

```
const char temp='\x4B';
printf("Sabit== %c\n\r",temp);
```

Yukarıdaki ifade ekrana K karakterini basacaktır.

Tür Dönüştürme

Bazen programda tanımlanmış bir türün farklı türlere çevirme ihtiyacı doğabilir. Bu durumda tür dönüştürücüler kullanılarak var olan bir tür istenilen türe dönüştürülür.

Gösterimi dönüştürülecek türün parantez içerisinde yazılması ile şeklindedir.

(tür ismi)Tanımlayıcı

Örnek

```
int a=10;
char b='K';
int z;
.....
z= a +(int)b;
```

Sabit ve Değişken İsimleri

Sabit Ve Değişken İsimleri verilirken bazı kurallara uyulmak zorundadır. Bunlar

- Değişken ve sabit isimleri en fazla 32 karakterden oluşabilir. 32 karakterden uzun isimlerde ilk 32 karakter kayda alınır . Geriye kalan karakterler işleme tabi tutulmaz.
- Değişken ve sabit isimleri alt çizgi, ingiliz alfabesinde bulunan karakterler (A-Z) veya (a-z) , rakamlar (0-9) içermelidir. Türkçe karakterler, özel karakter veya boşluk karakteri kullanılamaz.
- Değişken ve sabit isimleri herhangi bir rakam ile başlayamaz. İlk karakter bir harf veya alt çizgi olmalıdır. Sonrakiler rakamlardan oluşabilir.
- C 'nin özel sözcükleri sabit veya değişken ismi olarak kullanılamaz.

Örnek:

temp	Doğru
_adres	Doğru
1adres	Yanlış(rakamla başlamış)
adres2	Doğru
sayı	Yanlış (Türkçe kelime kullanılmış)

sizeof Operatörü

Belirli bir tür değerinin kaydedilebilmesi için gerekli olan bellek alanının gösterir. Gösterim şekli aşağıdaki gibidir.

sizeof(tür ismi)

İşlem sonucunda işleme giren türün byte cinsinden değerini veren *unsigned integer* türünde bir değer elde edilir.

Örnek

```
#include <stdio.h>
int main()
{
printf("Float boyutu = %d\n\r",sizeof(float));
      printf("Double boyutu = %d\n\r",sizeof(double));
return 0;
}
```

Yukarıdaki örneğin çıktısı aşağıdaki gibi olur.

```
Float boyutu = 4
Double boyutu = 8
```

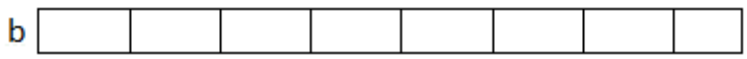
Diziler (Arrays)

Diziler elemanları aynı tür değerlerden oluşan veri yapılarıdır. Dizi elemanlarının her biri dizi içerisindeki konumlarına göre ayrı ayrı seçilebilir ve üzerlerinde işlem yapılabilir.

C dilinde tek boyutlu ve çok boyutlu olmak üzere iki çeşit dizi bulunur.

Tek Boyutlu Diziler

En basit dizi türüdür. Bu tür dizilerin elemanları kavramsal olarak düzenleniş tek bir satırdan oluşur. Örneğin aşağıda b adında tek boyutlu bir diziyi görselleştirebiliriz.

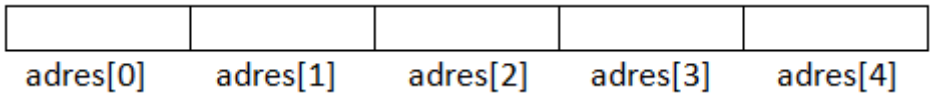


Bir dizinin program içerisinde kullanılabilmesi için bildiriminin yapılması gerekir. Dizinin bildirimi ile dizi ismi ile birlikle dizi elemanların sayısı ile türü belirtilir.

Örneğin aşağıda karakter türünde 5 elemandan oluşan adres adında bir dizi tanımlamış oluruz.

```
char  adres[5];
```

Dizinin herhangi bir elemanına erişmek için, dizinin ismi ile köşeli parantez içerisinde dizi elemanının yerini gösteren indeks numarasını yazmak yeterlidir. İndeks numarası, 0 ile dizi boyutunun bir eksiğine eşittir. Örneğin yukarıdaki adres[5] örneğini incelediğimizde dizinin indeksi 0 ile 4 arasında olur. Dizi elemanlarının gösterimi ise adres[0], adres[1]… adres[4] olur.



Örnek

Aşağıdaki kod parçasında adres dizisinin 3. elemanına 'c' karakteri atanıyor ve printf fonksiyınu ile bu değer ekrana gönderiliyor.

```
adres[2]='c';
printf("Adres gostergesi = %c", adres[2] );
```

For döngüsü ile dizi arasında yakın bir ilişki bulunur. Örneğin aşağıdaki örnekte dizinin her bir elemanına '0' karakteri atanır.

```
for(i = 0;i < dizi_boyutu;i++)
adres[i]='0';
```

Örnek

Aşağıdaki kod parçasında sivas olarak ilk değer girilen adres dizisinin her bir karakteri ekrana gönderiliyor.

```
adres[5]="sivas";
for(i = 0; i < 5; i++)
printf("%c", adres[i]);
```

Örnek

Aşağıdaki örnekte 10 elemana sahip bir dizinin elemanlarının toplamını veren bir kod bulabilirsiniz.

```
#define dizi_uzunlugu 10
int sayi[dizi_uzunlugu]
int toplam;
for(i = 0; i < dizi_boyutu; i++)
    toplam+=sayi[i];
```

for döngüsü kullanılırken döngünün nerede bitirileceğine dikkat edilmesi gerekir. for(i = 0; i <=dizi_boyutu; i++) ifadesi bazı derleyicilerde döngünün yanlış çalışmasına neden olacaktır.

Dizi indeksi sonucu tamsayı olan herhangi bir ifade de olabilir. Eğer indeks olarak ifade kullanılıyorsa, ifade sonucunun 0 ile (dizi boyutu-1) aralığında olmasına dikkat edilmelidir.

```
adres[sayi*2 - i]
```

Diziye İlk Değer Atama

Bildirimi esnasında diziye ilk değer ataması yapılabilir. Dizilere değer atama işleminin en yaygın biçimi değerlerin süslü parantez içerisinde aralarında virgül konularak yazılması şeklindedir. Bu şekilde yapılan ilk değer atamasında dizi elemanlarına sırasıyla parantez içerisindeki değerler atanır.

```
char  adres[5]= {'s','i','v','a','s'};
```

Yukarıdai bildirimi yapılan dizinin elemanlarının alacağı değerler aşağıdaki gibi olur.

```
adres[0]='s'; adres[1]= 'i'; adres[3]='v'; adres[4]='a'; adres[5]='s';
```

Örnek

```
int sayi[3]={3,2,1};
```

Eğer bildirim esnasında ilk değer olarak atanan değerler dizinin eleman sayısından az ise dizinin diğer elemanları varsayılan olarak 0 değerini alır.

Örnek

```
int sayi[8]={1,2,3,4};
```

bu dizinin elemanlarının ilk değerleri

```
int sayi[8]={1,2,3,4,0,0,0,0}; şeklinde olur.
```

Bu özellik bir dizinin bildirimi esnasında bütün elemanlarını 0 yapmak için kullanılır.

```
int sayi[8]={0}; //sayi dizisinin bütün elemanlarını 0 yap.
```

Dizinin bildirimi esnasında ilk değer olarak boş girilmesi uygun değildir. Bu yüzden ilk değer ataması gerekiyorsa yalnızca 1 tane 0 yazılması uygundur.

```
int sayi={ }; Yanlış
int sayi{0};
```

İlk değer atamasında atanan değerlerin dizinin eleman sayısından fazla olması da uygun değildir. Bu durumda derleyici program fazla olan değerleri göz ardı edecektir.

```
char  adres[5]= {'s','i','v','a','s','m','n','k'};
```

Yukarıdaki ifade de m,n ve k karakterleri derleyici tarafından işleme alınmayacaktır.

İlk değer ataması yapılan dizilerin bildirimleri esnasında dizi boyutunun yazılmasına gerek yoktur. Bu durumda derleyici program, dizinin boyutunu atanan değerlerin sayısından anlayacaktır. Böyle bir durumda dizinin bütün değerlerine atama yapılması gerekir.

```
int sayi[]={4,5,6,7,8};
```

Dizinin Belli Elemanlarına İlk Değer Ataması

Bazen dizinin yalnızca bazı elemanlarına atama yapılması gerekebilir. Bu durumda atama dizi elemanın indeks numarası kullanılarak yapılır. Bu atama şeklinde dizi elemanları sırayla yazılmasına gerek yoktur.

```
int sayi[10]={ [1]=5, sayi[8]=9, sayi[3]=7 };
```

Dizi işlemlerinde sizeof operatörünün kullanılması

Sizeof operatörü ile bir dizinin bellekte kapladığı alan bayt olarak hesaplanabilir. Örneğin char adres[5] şeklinde bildirimi yapılmış bir dizinin sizeof(adres) işlemi sonucu 5 olur. int sayi[5] şeklinde bildirimi yapılmış bir dizinin sizeof(sayi) işlemi sonucu 20 olur.

Ayrıca dizinin her bir elemanının bellekte kapladığı alan da hesaplanabilir. Örneğin sizeof(adres[1]) işlemi sonucu 1 , sizeof(sayi[1]) sonucu ise 4 olacaktır.

Bir dizinin boyutu sizeof(dizi /sizeof(herhangi bir dizi elemanı) işlemiyle bulunabilir.

Örnek:

Aşağıdaki kod parçası ile 5 elemanı bulunan tamsayı türündeki bir türün bütün elemanları 10 yapılır.

```
for(i=0 ;i<sizeof(sayi)/ sizeof(sayi[0]); i++)
sayi[i]=10;
```

Yukarıdaki teknik sayesinde dizinin uzunluğu sonradan değiştirilse bile işlem sonucu değişmeyecektir.

Çok Boyutlu Diziler

Birden fazla boyutlu dizilerdir. Aşağıda 5x7 kapasitesinde iki boyutlu bir dizi örneği görebilirsiniz.

```
int a[5][7]
```

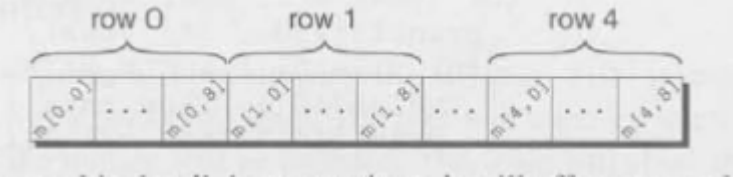
Bu dizinin gösterimi aşağıdaki gibidir.

	0	1	2	3	4	5	6
0							
1							
2							
3							
4							

Çok boyutlu dizilerde dizi elemanlarına ulaşım için indeks numaraları kullanılır. Dizide dizinin boyutu kadar indeks numarası bulunur.

Örneğin a çok boyutlu dizisinin 2 . satır 3. sütununda ki elamanına ulaşmak için a[2][3] yazılır.

Her ne kadar a[5][7] dizisini yukarıdaki gibi görselleştirilse de, dizinin gerçek bellekte saklanması biraz faklıdır. C dizileri bellekte satır sırasını baz alarak saklar. Örneğin yukarıdaki a dizisi bellekte aşağıda verilen şekilde saklanır.



Çok boyutlu dizilerde işlemler için genelde iç içe geçmiş for döngüsü kullanılır.

Örnek:

Bu örneğimizde birim matris oluşturalım. (Birim matris, köşegeni (diagon) 1 ve geri kalan sayıları 0 olan karesel matristir)

```
#define N 10
int birim_matris[N][N];
int i,k;
 for(i=0; i<N; i++)
{
    for(k=0; k<N;  k++)
    {
        if (i==k)
            birim_matris[i][k]=1;
        else
           birim_matris[i][k]=0;
   }
}
```

Çok Boyutlu dizilere ilk değer ataması

Çok boyutlu dizinin bildirimi sırasında diziye ilk değer ataması yapılabilir.

Örnek:

Aşağıdaki gösterimde iki boyutlu diziye yapılmış ilk değer ataması yapılmıştır. Bu atamada a dizisinin ilk elemanı olan a[0][0] değeri

```
int a[3][4] = { { 1, 1, 1, 0},      // a[0] satırındaki elemanlara atama yapılır.
                { 1, 0, 1, 0},      // a[1] satırındaki elemanlara atama yapılır.
                { 0, 1, 1, 0} };    // a[2] satırındaki elemanlara atama yapılır.
```

İlk değer atamasında atama yapılmamış elemanlar 0 ile doldurulur. Aşağıdaki örnekte a[0][4]'e ilk değer ataması yapılmadığı için bu değer 0 olur. Ayrıca a[3] satırı için atama yapılmadığı için bu satırın bütün elemanları 0 olur.

```
int a[3][4] = { { 1, 1, 1},
                { 0, 1, 1, 0} };
```

İlk değer atamasında satırlara karşılık gelen süslü parantezler yazılmayabilir.

```
int a[3][4] = { 1, 1, 1, 0,
                1, 0, 1, 0,
                0, 1, 1, 0 };
```

Yukarıdaki dizi bildirimini aşağıdaki gibi yazabiliriz.

```
int a[3][4] = { 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0 };
```

Dizi bildirimi esnasında sadece istenilen elemanlara ilk değer ataması yapılabilir. Atama yapılmayan değerler varsayılan olarak 0 değerini alır.

```
int a[3][4]= {[1][2]=10, [3][4]=20}
```

Örnek:

Bu örneğimizde girilen bir sayının tekrar eden rakamı varsa"Tekrar eden rakam var", yoksa "Tekrar eden rakam yok" ifadesini ekrana göndereceğiz.

Örneğin 354321 sayısı girildiğinde program “Tekrar eden rakam var” ifadesini ekrana gönderecektir.(3 den dolayı)

```
#include "stdio.h"
#define var 1
#define yok 0
int rakam_gorulme[10]= {yok};
long int sayi;
int rakam;
int main(void)
{
    printf("Bir sayi giriniz\n\r");
    scanf("%ld",&sayi);
    while(sayi>0)
    {   rakam = sayi % 10;
        if (rakam_gorulme[rakam]) break;
        rakam_gorulme[rakam]=var;
        sayi/=10;
    }
    if (sayi>0)
        printf("Tekrar eden rakam var");
    else
        printf("Tekrar eden rakam yok");
   return 0;
}
```

Örnek;

Yukarıdaki örneği biraz genişletelim ve tekrar eden rakamları ekrana yazdıralım. Örneğin 354321 sayısı girildiği zaman ekrana 3 rakamını yazalım.

```
#include "stdio.h"
#define yok 0
```

```
#define var 1
#define tekrar 2
int rakam_gorulme[10]= {yok};
long int sayi;
int rakam;
int i;
int main(void)
{
printf("Bir sayi giriniz\n\r");
scanf("%ld",&sayi);
    while(sayi>0)
    {
        rakam = sayi % 10;
        switch (rakam_gorulme[rakam])
        {
            case yok:
               rakam_gorulme[rakam]=var;
               break;
            case var:
               rakam_gorulme[rakam]=tekrar;
               break;
            default:
               break;
       }
       sayi/=10;
    }
    for(i=0; i<10; i++)
    {
        if(rakam_gorulme[i]==tekrar)
            printf(" %d\n\r",i);
    }
    return 0;
}
```

Program Yapıları

Bu bölümün amacı çalışan programı en kısa yoldan yazdırmaktır. İyi program belirlenen amacı gereksiz komut kullanmadan ulaşan ve işletimi sırasında oluşacak hatalara karşı koruması olan programdır.

Böyle program geliştirmenin birkaç yöntemi vardır, özellikle öğrenme aşamasındaki programcıların kullanabileceği en uygun yöntem structered programlamadır. Bu teknik aşağıdaki adımlardan oluşur.

- Akış diyagramı
- Pseudo kod
- Assembly dili

Akış diyagramı ve pseudo kod görsel işaretlerdir. Programın mantığını oluşturmak için kullanılır. Bu aşamalar problem çözülürken yapılacak işlemleri belirlemek için kullanılır, işlemin nasıl yapılacağı bu adımlarda bahsedilmez. Pseudo kod yapılacak işlemleri sırasıyla emir cümleleri seklinde yazmaktır. Assembly dili ise işlemcinin kabul ettiği kodlarda programın

yazılmasıdır.

Problem Çözme

Problem çözmede, soruna hemen girişmek yerine, dikkatli ve sistematik yaklaşım ilke olmalıdır. Problem iyice anlaşılmalı ve mümkün olduğu kadar küçük parçalara ayırılmaladır.

Descartes tarafından "Discourse on Method" isimli kitabında anlatılan problem çözme teknikleri.

- Doğruluğu kesin olarak kanıtlanmadıkça, hiçbir şeyi doğru olarak kabul etmeyin; tahmin ve önyargılardan kaçının.
- Karşılaştığınız her güçlüğü mümkün olduğu kadar çok parçaya bölün.
- Düzenli bir biçimde düşünün; anlaşılması en kolay olan şeylerle başlayıp yavaş yavaş daha zor ve karmaşık olanlara doğru ilerleyiniz.
- Olaya bakışınız çok genel, hazırladığınız ayrıntılı liste ise hiçbir şeyi dışarıda bırakmayacak kadar kusursuz ve eksiksiz olsun.

Algoritmalar

Belirli bir görevi yerine getiren sonlu sayıdaki işlemler dizisidir. İ.S. 9.yy da İranlı Musaoğlu Horzumlu Mehmet (Alharezmi adını araplar takmıştır) problemlerin çözümü için genel kurallar oluşturmuştur. Algoritma Alharezmi'nin Latince okunuşu.

Her algoritma aşağıdaki kriterleri sağlamalıdır.
Girdi: Sıfır veya daha fazla değer dışarıdan verilmeli.
Çıktı: En azından bir değer üretilmeli.
Açıklık: Her işlem (komut) açık olmalı ve farklı anlamlar içermemeli.
Sonluluk: Her türlü olasılık için algoritma sonlu adımda bitmeli.
Etkinlik: Her komut kişinin kalem ve kağıt ile yürütebileceği kadar basit olmalıdır.

Not: Bir program için 4. özellik geçerli değil. işletim sistemleri gibi program sonsuza dek çalışırlar .

Örnek 1.2.1 : 1'den 100'e kadar olan sayıların toplamını veren algoritma.

1. Toplam T, sayılar da i diye çağırılsın.
2. Başlangıçta T'nin değeri 0 ve i'nin değeri 1 olsun.
3. i'nin değerini T'ye ekle.
4. i'nin değerini 1 arttır.
5. Eğer i'nin değeri 100'den büyük değil ise 3. adıma git.
6. T'nin değerini yaz.

Algoritmaların yazım dili değişik olabilir. Günlük konuşma diline yakın bir dil olabileceği gibi simgelere dayalı da olabilir. Akış şeması eskiden beri kullanıla gelen bir yapıdır. Algoritmayı yazarken farklı anlamlar taşıyan değişik şekildeki kutulardan yararlanılır. Yine aynı amaç için kullanılan programlama diline yakın bir (sözde kod = pseudo code) dil, bu kendimize özgü de olabilir, kullanılabilir.

Aynı algoritmayı aşağıdaki gibi yazabiliriz.

1. T=0 ve i=0
2. i'nin değerini T'ye ekle.
3. i'yi 1 arttır.
4. i<101 ise 2.adıma git.
5. T'nin değerini yaz.

Yer Göstericiler (Pointers)

Assemblerda dolaylı adreslemede R_i yazaçları ile Dptr yazacı yer gösterici olarak kullanılır.

```
MOV R0,#40      ;Adresi belirle
MOV A,@RO       ;Belirlenen belleği oku

MOV R0,#40      ;Adresi belirle
MOVX A,@RO      ; Belirlenen belleği oku
MOVX A,@DPTR    ; Belirlenen belleği oku

CLR A
MOV DPTR,#0040 ; Adresi belirle
MOVC A,@A+DPTR ;Belirlenen belleği oku
```

Bu yazaçların yer gösterdiğini @ işareti belirtir.

C'De Yer Göstericiler

Değişkenlerin veya dizilerin bulunduğu yerleri gösteren pointerler C'de aşağıdaki şekilde tanımlanır. Veri veya sabitten ayırt etmek için pointerın önüne "*" işareti konur.

```
unsigned char *pointer0 ;
```

Assembler örneklerinde olduğu gibi iki işlemin yapılması gerekiyor.

1. Dolaylı olarak adresleyecek olan yazaca adres bilgisi yüklenir.
2. Uygun adresleme kipini kullanarak hedef seçilen adresteki veri okunur.

```
/* 1 - adreslenecek değişkeni belirle */
   unsigned char *pointer ;

/* 2 - dolaylı adreslenecek değişkenin adresini pointere yükle */
   pointer = &c_variable ;

/* 3 - '0xff' verisini dolaylı olarak değişkene yaz.*/
   *pointer = 0xff ;
```

C'de aynı veriyi doğrudan x = y, şeklinde veya dolaylı x = *y_ptr, şeklinde ulaşabiliriz. Bir örnek problem ile gösterelim.

```
/* Pointerın kullanımı */

void function(void)
{
unsigned char c_variable ;     // 1 - c değişkenini belirle, unsigned char
                       // 2 - pointerı tanımla (henüz bir adresi
işaretlemiyor)

  c_variable = 0xff ;          // 3 - değişkeni 0xff 'e eşitle
                               // veya pointer ile yap:
  ptr = &c_variable ;          // 4 - pointerı c_variable göstermesini zorla
  *ptr = 0xff ;                // 5 - 0xff'i değişkene yaz.
}
```

Not: satır 4'te pointer deişkene işaret etmektedir bunu yaptırmanın diğer bir yolu aşağıda gösterilmiştir.

```
/* pointerın kullanımına diğer bir örnek*/

void function (void)
{
unsigned char c_variable;          // 1- c değişkenini belirle
unsigned char *ptr = &c_variable; // 2- pointerı belirle ve c değişkenini
                                        işaret etsin
c_variable = 0xff ;            // 3 - değişkeni 0xff yap.
                               // veya pointerı kullanarak.
   *ptr = 0xff                 // 5 - değişkene 0xff bilgisini dolaylı yaz.
}
```

Pointerlar * işaretleri ile beraber diğer veri tipleri gibi kullanılabilirler.

Örnek:

```
x = y + 3 ;
```

işlemi şöyle gerçeklenebilir;

```
char x, y ;
char *x_ptr = &x ;
char *y_ptr = &y ;

*x_ptr = *y_ptr + 3 ;
```

veya;

```
x = y * 25 ;

*x_ptr = *y_ptr * 25 ;
```

Şu iki gösterim birbirinden farklıdır.

```
*ptr = var ;
```

anlamı poınterın dererini "var" yap. Diğer taraftan;

```
ptr = &var ;
```
Bunun anlamı ise pointer “var” değişkeninin adresini işaretliyor.
Sonuç olarak pointer şöyle ayarlanır;
```
ptr = &var ;
```
pointerın gösterdiği adrese şöyle erişilebilir;
```
var = *ptr ;
```

Pointerların doğrudan adresi göstermesi

C’de ROM, RAM ve çevrebirimleri sabit adreslerdedir. Bu bilgi hemen aklımıza pointerı nasıl bu sabit adresi gösterecek şekilde ayarlayabiliriz? Sorusunu getirir.

Bunun basit yöntemi pointerın doğrudan adrese eşitlenmesi şeklindedir.

```
char *abs_ptr = 0x8000 ;//pointerı belirle ve 0x8000’ a eşitle
```

örnek:
```
char *abs_ptr ; //pointerı tanımla
abs_ptr = (char *) 0x8000 ; // pointer 0x8000’ü göstersin.
*abs_ptr = 0xff ; // 0xff’i 0x8000’e yaz.
*abs_ptr++ ; //pointer bir sonraki adresi göstersin
```

Diziler ve Pointerlar

Değişkenlerin tanımlanması

```
unsigned char x ;
unsigned char y ;
```

8 bit bellek olduğunda:

```
unsigned int a ;
unsigned int b ;
```

a ve b değişkenleri 2 bayta yerleştirilir.

C'de diziler şöyle tanımlanabilir;

```
unsigned char a[10] ;
```

bu tanımlama ‘a‘ için 10 bayt ayırır bu alana herhangi bir başka veri yazılamaz.

Dizinin genel kullanımı

```
unsigned char test_array [] = { 0x00,0x40,0x80,0xC0,0xFF } ;
```

Başka şekillerde dizilerin C’de tanımlanması;

```
char test_array[] = { "HELLO!" } ;
```

C'de dizile arka arkaya dizili sayılardan oluşur elemanlar arası herhangibir ayraç yoktur.

```
char test_array = { "HELLO!" } ;
char *string_ptr = { "HELLO!" } ;
```

birinci tanımlamada "HELLO!". Yazısının ASCII kodları oluşturulacaktır. İkincide de aynı şey yapılacaktır, bunlara ek olarak pointer ataması da yapılmıştır.

İkinci tanımlama gerçekte aşağıdaki gibidir.

```
char test_array = { "HELLO!" } ;
```

ve işlenmesi aşağıdaki gibi olacaktır.

```
char arr_ptr = test_array ; //
char arr_ptr = &test_array[0] ;
```

Dizilerin Kullanımı

Örnek:

```
/* karakter katarının diziye kopyalanması*/
unsigned char source_array[] = { "HELLO!" } ;
unsigned char dest_array[7];
unsigned char array_index ;
array_index = 0 ; // ilk karakter indeksi
while(array_index < 7)   //dizinin sonunu denetle
{
    dest_array[array_index] = source_array[array_index] ;
    // karakter karakter boş diziye aktar.
    array_index++ ;  // bir sonraki karakter indeksi
}
```

Pointer ve diziler birbiri ile çok ilgilidir ve yukarıdaki programı aşağıdaki gib de yazabiliriz.

```
/* karakter katarının diziye kopyalanması*/
char *string_ptr = { "HELLO!" } ;
unsigned char dest_array[7] ;
unsigned char array_index  ;
array_index = 0 ;               //ilk karakterin indeksi
while(array_index < 7)          //dizinin sonunu denetle
{
   dest_array[array_index] = string_ptr[array_index] ;
    // karakter karakter boş diziye aktar.
    array_index++ ;
}
```

Deney Kartının Yapımı

Giriş

Donanım deneylerinde devre kurulduktan sonra gerekli yazılım program belleğine yüklenir. Deneme çalıştırması yapılır, beklenen sonuca ulaşıldı ise devre bağlantısında ve yazılımda herhangi bir hata yoktur. Eğer beklenen sonuç alınmadı ise öncelikle yazılan programın mantığı doğru mu? Denetlenmeli, bu işlemi programı önce simülatörde işletilerek yapabilirsiniz. İkinci aşamada ise işlemcinin programlanmasının doğru yapılıp yapılmadığıdır. Programcının bağlantısı ve tümdevrenin yerleştirilmesi doğru yapılmadı ise, işlemcinin programlama işlemi gerçekleşmez. Bu işlemin gerçekleşip gerçekleşmediği programlayıcı bağlantısı kontrol edildikten sonra, program denetleme (check, verify) yapılarak belirlenebilir.

Denetleme işlemine önce işlemcinin boş olup olmadığına bakılarak başlanır, eğer boş ise programla işlemi yapılamamıştır. Boş değilse, programlama yapılmış fakat hatalı yapılmış olabilir. Karşılaştırma (verify) yapılarak programın doğruluğu denetlenebilir, karşılaştırma işleminde bilgisayarda yazılı ***.HEX uzantılı dosya içeriği ile işlemcinin içeriği karşılaştırılır. Karşılaştırma sonucu hata var ise programlama tekrar yapılır, yoksa sorunun kaynağı yazılım değildir.

Yazılımın doğru olduğuna emin olduktan sonra kuşkular donanıma yönelmelidir. Bağlantılar, gücün elemanlara gelip gelmediği, elemanların kutupları denetlenmelidir. Bunlardan sonuç alınmadı ise işlemcinin osilatör devresi osilaskop yardımıyla denetlenmelidir. Denetleme sonuçları olumlu ise uygulamada kullandığınız eleman bozuk olabilir, sağlamlığı denetlenmeli veya yenisi ile değiştirilmelidir. Yapılan bu işlemler sonucunda devreniz çalışacaktır. Yapılacak işlemleri kısaca aşağıdaki gibi özetleyebiliriz:

- İşlemcinin doğru programlandığından emin olun.
- Gücün tüm devre elemanlarına bağlı olduğunu denetleyin.
- Tüm bağlantıları ve eleman kutuplarının bağlantılarını denetleyin.
- Osilatörün çalıştığını denetleyin.
- Kullandığınız elemanı başka devrede test edin, gerekiyorsa yenisi ile değiştirin.

NOT: Deneylerde kullanılacak 8051 kartının açık şeması ve baskı devre şemasını http://sorubank.ege.edu.tr/~mengin

İşlem Sırası

Baskı devre kartını ayrıntılı olarak inceleyin, üretim hataları varsa bunları kesici ve ince uçlu aletle temizleyin. Emin olmadığınız kısa devreleri ölçü aleti ile denetleyin.

- Devre elemanlarını 25 W'lık havya ile lehimleyin. Tümdevreler için soket kullanın.
- Tümdevreleri takmadan önce devrenin besleme gerilimini ölçün.
- Port 1'in 0 nolu uçuna LED bağlayın.
- P1.0'daki LED'i bir saniye yakan ve bir saniye söndüren bir test programı yazın.
- İşlemciyi programlamak için programlama kablosunu bilgisayarın paralel portuna takın ve Atmel ISP programını çalıştırın.
- Yazdığınız test programının test05.hex uzantılı dosyasını programın buffer'ına yükleyin ve işlemciyi programlayın.
- Programdan RUN TARGET'ı seçin, LED 1 saniyede aralıklarla yanıp sönecektir. Eğer

bu işlem gerçekleşmiyor ise deney kartınız çalışmıyor demektir!

- Öncelikle osilatörün çalışıp çalışmadığını denetleyin. Daha sonra besleme gerilimini denetleyin. (osilaskop ile saat işaretini gözlemleyin)
- Reset tuşunun doğru çalışıp çalışmadığını denetleyin. (osilaskop ile saat işaretini gözlemleyin)
- Bu denetlemelerin sonucu olumlu ise kartın çalışması gerekir.
- Çalışmıyor ise işlemcinin tüm bacaklarının gerilim seviyelerini osilaskopta gözleyin katalogu kullanarak normal olup olmadığını belirleyin.
- Devreniz çalışıyor diğer kısımlarının montajını tamamlayın.
- Test programını tekrar çalıştırın.

Giriş/Çıkış Portlarının Kullanımı

Giriş

8051'in portlarının diğer bazı mikroişlemcilerde olduğu gibi port hattının sürücü denetleme devresi yoktur. Port hattının giriş veya çıkış olarak mı kullanılacağı programcı tarafından seçilemez. Bu bazı programcılar tarafından yadsınabilir, fakat birçok uygulamada kolaylık sağlayacaktır. Şekil-1'de port hattına basmalı anahtarın bağlantısı gösterilmiştir. Bu devrede anahtara basılı olmadığı durumda hat iç yükseğe çekme devresi tarafından YÜKSEK seviyeye çekilecektir. Bu durumda bu hattın seviyesi "1" olarak okunur. Anahtara basıldığında hat DÜŞÜK seviyeye çekilecektir. Bu anda hattın seviyesi "0" olarak okunur. Anahtar bırakıldığında tekrar "1" seviyesine döner.

Bazı port okuma komutları tutucu çıkışını bazıları ise giriş ucunu okurlar. Tutucu çıkışını okuyan komutlar genellikle okudukları değeri değiştirip tekrar tutucuya yazarlar. Bu tür komutlara oku-değiştir-yaz komutları adı verilir. Bu komutların hedefleri portun tüm hatları veya bir hattı olabilir. Listedeki son üç komut oku-değiştir-yaz komutu değildir. Diğerleri belirtilen tutucu baytı veya bitini okur işlemi yaptıktan sonra aynı yere yazar. Oku-değiştir-yaz komutlarının çıkış ucu yerine tutucuya yazmalarının sebebi uçtaki gerilim seviyelerinin yanlış bir şekilde birbirlerine karışmasını engellemek içindir. Örnek olarak çıkış ucunun bir transistörün beyzini sürdüğünü varsayalım, çıkış ucuna "1" yazıldığında transistör iletime geçtiğinde çıkış ucunun gerilim seviyesi mantık "1"olarak kabul edilen 2,4 voltun altına düşebilir. Bu durumda oku-değiştir-yaz komutu tutucu yerine çıkış ucunu okusa gerçek değeri olan "1" değil "0" okuyacaktır ve hatalı işlem yapılacaktır.

Komut	Yaptığı işlem
ANL	Mantık VE
ORL	Mantık VEYA
XRL	Mantık ÖZELVEYA
JBC	Bit=1 ise temizle ve dallan.
CPL	Bitin değilini al tekrar yaz.
INC	Baytı bir arttır
DEC	Baytı azalt
DJNZ	Baytı bir azalt, sıfır değilse dallan.
MOV Px.y, C	Elde bayrağını port hattına yaz.
CLR Px.y	Port hattını sıfır yapar.
SET Px.y	Port hattını kurar.

8051'in portlarının bit adreslenebilir olması uygulamalar için diğer bir başka kolaylıktır. Diğer port hatlarının seviyelerini değiştirmeden istediğiniz tek bir port hattını tek bir komutla kurabilir, temizleyebilir veya tersleyebilirsiniz.
Sayısal sistemlerde çıkışlar birleşimsel ve sıralı mantık olmak üzere ikiye ayrılır. Birleşimsel mantıkta çıkış girişlerin boolean işleminden geçmiş halidir. Sıralı mantıkta ise girişlerin yanı sıra daha önceki çıkışın durumunda yeni çıkışın belirlenmesinde bir değişkendir. Bu deneyde daha sonraki denetim uygulamaları için birleşimsel ve sıralı mantık işlemleri için gerekli programlar yazılıp çalıştırılacaktır.

Bileşimsel Mantık

Bir sayısal sistemin X0, X1, X2 gibi üç girişi ve Y0 ve Y1 gibi iki adet çıkışı olsun. Çıkışların boolean eşitlikleri aşağıdaki gibi olsun.

Y0= X0 X1 + X2

Y1= (X0 + X1) (X0 + X2)

Sıralı Mantık

İki girişi ve iki çıkışı olan bir sayısal sistemi örnek olarak seçelim. Girişler X0 ve X1, çıkışlar Y1 ve Y0 olsun. Girişler ile çıkışlar arasındaki bağıntı aşağıdaki gibi olsun.

Y0=(X0 AND Y1)'

Y1=(X1 AND Y0)'

Y1 ve Y0 birbirlerine bağlıdır. Y1 veya Y0'ın değişmesi diğerini de değiştirecektir. Burada bir sıra takip edilmediğinde Y0 ve Y1'in doğruluğunu belirlemek mümkün değildir. Bir başlangıç noktası belirleyip bu noktadan yola çıkmak en doğrusudur. Bu eşitlikleri sayısal elektronikten hatırlayabilirsiniz. Sıralı mantığın en temel elemanı flip-flop'tur. Bu eşitliklerde flip-flopun çıkış eşitlikleridir.

İşlem Sırası

Aşağıdaki programı yazın.

```
;hat tanımlama

;giriş hatları
        X0      equ     p1.0
        X1      equ     p1.1
        X2      equ     p1.2
;çıkış hatları
        Y0      equ     p1.3
        Y1      equ     p1.4
;ara sonuçları saklama biti
        gecici  equ     40h
org 0
basla:
        mov p1, #0FFh           ;X0, X1, X2 hatlarını giriş yap.
;Y0 çıkışını hesaplat
        mov c, X0                       ;X0'ı C bayrağına al.
        anl c, X1                       ;X0 ∧ X1 → C
        orl c, X2                       ;(X0X1) V X2 → C
        mov Y0, c                       ;C'deki sonucu çıkış hattına yaz.
;Y1 çıkışını hesaplat
        mov c, X0                       ;X0'ı C bayrağına al.
        orl c, X1                       ;X0 V X1→ C
        Mov gecici, C           ;ara sonucu sakla.
        mov c, X0                       ;X0'ı C'ye al.
        orl c, X2                       ;X0 V X2 → C.
        anl c, gecici           ;(X0 V X1)(X0 V X2) → C
        mov Y1, c                       ;C'deki sonucu çıkış hattına yaz.
        ajmp basla              ;işlemi sürekli tekrarla
end
```

Programı simulatörde denedikten sonra programlayıcıyı kullanarak işlemciyi programlayın.

Girişlere basmalı DIP-switch ve çıkışlara LED bağlayarak devreye gerilim uygulayın.

aşağıdaki doğruluk devresini devreyi gerkli girişleri uygulayarak çıkışlarının durumlarını gözlemleyerek tamamlayın.

X0	X1	X2	Y0	Y1
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Tablo 1

Aşağıdaki programı yazın

```
;hat tanımlamaları.
;giriş hatları.
        X0      equ     P1.0
        X1      equ     P1.1
;çıkış hatları.
        Y0      equ     P1.3
        Y1      equ     P1.4
Org 0000h
Basla:
        Mov p1, #0FFh           ; giriş hatlarını hazırla.
        Mov c, X0               ;birinci girişi oku.
        Anl c, Y1               ;birinci giriş ile ikinci çıkışı VE'le.
        Cpl c                   ;sonucun değilini al.
        Mov Y0, c               ;sonucu yaz.
        Mov c, X1               ;ikinci girişi oku.
        Anl c, Y0               ;ikinci giriş ve birinci çıkışı VE'le.
        Cpl c                   ;sonucun değilini al.
        Mov Y1, c               ;sonucu hatta yaz.
        Sjmp Basla          ;işlemi sürekli yap.
```

Programı simulatörde çalıştırın doğruluğundan emin olduktan sonra programlayıcıyı kullanarak işlemcinizi programlayın.

X1 ve X0 girişlerine DIP Switch, Y0 ve Y1 çıkışlarına LED bağlayarak programı çalıştırın. X1 ve X0 girişlerine olası değerleri uygulayarak çıkışları gözlemleyin.

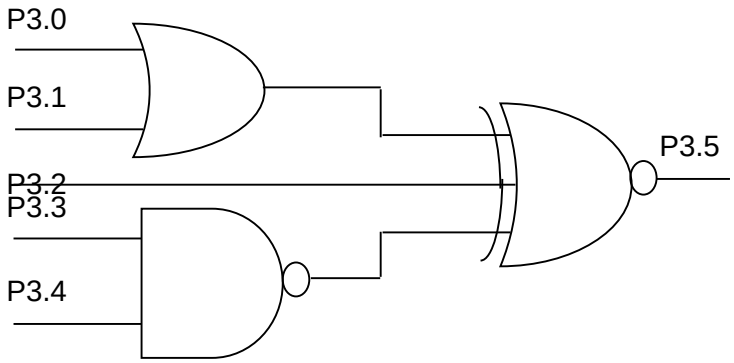
DEĞERLENDİRMESORULARI

1. Bir butonla LED'i hem yakan hem söndüren programı yazın simulatörde denedikten sonra işlemciyi programlayıp devre üzerinde çalıştırın.

X=AC+AD+BC+BD mantık işlemini yapan programı yazın.

X=(A'+B')(C'+D) mantık işlemini yapan programı yazın.

2. Devrenin işlevini yerine getiren programı yazın ve çalıştırın



LED Denetimi

Deney: LED.c

```
/* bu program LED'i 1 saniye yakar 1 saniye söndürür */
#include <reg52.h>

sbit      led=P3^7;
sbit      led1=P3^5;
void milisaniye(int gecikme) /* 1 milisaniyelik zaman geciktirme */
{         int k, 8
          for(k=0; k<gecikme; k++)
  {
          for (z=0; z<500; z++);
  }
}

void main (void)
{
          while(1)
  {
          led=0;                          //LED'i yak
          led1=1;                                 //LED'i söndür

          milisaniye(1000);       // 1 saniya bekle
          led=1;                          //LED'i söndür
          led1=0;                                 //LED'i yak

          milisaniye(1000);       // 1 saniya bekle
  }
}
```

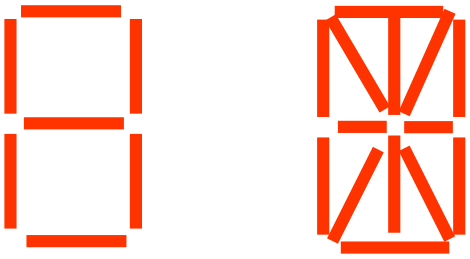
Deney: LED_Buton.c

```
#include <REG52.H>
#include <stdio.h>
sbit buton = P2^0;                    //Butonu'i P2.0'a bağla
sbit led = P1^0;                      //LED'i P1.0'a bağla
void main(void)
{
led=1;
        while(1)
        {       while(buton);
                while(!buton);
                led=!led;
        }

                                                        }
```

7-Elemanlı LED Gösterge Denetimi

Giriş

7-elemanlı gösterge 7 adet LED'in günlük yaşamda kullandığımız karakterleri özellikle de rakamları gösterecek şekilde dizilmesiyle elde edilmiştir. Daha sonra ondalık noktası için sekizinci eleman olarak bir LED daha eklenmiştir. /-elemanlı göstergelerin sınırlı sayıda karekteri görüntüleyebilmesi nedeniyle alfabede buluna bir çok karakter görüntülenemez. bu nedenle 10,13 ve 14 parçalı gösterge türleride üretilmiştir. Şekil-1'de örnek göstergeler gösterilmiştir.

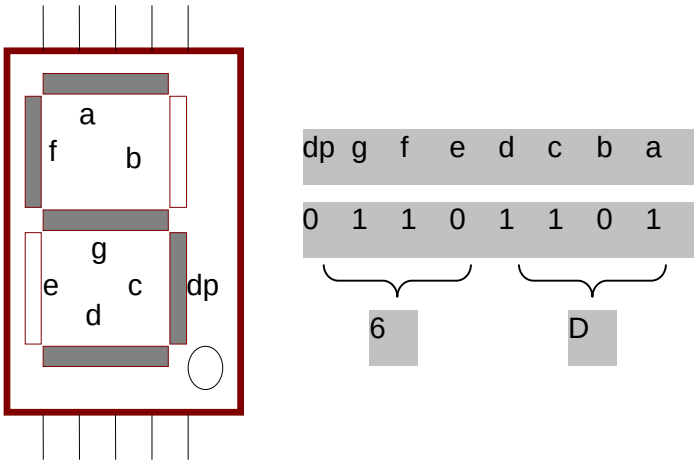


Şekil-1 7 parçalı ve 13 parçalı LED göstergeler.

Dizilen LED'lerin katotları birleştirilip tek uç olarak gösterge paketinin dışına çıkarıldı ise ortak katot gösterge, eğer anotları birleştirip tek bir uç olarak paketin dışına çıkarıldı ise ortak anot gösterge adı verilir. Her iki tür gösterge her devrede bağlantı değişikliği ile kullanılabilir.

İşlem Sırası

1. "5" karakterini nasıl elde edildiği şekil 2'de gösterilmiştir. Tablo 1'de gösterilen karakterlerin elde edilmesi için gerekli kodları belirleyin.
2. Akümülatörün içeriğini 7-parçalı LED gösterge koduna dönüştüren programı yazın. Akümülatör 8 bittir, fakat 7-parçalı LED gösterge sadece bört bitini gösterebilir. Bir baytlık veri kod dönüşümü sonrası iki bayt olacaktır. Düşük değerli dört bitin kodunu R0'da, yüksek değerli dörtlünün kodunu R1'de saklayın.

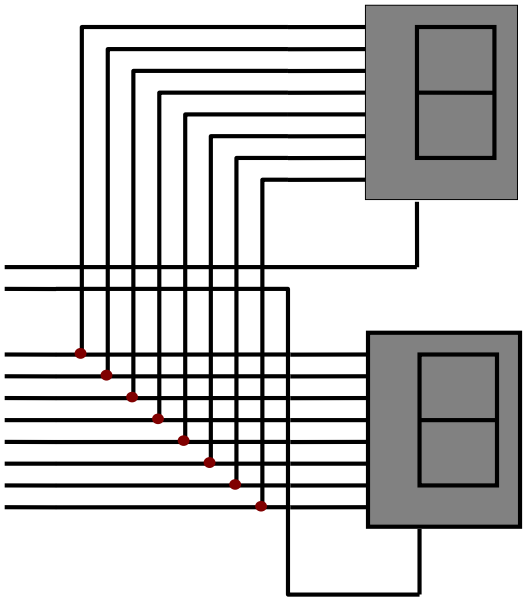


Şekil-2 Ortak katot 7-parçalı göstergede karakterinin kodunun yazılması

3. 0,1,2.....,A,B,C,D,E,F,0,1... şeklinde sayan ve saydığı karakteri P0'daki 7-elemanlı göstergede gösteren programı yazın. Programı derleyip simulatorda doğruluğunu denetleyin. Mikrodenetleyiciyi programlayıp çalıştırın.
4. Şekil 4'teki bağlantıyı yapın. Tek port kullanarak iki gösterge sürülecektir, birden fazla gösterge bağlanmak istendiğinde her gösterge için bir port ayrılması mümkün değildir. Gösterge katotları denetlenerek zaman paylaşımlı göstergeler etkin yapılacaktır. Saniyede 25 tarama yaparsanız zaman paylaşımlı kullanılmasına rağmen görüntü sürekli olacaktır. Bu yötemle 8 adete kadar gösterge bağlanabilir. Kotot uçlarının denetimi bir transistör ile yapılmalıdır. Bu iki göstergede 00..99 arası sayan onlu sayıcının programını yazın.

Onaltılı	dp	G	F	e	D	c	b	a	7-seg
0	0	0	1	1	1	1	1	1	3FH
1									
2									
3									
4									
5	0	1	1	0	1	1	0	1	6DH
6									
7									
8									
9									
A									
b									
C									
d									
E									
F									

5. Tablo 1 Belirtilen karakterlerin 7-parçalı LED gösterge kodlarını
6. Programı derleyin, işlemciyi programlayıp devreyi çalıştırın. Görüntü beklediğiniz gibi değilse, tarama hızınız yük sek veya düşük olabilir. Yüksek olduğunda her iki göstergede 88 görüntüsünü, düşük oldoğuda ise titreşimli bir görüntü gözlersiniz. Düzgün görüntü elde edinceye kadar programı düzeltin.



Şekil-4 iki göstergenin koşut bağlanması.

Deney: 7-Segment LED Gösterge ile Sayıcı Tasarımı

```
/*Bu program 7-segment display ile 0-99 sayıcı çalıştırır*/
#include <reg52.h>              // SFR tanýmlamalarý
#include <stdio.h>
//#include
"C:\Users\engin\Documents\2012 mkr\ileri mikrodenetleyiciler\2013_deney\p89v51rx2.h"
#define SYSCLK    12000000          // SYSCLK frequency in Hz
#define SEG_D P2              // 7-segment display buraya bagli
#define DIPSW P1
typedef unsigned char tByte;  // i_aretsiz karakteri tByte olarak çagir
sbit katot10 = P3^7;                    //onlar basamagi katodu bagli
sbit katot1 = P3^6;                     //birler basamagi katodu bagli
tByte code SEG_Tablosu[20] = {0x3f, 0x06, 0x5B, 0x4F, 0x66,
0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x77, 0x7c, 0x39, 0x5e, 0x79,
0x71, 0xFD, 0x87, 0xFF, 0xEF};
void msec(unsigned int gecik)                   // 1 msaniye geçiktirir.
{       unsigned int i, j;
  for(j=0; j<gecik; j++)
  {     for (i=0; i<50; i++);     }
}
void main (void)                        // ana program
{       unsigned char sayici;           //say1c1y1 s1f1rla
        while (1)
        {       for(sayici=0;sayici < ((DIPSW & 0xf0)>>4);sayici++)
{       SEG_D = SEG_Tablosu[sayici];
        msec (1000);
}
        }

}
```

Dot Matris LED Gösterge Denetimi

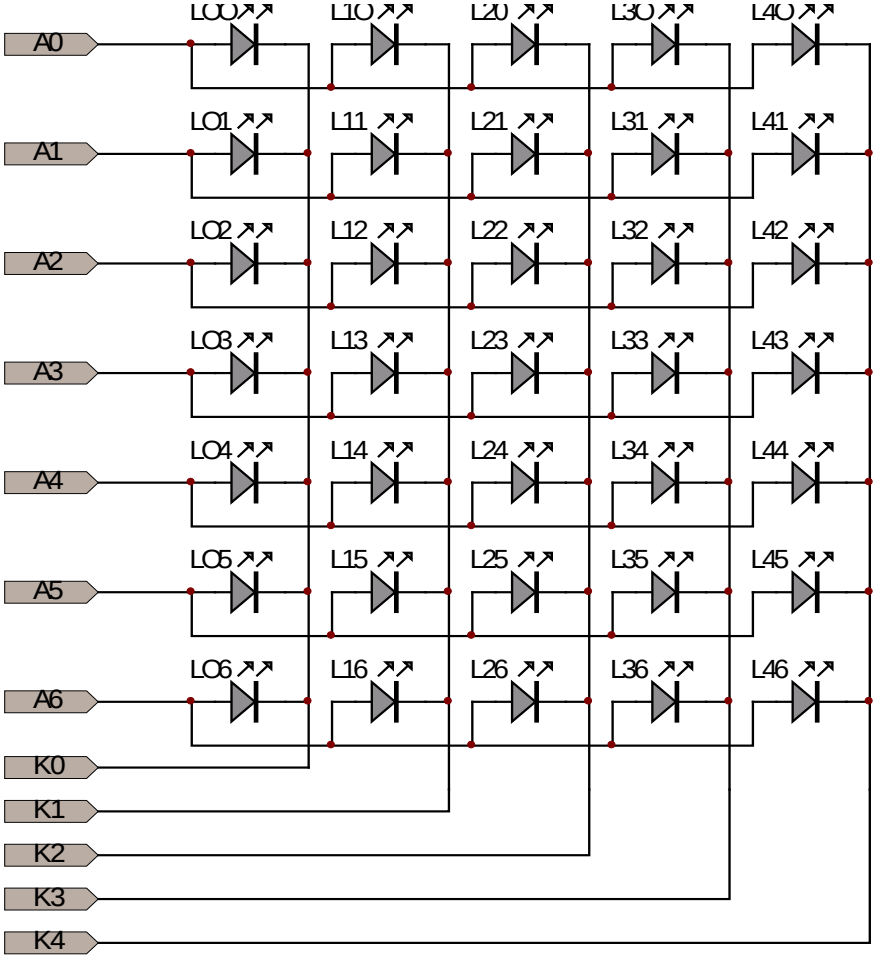
Giriş

7-parçalı LED göstergede tüm karakterlerin görüntülenememesi üreticileri arayışa itmiş ve nokta şekilli LED'lerin matris şeklinde dizilmesi ile elde edilen göstergeler üretilmiştir. Çalışma şekilleri 7-parçalı LED gösterge ile aynı olup değişik boyutlarda farklı büyüklükteki LED'lerle üretilen çeşitleri vardır. Genel olarak satırda dizili LED'lerin anotları sütunda dizili LED'lerin katotları birleştirilerek tek bir uç olarak paketlin dışına çıkarılır. Şekil 1'te 7X5 boyutlu matris LED göstergenin iç bağlantı şeması gösterilmiştir.

Karakter oluşturmak için tüm katotlar sıra ile DÜŞÜK seviyeye çekilirken, anotlara sıradaki sütunda yanması gereken LED'ler için YÜKSEK seviye sönük kalması gerekenler için DÜŞÜK seviye uygulanır. LED'lerin eşit parlaklıkta olması için katotlarda akım kaynağı kullanılmalıdır. Tarama işlemi insan gözünün algılamadığı frekans aralığında yapılmalıdır. Aslında mikroişlemci bu aralığı sağlayacak kadar hızlıdır. Fakat çok hızlı tarama yapıldığında LED'in ışık yayması ve sönmesi için belirli bir süre geçmesi gerekiyor. Bu süre dikkate alınmadığında göstergede sürekli tüm LED'ler yanık olarak görünecektir. Bu süre yaklaşık 100 µsaniye ile 1 msaniye arasındadır. LED en az 100 µsaniye yanık kalmalıdır. Gözü kandırmak için her bir karakter saniyede en az 25 defa taranmalıdır.

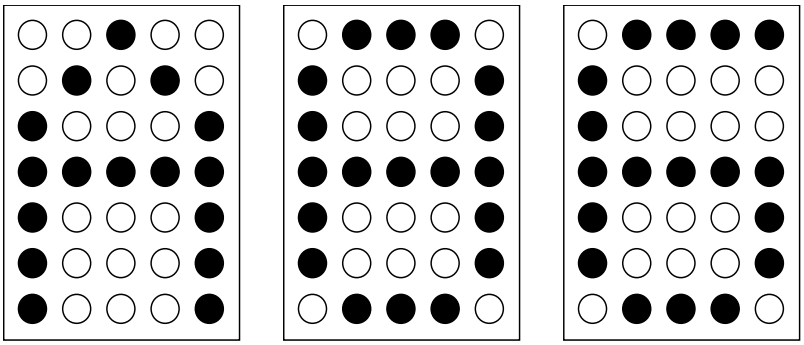
$$Bir\ \text{tarama için gerekli süre} = \frac{1\,\text{saniye}}{25} = 40\ \text{msaniye'dir}.$$

Her karakter 5 sütundan oluşur 40 mili saniyede karakter tamamlanacağına göre her sütun 8 mili saniye süre ile yanık kalacaktır



Şekil 1 7X5 boyutlu matris LED göstergenin iç bağlantısı.

Şekil 2'de "A", "B" ve "8" karakterlerinin oluşturulması için yanması gereken LED2le gösterilmiştir. Örnek olarak "A" karakterinin kodlarını oluşturalım. Sütunlar sıra ile taranacağı için sütün sayısı kadar kod oluşturmamız gerekir. Kodları kolay yazabilmek için tablo 1'e benzer bir tablo oluşturalım.



Şekil 2 "A", "B" ve "8" karakterlerinin 5X7 boyutlarındaki nokta matrisli LED göstergedeki görüntüleri.

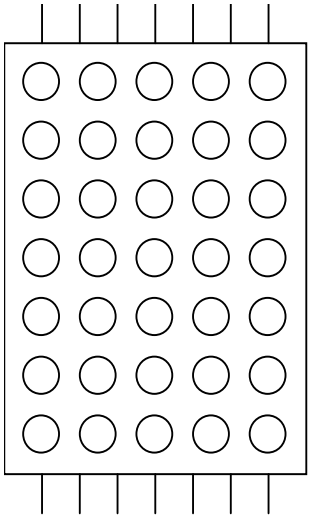
	1. ADIM	2. ADIM	3. ADIM	4. ADIM	5.ADIM
A0	0	0	1	0	0
A1	0	1	0	1	0
A2	1	0	0	0	1
A3	1	1	1	1	1
A4	1	0	0	0	1
A5	1	0	0	0	1
A6	1	0	0	0	1
K0	0	1	1	1	1
K1	1	0	1	1	1
K2	1	1	0	1	1
K3	1	1	1	0	1
K4	1	1	1	1	0

Tablo 1 “A” karakterinin elde edilmesi için gerekli kodlar.

Birinci adımda K0 “0” yapılacaktır ve 0 nolu stunda yanması gereken LED’in anoduna “1” sönmesi gerekene “0” uygulanacaktır. 2 adımda K1 ucuna “0” uygulanarak etkin yapılacak ve 1 nolu sütundaki LED’lerden yanması gerekenlerin anoduna 1 sönmesi gerekenlere 0 uygulanacaktır diğer stunlarda aynı mantıkla elde edilebilir. Elde edilen kodlar tablo 1’de gösterilmiştir.

İşlem Sırası

1. 5X7 boyutundaki matris dizili LED göstergenin bacak bağlantılarını ölçü aleti veya güç kaynağının bir ucuna bağlayacağınız 1 k’lik dirençle belirleyin. Şekil 3 üzerine yazın.

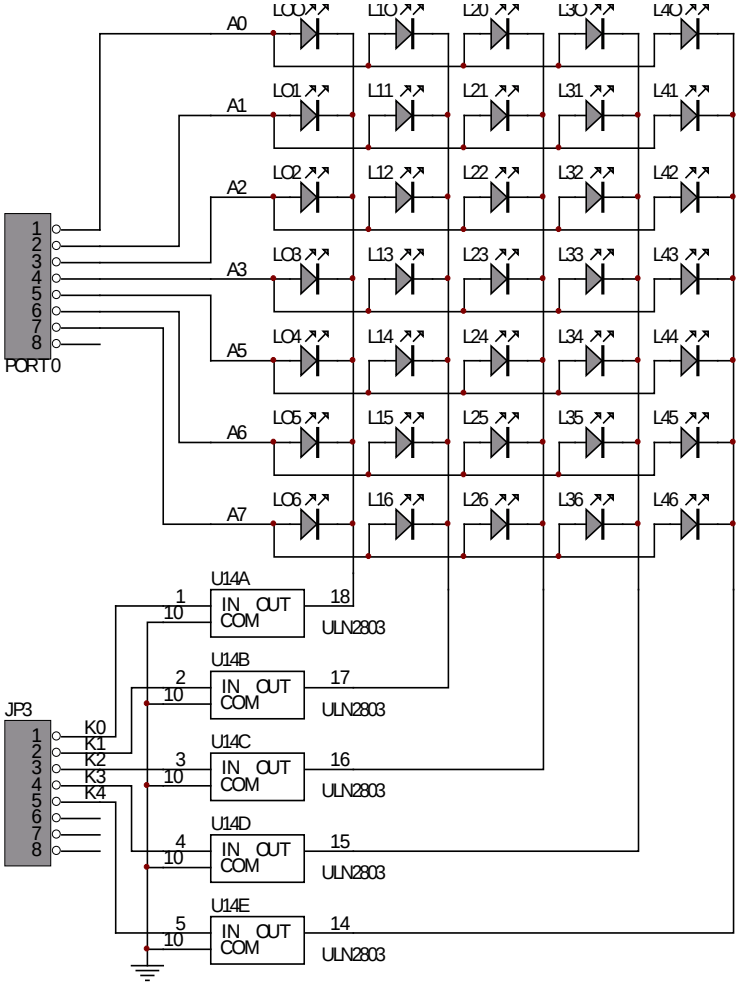


Şekil-3

2. Şekil-4’teki devreyi kurun yukarıda elde ettiğiniz “A” karakterini görüntüleyen programı yazın.
3. Yazdığınız programı derleyin, simulatorda doğruluğunu denetleyin.
4. Mikrodenetleyiciyi programlayıp devreyi çalıştırın.
5. Şekil 5’teki bağlantıyı yapın. Bu bağlantıda katotlar 4-16 kodçözücü ile sürülmüştür.
6. Bu bağlantıda “A” karakterini yürüyen yazı şeklinde gösteren programı yazın.
7. Programı derleyin, işlemciyi programlayıp devreyi çalıştırın.
8. Aynı bağlantıda deney gurubunda bulunan kişilerin ad ve soyadlarını kayan yazı şeklinde

yazan programı yazın.

9. programı derleyin, denetleyiciyi programlayıp çalıştırın.



Şekil 4 Nokta matris dizili LED göstergenin sürülmesi.

Deney: Dot Matris LED Gösterge Kullanımı

```
/*Bu program bir DOT matris LED Göstergede 0-9 ve a-z sayıcı yapar */
#include <reg52.h>              // SFR tanýmlamalarý
#include <stdio.h>
//#include
"C:\Users\engin\Documents\2012mkr\ileri mikrodenetleyiciler\2013_deney\p89v51rx2.h"
#define SYSCLK    12000000          // SYSCLK frequency in Hz
 // dot Matrix display buraya bagli
#define katot P1
sbit DS_1 = P2^0;                   //birler basamagi katodu bagli
sbit SH_CP = P2^1;                  //birler basamagi katodu bagli
sbit ST_CP = P2^2;                  //birler basamagi katodu bagli
unsigned char code Dot_Tablosu[95][5]=
{
   { 0x00, 0x00, 0x00, 0x00, 0x00 }, // " "
   { 0x00, 0x00, 0x4f, 0x00, 0x00 }, // !
```

```
{ 0x00, 0x07, 0x00, 0x07, 0x00 }, // "
{ 0x14, 0x7f, 0x14, 0x7f, 0x14 }, // #
{ 0x24, 0x2a, 0x7f, 0x2a, 0x12 }, // $
{ 0x23, 0x13, 0x08, 0x64, 0x62 }, // %
{ 0x36, 0x49, 0x55, 0x22, 0x50 }, // &
{ 0x00, 0x05, 0x03, 0x00, 0x00 }, // '
{ 0x00, 0x1c, 0x22, 0x41, 0x00 }, // (
{ 0x00, 0x41, 0x22, 0x1c, 0x00 }, // )
{ 0x14, 0x08, 0x3e, 0x08, 0x14 }, // *
{ 0x08, 0x08, 0x3e, 0x08, 0x08 }, // +
{ 0x00, 0x50, 0x30, 0x00, 0x00 }, // ,
{ 0x08, 0x08, 0x08, 0x08, 0x08 }, // -
{ 0x00, 0x60, 0x60, 0x00, 0x00 }, // .
{ 0x20, 0x10, 0x08, 0x04, 0x02 }, // /
{ 0x3e, 0x51, 0x49, 0x45, 0x3e }, // 0
{ 0x00, 0x42, 0x7f, 0x40, 0x00 }, // 1
{ 0x42, 0x61, 0x51, 0x49, 0x46 }, // 2
{ 0x21, 0x41, 0x45, 0x4b, 0x31 }, // 3
{ 0x18, 0x14, 0x12, 0x7f, 0x10 }, // 4
{ 0x27, 0x45, 0x45, 0x45, 0x39 }, // 5
{ 0x3c, 0x4a, 0x49, 0x49, 0x30 }, // 6
{ 0x01, 0x71, 0x09, 0x05, 0x03 }, // 7
{ 0x36, 0x49, 0x49, 0x49, 0x36 }, // 8
{ 0x06, 0x49, 0x49, 0x29, 0x1e }, // 9
{ 0x00, 0x36, 0x36, 0x00, 0x00 }, // :
{ 0x00, 0x56, 0x36, 0x00, 0x00 }, // ;
{ 0x08, 0x14, 0x22, 0x41, 0x00 }, // <
{ 0x14, 0x14, 0x14, 0x14, 0x14 }, // =
{ 0x00, 0x41, 0x22, 0x14, 0x08 }, // >
{ 0x02, 0x01, 0x51, 0x09, 0x06 }, // ?
{ 0x32, 0x49, 0x79, 0x41, 0x3e }, // @
{ 0x7e, 0x11, 0x11, 0x11, 0x7e }, // A
{ 0x7f, 0x49, 0x49, 0x49, 0x36 }, // B
{ 0x3e, 0x41, 0x41, 0x41, 0x22 }, // C
{ 0x7f, 0x41, 0x41, 0x22, 0x1c }, // D
{ 0x7f, 0x49, 0x49, 0x49, 0x41 }, // E
{ 0x7f, 0x09, 0x09, 0x09, 0x01 }, // F
{ 0x3e, 0x41, 0x49, 0x49, 0x7a }, // G
{ 0x7f, 0x08, 0x08, 0x08, 0x7f }, // H
{ 0x00, 0x41, 0x7f, 0x41, 0x00 }, // I
{ 0x20, 0x40, 0x41, 0x3f, 0x01 }, // J
{ 0x7f, 0x08, 0x14, 0x22, 0x41 }, // K
{ 0x7f, 0x40, 0x40, 0x40, 0x40 }, // L
{ 0x7f, 0x02, 0x0c, 0x02, 0x7f }, // M
{ 0x7f, 0x04, 0x08, 0x10, 0x7f }, // N
{ 0x3e, 0x41, 0x41, 0x41, 0x3e }, // O
{ 0x7f, 0x09, 0x09, 0x09, 0x06 }, // P
{ 0x3e, 0x41, 0x51, 0x21, 0x5e }, // Q
{ 0x7f, 0x09, 0x19, 0x29, 0x46 }, // R
{ 0x46, 0x49, 0x49, 0x49, 0x31 }, // S
{ 0x01, 0x01, 0x7f, 0x01, 0x01 }, // T
{ 0x3f, 0x40, 0x40, 0x40, 0x3f }, // U
{ 0x1f, 0x20, 0x40, 0x20, 0x1f }, // V
{ 0x3f, 0x40, 0x38, 0x40, 0x3f }, // W
{ 0x63, 0x14, 0x08, 0x14, 0x63 }, // X
```

```
    { 0x07, 0x08, 0x70, 0x08, 0x07 }, // Y
    { 0x61, 0x51, 0x49, 0x45, 0x43 }, // Z
    { 0x00, 0x7f, 0x41, 0x41, 0x00 }, // [
    { 0x02, 0x04, 0x08, 0x10, 0x20 }, // "\"
    { 0x00, 0x41, 0x41, 0x7f, 0x00 }, // ]
    { 0x04, 0x02, 0x01, 0x02, 0x04 }, // ^
    { 0x40, 0x40, 0x40, 0x40, 0x40 }, // _
    { 0x00, 0x01, 0x02, 0x04, 0x00 }, // `
    { 0x20, 0x54, 0x54, 0x54, 0x78 }, // a
    { 0x7f, 0x48, 0x44, 0x44, 0x38 }, // b
    { 0x38, 0x44, 0x44, 0x44, 0x20 }, // c
    { 0x38, 0x44, 0x44, 0x48, 0x7f }, // d
    { 0x38, 0x54, 0x54, 0x54, 0x18 }, // e
    { 0x08, 0x7e, 0x09, 0x01, 0x02 }, // f
    { 0x0c, 0x52, 0x52, 0x52, 0x3e }, // g
    { 0x7f, 0x08, 0x04, 0x04, 0x78 }, // h
    { 0x00, 0x44, 0x7d, 0x40, 0x00 }, // i
    { 0x20, 0x40, 0x44, 0x3d, 0x00 }, // j
    { 0x7f, 0x10, 0x28, 0x44, 0x00 }, // k
    { 0x00, 0x41, 0x7f, 0x40, 0x00 }, // l
    { 0x7c, 0x04, 0x18, 0x04, 0x78 }, // m
    { 0x7c, 0x08, 0x04, 0x04, 0x78 }, // n
    { 0x38, 0x44, 0x44, 0x44, 0x38 }, // o
    { 0x7c, 0x14, 0x14, 0x14, 0x08 }, // p
    { 0x08, 0x14, 0x14, 0x18, 0x7c }, // q
    { 0x7c, 0x08, 0x04, 0x04, 0x08 }, // r
    { 0x48, 0x54, 0x54, 0x54, 0x20 }, // s
    { 0x04, 0x3f, 0x44, 0x40, 0x20 }, // t
    { 0x3c, 0x40, 0x40, 0x20, 0x7c }, // u
    { 0x1c, 0x20, 0x40, 0x20, 0x1c }, // v
    { 0x3c, 0x40, 0x30, 0x40, 0x3c }, // w
    { 0x44, 0x28, 0x10, 0x28, 0x44 }, // x
    { 0x0c, 0x50, 0x50, 0x50, 0x3c }, // y
    { 0x44, 0x64, 0x54, 0x4c, 0x44 }, // z
    { 0x00, 0x08, 0x36, 0x41, 0x00 }, // {
    { 0x00, 0x00, 0x7f, 0x00, 0x00 }, // |
    { 0x00, 0x41, 0x36, 0x08, 0x00 }, // }
    { 0x02, 0x01, 0x02, 0x04, 0x02 }, // ~
};
void msec(unsigned int gecik)                   // 1 msaniye geçiktirir.
{       unsigned int i, j;
  for(j=0; j<gecik; j++)
  {
        for (i=0; i<50; i++);
   }
}
void main (void)                        // ana program
{ unsigned char sayici,i,j,k,z,dat;             //say1c1y1 s1f1rla
bit data_bit;
SH_CP=0;
ST_CP=0;
        while (1)
{
        for(sayici=0;sayici <96;sayici++)
{
```

```
        for(i=0;i<46;i++)
{
for(j=0;j<5;j++)
{
         dat= Dot_Tablosu[sayici][j];
        katot=1;
                for(k=0;k<8;k++)
{
                data_bit = dat & 0x80;
                DS_1 = data_bit;

                SH_CP=1;
                z++;
                SH_CP=0;
                dat = dat<<1;
}
        ST_CP=1;
        z++;
        ST_CP=0;
        katot=j+1;
        msec (3);
}
}
}
}
}
```

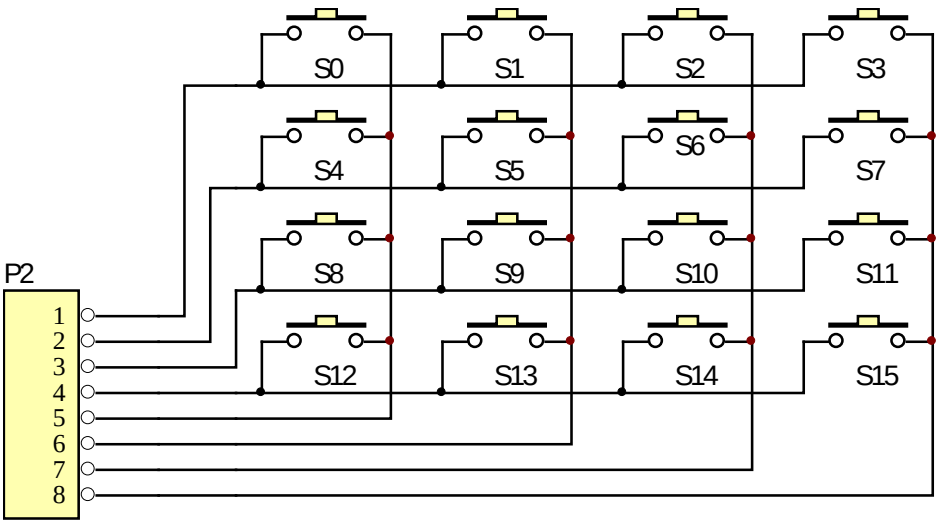
Keybord Bağlantısı

Giriş

Bu deneyde matris olarak dizilmiş tuşlardan oluşan keybordun yazılımı yazılacaktır. Öncelikle matris dizili tuşların çalışmasını açıklayalım. Oluşturulacak matris 4X4 boyutunda olacaktır. Matris 4 adet sütun ve 4 adet satır hattından oluşacaktır. 16 tuşdan oluşması 16'lı sayıların girilebilmesini sağlayacaktır. Her tuşun bir ucu satır hattına diğer ucu sütun hattına bağlanacaktır. Satır ve sütun hatları ise mikrodetleyicinin portuna bağlıdır. Portun yarısı çıkış yapılır, diğer yarısı ise giriş yapılır. Çıkış yapılan hatlara sütun hatları bağlanır ve düşük seviye ile sürülür. Giriş yapılan hatlara satır hatları bağlanır. Her hangi bir tuşa basılmadığında tüm giriş hatları yüksek seviyede okunacaktır. Herhangi bir tuşa basıldığında ise o satır hattından bir düşük seviye okunur. Her şey bu noktadan sonra başlar her satırda 4 tuş var, bunlardan hangisine basıldığının bulunması gereklidir. Basılan tuşun doğru belirlenebilmesi için sütunların hepsinin aynı anda düşük seviye yapılması yerine sıra ile sütun hatlarının düşük seviye yapılması ve satır hatlarının her sütun düşük seviye yapıldığında tekrar tarama yöntemi ile okunması gereklidir.

Basılı tuşun belirlenmesi için program sütun 0'dan başlayarak "0" yazar ve satır 0'dan başlayarak sıra ile satır 1,2 ve 3' ü okur. Düşük seviye bulduğu satırda durur, basılan tuş "0" yazılan sütun ile okunan satırın kesiştiği noktada bulunandır.

Basılan tuş belirlendiğinde tuşun değerini belirleyen alt program çağrılır ve gerekli işlemler yapılır. Yukarıda bahsedilen yöntemle tarama yapılırsa işlemci sürekli tuş denetleme işleminden başka programları işletemez. İşlemcinin sürekli meşgul olmasının önüne geçmek için kesme girişi kullanılarak tuşa basılıp basılmadığı denetlenebilir. Bunun dışında bu amaçla üretilmiş tümdevreler vardır.



Şekil – 1 4X4 Matris dizili keyboard bağlantısı

İşlem Sırası

1. Derste yazılan programı editöre yazıp derleyin, simulatorda doğruluğunu denetleyin.
2. Deney kartınızın keybord bağlantısı şekil–1'de gösterilmiştir. Önceki deneylerde kullandığınız göstergelerden herhangi birini kullanarak basılan tuşun değerini o göstergede gösterin.

3. İşlemciyi programlayıp çalıştırın tüm tuşlara ayrı ayrı basarak sonuçlarını gözleyin.
4. Aynı anda iki tuşa basmaya çalışın, sonuç ne oldu açıklayın.

LCD Denetimi

Karakter LCD Modülünün Kullanımı

LED göstergelerin fazla akım çekmesi ve kullanım zorluğu, son yıllarda LCD göstergelerin kullanımını daha popüler hale getirmiştir. İstenilen karakterin daha iyi çözünürlükte elde edilmesi ve yeni üretilen LCD modüllerinin devreye bağlantısı ve programlanmasının kolay hale gelmesi diğer gösterge türlerine göre daha fazla kullanılmasını sağlamıştır. Özellikle karanlık ortamlarda kullanılamaması ilk zamanlarda bir sakınca olarak ortaya çıkmıştı, daha sonra arka plan aydınlatma ile bu sakınca ortadan kaldırılmıştır. Son yıllarda renkli LCD'lerde yaygın bir şekilde kullanılmaya başlamıştır. Yaygın kullanıma rağmen hala fiyatları yüksektir.

Bu deneyde kullanacağımız LCD aslında sadece bir göstergeden ibaret değildir. Akılı bir çevre birimidir, birden fazla karakter içermesi ve bunların belirli hızlarda taranma zorunluluğu üretici firmaları LCD'leri içerisinde osilatörü, sistem mikrodenetleyicisi, kod üreteci gibi birimleri içeren modül olarak üretmelerine neden olmuştur. LCD modüller en son yazılan bilgileri yenisi yazılana kadar göstergede görüntüledikleri için ana işlemciyi fazla meşgul etmezler. LCD'ler 1 satırdan 4 satıra kadar, 16 karakterden 80 karaktere kadar ve 5X7, 5X10 nokta font gibi değişik ölçülerde üretilip satılmaktadır. Bazıları ise tüm ekran tek bir karakter gibi yapılandırılmıştır, bu türlerine grafik ekran adı verilmektedir. LCD modül ile iletişim TTL standardında 4 veya 8 veri hattı ile yapılır. 4 bit iletişim G/Ç hatlarının başka işler için kullanımını kolaylaştırırken iletişim süresini iki kat uzatmaktadır. LCD modüller veri hatlarının yanı sıra 3 ila 5 adet arası denetim hattına gereksinim duyarlar. Besleme mantık elemanları için 5 volt likit kristal sürücüler için ise farklı bir kaynağa gereksinim duyarlar. İkinci kaynak genellikle birinciden ayarlı potansiyometre yardımıyla elde edilir.

LCD Modül Bağlantı Uçları

LCD modüller üzerinde kullanılan denetleyiciler Hitachi firması tarafından üretilen HD44780U mikrodenetleyicisi standart oluşturmuştur. Bu standarttaki LCD modül bağlantı uçları tablo-8.1'de gösterilmiştir. Bağlantı uçlarını besleme, denetim ve veri olmak üzere üç grupta inceleyebiliriz.

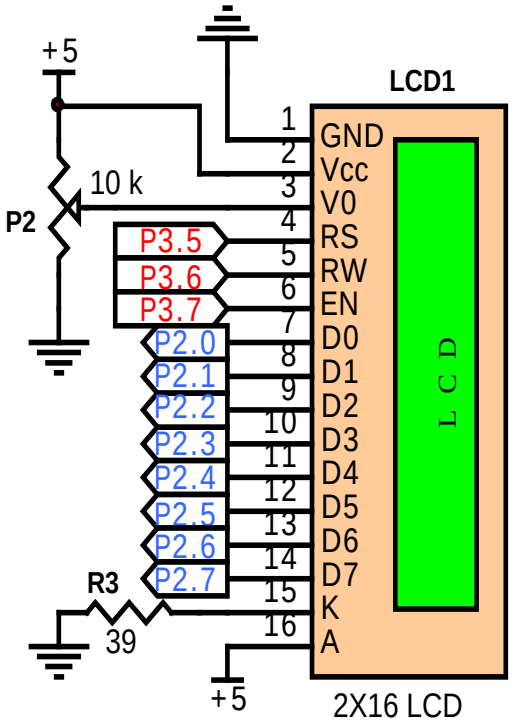
Besleme Uçları

HD44780U standardında besleme ile ilgili üç uç yer almaktadır. Bunlar Vcc, Vee, Vss'dir. Vss ve Vcc standart TTL gerilimi 0 ve 5 Volttur. Vee ise likit kristal sürücü gerilimidir, değeri de Vcc'den daha küçük olur. Normal sıcaklık tipi ve güçlendirilmiş sıcaklık tipi olmak üzere iki tür LCD vardır. Normal sıcaklık türlerinde TN bileşimli sıvı, güçlendirilmiş sıcaklık türlerinde NTN bileşimli sıvı kullanılır. TN bileşimli sıvı kullanılan modüllerde görüş açısı 40 derece ile sınırlıdır. NTN bileşimli sıvılarda görüş açısı 70 derecedir. Normal sıcaklık modüllerde Vee gerilimi pozitiftir. Güçlendirilmiş sıcaklık modüllerde ise Vee gerilimi GND'ye göre –1 veya –2 Volt olduğunda daha iyi görüntü vermektedir. Güçlendirilmiş sıcaklık LCD'ler için Vee geriliminin değeri ortam sıcaklığına bağlıdır. Normal sıcaklık LCD'lerin negatif gerilime gereksinim duyulmadığından V0 gerilimin değeri 0.6V<Vee<2.5V aralığında olmalıdır. Vcc ile GND arasına bağlanacak bir trimpotun orta ucunda V0 gerilimi elde edilebilir. Yine koyuluk ayarı bu trimpot ile çalışma ortamında yapılmalıdır. Şekil-3'de devresi gösterilmiştir.

Denetim Bağlantıları

LCD denetleyicisinin yavaş olmasında dolayı veri ve denetim hatları bellek haritalı mikroişlemcilerde veri yoluna ve RW hattına ve kod çözücüye bağlamak mümkün değildir. Eğer

böyle bir bağlantı yapılıyor ise özellikle RW, EN ve RS işaretleri yavaşlatılarak LCD'ye uygulanmalıdır. 8051 bağlantısı portlar kullanılarak yapılacağından bu hatlar için gerekli gecikme yazılım ile yapılacaktır.



Şekil-1 LCD gösterge bağlantısı.

Gösterge

HD44780U mikrodenetleyicisi göstergeyi denetlerken göstergede görüntüleyeceği karakterlerin ASCII kodlarını Gösterge veri RAM'i olarak adlandırılan bellekte saklar 7 bit ile adreslenebilen bu RAM belleğe DDRAM adı verilir. İki satır LCD göstergelerde DDRAM iki satıra bölünmüştür, birinci satırda 00-27H adresleri, ikinci satırda ise 40H-67H adresleri yer alır.

Gösterge Durumu	1	2	3	4	5	.....	39	40
DDRAM adresi (H)	00	01	02	03	04	.....	26	27
	40	41	42	43	44	.....	66	67

şekil-2 İki satır LCD'lerde DDRAM'in bellek haritası.

Karakter Kümesi

LCD önceden belirlenmiş veya kullanıcı tarafından belirlenmiş karakterleri görüntüleyebilir. LCD denetleyicisinde bir ROM olan karakter üreteci bulunur, ve bu bellekte 192 karakter kayıtlıdır. Her karakterin tablo-2'de gösterildiği gibi bir kodu vardır ve bu karakterlere kodları kullanılarak ulaşılabilir. 96 adet ASCII karakter kendi koduyla kullanılabilir. 64 adet Japon karakteri ve 32 adet özel karakter ve küçük harfler yer almaktadır. LCD denetleyicisi buna ek olarak kullanıcıların karakter tanımlayabilmeleri için 8 karakter saklayabilme kapasiteli karakter

üretme RAM belleğe sahiptir. Bu belleğe kısaca CGRAM adı verilir. Kullanıcı tarafından belirlenen karakterler önce CGRAM'e kaydedildikten sonra buradan kullanılabilir.

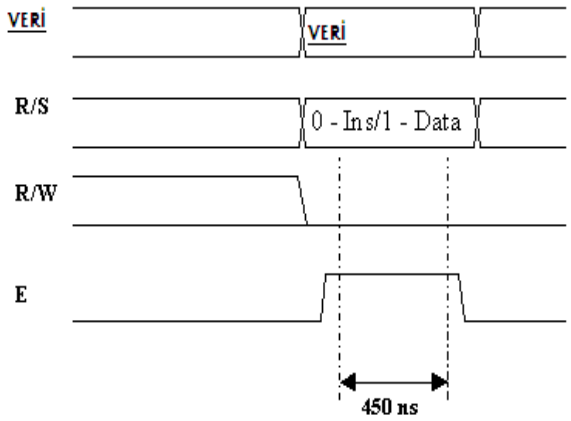
UÇ No	Simge	Seviye	İşlev	Açıklama
1	V_{SS}	0	GND	Eksi besleme kaynağı.
2	V_{CC}	+5 V	TTL Kaynak	Artı besleme kaynağı.
3	V_{ee}	Değişken	LCD Kaynak	Görüntü koyuluk ayar beslemesi.
4	RS	Y/D	Yazaç Seçme	Yazaç seçme girişi.
5	R/W	Y/D	Oku / Yaz	Oku-Yaz işlem seçme girişi
6	E	VURU	İzinleme	İşlem başlatma girişi.
7	DB0	Y/D	Veri bit 0	Veri giriş-çıkış hattı 0
8	DB1	Y/D	Veri bit 1	Veri giriş-çıkış hattı 1
9	DB2	Y/D	Veri bit 2	Veri giriş-çıkış hattı 2
10	DB3	Y/D	Veri bit 3	Veri giriş-çıkış hattı 3
11	DB4	Y/D	Veri bit 4	Veri giriş-çıkış hattı 4
12	DB5	Y/D	Veri bit 5	Veri giriş-çıkış hattı 5
13	DB6	Y/D	Veri bit 6	Veri giriş-çıkış hattı 6
14	DB7	Y/D	Veri bit 7	Veri giriş-çıkış hattı 7
15	K	0	Katot	Arka aydınlatma eksi beslemesi.
16	A	+5 V	Anot	Arka aydınlatma artı beslemesi.

Tablo-9.1 LCD modülün bağlantı uçları ve görevleri.

Denetim Hatları

EN, RW, RS olarak adlandırılan üç adet denetim hattı vardır. Enable (EN) hattı LCD'ye bilgi göndermek istediğimizi belirtmek için kullanılır. LCD ile iletişim kurmak isteyen program öncelikle bu hattın seviyesini yükseğe çekmelidir. Diğer iki denetim hattının seviyesi yapılacak işleme göre seçildikten sonra veri hatlarına veri yazılır ve EN hattı düşük seviye yapılır bu negatif geçiş LCD denetleyicisini işleme başlatır. Yazaç seçme (RS) hattı gönderilen verinin komut mu yoksa veri mi olduğunu belirtmek için kullanılır. Bu hat DÜŞÜK seviye olduğunda veri hatlarındaki bilgi komut olarak alınır ve işletilir. Eğer bu hat YÜKSEK seviye ise veri hatlarındaki bilgi karakter kodudur, bu kod alınır ve göstergede görüntülenir.

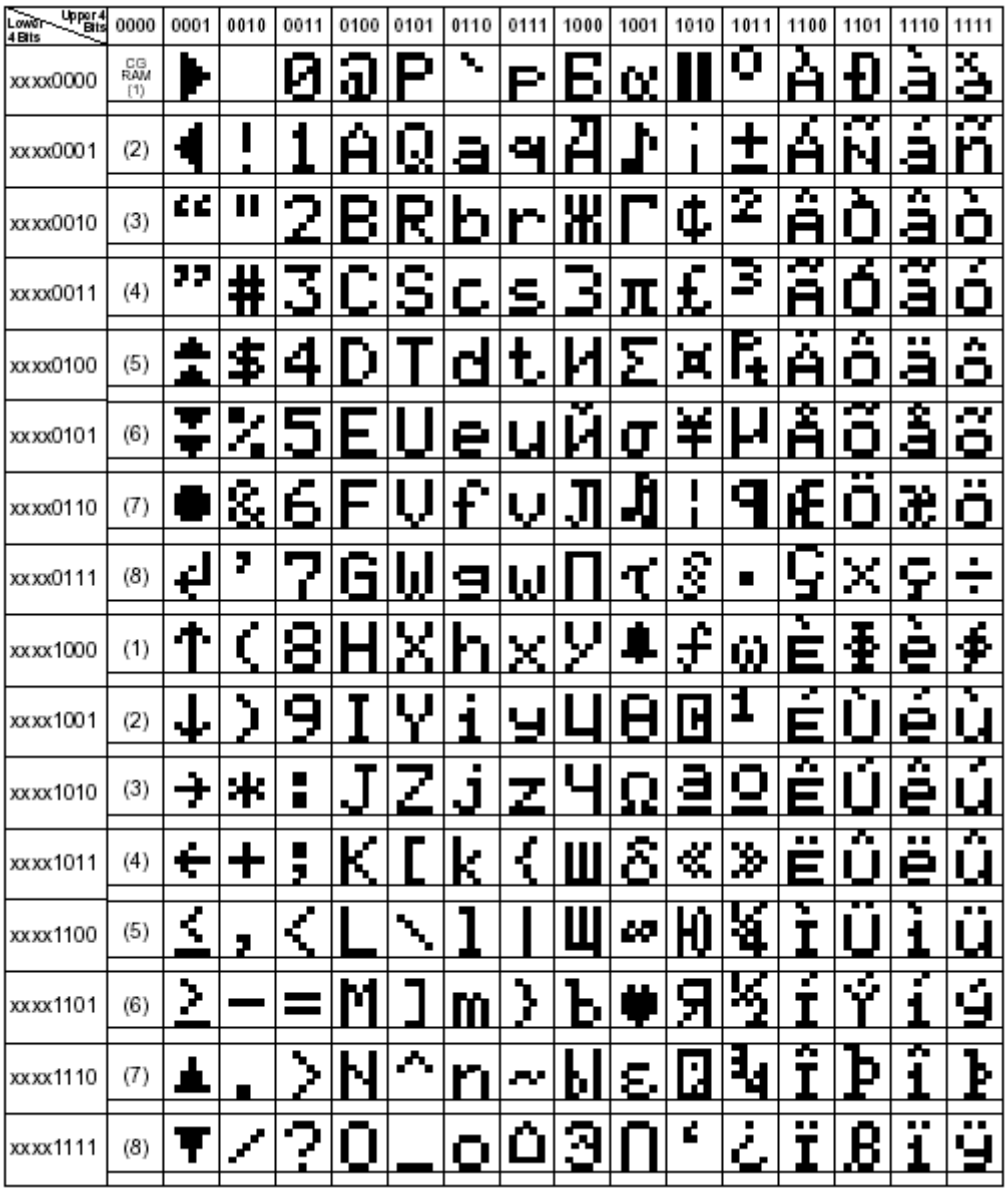
Oku-Yaz (RW) hattı LCD'den yapılacak işlemin okuma yada yazma olduğunu belirler. Bu hat DÜŞÜK seviye iken LCD'ye yazma yapılır. YÜKSEK seviye iken LCD yazaçları okunur. Okuma komutu birkaç tanedir, geri kalanı yazma komutudur. Genellikle uygulamalarda bu hat doğrudan GND'ye bağlanır.



Şekil-2 Denetim işaretlerinin zamanlama diyagramı

Lower 4 Bits \ Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		−	=	M	]	m	}			ュ	ス	ヘ	ン	ŧ	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	°	ö	■

ROM kodu A00 karakterleri



ROM kodu A02 karakterleri

Başlangıç Reseti

LCD modül gerilim uygulandığında kendisini kullanıma hazırlar. Kaynak LCD uyuşmazlığında bu işlem gerçekleşmeyebilir, bu durumda arka arkaya 3 adet 30h komutu ile ayarlama işlemi yazılım ile yapılabilir.

Function set:

Veri yolunun genişliği, karakter fontu ve göstergede kullanılacak satır sayısı belirlenir. Komutun açılışı şöyledir;

0 0 1 DL N F x x

bitlerin anlamları;

DL=0 ise 4 adet veri hattı, DL=1 ise 8 bit veri hattı kullanılacak.

N=0 1 satır, N=1 2 satır kullanılacak.

F=0 5X7 noktadan, F=1 5X10 noktadan oluşacak karakter fontu kullanılacak.

X; bu bitlerin değeri önemli değildir.

İmleci Gizle/Göster/Kırpıştır (Hide/Show/Blink Curser)

İmleç görüntüden gizlenebilir, hala bir sonraki yazılacak karakter konumunu gösterir. İmleç başlangıçta alt çizgi olarak göstergede görünür, istenirse yanıp sönen karakter olarak ta görüntülenebilir. Yanıp sönen imleçte istenirse görünmez yapılabilir. Komut bunların dışında göstergeyi açıp kapatan bite de sahiptir. Komutun açılışı aşağıdaki gibidir.

0 0 0 0 1 D C B

D=1 gösterge açık, D=0 gösterge kapalı.

C=1 imleç açık, C=0 imleç kapalı.

B=1 bulunduğu konumdaki karakteri yakıp söndürür.

B=0 imleç sabit alt çizgi olarak görüntülenir.

Entry Mode:

Okuma ve yazma işlemi sonrası imlecin ve göstergenin durumunu beliler. Genel kullanımı her yazma işlemi sonrası imlecin konumu bir arttırılırken gösterge sabit bırakılır. Bu kullanımda bir sonraki karakter bir sağ konuma yazılır. Komutun bitlerinin anlamı şöyledir;

0 0 0 0 0 1 I/D S

I/D=0 imlecin konumunu bir azalt (bir sola kaydır).

I/D=1 imlecin konumunu bir arttır (bir sağa kaydır).

S=0 gösterge sabit.

S=1 göstergeyi I/D bitine göre sağa veya sola doğru kaydır. Eğer I/D biti 1 ise gösterge sola doğru kayar, 0 ise sağa doğru kayar.

İmlec bir sonraki adımda karakterin yazılacağı konumu veya bir sonraki adımda okunacak karakterin konumunu gösterir. İmlecin işleyişi yukarıda entry mode ile belirlenmişti.

Göstergeyi Temizle (Clear Display):

Göstergeyi temizler. Gösterge temizlendiğinde tüm DDRAM satırlarına ASCII boşluk karakteri olan 20h yazılır. Komutun açılışı aşağıdaki gibidir.

0 0 0 0 0 0 0 1

Karakter yaratma (character generator) RAM'i komutları kullanıcının kendine özgü karakteri yaratması için kullanacağı komutlardır.

İmleci Evine Gönder (Home Curser)

İmlecin evi 0 adresli karakter konumudur. Bu tüm göstergelerde birinci satırın en soldaki karakterinin bulunduğu yerdir.

İmleci yerleştir (Move Curser)

İmleç DDRAM'ın her noktasına gönderilebilir. Gönderilen yer göstergenin görünen kısmında olmayabilir. İmleç istenilen yere adresi belirtilerek ile gönderilir. Komutun açılışı aşağıdaki gibidir.

1 A6 A5 A4 A3 A2 A1 A0

A0 –A6 DDRAM adresini belirtir. Birinci satırın adresi 00-27h aralığında ikinci satırın adresi 40h-67h aralığındadır.

Durum Gösterge Bitinin Okunması (Status Inquiry)

Durum denetlemesi olarak adlandıracağımız bu komut LCD denetleyicisinin yeni komut veya veri kabulüne hazır olup olmadığına bakmak için kullanılır. Aslında tek okuma yapan komuttur, eğer RW hattını GND'ye bağladığınızda bu komutu kullanamazsınız. Bu komutla okunan 8 bitten bir tanesi meşgul (busy) bayrağıdır diğer 7 bit ise en son işlem yapılan belleğin adres bilgisidir. Eğer DDRAM'den işlem yapıldı ise işlemi tamamlanan satırın adresi okunur. Bazen imlecin yeri bilinmediğinde bu komutla bulunabilir. Komutun açılışı aşağıdaki gibidir.

BF AC6 AC5 AC4 AC3 AC2 AC1 AC0

BF=1 ise denetleyici meşgul, BF=0 meşgul değil.

AC6-AC0 en son işlem yapılan belleğin adres bitleri.

İmleci Kaydır (Shift Curser)

İmleç veya gösterge sağa veya sola doğru kaydırılabilir. Komutun açılışı aşağıdaki gibidir.

0 0 0 1 S/C R/L x x

S/C=0 göstergeyi sabit tutar, S/C=1 göstergeyi kaydır.

R/L=0 sola, R/L=1 sağa doğru kaydır.

Karakter Üretecinin Adresinin Ayarlanması (Set CGRAM address)

İmleç konumu belirle komutuna benzer şekilde çalışır. Adresi belirlenen konum bir sonraki yazmanın yapılacağı satırdır. CGRAM 64 satıra sahiptir. Komutun açılışı aşağıdaki gibidir.

0 1 A5 A4 A3 A2 A1 A0

A0-A5 CGRAM adresini temsil eder. CGRAM'ın kullanımı daha sonra tartışılacaktır.

Veri İşlemleri

Veri gönderme için RS hattı YÜKSEK seviyeye getirilmelidir. İlk karakter gönderilmeden önce adres ayarlanmalıdır. Veri CGRAM veya DDRAM belleklerine gönderilebilir.İmleç konumu işlem yapılacak DDRAM satırını gösterir. DDRAM'e veri gönderme işlemi move curser, shift curser, home curser veya reset display komutları sonrası yapılabilir. İmleç kapalı olsa bile

imlecin konumu işlem yapılacak DDRAM satırını gösterir.

Entry mode göre imleç konumu her yazma işlemi sonrası bir azaltılacak veya arttırılacaktır. Eğer karakterler sağa doğru yazılmak isteniyorsa artırma seçilmiş olmalıdır. Sola doğru yazılmak isteniyorsa azaltma seçilmiş olmalıdır. Veri DDRAM'e RW hattı DÜŞÜK seviyeye çekilerek yazılır. DDRAM'den veri okumak için bu hat YÜKSEK seviyeye çekilmelidir. LCD ile veri ve komut iletimi 4 bit veya 8 bit ile yapılabilir. 4 bit veri iletimi seçildiğinde düşük değerli dört bit kullanılmaz. Komutlar veya veriler iki parçaya bölünür. Yüksek değerli dört bit önce sonrada düşük değerli dört bit gönderilir. Bir komut gönderme ancak iki adımda gerçekleştirilebilir. LCD'de reset sonrası 8 bit iletişim seçilmiş olur, 4 bit işleme geçmek için ayarlama sonrası hemen 20h bilgisi gönderilmelidir.

Zamanlama

LCD denetleyicisi normal işlemcilere göre oldukça yavaştır, verilen komutları yerine getirmek için tablo-11.5'de belirtilen süreye ihtiyaç duyar. Bekleme yazılım zamanlayıcısı ile veya meşgul bayrağını kontrol ederek yapılabilir. Yazılım zamanlayıcısı kullanırsanız verilen sürelerin 1,5 katı kadar bekleme yaparsanız daha iyi olur.

Komut	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0	Açıklama
Göstergeyi temizle	0	0	0	0	0	0	0	0	0	1	Göstergeyi temizler imleci evine gönderir, kod üretecinin içeriğini değiştirmez.
İmleci evine gönder	0	0	0	0	0	0	0	0	1	x	İmleç birinci satırın en soldaki karakterin bulunduğu konuma gelir. Yazılacak yeni karakter imlecin bulunduğu yere yazılır.
Otomatik kaydır	0	0	0	0	0	0	0	1	$\overline{ID}$	S	Karakter görüntülendikten sonra imlecin otomatik olarak kayma yönünü ve yazılı göstergenin içeriğinin kayıp kaymayacağını belirler.
Gösterge ve imleç aç kapa	0	0	0	0	0	0	1	D	C	B	D biti göstergeyi açar/kapatır, C biti imleci açar/kapatır, B biti imleci yakar söndürür.
Gösterge veya imleç kaydır	0	0	0	0	0	1	SC	RL	x	x	İmleç veya bulunduğu satırdaki karakterler belleğin içeriğini değiştirmeden kaydırılır.
İşlev seçme	0	0	0	0	1	DL	N	Z	x	x	DL biti veri hattı sayısının 4 veya 8 olarak seçerken, N biti gösterge birden fazla satıra sahipse hangi bir veya çok satır seçimini yapar.
Karakter üreteci adres belirleme	0	0	0	1							Oluşturulacak karakterin adresini belirler.
Veri belleği adres belirleme	0	0	1	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Veri belleği adresini belirler aynı zamanda imlecin konumunu da belirlemiş olur. Bir sonraki karakter bu

											adreste görüntülenir
Meşgul bayrağı ve imleç adresi oku	0	1	BF	A6	A5	A4	A3	A2	A1	A0	Meşgul bayrağı okunur. Bu bayrak 1 ise LCD yeni komuta daha hazır değildir. 0 olduğunda yeni komut veya veri kabul eder. Bu komut aynı zamanda imlecin bulunduğu konumun adresini de okur.
Veri yaz	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Veri belleğine gösterilecek karakterin ASCII karşılığı veya karakter üretecine karakter bilgisi yazılır.
Veri oku	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Veri belleği veya karakter üretecinin içeriği okunur.

Tablo-3 LCD komutları.

Assembler dilinde LCD programlama

İşlem Sırası

Aşağıdaki program deney setinde herhangi bir bağlantı yapmadan LCD ekranın üst satırın ortasında OK karakterlerini görüntüler.

```
$include(reg52.inc)
LCD_VERI        EQU     P2
LCD_EN          EQU     P3.7
LCD_RW          EQU     P3.6
LCD_RS          EQU     P3.5
LCD_VERI7       EQU     P2.7
SAT0            EQU     P1.0
SAT1            EQU     P1.1
SAT2            EQU     P1.2
SAT3            EQU     P1.3
SUT0            EQU     P1.4
SUT1            EQU     P1.5
SUT2            EQU     P1.6
SUT3            EQU     P1.7

KEY_LCD:
        ACALL LCD_AYAR
KEY_LCD1:
        MOV P1,#0F0H            ;Düşük değerli 4 bit çıkış ve “0”
        MOV A,#0F0H
        XRL A,P1                ;Herhangi bir tuşa basılmış mı?
        JZ KEY_LCD1:
        ACALL TUS_BUL           ;Basılan tuşu bul.
        ACALL Veri
        sjmp KEY_LCD1

TUS_BUL:
        MOV R0,#4               ;Satır sayısını belirle
        MOV P1,#01111111B
TUS_BUL1:
        MOV A,P1                ;Satır numarasını oku
        RL A                    ;Bir sonraki satır
```

```
        MOV P1,A                ;Satırı tara
        NOP
        XRL A,P1                ;bu satırda bir tuşa basılmış mı?
{veya alttaki üç komut
        MOV A, P1               ;Sütun bilgisini oku
        CPL A
        ANL A,#0F0H             ;Basılan tuşu denetle
}
        JNZ TUS_BUL2            ;Basılan tuşun değerini belirle
        DJNZ R0, TUS_BUL1       ;Basılmadı ise sonraki sütunu tara
TUS_BUL2:

        MOV A,P1                ;Basılan tuşun kodu
        MOV DPTR, #TUS_TABLO
        MOVC A,@A+DPTR          ;Tuşun değerini tablodan al
        RET


LCD_AYAR:
        MOV A, #38H             ; 2-satır, 5X7 matrix, 8 bit haberleşme.
        ACALL Komut
        MOV A, #0EH             ; LCD aç, imleç aç
        ACALL Komut
        MOV A, #01H              LCD ekranı temizle
        ACALL Komut
        MOV A, #06H             ; yazım sonrası sağa doğru imleci artır
        ACALL Komut
        MOV A, #86H             ; imleci, satır 1 6 nolu karaktere ayarla
        ACALL Komut
        MOV A, #'O'             ; O karakterinin ASCII kodunu gönder
        ACALL Veri
        MOV A, #'K'             ; K karakterinin ASCII kodunu gönder
        ACALL Veri
        ret
Komut:
        ACALL Hazir
        MOV LCD_VERI, A
        CLR LCD_RS              ; RS=0, Komut gönderilecek
        CLR LCD_RW              ; R/W=0, Yazma yapılacak
        SETB LCD_EN             ; E = 1, Enable vurusu oluştur
        CLR LCD_EN              ; E = 0
        RET
Veri:
        ACALL Hazir
        MOV LCD_VERI, A         ; Görüntülenecek karakterin kodunu yaz
        SETB LCD_RS             ; RS = 1, Veri gönderilecek
        CLR LCD_RW              ; R/W = 0, Yazma yapılacak
        SETB LCD_EN             ; Enable vurusu oluştur
        CLR LCD_EN
        RET
Hazir:
        MOV LCD_VERI,#0FFH      ; LCD veri hatlarını giriş olarak ayarla
        CLR LCD_RS              ; RS = 0, komut gönderileceğini bildir
        SETB LCD_RW             ; RW = 1, okuma için hazırla
hazir1:
```

```
        SETB LCD_EN             ; Enable vurusu oluştur
        CLR LCD_EN
        JB LCD_VERI7, hazir1    ; Meşgul olduğu sürece bekle
        RET
TUS_TABLO:
DB
DB
DB
DB
END
```

C dilinde LCD Programlama

Deney:lcd.c

```
//LCD denetleyen Program 2014
// M. Engin
#include <reg52.h>
#include <stdio.h>
#include "lcd_8b.h"
#define SYSCLK  12000000          /* SYSCLK frequency in Hz */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Mesajlar
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
char code *mes1 = "Mikrodenetleyici";
char code *mes2 = "     2014       ";
//*******************************************************
//-------------------------------------------------------
void main (void)
{
        init_LCD();
        lcd_cmdwr(satir1);
        disp_sat(mes1);
        lcd_cmdwr(satir2);
        disp_sat(mes2);
        msec(2000);
                while(1)
        {               //deney buraya yaz1l1r
}
}
```

lcd_8b.h

```
// Header:              LCD ile 8 bit haberleþme kütüphanesi
// File Name:   lcd_8b.h
// Author:              Mustafa Engin
#define DISPLAY_ON 0xC          /* display açik, imleç açik, no blink */
#define RETURN_HOME 0x2         /* Imlec evine*/
#define FUNCTION_SET 0X38       /* 8 bit haberleþme, 5x7 dots, 2 satir */
#define ENTRY_MODE 0X6          /* entry mode: yazma sonrasý imlecin yerini otomatil 1
arttýr */
#define CLEAR_DISPLAY 0x1                       /* Göstergeyi temizle */
#define DISPLAY_OFF 0x8                         /* Göstergeyi kapat */
#define satir1  0x80                    /* satýr 1'e git */
#define satir2  0xC0                            /* satýr 2'ye git */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 LCD Baglantilari 2012 (RW hattý GND'ye Baðlý)
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
        sbit    rs = P2^6;                      //      LCD RS hattý
        sbit    ena =P2^7;                      //      LCD EN hattý
        sfr lcddata = 0x80;                     // PORT 0
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        LCD Baglantilari 2011 (RW hattý sürekli 0) */
        sbit    rs = P0^5;              //      LCD RS hattý
        sbit    rw = P0^6;              //      RW hattý
        sbit    ena =P0^7;              //      LCD EN hattý
        sfr lcddata = 0xA0;     //      PORT 2
        rw = 0;                         //      sürekli 0
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++ */
        void msec(int Delay);
        void lcd_dwr(unsigned char dbyte);
        void lcd_cmdwr(unsigned char dbyte);
/*-------------------------------------------------------------------------
        LCD Initialize
------------------------------------------------------------------------- */
void init_LCD(void)
{
   rs=0;
        lcd_cmdwr(FUNCTION_SET);                //Yazýlým RST ve function set
        lcd_cmdwr(FUNCTION_SET);
        lcd_cmdwr(FUNCTION_SET);
   lcd_cmdwr(DISPLAY_ON);                       //gösterge ve imlec açik
   lcd_cmdwr(ENTRY_MODE);                       // her yazma sonrasi imleci saða arttýr
   lcd_cmdwr(CLEAR_DISPLAY);            // göstergeyi temizle
}
void lcd_dwr(unsigned char dbyte)
{       rs=1;
   ena=1;
   lcddata=dbyte;
   ena = 0;
   msec(5);
 }
void lcd_cmdwr(unsigned char dbyte)
{       rs=0;
   ena=1;
```

```
    lcddata=dbyte;
    ena = 0;
    msec(10);
 }
void msec(int gecikme)                                  /* 1 milisaniyelik zaman geciktirme */
{       int k, z;
        for(k=0; k<gecikme; k++)
  {
        for (z=0; z<500; z++);
  }
}
//*****************************************
void disp_sat (char str[])              //sýfýrla sonlandýrýlmýþ diziyi LCD'ye yazdýrýr
{       unsigned char j = 0;
        while (str[j]!= 0)
                {
                lcd_dwr(str[j]) ;
                j ++;                           // yaz
                }
        while (j < 16)                          //dizi 16 elemandan daha az ie boþ kalan
karekterler boþluk yazdýrýr.
                {
                lcd_dwr(' ') ;
                j ++;                           // yaz
                }
        }
```

Deney: LCD_Keyboard

```
//-------------------------------------------------------------------------
//LCD ve Keyboard Denetim Programı
//-------------------------------------------------------------------------
#include <reg52.h>       // SFR declarations
#include <stdio.h>
#include "8_LCD.h"
//-------------------------------------------------------------------------
#define SYSCLK   12000000                       // Clock speed in Hz
char code *mes0 = "LCD VE KEYBOARD " ;
char code *mes1 = "Deneyi 2009     " ;
//funtion protypes
get_key();
void key_init();
//-------------------------------------------------------------------------
// Ana Program
//-------------------------------------------------------------------------
void main (void)
{       unsigned char key,n;
        static unsigned char buf [16];
        lcd_init();
        key_init();
        lcd_cmdwr(satir1);
        disp_sat(mes0);                                                 //açýlýþ yazýsýný
yazdýr
        lcd_cmdwr(satir2);
        disp_sat(mes1);                                                 //açýlýþ yazýsýný
```

```
yazdýr
        while(1)
        {
                key=get_key();
                if (key!=0)
                {
                lcd_cmdwr(CLEAR_DISPLAY);                       //ekraný temizle
                n=sprintf(buf,"%01bX",key);
        lcd_cmdwr(satir1);
                disp_sat(buf);                                          //açýlýþ yazýsýný
yazdýr

                }
        }
}
```

Matris_keyboard.c

```
#include<reg52.h>
sfr keyport =0x90;     //P1'YE BAÐLI
sbit col1 =P1^0;                //column 1
sbit col2 =P1^1;       //column 2
sbit col3 =P1^2;       //column 3
sbit col4 =P1^3;       //column 4
#define TRUE 1
#define FALSE 0
/* +--------------------------------------+
| Return tipi: void
| Argument: None
| Taným: keyboard'ýn kullandýðý portlarý hazýrlar.
+--------------------------------------+ */
void key_init()
{
       keyport &=0x0F; //satýrlarý giriþ sütünlarý çýkýþ yap
}
/* +---------------------------------------------+
| Return Tipi: unsigned char                    |
| Taným: Tuþ takýmýndan deðeri okur+
-----------------------------------------------+ */
unsigned char get_key()
{
       unsigned char i,k,key=0;
       k=1;
       for(i=0;i<4;i++)
                {                       //4 satýr var
            keyport &=~(0x80>>i);               //her satýrý sýra ile 1 yap
                if(!col1)
                                                {                       //tuþa basýlmýþ mý
                        key = k+0;      //tuþun numarasýný belirle
                        while(!col1);   //tuþun býrakýlmasýný bekle
                        return key;     //tuþ deðeri ile dön
                }
                if(!col2){                      //key2 basýlmýþ mý?
                        key = k+1;      //
                        while(!col2);   //tuþ býrakýlana kadar bekle
                                                                return key;     //tuþ deðerin ile dön
```

```
            }
            if(!col3){                  //key3 basýlmýþ mý?
                  key = k+2;
                  while(!col3);   //tuþ býrakýlana kadar bekle
                  return key;     //rtuþ deðeri ile dö
            }
            if(!col4){                  //key4 basýlmýþ mý?
                  key = k+3;
                  while(!col4);   //tuþ býrakýlana kadar bekle
                  return key;     //tuþ deðeri ile dön
            }
      k+=4;
      keyport |= 0x80>>i;                 //satýrý yüksekte býrak
                  }
       return FALSE;                          //hiç bir tuþa basýlmadý ise 0 ile dön
}
```

Zamanlayıcıların Kullanımı

Deney: Frekans sayıcı

```
//Frekans Sayıc
// M. Engin
#include <reg52.h>
#include <stdio.h>
#include "lcd_8b.h"
#define SYSCLK  12000000          /* SYSCLK frequency in Hz */
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++
 Mesajlar

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++*/
unsigned int say;
unsigned char say1;
sbit T1_giris = P3^5;
char code *mes1 = "Mikrodenetleyici";
char code *mes2 = "     2013       ";
//*****************************************************
void zmn_gecikmesi(void);
void zam_ayar(void);

//-------------------------------------------------------
void main (void)
{
        static data unsigned char buf [16];
        data unsigned int n;
        say=0;
        init_LCD();
        zam_ayar();
        lcd_cmdwr(satir1);
        disp_sat(mes1);
        lcd_cmdwr(satir2);
        disp_sat(mes2);
        msec(2000);
        T1_giris=1;
        EA=1;
        lcd_cmdwr (CLEAR_DISPLAY);
        while(1)
{
        n = sprintf (buf,"f=%3u Hz",say);       //deðeri ASCII'ye dönüþtür
        lcd_cmdwr(satir1);
        disp_sat(buf);                                          //görüntüle

}
}
//Zamanlayýcýnýn kullanýmý
void zmn_gecikmesi(void)
{
// Zamanlayýcý 0 16-bit zamanlayýcý
        TMOD &= 0xF0;                   // T0 bitlerini sýfýrla
```

```
        TMOD |= 0x01;                   // T0 kip 1
        ET0 = 0;                        // kesme yok
// 50 ms gecikme için gerekli deðer
        TH0 = 0x3C;             //      YDB
        TL0 = 0xB0;             //      DDB
        TF0 = 0;                        // T0 Taþma bayraðýný temizle
        TR0 = 1;                        // T='ý baþlat
        while (TF0 == 0);       // Zaman aþýmýný bekle (TF0 == 1)
        TR0 = 0;                        // T0'ý durdur.
        }

void zam_ayar(void)
{
  TMOD=0x51;//t0 16 bit zamanlay1c1, T1 16 bit say1c1
  ET0=1;
  TR0 = 1;                      // T0'ý baþlat
  TR1 =1;                   // T1'i baþlat
}

void zmn0_kesme(void) interrupt 1
{
        TR0 = 0;        //Kesme geldiðinde yapýlmasýný istediðin programý yaz
        TH0 = 0x3C;             //      YDB
        TL0 = 0xB0;             //      DDB
        TF0 = 0;                        // T0 Taþma bayraðýný temizle
        TR0 = 1;                        // T='ý baþlat
        say1++;
        if (say1==20)
        { say1=0;
        say=((TH1<<8)| TL1); /*16 bitlik say1c1y1
                                                                8'er
bit olarak okuyup
                                                        16      bit
degiskene yazdik.*/
        TR1=0;                                          //T1'i sýfýrla
        TH1=0;
        TL1=0;
        TR1=1;
        }
        T1_giris=1;
}
```

Analog Veri İşleme

Deney: ADC ile Voltmetre Tasarımı

```
// ADC denetimi
//Gerçek zaman sayac1 ve EEPROM bellegi denetleyen Program
//Mayis 2014
// M. Engin
#include <reg52.h>
#include <stdio.h>
#include "lcd_8b.h"              //LCD
#include "ds1307.h"              // Gerçek zaman sayaci ayarlama, yazma okuma
#include "24xx512.h"  // EEPROM, 24CXX16, 24CXX32, 24CXX64, 24CXX128, 24CXX256,
24CXX512, 24CXX1024
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Mesajlar
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
char code *mes1 = "Mikrodenetleyici";
char code *mes2 = "     2014       ";
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Global degisken tanimlari
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
unsigned int say;
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 ADC baglantilari
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
sbit ADC_CS  =  P1^7;                    //1 nolu bacak CS
sbit ADC_CLK          =  P1^6;           //7 nolu bacak        CLK
sbit ADC_DO  =  P1^5;                    //6 nolu bacak DO
//sbit ADC_DI  =  P1^4;                  //5 nolu bacak DI ADC0831'de kullanýlmýyor
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
fuction tanimlari
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void UART_Init (void);
char ReadADC(unsigned char channel);        //ADC0831 'i okur
//-------------------------------------------------------
void main (void)
{       static data unsigned char buf [16];
        data unsigned int n,veri_o, veri_y;
        data unsigned char gun, ay, saat, dak, san,yil,veri_adc;
        say=0;
        init_LCD();
        UART_Init();
//Giris mesajini yazdir
        lcd_cmdwr(satir1);
        disp_sat(mes1);
        lcd_cmdwr(satir2);
        disp_sat(mes2);
        msec(1000);                                                    //2          saniye
görüntüle
        lcd_cmdwr (CLEAR_DISPLAY);      //süre doldu temizle
        veri_y=0x14;
        EEPROM_set(0,veri_y);                          /* EEPROM'a veri yaz */
/*saat ilk ayar için bu kýsmý  aç */
```

```
                saat=0X09;
                dak=0x25;
                DS1307_settime(saat, dak, 0x05);
                gun=0x20;
                ay=0x5;
                yil=0x14;
                DS1307_setdate(gun, ay, yil);
        while(1)
{           /*zaman ve tarihi oku*/
                san = DS1307_get(SEC);
                dak = DS1307_get(MIN);
                saat= DS1307_get(HOUR);
                gun= DS1307_get(DATE);
                ay = DS1307_get(MONTH);
                yil = DS1307_get(YEAR);
                veri_o=EEPROM_get(0);                   //EEPROM'u oku
                veri_adc=ReadADC(0);                    //adc'yi oku
                n=sprintf(buf,"%02bX/%02bX %02bX:%02bX:%02bX ", gun, ay, saat,dak,san);
                lcd_cmdwr(satir1);
                disp_sat(buf);                                          //görüntüle
                n=sprintf(buf,"YIL=20%02bX veri=%u", yil,veri_o);
                lcd_cmdwr(satir2);
                disp_sat(buf);                                          //görüntüle
                msec(1000);
                n=sprintf(buf,"ADC=%x veri=%u", veri_adc,veri_o);
                lcd_cmdwr(satir2);
                disp_sat(buf);                                          //görüntüle
        //bilgisayara veri gönderir.
                printf("%02bX/%02bX %02bX:%02bX:%02bX \n", gun, ay, saat,dak,san);
                printf("veri=%u ADC veri=%u \n", veri_o,veri_adc);
}
}
void UART_Init (void)
{
  SCON = 0x52;          // SCON 8 bit veri, 1 start bit, non stop bit
  TH1=0xfd;                                             //9600 baud
  TL1 = TH1;           // baudrate göre deðer hesapla yaz
  TMOD &= ~0xF0;        // TMOD: timer 1  8-bit autoreload  modunda
  TMOD =  0x20;
  TR1 = 1;             // START Timer1
  TI = 1;              // Indicate TX0 ready
}
//---------------------------------------
char ReadADC(unsigned char channel)          //ADC0831'i okur
{  unsigned char i,k;
  unsigned char AdcResult;   // 8 bit
  ADC_CS=0;                                                     // CS'yi aktif yap
  k++;                                                                  //   2
uS bekle
  ADC_CLK=0;                                                    // düsük yap
  k++;k++;
  ADC_CLK=1;                                                    // yüksek yap
  k++;k++;
  ADC_CLK=0;                                                    // düþük yap
  k++;k++;
```

```
  // 8 bit oku --------
  AdcResult=0;                                                    //verini    yazilacagi
yeri sifirla
  for(i=0;i<8;i++)
        {
    ADC_CLK=1;                                                    //vuru üret
    k++;k++;                                                      // 2 uS bekle
    ADC_CLK=0;
    k++;k++;                                                      // 2 uS bekle
    AdcResult<<=1;                                                //eki bit sola ötele
        AdcResult=AdcResult | (ADC_DO & 0x01); //yeni okunan biti eskiler ile birle_tir.
  }
  ADC_CS=1;                                                       //i_lem        tamam
ADC'yi kapat
  return(AdcResult);                                              //Sonucu götür
}
```

Deney: Gerçek zaman Sayıcıve EEPROM Bellek Kullanımı

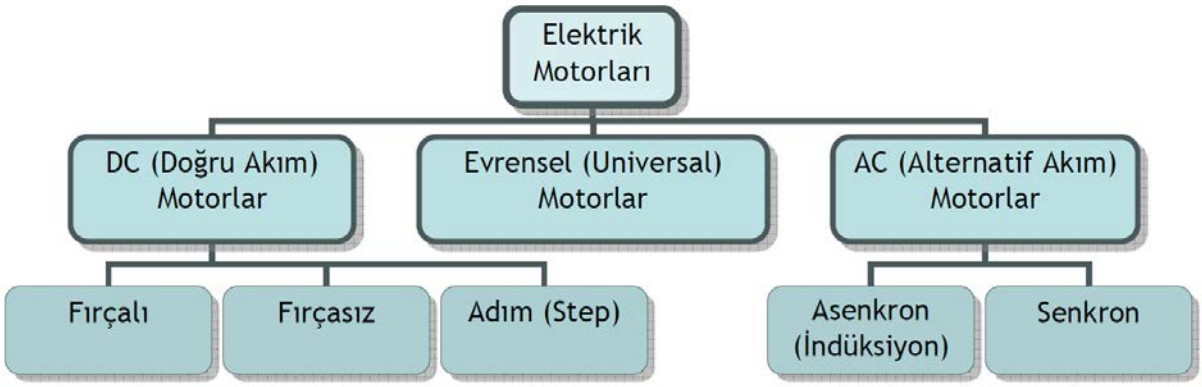
```
//May1s 2014
// M. Engin
#include <reg52.h>
#include <stdio.h>
#include "lcd_8b.h"              //LCD
#include "ds1307.h"              // Gerçek zaman sayaci ayarlama, yazma okuma
#include "24xx512.h"  // EEPROM, 24CXX16, 24CXX32, 24CXX64, 24CXX128, 24CXX256,
24CXX512, 24CXX1024
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Mesajlar

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++*/
char code *mes1 = "Mikrodenetleyici";
char code *mes2 = "     2014       ";
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++
 Global degisken tanimlari

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++*/
//unsigned int say;
//unsigned char say1;
//sbit T1_giris = P3^5;
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++
I2C baglantilari

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++*/
//sbit    SDA = P3^6;                                   //SDA P3.6'ya baglanmal1
//sbit    SCL = P3^7;                                   //SCL P3.7'ye baglanmal1
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++
fuction tanimlari

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++*/
void UART_Init (void);
//-------------------------------------------------------
void main (void)
{
        static data unsigned char buf [16];
        data unsigned int n,veri_o, veri_y;
        data unsigned char gun, ay, saat, dak, san, yil;
        //say=0;
        init_LCD();
        UART_Init();
//Giris mesajini yazdir
        lcd_cmdwr(satir1);
        disp_sat(mes1);
        lcd_cmdwr(satir2);
```

```
        disp_sat(mes2);
        msec(1000);                                                    //2 saniye
görüntüle
        lcd_cmdwr (CLEAR_DISPLAY);      //süre doldu temizle
        veri_y=0x14;
        EEPROM_set(0,veri_y);                          /* EEPROM'a veri yaz */
/*saat ilk ayar için bu kýsmý  aç */
                saat=0X09;
                dak=0x25;
                DS1307_settime(saat, dak, 0x05);
                gun=0x20;
                ay=0x5;
                yil=0x14;
                DS1307_setdate(gun, ay, yil);

        while(1)
{         /*zaman ve tarihi oku*/
                san = DS1307_get(SEC);
                dak = DS1307_get(MIN);
                saat= DS1307_get(HOUR);
                gun= DS1307_get(DATE);
                ay = DS1307_get(MONTH);
                yil = DS1307_get(YEAR);

                veri_o=EEPROM_get(0);           //EEPROM'u oku

                n=sprintf(buf,"%02bX/%02bX %02bX:%02bX:%02bX ", gun, ay, saat,dak,san);
                lcd_cmdwr(satir1);
                disp_sat(buf);                                         //görüntüle
                n=sprintf(buf,"YIL=20%02bX Veri=%u", yil,veri_o);
                lcd_cmdwr(satir2);
                disp_sat(buf);                                         //görüntüle
        //bilgisayara veri gönderir.
                printf("20%02bX/%02bX/%02bX  %02bX:%02bX:%02bX  \n",yil,  gun,  ay,
saat,dak,san);
                printf("veri=%u \n", veri_o);
}
}
void UART_Init (void)
{
  SCON = 0x52;          // SCON 8 bit veri, 1 start bit,  1 stop bit, non parity
  TH1=0xfd;                                             //9600 baud
  TL1 = TH1;           // baudrate göre deðer hesapla yaz
 // TMOD &= ~0xF0;        // TMOD: timer 1  8-bit autoreload  modunda
  TMOD =  0x20;
  TR1 = 1;             // START Timer1
  TI = 1;              // Indicate TX0 ready
}
```

Motor Denetimi

Giriş

Günlük hayatta en çok ihtiyaç duyduğumuz enerji türlerinden birisi de mekanik enerjidir. Bir arabadan uçağa, yazıcıdan fotoğraf makinesine, gelen birini algılayınca açılan alışveriş merkezi kapısından mutfak robotuna kadar hareket enerjisi hayatımızın neredeyse her anında mevcut.Bu hareketi (mekanik enerjiyi) ortaya çıkarabilmek için başka bir enerji türünü kullanmamız gereklidir. Bu bazen benzin, bazen buhar, bazen elektrik, bazen de bir başka enerji türüdür. Bir enerji türünü kullanarak mekanik enerji elde eden makinalara motor denir. Motorlar ve türleri çok çeşitlidir ancak bu yazıda yalnızca elektrik motorlarını (elektrik enerjisinden mekanik enerji üreten makinalar) ve çeşitlerini inceliyoruz. Basit bir elektrik motoru temelde manyetik alan değişimlerinden faydalanarak çalışır. Elektrik motorlarının temel gruplama yöntemlerinden biri aşağıda görünmektedir. Ektrik motorlarının çalışma prensibininiyi anlaşılabilmesi için temel niteliğindeki manyetizma konusu iyi anlaşılmalıdır.

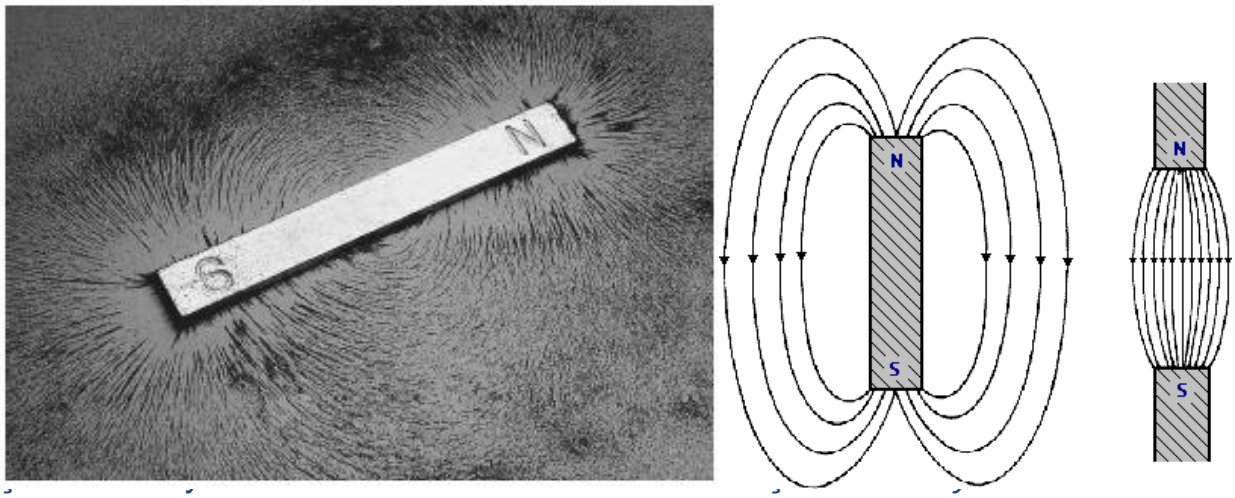


Şekil- 1. Elektrik motorlarının sınıflandırılması

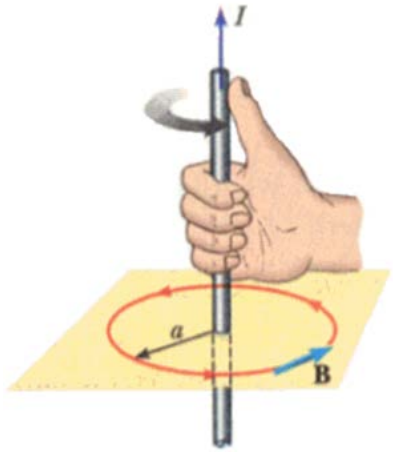
Manyetizma

Bazı maddeler yapıları gereği birbirlerine karşı itici veya çekici kuvvetler uygularlar. Bu maddelere manyetik madde denir. Manyetik özelliği bulunan maddelerden başlıcaları demir, çelik (demir içermesinden dolayı), nikel ve kobalttır. Manyetik materyallerin kuzey (North –N) ve güney (South – S) olmak üzere 2 kutbu vardır. Hayali manyetik alan çizgileri Kuzey kutbundan çıkıp Güney kutbuna girer. Bu manyetik alan çizgileri hiçbir zaman kesişmezler. Aşağıdaki resimde (Şekil 2) demir tozlarının içine konan bir mıknatıstan yayılan manyetik alan çizgilerinin demir tozlarını nasıl etkilediğini görebilirsiniz.

1819 yılında, Hollandalı bilim adamı Hans Christian Oersted, manyetizma ile elektrik arasında çok önemli bir ilişki keşfetti. Oersted, bir iletkenden geçen elektrik akımının sadece sürtünmeden dolayı ısı üretmediğini aynı zamanda kendi çevresinde bir manyetik alan oluşturduğunu fark etti. Akımın yönüyle, akımdan dolayı oluşan manyetik alanın yönünü ise "sağ el kuralı" diye nitelendirilen bir kurala göre belirledi. Bu kurala göre sağ elimizin başparmağını akımın (geleneksel akım yönü, + kutuptan – kutba) yönüne doğru uzatırsak diğer parmaklarımızı kıvırdığımızda manyetik alan çizgilerinin yönünü görebiliriz. Aşağıdaki resimde (Şekil 4) bunu daha net görebilirsiniz.

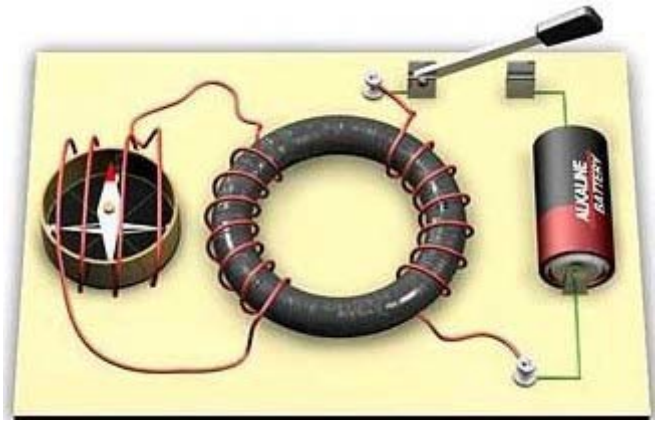


Şekilde görüldüğü gibi eğer akım (I) yukarı yöndeyse, sağ el kuralına göre manyetik alan çizgileri (B) yukarıdan bakıldığında saat yönünün tersinde oluşur. Sağ elinizle deneyerek de görebileceğiniz gibi, eğer akımın yönü değişirse manyetik alan çizgileride yön değiştirir. Oersted'in bu keşfinden sonra 1831 yılında İngiliz bilim adamı Michael Faraday, bir iletkenden geçen akımın manyetik alan oluştururken acaba bir manyetik alanın da bir iletken üzerinde akım oluşturup oluşturamayacağını (indüksiyon) merak etti. Bunun üzerinde Faraday aşağıdaki resimdekine benzer bir düzenek hazırladı.



Şekil- 4. Sağ el kuralı

Düşüncesine göre anahtarı kapattığında sağdaki sargı nedeniyle demir çekirdek manyetik olacak ve soldaki sargı da oluşan bu manyetik alan nedeniyle üzerinden akım geçirecekti. İndüklenen bu akım da, Oersted'in keşfine göre pusula etrafında manyetik alan oluşturacak ve pusula iğnesi sapma yapacaktı.



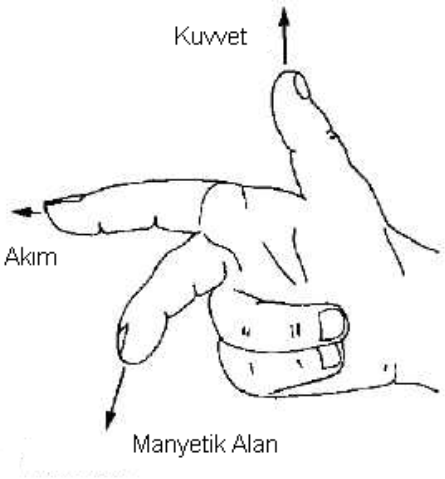
Şekil- 5. Manyetik alanine belirlendiği deney düzeneği

Ancak durum Faraday'ın tahmin ettiği gibi olmadı. Anahtarı kapatıp beklediğinde pusulanın sapmadığını gördü ancak bir şey fark etmişti. İşte bu fark ettiği nokta gelecekte üretilecek motorların ve üreteçlerin temelini oluşturacaktı. Fark ettiği nokta şu idi; anahtarı kapatıp beklediğinde pusula da herhangi bir sapma olmuyordu ancak anahtarı kapattığı anda pusula çok hızlı bir şekilde sapıyor ve eski pozisyonuna geri dönüyordu. Bunu bir de anahtarı açarak denedi ve gördü ki bu kez pusula çok hızlı bir şekilde ters tarafa sapmış ve eski pozisyonuna geri dönmüştü. Faraday bu deneyden, akımın beklediği gibi sabit bir manyetik alandan değil değişen manyetik alandan dolayı oluştuğunu (indüklendiğini) anladı ve Faraday Yasası ortaya çıktı.

$$emf = -N\frac{d\Phi_B}{dt}$$

Demir çekirdeğin solundaki iletkeni kesen manyetik alan çizgileri (manyetik akı) değiştiği anda, iletken üzerinde bu değişime karşı koyacak bir akım oluşuyordu ve manyetik alan değişmediği sürece akım kayboluyordu. Değişen manyetik akı ile oluşan akımın yönü arasındaki bu ilişkiyi ise Faraday'dan 2 yıl sonra 1833'te Heinrich Lenz keşfetti. Faraday Yasası'ndaki (-) işareti Lenz Yasası'ndan gelmektedir. Lenz yasasına göre manyetik alan değişiminden dolayı indüklenen akım, kendisini oluşturan manyetik alan değişimine zıt bir manyetik alan oluşturacak yönde akar.

Bu keşiflerin ışığında yapılan çalışmalar sonucu günümüz elektrik motorlarının temelini oluşturan önemli buluşlar yapıldı. Bir manyetik alanda üzerinden akım geçen bir iletkenin manyetik bir kuvvete maruz kaldığı görüldü. Bunun sebebi, iletkenin içinde bulunduğu manyetik alan ile, iletkenden geçen akımın oluşturduğu manyetik alanın etkileşimiydi. Bu kuvvetin yönü, iletkenden geçen akım ve ortamdaki manyetik alan arasında yeni bir sağ el kuralı oluşturuldu.

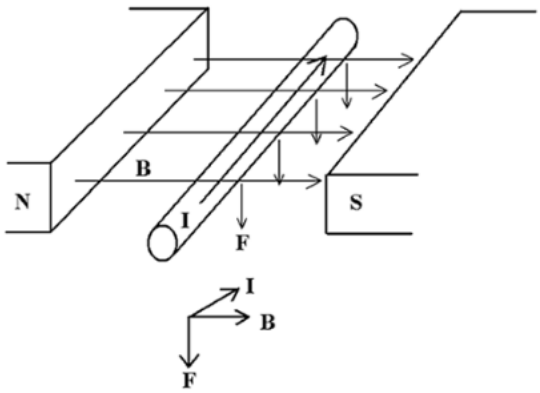


Şekil- 6. Sağ el kuralı.

Yandaki resimde de görebileceğiniz gibi, bu kurala göre sağ elimizin başparmağını, işaret parmağını ve orta parmağını birbirlerine 90 derece açı yapacak şekilde açarsak; işaret parmağımız akımın yönünü, orta parmağımız manyetik alanın yönünü (Kuzey kutbundan (N), Güney kutbuna doğru (S)) ve başparmağımız da manyetik alanın iletkene uyguladığı kuvvetin yönünü gösterir. Bu bileşenlerin birbirlerine dik olmalarının nedeni aşağıdaki formülde görüldüğü gibi, kuvvetin, akım ve manyetik alanın vektör çarpımı olmasıdır.

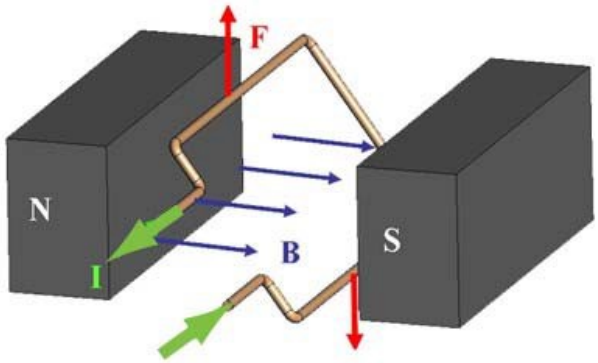
$$\vec{F} = \vec{I} \times \vec{B}$$

Manyetik alan ve akımdan dolayı oluşan bu kuvvet elektrik motorlarındaki hareketi elde etmemizi sağlar.



Şekil- 7. Manyetik alanine yönü.

Yukarıda gördüğümüz Kuvvet, Akım, Manyetik alan ilişkisi elektrik motorlarındaki hareketin olduğu gibi elektrik üreteçlerindeki akımın da temelidir. Yukarıdakine benzer bir şekilde, eğer üzerinden akım geçmeyen bir iletkeni bir manyetik alanda hareket ettirirsek, iletkenin manyetik alanı kesmesi nedeniyle üzerinde bir akım oluşur (indüklenir). Bu da üreteçlerin (jeneratör) ana fikridir.

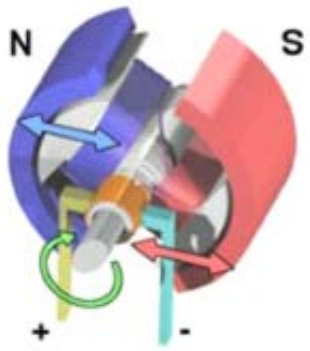


Şekil- 8. Generatörün prensip şeması.

Buradan hareketle aslında her motorun bir üreteç ve her üretecin bir motor olduğunu söyleyebiliriz. Yapısal olarak çok benzerdirler. Ancak motorlar, iletken üzerinden geçen akımdan dolayı oluşan kuvvet sonucu dışarıya mekanik enerji verirken, üreteçler dışarıdan uygulanan tork sonucu indüklenen akımı dışarıya elektrik enerjisi olarak verirler. Bu kadar manyetizma bilgisinden sonra DC (Doğru Akım) Motorların yapısına ve çalışma prensiplerine geçebiliriz. İlk olarak en temel doğru akım motoru olan fırçalı doğru akım motorunu (Brush DC Motor) inceleyeceğiz.

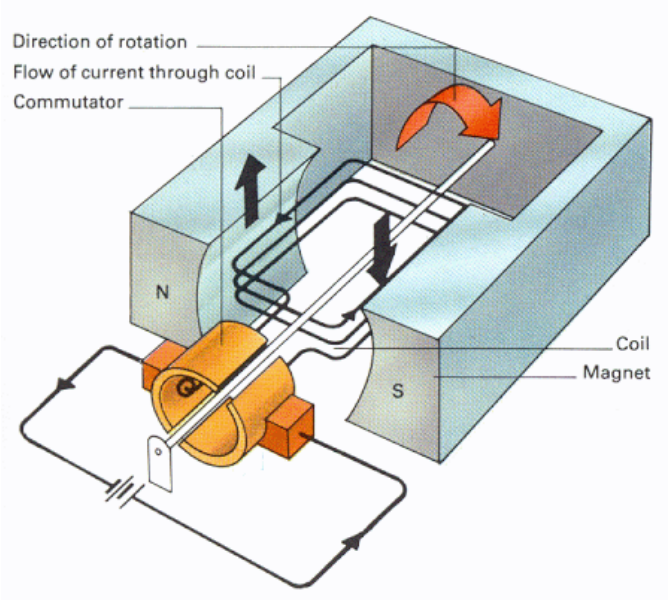
Fırçalı Doğru Akım Motoru (Brush DC Motor)

En temel doğru akım motoru fırçalı doğru akım motorudur. Motor, manyetik alanda merkezi dışarıya mekanik enerji aktaran bir şafta bağlı olarak dönen bir kısmı içerisinde barındıran mıknatıslı bir yapıdır. Manyetizma konusunda anlatılan yasalara göre bir manyetik alanda üzerinden akım geçen bir iletkene manyetik alan tarafından kuvvet uygulanır (Lorentz Force). Sabit manyetik alanı oluşturan ve içerdeki dönen kısmı çevreleyen yapıya stator denir. Üzerinde sargılar bulunduran ve şafta bağlı olarak manyetik alanda dönen iç kısma ise rotor (bazen armatür) denir.



Şekil- 9. Motorun çalışma prensibi.

Armatür, Lorentz kuvvetleri etkisinde hareket ederken aşağıdaki resime göre dik dikey pozisyonu geçtiğinde kuvvetler zıt yönde etki etmeye başlar. Bu da dönme hareketinin durmasına neden olur. Bu nedenle armatürdeki sargılara akımın geçmesini sağlayan çeviriciler (commutators) besleme voltajından gelen iletkenlere fırçalarla bağlıdır. Bu fırçalar sayesinde motor kuvvetin yön değiştireceği dikey pozisyonu geçtikten sonra sargılardaki akımın yönü değişir ve kuvvetler aynı yönde etki etmeye devam eder.



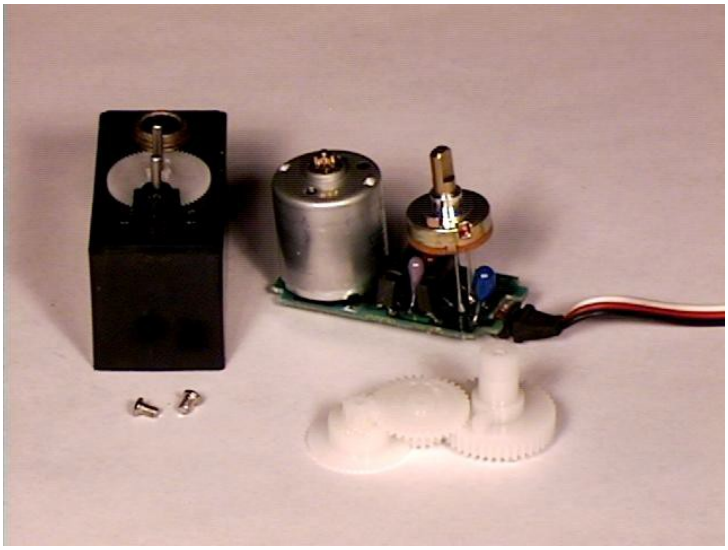
Şekil- 10. Motorun hareket ettirilmesi.

Servo Motorlar

İngilizce “servant” (hizmetçi, uşak) kelimesinden gelen Servo, bir DC motorun şaftının dişli sistemiyle bir potansiyometreye bağlanmasıyla oluşturulur. Servo üzerindeki dahili kontrol devresi potansiyometredeki voltaj bölünmesine göre servonun pozisyonu hakkında bilgi edinir ve gelen sinyalle kıyaslayıp doğru pozisyona dönmesini sağlar. Servo motor açılı dönebildiği için çok geniş kullanım alanları vardır.

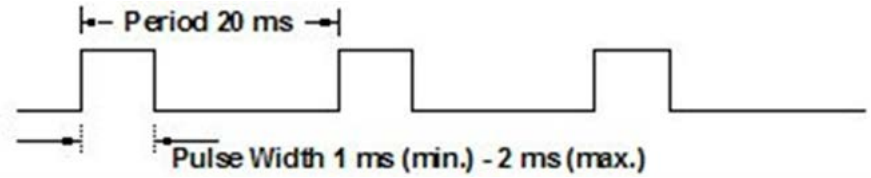
Şekil- 11. Servo motorun görüntüsü.

Servo motorlardan genellikle 3 adet kablo çıkar. Kırmızı renkte olan pozitif kutup kahverengi (veya siyah) olan kutup ise referans kutuptur (= 0). Farklı renkteki (genelde sarı veya turuncu) kablo ise sinyal kablosudur. Mikrodenetleyici servo ile bu sinyal kablosu üzerinden haberleşir. Bu haberleşme yönteminin adı Darbe Kod Modülasyonudur. (Pulse Code Modulation).



Şekil- 12. Servo motorun iç yapısı.

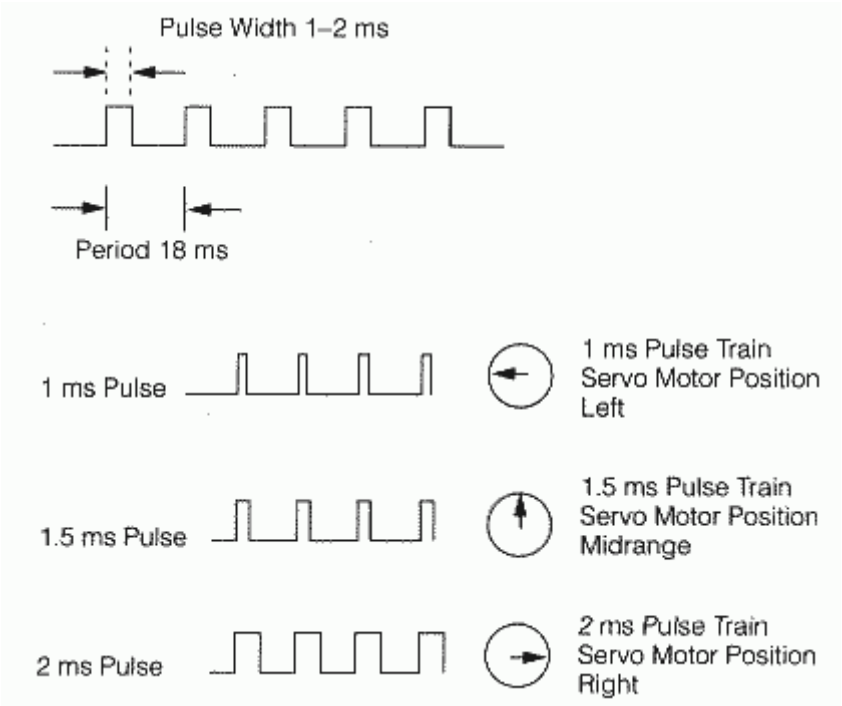
Servo üzerindeki kontrol devresi yaklaşık olarak 20ms'de bir komut bekler. Bu devre gelen sinyalin yüksek voltaj süresine göre konumlama yapar. Standart bir servo yaklaşık 180 derece dönebilir. Bu sınırı hem kontrol devresi belirler hem de dişliler üzerinde bu sınırı korumak için bir diş mevcuttur. Servo açıları orta konumda 0 derece, en sağ konumda +90 derece ve en sol konumda -90 derece olarak adlandırılır. Standart servolar yaklaşık olarak 1.5ms yüksek voltaj sinyalinde orta konuma (0 derece) gelirler. -90 derece için 1ms, +90 derece için 2ms yaklaşık değerlerdir. Ancak bu değerler aynı üreticinin eş servolarında bile farklılık gösterebildiğinden bu sınırları deneme yanılma yoluyla bulmak gerekir. Peki 1.5 ms yüksek voltaj sinyali ne demek? Aşağıdaki resme bakalım.



Şekil- 13. Örnek vuru

Bir görev için toplam süre 20ms'yi aşmadığı sürece düşük voltaj süresinin bir önemi yoktur. Servo konumu için önemli olan yüksek voltaj süresidir. Resimde gördüğünüz gibi eğer bir servoya 1.5ms yüksek voltaj ve 18.5ms (bu değer 20ms'yi aşmayacak biçimde herhangi bir değer olabilir) düşük voltaj uygularsak servo 0 konumuna gidecektir. Benzer şekilde +90 için 2ms, -90 için 1ms yaklaşık değerleridir.

Aşağıdaki resimde de toplam darbe süresi 18ms olarak belirlenmiştir. Yüksek voltaj süresi ile servo konumu arasındaki ilişkiye dikkat ediniz.



Şekil- 14. Vuru genişliği ile motor posisyonunun değiştirilmesi.

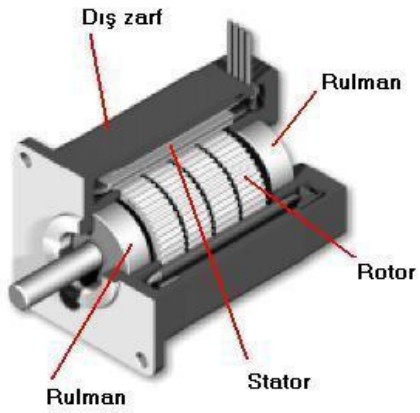
Servo motorlar bir konuma hareket ettikten sonra o pozisyonda kalırlar. Ancak servoya darbe göndermeye devam edilmezse dışarıdan bir kuvvet uygulandığında servo konumunu kaybeder. Servonun bulunduğu konumu değiştirmeye çalışan kuvvetlere karşı direnç gösterebilmesi için sürekli o konuma dair sinyal alması gerekir. Bu durumda servo, servonun üretim özelliği olan tork değerine kadar uygulanan torklara direnebilir.

Adım Motorların Tanımı ve Yapısı

Adım motorları adından da anlaŞılacaşı gibi adım adım hareket eden yani sargılarından birinin enerjilenmesi ile sadece 1 adım hareket eden motorlardır. Bu adımın kaç derece olacaşı motorun tasarımına başlıdır. Bu husus ileriki konularda anlatılacaktır. Adım motor, elektrik enerjisini dönme hareketine çeviren eletro-mekanik bir cihazdır. Elektrik enerjisi alındışında rotor ve buna başlı Şaft, sabit açısal birimlerde (adım-adım) dönmeye baŞlar. Adım motorlar, çok yüksek hızlı anahtarlama özellişine sahip bir sürücüye başlıdırlar (adım motor sürücüsü). Bu sürücü, bir encoder, PC veya PLC'den giriŞ darbeleri (pals) alır. Alınan her giriŞ darbesinde, motor bir adım ilerler. Adım motorlar bir turundaki adım sayısı ile anılırlar. Örnek olarak 400 adımlık bir adım motor bir tam dönüŞünde (360º) 400 adım yapar. Bu durumda bir adımın açısı 360/400 = 0.9º derecedir. Bu deşer, adım motorun hassasiyetinin bir göstergesidir. Bir devirdeki adım sayısı yükseldikçe adım motor hassasiyeti ve dolayısı ile maliyeti artar.

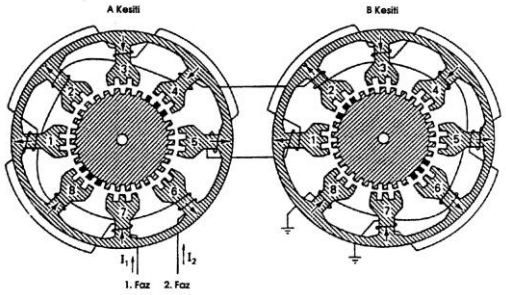
Şekil- 15. Çeşitli adım motorlar

Adım motorlar, yarım adım modunda çalıŞtıklarında hassasiyetleri daha da artar. Örnek olarak 400 adım/tur deşerindeki bir adım motor, yarım adım modunda tur baŞına 800 adım yapar. Bu da 0.9º"ye oranla daha hassas olan 0.045º bir adım açısı anlamına gelir. Bazı adım motorlarda mikroadım teknişi ile adım açılarının daha da azaltılması söz konusudur.



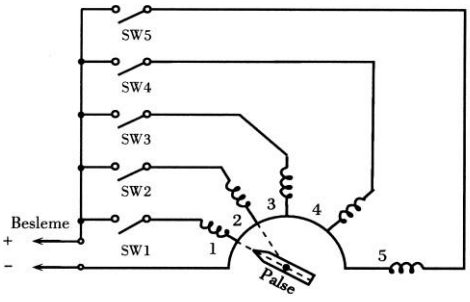
Şekil- 16. Adım motorun kesiti

Yapılarını anlayabilmek için bir adım motorun kesitini Şekil-1.1"de inceleyelim. Bir adım motor Şekil-16"de görüldüşü gibi stator, rotor, bunları kapatan bir dIŞ zarf, rotora başlı Şaftın rahat hareket etmesini saşlayan rulmanlardan oluŞmuŞtur. Adım motor statorunun birçok kutbu (genellikle sekiz) vardır. Bunların polaritesi elektronik anahtarlar yardımıyla deşiştirilir. Rotorun mıknatıslışı ise ya sabit mıknatıs ile veya dIŞ uyartım metodlarıyla oluŞturulur. Daha iyi seçiçilik elde etmek için rotor ve stator üzerine küçük dişler açılmaktadır.



Şekil- 17. Sekiz kutuplu adım motorun iç yapısı

Adım motorlar robot teknolojisinde sıkça kullanım alanı bulmuŞtur. Ayrıca maliyetinin düŞük olması dişer motorlara (servo) karşı bir üstünlüşüdür. Adım motorların tercih edilmesini ikinci bir nedeni tutma karakteristlişinin robotlarla başdaşmasıdır. Adım motorun çalışma esasları Şekil-17'de gösterilmiŞtir. Anahtarlar yardımıyla sargılara enerji uygulandışında rotor enerji uygulanan sargının karŞısına gelerek durur. Bu dönme miktarı motorun yapısına başlı olarak değişir. Bu dönme açısı adım motorlarda belirleyici bir parametredir. Adım motoru sürekli hareket ettirmek istersek sargılara sırasıyla enerji vermeliyiz. Bir sargıya enerji verdişimizde rotor sargını karŞısına gelerek durur. Diğer sargıya eneryi verinceye kadar burada kilitlenir. Bu da adım motorların bir özellişidir.



Şekil- 18. Adım motorun çalışma Şeması

Adım motorların özellikleri aŞaşıdaki gibi sıralanabilir.

- Hata yalnız adım hatasıdır.
- Motor bakımı kolaydır.
- Tasarım maliyeti ucuzdur.
- Otomatik kilitleme özellişine sahiptir.
- Yüke yeterli momenti saşlar.
- Isınma gibi olumsuzluklardan meydana gelebilecek zararlar en azdır.
- Hızı programlama yoluyla ayarlanabilir.
- Mikrobilgisayarlar ile kolayca kontrol edilebilir.
- Çalışma sırasında hızı sabit kalır.
- Kullanım ömrü uzundur.

Adım motorların kullanıldışı yerleri sıralayacak olursak; bant sürücüleri, imalat tezgâhları, yazıcılar, teyp sürücüleri, tıbbi cihazlar, makine tezgâhları, dikiŞ makineleri, taksimetreler, kart okuyucular vb. olarak sayılabilir. Sonuç olarak adım motorlar; her türlü denetlenmiŞ hareket ve pozisyon gerekli olan yerlerde, dijital bilgileri mekanik harekete çeviren bir eleman (transduser) olarak görev yapar. Adım motor seçiminde birçok kıstas mevcuttur. En iyi seçimi yapabilmek için ekonomik olmasını yanında kapsamlı mekanik yapı, yükün durumu ve elektronik sürücü devre ihtiyaçlarını göz önüne alınması gerekir. En basit seçim motorun tork ihtiyacı bakımından verimlilişi esas alınarak yapılandır.

Adım Motorların Çeşitleri

Adım motorlar yapılarına göre 5 çeşittir.

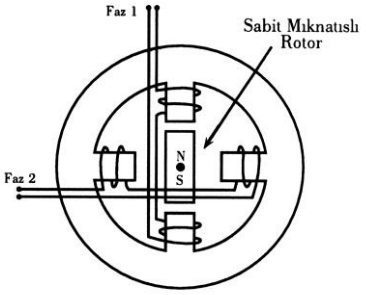
- Sabit mıknatıslı adım motorlar (PM)

- İki fazlı sabit mıknatıslı iki fazlı adım motor
- Orta uçlu sargılara sahip sabit mıknatıslı adım motor
- Disk tipi sabit mıknatıslı adım motor
- Dört fazlı sabit mıknatıslı adım motor
- Deşişken relüktanslı adım motorlar (VR)
- Tek parçalı
- Çok parçalı
- Hybrid adım motorlar
- Hidrolik adım motorlar
- Lineer adım motorlar

Sabit Mıknatıslı Adım Motorlar

Sabit Mıknatıslı İki Fazlı Adım Motor

En basit olarak sabit mıknatıslı adım motoru, oyuklu dört kutuplu statör içinde dönen iki kutuplu sabit mıknatıslı rotordan meydana gelmiŞtir. Böyle bir adım motorun yapısı ġekil-1.4‟te verilmiŞtir.

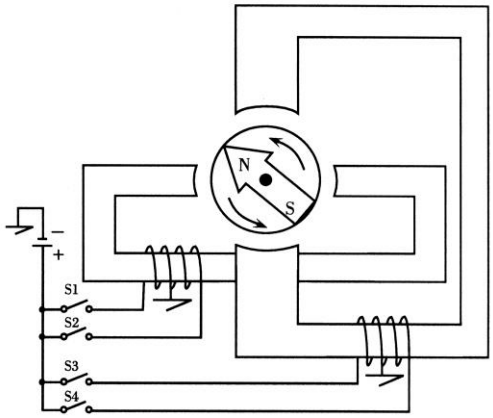


Şekil- 19. Sabit mıknatıslı adım motorun yapısı

Bu motorun çalışması, temel çalışma esaslarında açıklandışı gibidir. Birinci sargıya (faz 1‟e) gerilim uygulandışında rotor, bu sargıların karşısında duracak Şekilde hareket eder. Birinci sargı gerilimi kesilip ikinci sargıya (faz 2‟e) gerilim uygulandışında rotor, bu kez ikinci sargıların karşısında olacak Şekilde döner ve durur. Bu ġekilde 90°lik dönme tamamlanmıştır.(birinci adım=360º:4=90º) Dönmenin devamı için bu kez faz 1‟e uygulanacak gerilim öncekinin tersi yönünde olmalıdır. Bu dönüşün aynı yönde olması için Şarttır. Çünkü faz 1‟e gerilim deşiştirmeden uygulasaydık rotor ilk durumuna geri dönecekti. Bir ileri bir geri hareket ise dönme hareketi vermeyecektir.

Orta Uçlu Sargılara Sahip Sabit Mıknatıslı Adım Motor

Faz 1 ve faz 2‟ye uygulanacak gerilimi deşiştirmenin en kolay yolu orta uçlu (merkez uçlu) sargı kullanmaktır (Şekil-20). Çünkü orta uca göre yan uçlara uygulanacak aynı gerilim birbirinin zıttı manyetik alanlar oluŞturur. Ayrıca iki fazlı orta uçlu bobinlere sahip adım motora, orta uç üzerinden ayrı ayrı gerilim uygulanırsa dört fazlı motor gibi çalışması saşlanabilir. Bu durum uyartım yöntemlerinde anlatılacaktır. Adım motorun sargılarına uygulanacak gerilim yönüne göre rotorun hareketi saat ibresi yönünde (CW) veya saat ibresinin tersi yönünde (CCW) gerçekleştirilebilir. PM motorun statör sargıları DC kare dalga ile sürülür. Kare dalga darbeler ard arda uygulanacak olursa rotor, normal motorlarda olduşu gibi sabit hızda döner.



Şekil- 20. Orta uçlu sabit mıknatıslı adım motorun yapısı

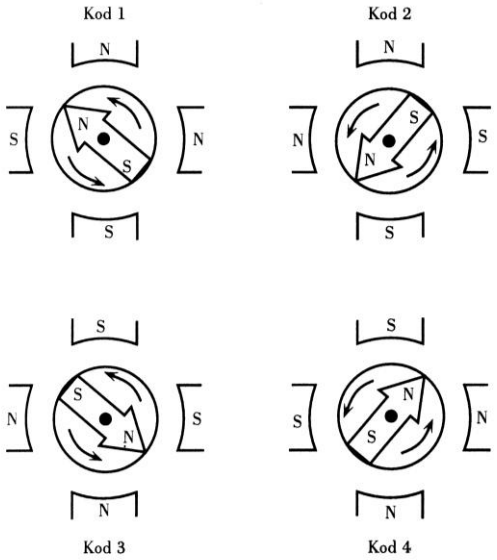
Orta uçlu sabit mıknatıslı bir adım motorun en basit kontrolü Şekil-21 ile gerçekleştirilebilir. Adım motorun çalışması için S1, S2, S3 ve S4 anahtarları üzerinden Faz 1 ve Faz 2 sargılarına sırası ile uygun faz ve gerilim uygunlanmalıdır. Devrede kullanılan motorun 90°'lik adımlarla dönmesini istersek Tablo 1.1'de verilen dört deşişik çalışma durumunu (kodlarını) ard arda uygulamalıyız.

Kod	S1	S3	S2	S4
1	0	0	1	1
2	1	0	0	1
3	1	1	0	0
4	0	1	1	0

Tablo-1. Sabit mıknatıslı orta uçlu adım motorun çalışma tablosu

Şekil-1.6'daki anahtarların dört deşişik çalışma durumunu (kodunu) veren Tablo 1 ve bu kodlara göre rotorun hareketleri adım adım çizilmiŞtir (Şekil-1.6). Bu Şekiller üzerinden S1, S2, S3 ve S4 anahtarlarının kapalı (1) açık (0) oluŞlarına göre motorun iki kutup arasında 90º lik adımlarla ve saat ibresinin tersi yönünde (CCW) nasıl döndüşü görülmektedir.

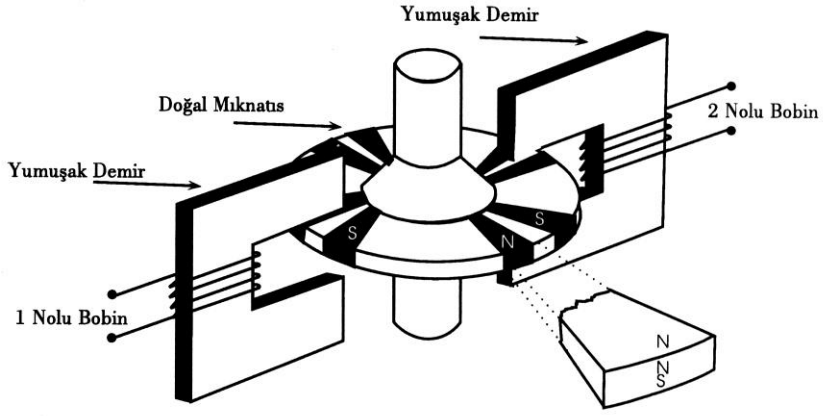
Ilk adım yani kod 1 için S2 ve S4 anahtarları kapatılır. Faz 1ve Faz 2 sargılarına uygulanan gerilim sonucu rotor, Şekil-1.6'daki kod 1 çalışmasını tamamlar ve durur. S4 anahtarı kapalı iken S2 açılıp S1 kapatılırsa rotor bu kez kod 2 çalışmasını tamamlar yani 90º döner ve durur. Kod 3 çalışması için S1 anahtarı kapılıyken S4 açılıp S3 kapatılır. Aynı Şekilde kod 4 çalışması için ise S3 kapalıyken S1 açılıp S2 kapatılmalıdır. Anahtarlar bu sırayla deşiştirilmeye devam edildişinde rotorda dönmeye devam edecektir. Adım motorun çalışma durumları deşiştirilmeye devam edildişi sürece buna başlı olarak da motor dönmeye devam edecektir. Adım motorun çalışma durumlarınını deşiştirmesinde sadece bir anahtarın deşiştişine dikkat ediniz. Bu durum, rotorun eşit adımlarla ve aynı yönde dönmesini saşlar.



Şekil- 21. Sabit mıknatıslı adım motorun çalışması

Disk Tipi Sabit Mıknatıslı Adım Motor

Rotoru ince ve mıknatıslışı seyrek olacak Şekilde yapılan adım motorlara disk tipi sabit mıknatıslı adım motor denir.

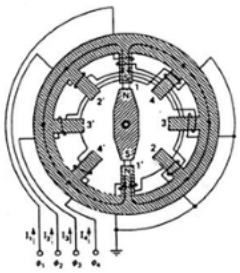


Şekil- 22. Disk tipi adım motorun yapısı

Disk Şeklindeki rotorun ince oluŞundan dolayı, bu disk üzerine 100"ün üzerinde sabit manyetik kutuplar yerleştirilir. Bu manyetik kutuplar sadece diskin kenarlarına yerleştirilirse bile yeterli olacaktır.

Dört Fazlı Sabit Mıknatıslı Adım Motor

Şekil-1.8"de görülen sabit mıknatıslı adım motorun dört fazı ve her faza ait iki kutup bulunmaktadır. Motorun adım açısı 45° dir. Buna göre dört fazlı sabit mıknatıslı adım motorun çalışmasını Şu Şekilde açıklayabiliriz.



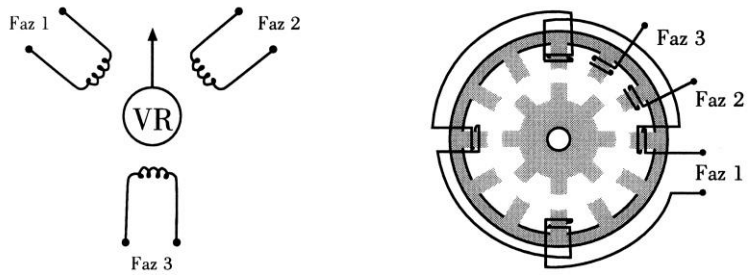
Şekil- 23. Dört fazlı sabit mıknatıslı adım motorun yapısı.

Sabit mıknatıslı adım motorun 180°"lik hareket yapması için faz sargıları 1,4, 3, 2 sırasıyla enerjilendirilir. 1. faz enerjilendirildişinde I1 akımı 1" deki kutup sargılarından geçerek devresini tamamlar. Rotorun N kutbunun karŞısındaki statör kısmı S ile kutuplandırılır. Rotorun S kutbunun karŞısındaki statör kısmı N ile kutuplandırılır.Birinci fazın enerjisinini kesilip dördüncü faz enerjilendirildişinde I4 akımı 4" ve 4 nu lu kutup sargılarından geçerek devresini tamamlar. 4 nu lu kutbun altı S ile 4" kutbunun üstü N ile kutuplanır. Böylece rotor 4-4" statör kutupları hizasına gelerek 45° lik hareket gösterir. Dördüncü fazın enerjisi kesilip üçüncü faz enerjilendirildişinde rotor 45° lik hareketle 3-3" statör kutupları hizasına gelir. Üçüncü fazın enerjisi kesilip ikinci faz enerjilendirildişinde rotor 45° lik hareketle 2- 2" statör kutupları hizasına gelir.

Deşişken Relüktanslı Adım Motorlar

Deşişken relüktanslı adım motorlarında da sabit mıknatıslı adım motorlarda olduşu gibi en az dört kutuplu statör bulunur. Sabit mıknatıslı adım motorlarından tek farkı ise rotorun, sabit mıknatıs yerine artık mıknatıslık özellişi göstermeyen olması ve diŞler açılmıŞ yumuŞak demirden imal edilmesidir. DiŞler, silindir eksenine paralel olarak açılmıŞ oluklarla ŞekillendirilmiŞitir. Şekil-1.9"da üç fazlı deşişken relüktanslı adım motorunun yapısı görülmektedir. Statördeki diŞ sayısının rotordaki diŞ sayısından fazla olduşu Şekilden görülmektedir. Örnekteki statorda 12 diŞ (kutup), rotorda ise 8 diŞ (kutup) bulunmaktadır. Stator kutupları arasındaki merkez açı 30º (360:12=30º) olduşu halde rotor kutupları arasında merkez açı 45º (360:8=45) olmaktadır.

Çalışması



Şekil- 24. Değişken mıknatıslı adım motorun yapısı

Faz 1"e ait seri başlı dört sargıya DC gerilim uygulandışında bu sargıların etrafında oluŞan manyetik alanlar rotor kutuplarını mıknatıslar ve rotoru bu sargıların karŞısına getirecek kadar hareket ettirir. Bu anda dişer kutuplar ise stator ve rotordaki diş sayısı eşit olmadışından stator kutupları karŞısında deşildir. Bu durum Şekilde görülmektedir. Faz 1enerjisini kesip faz 2"ye uygularsak bu kez statorda faz 2 bobinleri etrafında

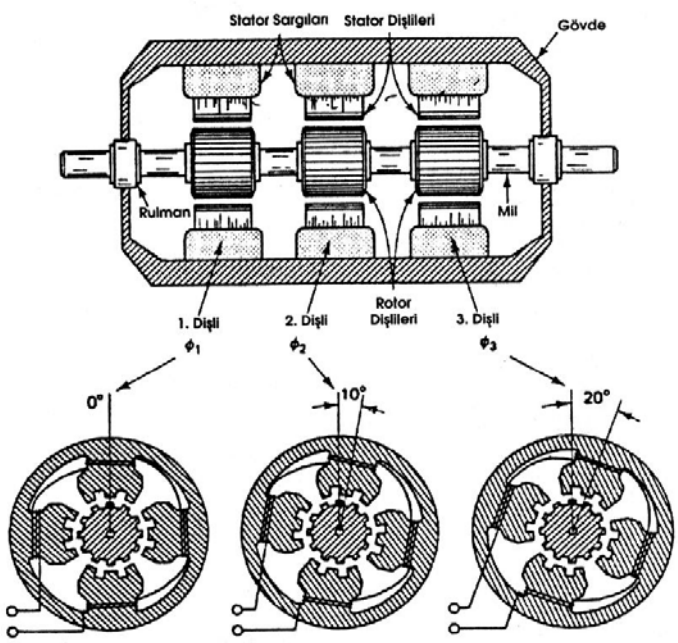
meydana gelen manyetik alan kutupları, rotorun faz 1 karşısındaki kutuplarını kendine çeker. Böylece rotorun dönmesi saşlanır. Üç fazlı (üç sargılı) sistemlerde rotorun devamlı dönmesi için stator argıları ard arda enerjilendirilmelidir. Faz 2 enerjisi kesilip faz 3"e uygulandışında bu kez rotor kutupları statordaki faz 3 sargılarının bulunduşu kutupların karŞısına gelecek Şekilde döner ve durur. Rotorun dönme yönü (saat ibresi yönü veye tersi) fazlara uygulanacak gerilimlerin yönüne başlıdır. Deşişken relüktanslı motorlarda rotor, hafif ve küçük boyutlu yapılır. Rotor ölçülerinin küçük olması eylemsizlik momentinin de küçük olmasını saşlar. Bunun sonucu fazlara uygulanan gerilim meydana getireceşi moment sebebiyle rotor çok hızlı hareket eder. Deşişken relüktanslı motorların harekete baŞlama, durma ve dönme adımları sabit relüktanslı adım motorlarından daha hızlıdır.

Deşişken relüktanslı adım motorlar iki çeşittir.

Tek parçalı deşişken relüktanslı adım motorlar: Statör kutupları tek parçadan oluşan adım motorlardır. Stator ve rotorları tek dişli olarak yapılan adım motorlara tek parçalı VR adım motor denir. Tek parçalı adım motor kesiti Şekil-25'de görülmektedir.Rotorun hareketinin saat ibresi yönünde devam etmesini istiyorsak 1, 3 ve 2 numaralı fazları sırasıyla sürekli olarak enerjilendirmeliyiz.

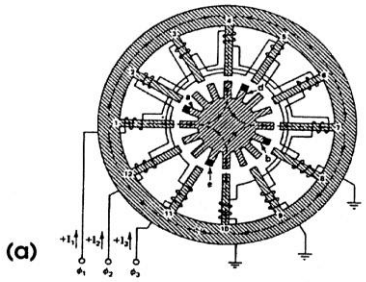
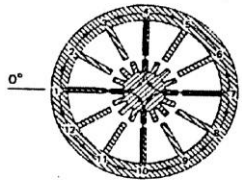
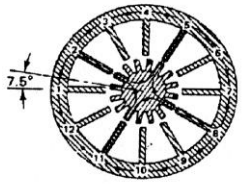
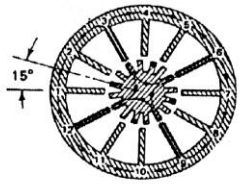
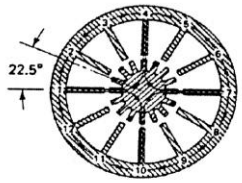
Çok parçalı deşişken relüktanslı adım motorlar

Üç fazlı deşişken relüktanslı adım motor tasarımı Şekil-25'te verilmiŞtir. Rotor 12 dişli olarak yapılmıŞtır. Stator ise her kutupta üç diŞli olmak üzere dört kutuptan ve böylece 12 dişliden oluŞmuŞtur.



Şekil- 25. Çok parçalı deşişken mıknatıslı adım motorun yapısı

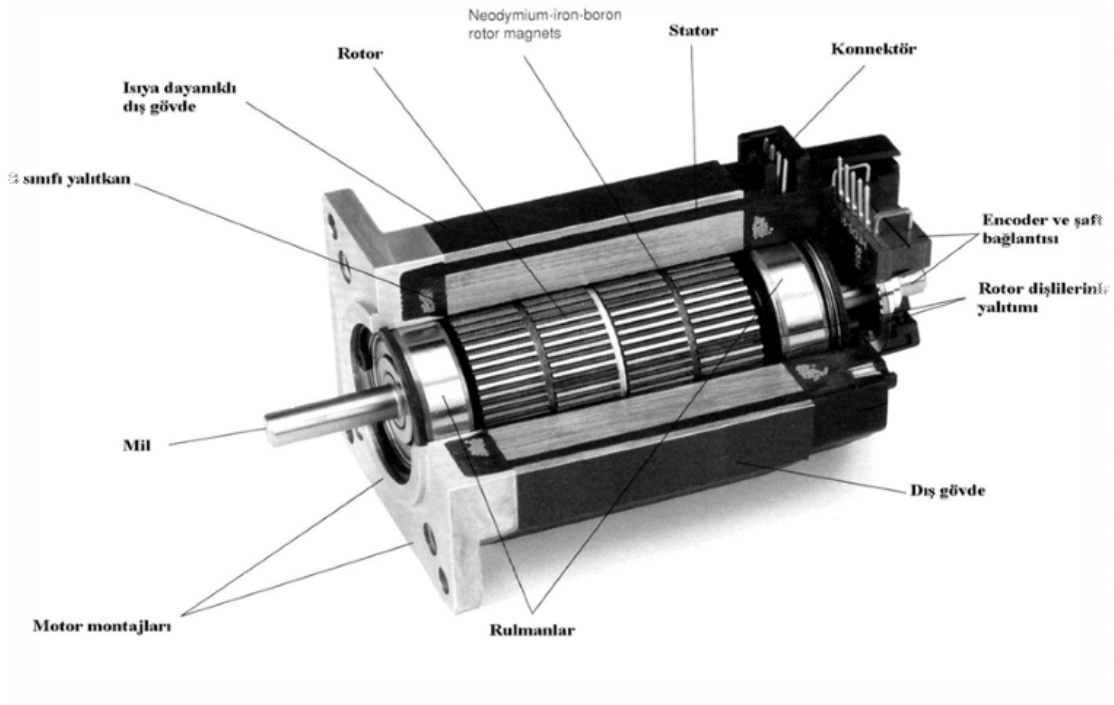
Şekil-26'da görüldüşü gibi stator diŞlilerinin arası 10 ve her kutupta üç diŞ, her faz üç kutuptan oluŞtuşu için bir fazda toplam 12 ve üç faz için toplam 36 kutup bulunmaktadır. Buna göre kutuplar arasındaki açı 360/36=10˚ olarak bulunur.

 (a)	Rotorun başlangıç pozisyonu
 (b)	Birinci faz enerjilenirse rotor diŞlileri 1. 4. 7 ve 10 numaralı stator diŞlilerinin karşısına gelir.
 (c)	***Birinci adım*** Üçüncü faz enerjilendişinde rotor diŞlileri 2,5,8 ve 11 numaralı stator diŞlilerinin karŞısına gelir. Rotor hareketi saat ibresi yönünde 7.50 dir
 (d)	***İkinci adım*** İkinci faz enerjilendişinde rotor diŞlileri 3,6,9 ve 12 numaralı stator diŞlilerinin karŞısına gelir (Rotor hareketi saat ibresi yönünde 7,50 dir. Toplam rotor hareketi 150 dir.
 (e)	***Üçüncü adım*** Birinci faz enerjilendişinde rotor diŞlileri 1,4,7 ve 10 numaralı stator diŞlilerinin karŞısına gelir. Rotor hareketi saat ibresi yönünde 7,50 dir. Toplam rotor hareketi ise 22,50 dir.

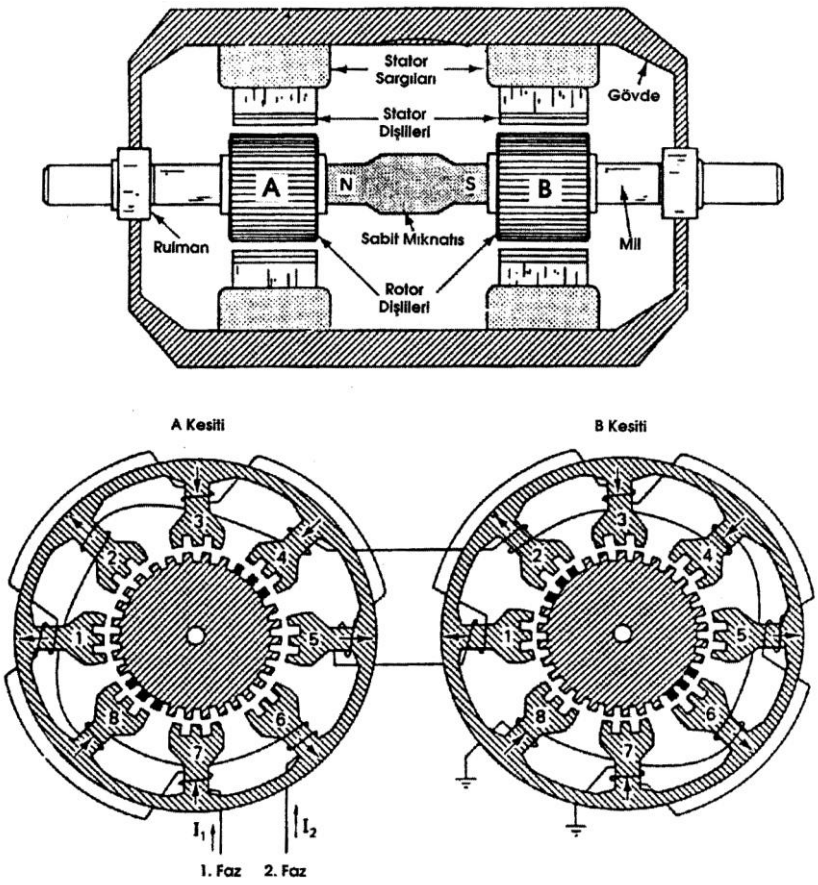
Şekil- 26. Tek parçalı deşişken relüktanslı adım motorun yapısı ve çalışma pozisyonu

Hybrid Adım Motorlar

Hybrid adım motorlar sabit mıknatıslı ve deşişken relüktanslı adım motorların birleştirilerek geliŞtirilmiŞ Şeklidir. Resim 1.2 "de hybrid adım motorlar görülmektedir. Resim 1.3"te hybrid adım motorun parçaları görülmektedir. Hybrid adım motorlarda rotor, sabit mıknatıslı olup çeşitli diŞli (kesit) sayısında yapılmaktadır. Ayrıca her bir dişli (kesit) üzerinde de çeşitli sayıda diŞler bulunmaktadır. Bu diŞlilerin arası diskler yardımıyla yalıtılmıŞtır. Resim 1.3 a"da dört diŞli (kesit) ve iki diŞli (kesit) adım motor rotorları görülmektedir. Hybrid adım motorlarda stator, çok parçalı deşişken relüktanslı tipindedir. Genel olarak stator kutbu 8 kadardır ve her bir kutup 2 – 8 arası diŞ sayısına sahiptir. Stator kutupları üzerine sargılar sarılmak suretiyle çeşitli kutup sayıları elde edilir. Resim 1.3 b"de boş stator ve sargıları görülmektedir.



Şekil- 27. Hybrid adım motorun yapısı.



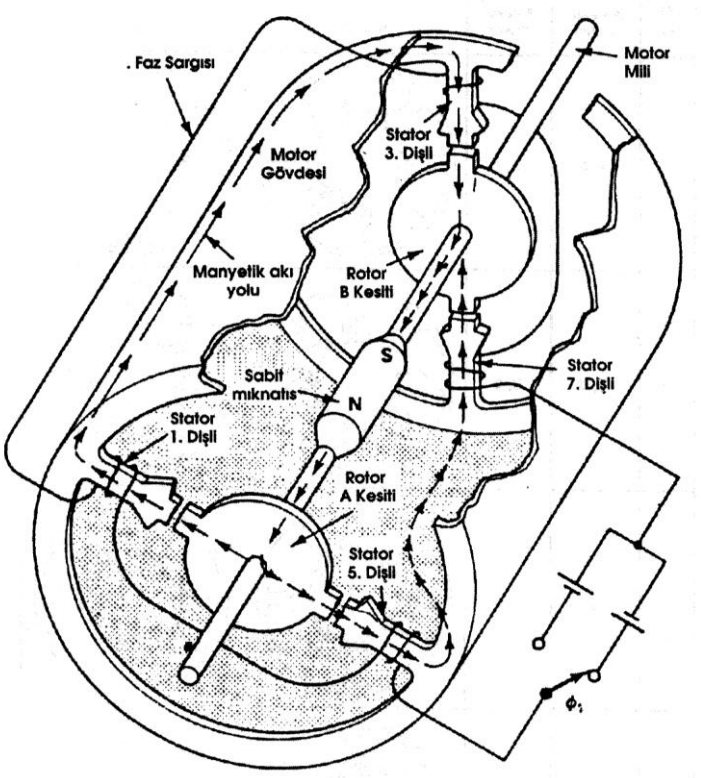
Şekil- 28. Hybrid adım motorun A – B kesitlerinin görünüŞü

Şafta (mile) paralel olarak kesiti yapılmış hybrid tipi adım motoru şekil-28'te verilmiştir. Şekil-28"de verilen adım motorun A ve B kesitlerinde rotor dişli sayısı 30, stator dişli

sayısı 24 ve adım açısı 3° dir. İki fazlı hybrid adım motorun, birinci faz 1,3,5,7 ve ikinci faz 2,4,6,8 kutuplarına yerleştirilir.

Çalışması: Şekil-29‟de gösterilen N ve S kutuplarından müteŞekkil sayılar sırasıyla enerjilenerek motor uyarılır. Saat ibresi yönü (CW) için faz uçları Şekil-1. 13‟te görüldüşü gibi 1+, 2-, 1-, 2+, 1+ Şeklinde beslenir. Birinci faz ve ikinci faz sargılarının enerjilenme sırası motorun dönüş yönünü ayarlar. Faz sargılarına 1+ düz gerilim, 1ise ters gerilim uygulandışını gösterir. Adım motorlar senkron çalıŞan makineler (rotor döner manyetik alanı izler) olup, her uyartımda bir manyetik hareket saşlanmaktadır. Söz konusu motorda, hareket uyartım kademesinden sonra ilk uyartım biçimine dönülerek sürdürülmektedir. Bilinen miktarda hareketin sürdürülmesi, bu andaki rotorun bir diŞ adımı kadar hareket etmesine başlıdır.

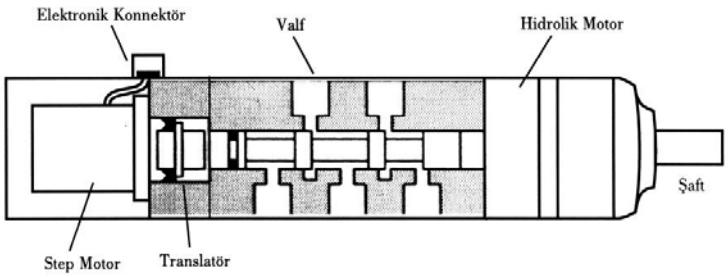
Şekil-29‟de verilen adım motorun birinci faz sargıları enerjilendişi zamanki manyetik akının takip ettişi yol Şekil-30‟te gösterilmiŞtir. Manyetik akının yolu; N den S‟ e doşrudur. N kutbundan çıkan akı, A kesitindeki 1 ve 5 numaralı kutup sargılarının olduşu kısımdan çıkar. B kesitindeki 3 ve 7 numaralı kutup sargılarından girerek S kutbuna ulaŞır. En fazla manyetik akının olduşu yol rotor ve stator diŞlilerindedir.



Şekil- 29. Hybrid adım motorlarda akım devresi

Hidrolik Adım Motorlar

Bir hidrolik motora ait servo valf ‟inin basınç giriŞ yolunu translatörlerle (dönebilir lineer çeviriciyle) kontrol eden adım motorlara hidrolik adım motor denir. Kısaca hidrolik motorun basınçlı yaş yolunu denetlemek suretiyle Şaftın hareketini ve yönünü tayin eden adım motorlara hidrolik adım motor denir. Hidrolik adım motorlara elektro–hidrolik adım motorlar da denilmektedir.



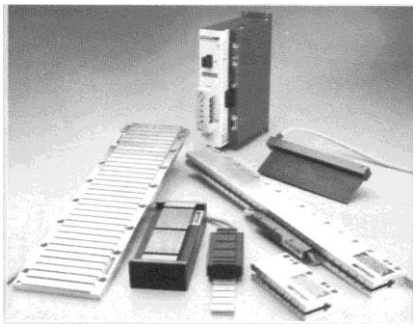
Şekil- 30. Hidrolik adım motor

Şekil-1.14‟te kesiti görülen hidrolik adım motor başlıca Şu parçalardan oluşmaktadır;

- Adım motoru
- Hidrolik motor
- Valf
- Translatör
- Elektronik konnektör

Lineer Adım Motorlar

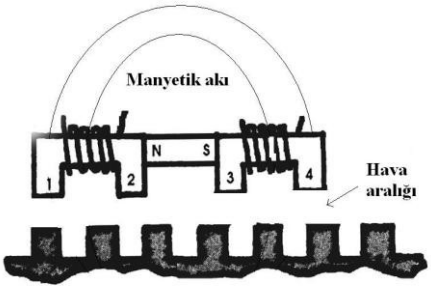
Mekanik hareketi dairesel bir hareket olmayıp yatay eksen (x veya y eksenleri) üzerinde hareket eden motorlara lineer motor denir. Yani lineer motorlar X ve Y yönlerinde veya X ve Y düzleminde herhangi bir vektör yönünde hareket ederler. Bu tür motorların tasarımı yapılırsa motor bir gövde üzerinde iki tane ortogonal elektromanyetik alanı içerir. Bu alanı tamamlamak için demir nüve kare Şeklinde yapılır. Böylece iki eksenli lineer adım motor oluŞturulur. Bu tip adım motorlara örnek olarak 1969 yılında Kaliforniya‟da gerçekleştirilen sawyer adım motoru gösterilebilir. Resim 1. 5‟te lineer adım motorlar ve sürücüleri görülmektedir.



Şekil- 31. Lineer adım motorlar ve sürücüleri

Bu motor iki ana mekanik bileŞenden oluŞur. Birinci mekanik bileŞen, gücü oluŞturan hareketli armatürdür. Armatürün statora sabitlendişi (demir nüve) kısım ikinci bileşendir. Armatür ve stator arasında sabit bir mil yataşa (hava aralışı) olup, kapalı geometrik Şekilde dönmeye izin verir. Yükü harekete geçirmek, demir nüve uzunluşuna başlı olan güçle deşişir. Bu deşişim bir yükü getiren motorun rotor hareketine benzemez. Ayrıca güç iletimi için mekanik üstünlüklere de sahip deşildir.

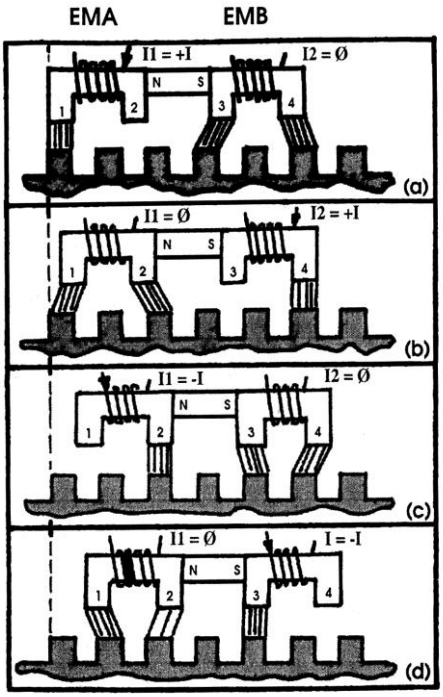
Şekil-1.15‟te gösterildişi gibi lineer adım motor, sabit mıknatıs (PM) ve dört kutuplu iki elektromıknatıs (EM) oluşur.



Şekil- 32. İki fazlı lineer adım motorun prensip Şeması

Çalışması

Manyetik alanın alt ve demir nüvenin üst noktaları arasındaki hava aralışı oluŞur. Kutup yüzeylerine sawyer motorda olduşu gibi oluklar açılmıŞtır. Oluklar, örnekteki demir nüvenin Şeklinde yapılırlar. Ayrıca oluklar arasındaki boŞluklar manyetik olmayan maddeler tarafından doldurulmuŞ olup ve bu düz yüzeyler manyetik alanın alt ve demir nüvenin üst noktasındaki hava aralışını oluŞturur. Manyetik alan içerisindeki küçük deliklerden hava basıncı saşlanmasıyla bu iŞ gerçekleştirilebilir. Bu hava aralışında ihmal edilmeyecek bir hareketli sürtünme yüzeyi oluŞturur.



Şekil- 33. İki fazlı lineer adım motorun hareketi

Sabit mıknatıs, demir nüve ve manyetik alanın etkisinin olmadışı kısmı birlikte etkiler. (Bu kısma hava aralışı dahil deşildir.) Buna başlı olarak demir nüvenin üzerindeki manyetik alanı alta veya üste hareket ettirmek mümkündür. Akım olmadışı durumda PM akıŞı hava aralışındaki Şekli demir nüve ve EM akıŞı EM‟nin iki kutbunda da eşit olur. Manyetik kutuplar yaklaŞık olarak aynı relüktansa sahip olduklarından PM akıŞı EM‟nin iki kutbunda da eşit olur (Şekil-1.16 a 3. ve 4. kutuplar). Eşer akım elektromıknatıslar tarafından anahtarlanırsa bu durumda deşişim olur. Genel olarak sabit mıknatıs tarafından oluŞturulan akım manyetik alan sargılarında üretilen akıma

yaklaŞık olarak eşittir. Yani akım deşiştişinde manyetik akı maksimumdan sıfıra kadar değişir.

Elektromanyetik alan ile demir nüve diŞleri arasındaki bu deşişim demir nüveye paralel, diŞlere ise dik Şekildedir. EM diŞleri bir kutuptan dişerine sıralandışı için PM akışı kutup diŞlerinin birleştişi yerde sabit mıknatıs tarafında deşiştirilir. Sonuç olarak böyle teşetsel kuvvet, elektromanyetik alan ve demir nüve boyunca hareket eder ayrıca elektromanyetik alan ile demir nüveyi birbirine doşru çeken ve hava aralışı için bir ön yük oluŞturan kuvvet vardır. Şekil-33 yukarıda anlatılan iŞlemleri göstermektedir. Her bir Şekilde akım ve manyetik akının yönleri oklarla gösterilmiŞtir. Eşer elektromıknatısta manyetik alan oluŞursa maksimum akı yoşunluşu ikinci kutupta aynı hızda oluŞur ve bu Şekil- a‟da gösterilmiŞtir.

Elektromanyetik mıknatıs enerjilenmeyip (EMA), EMB enerjilenirse maksimum akı yoşunluşu 3. kutupta minimum yoşunluk ise 4. kutupta oluŞur. 3. Kutuptaki bu kuvvet demir nüvenin sağ taraftaki kutup ile aynı sıraya gelirken diŞin saşa hareketi dörtte bir olarak gerçekleşir. Burada motor ve elektromanyetik alan iliŞkisi Şekil-1.16 b‟de gösterilmiştir. Eşer EMB enerjilendirmez EMA enerjilendirilirse hareket tekrar saşa doşru olur. Bu durumda birinci kutbun akı yoşunluşu maksimum ikincininki minimumdur. (3. ve 4. kutuplara ise PM uygulanmıŞtır.) Bu andaki EM alanı Şekil-33‟de gösterilmiştir.

Sonuç olarak EMA enerjilendirilmeyip EMB enerjilendirilirse 4 kutup maksimum akım 3. kutupta ise minimum akı yoşunluşu olur. (Bu durumda 1. ve 2. kutuplara PM uygulanmıŞtır.) Bir devri tamamlamak için Şekil-33‟de gösterildişi gibi EMA tekrar enerjilendirilir ve sistem hareketi demir nüvenin bir dişi kadar olur. Bir peryot boyunca akımın frekansı EM alanın hareket hızıyla belirlenir.

Elektromanyetik alanın demir nüve ile olan bu pozisyonlarında akımın her peryot boyunca yukarıda tanımlandışı gibi deşişmesi bu iki arasındaki iliŞkiyi açıkça gösterir. Bu durumda lineer adım motorlar kutup diŞleri tarafından bir full adım rezolüsyonuna sahiptir. Tipik bir örnek olarak bu deşer 0.04‟tür. Yani Şekil-1.16a‟da gösterilen sıralı hareket, her dörtte bir hareket için bu deşer 0,01‟dır. Bu hareketler bazen kardinal adım olarak adlandırılır. Adım basamakları arasında daha iyi rezolüsyon elde etmek için full-adım modunda çalışmada bu dörtte bir hareketler arasında bir akım deşeri bulamamak mümkündür. Daha öncede anlattışımız gibi lineer adım motorlar direkt sürücülü motorlardır. Direkt sürücülü, kontrol rezolüsyonu ve yükü sürmek için uygulanan kuvvet motorun yeteneşi olarak tanımlanır. Yani herhangi bir uygulama için gerekli diŞli rezolüsyonu micro adım motor kontrolü için istenilen rezolüsyonda kullanılması daha iyidir. Ayrıca motor sürücü devresi için çizilen hız-kuvvet eşrisi motorun işlem hızı üzerindeki gerekli kuvvetleri üretebilecek durumda olmalıdır. Lineer adım motorlarda yukarıda anlatılan aynı özellikler görülür ve senkronize kayıpları adım motorun rotorunda olduşu gibidir. Ama bu tür motorların kontrolü iki karakteristik açısından daha zordur. Bunlardan birincisi devrenin kendisinde olan “spring”dir. Motor armatürü iki diŞli aralışı, genişlişe kadar kısma oturur. Bundan dolayı, bu haricî kuvvetlerin giderilmesi gerekir. Eşer armatür hareketini engelleyen bu kuvvetlerin etkisi giderilmezse motorun senkronize kayıpları çok olacaktır.

Micro adım motorların kontrolünü zorlaŞtıran ikinci karakteristik ise, hava aralışı yüzeyinde armatür rezonansı oluŞturan karakteristiktir. Yani “spring” kütlesinin sönümünü saşlayan armatür ve engelleyici kuvvet tarafından oluŞturulan bir etken

vardır. Bu Şart motorun uyarılması için gereken akım frekansı rezonans frekansına yakındır. Yani hareket boyunca istenmeyen karıŞıklıktan dolayı motorun rezonans frekansına gelmesi uzun sürebilir.

Lineer adım motorların en büyük üstünlükleri:

- Yüksek güvenlişi bulunmaktadır.
- Gerekli işlemleri yerine getirmek için az ve basit devre elemanlarından oluşur.
- Uzun mesafeler arasında yüksek hızla hareket ederken, yüksek hassasiyete sahip olmalarıdır.
- Hava aralışı hemen hemen manyetik alandan başımsız olduşu için hiç bakım gerektirmezler.

Bu tür motorların lineer sürücü katları fiyatı sıkça bilinen de servomotor ve geribesleme katına göre daha yüksektir. Bu tür motorların fiyat mahzurları yanında, gerekli elektronik sürücüler osilasyonu ve senkronize kayıtları azaltır. Ayrıca kuvvet azalması dahil hız artıŞını saşlar. Lineer adım motoru ticari endüstriyel robotlarda kullanılmazlar. Bununla birlikte bunların maliyeti düŞürülürse bu tür direkt sürücü motorlar minimum eleman kullanarak güvenilir uygulama alanları bulunabilir.

Adım Motorların ÇalıŞtırılma Şekilleri ve Teknikleri

Adım motorlar çalışmalarında olduşu gibi uyartımda da fazla esneklişe sahiptirler. Bu esneklik, maksimum çıkıŞ güç, maksimum etki, maksimum tepki ve minimum giriŞ gücünde olmaktadır.

Başla-Dur Adımlama Oranı

Motor sargılarının sadece birisinin uyarıldışı uyartım cinsine tek-faz (Single Coil) uyartım denir. Uyartım CW (saat yönü) için 1000,0100,0010,0001 Şeklinde CCW (saat yönünün tersi) için 0001,0010,0100,1000 Şeklinde olmalıdır.

Adım	Faz 1	Faz 2	Faz 3	Faz 4
1	1	0	0	**0**
2	0	1	0	**0**
3	0	0	1	**0**
4	**0**	**0**	**0**	**1**

Tablo 1.2: Tek faz uyartım tablosu

Düzgün Hız

Motor sargılarının ikisinin sıra ile aynı anda uyarıltışı uyartım Şekline denir. İki faz uyartımda rotorun geçici durum tepkisi tek-faz uyartıma göre daha hızlıdır ancak burada güç kaynaşından çekilen güç iki katına çıkmıştır.

Adım	Faz 1	Faz 2	Faz 3	Faz 4
1	1	0	0	**1**
2	1	1	0	**0**
3	0	1	1	**0**
4	**0**	**0**	**1**	**1**

Tablo-3. İki faz uyartım tablosu

Rampalama

Bu uyartım modunda tek faz ve iki faz ard arda uygulanır. Burada rotor herbir uyartım sinyali için yarım adımlık bir hareket yapmaktadır. Bu uyartım modu sayesinde, örneşin fabrika çıkıŞı 2 derece olan bir motorun adım açısını 1 dereceye düŞürmüŞ oluruz.

Adım	Faz 1	Faz 2	Faz 3	Faz 4
1	1	0	0	**0**
2	1	1	0	**0**
3	0	1	0	**0**
4	0	1	1	**0**
5	0	0	1	**0**
6	0	0	1	**1**
7	0	0	0	**1**
8	**1**	**0**	**0**	**1**

Tablo-4. Yarım adım uyartım tablosu

Diğer Uyartım Yöntemleri

Adım motorların uyartım metotları faz sayısına göre Şöyle sıralanabilir.
İki fazlı motorlarda;
- İki faz uyartım modu
- İki faz düzeltme modu

Üç fazlı motorlarda;
- Üç faz uyartım modu
- Üç faz düzeltme modu

Dört fazlı motorlarda ya da orta ucu (müŞterek ucu) kullanılan iki fazlı motorlarda;
- Dört faz uyartım modu
- Dört faz düzeltme modu kullanılır.

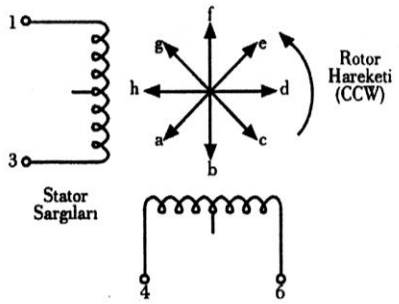
Not: Tablolada adım motorun sargılarına uygulanacak gerilimin yönüne göre rotorun hareketi,

CW: Saat ibresi yönünde

CCW : Saat ibresi tersi yönünde döndüşünü ifade etmektedir.

İki Fazlı Motorların Çalışma Şekilleri

Bazı adım motorlarda Şekil-1. 17‟de görüldüşü gibi iki faz sargısı (stator sargısı) bulunur. Şekil-1. 17‟de her iki sargının da (fazın) orta merkez ucu olduşuna dikkat ediniz. Bu motor her bir uyartım sargısının yarısı bir faz gibi uyartımla çalıŞtırılacak olursa dört fazlı bir motor olarak çalıŞabilir.



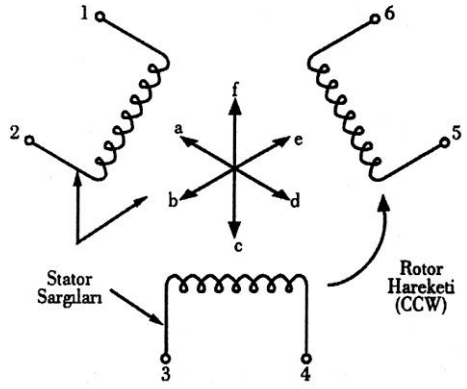
Modu	Enerjili Sargı	Pozisyon	Hareket
İki faz uyartımda	3-1	f	Index
	6-4	h	CCW
	1-3	b	CCW
	4-6	d	CCW
İki fazlı düzeltilmiş uyartımda	3-1,6-4	g	Index
	1-3,6-4	a	CCW
	1-3,4-6	c	CCW
	3-1,4-6	e	CCW

Şekil- 34.İki fazlı adım motor sargıları ve çalışma modları

İki faz uyartım modu: Bu çalışma Şeklinde sargılara gerilim, dıŞ uçlardan ve yönü deşiştirilerek uygulanılır. Bunun sonucunda rotor, Şekil-1.17‟deki tabloda verildişi aralıklarda ve yönde dönecektir. İki az düzeltme modu: Bu çalışma Şeklinde yine orta uçlar kullanılmaz. Ancak her iki sargıda uygun fazlı gerilimler uygulandışında Şekil-1.17‟de verilen pozisyon ve yönde dönecektir.

Üç Fazlı Motorların Çalışma Şekilleri

Üç fazlı adım motorlar başımsız üç sargıdan meydana gelir. Üç fazlı motorun, uyartım ve düzeltme modunda saat ibresinin tersi yönünde 60°'lik adımlarla hareket etmesi için Şekil-35'te verilen tabloda belirtilen sargılara sırayla gerilim uygulanmalıdır.



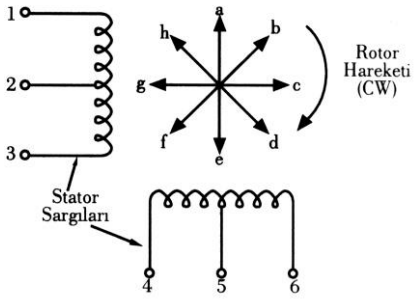
Modu	Enerjili Sargı	Pozisyon	Hareket
Üç faz uyartımda	2-1	a	CCW
	3-4	c	CCW
	5-6	e	CCW
Üç fazlı düzeltilmiş uyartımda	2-1, 3-4	b	CCW
	3-4, 5-6	d	CCW
	5-6, 2-1	f	CCW

Şekil- 35.Üç fazlı adım motor sargıları ve çalışma modları

Bu metot adım motorlarda çok kullanılan bir sistemdir. Bu çalışma Şeklinde üç faz, Şekil-1. 18‟de verildişi gibi sırayla polarılır. Bunun sonucu rotor tabloda görüldüşü pozisyon ve yönde hareket eder. Üç faz düzeltme modu: Bu modda üç fazdan yan yana olan ikisi aynı anda polarılır. Bu sargılara gerilim uygulanır. Adım adım gerçekleŞen dönme pozisyonu ve yönü aynı tabloda gösterilmiştir.

Dört Fazlı Motorların Çalışma Şekilleri

Dört fazlı motorlar başımsız dört sargıdan meydana gelir. Ancak daha önceden açıklandışı gibi müŞterek uçlu iki sargıya sahip iki fazlı motor, dört fazlı motor gibi çalıŞtırılabilir. Bu Şekilde dört fazlı bir motor gibi çalıŞtırılan adım motor Şekil-1. 19"da verilmiŞtir. Dört fazlı motorun, uyartım ve düzeltme modunda saat ibresi yönünde 90 □ adımlarla hareket etmesi için Şekil-1. 19"da verilen tabloda belirtilen sargılara sırayla gerilim uygulanmalıdır.

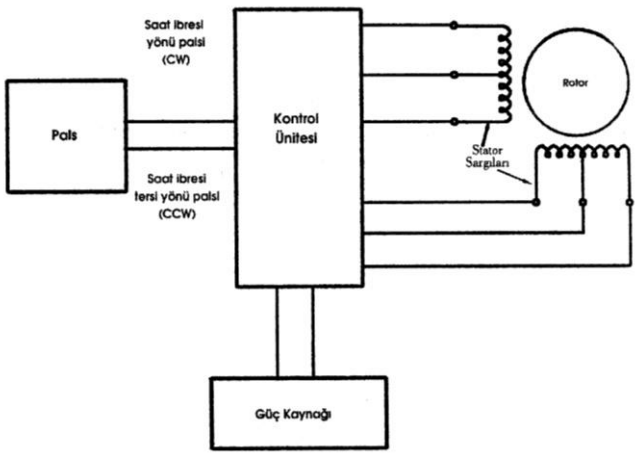


Modu	Enerjili Sargı	Pozisyon	Hareket
Dört faz uyartımda	1-3	a	Index
	4-6	c	CW
	3-1	e	CW
	6-4	g	CW
Dört fazlı düzeltilmiş uyartımda	1-2, 4-5	b	Index
	3-2, 4-5	d	CW
	3-2, 6-5	f	CW
	1-2, 6-5	h	CW

Şekil- 36.Dört fazlı adım motor sargıları ve çalışma modları

Adım Motor Sürücü Devreleri Yapısı ve Çalışması

Adım motorları istenilen yönde ve hızda çalıŞtırmak istendişinde sargılarına belli bir sırada darbeler uygulanmalıdır. Adım motorun kaç adım atacaşı uygulanan darbelere başlıdır. Fazlara uygulanacak darbeler (palsler-gerilimler) basit olarak bir anahtarlama sistemi ile yapılabilir. Bu iŞlemi yapan devrelere sürücü devresi veya kontrolor denir. Günümüzde elektronik devreler ile bu iŞlem çok kolay bir Şekilde yapılmaktadır. Adım motorların ve kullanılacak yerin özellişine göre hazırlanmıŞ mikroiŞlemci kontrollü sürücü kartları mevcuttur. Bu kartlar sayesinde adım motorların istenilen hızda ve istenilen hassasiyette çalıştırmak mümkündür. Bir adım motor sürücü devresinin blok diyagramı Şekil-de verilmiştir.



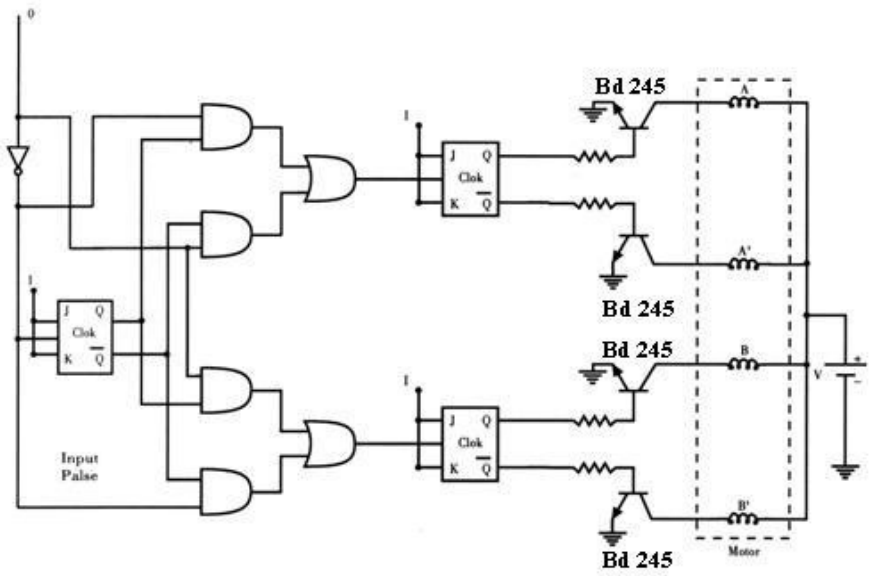
Şekil- 37.Adım motor sürücü devresinin blok diyagramı

Adım motorların sürülebilmesi için 2 temel noktaya dikkat etmek gerekmektedir. Bunlardan birincisi motorun başlanacaşı sürücü devresinin olmasıdır. İkincisi ise bu sürücü devresi yardımıyla motorun doşru sargılarına gerekli tetiklemeleri gönderebilmektir. Sürücü devresini hazır alabileceşimiz gibi amatör uygulamalar için

ileriki konularda anlatılan devreler gibi bir devreyi de kendimiz yapabiliriz. Sürücüyü tetiklemek için elektromekanik anahtarlar kullanabileceşimiz gibi bilgisayarın seri veya paralel portunu uygun bir yazılımla kullanabiliriz. Ayrıca günümüzde sanayide kullanılan adım motorlar için mikroiŞlemci kontrollü sürücüler ve bu iŞler için özel olarak tasarlanmıŞ PLC‟leride bulunmaktadır.

Adım Motor Sürücü Devreleri ve Yapıları

Adım motor sürücü devreleri genel olarak 3 temel ilke üzerine yapılır. Bu temel sürücü mantıkları adım motorları istenilen hız ve torkda çalışmasını saşlar. Bu sürücü mantıklarını kısaca Şöyle ifade edebiliriz. L/R Sürücüsü: Motor öngörülen voltaj ve akım deşerlerinde çalıŞtırılır. Bu durumda motor bobinlerindeki indüktif etkiden dolayı ufak bir hız artıŞında motor öngörülen akıma ulaŞamayacaşı için düŞük hızlar dıŞında motor verimli bir Şekilde sürülemeyecektir. L/nR Sürücüsü: Akım artışıda geçerli olan zaman sabitini (t=L/R) düŞürmek için motor bobinlerine seri direnç başlanarak yapılır. Bu durumda motor öngörülen voltajın n katı deşerde çalıŞtırılır. Bu sayede motorda belirgin bir hız artıŞı yaŞanır. Ancak başlanan seri direnç üstünden yüksek akım geçeceşi için bu devrelerde gereksiz güç tüketimi yaŞanır.



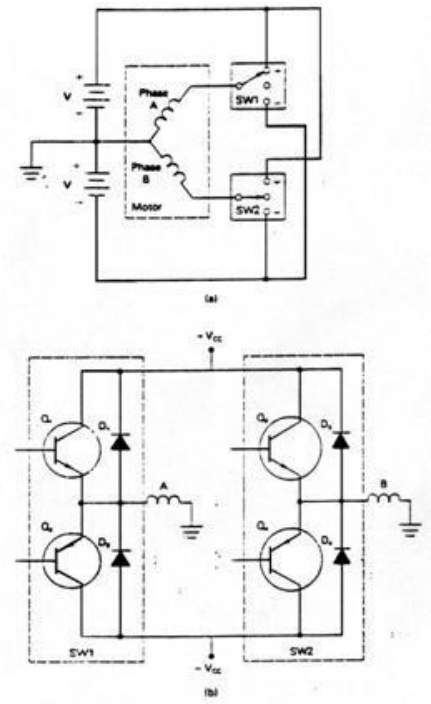
Şekil- 38.Basit bir adım motor sürücü prensip devresi

Chopper Sürücüsü: Motor öngörülen voltaj deşerinden 5-20 kat fazla bir voltajla beslenir. Akımın yükselme hızı dişer sürücülere göre oldukça çabuktur. Ancak yükselen akım belirli bir deşerde sınırlanmazsa motor gereşinden fazla akım çekeceşi için yanacaktır. Akım sınırlama mekanizması chopper sürücüsünün temelini oluŞturur. Motor bobinine seri başlı küçük bir sense direnci üzerindeki voltaj bir komparatör ile karŞılaŞtırılarak, bobine giden akım ayarlanır. Sürücü PWM (Pulse Width Modulation) mantışında çalıŞır. Kaynak zaman içinde açılıp kapandışı için güç tüketimi minumum düzeydedir. Yüksek voltajla beslemeden dolayı yüksek hızlarda tam tork ile çalıŞılabilir.

Bir adım motor için basit türden bir sürücü devresi, dıŞ resistans olmaksızın ve bir tek güç kaynaşı ile gerçekleştirilebilir. Şekil-39'de adım motoru sürmek için kullanılabilecek bir prensip devre gösterilmiŞtir. Devre darbe Şeklindeki bir iŞaretlerle sürülmektedir.

Devrede; lojik kapılar, flip-floplar ve anahtarlama amaçlı transistörler kullanılmıŞtır. Stator sargılarının indüktans ve rezistans içermesinden dolayı Is akımı, sargının L/R zaman sabitiyle ekpotansiyel olarak yükselir.

Şekil-39'da görülen motorun full-adım modu esnasında çift uçlu güç kaynaşı ile sürülebilir. Bunu saşlamak içinde sw1 ve sw2 gibi iki tane anahtarın olması gerekir. Bahsedilen Şekilde A fazının yükseldişini, B fazının ise baŞta kaldışını söyleyebiliriz. Bu devrelerin senkronize çalışmalarını saşlayan devre Şekil-39'da gösterilmiŞtir. Bu devrede kullanılan diyotlar güç transistörlerini, gerilim taŞmalarını, ters polarlanmalara karŞı korumak amacıyla kullanılmıŞ hızlı diyotlardır. Bu koruma sistemi olmaz ise anahtarlama esnasında armatürün kollektör-emiter arasına uygulanacak aŞırı gerilim sonucu transistör yanabilir.

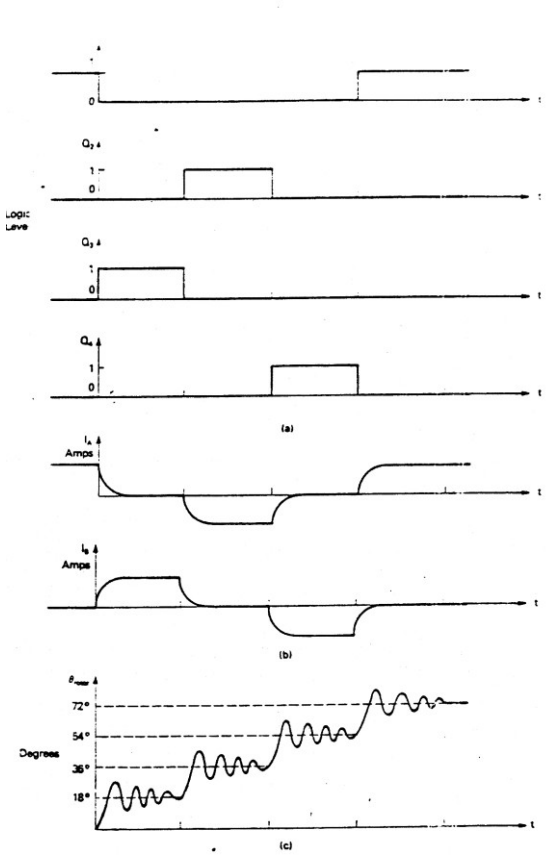


Q1,Q2,Q3,Q4=Bd 245

D1, D2, D3, D4=IN 4001

Şekil- 39. a) İki fazlı adım motorun çift kutbunun anahtarlamalı sürücüsü, b) a'yı gerçekleştiren prensip devre

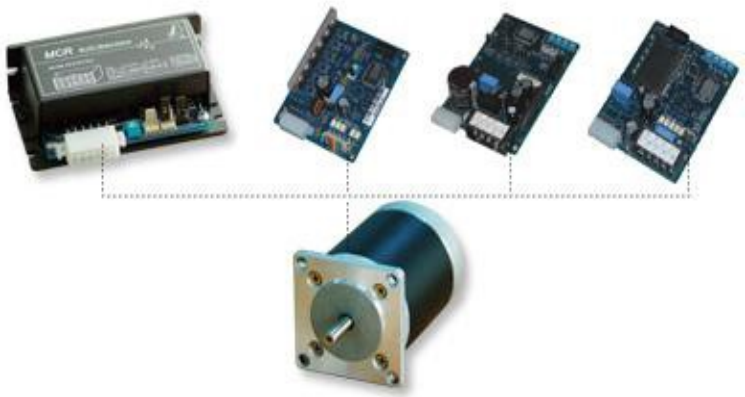
Toplam olarak dört rotor devri oluŞturmak için transistöre uygulanabilecek lojik seviyeli iŞaretlerin Şekli Şekil-40te gösterilmiŞtir. Bu Şekilde her bir adımın aldışı lojik iŞaretlerinin oluŞturacaşı hızlandırma ya da yavaŞlama, oluŞacak yük farklılışına raşmen aynıdır. Genel çalışmalarda, bir yükün hareket miktarının ihtiyacı olduşu adım sayısı mikroiŞlemciler kullanılarak gerçekleştirilir. MikroiŞlemciler robot eklemlerinin hareketinin hassas olması için kullanılır. Bu durumda iŞlemci, yönü, adım zamanını ve sayısını en uygun hareketi saşlayacak Şekilde lojik seviyede iŞaretlerle karar bölümüne iletir. Bu iŞlem, adım sayısına uygun, ardıŞık anahtarlamanın olmasıyla, istenen hareketin yapılmasını saşlar. Bu iŞlemleri açık-çevrim eklem kontrollü Şeklinde düŞünüp deşerlendirme ona göre yapılmalıdır. Daha önceden belirtildişi gibi adım motorun servo motora üstünlüşü açık- çevrim kontrolünde kullanılabilirlişidir.



Şekil- 40.a) Transistörün lojik sinyali b) Motor faz akımı c) Şekil-2.2 a'da olan motorun rotor hareketi

Sürücü devrelerin genel amacının, akımının düzenlenmesi ve sınırlanmasını saşlamak olduşundan bahsetmiŞtik. Tepki zamanı kısaltılmak istenmesi büyük bir akım deşeri getirir ki, bu da istenmeyen bir durumdur. Akımı sınırlamanın en basit yolu kaynaşa seri bir dıŞ rezistans yerleştirmektir. Seri rezistans sınırlama metodunun önemli bir dezavantajı vardır. Örneşin, dıŞ rezistans motor rezistansın 4 katı ise, gücün % 80"i motorun dıŞında harcanmaktadır. Bu ise düŞük verimli bir sisteme sebep olur. Akım sınırlamanın dişer bir yolu copper teknişidir. Burada yüksek gerilim, motorun aŞırı uyarımı için tekrar kullanılır. Fakat akımın belli bir limitin üzerine çıkmaması için gerilim on ve off Şeklinde peryodik olarak anahtarlanır. Anahtarlama motor sargısındaki ortalama akımı yükseltir ve sargı enerjisi bitene kadar devam eder. Buradaki avantaj yüksek verim elde edilmesidir, fakat sürücü devresi daha komplekstir.

Dişer bir metot da dual-voltaj (ikili gerilim) teknişidir. Ġsminden de anlaŞılacaşı gibi iki kaynak kullanılır. BaŞlangıçta motoru uyarmak için yüksek bir gerilim uygulanır. Akım belli bir deşere ulaŞtışında yüksek gerilim anahtarlaması, düŞük gerilim anahtarlamasına dönüŞür ve bu anda akım mevcut deşerini muhafaza eder. Burada verim yüksek olmasınakarŞılık, sürücü devre karmaŞıktır ve iki güç kaynak gerektirdişinden maliyet yüksek olur.

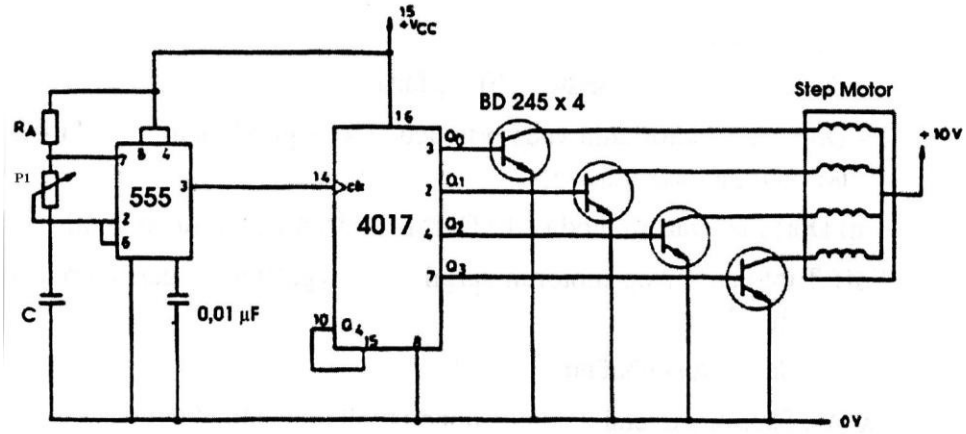


Şekil- 41.Adım motorlar – sürücüler ve sürücü kartları.

Adım Motor Sürücü Devreleri Çeşitleri

555 Osilatör Entegresi ve 4017 Sayıcı Enteşresi İle Yapılan Sürücü Devresi

555 ve 4017 sayıcı entegresi kullanarak yapılmıŞ baŞka bir sürücü devresi de Şekil-42'de verilmiŞtir. Bu devrede 555 osilatör olarak kullanılmıŞtır. P1 potansiyometresi yardımıyla üretilen sinyalin frekansı deşiştirilmekte bu da 4017'nin çıkıŞlarındaki sayma sürelerini deşiştrimektedir. 4017 gelen saat sinyalinin hızına göre çıkıŞlarını sırasıyla deşiştirir. ÇıkıŞlara başlı olan transistörler iletime geçerek sargılara enerjiyi vermiŞ olurlar. ÇıkıŞlar sırasıyla iletime geçeceşi için adım motor saat sinyali geldişi müddetçe dönecektir.

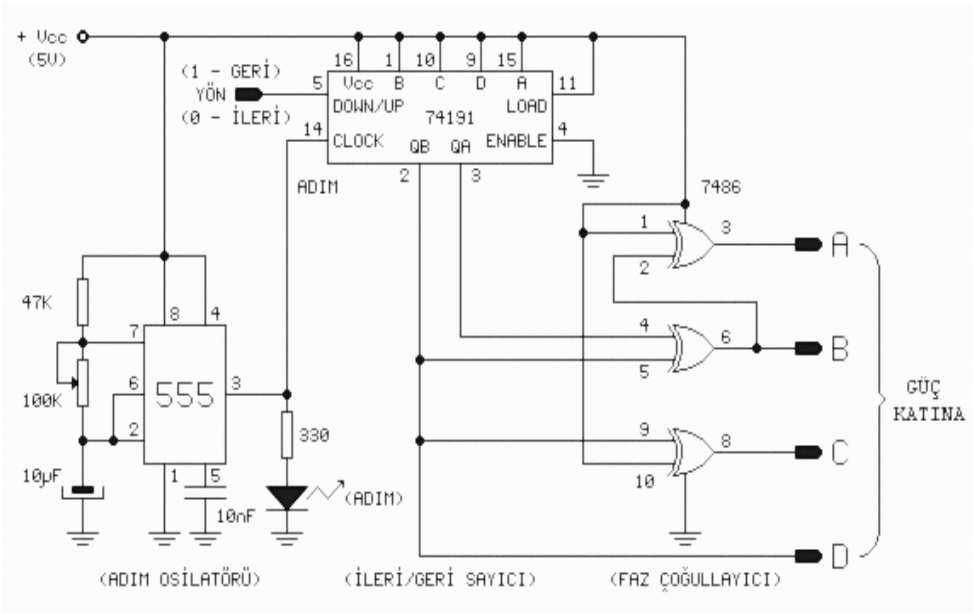


Şekil- 42.555 ve 4017'li sürücü devresi.

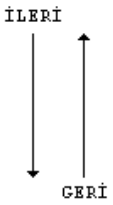
555 Osilatör 74191 Sayıcı Entegresi ile Yapılan Sürücü Devresi

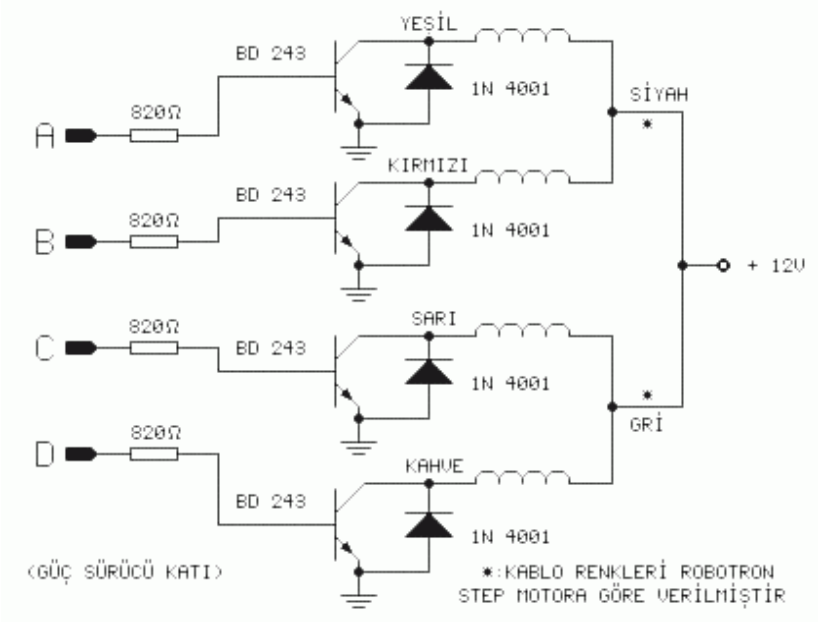
Şekil-43'te; adım osilatörü, sayıcı ve faz çoşullayıcıdan oluŞan adım motor kontrol devresi görülmektedir. 555 adım osilatörü adım motor için gerekli olan adım darbelerini üretir. Saat darbesinin frekansı düŞük ise motorun dönüŞü yavaŞ, frekans yüksek ise motorun dönüŞü hızlıdır. 74191 sayıcısı motorun ileri-geri yönde dönmesini saşlayacak sinyali üretir. 7486 ile yapılan faz çoşullayıcı, sayıcının ürettişi sinyali motorun 4 sargısı için çoşullar. Şekil-2.5'te kontrol devresi çıkıŞına başlanan güç

sürücü kat görülmektedir. Bu devre, adım motorun sargıları için gerekli olan sinyalin akımını arttırır.



QB	QA	A	B	C	D
1	1	1	0	0	1
0	0	1	0	1	0
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	0	0	1



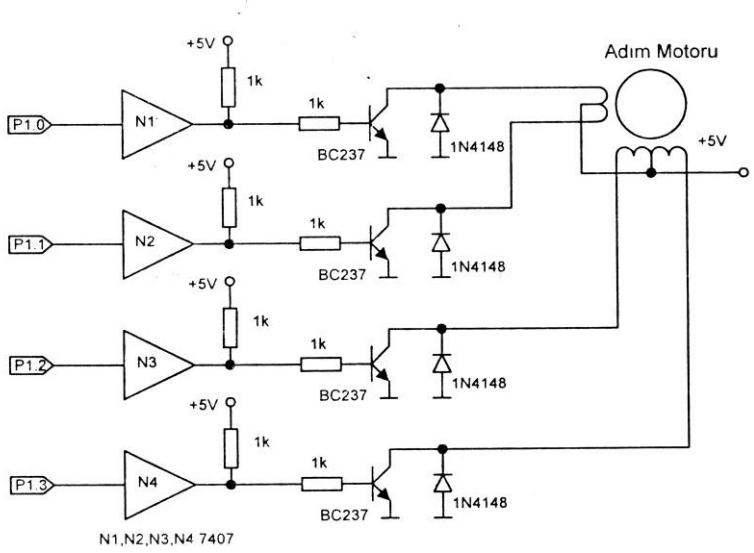


Şekil- 43.555 ve 74191 ile yapılan sürücü devresi

8051 Mikrodenetleyicisi ile Yapılan Sürücü Devresi

Şekil-46'da 8051 mikrodenetleyici ile kontrol edilebilen beş uca sahip bir adım motorun

devresi görülmektedir. Hazırlanacak program ile mikrodenetleyici yardımıyla adım motorunun denetimi yapılabilir. Adım motora gerekli faz iŞaretleri için 8051‟in port uçlarından 4 tanesi kullanılmıŞtır. Bu Şekilde her adımda adım motor sargılarından biri aktif edilerek adım motorun istenilen yönde dönmesi saşlanmıŞtır. Hazırlanacak bir bilgisayar programı ile kullanıcıdan adım motorun parametreleri istenir. Girilen parametrelere göre PC‟nin paralel portundan adım motor için gerekli palsler sürücü devresine uygulanır.



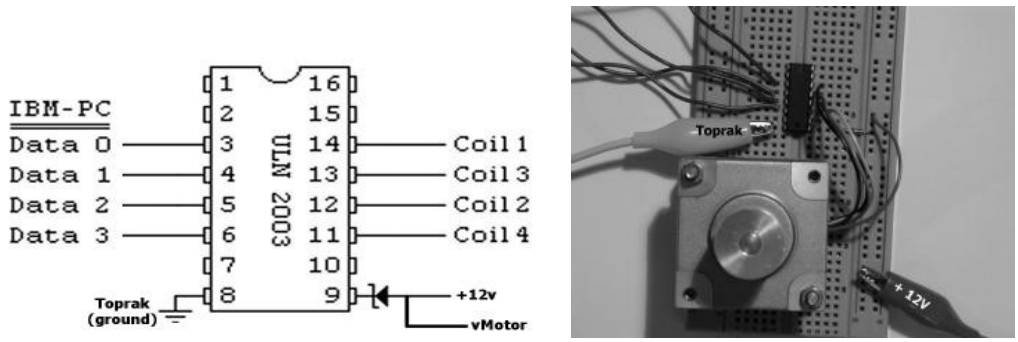
Şekil- 44.8051 Arayüz entegresi ile yapılan bilgisayar kontrollü sürücü devresi

Sürücü Devresi Yapımı

Yukarıdaki her iki devrede de adım motorun sargılarının enerjilenmesi için transistörler kullanılmıŞtır. Ama aŞaşıdaki devrede Darlington başlı tümleşik devre kullanılarak (ULN 2003) kolaylık saşlanmıŞtır. Sürücü devresi olarak kullanılan ULN2003 içerisinde 7 adet NPN transistör ve dahili diyod barındırdırmaktadır. Hâliyle bizi transistör bacaklarıyla uşraŞmaktan kurtarmaktadır. Kullanımı ise oldukça kolaydır. Devre Şemasından da anlaŞılabileceşi gibi 9 numaralı bacaşına +12 Volt ve 8 numaralı bacaşına da toprak (ground) uyguluyoruz. Daha sonra 3 ve

6 numaralı bacaklara da paralel portun DATA pinlerinden gelen +5 Voltluk deşerleri uygulayacaşız. Bu sayede örneşin 3 numaralı bacaşa +5 Volt (lojik voltaj) uyguladışımızda 14 numaralı bacak toprak olacaktır. Aynı Şekilde sırayla 4 için 13, 5 için 12, 6 için ise 11 numaralı bacaklar toprak olacaktır. HerŞeyden önce bir adım motora ihtiyacımız var. ĠŞimize en çok yarayacak olan adım motorunu eski 5 ¼ disket sürücülerinden kolayca sökebilirsiniz. Tabi bundan önce parçalayabilecek bir sürücü bulabilmeniz gerekli. Eşer bulamıyorsanız adım motor için sanırım biraz elektronikçi dolaŞmanız gerekecektir. Bulacaşınız adım motoru 4,5 ya da 6 kablolu olabilir. Bu kablolar avometre ile ölçerek sargıların uçlarını tespit etmeniz gerekmektedir. Uçları tespit etmek için Şu yol takip edilmelidir. Avometrenin X1 kademesinde uçları kendi aralarında ölçeriz. Kendi aralarında devre gösteren uçları ayırırız. Kendi aralarında devre gösteren uçlar aynı fazın uçlarıdır. Adım motor 4 uçlu ise fazlar ayrı ayrıdır. 5 uçlu ise ucun birisi ortak uçtur. Dişerleri faz uçlarıdır. 6 uçlu ise her iki fazın bir ortak ucu vardır. Ortak uç dişer iki uca göre daha az direnç gösteririr (yarısı kadar).

5 kablolu adım motorunun kablolarından bir tanesi vMotor dedişimiz ortak kablodur. Önemli olan bu kablonun hangisi olduşunu bulmaktır. Bunun için yukarıda anlatılan yöntem kullanılır. Biraz deneyerekbulabilirsiniz. ġekilde gözüktüşü gibi, dişer 4 kablo motor sargılarına (coil) gitmektedir. Bu 4 kablonun da bir sırası vardır. Bu sırayı da deneme yanılma yöntemiyle bulmak mümkün olacaktır. Eşer bu kabloları yanlıŞ sırada başlarsanız, motor dönmek yerine sadece titreme yapacaktır. Yukarıda da bahsettişim gibi motora adım attırmak için yapmamız gereken, vMotor kablosuna +12 Volt verirken, dişer sargılara başlı kablolara belli bir sıra ile toprak göndermektir.



Şekil- 45.ULN 2003 entegresi başlantı Şekli

Devrenin Kurulumu ve Çalışması

Yukarıda bahsedilen bu 4 kabloya toprak sinyalini göndermek için entegrenin 3,4,5 ve 6. bacaklarına sıra ile +5 Volt göndermemiz gerekiyor. Bu devrede kullanılan adım motoru 1.8 dereceliktir. Bu, motora attıracaşınız her normal adımda 1.8 derecelik bir dönme elde edeceşiniz demek oluyor. Bu da motorun bir tur atması için 200 normal adım atması gerektişi anlamına geliyor. Motorun vMotor dıŞında kalan dişer 4 kablosuna göndereceşiniz sinyallere göre bu adımın yönünü ve açısını deşiştirmeniz mümkün olacaktır. En basitinden motora ters adım attırmak için, sinyalleri D3‟ten D0 a doşru göndermeniz yeterli olacaktır. Çok hassas çalışmadışımızı ve motorumuzun 2 derece olduşunu ve 45 derecelik bir dönme gerçekleştirmek istedişimizi düŞünelim. Bunun için yukarıda anlattılan normal adım sinyalleri yeterli olmayacaktır. Bu durumda motoru 1‟er derecelik açılarla döndürebilmemiz gerekmektedir. Yarım adım attırma metodu ile bu iŞi kolayca yapmamız mümkündür. Bir dişer metod ise dalga sürümü sinyalleridir. Hassas hareketler üzerinde çalışmayacaksanız dalga sinyallerini kullanabilirsiniz. AŞaşıdaki tablolarda tam adım, yarım adım ve dalga sürümü için uçlara göndermeniz gereken sinyal çeşitlerini göndermeniz gereken deşerleri yazmaktadır. Deşerlerin ikilik sistemdeki karşılıkları D3-D0 sütunlarını soldan saşa doşru okuduşumuzdaki deşerlerine eşit olduşuna dikkat ediniz. Eşer ters yönde dönüŞ elde etmek istiyorsanız, sinyalleri ters yönde (tablodaki satırları aŞaşıdan yukarıya doşru okuyarak) gönderebilirsiniz.

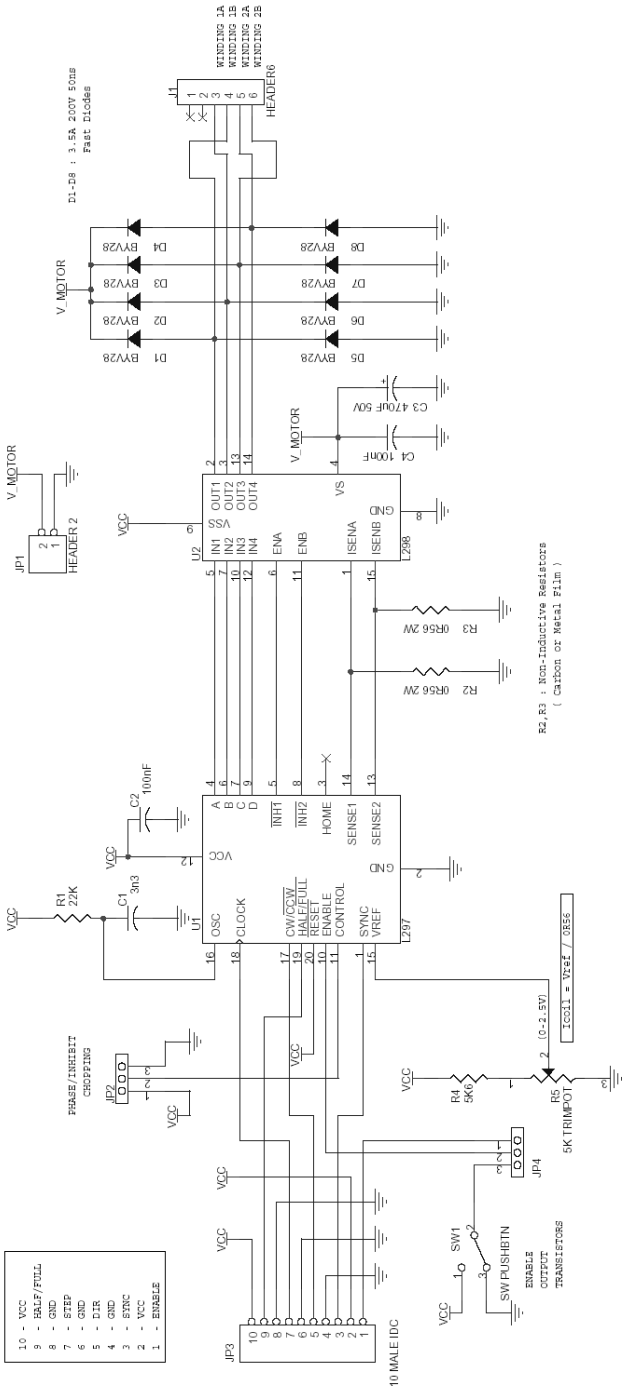
Bir dişer durumda adım motorun referans noktasını nasıl bulacaşıdır. Yani motorun durduşu en son pozisyonun ne olduşunu nerden bileceşiz? Döndürme iŞlemine baŞladışımız noktayı biliyorsak bu çok fazla sorun olmayacaktır. Fakat motoru daha döndürmeye baŞlamadan, elimizle biraz çevirdişimizi düŞünelim. Bu durumda baŞlangıç noktası kayacak ve motoru istedişimiz pozisyona getiremeyeceşiz. Disket sürücülerde kullanılan yöntem oldukça ilkel ama geçerli bir yöntemdir. Disket sürücü bir Şekilde diski okuyan kafanın nerede olduşunu bilmek zorundadır. Bunun için motoru bir yönde sürekli döndürerek, kafanın en baŞa dayanmasını saşlar. Bu gelinen noktaya referans noktası denir. Bu sebeple bazı adım motorların kendi etrafında sürekli

olarak dönmesini engelleyecek bir tırnak vardır. Motoru referans noktasına dayamak için bu tırnaktan yararlanılır. Biz Şimdilik hassas hareket yaptırmayacaşımızdan varsa bu tırnaşı sökebilirsiniz.

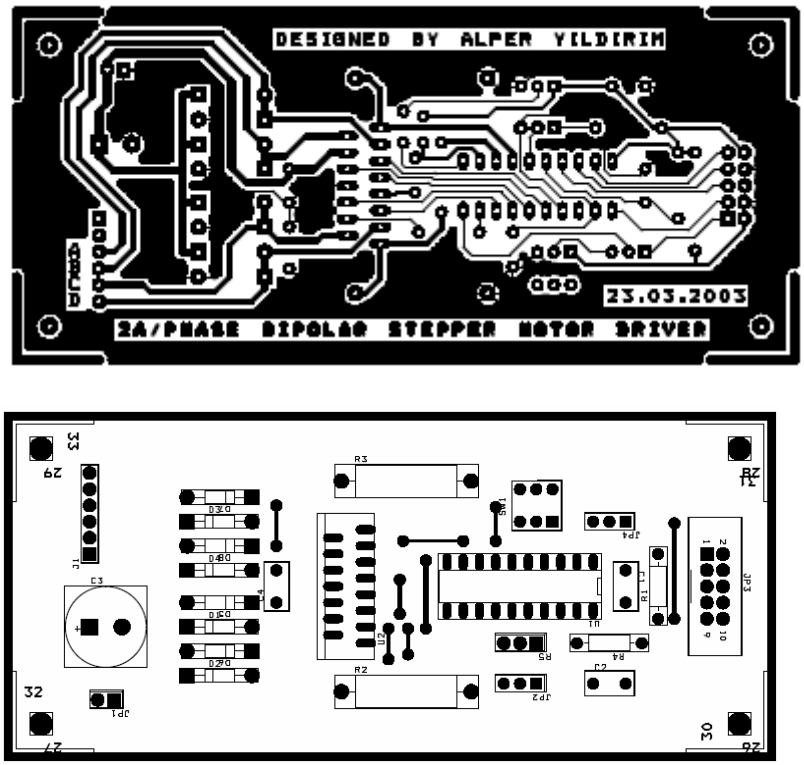
L297 adım motor kontrol entegresidir, GiriŞine uygulanan adım ve dir sinyalleri ile çıkıŞında adım motor faz sinyallerini üretmektedir. Entegre full-adım, half-adım ve wave- drive modlarında çalıŞabilmektedir. L298 H-bridge sürücü entegresidir. Bipolar adım motorların sürülmesi için tasarlanmıŞtır. Max 2A/phase akım verebilir. GiriŞine uygulanan faz sinyallerini çıkıŞa yükseltip vermektedir. ST firmasının sunduşu "application note" lar incelendikten sonra L297 ve L298 entegreleri birlikte kullanılarak adım motor sürücüleri yapılmıŞtır.

Sürücülerin özellikleri Şunlardır:

- Devrenin Başlantı Şeması
- Adım ve dir sinyalleriyle çalışma
- Max 45V motor voltajı
- Max 2A faz akımı
- Full-adım, Half-adım ve Wave-drive modlarında çalıŞabilme
- Ayarlanabilir faz akımı



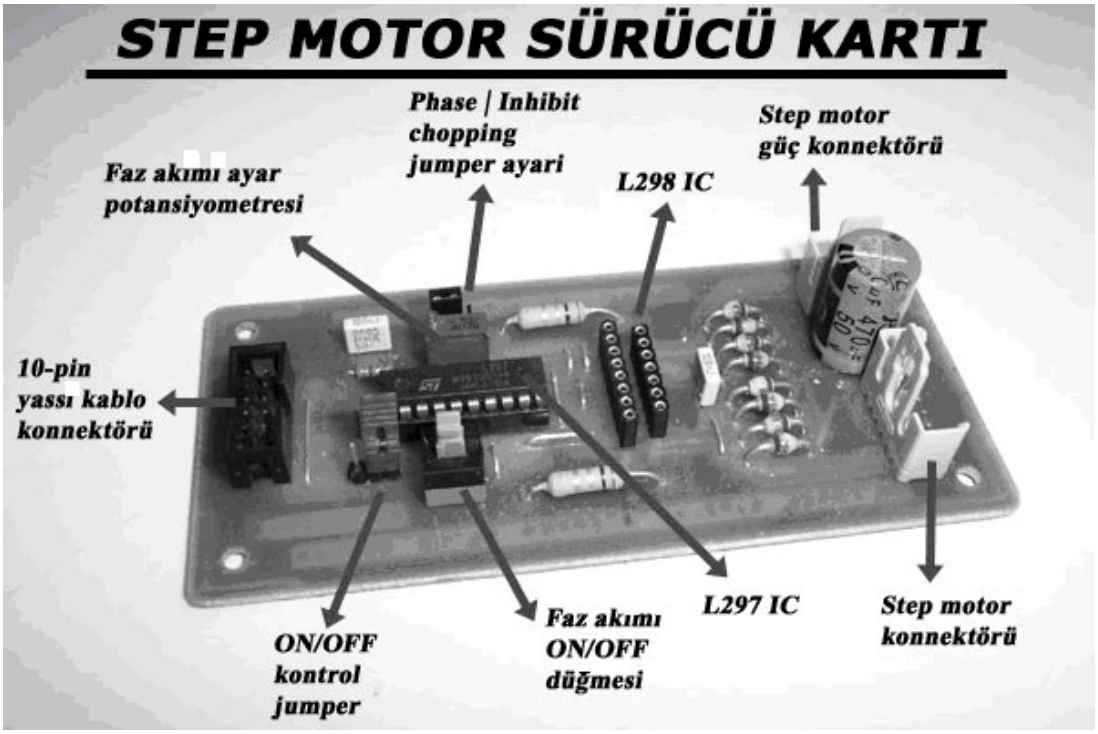
Şekil- 46.Sürücü devresinin açık devre şeması.



Şekil- 47.Sürücü devresinin baskı devre ve eleman yüzü görünüşü.

Malzeme Listesi

- C1 3n3
- C4, C2 100Nf
- C3 470µF 50V
- D1…8 470µ 50V
- JP1 HEADER 2
- JP2, JP4 JUMPER3lü
- JP3 10 MALE IDC
- J1 HEADER6
- R1 22K
- R2, R3 OR56 2W
- R4 5K6
- R5 5K TRIMPOT
- SW1 SW PUSHBIN
- U1 L297
- U2 L298



Şekil- 48.Sürücü kartı

PWM İle DA Motoru Denetimi

Deney PCA ile PWM

```
//-------------------------------------------------------------------------*/
#include "p89v51rx2.h"
#include <stdio.h>
        unsigned char gelen_karakter;
//********************** fonsiyon tanimlamalari *************************
void pca_pwm_ayar(void);        //PCA'i PWM üreteci olarak kullan
void UART_Init (void);
void T0_init() ;
void msec(int gecikme);
//***********************ana program ******************************
void main(void)
{
        pca_pwm_ayar();                 //ayarla
        T0_init() ;
        UART_Init (); //UART_Init();
        printf ("\n D oranini girin: ");
        gelen_karakter = getchar ();                                    //seri porttan gelen
karakteri oku
   printf ("\nGirilen D orani: %c",gelen_karakter);
   printf ("\nHexadecimal deðer: %bx",gelen_karakter);
while(1)
{
        printf ("\n D oranýný girin: ");
        gelen_karakter = getchar ();                                    //seri porttan gelen
karakteri oku
   printf ("\nGirilen D oraný: %c",gelen_karakter);
        CCAP1H = gelen_karakter;
        CCAP0H = (P3 & 0xF0);                                           //DSW P3.4, P3.5,
P3.6 ve P3.7'baðla
}
}
//********************** Functions and Procedures ***********************
void pca_pwm_ayar(void)
{
        CMOD = 0x04;        //t0 tasmasi  CMOD 1. ve 2. bitlerini ayarla.
        CL = 0x00;          //
        CH = 0x00;          //
        CCAPM0 = 0x42;      //Module 0 in PWM mode
        CCAP0L = 0x00;
        CCAP0H = 128;       //50 percent duty cycle
        CCAPM1 = 0x42;      //Module 1 in PWM mode
        CCAP1L = 0x00;
        CCAP1H = 50;        //50 percent duty cycle
        CCAPM2 = 0x42;      //Module  in PWM mode
        CCAP2L = 0x00;
        CCAP2H = 128;       //50 percent duty cycle
        CR=1;               //Turn on PCA timer
}
void UART_Init (void)
{
```

```
   SCON = 0x52;          // SCON 8 bit veri, 1 start bit, non stop bit
   TH1=0xfd;                                                  //9600 baud
   TL1 = TH1;           // baudrate göre deðer hesapla yaz
   TR1 = 1;             // timer 1'i ba_lat
   TI = 1;              // Gönderici bos ve hazir
}
void T0_init()
{
      TMOD = 0x22;
      TH0 = 250;                  // fosc/12/sayýcý deðeri/256=pwm_frekans,          TH0  =  253
1 Khz
                                  TL0 = 250;
      EA = 1;
         ET0 = 1;
      TR0 = 1;
}
void timer0() interrupt 1
{
   TF0 = 0;                                          //kesme bayraðýný temizle
}
void msec(int gecikme)                                    /* 1 milisaniyelik zaman geciktirme */
{        int k, z;
         for(k=0; k<gecikme; k++)
   {
         for (z=0; z<500; z++);
   }
}
```

PWM ile SERVO Motor denetimi

```
//-------------------------------------------------------------------------*/
#include "p89v51rx2.h"
#include <stdio.h>
        unsigned char gelen_karakter;
//********************** fonsiyon tanimlamalari ***************************
void pca_pwm_ayar(void);        //PCA'i PWM üreteci olarak kullan
void UART_Init (void);
void T0_init() ;
void msec(int gecikme);
//***********************ana program ******************************
void main(void)
{
        pca_pwm_ayar();                 //ayarla
        T0_init() ;

while(1)
{
        CCAP0H = (P3 & 0xF0);                                           //DSW  P3.4,  P3.5,
P3.6 ve P3.7'baðla
}
}
//********************** Functions and Procedures ************************
void pca_pwm_ayar(void)
{
        CMOD = 0x04;          //t0 tasmasi  CMOD 1. ve 2. bitlerini ayarla.
        CL = 0x00;            //
        CH = 0x00;            //
        CCAPM0 = 0x42;        //Module 0 in PWM mode
        CCAP0L = 0x00;
        CCAP0H = 128;         //50 percent duty cycle
        CCAPM1 = 0x42;        //Module 1 in PWM mode
        CCAP1L = 0x00;
        CCAP1H = 50;          //50 percent duty cycle
        CCAPM2 = 0x42;        //Module  in PWM mode
        CCAP2L = 0x00;
        CCAP2H = 128;         //50 percent duty cycle
        CR=1;                           //Turn on PCA timer
}
void UART_Init (void)
{
  SCON = 0x52;          // SCON 8 bit veri, 1 start bit, non stop bit
  TH1=0xfd;                       //9600 baud
  TL1 = TH1;           // baudrate göre deðer hesapla yaz
 //  TMOD &= ~0xF0;         // TMOD: timer 1  8-bit autoreload  modunda
 //  TMOD =  0x22;
  TR1 = 1;             // START Timer1
  TI = 1;              // Indicate TX0 ready
}
void T0_init()
{
      TMOD = 0x22;
```

```
      TH0 = 185;                  // fosc/12/sayýcý deðeri/256=pwm_frekans,          TH0  =  253
1 Khz
                                  TL0 = 185;
      EA = 1;
         ET0 = 1;
      TR0 = 1;
}
void timer0() interrupt 1
{
   TF0 = 0;                                  //kesme bayraðýný temizle
}
void msec(int gecikme)                              /* 1 milisaniyelik zaman geciktirme */
{         int k, z;
         for(k=0; k<gecikme; k++)
  {
         for (z=0; z<500; z++);
  }
}
```

Proje 1: Çift Eksen Güneş İzleyici

Sun_track.c

```
//-------------------------------------------------------------------------*/
#include "p89v51rx2.h"
#include <stdio.h>
        unsigned char gelen_karakter;
sbit s1=P3^4;
sbit s2=P3^5;
sbit s3=P3^6;
sbit s4=P3^7;
sbit mbrk1 =P2^0;
sbit mbrk2 =P2^2;
sbit mbrk3 =P2^2;
sbit mbrk4 =P2^3;
sbit mbrk5 =P2^4;
sbit mbrk6 =P2^5;
sbit mdir1 =P0^0;
sbit mdir2 =P0^1;
sbit mdir3 =P0^2;
sbit mdir4 =P0^3;
sbit mdir5 =P0^4;
sbit mdir6 =P0^5;

//********************** fonsiyon tanimlamalari *************************
void pca_pwm_ayar(void);        //PCA'i PWM üreteci olarak kullan
void UART_Init (void);
void T0_init() ;
void msec(int gecikme);
//**********************ana program ******************************
void main(void)
{
        pca_pwm_ayar();                 //ayarla
        T0_init() ;
        UART_Init (); //UART_Init();
                CCAP1H = 200;
        CCAP0H = 128;                                           //DSW P3.4, P3.5, P3.6 ve
P3.7'baðla
        P2=00;//brk
        P0=00; //yön
while(1)
{
        switch (((P3 & 0xf0) >> 4))
{
                case 0:
                        P2=0xff;        //bütün motorlar duruyor

                break;

        case 1:
                mdir1=0;
                mdir2=0;
                mdir3=0;
```

```
                mdir4=0;
                mdir5=0;
                mdir6=0;

        mbrk1 = 1;
        mbrk2 = 1;
        mbrk3 = 1;
        mbrk4 = 1;
        mbrk5 = 1;
        mbrk6 = 1;
                break;
        case 2:
                                mdir1=0;
                mdir2=0;
                mdir3=0;
                mdir4=0;
                mdir5=0;
                mdir6=0;

        mbrk1 = 1;
        mbrk2 = 1;
        mbrk3 = 1;
        mbrk4 = 1;
        mbrk5 = 1;
        mbrk6 = 1;

                break;
                case 3:
                                mdir1=0;
                mdir2=0;
                mdir3=0;
                mdir4=0;
                mdir5=0;
                mdir6=0;

        mbrk1 = 1;
        mbrk2 = 1;
        mbrk3 = 1;
        mbrk4 = 1;
        mbrk5 = 1;
        mbrk6 = 1;

                break;
        case 4:

                break;



        default:
                break;
}

}
}
```

```
//********************** Functions and Procedures ***********************
void pca_pwm_ayar(void)
{
        CMOD = 0x04;        //t0 tasmasi  CMOD 1. ve 2. bitlerini ayarla.
        CL = 0x00;          //
        CH = 0x00;          //
        CCAPM0 = 0x42;      //Module 0 in PWM mode
        CCAP0L = 0x00;
        CCAP0H = 128;       //50 percent duty cycle
        CCAPM1 = 0x42;      //Module 1 in PWM mode
        CCAP1L = 0x00;
        CCAP1H = 50;        //50 percent duty cycle
        CCAPM2 = 0x42;      //Module  in PWM mode
        CCAP2L = 0x00;
        CCAP2H = 128;       //50 percent duty cycle
        CR=1;               //Turn on PCA timer
}
void UART_Init (void)
{
  SCON = 0x52;          // SCON 8 bit veri, 1 start bit, non stop bit
  TH1=0xfd;                     //9600 baud
  TL1 = TH1;           // baudrate göre deðer hesapla yaz
 //  TMOD &= ~0xF0;        // TMOD: timer 1  8-bit autoreload  modunda
 //  TMOD =  0x22;
  TR1 = 1;             // START Timer1
  TI = 1;              // Indicate TX0 ready
}
void T0_init()
{
      TMOD = 0x22;
      TH0 = 255;                // fosc/12/sayýcý deðeri/256=pwm_frekans,          TH0  =  253
1 Khz
                TL0 = 255;
      EA = 1;
         ET0 = 1;
      TR0 = 1;
}
void timer0() interrupt 1
{
  TF0 = 0;                                      //kesme bayraðýný temizle
}
void msec(int gecikme)                                  /* 1 milisaniyelik zaman geciktirme */
{       int k, z;
        for(k=0; k<gecikme; k++)
  {
        for (z=0; z<500; z++);
  }
}
```

Proje 2: HEXAPOT ROBOT

```
#include "P89V51Rx2.H"
#include "servo.h"
#include "lcd_8b.h"
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Mesajlar
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
unsigned int say;
unsigned char say1;
sbit T1_giris = P3^5;
char code *mes1 = "HEXAPOT ROBOT";
char code *mes2 = "     2014       ";
//*******************************************************
//baglant1lar böyle olmal1
sbit en1=P2^0;                    //motoru izinler
sbit en2=P2^1;
sbit en3=P2^2;
sbit en4=P2^3;
sbit en5=P2^4;
sbit en6=P2^5;
sbit brk1=P2^0;          //motoru frenler
sbit brk2=P2^1;
sbit brk3=P2^2;
sbit brk4=P2^3;
sbit br5=P2^4;
sbit brk6=P2^5;
void zam_ayar(void); /*T0 zamanlayici kiP 1, T1 sayici kip 1   */

//-------------------------------------------------------
void main()
{         static data unsigned char buf [16];
          data unsigned int n;
          say=0;
          init_LCD();
//        zam_ayar();
          lcd_cmdwr(satir1);
          disp_sat(mes1);
          lcd_cmdwr(satir2);
          disp_sat(mes2);
          msec(1000);
//        T1_giris=1;
          lcd_cmdwr (CLEAR_DISPLAY);
while (1)
          {         P1 = 0xFF; /* Motor STOP */
                    msec (100);
                    Servos_init();
                    EA=1;
                    ServoR_forward();
                    ServoL_back();
          }
          }
void zam_ayar(void)
          {
```

```
            TMOD=0x51;
            ET0=1;
            TR0 = 1;                        // T0'ý baþlat
            TR1 =1;                   // T1'i baþlat
          }

void zmn0_kesme(void) interrupt 1
{         TR0 = 0;        //Kesme geldiðinde yapýlmasýný istediðin programý yaz
          TH0 = 0x3C;             //        YDB
          TL0 = 0xB0;             //        DDB
          TF0 = 0;                        // T0 Taþma bayraðýný temizle
          TR0 = 1;                        // T='ý baþlat
          say1++;
          if (say1==20)
          { say1=0;
          say=((TH1<<8)| TL1);
          TR1=0;                                  //T1'i sýfýrla
          TH1=0;
          TL1=0;
          TR1=1;
          }
          T1_giris=1;

}
```

Proje 3: 3 Eksen Robot

*/ 1. Unipolar adim motoru baglantisini ULN2803 Transistör paketini kullanarak yapin ve tam ve yarim adim örnek programlari yazarak çalistirin.
2. Tus takimindan 1 tusunu arttirma 2 tusunu eksiltme tusu olarak kullanan programi yazin ve çalistirin.
3.Orta tusu her basmada motorun yönünü degistiren islevi adim 2'e ekleyin ve programini yazarak çalistirin.
4. Zamanlayiciyi zaman geciktirme amaciyla kesmeli modda kullanan programi adim 3'te degistirmeleri yaparak çalistirin.
5. Dis kesme kaynagi 0 girisini motorun yön degistirme girisi olarak kullanan programi yazarak çalistirin. */

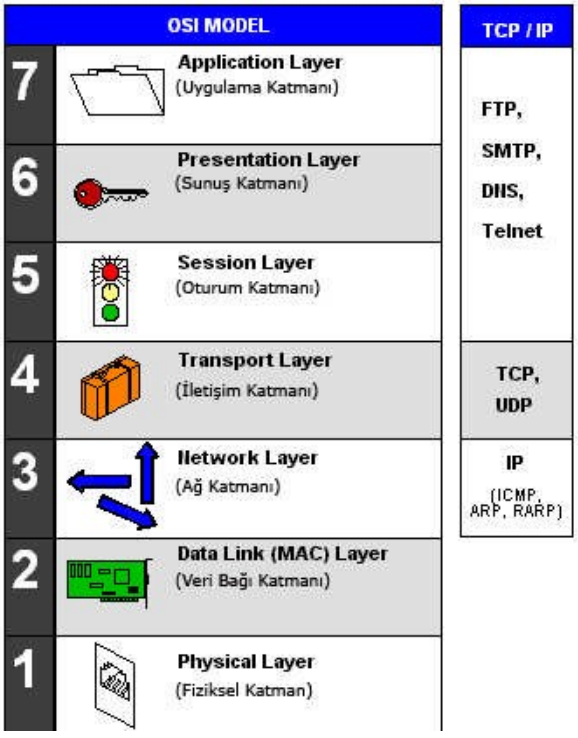
```
//May1s 2014 Mustafa Engin
#include <reg52.h>
#include <stdio.h>
#include <intrins.h>
#include "lcd_8b.h"
/*++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 Mesajlar
 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
char code *mes1 = "Mikrodenetleyici";
char code *mes2 = "     2014       ";
unsigned int say;
unsigned char say1,hiz,a,b;
sbit int0 = P3^2;
sbit but1=P3^6;
sbit but2=P3^7;
bit yon;
//*****************************************************
void zam_ayar(void);  //T1'i 16 bit say1c1 T0'1 16 bit kesmeli zamanlay1c1 olarak ayarlar
//-----------------------------------------------------
void main (void)
{
        //static data unsigned char buf [16];
        //data unsigned int n;
        say=0;
        a=0x11;
        //init_LCD();
        zam_ayar();
/*      lcd_cmdwr(satir1);
        disp_sat(mes1);
        lcd_cmdwr(satir2);
        disp_sat(mes2);
        msec(2000);
*/
        EA=1;
        hiz=10;
        but1=1;
        but2=1;
        int0=1;
//      lcd_cmdwr (CLEAR_DISPLAY);
        while(1)
{
        /* n = sprintf (buf,"F=%3u Hz",say);   //deðeri ASCII'ye dönüþtür
```

```
	lcd_cmdwr(satir1);
	disp_sat(buf);                                          //görüntüle
	*/

P1=a;
	msec(350);
	if (!but1)
	{
		hiz++;
	}
if (!(but2))
	{
		hiz--;
	}

}
}

void zam_ayar(void)
{
  TMOD=0x01;//t0 16 bit zamanlay1c1, T1 16 bit say1c1
  ET0=1;                        //zamanlaici 0 kesmesi açik
	EX0=1;                          //diskesme 0 kesmesi açik

  TR0 = 1;                      // T0'ý baþlat
// TR1 =1;                      // T1'i baþlat
}

void zmn0_kesme(void) interrupt 1
{
	TR0 = 0;                                //Kesme geldiðinde yapýlmasýný istediðin
programý yaz
	TH0 = 0x3C;             //      YDB
	TL0 = 0xB0;             //      DDB
	TF0 = 0;                                // T0 Taþma bayraðýný temizle
	TR0 = 1;                                // T='ý baþlat
	say1++;
	if (say1==hiz)
	{ say1=0;
		if (yon)
		{
			b = _crol_(a,1);
			a=b;
		}
		else
		{
			b = _cror_(a,1);
			a=b;
		}
	}
}
void tersle(void) interrupt 0
{
	yon=!yon;
}
```

Sayısal Veri İlerimi

Giriş

TCP/IP protokolü OSI (Open Systems Interconnection-değişik işletim sistemine sahip makinelerin birbiriyle haberleşmesini sağlayan model) modeli içerisinde, uygulamalar arasında iletişimi sağlayan katmanı (transmission layer) oluşturur. TCP/IP, TCP ve UDP olmak üzere data iletişimini farklı şekillerde sağlamakla görevli olan iki protokolü bünyesinde barındırır. TCP (Transmission Control Protocol); son uçtan son uca veri dağıtım fonksiyonu sağlar; verinin güvenli iletimi için gerekli mekanizmaları içerir. Bu mekanizmalar hata denetimi, sıra numarası, onay ve yeniden gönderim fonksiyonlarını içerir. TCP güvenli ve sıralı hâle getirilmiş veriyi uygulama katmanına sunar. UDP (User Datagram Protocol) ise; veri iletimi sırasında gönderilen bilgi paketlerinin hedef bilgisayarlara ulaşacağını garanti edemez ve akış kontrolü sağlayamaz. UDP, gönderilen paketlerin sadece belirli bir bilgisayarı hedef aldığı uygulamalarda kullanılmaktadır. Bu uygulamalardan bazıları: DNS, toplu yayın (broadcast) ve grup yayını (multicast) ve RIP protokolü uygulamalarıdır. TCP güvenli iletişim için birçok kontrolü mekanizmasını işletim sırasında devreye soktuğundan üzerinde çalıştığı kullanıcıya yük getirir ve iletişimde bir miktar gecikmeye neden olur. TCP ve UDP, İnternet protokolünün üzerinde çalışır ve bu protokol üzerinden aldığı hizmetlere işlevsellik kazandırır. TCP ve UDP protokollerini kullanarak uzaktaki makinelere doğrudan veri iletimi yapılmaz. Bunun yerine bilgisayarda çalışan uygulamalar arasında veri iletimi yapılır.



Şekil 1.2: OSI Modeli ve TCP/IP modeli karşılaştırması

Kısaca iletim katmanı; OSI modelindeki aktarım katmanının karşılığıdır. Temel fonksiyonu, uygulamalar arasındaki haberleşmeleri sağlamaktır. İnternet katmanı sadece bir veri dağıtım servisi sağlar. Aktarım katmanı ise güvenli iletişim, hata

düzeltme, gecikme kontrolü ve benzeri fonksiyonlarla ilgilenir. Bu fonksiyonlarla uygulama katmanına servis sunar.

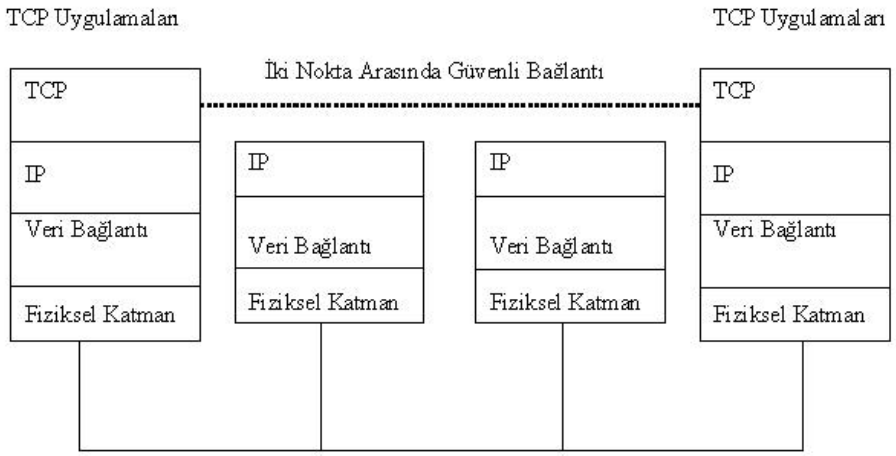
TCP

TCP yani Transmission Control Protocol yedi katmanlı OSI referans modelinin, aktarım katmanında yer alır. TCP iki hostun birbirleriyle güvenilir ve bağlantılı haberleşmesini sağlar. Telnet, FTP, SMTP gibi protokoller TCP' yi kullanır.

TCP Protokolünün Yapısı

TCP protokolü, bağlantılı ve güvenli veri akışını sağlayarak iletim katmanına çok önemli hizmetler sunar. Çoğu uygulama kendi veri iletişim kontrol mekanizmasını oluşturmaktansa TCP protokolünün sağlamış olduğu hizmetleri kullanır. TCP, sunduğu hata denetimi, veri akış kontrolü gibi hizmetler sayesinde kendisini kullanan uygulamalara tatmin edici düzeyde güvenlik, hata denetimi ve akış kontrolü sağlar.

TCP protokolü, bilgisayarlarda çalışan uygulamalar arasında <İstemci IP Adresi, Port Numarası>, <Sunucu IP Adresi, Port Numarası> ikililerini temel alan bağlantılar kurar. Her TCP bağlantısı, bu ikililerle ifade edilir. İnternet protokolü bağlantısızdır ve gönderilen paketlerin hedeflerine ulaşmalarını garanti edemez. Bu sorunları ortadan kaldırmak için TCP protokolüne ihtiyaç duyulur. Bu protokol, güvenli bilgi akışını sağlayabilmek için çeşitli yöntemler kullanır.



Şekil 1.3: IP ağları arasında TCP protokolü işleyişi

TCP en çok kullanılan, bağlantıda olan bilgisayarlar arasında güvenli veri iletişimi sağlayan, sanal devre bağlantısı mantığı ile çalışan iletim protokolüdür. TCP sıklıkla IPv4 ve IPv6 protokolleriyle beraber kullanılır. TCP çalıştığı bilgisayar ağı alt yapısından bağımsızdır. Ağ altyapısı ve mimarisi sadece veri iletim hızını etkiler.

TCP protokolünün en önemli özellikleri şunlardır:

- Bağlantı noktaları arasında veri iletişimini sağlaması
- Güvenli veri iletimi sağlaması
- Bağlantıda olan iki bilgisayar arasında akış kontrolü sağlaması

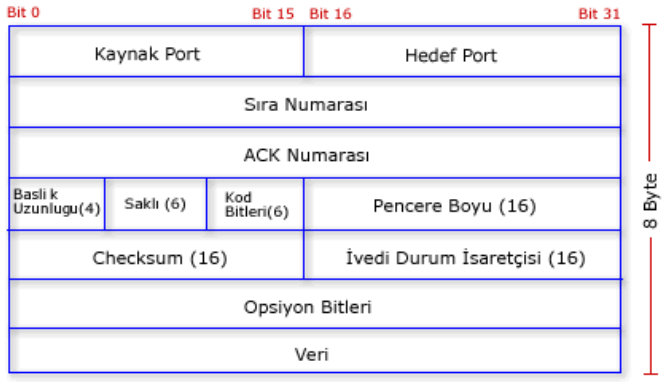
- Çoklama (Multiplexing) yöntemi ile birden fazla bağlantıya izin vermesi
- Sadece bağlantı kurulduktan sonra veri iletimi sağlaması
- Gönderilen mesaj parçaları için öncelik ve güvenlik tanımlaması yapılabilmesi

TCP ile İletişim

TCP'de tanımlı temel görevleri aşağıdaki gibi sıralayabiliriz:

- Bir üst katmandan gelen verinin uygun uzunlukta parçalara bölünmesi
- Her bir parçaya alıcı kısımda aynı biçimde sıraya koyulabilmesi amacıyla sıra numarası verilmesi
- Kaybolan veya bozuk gelen parçaların tekrarlanması
- Uygulamalar arasında yönlendirme yapılması
- Güvenilir paket dağıtımın sağlanması
- Hostlarda veri taşmasının önlenmesi

TCP kendisine atanmış olan bu görevleri yapabilmek amacıyla aktarım katmanında veri parçalarının önüne başlık bilgisi ekler. Başlık bilgisi ve veri parçası birlikte TCP segmenti olarak anılır. Her segmente sıra numarası verilir. Bu segmentler belli sayılarda gönderilir. Alıcı bilgisayar da frameler yani segmentler kendine ulaştıkça bunları tampon belleğine yerleştirir. İki ardışık çerçeve tampon belleğe yerleşince alıcı bilgisayar, gönderilen en son çerçeve için bir onay mesajını gönderici bilgisayara yollar. TCP segmentinde başlık içindeki alanların kullanımı amaçları aşağıdaki gibidir.



Şekil 1.4: TCP segment formatı

Kaynak Port (Source Port): Gönderen bilgisayarın kullandığı TCP portu. Bir üst katmanda TCP isteyen protokol sürecinin kimliği durumundadır. Karşı mesaj geldiğinde bir üst katmana iletmek için, o protokolün adı değil de port numarası kullanılır. 16 bitlik kaynak port alanı bulunur.

- Hedef Port (Destination Port): Alıcı bilgisayarın kullandığı TCP portu.

Gönderilen veri paketinin alıcı tarafta hangi uygulama sürecine ait olduğunu belirtir. Varış noktasındaki üst katman protokolünün portunu gösterir. 16 bitliktir.

- Sıra Numarası (Sequence Number): Gönderilen paketin sıra numarasını gösterir. Gönderilmeden önce daha küçük parçalara ayrılan verinin, alıcı
- kısımda yeniden aynı sırada elde edilmesinde kullanılır. 32 bitliktir.

- ACK Numarası (Acknowledgment Number): Gönderilen verinin en son hangi sekizlisinin alındığını göndericiye iletmek için kullanılır.
- Başlık Uzunluğu (Header Length): TCP segmentinin uzunluğu, TCP
- başlığında var olan 32 bit uzunluğundaki sözcüklerin sayısını gösterir.
- Saklı Tutulmuş (Reserved): İleride olabilecek genişleme için saklı tutulmuştur. Gelecekte kullanılmak üzere saklı tutulmuş anlamına gelir.

Kod Bitleri (Bayraklar, Flags): Kontrol bilgilerini taşımak için kullanılırlar.

Segmentin içeriğine dair bilgi taşırlar.

- Pencere (Window): TCP penceresinde ne kadar alan olduğunu gösterir. Alış denetimi için kullanılır. 16 bitliktir.
- Hata Sınama Bitleri (Cheksum): Verinin ve başlığın hatasız aktarılıp aktarılmadığını sınamak için kullanılır. 16 bitliktir.

Acil İşaretçisi (Urgent Pointer): Acil olarak aktarımı sonlandırma, bayraklar kısmında acil olan bir verinin iletilmesi gibi durumlarda kullanılır. Acil veri, alıcının uygulama katmanında öncelikle değerlendirmesi gereken veridir. TCP protokolü İnternette kullanılan ana protokoldür. Dosya transferi ve uzak oturumlar gibi kritik işleri sağlar. TCP diğer protokollerden farklıdır. Güvensiz bir iletişim ortamında verilerin aynı şekilde hedefe ulaşacağından emin olamazsınız. Ancak TCP gönderilen verilerin gönderildiği sırayla karşı tarafa ulaşmasını sağlayarak güvenli veri iletimini sağlar.

TCP iki makine arasında kurulan sanal bir bağlantı üzerinden çalışır. Üç kısımlı bir işlemden oluşur. Bu bağlantı ve three-part handshake olarak bilinir. (TCP/IP three way handshake) TCP/IP üzerinde yapılan bazı saldırı tekniklerini iyi anlayabilmek için TCP'nin çalışma mantığını iyi anlamak gerekmektedir. Three-way handshake işleminde öncelikle istemci sunucuya port numarasıyla birlikte bir bağlantı isteği gönderir. İsteği alan sunucu bu isteğe bir onay gönderir. En sonunda da istemci makine sunucuya bir onay gönderir ve bağlantı sağlanmış olur. Bağlantı yapıldıktan sonra veri akışı her iki yönde de yapılabilmektedir. Buna genellikle full-duplex iletişim denmektedir.

TCP aynı zamanda hata kontrol mekanizması da sağlıyor. Gönderilen her veri bloğu için bir numara üretilmektedir. Karşılıklı iki makine de bu numarayı kullanarak transfer edilen blokları tanımaktadır. Başarılı olarak gönderilen her blok için alıcı makine, gönderici makineye bir onay mesajı gönderir. Ancak transfer sırasında hata olursa alıcı makine ya hata mesajları alır ya da hiçbir mesaj almaz. Hata oluştuğu durumlarda, oturum kapanmadığı sürece, veriler tekrar gönderilir. TCP protokolü ile verinin iki makine arasında nasıl transfer edildiğini gördük. Şimdi istemcinin isteğinin karşı tarafa ulaştığında ne olup bittiğine bakalım. Bir makine başka bir makineye bağlantı isteği gönderdiği zaman belli bir hedef adresi belirtir. Bu adres bir IP adresi ve fiziksel adrestir. Ancak sadece bu adreste yeterli değildir, istemci karşı makinede hangi uygulamayla konuşmak istediğini de belirtmek durumundadır. Connection-oriented veri iletiminde gönderilen veri paketler kullanıcıya aynı düzende ve güvenli bir şekilde iletilmelidir. Eğer herhangi bir paket kaybolur, zarar görür, aynısı gönderilir veya alıcı tarafından aynı düzende alınamazsa protokol başarısız olmuş sayılır. Bunun en kolay çözüm yolu ise her gönderilen paketten sonra bir sonraki paketin bilgisini alıcıya göndermektir.

Eğer gönderici her gönderilen paketten sonra diğer paketin bilgisini gönderirse bu iş/zaman oranında önemli ölçüde düşüşe neden olur. Birçok Connection-oriented güvenlik protokolü, ağ üzerinden iletişim zamanın çok önemli olduğundan aynı anda birden fazla paket gönderir. Alıcı tarafından alınan bu paketlerden sonra gönderilen paketlerin tümü ile alakalı olan bir bilgi alıcıya gönderilir. İşte buna pencere boyutu veya pencereleme denilir.

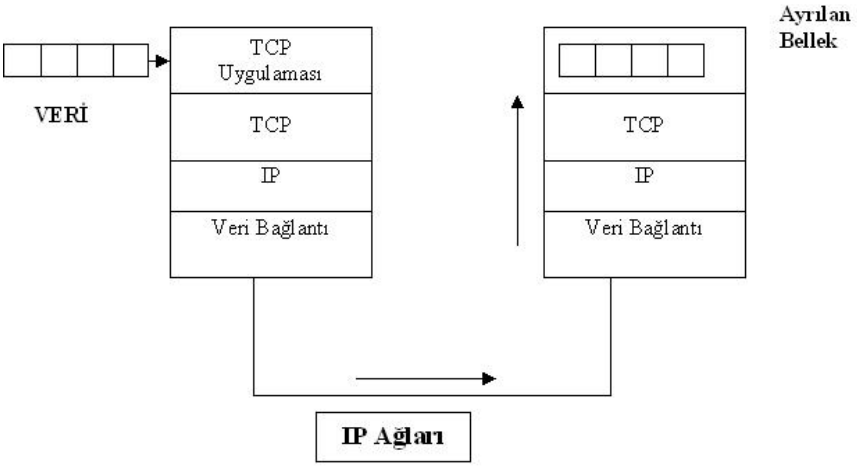
Bir cihazdan diğer bir cihaza güvenli bir iletişim veri tekrarlaması veya kaybı olmadan veri dizisinin iletimini garanti eder. Pozitif bilgilendirme verinin güvenli bir şekilde iletildiğini garanti eden bir tekniktir. Bu teknikte belirli sayıda veri paketleri gönderildikten sonra alıcı tarafından paketlerin güvenli bir şekilde alındığı kaynağa iletilir. Kaynak cihaz, bu bilgilerin tamamını tutar ve istenmeyen bir durum olduğunda hedef kaynağa paketlerin yeniden gönderilmesi için başka bir bilgi paketi göndererek verinin tamamı düzgün gelene kadar bu işlem sürer.

Veri Transferi

TCP protokolünün en önemli özelliği sürekli ve her iki yönde veri akışını sağlamasıdır. Gönderilen veriler 8 bitlik (oktet) gruplar hâlindedir. Bu veriler, bağlantıda olan sistemlerde yürütülen TCP protokolünü işleten uygulamalara parçalar hâlinde iletilir. Mesaj parçaları değişik uzunluklarda olabilir. Bu parçalar gönderilebilecek en büyük parça değerini aşmayacak uzunlukta olmalıdır. TCP protokolünde mesaj uzunluğunu sınırlandıracak herhangi bir kısıtlama yoktur. Gönderilen datagramların uzunluğunu sınırlamak IP katmanının görevidir. Bu nedenle TCP protokolü, bağlantıların daha etkin ve verimli olabilmesi için “MSS” (Maksimum Segment Size-En büyük Parça Büyüklüğü) için bir değer belirlemek zorundadır. MSS gönderilecek en büyük parça büyüklüğüdür. Kurulan her bağlantı için tanımlanır.

TCP protokolünde gönderilen mesaj segmentleri sekiz bitlik veriler hâlindedir. TCP protokolü gönderilen ve alınan her biti işaretleyerek takip eder. İşaretleyerek gönderdiği her parça için bağlantıda olduğu uçtan cevap bekler. Bu işaretleme sistemi sayesinde iletim sırasında kaybolan parçalar yeniden transfer edilebilir. Çoklama (Multiplexing) yöntemi ile TCP, iki bilgisayar arasında birden fazla bağlantı kurabilir. Bu protokol her büyüklükte verinin gönderilmesi için uygundur. Gönderilen paketler sadece bir oktetlik veri içerebileceği gibi, paketler 100 mega oktetlik veri transfer işleminde de kullanılabilir. TCP gönderdiği her okteti numaralandırır. Bu numaralar kullanılarak gönderilen verilerin gönderildiği sıra ile bağlantıda olan alıcı tarafından alınması sağlanır. Buna (sequencing of oktet) alınan oktetlerin sıralanması adı verilir.

Uygulamalar TCP protokolünü kullanarak, her defasında birden fazla oktet içeren segmentler gönderebilir. TCP aldığı bu mesaj segmentlerini depolar; bunları tek bir parça veya parçalar hâlinde gönderir. TCP protokolü ilettiği verinin gönderdiği sıra ile alınmasını garanti etmek zorundadır. Örneğin TCP uygulamasının 1024 oktetlik veri yollaması gerekirse, bu bilgiler 1024 tane 1 oktetlik veya 256 tane 4 oktetlik parçalar hâlinde gönderilebilir.



Şekil 1.5: İnternet protokolü kullanımı ile veri iletimi

TCP protokolü verileri sıra ile dizilmiş bilgiler hâlinde iletir. Gönderilen oktetlerde mesajın sonu olduğunu anlatacak herhangi bir belirteç yoktur. Tüm verilerin gönderildiğini TCP modülüne anlatabilmek için, TCP protokolünün "push" fonksiyonunu kullanması gerekir. "Push" fonksiyonu alınan paketin bir üst katmana iletilmesini sağlar.

TCP tarafından gönderilen verilerin herhangi bir yapısal özelliği yoktur. TCP protokolü, gönderilen verilerin yapısı hakkında herhangi bir bilgiye sahip değildir. Protokol veritabanı bilgileri veya şifrelenmiş veriler taşıyabilir. Yapısal olarak bu verilerin hangi uygulamalara ait olduğunu belirleyemez. Bu, ancak TCP protokolü kullanan uygulamalar tarafından sağlanabilir. Alınan verileri çözümleyen uygulamalar bu bilgileri kendi içlerinde kullanacakları şekillere uyar.

TCP protokolünün en önemli özelliği iki nokta arasında güvenli bağlantı sağlamasıdır. Bunun için TCP zarar görmüş, kaybolmuş, iki defa gönderilmiş ve düzenli sıraya göre gönderilmemiş datagramlarla uğraşmak ve bu hatalardan kaynaklanan sorunları gidermek zorundadır. Bunun için TCP "Pozitif Bilgilendirme Şeması" (Positive Acknowledgement Scheme) olarak bilinen PAR yöntemini kullanır. Alınan her veri parçasına karşılık bir bilgi paketi gönderilir.

TCP protokolü, gönderdiği her veriye karşılık bir dizi numarası (sequence number) yaratır ve buna karşılık gönderdiği her veri mesajına karşılık bir bilgi numarası (acknowledgement number) bekler. Eğer gönderdiği veriye karşılık belirlenen süre (timeout süresi) sonunda bilgi (ACK) alamazsa, paketi yeniden göndermek durumunda kalır. Gönderilen verilere dizi (sequence) numarası atanması, verilerin belirli bir sıra hâlinde gönderilmesini ve iki defa yinelenmesini önler. Gelen paketlerdeki herhangi bir bozulma TCP başlığında yer alan "kontrol toplamı değeri" alanından (TCP header checksum) tespit edilir. Alınan paketlerdeki kontrol toplam alanı (checksum) geçerli değilse paketler önemsenmez ve kullanıma konulmaz.

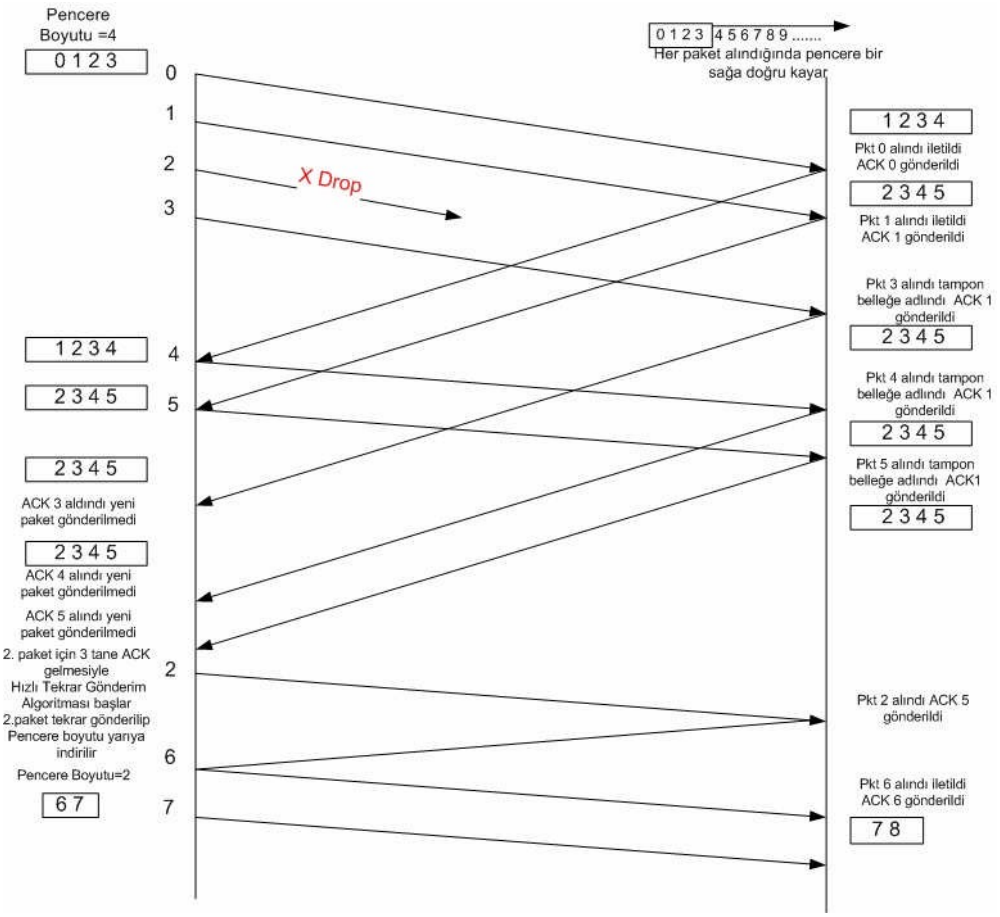
Akış Kontrolü

İletim katmanı veri gönderirken güvenlikli olduğu için hiç veri kaybolmaz. Alıcı kullanıcı veriyi aldığında güvenlik protokolleri nedeniyle veriyi hemen işleyemez. TCP

kullanılarak gönderilen veride bunlar olurken UDP kullandığımız takdirde daha az güvenlik ihtiyacı olan veriler gönderilir ve güvenlik protokolleri kullanılmadığı için alıcı veriyi daha kolay işleyerek kullanıcını üzerindeki yükü azaltır ve daha hızlı veri transferi sağlar.

TCP protokolü, veri parçalarını alan ve gönderen bilgisayarlar CPU hızı ve ağ bant genişliği gibi nedenlerden dolayı farklı hızlarda çalışabilirler. Bu nedenle, veri gönderen bilgisayarın karşı tarafın baş edemeyeceği hızda bilgi akışı sağlaması olasıdır. TCP akış kontrolü mekanizması ile gönderilen verinin miktarını kontrol eder. Protokol, kayan pencere (sliding window) tekniği kullanarak bağlantıda olan bilgisayarların senkronize olarak veri alışverişi yapmasını sağlar. TCP protokolünde akacak olan her veri oktetler (8 bitlik gruplar) halinde numaralandırılır. Oktetlere verilen her numaraya dizi numarası (sequence number) denir.

Alıcı, gelen verileri aldıktan sonra karşı tarafa bilgi paketi ile beraber, kabul edebileceği büyüklükteki dizi (sequence) numarasını gönderir. Kabul edilebilir dizi (sequence) numarası aralığına pencere (window) denir. Pencere, karşı taraftan gönderme izni alındıktan sonra göndericinin iletebileceği oktet sayısını belirler. Böylece gönderici verilerin alındığına dair bilgi mesajı almadan belirtilen miktarda veri transfer edebilir. Bu da protokolün veri iletim hızını artırır.

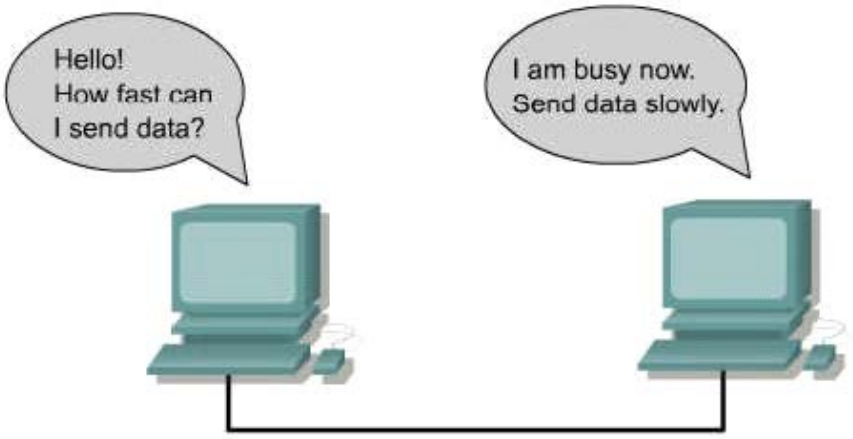


Şekil 1.6: Pencereleme

TCP veri akış kontrol mekanizması şöyle özetlenebilir:

- Pencere alanı dışında solda yer alan oktetler gönderilmiş ve cevap alınmış oktetlerdir.
- Pencere alanı içinde yer alan oktetler beklenilmeden yollanabilecek oktetlerdir.
- Pencere içindeki oktetlerin bir bölümü gönderilmiş ve yanıt bekliyor olabilir. Diğer oktetler ise gönderilmeyi bekleyen oktetlerden oluşabilir.
- Pencere alanı dışında sağ tarafta yer alan oktetler pencere alanı içine girmediği sürece gönderilmeyecek olan oktetlerdir.

- Pencere alanı, alıcının alınacak olan veri için ayırdığı bellek (buffer) büyüklüğünü bildirir. Eğer bellek dolarsa alıcı daha küçük pencere alanı ayırır. Bazı durumlarda gönderenin sadece bir oktet gönderilmesinin istenmesi de olasıdır. Bu duruma " silly window syndrome" denir.



Şekil 1.7: Akış kontrolü

Servis Saldırıları

Bilgisayar ağlarında, istenmeyen bir bilgisayar, ağ iletişimini kullanarak bilgisayar hafızasını ele geçirebilir. Servis hareketlerinin reddedilmesi (DoS), sunucuların iletişim kurma çabalarını reddetmek amacı ile dizayn edilmiştir. DoS (Denial Of Services) hareketlerinin genel yöntemi hackerların sistem cevaplarından yararlanmaktır. Bu senkronizasyon taşması olarak da bilinir. DoS açılımı Denial Of Services olan bu saldırı çeşidi bir hizmet aksatma yöntemidir. Bir kişinin sisteme arka arkaya yaptığı saldırılar sonucunda hedef sistemin hiç kimseye hizmet veremez hâle gelmesi veya o sisteme ait tüm kaynakların tüketimini amaçlayan bir saldırı çeşididir.

DoS saldırılarında saldırgan, senkronizasyonu başlatır. Fakat kaynak adresini kandırırlar. Kandırmalar, alım sürecinde kullanılarak, mevcut olmayan cihaz yanıtları, ulaşılamayan IP adresleri gibi sorunlar oluşturur. Ondan sonra olayı başlatan kimsenin onay verilerinin bitmesi için durum-bekle politikası uygulanır. Gerçek olmayan aygıtın bir şeye cevap verdiği dönemdir, ulaşılamayan bir adrestir. Başlangıçtan en son onaylama olmayı beklerken bekleme durumunda konumlanır. Bekleme isteği kuyruk bağlantısında ve hafızadaki tutma bölgesinde konumlanır. Bu bekleme durumu, aygıtın hafıza gibi sistem kaynaklarına saldırmasını sağlar. Bağlantı süresi bitene kadar bekleme işlemine devam eder. Saldırganlar sahte senkronizasyon ile sunucuya saldırıları başlatacaklardır. Tekrar kaynaklara bağlanarak sahte bağlantılar için beklerler.

Bu saldırılara karşı korunmak için sistem yöneticileri zaman dışı iletişim sürecini artırabilirler ve iletişim kuyruk büyüklüğünü artırabilirler. Ayrıca bu tip saldırılardan korunmak için ve savunmaya yönelik ölçümü başlatmak için yazılımlar mevcuttur. Bu tür yazılımlara Firewall (Güvenlik Duvarı) adı verilmektedir. Masaüstü bilgisayarlara kurulan güvenlik duvarı, bilgisayara İnternet ve/veya yerel ağ üzerinden gelen ve bilgisayardan İnternet'e ve/veya yerel ağa gönderilen paketleri, kendi tanım dosyasında bulunan güvenlik sorunu oluşturan paketlerle karşılaştırır ve bunlar arasından sorunlu olanları kullanıcıya haber verir. Güvenlik duvarı, dışardan bilgisayara gelen ve bilgisayardan dışarıya giden tüm paketleri inceler. Güvenlik duvarı sayesinde o anki TCP/IP/UDP gelen giden paketleri izleyebilme, saldırganın IP'sini görme gibi bilgilere ulaşılmasını sağlar. Firewall'un güzel bir tarafı da bilgisayarımızın dışarıyla/yerel ağla haberleşmesini sağlayan portları kontrol etmeyi sağlamasıdır. Nette kullanmayacağımız, işimize yaramayacak portları Firewall ile kapatmalıyız. Saldırgan, sistemlere saldıracağında öncelikle port taraması yöntemini kullanacaktır. Sizin açık portlarınız, eğer kapatmadıysanız gözükecek ve saldırgan bu port üzerinden size bağlanmaya çalışacaktır. Günümüzde Zone Alarm, MCAfee, Kaspersky, Norton İnternet Security gibi birçok firewall programı kullanılmaktadır. Bunun dışında Windows XP işletim sisteminin içinde de bir güvenlik duvarı mevcuttur.

Bir de DDoS (Distributed Denial Of Services) saldırı çeşidi vardır. Bir saldırgan daha önceden tasarladığı birçok makine üzerinden hedef bilgisayara saldırı yaparak hedef sistemin kimseye hizmet veremez hâle gelmesini amaçlayan saldırı çeşididir. Koordineli olarak yapılan bu işlem, hem saldırının boyutunu artırır hem de saldırıyı yapan kişinin gizlenmesini sağlar. Bu işlemleri yapan araçlara zombi denir. Bu saldırı çeşidinde saldırganı bulmak zorlaşır. Çünkü saldırının merkezinde bulunan saldırgan, aslında saldırıya katılmaz. Sadece diğer IP numaralarını yönlendirir. Eğer sadece tek bir IP adresinden yapılırsa bir Firewall bunu rahatlıkla engelleyebilir. Fakat saldırının daha yüksek sayıdaki IP adresinden gelmesi Firewall'ın devre dışı kalmasını sağlar. İşte bu özelliği onu DoS saldırısından ayıran en önemli özelliğidir.

Şimdi bilgisayarımızı bir ağ ortamına dâhil ederek önce TCP/IP yükleyelim, ardından güvenlik duvarı ayarını yapalım. Daha önceki derslerinizde de gördüğünüz gibi bilgisayarımızı ağ ortamına dâhil etmek için Ethernet kartı, switch ya da hub, cat5 kablo yardımı ile ağa bağlayabiliriz.

Basit TCP/IP Hizmetlerini yüklemek için

- Denetim Masası'nda Program Ekle/Kaldır'ı açın.
- Windows Bileşenlerini Ekle/Kaldır'ı tıklatın.
- Bileşenler'den Ağ Hizmetleri'ni ve sonra da Ayrıntılar'ı tıklatın.
- Ağ Hizmetleri'nde, Basit TCP/IP Hizmetleri'ni seçin ve Tamam'ı tıklatın.
- İleri'yi tıklatın.
- İstenirse, dağıtım dosyalarının bulunduğu yolu yazın ve sonra Tamam'ı tıklatın.
- Son'u sonra da Kapat'ı tıklatın.

UDP (User Datagram Protocol)

UDP, TCP / IP protokol grubunun iki aktarım katmanı protokolünden birisidir. Gelişmiş bilgisayar ağlarında paket anahtarlamalı bilgisayar iletişiminde bir data gram modu oluşturabilmek için UDP protokolü yazılmıştır. Bu protokol minimum protokol mekanizmasıyla bir uygulama programından diğerine mesaj göndermek için bir

prosedür içerir. Bu protokol 'transaction' yönlendirmelidir.

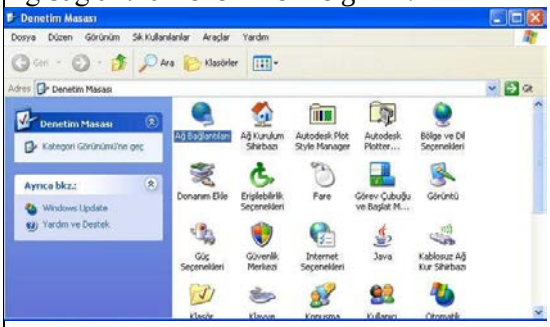
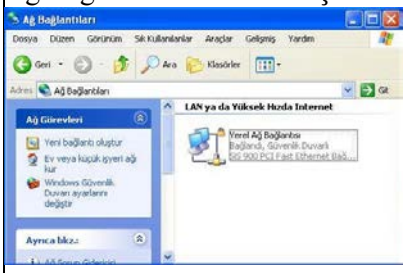
- Geniş alan ağlarında (WAN) ses ve görüntü aktarımı gibi gerçek zamanlı veri aktarımlarında UDP kullanılır.
- UDP bağlantı kurulum işlemlerini, akış kontrolü ve tekrar iletim işlemlerini yapmayarak veri iletim süresini en aza indirir.
- UDP ve TCP aynı iletişim yolunu kullandıklarında UDP ile yapılan geçek zamanlı veri transferinin servis kalitesi TCP'nin oluşturduğu yüksek veri trafiği nedeniyle azalır.

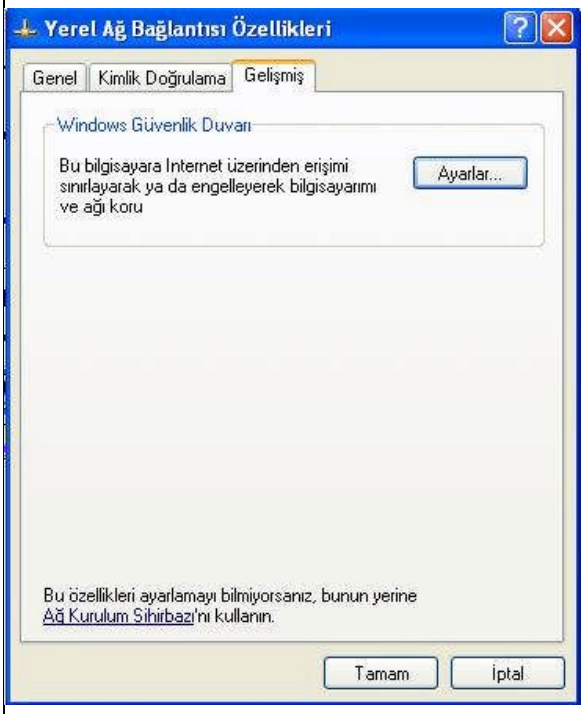
UDP'yi kullanan genel protokoller DNS, TFTP, ARP, RARP ve SNMP protokolleridir. Uygulama programcıları birçok zaman UDP'yi TCP'ye tercih eder. Çünkü ağ üzerinde fazla bant genişliği kaplamaz. UDP güvenilir olmayan bir aktarım protokolüdür. UDP protokolü ağ üzerinden paketi gönderir ve gidip gitmediğini takip etmez. Paketin yerine ulaşıp ulaşmayacağına onay verme yetkisi yoktur. UDP protokolü basit bir protokol olduğu için hızlı iletişim kurmamız gereken yerlerde kullanmamız yararımıza olacaktır. Buradaki basitlikten kasıt TCP protokolü gibi verinin gönderilmesi gibi kontrolleri içermediği içindir. UDP protokolünü kullanan programlara örnek olarak 161 nu.lu portu kullanan SNMP servisini verebiliriz.

Şekil- 8: UDP segment formatı

UDP datagramların belirli sıralara konmasının gerekli olmadığı uygulamalarda kullanılmak üzere dizayn edilmiştir. TCP'de olduğu gibi UDP'de de bir başlık vardır. Ağ yazılımı bu UDP başlığını iletilecek bilginin başına koyar. Ardından UDP bu bilgiyi IP katmanına yollar. IP katmanı, kendi başlık bilgisini ve protokol numarasını yerleştirir (bu sefer protokol numarası alanına UDP' ye ait değer yazılır). Fakat UDP, TCP'nin yaptıklarının hepsini yapmaz. Bilgi burada datagramlara bölünmez ve yollanan paketlerin kaydı tutulmaz. UDP'nin tek sağladığı port numarasıdır. Böylece pek çok program UDP'yi kullanabilir. Daha az bilgi içerdiği için doğal olarak UDP başlığı TCP başlığına göre daha kısadır. Başlık, kaynak ve varış port numaraları ile kontrol toplamını içeren tüm bilgidir.

Deney

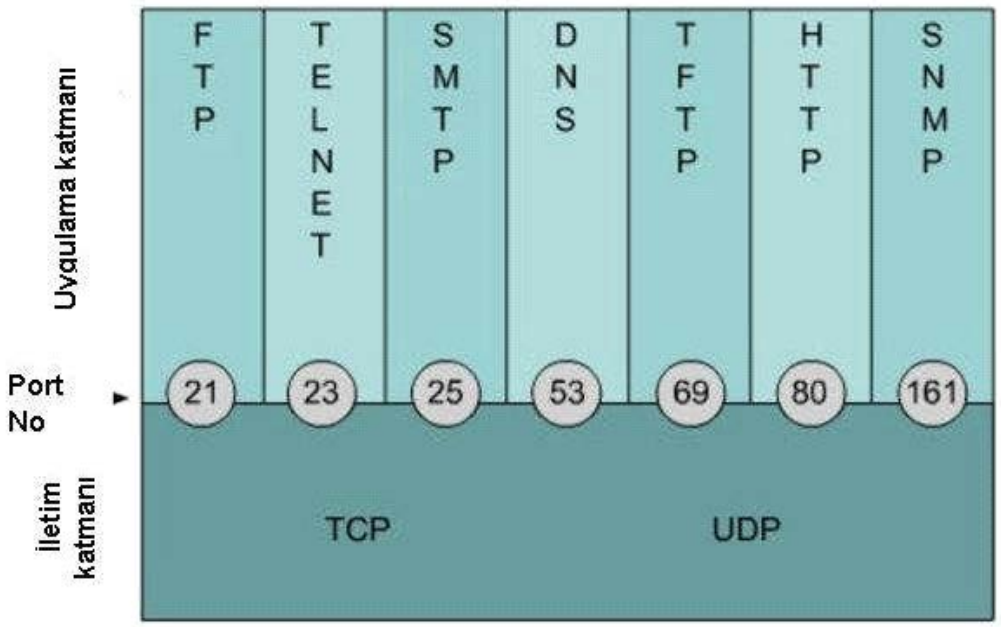
İşlem Basamakları	Öneriler
Ağ bağlantıları özelliklerine girilir. 	Denetim Masasından Ağ Bağlantılarına tıklayarak gelebilirsiniz.
Ağ Bağlantılarıma fare ile çift tıklanır. 	
Yerel Ağ Bağlantısı seçeneğine gelindiğinde, fare sağ tuş ile özellikler seçeneğine tıklanır.	Özellikler sekmesine tıklandığında yandaki resimde görünen ekran karşımıza gelecektir. Aynı işlemi **Denetim Masasından Güvenlik Duvarı** seçeneğinden de yapabilirsiniz.

Buradan Gelişmiş Kısmına tıklayınız.  	Bilgisayarımız internete bağlandığında istediğimiz programlara; internet üzerinden erişim verip istediğimizi sınırlandırabiliriz. Bu işlemi başka bir firewall programı kullanarak da yapabilirsiniz.
Ayarlar kısmına tıkladığımızda Windows Güvenlik Duvarı ile ilgili seçenekler karşımıza gelecektir.  	Buradaki ayarları yaparken **Açık** işaretlenmesi tavsiye edilir. Böylece bilgisayarımızda herhangi bir firewall programı olmasa da kısmen güvenliliğini sağlamış oluruz.

İletim Katmanı Servisleri

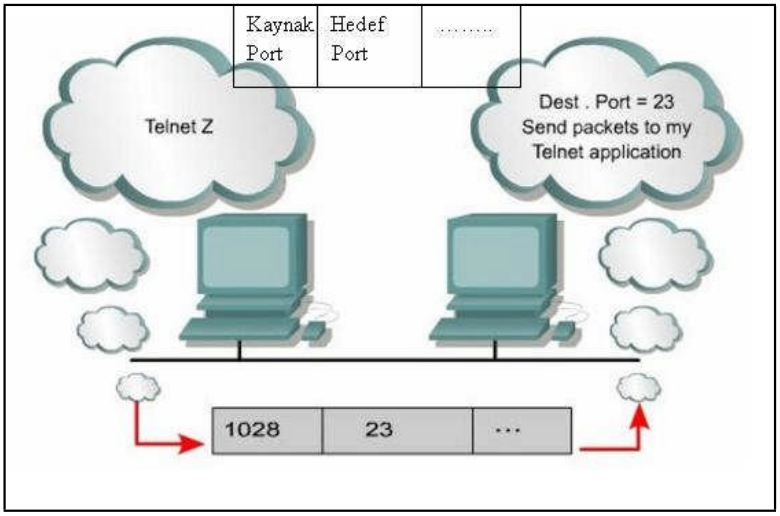
TCP protokolü, tek bir makine üzerinden birçok TCP servisi sunulmasını sağlar. Bu sayede ağlarda binlerce bilgi paketi, yüzlerce birbirinden farklı servise taşınmaktadır. Genellikle sunucular, paketlerin adreslenmesi için adreslerde çok az problem yaratan çapraz servisler kullanır. Eğer sunucu SMTP ve WWW'nin her ikisini çalıştırıyorsa, o servisin istediği bölgeyi belirlemede kullanılır. Sadece sunucu IP adresleri için paket oluşturulamaz. Çünkü hedef, servisin ne olduğunu bilemez. Port numaraları, sunucular arasındaki diyalog ile ilişkilendirilmiş olmalıdır. Sunucuda uygun servislerde paket ulaştırılabilir. İstemcinin e- mail, web sayfalarını gözlemleme gibi işlemleri kısa bir sürede bir sunucu kullanarak yapmaya gücü yetmez. İletim katmanı uygulamaları için ayrılmış metotlar kullanılmalıdır.

İletim katmanında sunucular, TCP/IP ile ilişkilendirilmiş portları çalıştırırlar. Aynı zamanda ağda farklı çapraz diyalogları yakalamak için port numaraları kullanılır. Port numaralarına, sunucu çoklu servis çalıştıran sunucularla iletişime geçtiği zaman ihtiyaç duyulur. TCP ve UDP'nin ikisi de port ve soket numaralarını üst katmanın bilgilerine geçmek için kullanır. FTP (File Transfer Protocol-Dosya Aktarım Protokolü) uygulamasını standart olarak 21 numaralı porttan kullanır.



Şekil 1.1:TCP port numaraları

Veri alışverişi uygulamaları ile iyi bilinen port numaralarını karıştırmazlar. Özel bir sıra içerisinden karışık olarak seçilmişlerdir. Port numaraları TCP parçasında kaynak ve hedef adresleri kullanır.



Şekil 2.2: TCP veri iletimi

Port numaraları aşağıdaki görev dizisindedirler:

- 255'ten aşağı numaralar halka açık uygulamalar için ayrılmıştır.
- 255t'ten 1023 e kadar olan numaralar satılabilir uygulamalar için şirketlere ayrılmıştır.
- 1023'ten yukarı düzenlenmemiştir.

Son sistemler uygun uygulamalarda seçilen port numaralarını kullanır. Kaynak port numaraları sunucu tarafından dinamikleştirilir. Genellikle 1023'ten büyük numaralardır. Port numaraları 0-1023 arasında ise Internet Numara Yetkilendirme Dairesi (IANA) tarafından kontrol edilir. Posta ofisi kutu numaraları, port numaraları için iyi bir benzetmedir. Mesajın bir kısmı posta şehir koduna, şehre ve posta kutusuna gönderilebilir. Şehir kodu ve şehir, posta kutusuna mektubun doğru bir şekilde gelmesini sağlar. Posta kutusu, mektubun adreslendiği yere ulaşmasını sağlarken, posta ve şehir kodu genel mesaj seklinde yollanır. Posta numaraları için iyi bir ön sıralamadır. Mesajın bir kısmi posta, şehir koduna gönderilebilir. Posta kutusu mailin adreslendiği yere ulaşmasını sağlarken, posta ve şehir kodu genel mesaj seklinde yollanır. Benzer olarak IP adresleri doğru server'a gönderilir fakat TCP ve UDP numaraları paketlerin doğru başvuruya geçtiğini garantiler. Benzer olarak IP adresleri paketleri doğrudan sunucuya gönderir. Fakat TCP ya da UDP port numaraları paketlerin doğru uygulamaya garantili bir şekilde geçmesini sağlar.

Servis Portları

Port numaraları, servisler sunucularda çalışırken iletişimde bulunabilmeleri için gereklidir. Servisler uzaktaki sunucuya bağlantı istedikleri zaman iletim katmanı protokolü ve portları kullanmak isteyecektir. Bazı portlar RFC 1700'ün içinde tanımlıdır. TCP ve UDP her ikisinin içerisinde saklanmış iyi bilinen portlardır

Decimal	Keyword	Description
0		Reserved
1-4		Unassigned
5	RJE	Remote Job Entry
7	ECHO	Echo
9	DISCARD	Discard
11	USERS	Active Users
13	DAYTIME	Daytime
15	NETSTAT	Who is Up or NETSTAT
17	QUOTE	Quote of the day
19	CHARGEN	Character Generator
20	FTP-DATA	File Transfer Protocol (data)
21	FTP	File Transfer Protocol
23	TELNET	Terminal Connection
25	SMTP	Simple Mail Transfer Protocol
37	TIME	Time of Day
39	RLP	Resource Location Protocol

Şekil 2.3: Servis portları

Çok sık kullanılan portlar uygulamalarda tanımlanmıştır. İletim katmanı protokollerinin üstünde çalışabilir. Örnek verecek olursak; sunucular FTP servisini kullanırken TCP bağlantılarını 20.portu kullanarak iletirler ve 21.porttan FTP uygulamalarını gerçekleştirirler. Alışveriş başladığında yolda sunucu, uzaktaki kullanıcının hangi servisi kullanmak istediğine karar verir. TCP ve UDP iletimde doğru servise karar vermek için port numaralarını kullanırlar.

Decimal	Keyword	Description
42	NAMESERVER	Host Name Server
43	NICNAME	Who Is
53	DOMAIN	Domain Name Server
67	BOOTPS	Bootstrap ProtocoalServer
68	BOOTPC	Bootstrap Protocol Client
69	TFTP	Trivial File Transfer Protocol
75		Any Private Dial-out Service
77		Any Private RJE Service
79	FINGER	Finger
80	HTTP	HyperText Transfer Protocol
95	SUPDUP	SUPDUP Protocol
101	HOSTNAME	NIC Host Name Server
102	ISO-TSAP	ISO-TSAP
110	POP3	Post Office Prorotol for client to retrieve mails from mail server
113	AUTH	Authentication Service
117	UUCP-PATH	UUCP Path Service

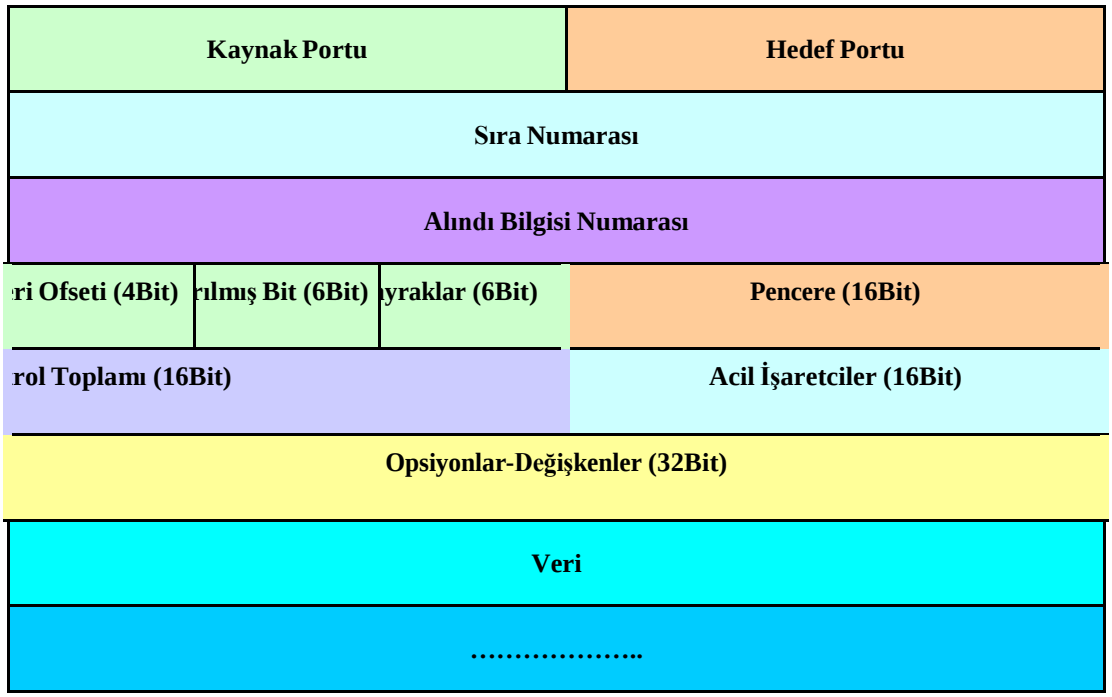
Şekil 2.4: Servis portları

İstemci Portları

İstemciler, sunuculardaki servislere bağlanmak durumunda kaldığında kaynak ve hedef portları belirtmek zorundadır. TCP ve UDP parçaları kaynak ve hedef portları belirtmek için alan bulundurur. Hedef portları ya da servisin herkesin bildiği çok kullanılan portlar tanımlanır.

İstemciler genelde kaynak portları 1023'den yukarı olan karışık olarak seçerek

tanımlar. Mesela; istemci iletişim kurmak istediğinde web sunucusu ile TCP'yi kullanırken 80.portu kullanır ve kaynak portu ise 1045'tir. Paket sunucudan vardığında iletim katmanından geçmiş demektir. Sonunda http servisi ile 80.portta işletilir. http sunucu istemcilere 80.portu kullanarak cevap verir. Hedef olarak 1045 portu kullanılır. Alışveriş yapılırken yolda sunucular ve servisler ilişkilendirildikleri portları kullanırlar.



Tablo 2.1:TCP'de gönderilen Bilgi Paketi

Port Numaraları

TCP katmanı içindeki veya dışındaki herhangi bir communication circuit (iletişim merkezi) iki numaranın birleşmesinden oluşan socket numaraları tarafından tanımlanır. Bu numaralar makinenin IP adresi ve TCP yazılımı tarafından kullanılan port numarasıdır.

Port numaraları, TCP ve UDP parçaların çerçevelerinde 2 bayt ile ifade edilir. Bu 16 bit değerine denk gelmektedir. Port numaraları 0'dan 65535'e kadar değişmektedir. Aralarında iletişim olan makinelerde korunan bir port tablosu vardır. Bu tabloda iletişimde bulunan makinelerin kaynak ve hedef port numaraları karşılıklıdır. Port numaraları üç farklı kategoriye bölünmüştür:

- İyi bilinen portlar: İlk 1023 port en çok bilinen portlardır. Ağ servislerinde iyi bilindikleri için bu isim kullanılmıştır.FTP, Telnet ya da DNS gibi.
- Kayıtlı portlar: 1024'ten 49151'e kadar kayıtlı portlardır.
- Dinamik ya da özel portlar: 49152 ile 65535 arasındaki portlardır.

Ağ Servisi	PORT NO
FTP veri transferi	TCP Port 20
FTP kontrol	TCP Port 21
Telnet	TCP Port 23
SMTP	TCP Port 25
DNS	UDP Port 53
http	TCP/UDP Port 80
POP3	TCP Port 110
SHTTP	TCP/UDP Port 443

Tablo 2.2: TCP port numaraları

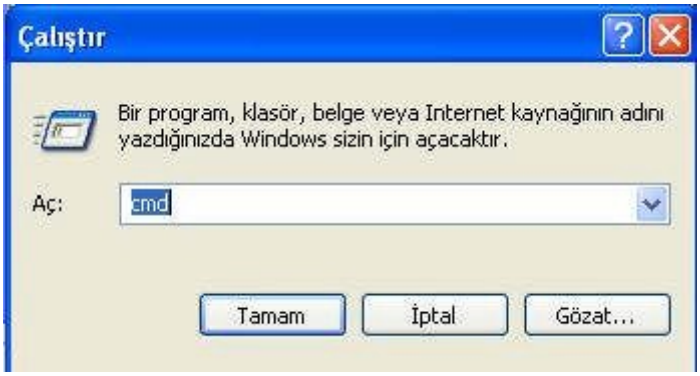
Port numaraları sunucular arasındaki çoklu oturumlar ortaya çıkabildiği için kullanılırlar. Kaynak ve hedef port numaraları soketten ağ adresleri ile bütünleşiktir. Soket çiftleri her sunucuda bir tanedir. Örneğin bir telnet bağlantısı için port 23'tür. Bazı zamanlar Net'te gezerken port 80 olabilir. IP ve MAC adresleri aynı olabilir. Çünkü paketler aynı sunucudan gelebilir. Bu yüzden diyalog kaynakta kendi port numarasına ihtiyaç duyabilir. Her server yanıtlarken kendi port numarasına ihtiyaç duyabilir.

Örneğin bir mektupta adresler isim, sokak, şehir ve ülkeyi barındırır. Ağ verisi için port, MAC ve IP adresleri kullanarak karşılaştırma yapabiliriz. Port numarası ismi, Mac adresi sokak adresini, şehir ve ülke adresini de IP adresi olarak düşünelim. Çoklu mektuplar sokak adreslerine, şehir ya da ülkeye gönderilebilir. Fakat mektuplardaki alıcıların isimleri farklıdır. Örneğin elimizde bir adrese Baki SAKALLI ve Fevzi SAKALLI adına iki mektup gönderilmiş olsun. Bunu farklı port numaralarına benzetebiliriz. Bilgisayarımız herhangi bir ağa bağlandığında etkin TCP bağlantılarını, bilgisayarın bağlı olduğu bağlantı noktalarını (port) görebilmemiz mümkündür. Bunu birtakım programlar yapabildiği gibi cmd (dos ekranı) de "netstat" komutuyla da görebiliriz. Komut satırında yürütülen "netstat" komutu etkin TCP bağlantılarını, bilgisayarın bağlı olduğu bağlantı noktalarını, Ethernet istatistiklerini, IP yönlendirme tablosunu, IP, ICMP, TCP ve UDP iletişim kuralları için IPv4 istatistikleri ile IPv6, ICMPv6, IPv6 üzerinden TCP ve IPv6 iletişim kuralları üzerinden UDP için IPv6 istatistiklerini görüntüler. Parametreler olmadan kullanılan "netstat", etkin TCP bağlantılarını görüntüler.

Şimdi netstat komutunu kullanarak bilgisayarımızdaki etkin TCP bağlantılarını görüntüleyelim.

Aktif portları tespit etmek için;

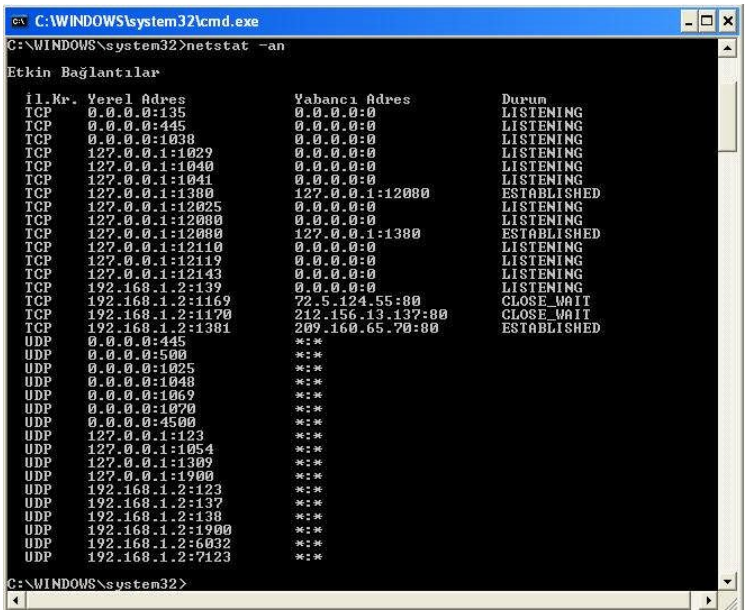
- Öncelikle bilgisayarımızın Başlat seçeneğinden Çalıştır komutunu tıklıyoruz.
- Dos komut istemcisini açmak için cmd yazıyoruz ve tamam seçeneğinii tıklıyoruz.



Çalıştır ekranı

Karşımıza gelen ekrana “netstat –an” ya da “netstat” komutunu yazıyoruz.

Aynı işlemi herhangi bir firewall ya da IP numarası izleyen programlar (xns5,ipscan vs.) yardımı ile yapmak mümkündür.



“netstat” komutunun uygulanması

Burada;

- "Proto" ("İl.Kr.") başlığı altındaki karakterler ilgili port için kullanılan protokol tipini gösterir.
- "Local Address" (Yerel Adres) ise bilgisayarınızın ağ üzerindeki isminin yanı sıra gelen bağlantıları kabul ettiğiniz ve rastgele üretilen port numarasını gösterir.
- "Foreign Address" (Yabancı Adres) kısmı ise uzak bilgisayarın adını ve bağlantıyı gerçekleştirmek için kullandığı port numarasını gösterir.
- Adından da anlaşılacağı üzere "State" (Durum) bağlantının durumunu gösterir.

- Bu başlık altında görülebilecek durumlar şunlardır:
- ESTABLISHED - İki bilgisayar da bağlı.
- CLOSING - Uzak bilgisayar bağlantıyı kapatmaya karar vermiş.
- LISTENING - Bilgisayarınız gelen bir bağlantı isteği için bekliyor.

- SYN_RECV - Uzak bir bilgisayar bağlantı isteğinde bulunmuş.
- SYN_SENT - Bilgisayarınız bağlantı isteğini kabul etmiş.
- LAST_ACK - Bilgisayarınız bağlantıyı kapatmadan önce paketleri siliyor.
- CLOSE_WAIT - Uzak bilgisayar bilgisayarınızla olan bağlantıyı kapatıyor.
- FIN_WAIT 1 - Bir istemci bağlantıyı kapatıyor.
- FIN_WAIT 2 -İki bilgisayar da bağlantıyı kapatmaya karar vermiş.

Komutun Parametreleri

Netstat [-a] [-e] [-n] [-o] [-p Protokol] [-r] [-s] [Aralık]

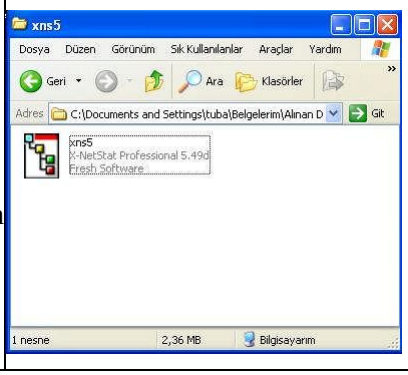
-a
Tüm etkin TCP bağlantılarıyla birlikte, bilgisayarın bağlı olduğu TCP ve UDP bağlantı noktalarını görüntüler.
-e
Gönderilen ve alınan bit ve paket sayısı gibi, Ethernet istatistiklerini görüntüler. Bu parametre -s ile birleştirilebilir.
-n
Etkin TCP bağlantılarını görüntüler. Ancak adresler ve bağlantı noktası numaraları sayısal olarak ifade edilir, herhangi bir ad konulmaz.
-o
Etkin TCP bağlantılarını görüntüler ve her bağlantının işlem kimliğini (PID) içerir. Uygulamaları PID'e göre bulmak istiyorsanız, Windows Görev Yöneticisi içindeki İşlemler sekmesine bakın. Bu parametre -a , -n ve -p ile birleştirilebilir.
-p İletişim Kuralı
İletişim Kuralı tarafından belirlenmiş iletişim kuralı bağlantılarını gösterir. Bu durumda İletişim Kuralı; tcp, udp, tcpv6 veya udpv6 olabilir. Bu parametre, iletişim kuralına göre istatistikleri görüntülemek üzere -s ile birlikte kullanılırsa, İletişim Kuralı; tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6 veya ipv6 olabilir.

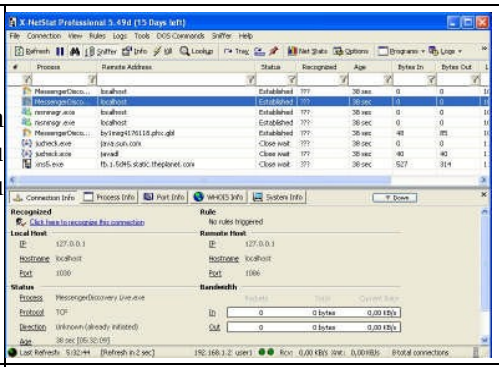
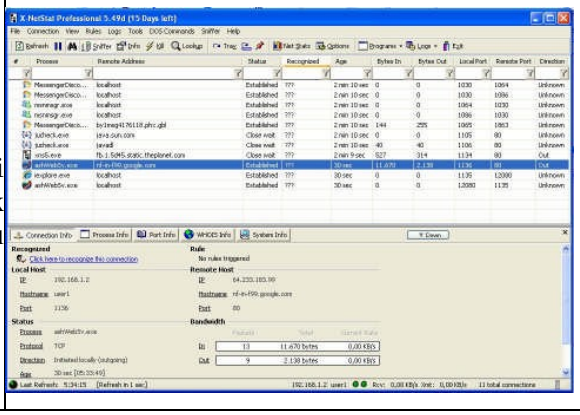
IP yönlendirme tablosunun içeriğini görüntüler. Bu, "route print" komutu ile eş değerdir.

Bu uygulamayı başka bir program kullanarak gerçekleştirelim:

Bilgisayarınıza herhangi bir port kontrol programı kurunuz. Burada kullanılan program “xns5” adında bir IP izleyici programdır.

Aktif portları tespit etmek

Bu tür programları İnternetten bulabilirsiniz. Programı çalıştıralım.	

Programı çalıştırdığımızda ekrana bilgisayarımızda açık olan programlar ve bu programların hangi portları kullandıkları görülmektedir.	
Program aracılığıyla hangi programın hangi portu kullandığı yerel ağda ve uzak bağlantısında aldığı IP numarası görülebilmektedir.	

PIC Mikrodenetleyiciler

Giriş

PIC mikrodenetleyiciler orta ve küçük ölçekli endüstriyel kontrol devrelerinde kullanılmak üzere Microchip tarafından geliştirilmiş yüksek performanslı, ucuz, CMOS, tam statik, 8-bit mikrodenetleyicilerdir. RISC mimarisi ile üretilmiş olan bu ailede, 8 seviyeli yığın, çoklu iç ve dış kesme kaynakları gibi genişletilmiş çekirdek yapıya sahiptir. Komut ve veri için farklı iki yola sahip olması nedeniyle aynı saat periyodunda her iki bellek alanı kullanılabilir. Toplam 35 temel komuta sahip olması öğrenilmesini kolaylaştırır. Bu az sayıdaki komutla yetenekli program yazılabilmesi için yeterli sayıda yazaç tümdevre üzerine yerleştirilmiştir.

PIC CPU'lar 4 farklı çekirdekte üretilir.

- 12 bit çekirdek 33 komut: 12C50x, 16C5x
- 14 bit çekirdek 35 komut: 12C67x,16Xxxx
- 16 bit çekirdek 58 komut: 17X4x,17X7xx
- 'Gelişmiş' 16 bit çekirdek 77 komut: 18xxxx

PIC16 mikrodenetleyici ailesi komutları eşdeğeri diğer 8 bit mikrodenetleyicilere göre 2:1 oranında sıkıştırılırken, komut işleme hızı iki kat arttırılmıştır. PIC16 ailesi mikrodenetleyicilerde dışarıdan bağlanması gereken eleman sayısı en aza indirgenmiş ve bu sayede sistem güvenirliği artırılmış ve güç tüketimi azaltılmıştır.

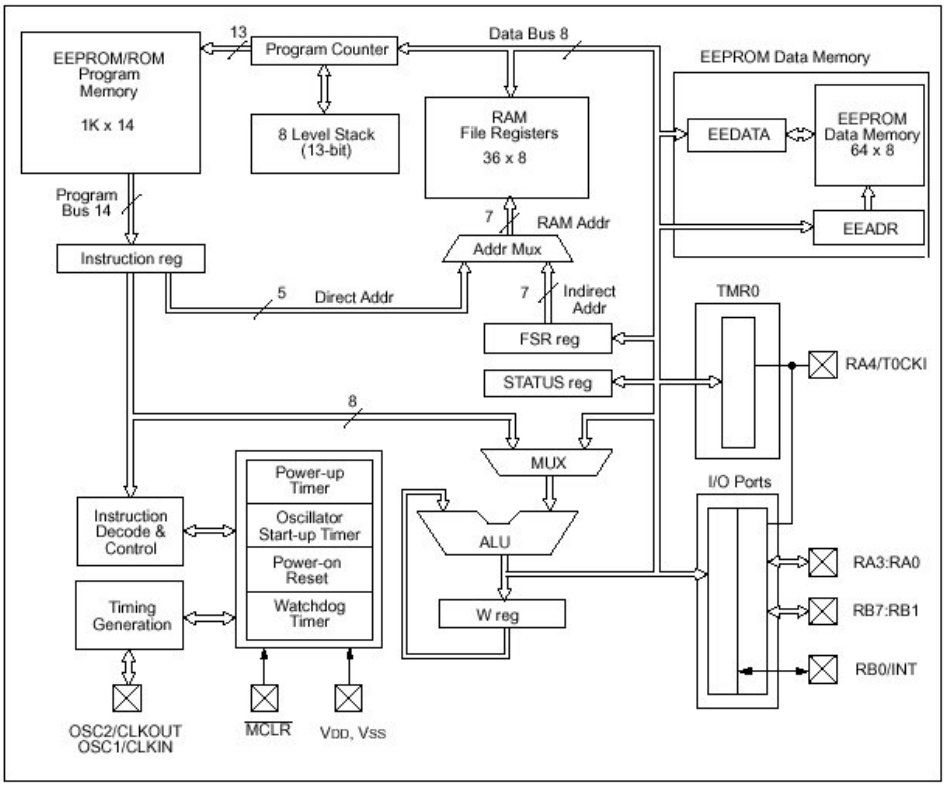
Tablo-1'de PIC16C84'ün içerdiği birimler verilmiştir. Bu mikrodenetleyici yüksek hızlı otomotiv ve elektrikli araçların motor denetim devrelerinden düşük güç isteyen uzaktan denetimli sensör devrelerine kadar geniş bir kullanım alanına sahiptir. Ayrıca elektronik kilitlerde, akıllı kartlarda, güvenlik elemanlarında (ev ve araba hırsız alarm devrelerinde) da kullanılır. EEPROM teknolojisi kontrol programının kullanıcı tarafından kendine göre düzenlemesini olanaklı kıldığı için (örneğin transmitter kodlarının çözülmesi saklanması, motor hızının kullanıcı tarafından belirlenmesi ve ölçülen hıza göre programın işletilmesi, receiver frekansının ayarlanması, güvenlik kodlarının saklanması ve daha sonra kullanıcıya göre değiştirilmesine izin vermesi) yaygın kullanım alanına sahiptir. Boyutlarının küçük olması alan sorunu olan uygulamalarda tercih edilmesinin sebebidir. Düşük güç harcaması, ucuz olması, giriş/çıkış hatlarının amaca yönelik ayarlanabilmesi bugüne kadar mikrodenetleyici kullanılması düşünülmemiş uygulamalarda bile kullanılmasını sağlamıştır (örneğin PWM, zamanlayıcı, değer yakalama, karşılaştırma, seri iletişim gibi).

	Program Belleği			Veri Belleği	
İşlemci	Flash	EEPROM	ROM	RAM	EEPROM
PIC16C84		1K		36	64
PIC16F84	1K			68	64
PIC16CR84			1K	68	64
PIC16F83	512			36	64
PIC16CR83			512	36	64

Tablo-1. PIC16X8X ailesi mikrodenetleyiciler.

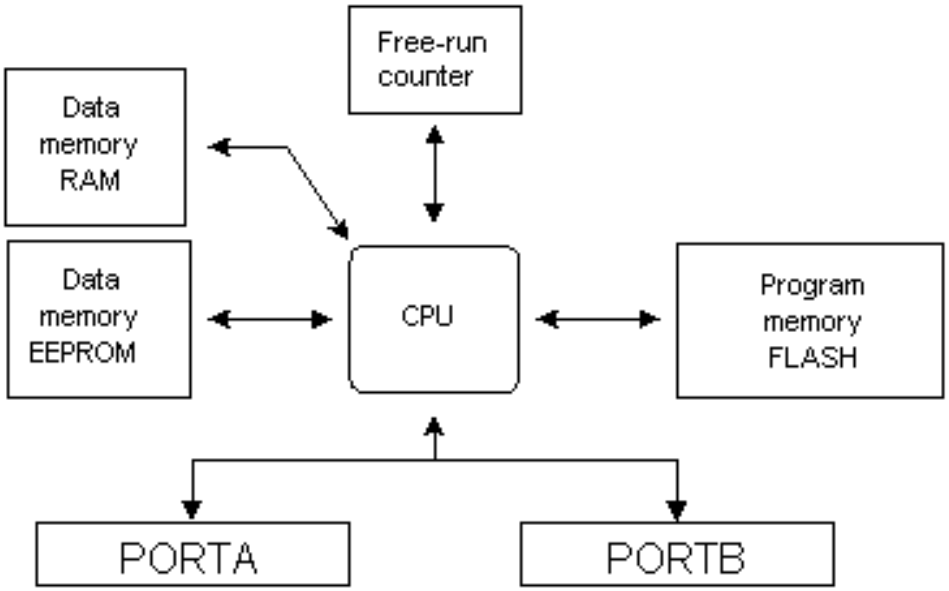
PIC16X8X'in Yapısı

PIC16CXX ailesinin yüksek performansının kaynağı tüm RISK tipi yapıya sahip mikroişlemcilerde de bulunan yapısal ayrıntılara dayanmaktadır. PIC16CXX Harvard mimarisinde yapılandırılmıştır, bu yapıda işlemci veri ve program için iki farklı bellek kullanır. Dolayısıyla iki farklı yola sahiptir, veri belleği yolu ve program belleği yolu. Bu özellik geleneksel yöntem olan Neumann mimari yapısında üretilen (veri belleği ve program belleği aynı yolu kullanır) işlemcilere göre bant genişliğini arttırmıştır. Program ve veri belleklerini ve yollarını ayırmak işlem kodu boyutunu veri uzunluğundan daha büyük boyutta yapılabilmesini olanak sağlar. PIC16CXX'nin işlem kodu uzunluğu 14 bittir, bir işlem saykılında 14 bit işlenir. İki konumlu pipeline komut getirme ve komut işleme işlemlerinin üst üste binmesini yani aynı anda yapılmasını sağlar. Bu mimariyi kullanmayan işlemcilerde önce komut program belleğinden getirilir ve sonra MİB içerisinde komutun kodu çözülerek işlenir. İşlenme kısmı ancak getirme işleminin bitmesinden sonra yapılır. PIC16CXX mikrodenetleyicisi tüm yazaçları ve veri belleğini doğrudan veya dolaylı olarak adresleyebilir. Tüm özel amaçlı yazaçlar veri belleği içerisinde de yer alır. PIC16C84 36x8 SRAM ve 64X8 EEPROM veri belleğine sahiptir.

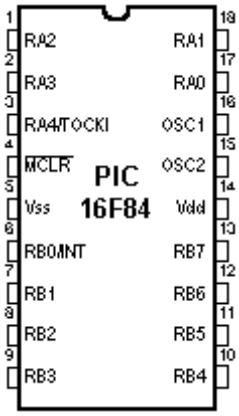


Şekil-1 PIC16C84'dün iç yapısı.

PIC16XXX ailesi mikrodenetleyiciler 8 bit ALU ve Work yazacına (Working register) sahiptir. ALU'nun görevi aritmetik ve mantık işlemlerini yapmaktır. İşlem yapılacak birinci veri çalışma yazacında ikinci veri diğer yazaçlardan herhangi birinde olmalıdır. Work yazacının (W) adresi yoktur. PIC16C84'dün iç blok diyagramı şekil-8.1'de verilmiştir. Bacak numaraları ve isimleri ise şekil-8.2'de verilmiştir. tablo-8.2'de her hattın görevleri açıklanmıştır.



Şekil-2 PIC16X84'ün blok şeması.



Şekil-3 PIC16F84'ün bacak numaraları ve isimleri.

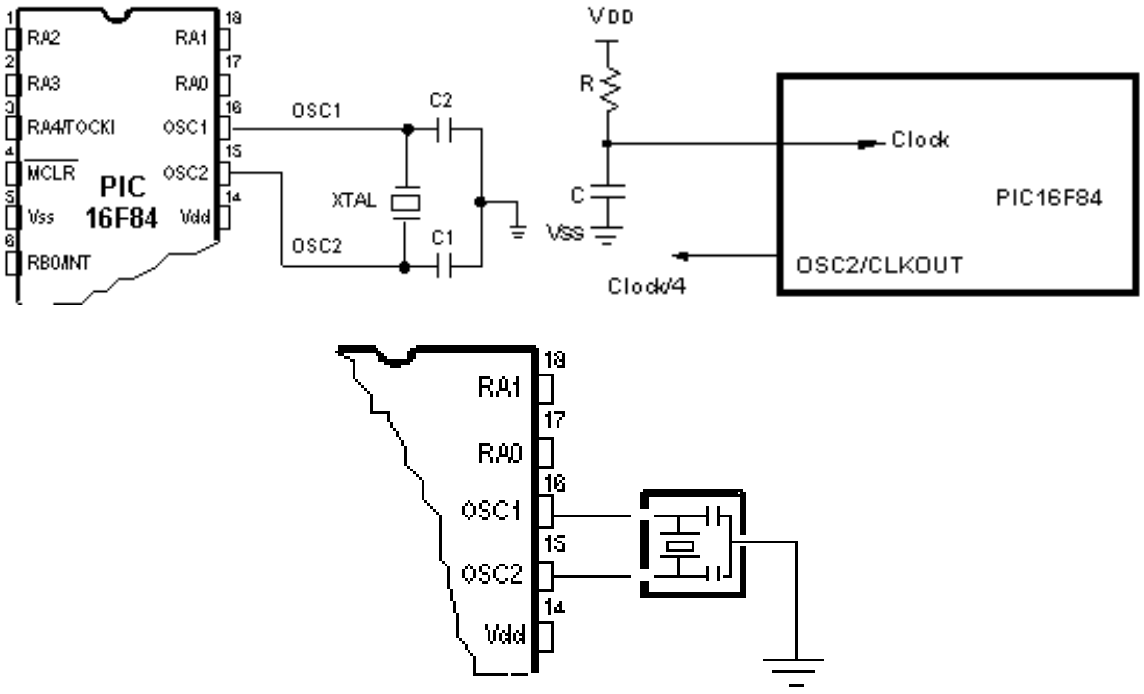
Saat Devresi

Dört çeşit osilatör devresi bağlanabilir, RC osilatör en ucuz çözümü sunarken, LP osilatör güç tüketimini azaltır, XT standart kristal osilatördür ve HS yüksek hızlı kristal osilatör devresidir. OSC1 saat girişinden verilen işaret içeride yüksek seviyeleri birbiri üzerine binmeyecek şekilde Q_1, Q_2, Q_3, Q_4 olarak adlandırılan dört farklı saat işareti üretilir. Her Q_1 etkin olduğunda program sayacı bir artırılır, daha sonra komut program belleğinden getirilir ve Q_4 etkin olduğunda komut yazacı tarafında tutulmaya başlanır. Komutun bundan sonraki Q_1 - Q_4 saykıllarında kodu çözülür ve işletilir. Şekil-8.5'te saat işaretleri gösterilmiştir.

Adı	Buffer	Açıklama
OSC1/CLKIN	ST	Kristal girişi/osilatör girişi.
OSC2/CLKOUT	----	Kristal osilatör modu seçildiğinde kristal veya rezonatörün ikinci bacağına bağlanır. RC osilatör modunda ise osilatör frekansının dörde bölünmüş hali bu uçtan çıkış olarak verilir.
MCLR	ST	Normal çalışmada reset, programa sırasında programlama geriliminin uygulandığı uçtur.

RA0	TTL	Port A Giriş ve çıkış hattı.
RA1	TTL	Port A Giriş ve çıkış hattı.
RA2	TTL	Port A Giriş ve çıkış hattı.
RA3	TTL	Port A Giriş ve çıkış hattı.
RA4/T0CKI	ST	Port A Giriş ve çıkış hattı ve TMR0'ın dış tetiklemegirişi.
RB0/INT	TTL	Port B Giriş ve çıkış hattı ve dış kesme girişi
RB1	TTL	Port B Giriş ve çıkış hattı.
RB2	TTL	Port B Giriş ve çıkış hattı.
RB3	TTL	Port B Giriş ve çıkış hattı.
RB4	TTL	Port B Giriş ve çıkış hattı.
RB5	TTL	Port B Giriş ve çıkış hattı.
RB6	TTL/ST	Port B Giriş ve çıkış hattı.
RB7	TTL/ST	Port A Giriş ve çıkış hattı.
VSS	----	Ortak uç.
VDD	----	Artı besleme.

Tablo-2 PIC16C84'ün bacak isimleri ve görevleri.



Şekil-5 PIC'lerin osilatör bağlantıları, Kristal, RC, rezonatör bağlantısı.

Komut işleme hızı = 1/4 osilatör hızı

Tcyc = 4 * Tclk

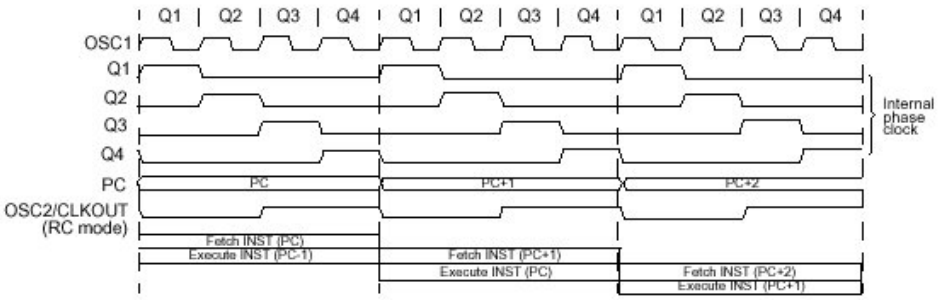
Tüm PIC'ler DC'den belirlenen maksimum hızlarına kadar çalışırlar. Aşağıda PIC mikrodenetleyicilerin çalışabilecekleri en yüksek çalışma frekansları verilmiştir.

12C50x 4MHz

12C67x	10MHz
16Cxxx	20MHz
17C4x / 17C7xxx	33MHz
18Cxxx	40MHz

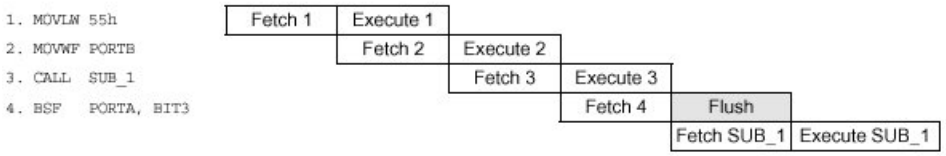
Komut işleme sırası

Bir komut işleme saykılı dört Q saykılından oluşur. Komut getirme ve yürütme olarak pipeline yapılmıştır, getirme işlemi bir işlem saykılında yapılır, onu takip eden işlem saykılında bu komut yürütülür. Pipeline sayesinde dallanma ve bağlanmaların dışında tüm komutlar bir işlem saykılında yürütülür.



Şekil-6 Saat işaretleri ve komut getirme ve yürütme saykılları.

Dallanma ve bağlanma komutlarında program sayacının içeriği değişmesi gerektiğinden artı bir işlem saykılına gereksinim duyulur. Getirme saykılı Q_1'in yükselen kenarında program sayacının arttırılması ile başlar. Yürütme saykılı getirilen komutun Q_1 işareti ile komut yazacına tutulması ile başlar. Bu komutun Q_2, Q_3, Q_4 işaretleri ile kodu çözülür ve yürütülür. Veri belleği Q_2 işareti ile işleneni almak için okunur, Q_4 işareti ile sonucu yerleştirmek için yazılır. Şekil-8.6'da komut işleme saykılı örnek programla gösterilmiştir.



Şekil-7 komut pipeline akışı.

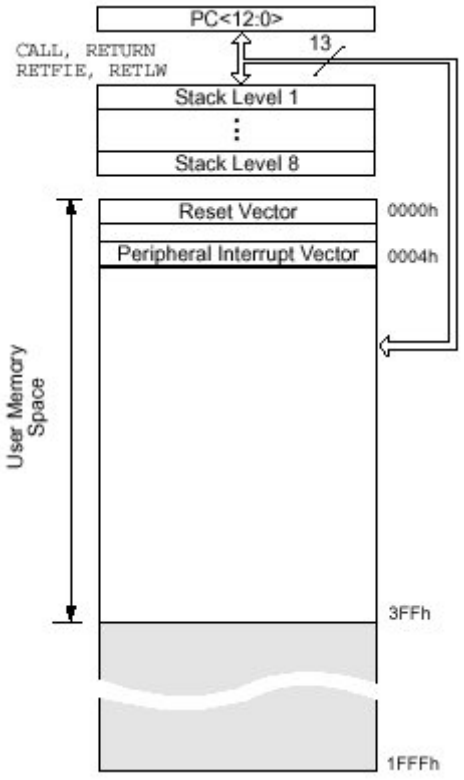
Bellek Yapısı

PIC16C84'te iki türlü bellek yer almaktadır. Veri belleği ve program belleği, her iki bellek kendi yollarına sahip olduklarından aynı işlem saykılında her iki bellekten işlem yapılabilir. Veri belleği genel amaçlı RAM ve özel amaçlı yazaçlar (special function register) olmak üzere iki kısma ayrılır. Özel amaçlı yazaçların görevi çevre birimlerinin çalışma şekillerini ve şartlarını belirlemektir. Bazı SFR'ler çevre elemanının yaptığı işlemin sonucu gösterir. Veri belleği içerisinde EEPROM veri belleği olan bir kısım vardır. Bu kısma sadece dolaylı adresleme modu ile ulaşılabilir. 64 bayt boyutunda olan EEPROM belleğin adresi 00h-03h aralığıdır. Ayrıntıları ileriki bölümlerde incelenecektir.

Program Belleğinin İşleyişi

PIC16XXX 13 bit program sayacı ile 8 K x 14 bit belleği adresleyebilir. 0000h-03FFh adres aralığı fiziksel olarak kullanılmıştır. Kullanılmayan 0400h-FFFFh aralığı ise fiziksel olarak yer

almadığından bu alanlar okunduğunda kullanılan alanların simetriği komutlara ulaşılacaktır. Reset vektörü 0000h adresinde, kesme vektörü 0004h adresinde yer alır.



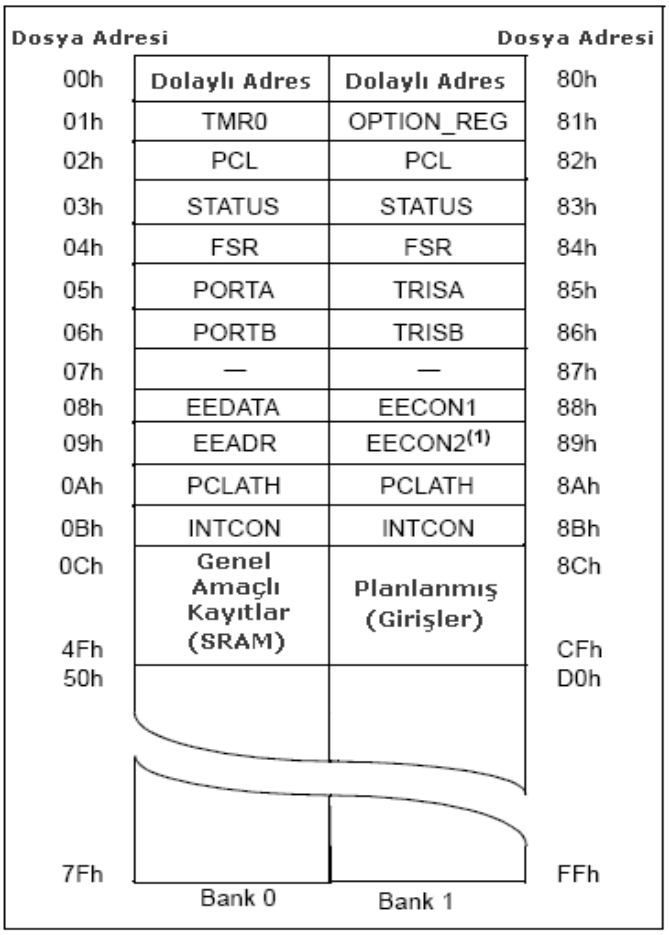
Şekil-8 Program belleği ve yığın yapısı.

Veri Belleğinin İşleyişi

Veri belleği iki kısma ayrılmıştır, özel amaçlı yazaçlar (SFR) ve genel amaçlı (GPR) yazaçlar. Şekil-8.6'da yazaçların bellekte yerleri verilmiştir. Veri belleğinin her iki kısmı parçalı olarak banklara ayrılmıştır. Banklara ayrılmasının nedeni daha fazla sayıda veri belleğini adresleyebilmektir. Bank seçme işlemi durum yazacı içerisinde yer alan bank seçme bitleri ile yapılır. Kullanılmak istenilen veri belleği satırına her doğrudan adresi kullanılarak yada dosya seçme yazacı (FSR) kullanılarak dolaylı ulaşılabilir. Dolaylı adresleme banklara ulaşmak için RP1 ve RP0 bitlerinin o anki içeriklerini kullanır.

Veri belleği özel amaçlı yazaçları ve genel amaçlı yazaçları kapsayan iki kısımdan oluşur. Bank 0 RP0 biti temizlenerek, Bank 1 RP0 biti kurularak seçilir. Her iki banka 7Fh adresine kadar gider, her bankanın ilk 12 satırı özel amaçlı yazaçlara ayrılmıştır. Genel amaçlı yazaçların olduğu kısım statik RAM'dan yapılmıştır. Bu alana doğrudan veya FSR yazacı kullanılarak dolaylı olarak ulaşılabilir. İşlemlerde elde edilen ara değerleri geçici süre ile saklamak için kullanılır. Bank 1'deki genel amaçlı yazaçlar bank 0'da adreslidir. Örneğin 0Ch ile 8Ch adresleri fiziksel olarak

Özel amaçlı yazaçlar CPU veya çevre birimleri tarafından çevre elemanlarının çalışma koşullarını belirlemek veya sonuçları gözlemlemek amacı ile kullanılırlar. Bu yazaçlar iki ana gruba ayrılabilir, birinci grupta çekirdek yapıyı oluşturan yazaçlar yer alırken ikinci grupta çevre birimlerini denetleyen veya durumlarını gösteren yazaçlar yer almaktadır. Bu bölümde çekirdek yapıyı oluşturan özel amaçlı yazaçlar incelenecektir. İkinci gruba giren özel amaçlı yazaçların görevleri denetlediği birimler incelenirken açıklanacaktır.



Şekil-9 Veri belleğinin yapısı.

Durum Yazacı

Durum yazacı ALU'nun yaptığı aritmetik işlemlerin durumunu, reset durumunu ve veri belleği için banka seçme bitlerini içerir. Diğer yazaçlar gibi durum yazacı da komutların yazma hedefi olabilir, eğer komut işlendikten sonra Z, C, DC bitleri etkilenecek ise bu bitler yazma korumalı hale gelirler. Bu bitleri bu tür komutlarda sadece ALU değiştirebilir. Ayrıca /TO ve /PD bitleri yazmaya karşı kapalı sadece okunabilir bitler olduğu için işlemsonunda olmasını istediğimiz değer durum yazacında olmayabilir. Sadece BCF, BSF, SWAPF ve MOVWF komutları durum yazacı içeriğini istenilen şekilde değiştirilmesini olanak sağlar. Bu komutlar durum yazacı içerisindeki Z, C, DC bitlerini etkilemezler. Tablo-8.4'te durum yazacı bitleri ve görevleri verilmiştir.

Toplama işleminde C, DC bayrakları elde eldeleri gösterirken aynı bayraklar çıkarma işleminde işleminde borç bayrakları olarak çalışırlar.

Her bankta 128 bayt yer alır, 16x8x ailesinde sadece bank 1 kullanılmıştır RP1 temizlenmelidir.

Uyku modu işlemcinin güç tüketimini azaltır, programcı işlemciyi iç veya dış kesmelerden birini veya sistem resetini kullanarak uyandırabilir. Yüksek güvenirlikli RC osilatörü ile tümdevre üzerinde yer alan Watchdog zamanlayıcı yazılım kilitlenmeleri için bir koruma işlemi yapar.

Dolaylı adreslemede bank seçme biti 16X8X ailesinde kullanılmamıştır. reset sonrası 0 değerini alır bu değeri değiştirilmemelidir.

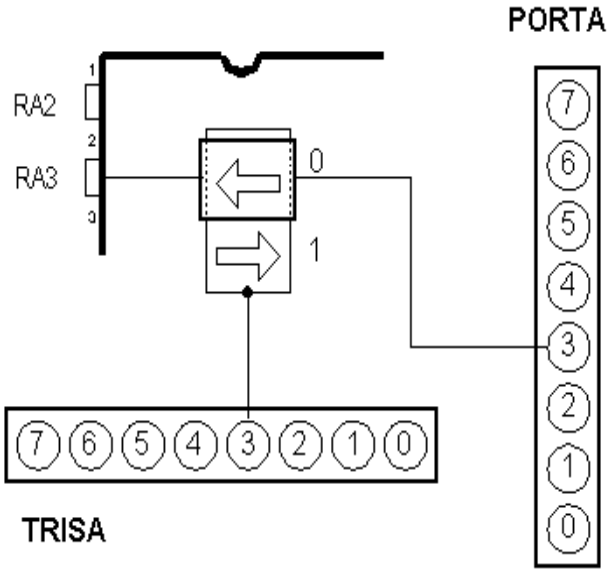
PortA, PortB, TrisA,TrisB

PortA ve PortB giriş çıkış portlarıdır. PortA 5 adet giriş çıkış hattına, PortB ise 8 adet giriş/çıkış hattına sahiptir. TrisA ve TrisB yazaçları ise PortA ve PortB hatlarını yönlendirmesini yapar.

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7							bit0

Bit No	Adı	Görevi
Bit 7	IRP	Dolaylı adreslemede bank seçme biti.
Bit 6-5	RP1 RP0	Doğrudan adreslemede bank seçme bitleri. 00: Bank 0 (00h-7Fh) 01: Bank 1 (80h-FFh) 10: Bank 2 (100h-17Fh) 11: Bank 3 (180h-1FFh)
Bit 4	TO	WD Zamanlayıcısı taşma bayrağı. Güç verildikten ve CLRWDT veya SLEEP komutları işletildikten sonra 1, WDT zaman aşımı gerçekleştiğinde 0 olur.
Bit 3	PD	Kısık güçte çalışma biti. Güç verildikten sonra veya CLRWDT işletildikten sonra 1, SLEEP komutu işletildiğinde 0 olur.
Bit 2	Z	Sıfır bayrağı. Aritmetik veya mantık işlemin sonucu sıfır ise 1, değilse 0 olur.
Bit 1	DC	Sayı elde bayrağı İKO sayıların toplamasında veya çıkarılmasında kullanılır. Dördüncü bitlerin toplanmasından elde var ise 1, yok ise 0 olur.
Bit 0	C	Elde/Borç bayrağı. En yüksek değerli bitlerin toplanmasından elde oluşur ise 1 oluşmaz ise 0 olur.

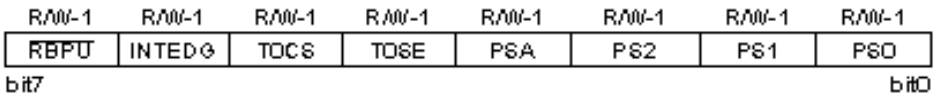
Tablo-4 durum yazacının bitlerinin görevleri.



Şekil-9 Port A ve TRISA'nın bağlantısı.

Option Yazacı

Option yazacına zamanlayıcıların, dış kesme ve PORTB'nin yükseğe çekme dirençlerinin çlaışmasını düzenleyen görevler verilmiştir.



PS0-PS2 ön bölme bitleri, zamanlayıcı veya WDT'ın girişine gelen tetikleme işaretinin bölme miktarını belirlerler.

PSA (Prescaler Assignment) biti hangi zamanlayıcının ön bölme devresini kullanacağını belirler. 1 olduğunda WDT, 0 olduğunda TMR0.

T0SE, (TMR0 Source Edge Select bit) biti, zamanlayıcı dıştetikleme girişi kenarını seçer. 1 ise düşen kenar, 0 ise yükselen kenarda zamanlayıcı tetiklenir.

T0CS (TMR0 Clock Source Select) zamanlayıcının tetikleme kaynağını seçer. 1 ise RA4'ten, 0 ise iç osilatörden tetiklenir.

INTDEG (Interrupt Edge Select bit) Interrupt'ın hangi kenarda algılanacağını seçer.1 yükselen kenar, 0 düşen kenar. PortB iç yükseğe çekme dirençlerinin kullanımını seçer. 1 kullanılmayacağını, 0 kullanılacağını belirtir.

Bit 2-0	TMR0	WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Tablo-5 Ön bölme oranları.

INTCON Yazacı

INTCON yazacı kesme izinleme bitlerini ve PIC16XXX'in ksemelerinin bayraklarını içerir. Genel kesme izinleme biti ile kesme kullanılıp kullanılmayacağı belirlenir. eğer kullanılacak ise hangi kesmelerin algılanacağı ilgili kesmenin kesme izin biti ile seçilir. izin verilen kesmenin oluşması durumunda o kesmenin bayrağı 1 olur.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit7							

RBIF (RB Port Change Interrupt Flag bit) RB 4, RB 5, RB 6 ve RB 7'den herhangi birindeki değişimin kesme olarak algılanıp algılanmayacağını belirler.

INTF (RB0/INT Interrupt Flag bit) Dış kesme bayrağıdır.

T0IF (TMR0 Overflow Interrupt flag bit) Zamanlayıcı zaman aşım bayrağıdır.

RBIE (Port Change Interrupt Enable bit) RB 4, RB 5, RB 6, RB 7 hatlarındaki değişimi kesme olarak algılama izin biti.

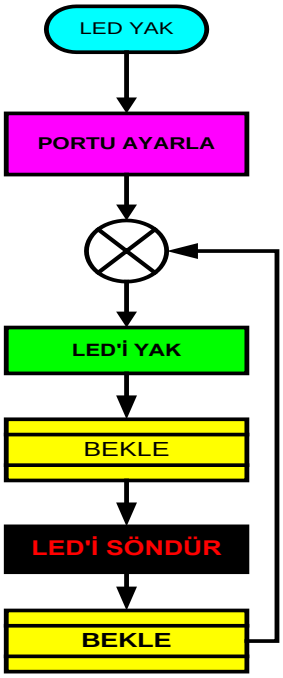
INTE (RB0/INT Interrupt Enable bit) Dış kesme izin biti.

T0IE (TMR0 Owerflow Interrupt Enable bit) Zamanlayıcı kesmesi izin biti.

GIE (Global Interrupt Enable bit) genel kesme izin biti.

Portların Kullanımı

Deney: RA1 hattına bağlı LED'i flash yapar.

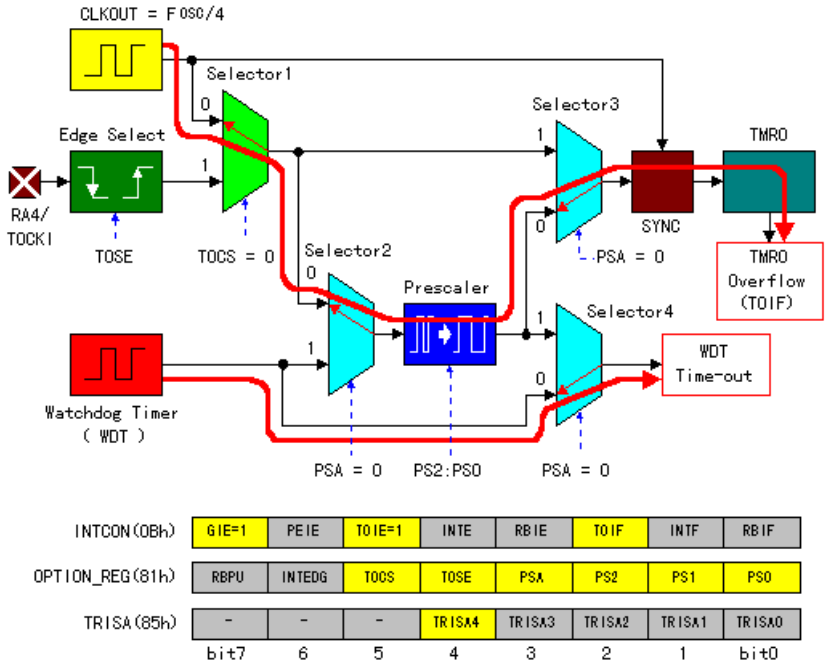



```
INCLUDE "P16F84.INC"
say1    equ     0x0C
say2    equ     0x0D
Ayar:
        bsf     SATUS,5       ;Bank 1'e geç
        movlw   0x00
        movwf   TRISA         ;Port A'nın hatlarını çıkış yap
```

```
        bcf   STATUS,5                  ; Bank 1'e geç
Yak:
        movlw  0x02
        movwf  PORTA                    ;LED'i yak
        call  gecik
        movlw  0x00
        movwf  PORTA                    ;LED'i söndür
        call  gecik
        goto  Yak               ;Sürekli yap
gecik:
        MOVLW 250
        MOVWF say2      ;Zaman geciktirmenin süresini ayarla.
gecik1:
        MOVLW 100
        MOVWF say1 ; Zaman geciktirmenin süresini ayarla.
gecik2:
        decfsz  say1,1          ;say 1 kadar beklet
        goto   gecik2
        decfsz say2,1           ;say1 * say2 kadar beklet
        goto  gecik1
        return
end
```

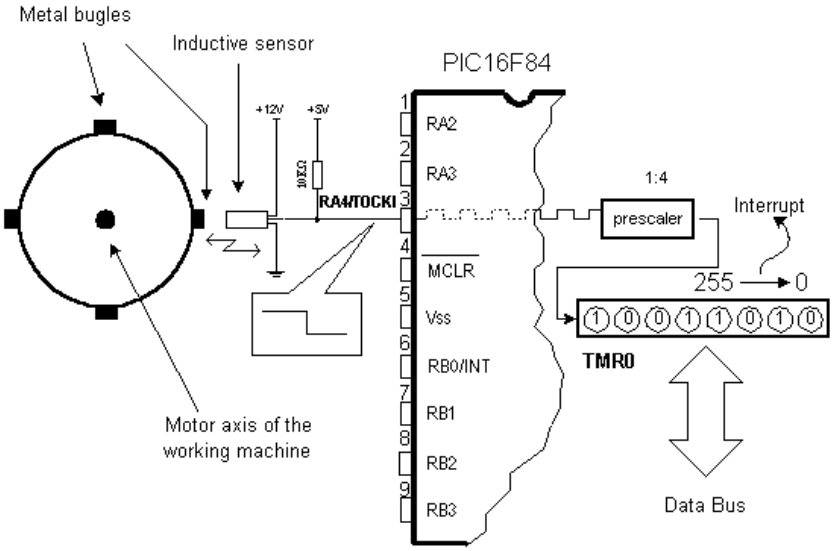
Deney: PB.0'dan Kare dalga üretmek.

```
INCLUDE "P16F84.INC"
DEL EQU 0CH         ;0Ch'ta DEL değişkeni tanımlandı.
SEL EQU 0DH                 ;0Dh'ta SEL değişkeni tanımlandı.
ANA:
        BSF STATUS, RP0    ;Bank1 seçildi.
        CLRF TRISB         ;PORTB çıkış
        BCF STATUS, RP0    ;Bank 0 dön
LED:
        BSF PORTB,0                 ;LED'i yak
        CALL bekle         ;Bekle
        BCF PORTB,0        ;LED'i söndür
        CALL bekle
        GOTO LED
bekle:
        MOVLW 250
        MOVWF DEL          ; DEL'i 250 yap
DEL1:
        MOVLW 100
        MOVWF SEL          ;W'deki sayı SEL değişkenine atandı.
DEL2:
        NOP
        DECFSZ SEL,F       ;SEL'i azalt, 0 ise alt satırı atla
        GOTO DEL2          ;DEL2'ye git.
        DECFSZ DEL,F       ;DEL'i  azalt, 0 ise alt satırı atla
        GOTO DEL1                   ;DEL1'e git.
        RETURN
END
```



Şekil-8 Zamanlayıcının yapısı

Deney: Tarama ile devir sayısının belirlenmesi




```
ORG 0x00
AYAR:
        Clrf TMR0           ;Başlangıç için temizle
        Clrf INTCON         ;Kesmeleri kapat
        Bsf STATUS,RP0      ;Bank 1'e geç
        MOVLW 00010000B     ;RA4 giriş
        MOVWF TRISA
        MovlW 00110001B
        Movwf OPTION        ;Ön bölme ¼, düşen kenarda say,
                            ;Port B iç yükseğe çekme dirençleri kullanılsın
T0_TEST:

        Btfss INTCON,TOIF   ;Taşma oldumu?
```

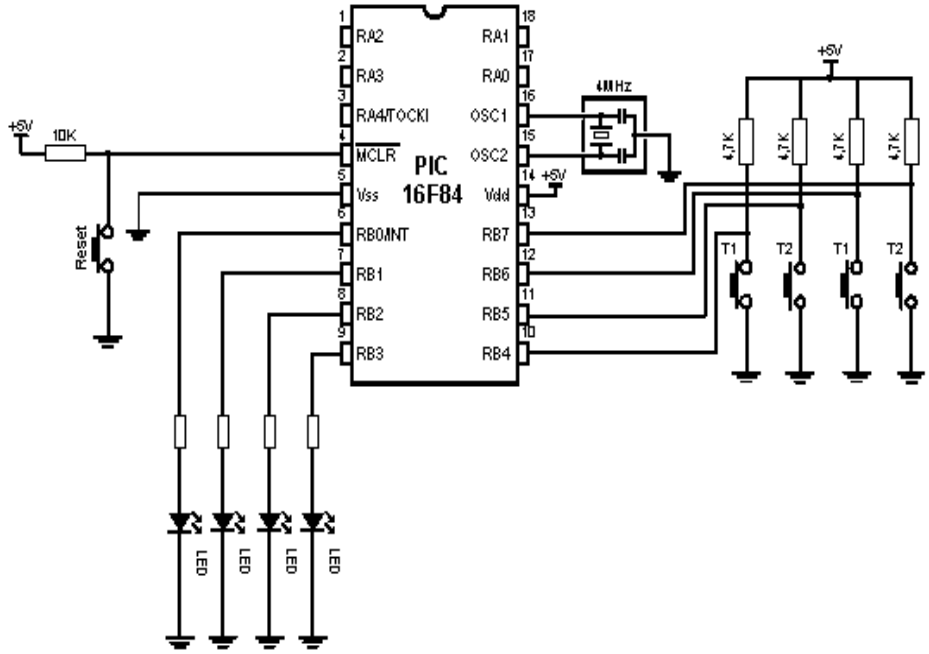
```
        Goto T0_TEST        Hayır
        Call GOSTER         ;Evet devir sayısını göster
        Goto T0_TEST        ;Devir sayısını saymaya devam et
end
```

PIC16F84'ün Kesmeleri

PIC16F84 4 adet kesme kaynağına sahiptir

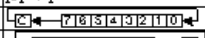
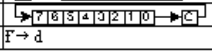
1. EEPROM'a veri yazma bitti
2. TMR0 zaman aşımı
3. 3.RB4, RB5, RB6 and RB7 hatlarında değişim var
4. Dış kesme, RB0/INT

Deney: Tuşlardan birine basıldığında aynı numaralı LED sönsün.




```
Org 0x00
        GOTO ANA
Org 0x04
        GOTO KSP
ANA:
        BSF STATUS,RP0      ;TRISB'yı seç
        MOVLW 0xf0          ;LED'ler çıkış, Tuslar giriş
        movwf TRISB
        BCF STATUS,RP0      ;PORTB'yi seç
        MOVLW 0x0f
        MOVLW PORTB                 ;LED'leri yak
        BSF INTCON,RBIE     ;PORTB değişim kesmesini izinle
        BSF INTCON,GIE      ;Kesme algılanmasına izin ver
BEKLE:
        GOTO BEKLE          ;Yapılacak başka iş yok burada bekle
KSP:
        BCF INTCON,RBIF     ;Başka kesme oluşmasına izin verme.
```

```
        BTFSS PORTB,7
        GOTO LED0
        BTFSS PORTB,6
        GOTO LED1
BTFSS PORTB,5
        GOTO LED2
        BTFSS PORTB,4
        GOTO LED3
        RETFIE          ;Kesme oluşmasına izin ver
LED0:
        BCF PORTB,0             ;LED0'ı söndür
        RETFIE          ;Kesme oluşmasına izin ver
LED1:
        BCF PORTB,1             ;LED1'i söndür
        RETFIE          ;Kesme oluşmasına izin ver
LED2:
        BCF PORTB,2             ;LED2'yi söndür
        RETFIE          ;Kesme oluşmasına izin ver
LED3:
        BCF PORTB,3             ;LED3'ü söndür
        RETFIE          ;Kesme oluşmasına izin ver
END
```

Mnemonic		Description	Operation	Flag	Cycle	Notes
		Data transfer				
MOVLW	k	Move constant to W	k → W		1	
MOVWF	f	Move W to f	W → f		1	
MOVF	f, d	Move f	f → d	Z	1	1,2
CLRW	-	Clear W	0 → W	Z	1	
CLRF	f	Clear f	0 → f	Z	1	2
SWAPF	f, d	Swap nibbles in f	f(7:4), (3:0) → f(3:0),(7:4)		1	1,2
		Arritmetic and logic				
ADDLW	k	Add constant and W	W+1 → W	C,DC,Z	1	
ADDWF	f, d	Add W and f	W+f→ d	C,DC,Z	1	1,2
SUBLW	k	Subtract W from constant	W-k → W	C,DC,Z	1	
SUBWF	f, d	Subtract W from f	W-f→ d	C,DC,Z	1	1,2
ANDLW	k	AND constant with W	W.AND.k→ W	Z	1	
ANDWF	f, d	AND W with f	W.AND.f→ d	Z	1	1,2
IORLW	k	OR constant with W	W.OR.k → W	Z	1	
IORWF	f, d	OR W with f	W.OR.f→ d	Z	1	1,2
XORLW	k	Exclusive OR constant with W	W.XOR.k→ W	Z	1	1,2
XORWF	f, d	Exclusive OR W with f	W.XOR.f→ d	Z	1	
INCF	f, d	Increment f	f+1 → f	Z	1	1,2
DECF	f, d	Decrement f	f-1 → f	Z	1	1,2
RLF	f, d	Rotate Left f trough carry		C	1	1,2
RRF	f, d	Rotate Right f trough carry		C	1	1,2
COMF	f, d	Complement f	f→ d	Z	1	1,2
		Bit operations				
BCF	f, b	Bit Clear f	0 → f(b)		1	1,2
BSF	f, b	Bit Set f	1 → f(b)		1	1,2
		Directing a program flow				
BTFSC	f, b	Bit Test f, Skip if Clear	jump if f(b)=0		1 (2)	3
BTFSS	f, b	Bit Test f, Skip if Set	jump if f(b)=1		1 (2)	3
DECFSZ	f, d	Decrement f, Skip if 0	f-1 → d, jump if Z=1		1(2)	1,2,3
INCFSZ	f, d	Increment f, Skip if 0	f+1 → d, jump ifZ=0		1(2)	1,2,3
GOTO	k	Go to address	W.AND.k→ W		2	
CALL	k	Call subroutine	W.AND.f→ d		2	
RETURN	-	Return from Subroutine	W.OR.k → W		2	
RETLW	k	Return with constant in W	W.OR.f→ d		2	
RETFIE	-	Return from interrupt	W.XOR.k→ W		2	
		Other instructions				
NOP	-	No Operation			1	
CLRWDT	-	Clear Watchdog Timer	0 → WDT,1↦TO,1 → PD	$\overline{TO}$,PD	1	
SLEEP	-	Go into standby mode	0 → WDT,1↦TO,0→ PD	$\overline{TO}$,PD	1	

EK- A: PIC Komut Kümesi

KULLANILAN KISALTMALAR

f:	Yazaç dosya adresi (0x00 to 0x7F)
W:	Working (accumulator) yazacı
b:	Bit adres, 8-bit dosya yazaç bölgesinde adresli
k:	Sabit veri veya etiket.
X:	Farketmez (= 0 or 1)
d:	hedef seçme biti; d = 0 ise, sonuç W yazacında, d = 1 ise sonuç f yazacının içeriğinde adresi belirtilen yazaçta saklanır. Başlangıçta d = 1'dir.
Label:	Etiket.
TOS:	Yığının üst noktası.
PC:	Program Sayacı
PCLATH:	Program sayacı YDB tutucusu.
GIE:	Genel kesme izinleme biti.
WDT:	Watchdog Timer/Counter
TO:	Time-out biti
PD:	Power-down biti
Dest:	hedef, (W yazacı veya adresi f' de yazılı yazaçlardan biri)
[ ]	Opsiyon
( )	içeriği
< >	yazaç bit alanı

ADDWF

W yazacı ile adresi belirtilen yazacın içeriklerini toplar, sonucu toplanan sayıların birinin bulunduğu yazaca yazar. Sonucun hangi yazaca yazılacağı komut içerisinde belirtilir.

Etkilenen bayraklar: C, DC, Z

İşletilme süresi: 1 makine saykılı.

ÖRNEK:

ADDWF 31h,w
W yazacının içeriği = 25h
RAM'in 31h nolu satırın içeriği = 60h

Olduğunu varsaydığımızda, komut işlendikten sonra yazaçların içerikleri aşağıdaki gibi olacaktır.

25h + 60h = 85h

RAM'in 31h nolu satırın içeriği hala = 60h

Sonuç W yazacında saklanır, W= 85h

C = 0 DC = 0 Z = 0

ADDLW

W yazacı ile adresi belirtilen sayıyı toplar, sonucu W yazacına yazar.

Etkilenen bayraklar: C, DC, Z

ÖRNEK:

ADDLW 14h

W yazacının içeriği = 25h

Olarak kabul edilirse;
25h + 14h = 39h
sonuç W yazacına yazılır, W= 39h
C = 0 DC = 0 Z = 0

ANDWF

W yazacı ile adresi belirtilen yazacın içeriklerini VE'ler, sonucu VE'lenen sayıların birinin bulunduğu yazaca yazar. Sonucun hangi yazaca yazılacağı komut içerisinde belirtilir.

Etkilenen bayraklar: Z

İşletilme süresi: 1 makine saykılı.

ÖRNEK:

ANDWF 32h,w

W yazacının içeriği = 50h 50h 0 1 0 1 0 0 0 0

RAM'in 31h nolu satırın = 45h 45h 0 1 0 0 1 0 0 1

Olduğunu varsayarsak;

50h Λ 45h = 40h 40h 0 1 0 0 0 0 0 0

RAM'in 30h nolu satırın içeriği hala = 45h

Sonuç W yazacında saklanır, W= 40h

Z = 0

ANDLW

W yazacının içeriği ile belirtilen sayıyı VE'ler, sonuç W yazacına yazılır.

Etkilenen bayraklar: Z

İşletilme süresi: 1 makine saykılı.

ÖRNEK:
ANDLW 45h
W yazacının içeriği = 50h 50h 0 1 0 1 0 0 0 0
Belirtilen sayı = 45h 45h 0 1 0 0 1 0 0 1

Olduğunu varsayarsak;
50h Λ 45h = 40h 40h 0 1 0 0 0 0 0 0
Sonuç W yazacında saklanır, W= 40h
Z = 0

BCF

Adresi belirtilen yazaçtaki numarası verilmiş bit temizlenir.

Etkilenen bayraklar: Hiç biri.

İşletilme süresi: 1 makine saykılı.

ÖRNEK:

BCF 13h,4
13h nolu RAM yazacın içeriği = 77h 0 1 1 1 0 1 1 1
4 nolu biti temizlendiğinde 0 1 1 0 0 1 1 1
Sonuç 13h nolu RAM yazaçta saklanır.

BSF

Adresi belirtilen yazaçtaki numarası verilmiş bit kurulur.

Etkilenen bayraklar: Hiç biri.

İşletilme süresi: 1 makine saykılı.

ÖRNEK:

BSF 13h,3
13h nolu RAM yazacın içeriği = 77h 0 1 1 1 0 1 1 1
3 nolu bitikurulduğunda 0 1 1 1 1 1 1 1
Sonuç 13h nolu RAM yazaçta saklanır.

BTFSC

Adresi belirtilen yazacın numarası verilmiş biti test edilir eğer sıfır ise bir sonraki komut NOP komutu gibi işletilir, değilse bir sonraki komut aynen işletilir. Bu komut diğer işlemcilerde bulunan koşullu dallanma komutu yerine kullanılır.

Etkilenen bayraklar: Hiç biri.

İşletilme süresi: test edilen bit 1 ise 1 makine saykılı, 0 ise 2 makine saykılı.

ÖRNEK:

BTFSC 15h,0
15h nolu RAM yazacın içeriği= 0 0 0 0 0 0 0 0 0
Bit 0 = 0
Bir sonraki komut NOP olarak işletilir.

BTFSS

Adresi belirtilen yazacın numarası verilmiş biti test edilir eğer 1 ise bir sonraki komut NOP komutu gibi işletilir, değilse bir sonraki komut aynen işletilir. Bu komut diğer işlemcilerde bulunan koşullu dallanma komutu yerine kullanılır.

Etkilenen bayraklar: Hiç biri.

İşletilme süresi: test edilen bit 0 ise 1 makine saykılı, 1 ise 2 makine saykılı.

ÖRNEK:

BTFSC 15h,0
15h nolu RAM yazacın içeriği= 0 0 0 0 0 0 0 0 0
Bit 0 = 0
Bir sonraki komut aynen işletilir.

CALL

Alt program çağırır. Öncelikle dönüş adresi (PC+1) yığına atılır. Program sayacına yüklenecek adresin 11 biti komutla birlikte verilir, geri kalan 2 bit ise PCLATH içerisinde alınır. Program yürütülmeye bu adresten devam edilir.

İŞLEV:

(PC)+1→ TOS

k → (PC <10:0>)

(PCLATH <4:3>) → (PC <12:11>)

Etkilenen bayraklar: Hiç biri.

İşletilme süresi: 2

ÖRNEK:

CALL TOPLA

TOPLA alt programı 100h nolu satırda yer almaktadır. Bu adres assembler tarafından hesaplanacak ve object programa yazılacaktır. Sizin sadece etiket kullanmanız yeterlidir. PIC16F84'te 1K uzunluğunda bellek yer aldığı için PCLATH içerisindeki iki bit hiç kullanılmayacaktır. Bu bitler ihmal edilebilir.

PCLATH içeriği = 00h

PCLATH		TOPLA
0 0 0 0 0 0 0 0		0 0 1 0 0 0 0 0 0 0 0
0 0	+	0 0 1 0 0 0 0 0 0 0 0
		0 0 0 0 1 0 0 0 0 0 0 0

Bir adım sonra işletilecek komutun adresi 0100h

CLRF

Belirtilen yazacın içeriğini temizler.

Etkilenen bayraklar: Z

İşletilme süresi: 1 makine saykılı.

ÖRNEK:

CLRF 24h

24h nolu RAM belleğin içeriği=0

Z = 1

CLRW

W yazacının içeriğini temizler.

Etkilenen bayraklar: Z

İşletilme süresi: 1 makine saykılı.

ÖRNEK:

CLRW

W yazacının içeriği=0

Z = 1

CLRWDT

WATCHDOG zamanlayıcısını sıfırlar. Bu komut ayrıca STATUS yazacında yer alan TO ve PD bitlerini kurar.

Etkilenen bayraklar: TO, PD

TO = 1

PD = 1

İşletilme süresi: 1

COMF

Belirtilen yazacın içeriğini 1'e tümler.

Etkilenen bayraklar: Z

İşletilme süresi: 1 makine saykılı.

ÖRNEK:

COMF 12h,f

12h nolu RAM satırının içeriği = 10h

işlem sonrası;

12h nolu RAM satırının içeriği = EFh

Z = 0 Olur.

DECF

Belirtilen yazacın içeriği bir azaltılır, sonuç W yazacına veya içeriği azaltılan yazaca yazılır.

Etkilenen bayraklar: Z

İşletilme süresi: 1 makine saykılı.

ÖRNEK:

DECF 25h,w

25h nolu RAM satırının içeriği = 23h

23h - 1 = 22h

İşlem sonunda;

25h nolu RAM satırının içeriği = 23h

W yazacının içeriği=22h

Z = 0 Olur.

DECFSZ

Belirtilen yazacın içeriği bir azaltılır, eğer azaltma sonucu yazacın içeriği “0” ise bir sonraki komutu NOP olarak işlet. Başka bir söyleyişle bir sonraki komutu atla.

Etkilenen bayraklar: Yok

İşletilme süresi: Eğer sonuç “0” ise 2 makine saykılı, değilse 2 makine saykılı.

Örnek:

DECFSZ 25h,w

25h nolu RAM satırının içeriği = 23h

23h - 1 = 22h

İşlem sonunda;

25h nolu RAM satırının içeriği hala = 23h

W yazacının içeriği=22h

Sonuç sıfırdan farklı olduğu için bir sonraki komut işletilir.

GOTO

Koşulsuz bağlanma komutudur belirtilen adrese bağlanır. Adresin 11 biti hemen komuttan sonra, diğer iki biti ise PCLATH yazacının 3 ve 4 nolu bitleri ile belirtilir. PIC16F84’te program belleği 1 k olduğu için 10 bit yeterlidir. Diğer bitler göz ardı edilebilir.

İŞLEV:

k → (PC <10:0>)

(PCLATH <4:3>) → (PC <12:11>)

Etkilenen bayraklar: hiç biri

İşletilme süresi: 2

ÖRNEK:

GOTO BASLA

BASLA etiketli satırın adresi 85h

PCLATH yazacının içeriği = 00h

İki adresi birleştirdiğimizde işlem yapılacak adres elde edilir.

PCLATH BASLA

0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1

0 0 + 0 0 0 1 0 0 0 0 1 0 1

0 0 0 0 0 1 0 0 0 0 1 0 1

INCF

Belirtilen yazacın içeriğini bir arttır.

Etkilen bayrak: Z

İşletilme süresi: 1

Örnek:

INCF 25h,w

25h nolu RAM satırının içeriği = 23h

23h + 1 = 24h

İşlem sonunda;

25h nolu RAM satırının içeriği hala = 23h

W yazacının içeriği=24h

Z = 0

INCFSZ

Belirtilen yazacın içeriğini bir arttır. Sonuç sıfır ise bir sonraki komutu NOP olarak işletir.

Etkilen bayrak: Hiçbiri

İşletilme süresi: Eğer sonuç sıfırdan farklı ise 1 makine saykılı, sıfır ise 2 makine saykılı.

Örnek:

INCFSZ 25h,w

25h nolu RAM satırının içeriği = FFh

FFh + 1 = 0h

İşlem sonunda;

25h nolu RAM satırının içeriği hala = 0h

Sonuç sıfır olduğu için bir sonraki komut NOP olarak işletilecektir.

IORWF

W yazacı ile adresi belirtilen yazacın içeriği VEYA'lanır, sonuç belirtilen yazaca yazılır.

Etkilen bayrak: Z

İşletilme süresi: 1 makine saykılı.

Örnek:

IORWF 12h,f

W yazacının içeriği = 50h		50h	0 1 0 1 0 0 0 0
12h nolu RAM'in içeriği = 10h	10h		0 0 0 1 0 0 0 0
50h VEYA 10h = 50h		50h	0 1 0 1 0 0 0 0

Sonuç 12h nolu RAM'e yazılır.

IORLW

W yazacının içeriği ile sabit sayıyı VEYA'lar, sonuç W yazacına yazılır.

İŞLEV:

k V (W) $\rightarrow$ (W)

Etkilenen bayraklar: Z

İşletilme süresi: 1 makine saykılı

ÖRNEK:

IORLW 23h

W yazacının içeriği = 50h	50h	0 1 0 1 0 0 0 0
Literal değer = 23h	23h	0 0 1 0 0 0 1 1
50h V 23h = 73h	73h	0 1 1 1 0 0 1 1

sonuç W yazacına yazılır, W= 73h

Z = 0

MOVLW

W yazacına sabit değer yükler.

İŞLEV:

k → (W)

Etkilenen bayraklar: Hiç bir

İşletilme süresi: 1 makine saykılı

ÖRNEK:

MOVLW 25h

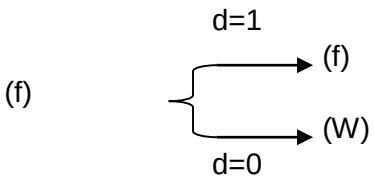
Literal değer = 25h

Sonuç W yazacına yazılır, W=25h

MOVF

İçeriği belirtilen yazacın içeriği hedefe aktarılır. Hedef d=1 ise kendi üzerine yazılır, anlamsız gibi gelen bu işlem sonunda Z bayrağı etkilendiği için yazacının içeriği test edilmiş olur.

İŞLEV:



Etkilen bayrak: Z

İşletilme süresi: 1 makine saykılı.

Örnek:

MOVF 12h,w

12h nolu RAM bellek satırının içeriğini W yazacına aktarır

12h nolu RAM belleğin içeriği = 55h

Komut işletildikten sonra;

W=55h

12h nolu RAM belleğin içeriği = 55h

Z = 0

MOVWF

W yazacının içeriği adresi belirtilen yazaca aktarılır.

İŞLEV:

(W) → (f)

Etkilen bayrak: Yok

İşletilme süresi: 1 makine saykılı.

Örnek:

MOVWF 14h

W yazacının içeriği = 34h

Komut işletildikten sonra;

W=34h

14h nolu RAM belleğin içeriği = 34h

NOP

İşlem yapmadan 1 makine saykılı harcar.

Etkilen bayrak: Hiç biri.

İşletilme süresi: 1 makine saykılı.

OPTION

W yazacının içeriği OPTION yazacına kopyalanır. Bu komut yeni ürün PIC'lerde yer almaz. Yeni PIC'lere uyum için kullanılmaması tavsiye edilir. Aslında bu komuta da gerek yoktur. Çünkü OPTION yazacı programcı tarafından diğer komutlar ile okunabilen ve yazılabilen bir yazaçtır.

Etkilenen bayraklar: Hiç biri.

İşletilme süresi: 1

RETFIE

Kesme servis alt programından geri dön. Yığının en üstünde yer alan bilgi PC'ye çekilir ve genel kesme izinleme biti kurulur.

İŞLEV:

TOS → (PC)

1 → GIE

Etkilenen bayraklar: Hiç biri.

İşletilme süresi: 2

RETLW

W yazacına komutun işlendiği satırda yer alan 8 bit sabit sayı yüklenir. Yığının en üstünde yer alan bilgi PC'ye alınarak yeni adresten program işletilmeye devam eder. Bu komut baş vuru tablosu yapmak için kullanılır. CALL komutu ile tablonun birinci satırına gidilir, birinci satırda PC'ye W yazacında yer alan satır numarası toplanarak istenilen değer tablodan alınarak ana programa geri dönülür.

İŞLEV:

k → (W)

TOS → (PC)

Etkilenen bayraklar:

Hiç biri.

İşletilme süresi: 2

ÖRNEK:

```
CALL TABLO  ;W yazacı tablodan yüklenecek sayının satır ;numarasını içeriyor.
.
.
.
TABLO
        ADDWF PC    ;satır numarası elde ediliyor.
        RETLW K1    ;Tablonun başlangıç satırı
        RETLW K2
        RETLW K3
        .
        .
        .
        .
        RETLW Kn
RETURN
```

Alt programdan geri dön. Yığının en üstünde yer alan adres PC'ye çekilir, program bu adresten işletilmeye devam eder.

İŞLEV:

TOS → (PC)

Etkilenen bayraklar: Hiç biri.

İşletilme süresi: 2 makine saykılı.

RLF

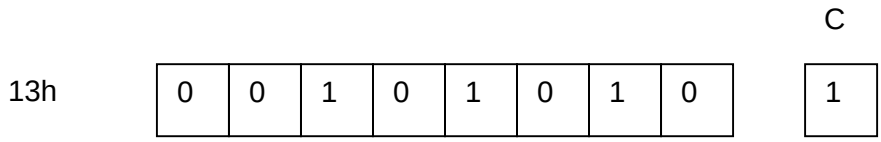
Adresi belirtilen yazacın içeriği elde üzerinden sola döndürülür.

Etkilen bayrak: C

İşletilme süresi: 1 makine saykılı.

Örnek

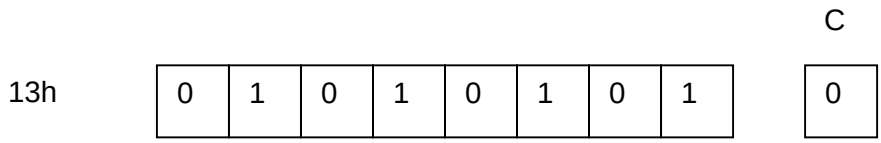
RLF 13h



13h nolu RAM belleğin içeriği = 2Ah

Elde bayrağı C=1

Komut işletildikten sonra;



RRF

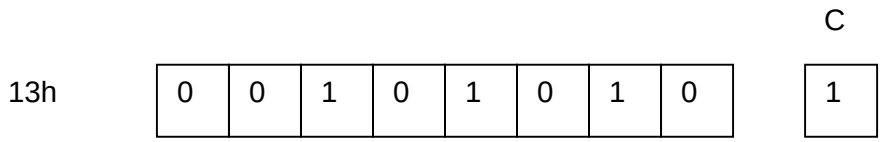
Adresi belirtilen yazacın içeriği elde üzerinden sağa döndürülür.

Etkilen bayrak: C

İşletilme süresi: 1 makine saykılı.

Örnek

RRF 13h



13h nolu RAM belleğin içeriği = 2Ah

Elde bayrağı C=1

Komut işletildikten sonra;

									C
13h	1	0	0	1	0	1	0	1	0

SLEEP

İşlemciyi yedek konumuna götürür. Portlar seviyelerini, yazaçlar içeriklerini korur. Watchdog zamanlayıcısı, PD, ve WDT prescaler temizlenir, TO kurulur. Osilatör durur.

İŞLEV:

00h → WDT

00h → WDT Prescaller

1 → TO

0 → PD

Etkilenen bayraklar: PD, TO

TO = 1

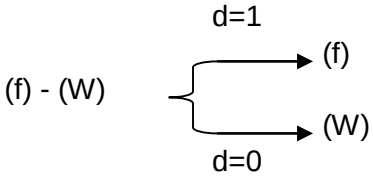
PD = 0

İşletilme süresi: 1

SUBWF

Adresi belirtilen yazacın içeriğinden W yazacının içeriği çıkarılır. 2'ye tümleyen aritmetiği kullanılır. Elde bayrağı borç bayrağı olarak kullanılır. Elde bayrağının 0 olması borç alındığını sonucun negatif olduğunu belirtir. Hedef biti ile sonucun yazılacağı yer belirlenir.

İŞLEV:



Etkilen bayraklar: C, DC, Z

İşletilme süresi: 1 makine saykılı.

W= ÇIKAN

YAZAÇ= ÇIKARILAN

SONUÇ=W veya YAZAÇ

Örnek:1

SUBWF 31h,w

W yazacının içeriği = 25h

31h nolu RAM bellek satırının içeriği = 60h

Komut işletildikten sonra;

C	?		Z	?		DC	?	
31h	0	1	1	0	0	0	0	0
W	0	0	1	0	0	1	0	1
W	0	0	1	1	1	0	1	1
C	1		Z	0		DC	0	

31h nolu RAM bellek satırının içeriği = 60h

Sonuç W yazacına yazılır ve pozitiftir.

Örnek:2

SUBWF 31h,w

W yazacının içeriği = 60h

31h nolu RAM bellek satırının içeriği = 25h

Komut işletildikten sonra;

C	?		Z	?		DC	?	
31h	0	0	1	0	0	1	0	1
W	0	1	1	0	0	0	0	0
W	1	1	0	0	0	1	0	1
C	0		Z	0		DC	0	

31h nolu RAM bellek satırının içeriği = 25h

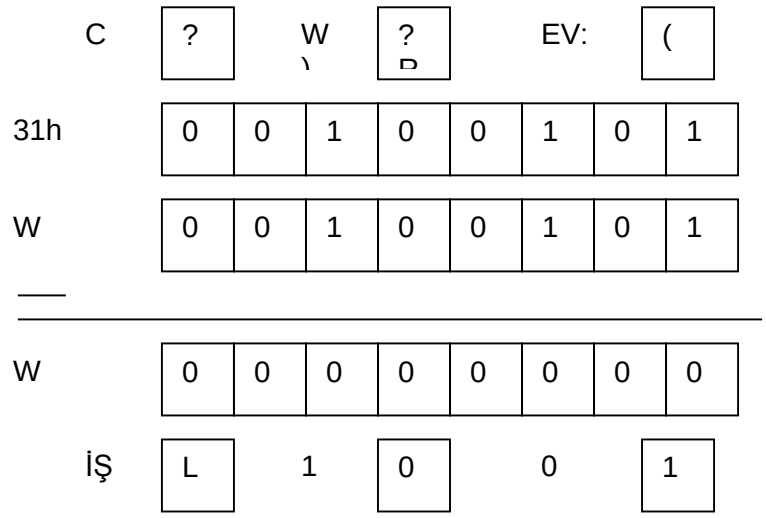
Sonuç W yazacına yazılır ve negatiftir.

Örnek:3

SUBWF 31h,w

W yazacının içeriği = 25h

31h nolu RAM bellek satırının içeriği = 25h



Komut işletildikten sonra;

31h nolu RAM bellek satırının içeriği = 25h

Sonuç W yazacına yazılır ve sıfırdır.

SUBLW

8 bit sabit sayıdan W yazacının içeriği çıkarılır. 2'ye tümleyen aritmetiği kullanılır. Elde bayrağı borç bayrağı olarak kullanılır. Elde bayrağının "0" olması borç alındığını ve sonucun negatif olduğunu belirtir. Sonuç W yazacına yazılır.

0 0 → 0

Etkilen bayraklar: C, DC, Z

İşletilme süresi: 1 makine saykılı.

W= ÇIKAN

YAZAÇ = ÇIKARILAN

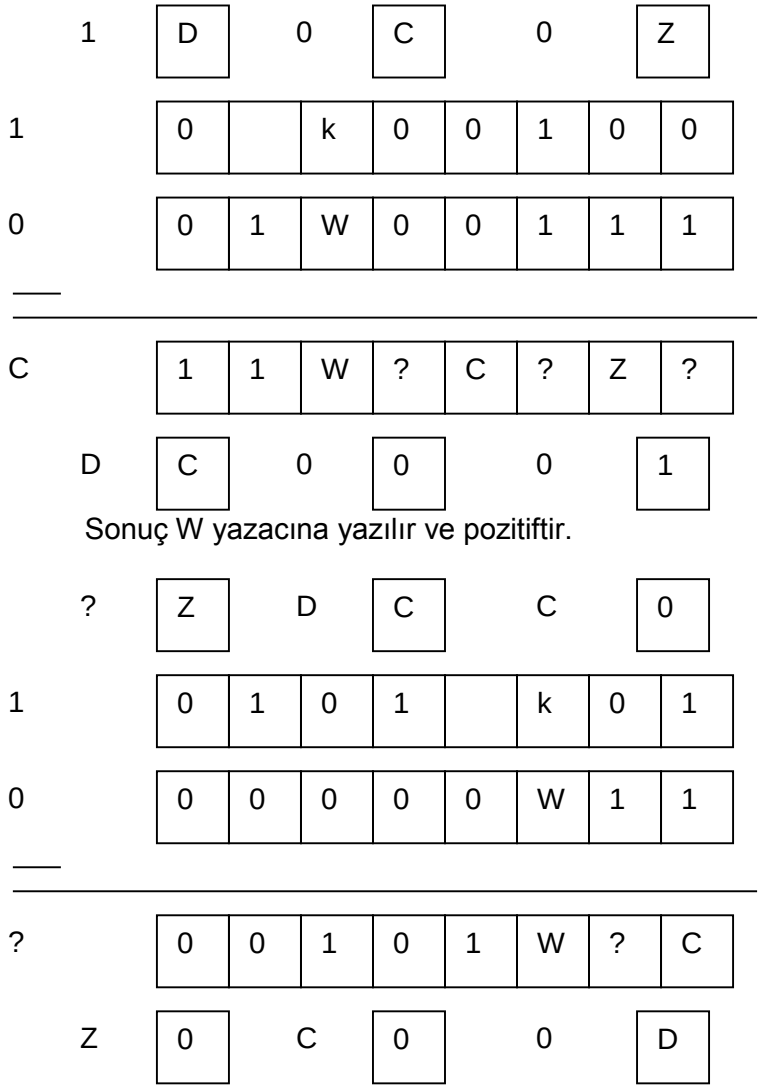
SONUÇ=W veya YAZAÇ

Örnek:1

SUBLW 60h

W yazacının içeriği = 25h

Komut işletildikten sonra;



Sonuç W yazacına yazılır ve pozitiftir.

Örnek:2

SUBLW 25

W yazacının içeriği = 60h

Komut işletildikten sonra;

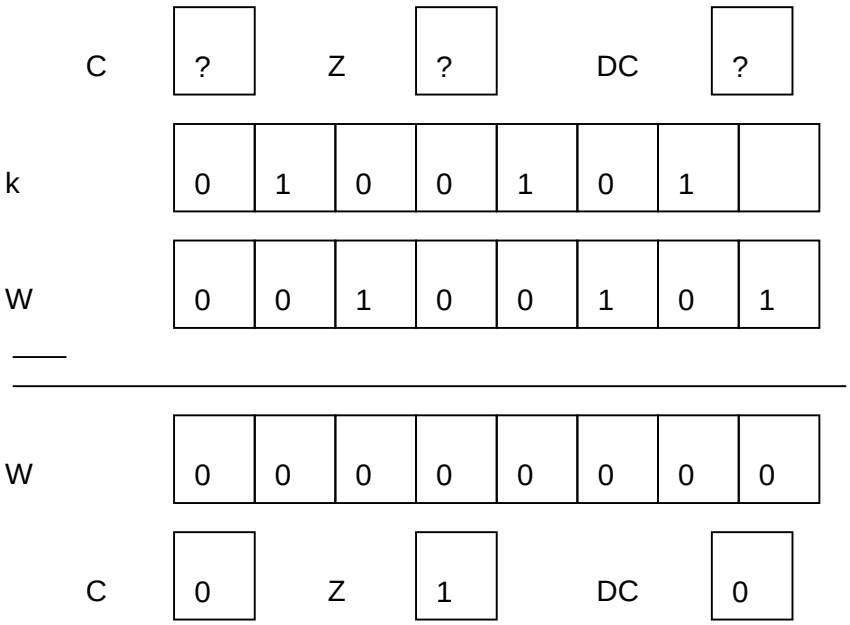
Sonuç W yazacına yazılır ve negatiftir.

Örnek:3

SUBLW 25h

W yazacının içeriği = 25h

Komut işletildikten sonra;



Sonuç W yazacına yazılır ve sıfırdır.

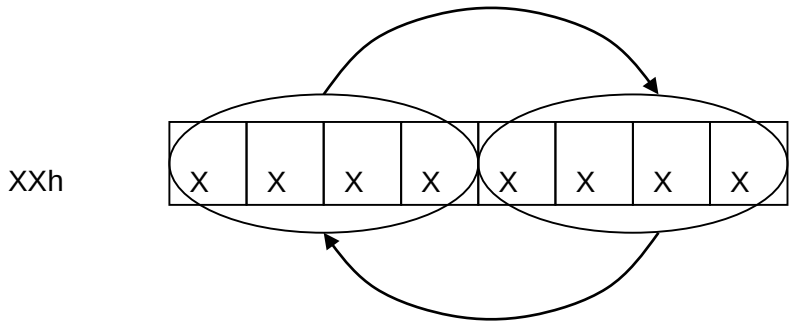
SWAPF

Belirtilen yazacın düşük değerli dört biti ile yüksek değerli dört bitini yer değiştirir.

Etkilen bayraklar: hiç biri

İşletilme süresi: 1 makine saykılı.

İŞLEV:



ÖRNEK.

SWAPF 34h

34h nolu RAM Yazacın içeriği = F0h 1 1 1 1 0 0 0 0

Komut işletildikten sonra;

34h nolu RAM Yazacın içeriği 0 0 0 0 1 1 1 1

TRIS

W yazacının içeriği TRIS yazacına kopyalanır. Sadece PIC16C5X ailesine komut uyumu olan PIC ailelerinde geçerlidir. Aslında bu komuta gerek yoktur, çünkü programcı başka komutlar

ile bu yazaçlara yazma yapabilir veya okuyabilir. Yeni sürümlerinde bu komut yer almamaktadır.

Etkilenen bayraklar: Hiç biri

İşletilme süresi: 1

Example

TRIS PortB yazacının içeriği = 00h

W yazacının içeriği = 34h

TRIS PortB

Komutu işlendikten sonra;

TRISB yazacının içeriği = 34h

XORLW

W yazacı ile sabit sayıyı ÖZEL VEYA'lar sonuç W yazacına yazılır.

İŞLEV:

K XOR (W) $\rightarrow$ (W)

Etkilen bayraklar: Z

İşletilme süresi: 1 makine saykılı.

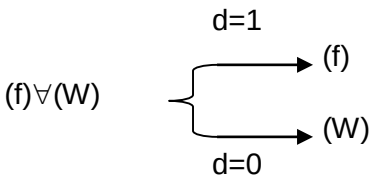
XORWF

W yazacı ile adresi belirtilen yazacın içeriklerini ÖZEL VEYA'lar sonuç belitilen adrese yazılır.

Etkilen bayraklar: Z

İşletilme süresi: 1 makine saykılı.

İŞLEV:



ÖRNEK:

XORWF 12h,f

12h	0	0	1	0	0	0	0	1
W	0	1	0	1	0	0	0	0

İşlem sonrası;

W

0	1	0	0	0	0	0	1

Z = 0