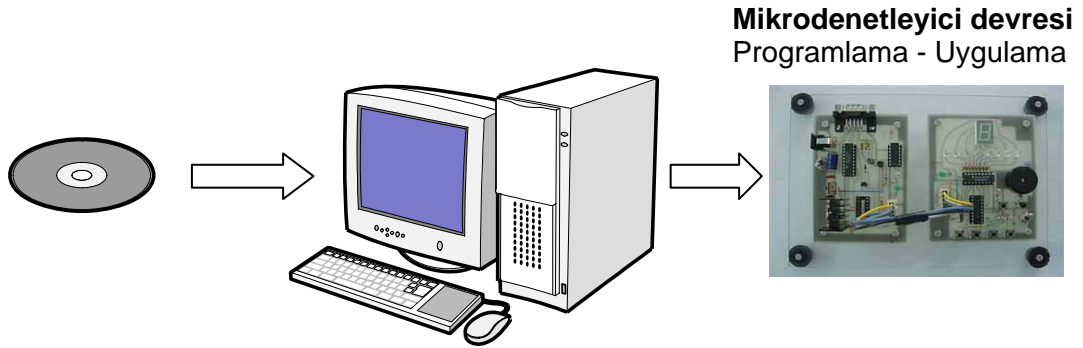


1. Mikrodenetleyici Programlamada Giriş-Çıkış İşlemleri

1.1. Programlamada dil seçimi

Mikrodenetleyici programlama düşük seviyeli (assembler) veya yüksek seviyeli programlama dilleri (C, C++, BASIC) ile yapılabilir. Bunun için gerekli olan bir mikrodenetleyici devresi ve derleyici yazılımıdır. Mikrodenetleyici devresi programlayıcı ve uygulama kısımlarından oluşur. Bilgisayar programlama editöründe yazılan program çeşitli iletişim metotları ile (paralel iletişim, USB) mikrodenetleyiciye gönderilir. Devrenin uygulama kısmında ise program sonuçları gözlenir.



Şekil 1.1. Mikrodenetleyici programlama

Mikrodenetleyici için yazılan basit bir assembler programını örnek alalım. Program uygulama devresindeki butona basılmasıyla LED'in ışık vermesi işlemini yapmaktadır.

MAIN	BTFSC	PORTA, SWITCH	; butona basılana kadar bekle
	BSF	PORTB, LED	; LED on
LOOP	BTFSS	PORTA, SWITCH	; buton açık mı?
	GOTO	LOOP	; buton bırakılana kadar bekle
	BCF	PORTB, LED	; LED off
	GOTO	MAIN	; başa dön

Bu programın C dili eşdeğeri aşağıdaki gibidir.

```

main()
{
    set_tris_b(0x00);           // port b yi çıkış olarak ata
    while(1)
    {
        if(input(PIN_A0))      // buton kapalı mı?
            output_high(PIN_B0); // eğer kapalı ise LED on
        else
            output_low(PIN_B0); // buton açık ise LED off
    }
}

```

C dilinde yazılan program makine koduna çevrildiğinde sonuç aşağıdaki gibidir.

main()			
{			
set_tris_b(0x00);	0007	MOVLW	00
	0008	TRIS	6
while(1)			
{			
if(input(PIN_A0))	0009	BTFSS	05,0
	000A	GOTO	00D
output_high(PIN_B0);	000B	BSF	06,0
else	000C	GOTO	00E
output_low(PIN_B0);	000D	BCF	06,0
}	000E	GOTO	009
}			

Derlenmiş olan C programı bellekte assembler programından daha fazla yer tutmaktadır. Assembler dili mikrodnetleyici donanımının öğrenilmesinde temel teşkil eder. C dili ise kullanıcıya daha yakındır ve fonksiyon desteği sağlar. C dilinin sakıncası olan bellek kullanımı ancak etkin ve ileri düzey programlama metotları ile mümkündür.

1.2. Mikrodnetleyici programlamaya hazırlık işlemleri

1.2.1 C dili derleyicisi

Bu modülde kullanacağımız derleyici, 14 bit program hafızasına ait mikrodnetleyici için kullanılır. Derleyicinin daha gelişmiş sürümleri 16 bit ve 18 bit mikrodnetleyiciler için uygundur. Derleyici özellikleri aşağıdaki şekilde özetlenebilir.

1. Derleyicinin mikrodenetleyici programlama editörü (MPLAB) ile kullanımı

C dili derleyicisi kaynak kodları MPLAB programı altında çalıştırılabilir. Bu işlemde MPLAB programı içinde c derleyicisi tanımlanır ve bu şekilde MPLAB programı, derleyici desteği altında çalışır. Yazılan programı izlemek ve hata ayıklamak mümkündür. Hata ayıklama işlemi derleyicinin oluşturduğu makine kodları ile yapılır.

2. Fonksiyon desteği

Derleyici içindeki fonksiyonlar ile RS232 seri iletişim, A/D çevirici, I/O (giriş - çıkış), bit – byte düzeyinde işlemler, I2C, LCD display gibi mikrodenetleyici uygulamaları kolaylıkla yapılabilir.

3. Veri tanımlamaları

1 bit, 8 bit, 16 bit, ve 32 bit bilgi program içerisinde tanımlanabilir. Gerçel sayılar (float) 32 bit olarak kullanılır.

4. Assembler komutları kullanılabilir.

C dilinde yazılmış programın içinde assembler komutları kullanmak mümkündür. Assembler dili ile C dili arasındaki değişken dağılımı desteklenmektedir.

5. Standart giriş – çıkış fonksiyonları

RS- 232 seri iletişim metodu ile bilgisayar bağlantısı yapılarak standart giriş – çıkış fonksiyonları kullanılabilir.

6. Donanım desteği

Derleyici içindeki başlık dosyaları ile mikrodenetleyici uygulamalarındaki çeşitli entegrelerin kullanım desteği sağlanmıştır.

7. Değişken alanının etkin kullanımı

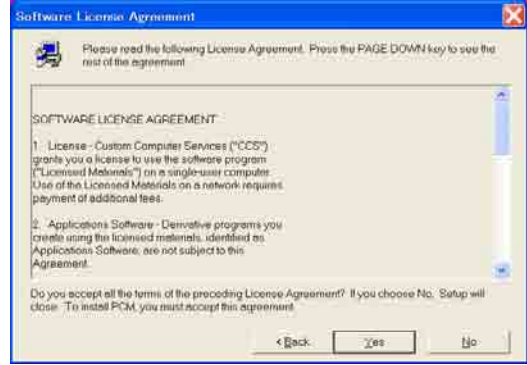
Komutlar program belleğinde, değişkenler ise değişken alanında saklanır. Komutlar ve değişkenlerin geçici paylaşımını sağlamak için en az değişken alanı ayrılır.

1.2.1.1 C derleyicisi kurulumu

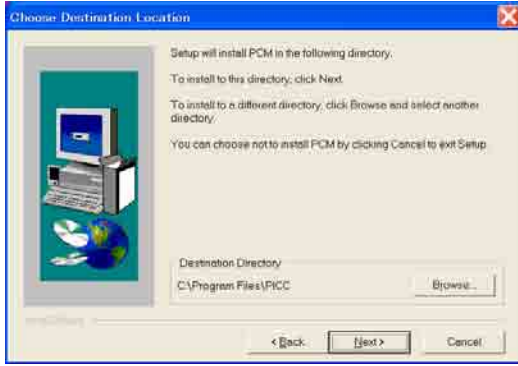
Bu modülde 12 bitlik mikrodenetleyiciler için C dili derleyicisi kullanılacaktır. PCM derleyici paketi 2 kurulum disketi ve kullanım kılavuzundan oluşmaktadır. Derleyicinin kurulumu aşağıda açıklanan işlem sırasına göre olmalıdır.



Şekil 1.2. Kurulum mesajı (disket 1)



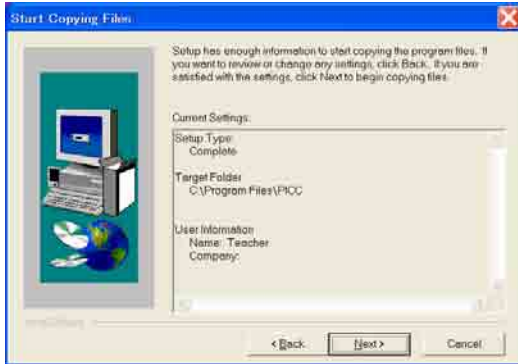
Şekil 1.3. Yazılım lisans anlaşması



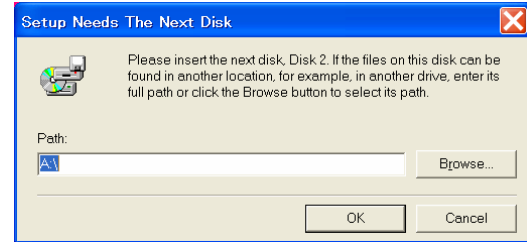
Şekil 1.4. Dizin seçimi



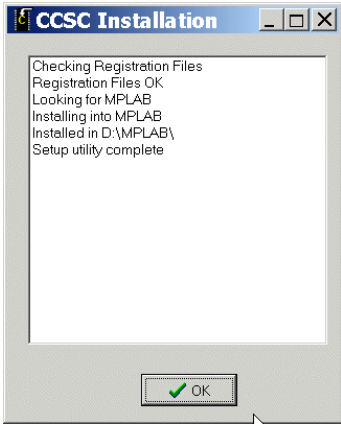
Şekil 1.5. Dizin ismi seçimi



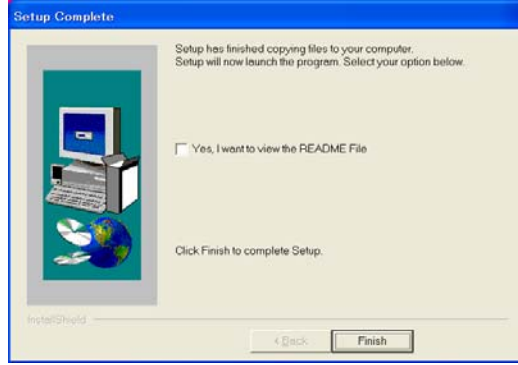
Şekil 1.6. Kurulumu doğrulama



Şekil 1.7. Disket 2



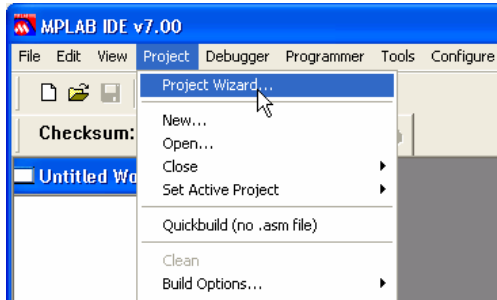
Şekil 1.8. MPLAB ile bağlantı



Şekil 1.9. Kurulum sonu

1.2.1.2 Derleyicinin MPLAB ile kullanımı

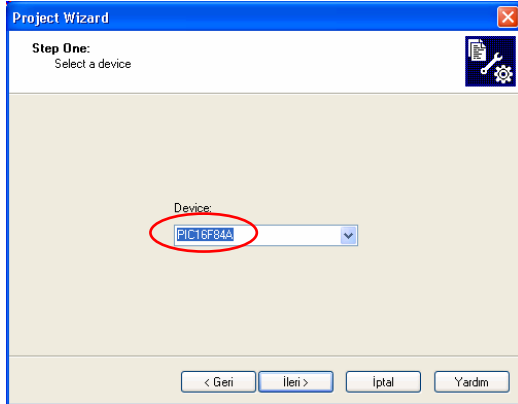
Mikrodenetleyici C derleyici kurulduktan sonra MPLAB içinde tanımlanması gerekir.



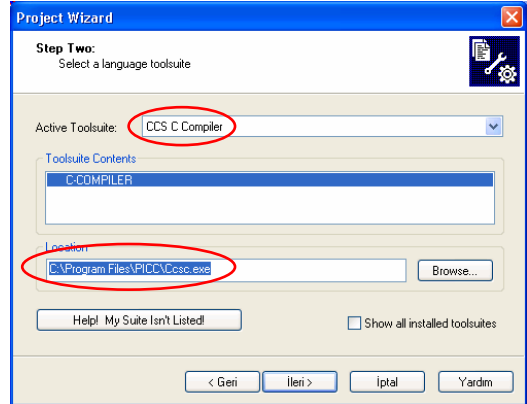
Şekil 1.10. Proje oluşturma listesi



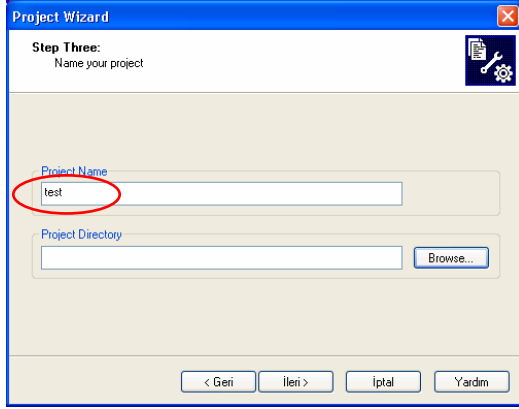
Şekil 1.11. Proje sihirbazı



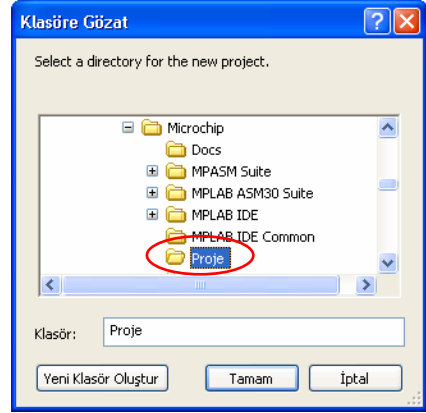
Şekil 1.12. Mikrodenetleyici seçimi



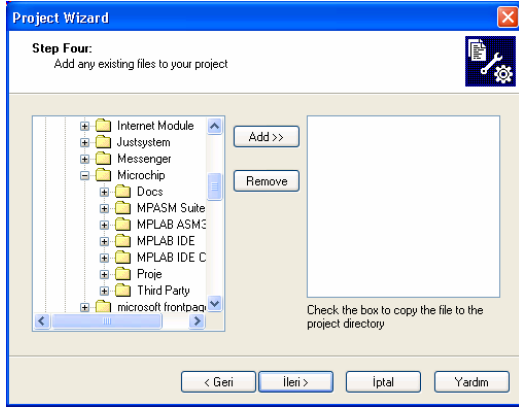
Şekil 1.13. Programlama dil seçimi



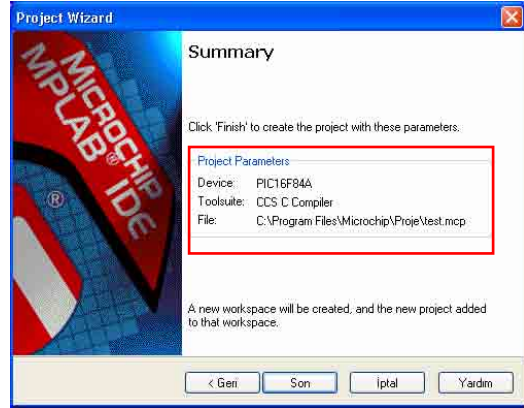
Şekil 1.14. Proje ismi seçimi



Şekil 1.15. Proje dizin konumu



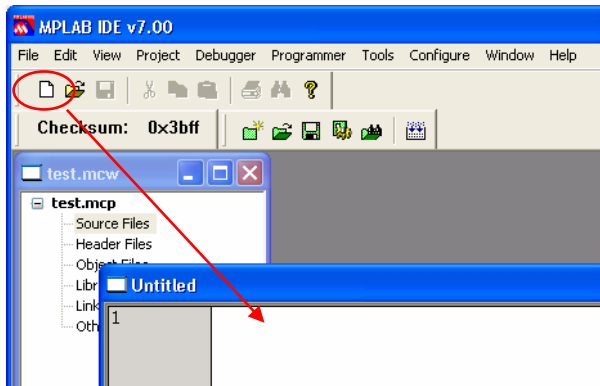
Şekil 1.16. Önceki dosyaların aktarımı



Şekil 1.17. Proje oluşturma işleminin sonu

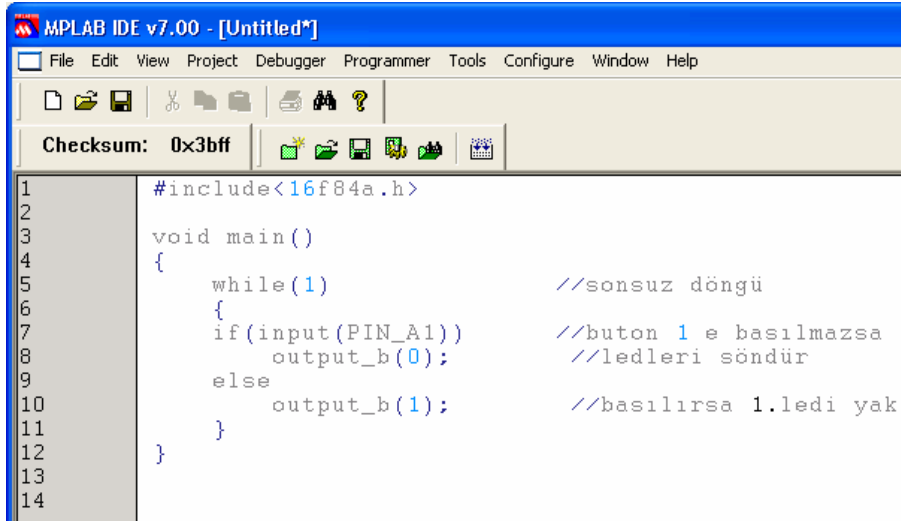
Yukarıdaki işlemlerin sonucunda MPLAB programı, C dili derleyicisi ile bağlanmıştır. Bu aşamadan sonra programlar, MPLAB editörü seçenekleri ile yazılacak ve derleyici ile derlenecektir.

1.2.1.3 Derleme işlemi



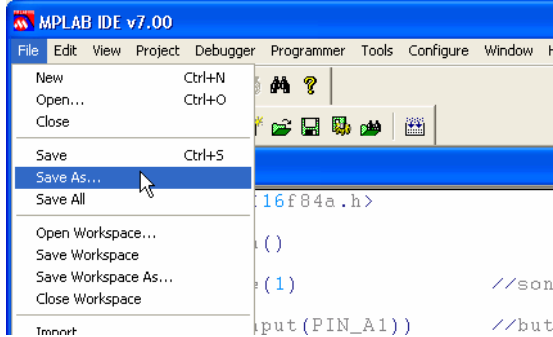
Şekil 1.18. Yeni çalışma sayfası

MPLAB editöründe C programının yazımı için yeni çalışma sayfası açılır

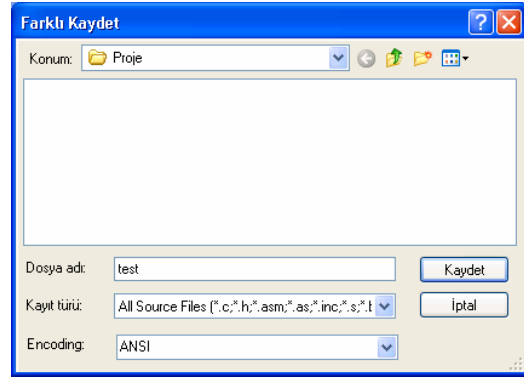


```
1 #include<16f84a.h>
2
3 void main()
4 {
5     while(1) //sonsuz döngü
6     {
7         if(input(PIN_A1)) //buton 1 e basılmazsa
8             output_b(0); //ledleri söndür
9         else
10            output_b(1); //basılırsa 1.ledi yak
11     }
12 }
13
14
```

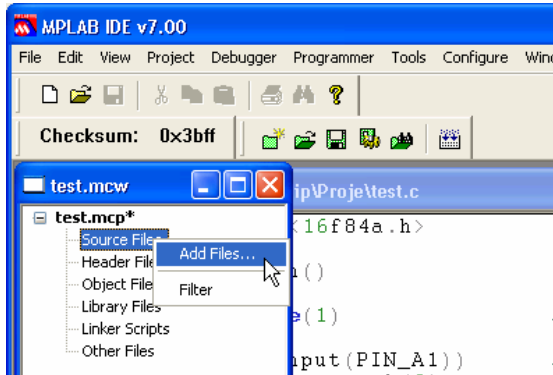
Şekil 1.19. Örnek programın yazılması



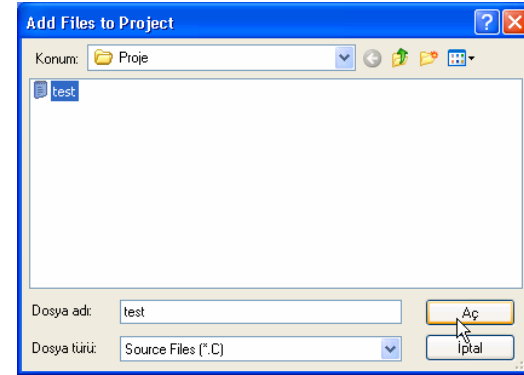
Şekil 1.20. Programın farklı kaydedilmesi



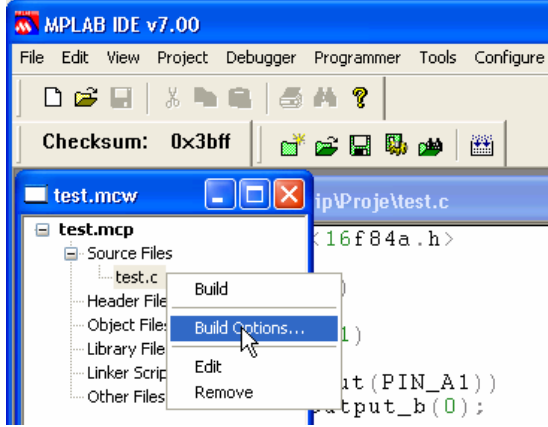
Şekil 1.21. Proje → test.c



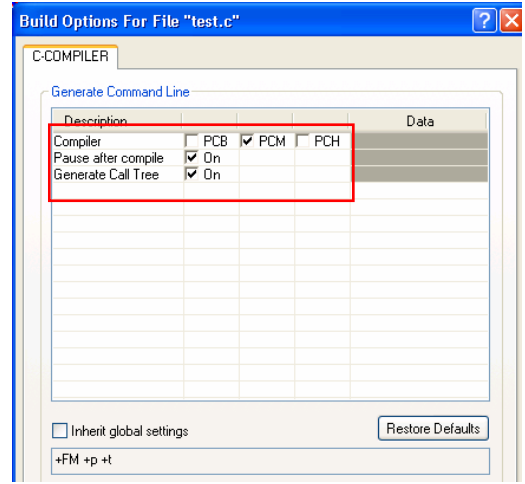
Şekil 1.22. test projesine kaynak dosya eklenmesi



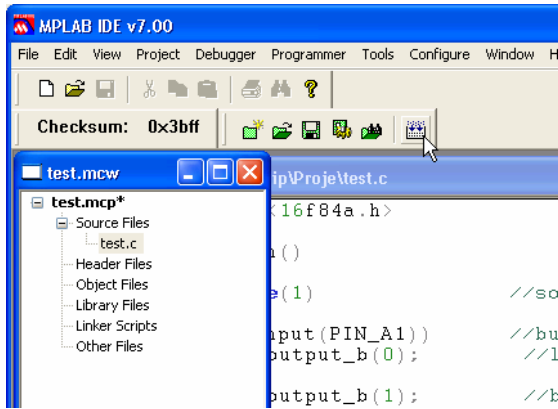
Şekil 1.23. Kaynak dosyanın seçimi



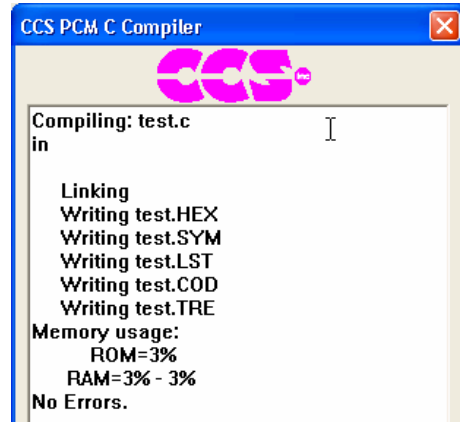
Şekil 1.24. Kaynak dosya fare ile sağ tıklanır



Şekil 1.25. Derleme seçenekleri



Şekil 1.26. Kaynak dosyanın derlenmesi



Şekil 1.27. Derleme sonuçları

Derleme sonucunda test.c adlı kaynak dosya ile çeşitli derleme dosyaları oluşturulur. Bu dosyaların anlamları aşağıdaki tabloda görülmektedir.

Uzantı	Dosya tipi	Açıklama
COD	Object file	Derleme ve hata ayıklamada kullanılan nesne dosyası
ERR	Error file	Derleme işleminde oluşabilecek hataları içerir.
HEX	HEX file	Derleme sonucunda oluşan hegzadesimal kodları içerir.
LST	Compilation list	Assembler listesi
PRJ	Project file	İçinde kaynak dosyanın oluşturulur.
SYM	Symbol list file	Tüm değişkenler , adresler ve fonksiyon listeleri
TRE	Function tree	Fonksiyon listesi


Tablo 1.1. Derleme dosyaları

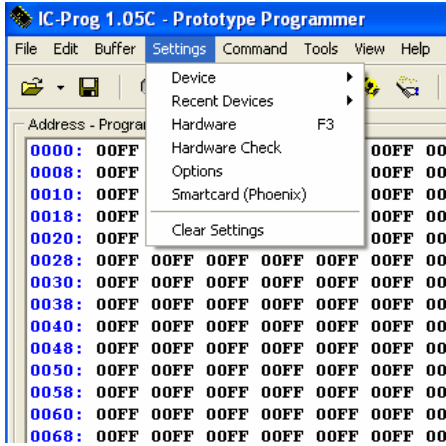
1.2.2 Mikrodenetleyici veri yükleme programı

Bilgisayarda çeşitli dillerde yazılmış programlar derlenerek HEX dosya haline getirilmektedir. Bu dosyaları da mikrodenetleyiciye yazmamız gerekmektedir. Bunun için çeşitli devreler ve yazılımlar mevcuttur. En yaygın olarak kullanılanlardan biri de IC-PROG yazılımıdır. IC-PROG yazılımı ile mikrodenetleyici çeşidinin büyük bir çoğunluğunu programlamamız mümkündür. Bunun yanı sıra birçok programlayıcı devreyi de desteklemektedir. Bu kitaptaki yazıcı devresi AN589 Programmer donanımına göre yapılmıştır. Programı çalıştırdığımızda donanım kısmından AN589 seçmemiz gerekmektedir. Bu yazılımı kurmamıza da gerek yoktur. Herhangi bir yerden kopyalamamız veya internette indirmemiz yeterli olacaktır. Programın internet adresi www.ic-prog.com 'dur.

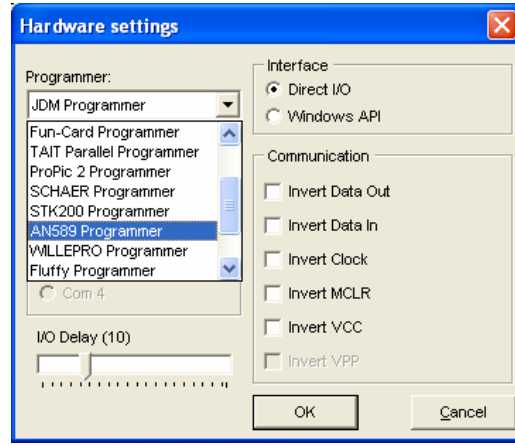
1.2.2.1Yazıcı ayarının yapılması

Yukarıdaki ikona çift tıkladığımızda ilk defa karşımıza İngilizce versiyonu açılır. İlk donanım seçimi ise JDM programmer'dir. Makinemize bu ayarları bir defa yapmamız yeterli olacaktır. Kapanıp açıldığında bizim son ayarlarımız gelmektedir. Öncelikle yazıcı tipimize göre donanım (hardware) AN589 Programmer' e seçmemiz gerekmektedir. Bunun için

Setting ve Hardware menülerini takip etmemiz, F3 tuşuna basmamız veya ana sayfada  tuşuna basmamız yeterli olacaktır. Her üç yolda aynı menüyü açar.



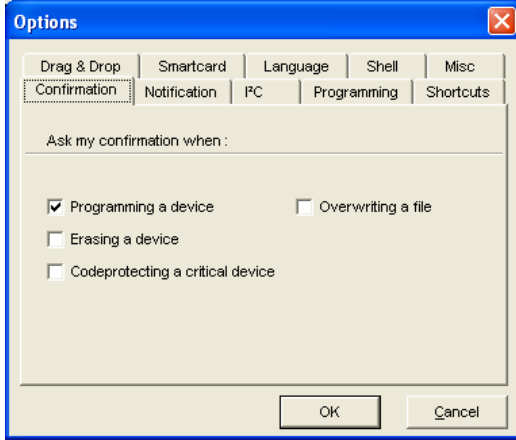
Şekil 1.29. Programlayıcı menüsü



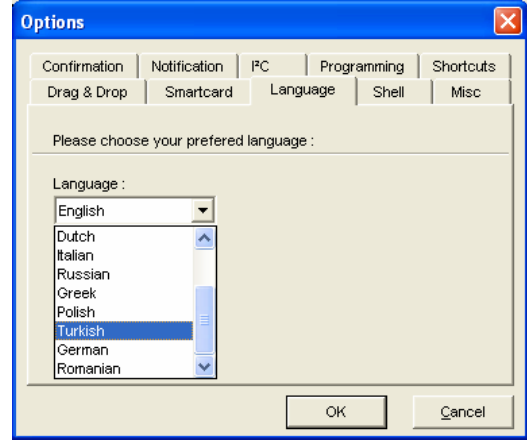
Şekil 1.30. Programlayıcı devre seçimi

1.2.2.2. Yazılımın dil seçimi

Kullandığımız yazılımın Türkçe olabilmesi içinde Setting Options menüsünü takip etmemiz gerekmektedir. Açılan Options menüsünden Language kısmından Turkish seçilip OK ile çıktığında program kendini bir defa açıp kapamak suretiyle programımız artık Türkçe olmuştur.



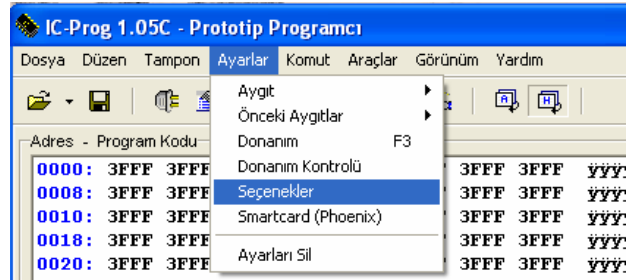
Şekil 1.31. Seçenekler penceresi



Şekil 1.32. Dil ayarları



Şekil 1.33. Programı yeniden başlatma



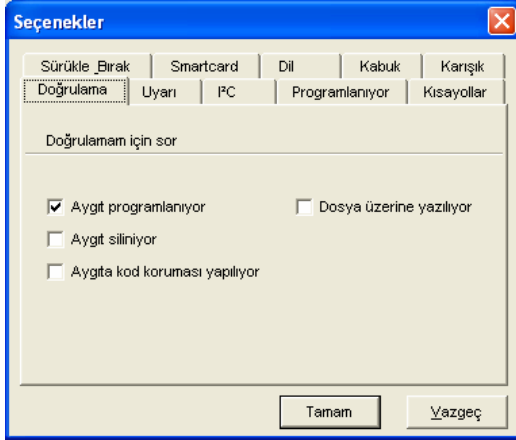
Şekil 1.34. Yazılımın Türkçe olması

1.2.2.3. Mikrodenetleyici seçimi

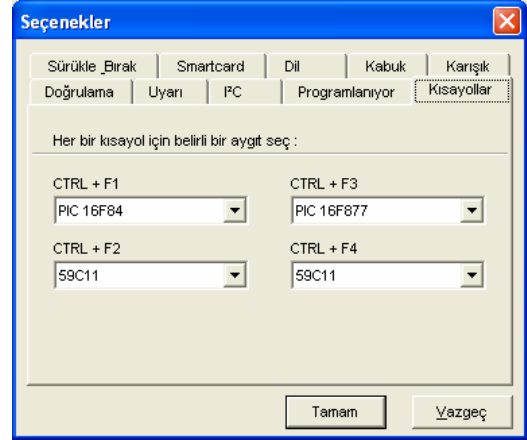
Bu yazılım ve yazıcı devremiz ile birçok mikrodenetleyici çeşidini rahatlıkla programlayabilmekteyiz. Tabi ki her defasında programlamak istediğimiz mikrodenetleyici çeşidini bulmak zaman almakta ve sıkıcı olmaktadır. Bunu için daha önceden belirleyeceğimiz 4 adet PIC çeşidine CTRL + F1 den CTRL + F4 e kadar kısa yol oluşturabilmek mümkündür.

Bunu gerçekleştirebilmek için Ayarlar Seçenekler menüsünü takip ettiğimizde karşımıza aşağıdaki menü çıkmaktadır. Burada kısayollar menüsü seçilerek hangi kısa yol tuşuna hangi mikrodenetleyici çeşidi atanacaksa bunlar seçilir.

Tamam ile menüden çıktığında kısa yol tuşlarına seçilen mikrodenetleyiciler atanmış olur ve artık sürekli onlar karşımıza çıkar. Mikrodenetleyici çeşidi seçmenin bir başka yolu ise ana menüde aşağıdaki şekilde olduğu gibi açılan pencereden olmaktadır.



Şekil 1.35. Seçenekler penceresi



Şekil 1.36. Kısa yolların belirlenmesi

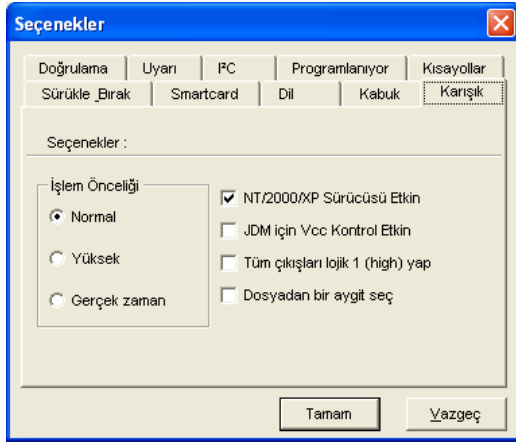


Şekil 1.37. Mikrodenetleyici türünün belirlenmesi

1.2.2.4. Kullanılan işletim sistemine uyum sağlama

IC-PROG yazılımını kullanmaya başlamadan evvel son yapacağımız ayar ise kullanacağımız bilgisayarda kurulu olan yazılıma uyum sağlamak olacaktır. Programı eğer Windows 95 – Windows 98 – Windows 98 me gibi yazılım yüklü makinelerde kullanacaksak bunun için ayarlarında herhangi bir değişiklik yapmamıza gerek yoktur. Ancak Windows 2000 - Windows XP – Windows NT gibi yazılım yüklü makinelerde kullanacaksak ayarlarını buna uyumlu hale getirmemiz gereklidir.

Bunun için Ayarlar Seçenekler menüsü takip ederek Karışık yazan kısmı seçmemiz daha sonra karşımıza çıkan pencerede ise NT/2000/XP kutucuğunun işaretli olması gerekmektedir. Bunu işaretleyip Tamam ile pencereyi kapattığımızda ayarların geçerli olabilmesi için program kendini bir kez açıp kapaması gerekmektedir. Tekrar açılan programda artık kullandığımız yazılım türüne uygun hale gelmiş olmaktadır.



Şekil 1.38. İşletim sistemini seçme



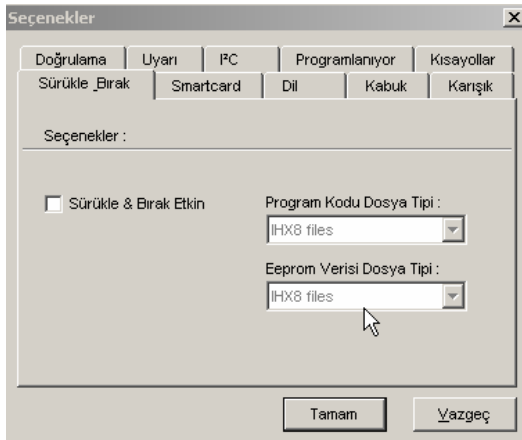
Şekil 1.38. Programı yeniden başlatma

1.2.2.5. Sürükle bırak ayarı

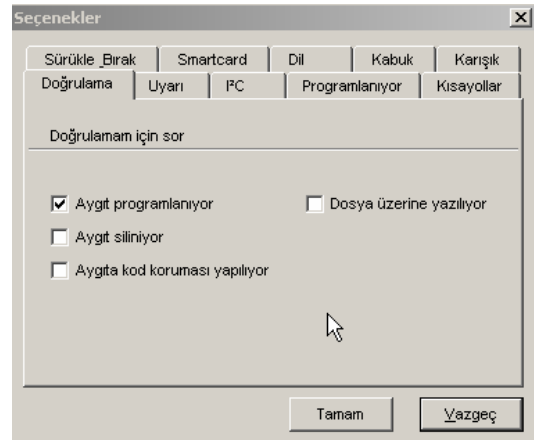
Kullanımı daha kolay hale getirebilmek için değişik ayarlar da gerçekleştirebiliriz. Bunlardan HEX uzantılı bir dosyayı sürükleyerek programımızın üzerine bıraktığımızda açılmasını istiyorsak Ayarlar / Seçenekler / Sürükle Bırak kısmına ulaşp buradaki kutucuğu işaretleyip Tamam ile çıkmamız yeterli olacaktır. Bu ayarı bir defa yapmamız yeterli olacaktır.

1.2.2.6. Doğrulama ayarı

Ic-prog yazılımı ile programlama gerçekleştirirken veya mikrodenetleyicideki bir bilgiyi silme esnasında bunun yanlışlıkla yapılabileceğini düşünenler için doğrulama bölümünü aktif hale getirmemiz mümkündür. Hangi durumda onaylama istiyorsak bunun kutucuğunu işaretlememiz yeterli olacaktır. Örneğin Aygıt programlanıyor kutucuğu işaretli ise ve biz yazılıma programlama komutunu uyguladığımızda aşağıdaki soruyu mutlaka sorarak doğrulama gerçekleştirir. Eğer işaretli değilse direk programlama işlemini gerçekleştirir.



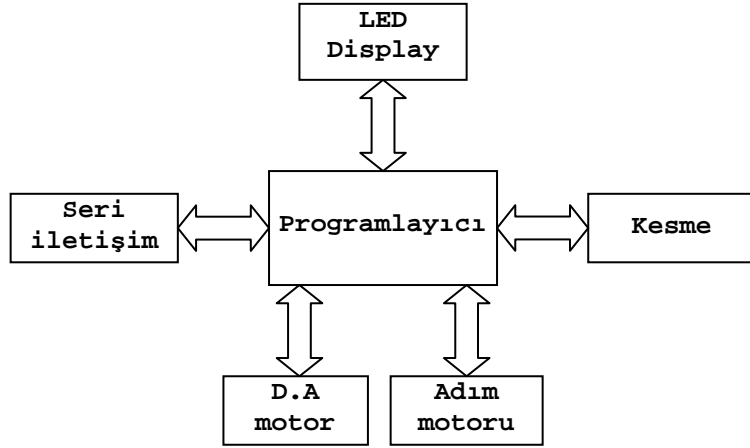
Şekil 1.39. Sürükle bırak ayarı



Şekil 1.40. Doğrulama ayarı

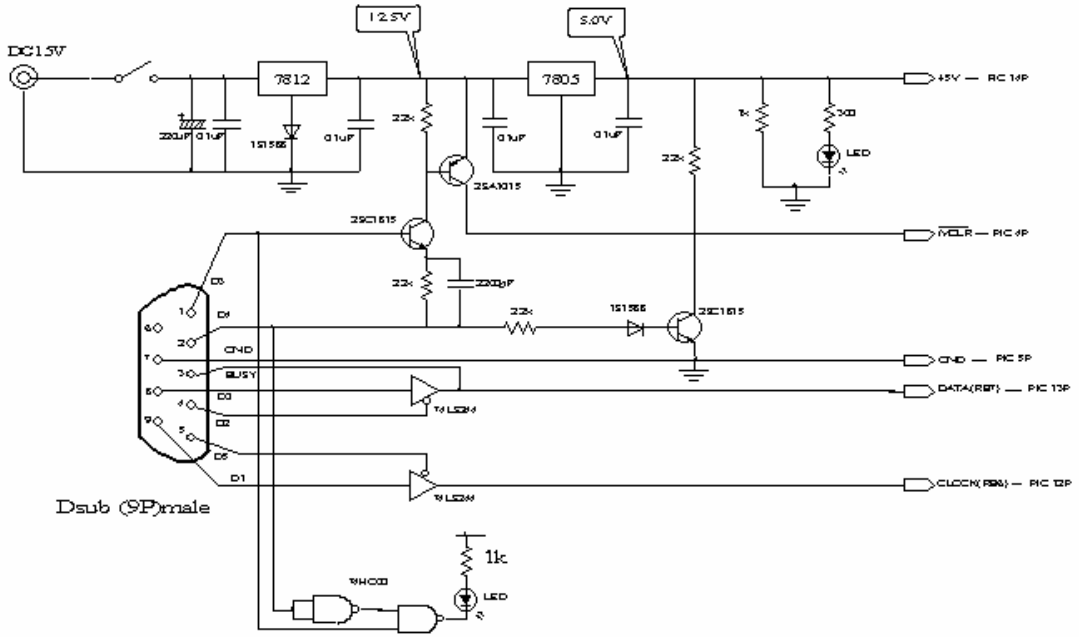
1.2.3. Mikrodenetleyici devreleri

Bu kursda yapacağınız uygulamalarda çeşitli mikrodenetleyici devrelerine ihtiyaç duyacaksınız. Bunun için uygulama örneklerinde devre şemalarının yapısını titiz bir şekilde inceleyiniz. Modüler yapıya sahip olan bu devrelerin blok şeması aşağıda görülmektedir.



Şekil 1.41. Mikrodenetleyici devreleri

Programlayıcı devresi



Şekil 1.42. Programlayıcı devresi

1.3. C dilinin bileşenleri

Bütün C programlarının temel yapısında program deyimleri (statements) ve fonksiyonlar vardır. Terimler, işlemleri gerçekleştiren program parçalarıdır. Bir C programı bir veya birden fazla fonksiyona sahip olabilir. Fonksiyonlar aynı zaman alt programlardır. Bir veya birden fazla program terimini içeren fonksiyonlar programın diğer fonksiyonları tarafından çağırılabilirler.

C dili ile programlamada satır girintileri, boş satırlar ve yorumlar programın anlaşılabilir olmasını sağlar. Bu sadece programı yazan için değil, programı inceleyen veya uygulayan diğer programcılar için önemlidir. Aşağıdaki örnek, bir C programının gerekli en temel bileşenlerini gösterir.

```
/* ilk C programım */           //1
#include <stdio.h>              //2

main()                          //3
{
    printf("merhaba C dili !"); //4
}
```

1. **/* ilk C programım */** bir yorum ifadesidir. Yorum ifadeleri **/*.....*/** arasında belirtilir ve birden fazla satır için yazılabilir.

Örnek : /* yorum ifadeleri programın tanımlanması ve
 program satırlarının açıklanması için gereklidir */

Programda her bir satır için yorum ifadesi kullanmak // karakterleri ile mümkündür.

Örnek: // yorum ifadeleri, programın tanımlanması ve
 // program satırlarının açıklanması için gereklidir

2. **#include <stdio.h>** ifadesi, programda **stdio (standard input output – standart giriş çıkış)** başlık dosyasına (header file) ait giriş çıkış fonksiyonlarının kullanıldığını belirtir. Başlık dosyaları .h uzantılıdır.

3. Bütün C programları **main()** fonksiyonuna sahiptir. Bu programın başlangıç noktasıdır.

```
fonksiyon ismi()
{
    program kodları
}
```

Fonksiyon içindeki program terimleri, açık durumdaki süslü parantezden { kapalı durumdaki süslü paranteze } kadar sıra ile işletilir. Bu parantezler {} fonksiyon içindeki program bloğunu gösterir.

4. **printf("merhaba C dili !");** stdio.h C kütüphanesinde tanımlanan formatlı yazım terimidir. C dilinde program terimlerinin sonunda noktalı virgül (;) kullanılır. Bu şekilde, program teriminin nerede son bulduğu ve takip eden terimin nerede başlayacağı derleyiciye bildirilir.

1.4. Mikrodenetleyici için C dilinin özellikleri

1.4.1. #include

Başlık dosyaları sadece temel C kütüphanelerinin değil, mikrodenetleyici donanımına ait port yazmaçları, zamanlayıcılar (timer0, timer1, watchdog timer), giriş-çıkış fonksiyonları gibi bilgileri de içerir. Bu dosyalar mikrodenetleyici C derleyicisinin içinde tanımlanmıştır.

Programcı tarafından oluşturulabilecek herhangi bir dosya da başlık dosyası olarak belirtilebilir. Eğer dosya, <> karakterleri arasında **#include<dosya ismi>** şeklinde yazılırsa, daha önce tanımlanmıştır. Derleme işleminde ilk defa kullanılacak dosyalar "" karakterleri ile **#include"dosya ismi"** olarak belirtilir.

Mikrodenetleyici tipi	Başlık dosyaları	#include<16f84a.h>
PIC16F84a	16f84a.h	:
PIC16F873	16f873.h	:
PIC16F877	16f877.h	:

Başlık dosyaları program terimi olmadığı için sonlarında noktalı virgül (;) kullanılmaz. Önışlemci olarak adlandırılan bu dosyalar derleme aşamasında kaynak kodlara eklenir.

1.4.2. Donanım özellikleri

Mikrodenetleyici C programında, kullanılacak donanım özellikleri belirtilmelidir. Aksi halde derleme işleminden sonra program uygulaması yanlış sonuçlar verir. Programı yazmadan önce donanım özellikleri aşağıdaki şekilde yazabiliriz.

```
#include<16f84a.h> //1
#fuses HS,NOWDT,PUT,NOPROTECT //2
#use delay(CLOCK=1000000) //3
#use rs232(BAUD=9600,XMIT=PIN_C6,RCV=PIN_C7) //4
:
:
```

- 1.Kullanılacak mikrodnetleyici donanım bilgilerini içerir.
- 2.Mikrodnetleyici donanımındaki osilatör tipini, watchdog (bekçi köpeği) zamanlayıcısını, power up zamanlayıcısını ve kod korumasını açıklar.
- 3.Osilatör hızını belirtir.
- 4.Seri iletişim hızı ve alıcı, verici pin numaralarını açıklar.

Aşağıdaki tablo, PIC 16F84 tipi mikrodnetleyiciye ait sigorta (fuses) açıklamalarıdır

Özellikler	Açıklamalar	
Osilatör çeşidi	LP	Dış kaynaktan 200kHz veya altında
	XT	Dış kaynaktan 4MHz veya altında
	HS	Dış kaynaktan 4MHz veya üstü
	RC	Dahili RC osilasyon
Watchdog timer	NOWDT	Watchdog timer kullanılmaz (Genellikle)
	WDT	Watchdog timer kullanılır.
Power-up timer	NOPUT	Power-up timer kullanılmaz.
	PUT	Power-up timer kullanılır (Genellikle)
Kod koruması	PROTECT	Kod koruması yapılır
	PROTECT_5%	Kod koruması %5
	PROTECT_50%	Kod koruması %50
	NOPROTECT	Kod koruması yok

Tablo 1.2. Mikrodnetleyici sigortaları (fuses)

Donanım tanımlamalarında, mikrodnetleyici yazmaçları belirtilebilir. Bu değerler bit veya byte seviyesinde olabilir.

```
#include<16f84a.h>
#fuses HS,NOWDT,PUT,NOPROTECT
#use delay(CLOCK=1000000)
#bit carry=3.0
#byte portb=6
#byte porta=5
:
:
```

1.4.3. Sabitler

Mikrodnetleyici ile C programlamada sabit, program işletiminde değiştirilmeyen değerlerdir. Bu değerler tamsayı (int), ondalıklı sayı (float) veya karakter (char) olabilir. Örneğin 75 veya -10 tamsayı, 456,78 ondalıklı sayı, 'A' veya 'w' ise karakter sabiti olarak gösterilebilir.

```
# define etiket değer
```

Sabitler `#define` terimi ile beraber kullanılır. Burada etiket, programda kullanacağınız değer için seçtiğiniz isimdir. Bu tanımlama ile program işletimi süresince sabitler bu isimle çağırılır. Sabitler hegzadesimal veya oktal sayı olarak da belirlenebilir. Hegzadesimal 2E sayısının gösterimi `0x2E` şeklindedir.

```
#include<16f84a.h>
#fuses HS,NOWDT,PUT,NOPROTECT
#use delay(CLOCK=1000000)
#define buton input(PIN_A0)
#define ALL_IN 0xff
#define ALL_OUT 0
:
:
```

Sabitler, program içindeki koşul veya işlemlerde de kullanılabilir.

```
#define BUTON_BAS input(PIN_A1)
:
:
if(BUTON_BAS)
    port_b=0x0f;
```

Sabitlerin mikrodenetleyici ROM belleği içinde saklanması için **const** terimi kullanılır.

```
int const display[4]={0x3f,0x06,0x5b,0x4f}
```

1.4.4. Ana fonksiyon

Mikrodenetleyici C programında program terimleri ve varsa diğer fonksiyonlar main fonksiyon içinde işletilir. Fonksiyon ile kullanılan parantezlerin () boş olması fonksiyon içinde herhangi bir parametre kullanılmadığını belirtir. Bu durumu göstermek için **void main()** ifadesi kullanılır.

```
#include<16f84a.h>
#fuses HS,NOWDT,PUT,NOPROTECT
#use delay(CLOCK=1000000)
void main()
{
    program kodları
}
```

1.4.5. Veri tipleri

C programlamada verilerin ne şekilde kullanılacağı çok önemlidir. Veri kısaltmalarının anlamları ve bunların alacağı sayısal değerler aşağıdaki tablolarda görülmektedir.

Veri tipi	Veri boyutu (bit)	İşareti	Veri aralığı
int1, short	1	unsigned	0 veya 1
int8, int	8	unsigned	0 ~ 255
		signed	-128 ~ 127
int16, long	16	unsigned	0 ~ 65536
		signed	-32768 ~ 32767
int32	32	unsigned	0 ~ 4,294,967,295
		signed	-2,147,483,648 ~ 2,147,483,647
float	32	signed	3.4E-38 ~ 3.4E+38
char	8	unsigned	0 ~ 255
		signed	-128 ~ 127

Tablo 1.3. Veri tipleri

short	Kısa aralıklı sayı
int	Tamsayı
long	Uzun aralıklı sayı
float	Ondalık sayı
char	Karakter
signed	Pozitif ve negatif
unsigned	Pozitif

C programlamada değişkenler aşağıdaki şekilde tanımlanır.

veri tipi <i>değişken ismi</i> ;	int a ;
<i>değişken ismi</i> = sayısal değer ;	a = 100 ;

Bu tanımlama birleştirilerek aynı anda bir çok değer farklı değişkenlere atanabilir.

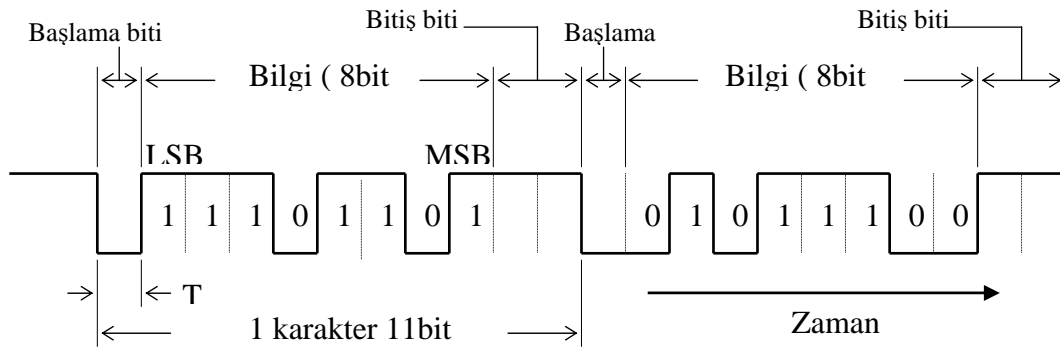
int a = 100 , b = 0 , c = 0xff ;

Bununla birlikte tanımlanmış bir değişkenin değeri başka bir değişkene atanabilir.

int a = 100 , b ;
b = a ;

DONANIM BİLGİSİ – Seri iletişim

Seri iletişimde bilgi iletimi aşağıdaki şekilde görülmektedir. Bilgi çıkışı iletişim hattında şekildeki gibi soldan sağa doğrudur. Bilgi sürekli 1 ise iletişim yok demektir. Bilgi gönderilmeye başlandığı zaman başlangıç biti olarak 1 bit 0 gönderilir. Sonra çıkış bilgisi ardı ardına en az bitle (LSB) iletilir. Şekilde 8 bitlik bir bilgi gönderilmesi örneklenmiştir. Daha sonra gönderilen bilginin sonunu belirtmek için bitiş biti gönderilir. Bu stop biti şekilde görüldüğü gibi 2 bitten meydana gelmektedir. Böylece bir karakter 1 bit başlangıç biti, 8 bit bilgi biti, 2 bitte bitiş biti olmak üzere 11 bitten iletişim sağlayabilmektedir. Alıcı (receiver), bir karakter için başlangıç ve bitiş bitini ayırarak bilgi bitini de ekleyerek paketleme yapar.

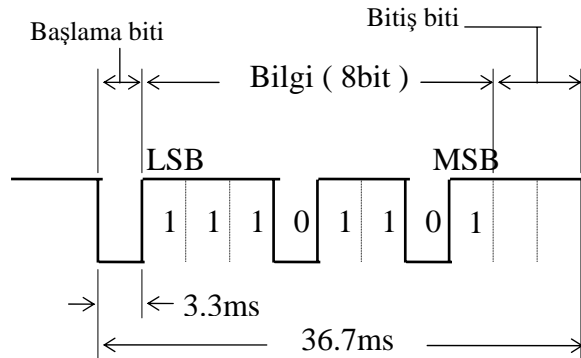


Bilgi iletişim hızı 1 saniyede gönderilen bitlerin sayısı ile ifade edilir. Birim olarak (bit per second) bps kullanılır. Tipik olarak veri iletim hızları 300, 600, 1200, 2400, 4800, 9600 bps dir.

Bilgi iletim hızı 300 bps olan dalganın bit genişliği aşağıdaki gibidir.

$$T = \frac{1}{bps} = \frac{1}{300} = 0.00333 = 3.3[ms]$$

300 bps den anladığımız bir karakter 11 bitten oluştuğundan dolayı bir karakterin gidiş hızı $11 \times 3,3 = 36,3ms$ olur. Ayrıca 1 saniyede giden karakter sayısını da yaklaşık olarak bulmak istersek $300bps / 11bit = 27$ karakter olur.

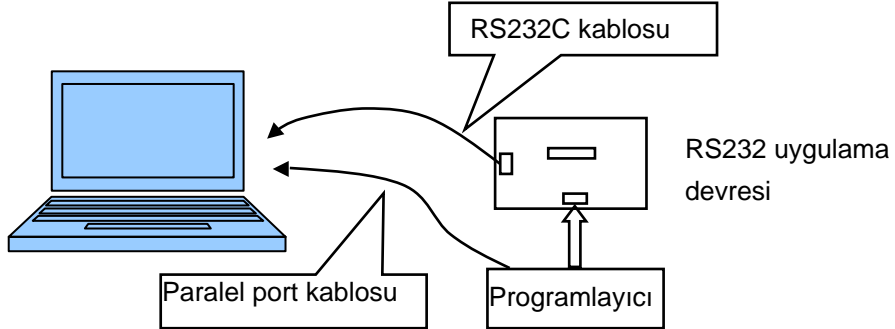
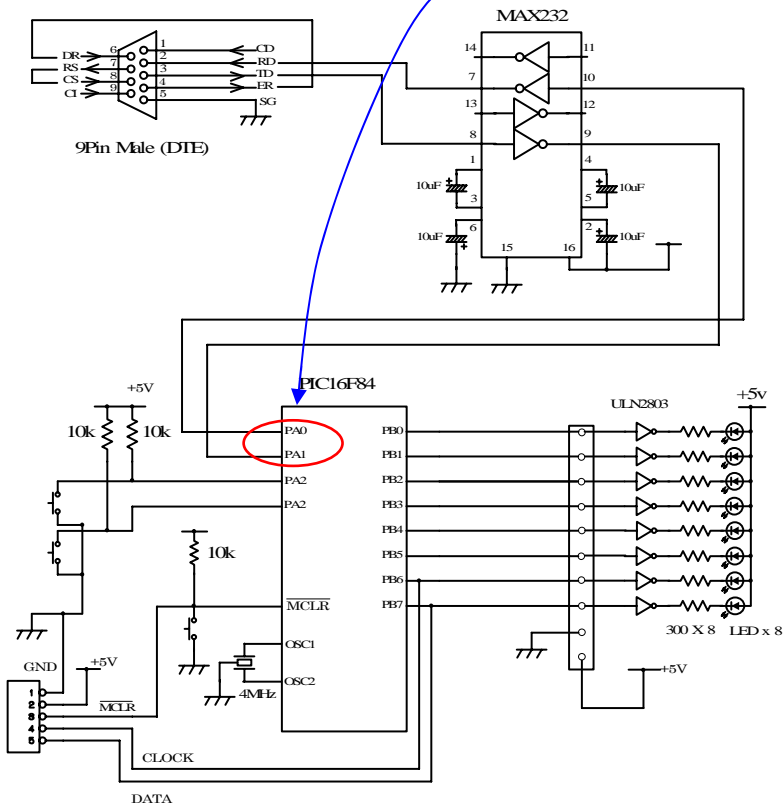


DONANIM BİLGİSİ – Seri iletişim

Seri iletişim ile ilgili uygulamalarınızda aşağıdaki devreyi kullanabilirsiniz. Bu devrede RS-232 entegresinin 7 ve 10 nolu uçları verici olarak mikrodnetleyicinin PA0 pinine, 8 ve 9 nolu uçları da alıcı olarak PA1 pinine bağlanmıştır.

Buna göre, yazacağımız programda seri iletişim tanımlamasını aşağıdaki şekilde yapabiliriz.

```
#use rs232(BAUD=9600,XMIT=PIN_A0,RCV=PIN_A1)
```



1.4.6. Formath yazım (printf) fonksiyonu

Standart C dili kütüphane dosyasında bulunan printf fonksiyonu, belirli bir veriyi bilgisayar ekranında görüntülemek için kullanılır. Bu işlem, mikrodenetleyici ile bilgisayar arasında kurulan **seri iletişim (RS 232)** bağlantısı ile yapılır. Fonksiyonun genel yazımı aşağıdaki şekildedir.

```
printf (“ string ” , değişken veya sayısal değerler ) ;
```

Printf fonksiyonunda string olarak herhangi bir karakter, sayı veya sembol grubu yazılabilir. Fonksiyon içinde kullanılan bazı özel karakterler vardır ki bunlara tip dönüşüm belirteçleri denir. Tip dönüşüm belirteçleri % karakteri ile kullanılırlar ve fonksiyonda tanımlanmış değişken veya sayısal değerleri göstermek için kullanılırlar. Aşağıda printf fonksiyonunun kullanım şekilleri görülmektedir.

```
printf (“ Mazhar Zorlu Anadolu Teknik Lisesinde ” ) ;
printf (“ Okul Numaram %d dir” , 28) ;
```

```
Mazhar Zorlu Anadolu Teknik Lisesinde Okul Numaram 28 dir
```

```
int top = 25 ;
float ort = 10.25 ;
printf (“ toplam sonuc = %d ortalama = % f dir ”, top, ort) ;
```

```
toplam sonuc = 25 ortalama = 10.25 dir
```

Tip Dönüşüm Belirteçleri	Fonksiyon
%d	int tipindeki değerleri desimal hale dönüştürür.
%ld	long tipindeki değerleri desimal hale dönüştürür.
%x , %X	int tipindeki değerleri hegzadesimal hale dönüştürür. (%x a' dan f' ye kadar olan küçük karakterleri kullanır. %X ise A dan F ye kadar olan büyük harfleri kullanır.)
%f	Float ve double tipindeki değerleri desimal ve gerçel hale dönüştürür.
%c	Kodu karaktere dönüştürür.
%s	String ifadeyi gösterir.
%p	Adresi gösterir.
%%	% karakterini gösterir.

Tablo 1.4. Tip dönüşüm belirteçleri

Çıktı kodu	Fonksiyon
\n	İmleci bir sonraki satırın başına konumlandırır.
\t	İmleci bir sonraki tab konumuna götürür.
\r	İmleci satır başına getirir.
\b	İmleci sola kaydırır.
\f	Sayfa atlama
\v	Dikey boşluk
\a	Bilgisayarın zilini çalar
\"	” çift tırnak işaretini gösterir
'	' tırnak işaretini gösterir
\\	\ karakterini gösterir.

Tablo 1.5. Çıktı kodu karakterleri

Formatlı yazım fonksiyonunda, tip dönüşüm belirteci ile kullanılan sayılar, ekran çıktısı için ayrılan alanın genişliğini belirler.

```
#include<16f84a.h>
#fuses HS,NOWDT,PUT,NOPROTECT
#use delay (CLOCK=1000000)
#use rs232(BAUD=9600,XMIT=PIN_A0,RCV=PIN_A1)
void main()
{
    int top=25;
    float ort=10.25;
    printf("toplama sonuc=%5d ortalama=%6.2f dir",top,ort);
}
```

Bu programın ekran çıktısı aşağıdaki gibidir.

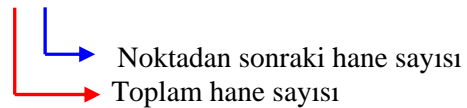
```
toplama sonuc = __ _25 ortalama = _10.25 dir
```

Fonksiyonda kullanılan %5d ifadesinde 5 rakamı, desimal sayının gösterimi için kullanılacak hane sayısını belirtir, %6.2f ise gerçel sayının 6 haneli olacağını noktadan sonra 2 hane kullanılacağını belirtir.

%5d



%6.2f



C derleyicisi aşağıdaki seri iletişim giriş çıkış fonksiyonlarını destekler.

Fonksiyon	Anlamı
<code>getc()</code> <code>getch()</code> <code>getchar()</code>	RS232 alıcı pinden gelen karakteri bekler ve geri döndürür. Kullanım : <code>value=getc()</code> , <code>value=getch()</code> , <code>value=getchar()</code>
<code>putc()</code>	Rs232 gönderici pin üzerinden karakter gönderir. Fonksiyonun geçerli olması için RS232 pinleri ve baud oranı tanımlanmış olmalıdır. Kullanım : <code>putc=karakter</code> , <code>putchar=karakter</code> (karakter=8 bit veri)
<code>gets()</code>	Bilgisayar klavyesinden string ifadeyi alır. Kullanım : <code>gets(string)</code>
<code>puts()</code>	String ifadeyi RS232 pin üzerinden gönderir. Kullanım : <code>puts(string)</code>
<code>printf()</code>	String ifadeyi ekranda gösteren formatlı yazım fonksiyonudur.
<code>kbhit()</code>	Klavyede herhangi bir tuşun basılmasını algılar. <code>value = kbhit()</code>
<code>set_uart_speed</code>	Seri iletişim hızını değiştirir.

Tablo 1.6. Aritmetik operatörler

1.4.7. Operatörler

1.4.7.1. Aritmetik operatörler

Sembol	İşlem
+	$a+b$ → a ve b değişkenlerinin toplamı
-	$a-b$ → a ve b değişkenlerinin farkı $a=-a$ → a değişkeninin işaret değişimi
*	$a*b$ → a ve b değişkenlerinin çarpımı
/	a/b → a değişkeninin b ye bölümü
%	$a\%b$ → a değişkeninin b ye bölümünden kalan

Tablo 1.7. Aritmetik operatörler

1.4.7.2. Karşılaştırma operatörleri

Sembol	Anlamı	İşlem	Anlamı
<	küçük	$a<b$	a küçüktür b
>	büyük	$a>b$	a büyüktür b
<=	küçük eşit	$a<=b$	a küçük eşit b
>=	büyük eşit	$a>=b$	a büyük eşit b
==	eşit	$a==b$	a eşittir b
!=	eşit değil	$a!=b$	a eşit değil b

Tablo 1.7. Karşılaştırma operatörleri

1.4.7.3. Mantıksal operatörler

Sembol	Anlamı	İşlem	Anlamı
&&	VE (AND)	(a>3) && (a<10)	a büyüktür 3 ve a küçüktür 10
	VEYA (OR)	(a>=3) (b<=10)	a büyük eşit 3 veya b küçük eşit 10
!	DEĞİL (NOT)	!(a==b)	a eşit değil b

Tablo 1.8. Mantıksal operatörler

1.4.7.4. Bit işlem operatörleri

Sembol	Anlamı	İşlem	Anlamı
&	VE (AND)	a&b	a ile b sayısının and işlemi
	VEYA (OR)	a b	a ile b sayısının or işlemi
^	XOR (ÖZELVEYA)	a^b	a ile b sayısının özelveya işlemi
~	1. tümleyeni	~a	a sayısının 1. tümleyeni
<<	sola kaydırma	a<<n	a sayısını n bit sola kaydırma
>>	sağa kaydırma	a>>n	a sayısını n bit sağa kaydırma

Tablo 1.9. Bit işlem operatörleri

a&b	a b	a^b
a→00001101	a→00001101	a→00001101
b→00000111	b→00000111	b→00000111
00000101	00001111	00001010
~a	a<<3	a>>3
a→00001101	a→00001101	a→00001101
11110010	01101000	00000001

C dilinde aritmetik işlemlerin kısa yolları aşağıdaki tabloda görülmektedir.

Kısayol	İşlem	Anlamı
a+=b	a=a+b	a ile b toplamını a değişkenine ata
a-=b	a=a-b	a ile b farkını a değişkenine ata
a*=b	a=a*b	a ile b çarpımını a değişkenine ata
a/=b	a=a/b	a ile b bölümünü a değişkenine ata
a%=b	a=a%b	a ile b modunu a değişkenine ata
a<<=b	a=a<<b	a sayısını b kadar sola kaydır ve a değişkenine ata
a>>=b	a=a>>b	a sayısını b kadar sağa kaydır ve a değişkenine ata
a&=b	a=a&b	a ile b sayısının AND işlemi sonucunu a değişkenine ata
a^=b	a=a^b	a ile b sayısının XOR işlemi sonucunu a değişkenine ata
a =b	a=a b	a ile b sayısının OR işlemi sonucunu a değişkenine ata

Tablo 1.10. Aritmetik işlem kısa yolları

1.4.7.5. Artırma ve eksiltme operatörleri

Kısayol	İşlem	Anlamı
a++ , ++a	a=a+1	a değişkenini 1 arttır ve a değişkenine ata
a-- , --a	a=a-1	a değişkenini 1 eksilt ve a değişkenine ata

Tablo 1.11. Artırma ve eksiltme operatörleri

1.4.7.6. İşlem öncelikleri

Aşağıdaki tabloda işlem öncelikleri yukarıdan aşağıya sıralanmıştır.

Öncelik	İşlemler
1	()
2	! ~ a++ ++a -a a-- sizeof
3	* / %
4	+ -
5	<< >>
6	< <= > >=
7	== !=
8	&
9	^
10	
11	&&
12	
13	= += -= *= /= %= >>= <<= &= ^= =

Tablo 1.12. İşlem öncelikleri

İşlem öncelikleri ve operatörler hakkında aşağıdaki örnekleri inceleyiniz.

$$a = 15 - 3 * 5 \rightarrow a = 0$$

$$b = (15 - 3) * 5 \rightarrow b = 60$$

$$c = 20 \% 3 * 5 \rightarrow c = 10$$

$$d = 5 * 20 \% 3 \rightarrow d = 1$$

$$e = \sim 0x00 \& 0x0f \rightarrow e = 15$$

$$f = 0b00001111 \wedge 0b11110000 \rightarrow f = 255$$

Örnek 1.1.

```

/* Ornek 1.1. ---- RS-232 devresi kullanılacaktır */

#include<16f84a.h>
#fuses HS,NOWDT,PUT,NOPROTECT
#use delay (CLOCK=10000000)
#use rs232(BAUD=9600,XMIT=PIN_A0,RCV=PIN_A1)
void main()
{
    printf("\r merhaba !!\n");
    printf("\r mikrodenetleyicide C dili %d numara\n",1);
}

```

Örnek 1.2.

```

/* Ornek 1.2. ---- RS-232 devresi kullanılacaktır */

#include<16f84a.h>
#fuses HS,NOWDT,PUT,NOPROTECT
#use delay (CLOCK=10000000)
#use rs232(BAUD=9600,XMIT=PIN_A0,RCV=PIN_A1)
void main()
{
    int i,j;
    printf("\r bolum=%d mod=%d\n",5/3,5%3);
    i=3;
    printf("\r i=%d\n",i);
    printf("\r i++=%d\n",i++);
    printf("\r i=%d \n",i);
    printf("\r ++i=%d\n",++i);
    i=15 - 3 * 5;
    printf("\r i=%d\n",i);
    j=20 % 3 * 5;
    printf("\r j=%d\n",j);
    printf("\r-----\n");
}

```

Örnek 1.3.

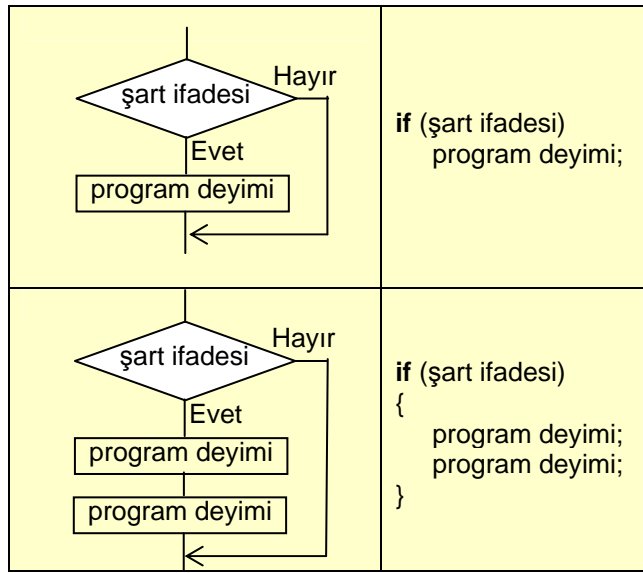
```
/* Ornek 1.3. ---- RS-232 devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOWDT,PUT,NOPROTECT
#use delay (CLOCK=4000000)
#use rs232(BAUD=9600,XMIT=PIN_A0,RCV=PIN_A1)
void main()
{
    int i,j,k;
    i=0x0f;
    j=0xf0;
    k=i&j;
    printf("\r\n i&j==>%d\n",k);
    k=i^j;
    printf("\r\n i^j==>%d\n",k);
    k=i|j;
    printf("\r\n i|j==>%d\n",k);
    printf("\r\n kaydirma islemi=%2X %2X %2X\n",i,i<<2,i>>2);
    printf("\r-----");
}
```

2. Mikrodenetleyici Programlamada Kontrol Yapıları

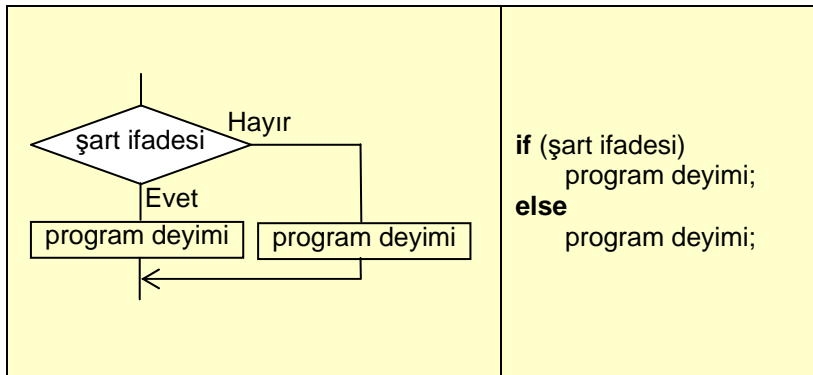
2.1. Eğer (if) karar yapısı

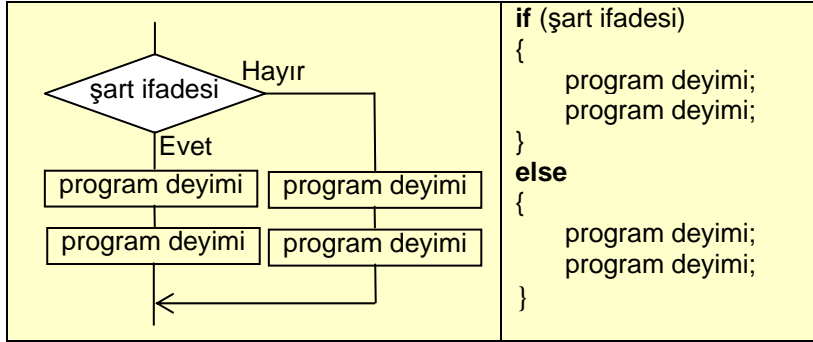
Mikrodenetleyici için C programlamada karar yapısı olarak, if deyimi kullanılır. Karar yapısına gelen bir değer, tanımlanan şart ifadesine göre doğru ise if deyiminin altındaki program deyimi veya deyimleri çalıştırılır. if deyiminin bu şekildeki kullanımı tek yönlü karar ifadesi olarak adlandırılır. Tek yönlü karar yapısı aşağıdaki gibidir.



Tablo 2.1. Tek yönlü karar yapısı

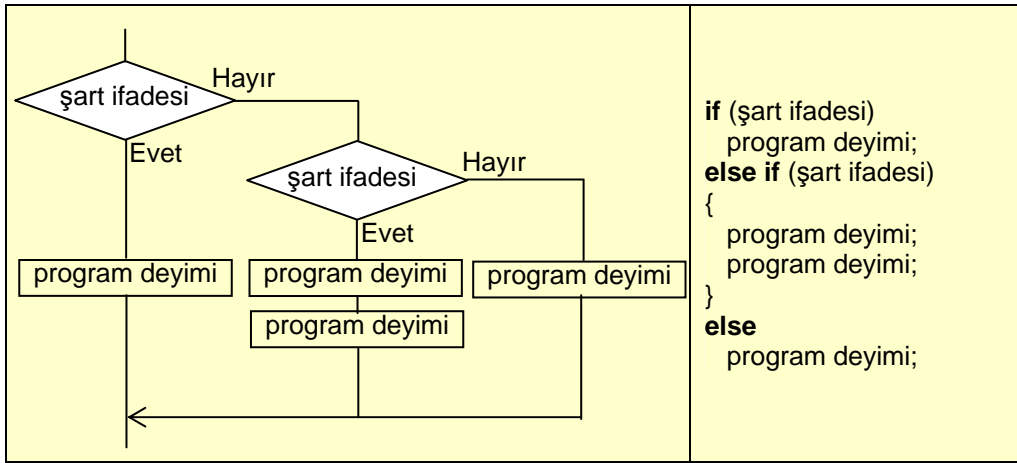
Tek yönlü karar yapısında şart ifadesine göre doğru olan değerler için program deyimleri yürütülüyordu. Çift yönlü karar yapısında ise şartın dışındaki değerler için else ifadesi kullanılır.





Tablo 2.2. Çift yönlü karar yapısı

Çift yönlü karar yapısında şart ifadesine uymayan değerler için başka bir şart ifadesi kullanılıyorsa böyle bir yapı iç içe karar yapısı olarak adlandırılır. İç içe karar yapısında her şart ifadesi bir öncekine bağlıdır.



Tablo 2.3. İç içe karar yapısı

if karar yapısında şart ifadesi olarak karşılaştırmalı operatörlerin yanında mantıksal operatörler de kullanılır. 1. öğrenme faaliyetinde gördüğümüz bu operatörleri if karar yapısında aşağıdaki gibi kullanabilirsiniz.

```

if(not<45)                if(not= =100)            if(not!=0)
    printf("notunuz 45den kucuk");    printf("notunuz 100");    printf("notunuz 0 degil");
                    
```

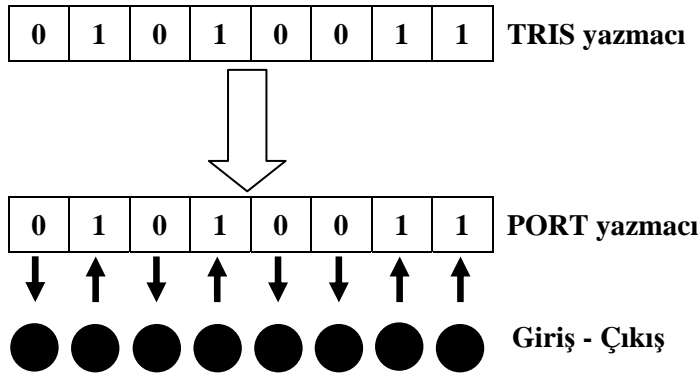
```

if(not>=45 && not<=55)    if(not1>80 || not2> 80)
    printf("notunuz 45 ve 55 arasında");    printf("notlarinizdan en az biri 80 den buyuk ");
                    
```

DONANIM BİLGİSİ – Mikrodenetleyicide Port Tanımlamaları

Mikrodenetleyici içerisinde birçok giriş çıkış (I/O) ucu vardır. Her biri 1 bit'e karşılık gelen bu uçlar 8 bitlik üniteler halindedir. Bu ünitelere "port" adı verilir ve mikrodenetleyici yapısı içerisinde port yazmaçları tarafından kontrol edilir.

Mikrodenetleyicinin giriş – çıkışları TRIS yazmacı ile kontrol edilir. Bu uçların 1 veya 0 olmasına göre portların durumu belirlenir. Eğer port ucu 0 ise çıkış olarak , 1 ise giriş olarak dış devrede değerlendirilir. Başka bir ifade ile çıkış ucu **0:low** , giriş ucu **1:high** olarak adlandırılır. Bu portlar TRIS yazmacı ile giriş veya çıkış olarak tanımlanabilir. Aşağıdaki şekil TRIS yazmacı ile fiziksel portlar arasındaki ilişkiyi göstermektedir.



Mikrodenetleyici için port tanımlaması aşağıdaki gibidir.

02	
03	
04	
05	PORT A
06	PORT B
07	
08	
09	

BANK 0

#byte port_a=5
#byte port_b=6

DONANIM BİLGİSİ – Dahili Fonksiyonlar

Derleyici tarafından sunulan çeşitli dahili fonksiyonlardan bazıları (built-in functions) aşağıdaki tabloda açıklanmaktadır.

Fonksiyon tipi	Açıklamalar
<code>output_bit(pin,değer)</code>	Belirlenen çıkışın 1 veya 0 olma durumu <code>output_bit(PIN_B0,0)</code>

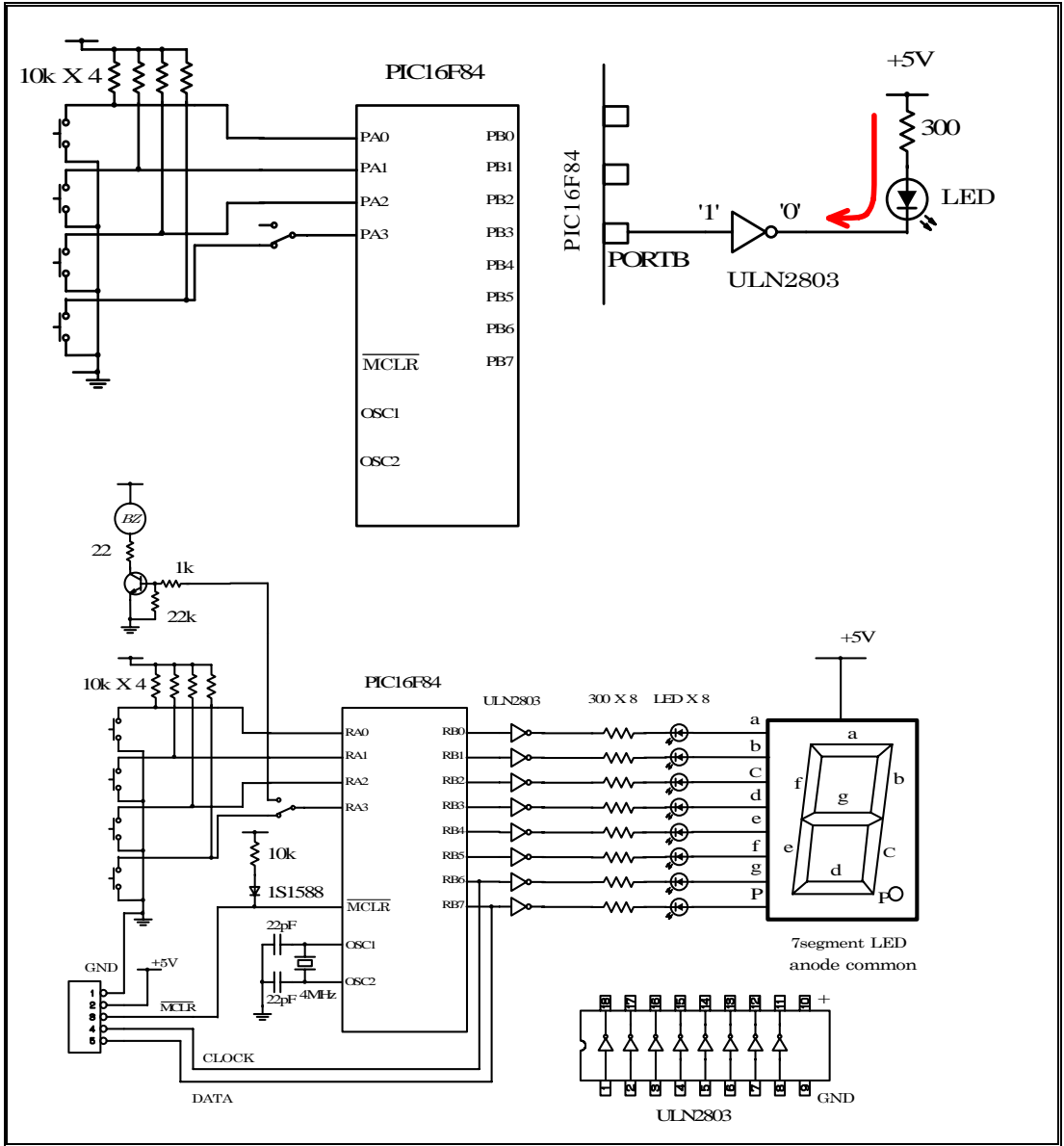
<code>output_float(pin)</code>	Herhangi bir portun çıkış olarak atanması veya yüksek empedans haline getirilmesi output_float(PIN_A0)
<code>output_high(pin)</code>	Belirlenen çıkışın 1 olma durumu output_high(PIN_A0)
<code>output_low(pin)</code>	Belirlenen çıkışın 0 olma durumu output_low(PIN_A0)
<code>port_b_pull-ups(değer)</code>	Port B de pull-up kontrolü değer= FALSE (pull-ups etkin değil) değer=TRUE (pull-ups etkin) port_b_pull-ups(FALSE)
<code>a=input(pin)</code>	Giriş bilgisinin a değişkenine atanması a: short int tipi değişken (0 veya 1) 1=input(PIN_A0) veya !input(PIN_A0)
<code>set_tris_x(değer)</code>	Port X in çıkış veya giriş olarak belirlenmesi Değer 0 olursa 8 bitlik port çıkış , 1 olursa giriş olur. X'in değeri Mikrodenetleyici yapısına göre A, B, C, D, veya E olabilir. set_tris_b(0x00)
<code>output_x(değer)</code>	Değer : 8 bitlik portun tamsayı (int) değeri output_b(0xF0) 0xF0 port bilgisinin onaltılık olarak gösterimidir. Burada "F" port 7-4 "0" ise port 3-0 ü temsil eder
<code>değer=input_x()</code>	değer : tam sayı (int) türünde değişkendir ve fonksiyon tamsayı olarak geri döner. input_x() : 8 bit değerindeki port bilgisi data = input_b()

DONANIM BİLGİSİ – LED – Display Kontrol Devresi

Şekildeki devrede A portu (RA0 – RA3) giriş, B portu (RB0 – RB7) ise çıkış olarak kullanılmıştır.

A portuna bağlanmış olan butonlara basılmadığı sürece ilgili pinlere pull – up dirençleri üzerinden mantık "1" bilgisi gelmektedir. Butonlara basıldığı zaman mantık "0" bilgisi verilir.

B portu pinlerine mantık "1" bilgisi verildiğinde ULN 2803 entegresi ile bu bilgi terslenir ve bağlı olan led katoduna "0" bilgisi, anod ucuna kaynak gerilimi geldiği için led ışık verir.



Örnek 2.1.

Giriş – çıkış fonksiyonlarının kullanımını inceleyerek anlayabiliriz. Devrede mikrodenetleyicinin A portu giriş, B portu çıkış olarak belirlenmiştir. Giriş portlarına ait butonlara basıldığında aynı bit değerine karşılık gelen ledler ışık vermektedir.

Bu işlem “while” sonsuz döngüsü ile sürekli hale getirilmiştir.

```

/* Ornek 2.1. ---- LED - Display devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOWDT,PUT,NOPROTECT
#byte port_a=5           // a portunun tanımlanması
#byte port_b=6           // b portunun tanımlanması
void main()
{
    set_tris_a(0x0f);
    set_tris_b(0x00);
    while(1)
    {
        output_b(input_a());    // Port B <----- Port A
    }
}

```

Not : Yukarıdaki ifadeyi **port_b=(port_a)&0x1f** şekline yazabiliriz.

Örnek 2.2.

A portunda herhangi bir bit tanımlamasıyla b portunda bit kontrolü aşağıdaki şekilde yapılabilir.

```

/* Ornek 2.2. ---- LED - Display devresi kullanılacaktır */

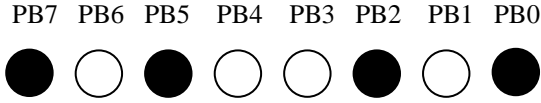
#include<16f84a.h>
#fuses XT,NOWDT,PUT,NOPROTECT
#byte port_a=5           // a portunun tanımlanması
#byte port_b=6           // b portunun tanımlanması
void main()
{
    set_tris_a(0x0f);    // a portunun giriş olarak atanması
    set_tris_b(0x00);    // b portunun çıkış olarak atanması

    while(1)             // sonsuz döngü
    {
        output_bit(PIN_B0,input(PIN_A0));    // PB0 <----- PA0
        output_bit(PIN_B1,input(PIN_A1));    // PB1 <----- PA1
    }
}

```

Örnek 2.3.

Aşağıdaki program, ledlerin aşağıdaki şekilde kontrolünü sağlar



```

/* Ornek 2.3. ---- LED - Display devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOVDT,PUT,NOPROTECT
#byte port_b=6          // b portunun tanımlanması

void main()
{
    set_tris_b(0x00);    // b portunun çıkış olarak atanması
    port_b=0;           // b portunun 0 değerinin atanması
    port_b=0xa5;        // b portuna 10100101 değerinin atanması
}

```

Örnek 2.4.

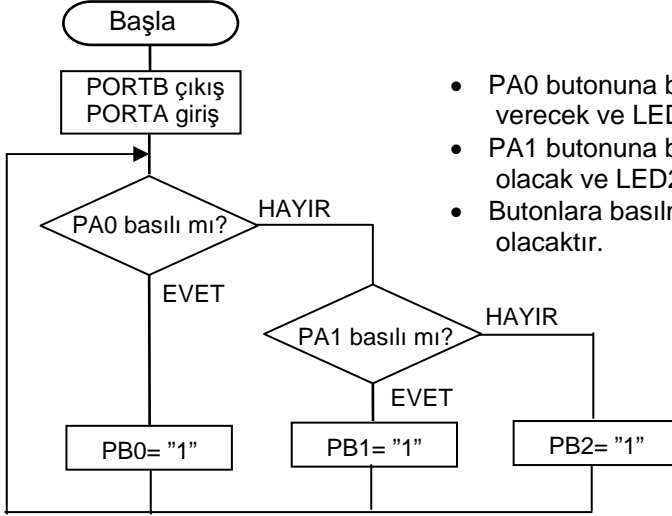
A0 pinine bağlı butona basıldığında ledleri yukarıdaki şekilde kontrol edelim.

```

/* Ornek 2.4. ---- LED - Display devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOVDT,PUT,NOPROTECT
#byte port_a=5          // a portunun tanımlanması
#byte port_b=6          // b portunun tanımlanması
void main()
{
    set_tris_a(0x0f);    // a portunun giriş olarak atanması
    set_tris_b(0x00);    // b portunun çıkış olarak atanması
    port_b=0;           // b portunun 0 değerinin atanması
    while(1)            // sonsuz döngü
    {
        if(input(PIN_A0)==0) // A0 butonuna basıldı mı?
            port_b=0x54;
    }
}

```

Örnek 2.5.

- PA0 butonuna basıldığında, LED1(PB0) ışık verecek ve LED2(PB1) sönmek olacaktır.
- PA1 butonuna basıldığında, LED1(PB0) sönmek olacak ve LED2(PB1) ışık verecektir.
- Butonlara basılmadığı sürece ledler sönmek olacaktır.

```

/* Ornek 2.5. ---- LED - Display devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOVDT,PUT,NOPROTECT
#byte port_a=5 // a portunun tanımlanması
#byte port_b=6 // b portunun tanımlanması
void main()
{
    set_tris_a(0x0f); // a portunun giriş olarak atanması
    set_tris_b(0x00); // b portunun çıkış olarak atanması
    while(1)
    {
        if(input(PIN_A0)==0) //PA0 basılı=[0] basılı değil=[1]
        {
            port_b=1; // port_b <-- 00000001
            break;
        }
        else if(input(PIN_A1)==0) // PA0 basılı=[0] basılı değil=[1]
        {
            port_b=2; // port_b <-- 00000010
            break;
        }
        else
            port_b=0; // port_b <-- 00000000
    }
}

```

Örnek 2.6.

```

/* Örnek 2.6. ---- LED - Display devresi kullanılacaktır */

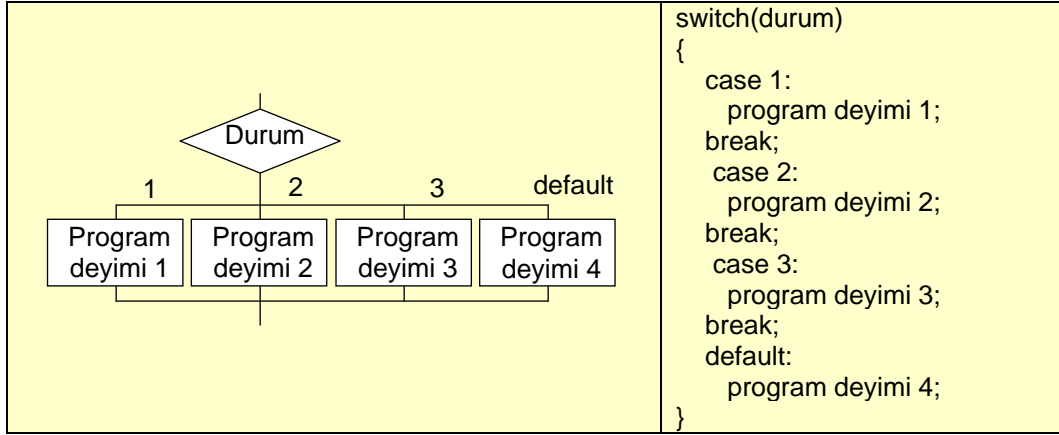
#include<16f84a.h>
#fuses XT,NOWDT,PUT,NOPROTECT
#byte port_a=5 // a portunun tanımlanması
#byte port_b=6 // b portunun tanımlanması
#define buton_1 input(PIN_A0) // butonların tanımlanması
#define buton_2 input(PIN_A1)
#define buton_3 input(PIN_A2)
#define buton_4 input(PIN_A3)

void main()
{
    set_tris_a(0x0f); // a portunun giriş olarak atanması
    set_tris_b(0x00); // b portunun çıkış olarak atanması
    while(1)
    {
        if((buton_1)&&(buton_2)) // buton 1 ve 2 ye basılmadı ise
            output_b=0; // port_b <-- 00000000
        if(!buton_1)&&!buton_2) // buton 1 ve 2 ye basıldı ise
            output_b(0xff) ; // port_b <-- 11111111
        if(!buton_3)&&(buton_4) // buton 3 e basıldı ise
            output_bit(PIN_B2,1); // port_b <-- 00000100
        if((buton_3)&&!buton_4) // buton 4 e basıldı ise
            output_bit(PIN_B3,1); // port_b <-- 00001000
    }
}

```

2.2. Çoklu karar yapısı (switch case)

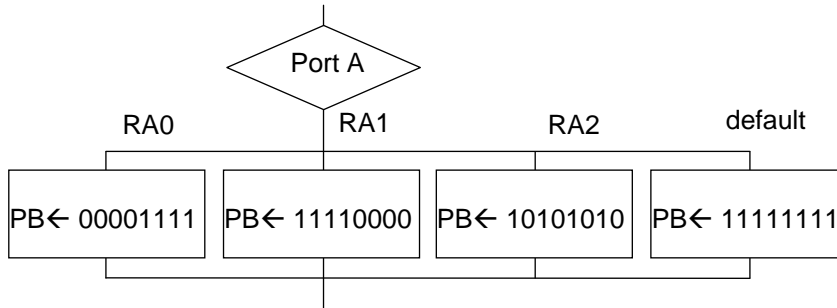
İç içe karar yapısındaki durumların kontrolü birbirine bağlı şartlara bağlıydı. Çoklu karar yapısında (switch-case) ise belirlenen durumların kendi aralarında bir önceliği yoktur. Aşağıdaki tabloda görüldüğü gibi belirlenen durum 1, 2 veya 3 değerlerine uyuyorsa, ilgili program deyimleri işletilir. Diğer değerler için default ifadesinin altındaki program deyimini geçerlidir.



Tablo 2.4. Çoklu karar yapısı

Örnek 2.7.

- RA0 butonuna basıldığında, B portu 00001111
- RA1 butonuna basıldığında, B portu 11110000
- RA2 butonuna basıldığında, B portu 10101010 durumunu alacaktır.
- Bunların dışında B portunun durumu 11111111 şeklinde olacaktır.



```

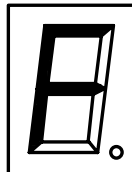
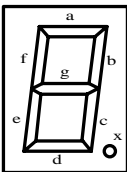
/* Ornek 2.7. ---- LED - Display devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOWDT,PUT,NOPROTECT
#byte port_a=5          // a portunun tanımlanması
#byte port_b=6          // b portunun tanımlanması
void main()
{
    int bilgi;
    set_tris_a(0x0f);    // a portunun giriş olarak atanması
    set_tris_b(0x00);    // b portunun çıkış olarak atanması
    while(1)             // sonsuz döngü
    {
        bilgi=port_a;    // a portunun okunması
        switch(bilgi)
        {
            case 14:      // port_a <-- 1110
                output_b(0x0f); // port_b <-- 00001111
                break;
            case 13:      // port_a <-- 1101
                output_b(0xf0); // port_b <-- 00001111
                break;
            case 11:      // port_a <-- 1011
                output_b(0xaa); // port_b <-- 00001111
                break;
            case 7:       // port_a <-- 0111
                output_b(0x00); // port_b <-- 00000000
                break;
        }
    }
}

```

Örnek 2.8.

Aşağıdaki şekilde 7 elemanlı display yapısı görülmektedir. Bu yapıya göre, A portuna bağlı butonlara 2 tabanlı sayı sistemine göre basıldığında display 0 ile F arasındaki sayıları gösterecektir.



Display bilgisi	x	g	f	e	d	c	b	a
PORTB	0	1	1	1	0	0	0	1

```

/* Ornek 2.8. ---- LED - Display devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOVDT,PUT,NOPROTECT
#byte port_a=5          // a portunun tanımlanması
#byte port_b=6          // b portunun tanımlanması
void main()
{
    set_tris_a(0x0f);    // a portunun giriş olarak atanması
    set_tris_b(0x00);    // b portunun çıkış olarak atanması
    int bilgi;
    while(1)             // sonsuz döngü
    {
        bilgi=~port_a&0x0f; // a portunun okunması *
        switch(bilgi)
        {
            case 0x0:port_b=0x3f;break; // port_a <-- 0000
            case 0x1:port_b=0x06;break; // port_a <-- 0001
            case 0x2:port_b=0x5b;break; // port_a <-- 0010
            case 0x3:port_b=0x4f;break; // port_a <-- 0011
            case 0x4:port_b=0x66;break; // port_a <-- 0100
            case 0x5:port_b=0x6d;break; // port_a <-- 0101
            case 0x6:port_b=0x7d;break; // port_a <-- 0110
            case 0x7:port_b=0x07;break; // port_a <-- 0111
            case 0x8:port_b=0x7f;break; // port_a <-- 1000
            case 0x9:port_b=0x6f;break; // port_a <-- 1001
            case 0xa:port_b=0x77;break; // port_a <-- 1010
            case 0xb:port_b=0x7c;break; // port_a <-- 1011
            case 0xc:port_b=0x58;break; // port_a <-- 1100
            case 0xd:port_b=0x5e;break; // port_a <-- 1101
            case 0xe:port_b=0x79;break; // port_a <-- 1110
            case 0xf:port_b=0x71;break; // port_a <-- 1111
        }
    }
}

```

* Açıklama :

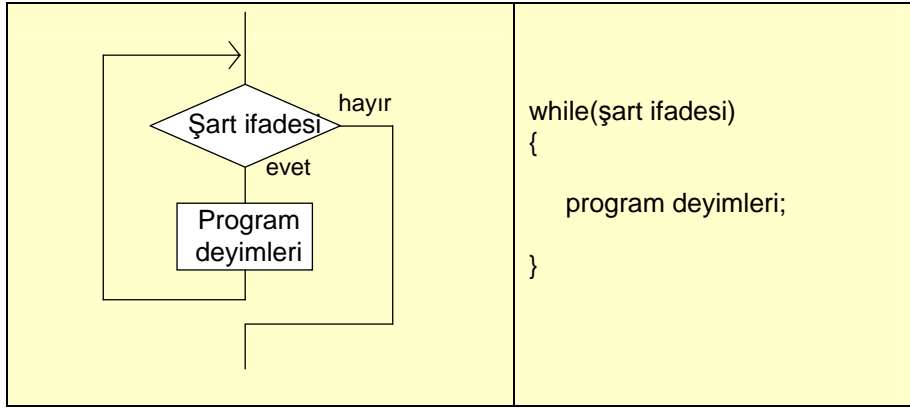
~port_a&0x0f	0001&0x0f
~port_a → 00000000	~port_a → 00000001
0x0f → 00001111 &	0x0f → 00001111 &
bilgi ← 00000000	bilgi ← 00000001

2.3. Döngü yapıları

Her programlama dilinde olduğu gibi mikrodenetleyici için C programlamada bir program parçasının yinelemeli olarak çalıştırılmasını sağlamak için döngüler kullanılır. Genel olarak döngüler aşağıdaki gibi sınıflandırılır;

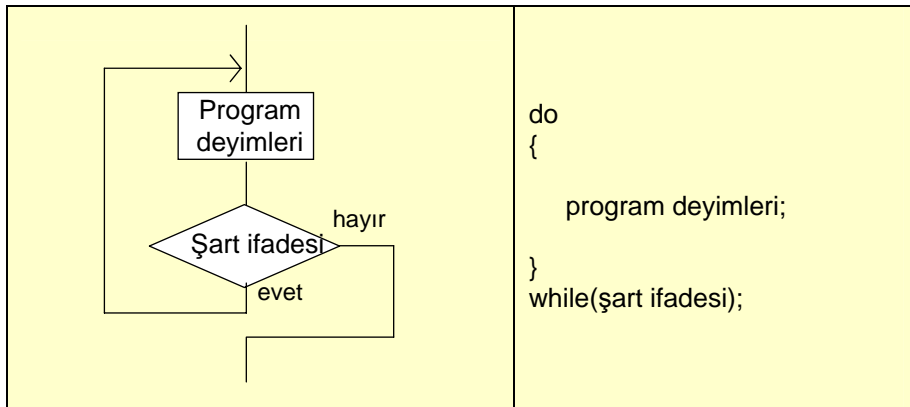
- Kontrolün başta yapıldığı **while** döngüleri
- Kontrolün sonda yapıldığı **while** döngüleri (**do while** döngüleri)
- **for** döngüleri

Aşağıdaki genel akış diyagramına göre **while** döngüsünde şart ifadesi döngü içerisine girilmeden önce kontrol edilmektedir. Şart ifadesi doğru olduğunda, döngü içerisindeki komutlar şart ifadesi yanlış oluncaya kadar tekrarlanır.



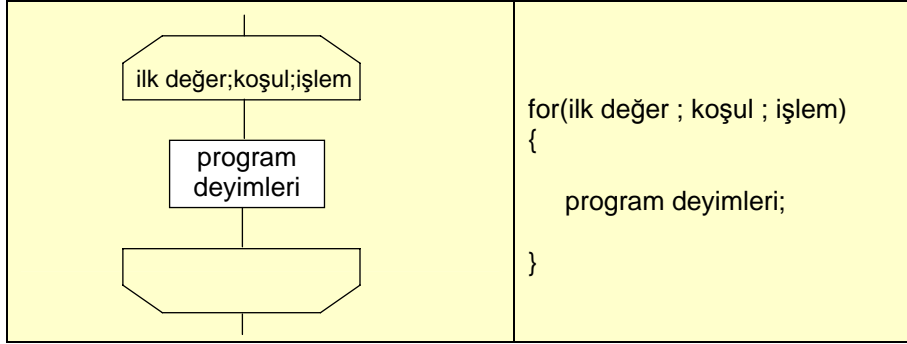
Tablo 2.5. While döngüsü

Kontrolün sonda yapıldığı döngüler **do while** döngüleridir. Bu döngü tipinde önce program deyimleri işletilir, şarta bağlı sınama işlemi ise daha sonra yapılır.



Tablo 2.6. Do while döngüsü

Diğer döngülerde olduğu gibi **for** döngüsünde de döngüye giriş için gerekli **ilk değer**, döngü değerlerinin sorgulandığı **şart ifadesi** ve **işlem** yer almaktadır. Aşağıda for döngüsünün genel yapısı görülmektedir.



Tablo 2.7. For döngüsü

Mikrodenetleyici programlamada yapılan uygulamaların sürekli olması için sonsuz döngü kullanılır. (**while(1)**, **while(true)** veya **for(;;)**)

DONANIM BİLGİSİ – Dahili Fonksiyonlar

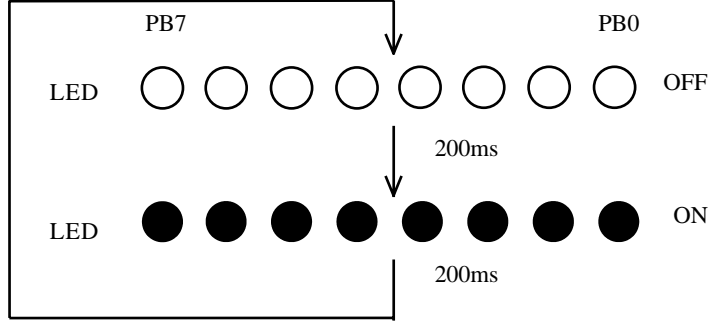
Program içinde gecikme işlemleri için C derleyicisi tarafından sağlanan fonksiyon kullanılır. Bu fonksiyonları etkin yapmak için mikrodenetleyici osilatör frekansı tanımlanmalıdır.

```
#use Delay(Clock=1000000)
```

Fonksiyon tipi	Açıklamalar
delay_cycles(değer)	Mikrodenetleyici komut saykılına bağlı olarak gecikme sağlar. Komut saykılı : 4MHz osilatör frekansı için 1µsn. değer : 1- 255 arası tam sayı değişkenidir. delay_cycles(50);
delay_ms(değer)	Milisaniye birimine göre gecikme sağlar. değer : değişken olarak 0- 255, sabit olarak 0- 65535 aralığına göre seçilebilir. delay_ms(200);
delay_us(değer)	Mikrosaniye birimine göre gecikme sağlar. değer : değişken olarak 0- 255, sabit olarak 0- 65535 aralığına göre seçilebilir. delay_us(1000);

Örnek 2.9.

Bu programda B portuna bağlı olan ledler belli bir zaman aralığı ile yanıp sönecektir. Bu nedenle delay (gecikme) fonksiyonu kullanılacaktır.



```

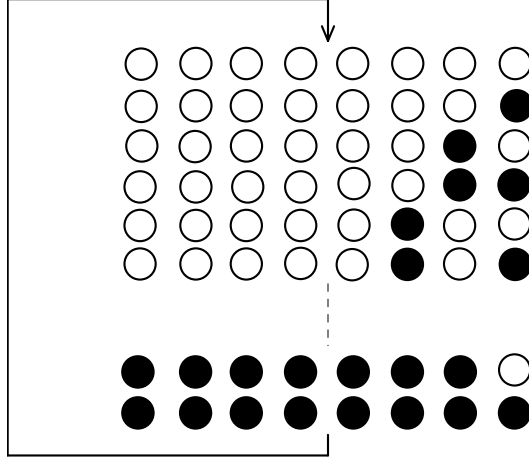
/* Ornek 2.9. ---- LED - Display devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOVDT,PUT,NOPROTECT
#use Delay(Clock=4000000) // osilatör frekansı 4MHz
#byte port_b=6 // b portunun tanımlanması
void main()
{
    set_tris_b(0x00); // a portunun giriş olarak atanması
    while(1) // sonsuz döngü
    {
        port_b=0x00; // port_b ← 00000000
        delay_ms(200); //gecikme 200ms
        port_b=0xff; // port_b ← 11111111
        delay_ms(200); //gecikme 200ms
    }
}

```

Örnek 2.10.

0.1saniye zaman aralığı ile ışık veren ledlerin 0 dan 255 e kadar artarak iki tabanlı sayı değerini veren programı yazınız.



```

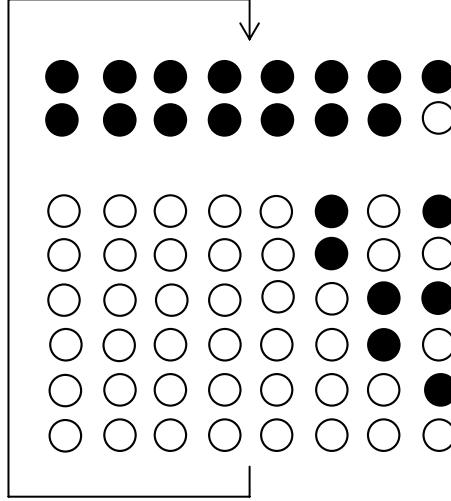
/* Ornek 2.10. ---- LED - Display devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOVDT,PUT,NOPROTECT
#use Delay(Clock=4000000) // osilatör frekansı 4MHz
#byte port_b=6 // b portunun tanımlanması
void main()
{
    int say=0;
    set_tris_b(0x00); // a portunun giriş olarak atanması
    while(1) // sonsuz döngü
    {
        while(say<=255)
        {
            output_b(say) ; // say değişkenini port b ye ata
            delay_ms(200); // gecikme 200ms
            say++; // say değişkenine 1 ekle
        }
    }
}

```

Örnek 2.11.

0.5 saniye zaman aralığı ile ışık veren ledlerin 255 den 0 a kadar eksilerek iki tabanlı sayı değerini veren programı yazınız.



```

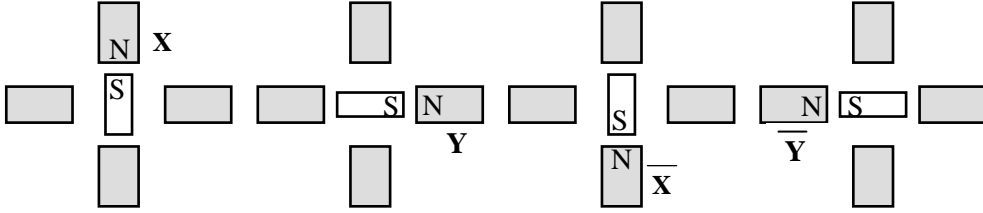
/* Ornek 2.11. ---- LED - Display devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOVDT,PUT,NOPROTECT
#use Delay(Clock=4000000) // osilatör frekansı 4MHz
#byte port_b=6 // b portunun tanımlanması
void main()
{
    int say;
    set_tris_b(0x00); // b portunun çıkış olarak atanması
    while(1) // sonsuz döngü
    {
        say=0x10; // en büyük degerli bit
        do
        {
            say--; // say degiskenini 1 eksilt
            output_b(say); //degeri cikisa aktar
            delay_ms(500); //gecikme 500ms
        }while(position>0);
    }
}

```

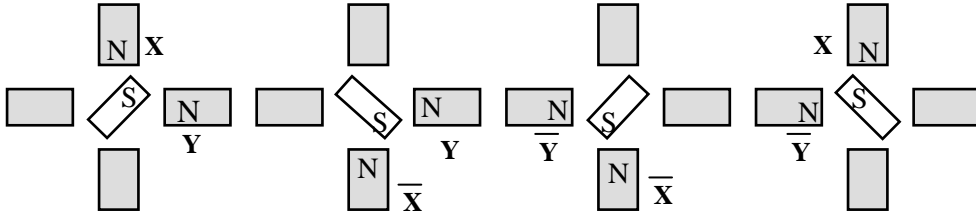
DONANIM BİLGİSİ – Adım motoru kontrolü

• Unipolar adım motoru 1-fazlı sürme metodu



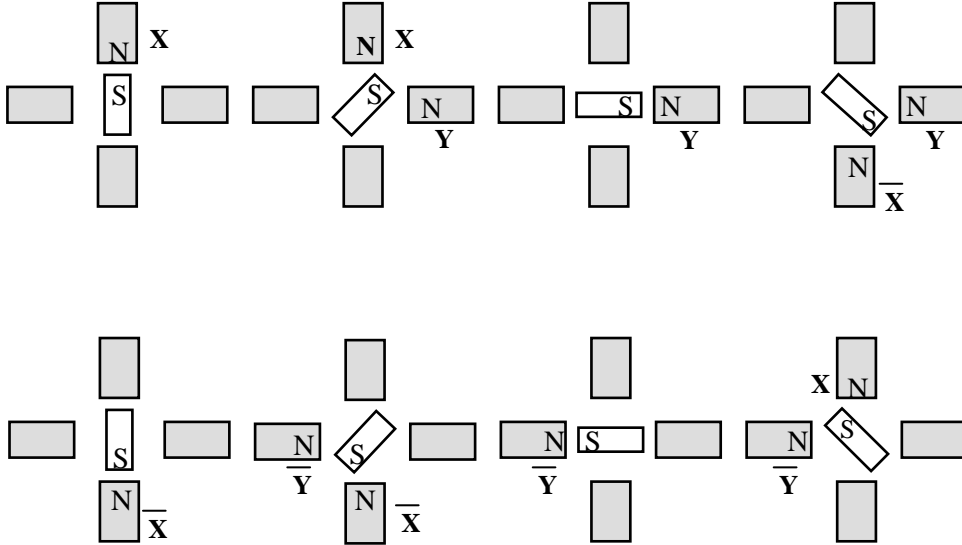
	X	Y	\bar{X}	\bar{Y}		X	Y	\bar{X}	\bar{Y}
saat yönü	1	0	0	0	saat yönü tersi	0	0	0	1
	0	1	0	0		0	0	1	0
	0	0	1	0		0	1	0	0
	0	0	0	1		1	0	0	0

• Unipolar adım motoru 2-fazlı sürme metodu



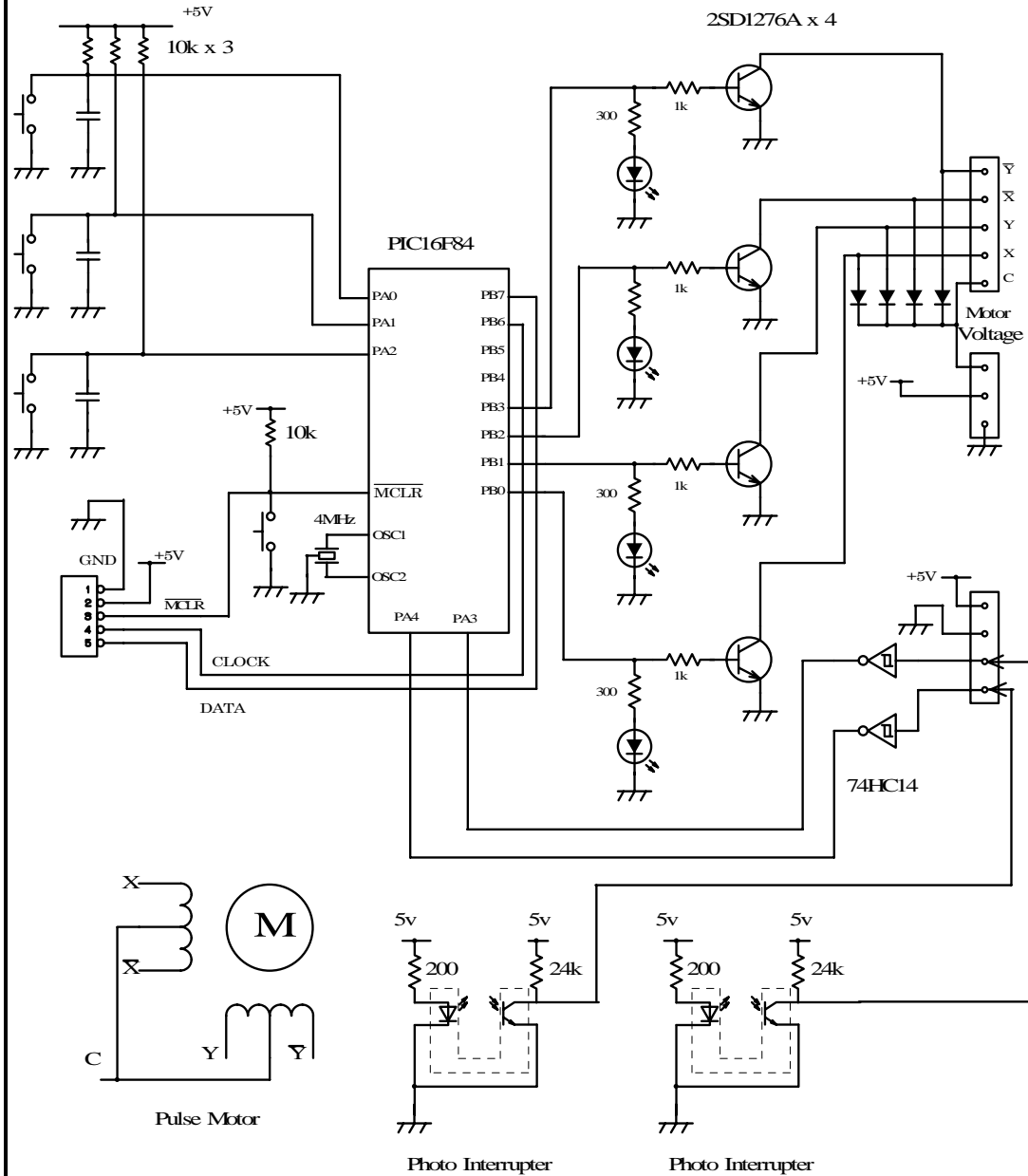
	X	Y	\bar{X}	\bar{Y}		X	Y	\bar{X}	\bar{Y}
saat yönü	1	1	0	0	saat yönü tersi	1	0	0	1
	0	1	1	0		0	0	1	1
	0	0	1	1		0	1	1	0
	1	0	0	1		1	1	0	0

• Unipolar adım motoru 1-2 fazlı sürme metodu



	X	Y	\bar{X}	\bar{Y}		X	Y	\bar{X}	\bar{Y}	
saat yönü	1	0	0	0	saat yönü tersi	1	0	0	1	
	1	1	0	0		0	0	0	1	
	0	1	0	0		0	0	1	1	
	0	1	1	0		0	0	1	0	
	0	0	1	0		0	1	1	0	
	0	0	1	1		0	1	0	0	
	0	0	0	1		1	1	1	0	0
	1	0	0	1		1	1	0	0	0

DONANIM BİLGİSİ – Adım motoru kontrol devresi



Örnek 2.12.

Adım açısı 1.8° olan adım motoru PA0 butonuna basıldığında 1 fazlı sürme metoduna göre saat yönünde dönecektir.

```

/* Ornek 2.12. ---- Adım motoru kontrol devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOVDT,PUT,NOPROTECT
#use Delay(Clock=4000000) // osilatör frekansı 4MHz
#byte port_a=5 // a portunun tanımlanması
#byte port_b=6 // b portunun tanımlanması
void main()
{
    int buton;
    set_tris_a(0x0f); // a portunun giriş olarak atanması
    set_tris_b(0x00); // b portunun çıkış olarak atanması
    while(1) // sonsuz döngü
    {
        buton=port_a&0x01; // a portunun okunması
        while(buton==0)
        {
            port_b=0x01;delay_ms(10);
            port_b=0x02;delay_ms(10);
            port_b=0x04;delay_ms(10);
            port_b=0x08;delay_ms(10);
        }
    }
}

```

Örnek 2.13.

Adım açısı 1.8° olan adım motoru PA0 butonuna basıldığında 2 fazlı sürme metoduna göre saat yönünde 360° dönecektir.

```

/* Ornek 2.13. ---- Adım motoru kontrol devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOWDT,PUT,NOPROTECT
#use Delay(Clock=4000000) // osilatör frekansı 4MHz
#byte port_a=5 // a portunun tanımlanması
#byte port_b=6 // b portunun tanımlanması
void main()
{
    int buton,aci;
    set_tris_a(0x0f); // a portunun giriş olarak atanması
    set_tris_b(0x00); // b portunun çıkış olarak atanması
    while(1) // sonsuz döngü
    {
        buton=port_a&0x01; // a portunun okunması
        while(buton==0) // butona basıldı mı?
        {
            for(aci=1;aci<200;aci++)
            {
                port_b=0b00001100;delay_ms(10);
                port_b=0b00000110;delay_ms(10);
                port_b=0b00000011;delay_ms(10);
                port_b=0b00001001;delay_ms(10);
            }
            if(aci==200) // 200 * 1.8 = 360
                break;
        }
    }
}

```

2.4. Fonksiyon yapıları

C dili fonksiyonlardan oluşmuş bir dildir. C dilinin standart kütüphanesinde tanımlı olan fonksiyonların dışında (printf, scanf, main) kullanıcının da kendisine özel fonksiyon oluşturma, kullanma ve hatta oluşturduğu fonksiyonların kütüphanelerini yaratma olanağı vardır. Fonksiyonların kullanımı ile programlar modüler bir yapıya kavuşur. Böylece program yazımı, kontrolü ve hata ayıklama işlemi daha basit hale gelir.

Aşağıda örnekte main fonksiyonu tarafından çağırılan fonksiyonların temel kullanımları gösterilmiştir.

```
main()
{
    f1();
}
f1()
{
    return 1;
}
```

Yukarıdaki örnekte main fonksiyondan f1 fonksiyonu çağırılmıştır. f1, main fonksiyonundan sonra yazıldığı için program hata verecektir. Bunun nedeni f1 fonksiyonunun tanımlanmasından önce kullanılmasıdır. Bu hatayı gidermek için programımızı aşağıdaki gibi düzenlemek ya da fonksiyon prototipi kullanmak gereklidir.

```
f1()
{
    return 1;
}
main()
{
    f1();
}
```

Bir fonksiyon çağırıldığında doğal olarak bir işi yerine getirmesi beklenir. Fonksiyonun döndürdüğü (ürettiği) değer onu çağırılan fonksiyona aktarılarak bu değer işlenmeye devam edebilir veya fonksiyon bir değer üretir, ancak bu değer ana programa aktarılamaz. Bir fonksiyon çalıştıktan sonra kontrol yeniden onu çağırılan programa geçer. Ana programın bir alt satırından itibaren diğer deyimler çalışmaya devam eder.

Fonksiyonları kullanılabilmesi için belirli bir biçimde tanımlanması gerekir. Her şeyden önce bir fonksiyonun mutlaka bir adı olmalıdır.

```
fonksiyon tipi fonksiyon adı();
```

Bu tanıma göre fonksiyondan elde edilebilecek değerin türü belirlenebilir. Örneğin test isimli bir fonksiyon tamsayı sonuç döndürecek ise `int test()` şeklinde tanımlama yapılabilir.

Fonksiyon prototiplerinde ise geri dönen değerlerin yanı sıra fonksiyonda kullanılacak olan parametre listesi de tanımlanır. Parametrelerin, fonksiyon bildirimindeki değerlerle birebir eşleşmesi gerekir. Fonksiyon prototipinin genel kullanımı aşağıdaki gibidir.

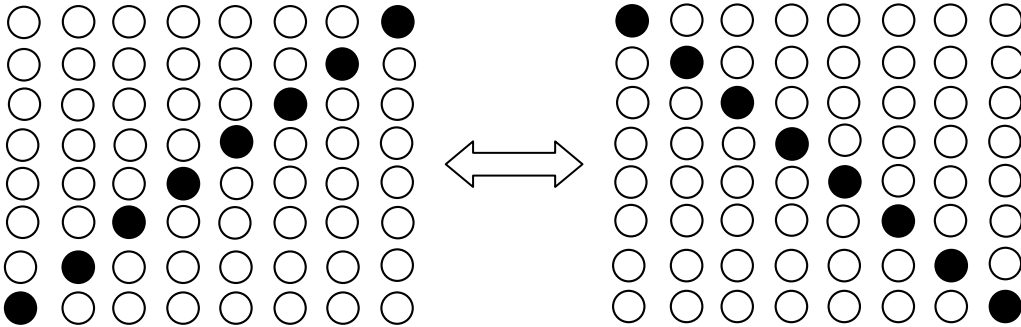
```
fonksiyon tipi fonksiyon adı(parametre listesi);
```

Aşağıdaki örnekte fonksiyon prototipinin kullanımı gösterilmiştir.

```
int us(int sayi, int ussu);
main()
{
    printf("islem sonucu:%d",us(5,2));
}
int us(int sayi,int ussu)
{
    int sonuc=1;
    int i;
    for(i=0;i<ussu;i++)
        sonuc=sayi*sonuc;
    return(sonuc);
}
```

Örnek 2.14.

1 sn. zaman aralığı ile port B ye bağlı ledlerin sağa ve sola kaydırma işlemini yapan programı yazınız.



```

/* Ornek 2.14. ---- LED kontrol devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOWDT,PUT,NOPROTECT
#use Delay(Clock=4000000) // osilatör frekansı 4MHz
#byte port_a=5 // a portunun tanımlanması
#byte port_b=6 // b portunun tanımlanması
io_set()
{
    set_tris_a(0x0f); // a portunun giriş olarak atanması
    set_tris_b(0x00); // b portunun çıkış olarak atanması
}
sola_git()
{
    int n;
    for(n=0;n<8;n++)
    {
        port_b=0x01<<n; // port_b ( <-- 00000001)
        delay_ms(100); // gecikme
    }
}
saga_git()
{
    int n;
    for(n=0;n<8;n++)
    {
        port_b=0x80>>n; // port_b ( <-- 10000000)
        delay_ms(100); // gecikme
    }
}
void main()
{
    int buton;
    io_set();
    while(1)
    {
        sola_git();
        saga_git();
    }
}

```

Örnek 2.15.

A portuna bağlı butonlara basma süresine göre port B ye bağlı ledlerin sağa ve sola kaydırma işlemini yapan programı yazınız.

```

/* Ornek 2.15. ---- LED kontrol devresi kullanılacaktır */

#include<16f84a.h>
#fuses XT,NOWDT,PUT,NOPROTECT
#use Delay(Clock=4000000) // osilatör frekansi 4MHz
#byte port_a=5 // a portunun tanimlanmasi
#byte port_b=6 // b portunun tanimlanmasi
void sola_git(int hiz);
void saga_git(int hiz);
io_set()
{
    set_tris_a(0x0f); // a portunun giris olarak atanmasi
    set_tris_b(0x00); // b portunun çikis olarak atanmasi
}
void main()
{
    int hiz,buton;
    io_set(); // giris - çikis fonksiyonu
    while(1)
    {
        buton =(~port_a) & 0x0f; // buton <---- port_a
        switch(buton) // butona basma bilgisi
        {
            case 1:hiz=12;break; // gecikme 12ms
            case 2:hiz=25;break; // gecikme 25ms
            case 4:hiz=50;break; // gecikme 50ms
            case 8:hiz=100;break; // gecikme 100ms
            default:hiz=200;break; // gecikme 200ms
        }
        sola_git(hiz);
        saga_git(hiz);
    }
}
sola_git(int hiz)
{
    int n;
    for(n=0;n<8;n++)

```

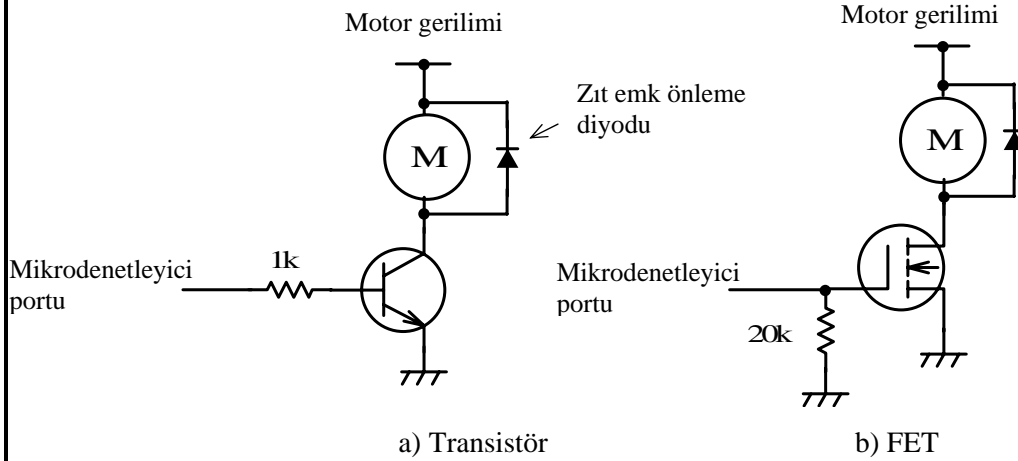
```

{
    port_b=0x01<<n;        // port_b ( <-- 00000001)
    delay_ms(hiz);        // gecikme
}
}
saga_git(int hiz)
{
    int n;
    for(n=0;n<8;n++)
    {
        port_b=0x80>>n;    // port_b ( <-- 00000001)
        delay_ms(hiz);    // gecikme
    }
}

```

DONANIM BİLGİSİ – D.A. motoru kontrol yöntemleri

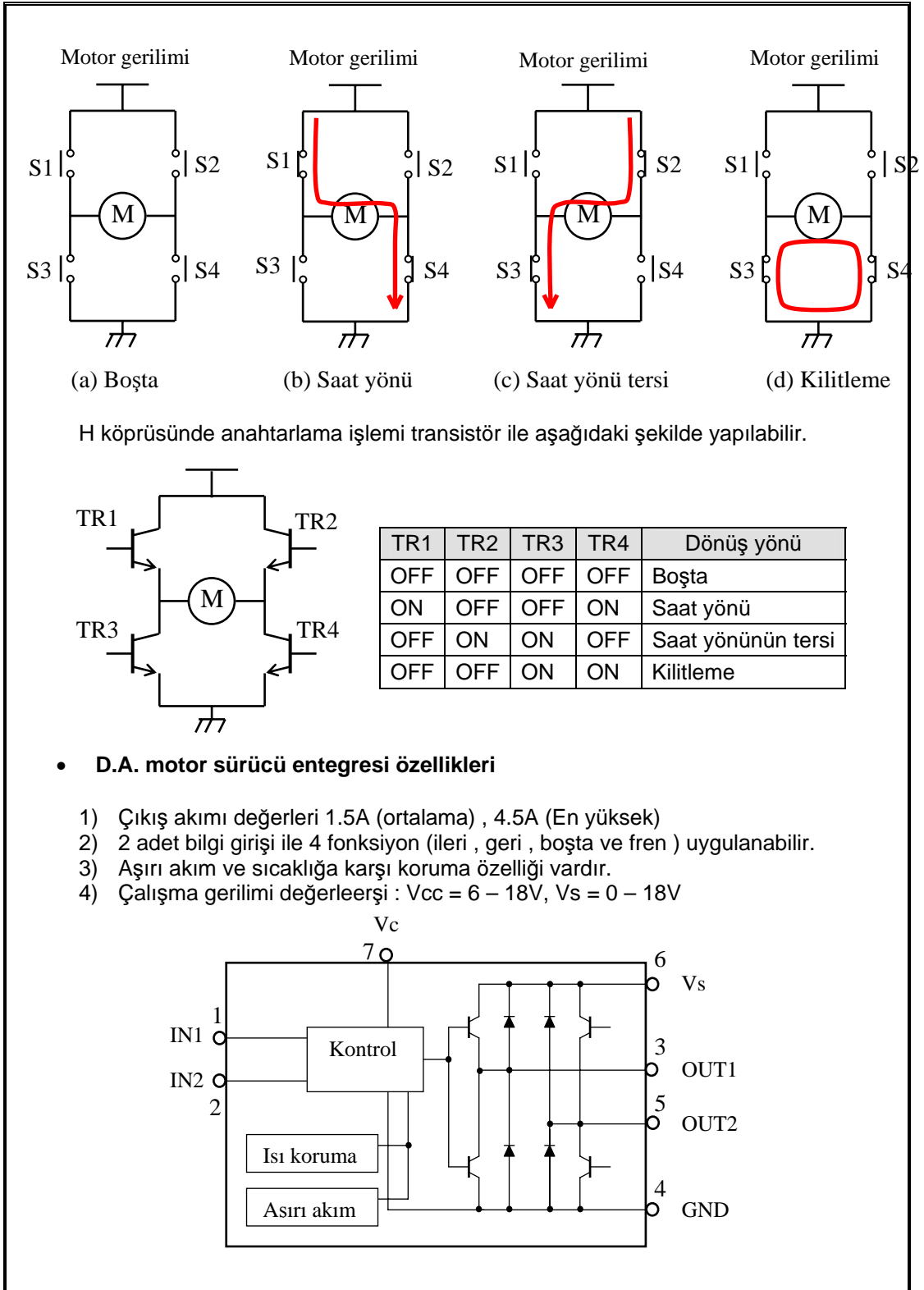
- **Transistör ve FET ile D.A. motor kontrolü**



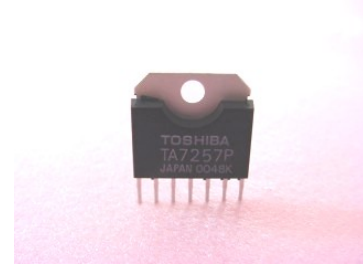
Mikrodenetleyici çıkış portundan mantık "1" sinyali gelirse (5V) , motor dönecektir. Eğer sinyal mantık "0" olur ise motor durur. Bu metod ile motor bir yönde döner.

- **Sürücü entegre ile motor kontrolü**

Sürücü entegresinde H köprü yöntemi kullanılır. H köprüsünü genel olarak aşağıdaki anahtarlama devresi ile açıklayabiliriz.



PIN No.	SEMBOL	FONKSİYON AÇIKLAMASI
1	IN1	Giriş
2	IN2	Giriş
3	OUT1	Çıkış
4	GND	Toprak
5	OUT2	Çıkış
6	Vs	Motor gerilim kaynağı
7	Vcc	Sinyal gerilimi



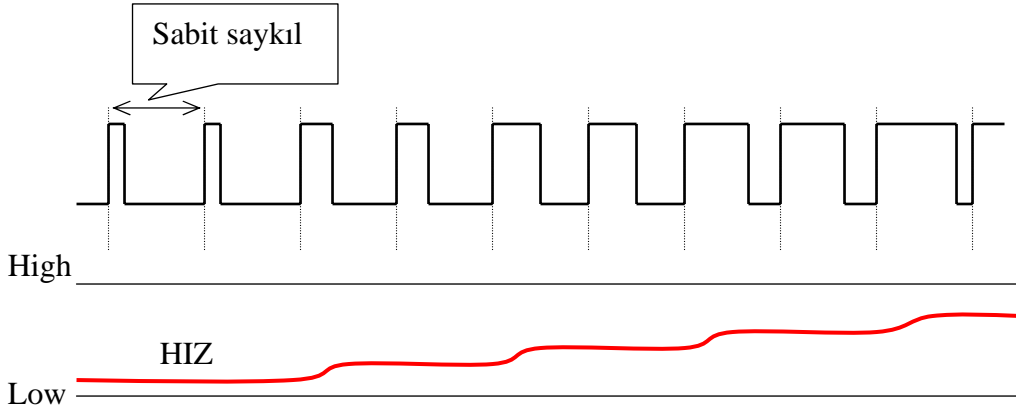
IN1	IN2	OUT1	OUT2	DURUM
1	1	L	L	Fren
0	1	L	H	CW/CCW *
1	0	H	L	CCW/CW *
0	0			Durma

* CW : Saat yönü
CCW : Saat yönü tersi

- **D.A. motoru hız kontrolü**

D.A. motoru hız kontrolü için 2 yöntem vardır. Bunlar gerilim kontrolü ve sinyal kontrolüdür. Mikrodenetleyici uygulamalarında sinyal kontrolü (pulse width modulation - PWM) yöntemi kullanılır.

Sinyal kontrolü, motora uygulanan sinyal darbesi genişliği ile ilgilidir. Şekilden de görüldüğü gibi darbe genişliği büyük olursa motor hızı artar. Motora gönderilen bu sinyal aralığı sayısal olarak belirlenebilir. Bu işlem program veya mikrodenetleyicilerin PWM fonksiyonları ile yapılabilir. Bu metot motor hızı ve lamba ışık kontrolü için etkili bir yöntemdir.



Aşağıdaki şekilde motor kontrolü için FET (alan etkili transistör) kullanılmıştır. Bunun nedeni FET in anahtarlama özelliğinin ve PWM karakteristiğinin iyi olmasıdır.

Örnek 2.16.

Aşağıdaki şartlara uygun programı yazınız.

- PA0 butonuna basıldığında motor dönecek
- PA1 butonuna basıldığında motor duracak

```

/* Ornek 2.16. ---- D.A. motor devresi kullanılacaktır */

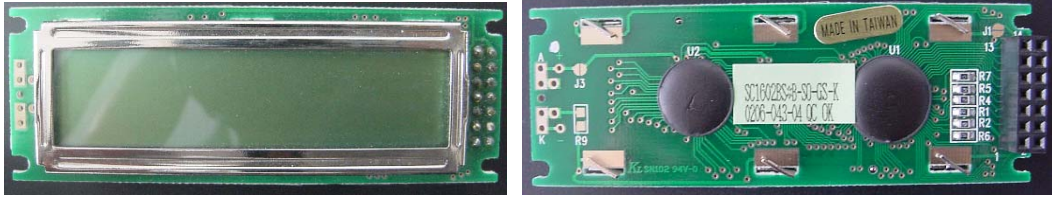
#include<16f84a.h>
#fuses HS,NOWDT,PUT,NOPROTECT
#use Delay(Clock=4000000) // osilatör frekansı 4MHz
#byte port_a=5 // a portunun tanımlanması
#byte port_b=6 // b portunun tanımlanması
io_set()
{
    set_tris_a(0x0f); // a portunun giriş olarak atanması
    set_tris_b(0x00); // b portunun çıkış olarak atanması
}
calis()
{
    port_b=0x02; // PB1 <-- 1 (D.A. motor çalışır)
}
dur()
{
    port_b=0x00; // PB <-- 0 (D.A. motoru durur)
}
void main()
{
    int buton;
    io_set();
    while(1)
    {
        buton=port_a & 0x01; // PA0 bilgisi
        while(buton==0) // PA0 e basıldı mı?
        {
            calis();
            if(!input(PIN_A1)) // PA1 e basıldı mı?
                dur();
        }
    }
}

```

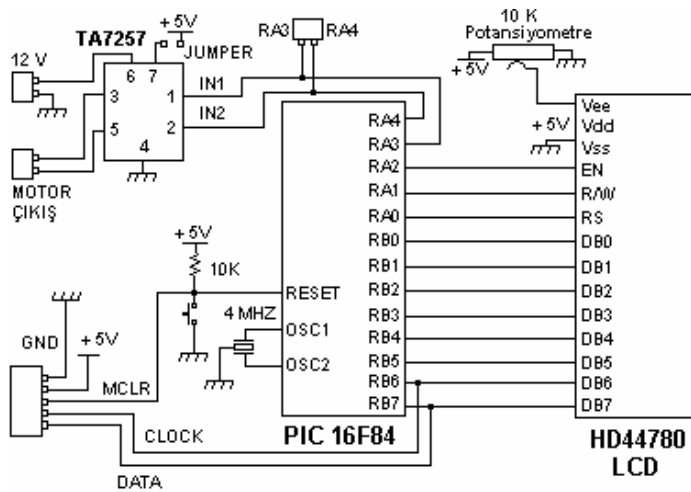
DONANIM BİLGİSİ – LCD kontrol yöntemleri

- **LCD yapısı**

LCD (Liquid Crystal Display)'ler bilginin uygulanış yöntemine göre paralel ve seri girişli olmak üzere iki kısımda incelenebilmektedir. LCD'nin kullanım amacı elektronik sistemlerde bilgilendirme için çeşitli karakterleri ve karakter topluluklarını göstermektir. LCD içerisindeki hazır karakterler kullanılarak veya özel karakterler hazırlanarak üzerinde anlamlı görüntüler oluşturulabilir.



LCD'lerde DDRAM ve CGRAM olmak üzere iki adet RAM hafıza hücresi yer alır. CGRAM karakter üretici RAM bellektir ve 64 Baytlık bir hafızadır. Özel karakterler üretmek için kullanılır. DDRAM ise display veri RAM belleğidir ve 80 Baytlık bir hafızadır. LCD'nin satır ve sütunlarına karakter yazmak için kullanılır. DDRAM içeriği LCD ekranında görülür, fakat görünen veri ancak LCD'nin sütun sayısı ile sınırlıdır. Örneğin 2x16'lık bir LCD'de ancak 16 karakter ekranda görülebilir. Diğer kayıtlı veriler DDRAM'da saklanmaya devam eder. Şekil 1.11'de DDRAM adreslerinin değişimi görülmektedir. LCD'nin içinde bulunan diğer hafıza ise CGROM'dur. Karakter üretici ROM bellek olarak tanımlanır ve 192 Baytlık bir hafızadır. CGROM içerisinde ASCII karakter tablosunda yer alan standart karakterler kayıtlıdır. Toplam karakter sayısı 192'dir. Bunlardan 64 tanesi genel olarak kullanılan ascii karakterleri, 64 tanesi Japon karakterleri ve 32 tanesi de özel yunan alfabesi karakterleridir.



No	Fonksiyonu
1	Vdd (5V gerilim ucu)
2	Vss (Şase bağlantı ucu)
3	Vee (Kontrast gerilimi ucu)
4	RS (Komut/Veri seçim ucu)
5	R/W (Yazma-Okuma ucu)
6	E (Uygulama izin ucu)
7	D0 (0. veri (LSB) ucu)
8	D1 (1. veri ucu)
9	D2 (2. veri ucu)
10	D3 (3. veri ucu)
11	D4 (4. veri ucu)
12	D5 (5. veri ucu)
13	D6 (6. veri ucu)
14	D7 (7. veri (MSB) ucu)

Açıklamalar:

a. Vdd ve Vss: LCD Vdd ucuna bağlanan 5V gerilim ile çalışır. Bağlandığı kontrol elemanı ile şase bağlantısı Vss ile mutlaka yapılmalıdır.

b. Vee: Vee ucu ile 5V arasına 10K değerinde bir trimpot bağlanarak LCD ekranın kontrast seviyesi değiştirilebilir.

c. RS: RS ucu lojik-0 ise komut gönderiliyor, lojik-1 ise veri gönderiliyor anlamındadır. Kontrol devresinden RS ucuna önce lojik-0 gönderilir. Sonra işlenecek olan komut veri uçlarına(D0..D7) yazılır. RS ucu lojik-1 yapılarak komutun işlevi ile ilgili veri diğer bir deyişle ekranda gösterilecek olan veri aynı uçlara gönderilir.

d. R/W: 0 ise LCD hafıza hücresine veri yazılır. 1 ise hafıza hücresinden veri okunur.

e. E: Gönderilen komutun uygulanması için veya verinin işlenmesi için kontrol devresinden bu uca lojik-1'den 0'a düşen (düşen kenar) bir kare dalga sinyal gönderilir.

f: D0..D7: Komut girişi ve veri giriş-çıkışı için kullanılırlar. 8 bitlik modda tümü kullanılırken 4 bitlik LCD çalışmasında D7..D4 uçları kullanılır.

- **LCD programlama**

CCS C derleyici yaması bilgisayara yüklenip MPLAB programına bağlandığında PICC dili ile program yazılıp derlenebilir. Tabii istenirse başka programlar ve yamaları kullanılabilir. Bilgisayardaki PICC klasörü incelendiğinde iki alt klasör göze çarpar. Bunlar "Devices" ve "Drivers" klasörleridir.

"Drivers" klasörü içerisinde çeşitli elektronik yapıların sürücü dosyaları yer almaktadır. Burada yer alan dosyaların uzantıları .C ile biter. Bu klasörde LCD'ye özel LCD.C dosyası vardır ve bu dosya içerisinde LCD kullanımı için fonksiyonlar yer almaktadır. Eğer LCD kullanılmak isteniyorsa LCD.C sürücü dosyası programın başında tanımlanmalıdır. Bu amaçla "#include <LCD.C>" komutu kullanılır.

LCD.C sürücüsü kullanılarak aşağıdaki hazır fonksiyonlar PIC-C programı içerisinde kullanılabilir. Sürücü programın başında tanımlanmazsa program derlemesi sırasında hata meydana gelecektir. Hazır fonksiyonlar ve görevleri aşağıda verilmiştir.

1. lcd_init()

LCD'yi hazırlamak için mutlaka öncelikle kullanılmalıdır.

2. lcd_putc(c)

LCD ekranında istenilen koordinata "c" ile ifade edilen karakteri veya karakter dizisini yazar. Bu fonksiyon ile yardımcı anahtarlar kullanılabilir.

\f : Ekranı temizler ve kursörü 1. satırın 1.sütununa alır.

\n : Kursörü 2. satırın başına getirir.

\b : Kursörü 1 adım geriye hareket ettirir.

3. lcd_gotoxy(x,y)

LCD ekranında yazma koordinatını ayarlar. X sütun numarasını, Y ise satır numarasını temsil eder. LCD ekranında sol üst köşenin koordinatı (1,1)'dir.

4. lcd_getc(x,y)

(x,y) ile ifade edilen koordinattaki karakteri fonksiyon üzerinden döndürür. Diğer bir ifade ile istenen koordinattaki karakteri okur.

Örnek 2.17.

LCD ekranında 80H adresine L harfini yazan programı yapınız.

```
/* Ornek 2.17. ---- LCD devresi kullanılacaktır */  
  
#include <16F84a.h>  
#use delay(clock=4000000)           // 4Mhz'lik kristal  
#fuses HS,NOWDT,NOPROTECT  
#byte port_a=5                       // port_a nin adresi  
#byte port_b=6                       // port_b nin adresi  
#define RS PIN_A0                    // RS  
#define RW PIN_A1                    // RW  
#define E PIN_A2                     // E  
#define PB0 PIN_B0                   // veri uçları  
#define PB1 PIN_B1  
#define PB2 PIN_B2  
#define PB3 PIN_B3  
#define PB4 PIN_B4
```

```

#define PB5 PIN_B5
#define PB6 PIN_B6
#define PB7 PIN_B7

io_set()
{
    set_tris_a(0x00);           // port_a = çıkis
    set_tris_b(0x00);           // port_a = çıkis
}
komutyaz()
{
    Output_Low(RS);             // RS=0 Komut geliyor
    Output_Low(RW);             // R/W=0 Yazma modu
}
veriyaz()
{
    Output_High(RS);            // RS=1 veri geliyor
    Output_Low(RW);            // R/W=0 Yazma modu
}
uyg()
{
    Output_High(E);             // Uygulamadan önce portların kurulması
                                // için 1 ms ile beklenir
    delay_ms(1);                // Kullanılmazsa LCD çalışması kitlenmektedir
    Output_Low(E);
    delay_ms(1);
}
clrdsply()
{
    komutyaz();
    port_b=0x01;                //01H ekran temizleme komutu gönderiliyor
    uyg();
}
dsplay_on()
{
    komutyaz();
    port_b=0x0C;                //0CH Display açma komutu gönderiliyor
    uyg();
}
lcdreset()
{

```

```

    komutyaz();
    port_b=0x30;
    uyg();
    port_b=0x30;
    uyg();
    port_b=0x30; //8 bit modda resetleme için 3 defa 30h
    uyg();      // uygulanıyor
}

lcdkur()
{
    komutyaz();
    port_b=0x03; // 03h komutu uygulandı. 8 bit modu 2 satir
    uyg();      // 5*7 fontu kuruldu
}

adreskur()
{
    komutyaz();
    port_b=0x80; // 80H PortB'den gönderiliyor
    uyg();      // DDRAM adresi 00h olarak kuruluyor
    port_b=0x00; // 1.satir 1.sütun
    uyg();
}

verigonder()
{
    veriyaz(); // RS'nin 1'de kalması sağlanıyor
    port_b=0x41; // A harfi
    uyg();
}

giriskur()
{
    komutyaz();
    port_b=0x06; // 06H PortB'den gönderiliyor - Giriş modu
    uyg();
}

void main()
{
    io_set();
    lcdreset();
}

```

```

lcdkur();
clrdsply();
dsplay_on();
giriskur();
adreskur();
verigonder();
}

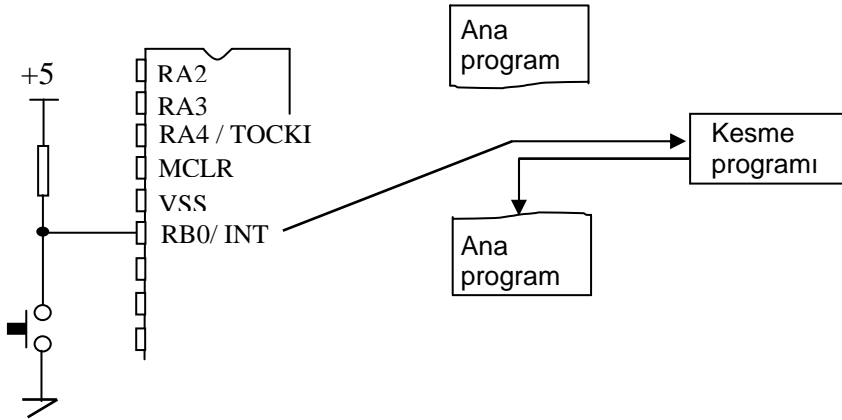
```

2.5 Donanım fonksiyonları

Mikrodenetleyici ile C programlamada derleyicinin sağladığı çeşitli fonksiyonlarla kesme (interrupt), darbe genişliği modülasyonu (PWM), AD – DA dönüşüm işlemleri daha kolay ve pratik olarak uygulanabilir. Bu bölümde donanım fonksiyonları arasında yer alan kesme fonksiyon örnekleri verilecektir.

DONANIM BİLGİSİ – Kesme (interrupt)

Kesme, mikrodenetleyici giriş portlarından veya donanım içindeki sayıcıdan gelen sinyale göre çalışmakta olan programın kesilmesi olayıdır. Program kesildiğinde kesme programı çalışır. Kesme programı çalışmasını bitirdiği andan itibaren program kaldığı yerden çalışmasına devam eder. Kesme işlemi dışarıdan veya mikrodenetleyici içindeki timer 0 zamanlayıcısı ile yapılabilir. Bu bölümde dahili kesme işlemleri üzerinde durulacaktır.



Kesme işlemleri genel olarak İNTCON yazmacı üzerinden ayarlanır. Aşağıdaki şekilde bu yazmacın içeriği görülmektedir.

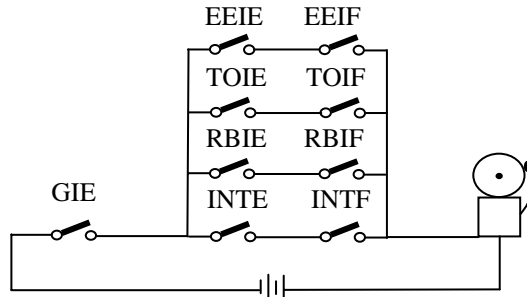
INTCON YAZMACI (h'0b)

GIE	EIEE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
7	6	5	4	3	2	1	0
RBIF	Port b (RB4 ~ RB7) değışiklik bayrađı						
INTF	Harici kesme bayrađı						
TOIF	Tmr0 zaman sayıcı zaman aşımı bayrađı						
RBIE	Port b (RB4 ~ RB7) değışiklik kesmesini aktif yapma bayrađı						
INTE	Harici kesmeyi aktif yapma bayrađı						
TOIE	Tmr0 sayıcı kesmesini aktif yapma bayrađı						
EIEE	EEPROM belleđe yazma işlemini tamamlama kesmesi						
GIE	Tüm kesme işlemlerini iptal etme						

PIC 16f84 kesmeleri aşağıdaki 4 kaynaktan gelebilir.

- 1) Harici kesme RB0 / INT
- 2) Tmr0 sayıcısında oluşan zaman aşımı kesmesi
- 3) Port b 4-5-6-7 bitlerindeki lojik seviye değışikliđi
- 4) EEPROM yazma işleminde tamamlama sırasında meydana gelen kesme

Genel olarak kesme işlemleri aşağıdaki şekilde açıklanabilir.



DONANIM BİLGİSİ - Timer 0 zamanlayıcısı

• **TMR0 sayıcısının özellikleri**

1. 8 bitlik bir sayıcıdır
2. Yazılabilir ve okunabilir
3. Programlanabilen frekans bölme değeri vardır
4. Sayıcı artışı dahili veya harici saat pulsü ile olabilir
5. TMR0 değeri FF → 00 olduğunda ilgili bayrađı "1" yaparak kesme oluşturur.

Tmr0 sayıcısı OPTION REGISTER ile kontrol edilmektedir.

OPTİON YAZMACI (h'0b)

RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
7	6	5	4	3	2	1	0

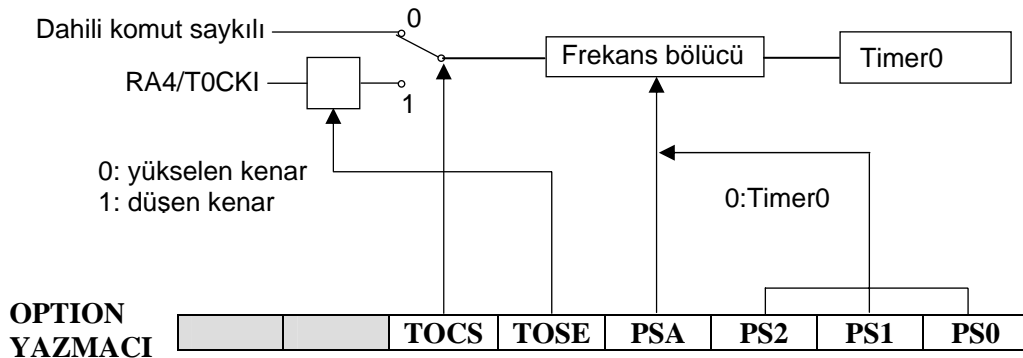
PSA Frekans bölücü seçme biti
 0 → TMR0 1 → WDTMR
 TOSE TMR0 sinyal kaynağı seçme biti
 0 → Düşen kenar 1 → Yükselen kenar
 TOCS TMR0 sinyal kaynağı seçme biti
 0 → Dahili komut saykılı 1 → Harici komut saykılı

Tmr0 frekans bölme değerleri aşağıdaki gibidir.

PS2	PS1	PS0	Tmr0
0	0	0	1/2
0	0	1	1/4
0	1	0	1/8
0	1	1	1/16
1	0	0	1/32
1	0	1	1/64
1	1	0	1/128
1	1	1	1/256

Tmr0 zamanlayıcısı aşağıdaki işlem sırasına göre programlanır

- 1) TOCS bayrağı 0 yapılarak dahili komut saykılı seçilir.(Bu değer osilatör frekansının ¼ ü kadardır.)
- 2) TOSE bayrağının 0 yapılması ile yükselen kenar seçilir.
- 3) PSA bayrağı ile frekans bölücü aktif hale getirilir.
- 4) PS2 , PS1 ve PS0 bayraklarına yukarıdaki tabloda görülen sayılarla frekans bölme işlemi yapılır.



• **TMR0 oranı**

Frekans bölücü bayraklarına (PS2,PS1,PS0) uygulanan sayılar ile Tmr0 sayıcısının kaç saykıl da bir artmasını belirleyen orandır.

Örneğin Tmr0 oranı 1/2 ise 2 komut saykılında
 1/8 ise 8 komut saykılında
 1/256 ise 256 komut saykılında sayıcı 1 artar

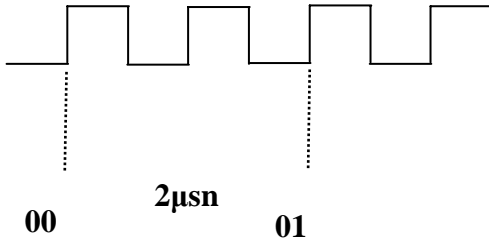
Program belleğinde yerleştirilen komutların çalışması için PIC mikrodenetleyiciye uygulanan frekans 4 e bölünür.4 Mhz değerinde osilatör kullanılan bir PIC mikrodenetleyicide komut saykılı

$$F = F_{osc} / 4 = 4 \text{ Mhz} / 4 = 1 \text{ Mhz} \quad \text{Komut saykılı için geçen süre ise}$$

$$T = 1/F = 1/ 1\text{Mhz} = 1\mu\text{sn dir}$$

Örnek

Frekans bölücü değeri PS2 – PS1 – PS0 = 0 0 0 seçilirse Tmr0 komut frekansı 1/2 olur.Başka bir ifade ile 2 saykıl da bir sayıcı değer artırır.

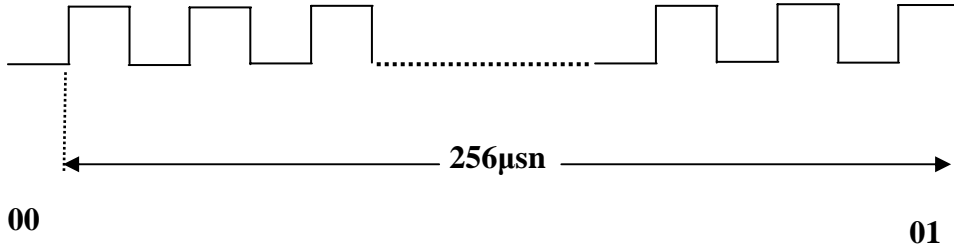


Tmr0 sayma değeri 00 → FF arasında olduğu için geçen zaman

$$2\mu\text{sn} \times 256 = 512 \mu\text{sn dir.}$$

Örnek

Frekans bölücü değeri PS2 – PS1 – PS0 = 1 1 1 seçilirse Tmr0 komut frekansı 1/256 olur.Başka bir ifade ile 256 saykıl da bir sayıcı değer artırır.



Tmr0 sayma değeri 00 → FF arasında olduğu için geçen zaman

$$256\mu\text{sn} \times 256 = 35536 \mu\text{sn dir}$$

Örnek 2.18.

TMRO sayıcısını kullanarak 1 sn aralıkla sayma işlemi yapan programı yazınız.

```

/* Ornek 2.18. ---- LED kontrol devresi kullanılacaktır */

#include <16f84a.h>           // PIC özelliklerinin tanımlanması
#define Delay(Clock=4000000) // osilatör frekansı
#define port_a=5             // port_a nın TRISA daki adresi
#define port_b=6             // port_b nin TRISA daki adresi
#define int_per_sec 61       // 4000000/(4*256*64)=61.035
int sec_count;               // kesme zamanı oluşturulması)
int int_count;               // sayaç değişkeni
/* ++++++ kesme fonksiyonu ++++++ */
// Timer0 sayacı aşağıdaki şekilde kurulur.
//   frekans : 4MHz ise 1 saykıl : lusec
//   prescale : 64
//   kesme saykılı = 1*256*64=16.384ms
//   aralık = 16.384ms
// ++++++
interrupt_set()
{
    setup_counters(RTCC_INTERNAL,RTCC_DIV_64);
} // RTCC_INTERNAL=CLOCK 1/64

/* ++++++ RTCC kesme fonksiyonu ++++++ */
#define INT_RTCC // kesme tmr0
rtcc_isr() // Bu fonksiyon her zaman çağırılır.
{
    if(--int_count==0) // RTCC(timer0) taşma (255-->0)
    {
        sec_count++; // 61/saniye( 0.99942sn = yaklaşık 1sn)
        int_count=int_per_sec;
    }
}

/* ++++++ i/o fonksiyonu ++++++ */
io_set()
{
    set_tris_a(0x1f); // port_a = giriş
    set_tris_b(0x00); // port_b = çıkış
}

```

```
/* ===== ana fonksiyon ===== */
main()
{
    io_set()                // Giriş - çıkış fonksiyonu
    interrupt_set();        // kesme kurulumu
    sec_count=0;            // sec_data 0
    enable_interrupts(INT_RTCC); // kesme
    enable_interrupts(GLOBAL); // kesme başlangıcı
    while(1)
    {
        port_b = sec_count; // saniye ---> port_b
    }
}
```