

GİRİŞ	HATA! YER İŞARETİ TANIMLANMAMIŞ.
MİKROİŞLEMCİ NEDİR?	Hata! Yer işareti tanımlanmamış.
MİKRODENETLEYİCİ NEDİR?	Hata! Yer işareti tanımlanmamış.
Neden Mikroişlemci Değil de Mikrodenetleyici Kullanılıyor?	Hata! Yer işareti tanımlanmamış.
MİKRODENETLEYİCİLER HAKKINDA GENEL BİLGİLER	Hata! Yer işareti tanımlanmamış.
Neden PIC?	Hata! Yer işareti tanımlanmamış.
Neden PIC16F84?	Hata! Yer işareti tanımlanmamış.
PIC PROGRAMLAMAK İÇİN NELERE İHTİYACINIZ VAR?	Hata! Yer işareti tanımlanmamış.
IBM Uyumlu Bilgisayar.....	Hata! Yer işareti tanımlanmamış.
Metin Editörü	Hata! Yer işareti tanımlanmamış.
Assembler Programı.....	Hata! Yer işareti tanımlanmamış.
PIC Programlayıcı Yazılımı	Hata! Yer işareti tanımlanmamış.
Programlanmış PIC'i Deneme Kartı	Hata! Yer işareti tanımlanmamış.
PIC DONANIM ÖZELLİKLERİ	HATA! YER İŞARETİ TANIMLANMAMIŞ.
PIC ÇEŞİTLERİ	Hata! Yer işareti tanımlanmamış.
PIC'LERİN DIŞ GÖRÜNÜŞÜ	Hata! Yer işareti tanımlanmamış.
PIC BELLEK ÇEŞİTLERİ	Hata! Yer işareti tanımlanmamış.
PIC16F84	HATA! YER İŞARETİ TANIMLANMAMIŞ.
PIC 16F84'ÜN PIN GÖRÜNÜŞÜ	Hata! Yer işareti tanımlanmamış.
BESLEME GERİLİMİ	Hata! Yer işareti tanımlanmamış.
CLOCK UÇLARI ve CLOCK OSİLATÖRÜ ÇEŞİTLERİ	Hata! Yer işareti tanımlanmamış.
RESET UÇLARI VE RESET DEVRESİ	Hata! Yer işareti tanımlanmamış.
I/O PORTLARI	Hata! Yer işareti tanımlanmamış.
PIC16F84'ÜN BELLEĞİ	Hata! Yer işareti tanımlanmamış.
Program Belleği.....	Hata! Yer işareti tanımlanmamış.
RAM Bellek	Hata! Yer işareti tanımlanmamış.
W register	Hata! Yer işareti tanımlanmamış.
PIC ASSEMBLY	HATA! YER İŞARETİ TANIMLANMAMIŞ.
ASSEMBLER NEDİR?	Hata! Yer işareti tanımlanmamış.
PIC ASSEMBLY DİLİ NEDİR?	Hata! Yer işareti tanımlanmamış.
PIC ASSEMBLY DİLİ YAZIM KURALLARI	Hata! Yer işareti tanımlanmamış.
Noktalı Virgül (;).....	Hata! Yer işareti tanımlanmamış.
Girintiler ve Program Bölümleri	Hata! Yer işareti tanımlanmamış.
Başlık.....	Hata! Yer işareti tanımlanmamış.
Etiketler	Hata! Yer işareti tanımlanmamış.
Atama deyimi (EQU)	Hata! Yer işareti tanımlanmamış.
Sabitler	Hata! Yer işareti tanımlanmamış.
ORG Deyimi	Hata! Yer işareti tanımlanmamış.
Sonlandırma Bloğu.....	Hata! Yer işareti tanımlanmamış.
Büyük ve Küçük Harflerin Kullanımı	Hata! Yer işareti tanımlanmamış.
PIC ASSEMBLY KOMUTLARININ YAZILIŞ BİÇİMİ	Hata! Yer işareti tanımlanmamış.
Byte-Yönlendirmeli Komutlar	Hata! Yer işareti tanımlanmamış.
Bit-Yönlendirmeli Komutlar	Hata! Yer işareti tanımlanmamış.
Sabit İşleyen Komutlar.....	Hata! Yer işareti tanımlanmamış.
Kontrol Komutları	Hata! Yer işareti tanımlanmamış.

SAYI VE KARAKTERLERİN YAZILIŞ BİÇİMİ.....	Hata! Yer işareti tanımlanmamış.
Heksadesimal sayılar.....	Hata! Yer işareti tanımlanmamış.
Binary Sayılar.....	Hata! Yer işareti tanımlanmamış.
Desimal sayılar.....	Hata! Yer işareti tanımlanmamış.
ASCII Karakterler	Hata! Yer işareti tanımlanmamış.
PIC ASSEMBLY KOMUTLARI	Hata! Yer işareti tanımlanmamış.
PIC PROGRAMLAMA.....	HATA! YER İŞARETİ TANIMLANMAMIŞ.
İLK PROGRAMINIZ.....	Hata! Yer işareti tanımlanmamış.
Bank Değiştirme.....	Hata! Yer işareti tanımlanmamış.
Port'ların Giriş veya Çıkış Olarak Yönlendirilmesi	Hata! Yer işareti tanımlanmamış.
AKIŞ DİYAGRAMI SEMBOLLERİ	Hata! Yer işareti tanımlanmamış.
AKIŞ DİYAGRAMININ ÇİZİLMESİ.....	Hata! Yer işareti tanımlanmamış.
ASSEMBLY PROGRAM KOMUTLARININ YAZILMASI.....	Hata! Yer işareti tanımlanmamış.
Atama (EQU) Komutu Kullanarak Program Yazmak	Hata! Yer işareti tanımlanmamış.
PROGRAMLARIN DERLENMESİ (MPASM)	Hata! Yer işareti tanımlanmamış.
PROGRAMIN PIC'E YAZDIRILMASI.....	Hata! Yer işareti tanımlanmamış.
P16PRO'nun Başlatılması	Hata! Yer işareti tanımlanmamış.
PIC Seçme (F3)	Hata! Yer işareti tanımlanmamış.
Program Dosyasını Açma (F1).....	Hata! Yer işareti tanımlanmamış.
PIC Konfigürasyonunu Ayarlama (F2)	Hata! Yer işareti tanımlanmamış.
Programı PIC'e Yazdırma (F4).....	Hata! Yer işareti tanımlanmamış.
PROGRAMLANMIŞ PIC'İN DENENMESİ	Hata! Yer işareti tanımlanmamış.
BreadBoard Üzerine Kurulan Devre ile Denemek.....	Hata! Yer işareti tanımlanmamış.
PIC Deneme Kartı ile Denemek.....	Hata! Yer işareti tanımlanmamış.
MPASM'NİN ÜRETTİĞİ DİĞER DOSYALAR	Hata! Yer işareti tanımlanmamış.
.LST Dosyası.....	Hata! Yer işareti tanımlanmamış.
.ERR Dosyası	Hata! Yer işareti tanımlanmamış.
INCLUDE DOSYALARI	Hata! Yer işareti tanımlanmamış.
INCLUDE Dosyası Kullanarak Program Yazmak.....	Hata! Yer işareti tanımlanmamış.
KONFIGÜRASYON BİTLERİNİN YAZILMASI.....	Hata! Yer işareti tanımlanmamış.
VERİ TRANSFERİ VE KARAR İŞLEMLERİ.....	HATA! YER İŞARETİ TANIMLANMAMIŞ.
W REGİSTERİN KULLANIMI (MOVLW, MOVWF KOMUTLARI)...	Hata! Yer işareti tanımlanmamış.
Veri Transferi	Hata! Yer işareti tanımlanmamış.
Sonsuz Döngü	Hata! Yer işareti tanımlanmamış.
BİT TEST EDEREK KARAR VERMEK (BTFSC, BTFSS)	Hata! Yer işareti tanımlanmamış.
DÖNGÜ DÜZENLEMEK	HATA! YER İŞARETİ TANIMLANMAMIŞ.
SAYAÇ KULLANARAK DÖNGÜ KULLANMAK (DECFSZ)	Hata! Yer işareti tanımlanmamış.
Basit Bir Zaman Gecikme Döngüsü Yapmak	Hata! Yer işareti tanımlanmamış.
KARŞILAŞTIRMA YAPARAK DÖNGÜ DÜZENLEMEK (SUBLW, SUBWF, INCF, DECF KOMUTLARI)	Hata! Yer işareti tanımlanmamış.
SUBWF Komutu	Hata! Yer işareti tanımlanmamış.
SUBLW Komutu.....	Hata! Yer işareti tanımlanmamış.
STATUS REGISTER.....	Hata! Yer işareti tanımlanmamış.
STATUS REGİSTER BIT'LERİ.....	Hata! Yer işareti tanımlanmamış.

ZAMAN GECİKTİRME VE ALT PROGRAMLAR..... HATA! YER İŞARETİ TANIMLANMAMIŞ.

ZAMAN GECİKTİRME DÖNGÜLERİHata! Yer işareti tanımlanmamış.

Dahili Komut Saykılı **Hata! Yer işareti tanımlanmamış.**

Tek Döngü ile Minimum Zaman Geciktirme..... **Hata! Yer işareti tanımlanmamış.**

Tek Döngü ile Maksimum Zaman Geciktirme **Hata! Yer işareti tanımlanmamış.**

Komut Saykıl Sayısının Bulunması **Hata! Yer işareti tanımlanmamış.**

N sayısının bulunması **Hata! Yer işareti tanımlanmamış.**

Çift Döngülü Zaman Geciktirme..... **Hata! Yer işareti tanımlanmamış.**

N Sayısının Hesaplanması..... **Hata! Yer işareti tanımlanmamış.**

ALT PROGRAMLARHata! Yer işareti tanımlanmamış.

BİT KAYDIRMA VE MANTIKSAL İŞLEM KOMUTLARI HATA! YER İŞARETİ TANIMLANMAMIŞ.

SOLA KAYDIRMA (RLF)Hata! Yer işareti tanımlanmamış.

SAĞA KAYDIRMA.....Hata! Yer işareti tanımlanmamış.

COMF VE SWAPF KOMUTLARI.....Hata! Yer işareti tanımlanmamış.

MANTIKSAL İŞLEM KOMUTLARI.....Hata! Yer işareti tanımlanmamış.

ANDLW Komutu (İstenilen bir ya da birkaç bit'i "0" yapmak) **Hata! Yer işareti tanımlanmamış.**

ANDWF Komutu **Hata! Yer işareti tanımlanmamış.**

IORLW Komutu (İstenilen bit'in değerini "1" yapmak) **Hata! Yer işareti tanımlanmamış.**

IORWF Komutu..... **Hata! Yer işareti tanımlanmamış.**

XORLW Komutu (İstenilen bir bit'i terslemek)..... **Hata! Yer işareti tanımlanmamış.**

XORWFKomutu **Hata! Yer işareti tanımlanmamış.**

Bir Byte'lık İki Veriyi Karşılaştırmak (XORLW, XORWF) **Hata! Yer işareti tanımlanmamış.**

Bir Byte'lık Veriyi "0" ile Karşılaştırmak (IORLW, IORWF)..... **Hata! Yer işareti tanımlanmamış.**

ARİTMETİK İŞLEMLER..... HATA! YER İŞARETİ TANIMLANMAMIŞ.

ARİTMETİK İŞLEM KOMUTLARI.....Hata! Yer işareti tanımlanmamış.

8 - BIT TOPLAMA..... **Hata! Yer işareti tanımlanmamış.**

16- BİT TOPLAMA..... **Hata! Yer işareti tanımlanmamış.**

8 - BIT ÇIKARMA **Hata! Yer işareti tanımlanmamış.**

16- BİTÇIKARMA **Hata! Yer işareti tanımlanmamış.**

ÇEVİRİM TABLOLARI HATA! YER İŞARETİ TANIMLANMAMIŞ.

ÇEVİRİM TABLOSU (LOOKUP TABLE) NEDİR?.....Hata! Yer işareti tanımlanmamış.

PROGRAM COUNTER (SAYICI)Hata! Yer işareti tanımlanmamış.

RETLW Komutu **Hata! Yer işareti tanımlanmamış.**

STEP MOTOR KONTROLÜHata! Yer işareti tanımlanmamış.

KESMELER (INTERRUPTS)..... HATA! YER İŞARETİ TANIMLANMAMIŞ.

KESME (INTERRUPT) NEDİR?Hata! Yer işareti tanımlanmamış.

INTCON REGİSTERİ.....Hata! Yer işareti tanımlanmamış.

KESME KAYNAKLARI.....Hata! Yer işareti tanımlanmamış.

Harici Kesmeler..... **Hata! Yer işareti tanımlanmamış.**

TMRO Sayıcı Kesmesi..... **Hata! Yer işareti tanımlanmamış.**

PORTB Lojik Seviye (RB4-RB7) Değişiklik Kesmesi .. **Hata! Yer işareti tanımlanmamış.**

KESME ALT PROGRAMLARININ DÜZENLENMESİ Hata! Yer işareti tanımlanmamış.

Tüm Kesme İşlemlerini Aktif Yapma Bayrağı (GIE)..... **Hata! Yer işareti tanımlanmamış.**

Kesme Esnasında W ve Status Registeri Saklamak **Hata! Yer işareti tanımlanmamış.**

Kesme Alt Programları Nereye Yazılmalı?	Hata! Yer işareti tanımlanmamış.
Kesme Gecikmesi.....	Hata! Yer işareti tanımlanmamış.
Tek Pals Üreticinin Kullanımı.....	Hata! Yer işareti tanımlanmamış.
DONANIM SAYICILARI.....	100
DONANIM SAYICISI/ZAMANLAYICISI NEDİR?.....	100
TMR0 SAYICI/ZAMANLAYICISI(TIMER/COUNTER).....	100
OPTION REGISTER	100
TMR0 SAYICININ ÖZELLİKLERİ	101
Frekans Bölme Sayısının (Prescaler) Kullanılması.....	102
TMR0 ve WDT Oranı	102
Frekans Bölme Sayısının Atanması	104
TMR0'dan WDT'ye Prescaler Değeri Atamak:.....	104
WDT'den TMR0'a Prescaler Değeri Atamak:.....	104
TMR0 Sayıcısının Kullanılması	104
TMR0 Sayıcı Kesmesine Ait Örnekler	108
TMR0 Sayıcısını İstenilen Bir Sayıdan Başlatmak.....	110
WDT ZAMANLAYICISI (WATCHDOG TIMER)	111
Zaman Aşımı Süresi	112
SLEEP Komutunun Kullanılması	113
D/A VE A/D ÇEVİRME İŞLEMLERİ.....	HATA! YER İŞARETİ TANIMLANMAMIŞ.
DİJİTAL/ANALOG ÇEVİRİCİ	Hata! Yer işareti tanımlanmamış.
Ladder(Merdiven) Direnç Devresi Kullanmak	Hata! Yer işareti tanımlanmamış.
8 Bit D/A konvertör Entegresi Kullanmak	Hata! Yer işareti tanımlanmamış.
PWM (Pulse Width Modulation) Metodu Kullanmak	Hata! Yer işareti tanımlanmamış.
PWM Metodu	Hata! Yer işareti tanımlanmamış.
İş ve Bekleme Süresinin Tespit Etmek	Hata! Yer işareti tanımlanmamış.
ANALOG / DİJİTAL ÇEVİRİCİ.....	Hata! Yer işareti tanımlanmamış.
PIC16F84'ün Giriş Seviyesinin Ölçümü	Hata! Yer işareti tanımlanmamış.
A/D çevrim Metodu Kullanarak Direnç Ölçmek	Hata! Yer işareti tanımlanmamış.

1 GİRİŞ

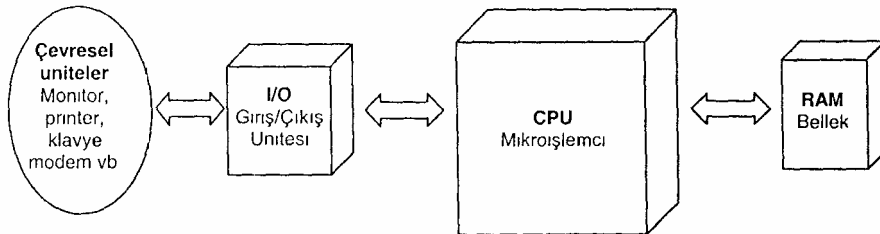
- ❑ MİKROİŞLEMCİ NEDİR?
- ❑ MİKRODENETLEYİCİ NEDİR?
- ❑ MİKRODENETLEYİCİLER HAKKINDA GENEL BİLGİLER
- ❑ PIC PROGRAMLAMA İÇİN NEYE İHTİYACIMIZ VAR

MİKROİŞLEMCİ NEDİR?

Günümüzde kullanılan bilgisayarların özelliklerinden bahsedilirken duyduğunuz 80386, 80486, Pentium-II, Pentium-III birer mikroişlemcidir (Microprocessor). Mikroişlemciler bilgisayar programlarının yapmak istediği tüm işlemleri yerine getirdiği için, çoğu zaman merkezi işlem ünitesi (**CPU**- Central Processing Unit) olarak da adlandırılır. PC adını verdiğimiz kişisel bilgisayarlarda kullanıldığı gibi, bilgisayarla kontrol edilen sanayi tezgahlarında ve ev aytıklarında da kullanılabilir. Bir mikroişlemci işlevini yerine getirebilmesi için aşağıdaki yardımcı elemanlara ihtiyaç duyar. Bunlar:

1. Input (Giriş) ünitesi.
2. Output (Çıkış) ünitesi.
3. Memory (Bellek) ünitesi.

Bu üniteler CPU chip'inin dışında, bilgisayarın ana kartı üzerinde bir yerde farklı chip'lerden veya elektronik elemanlardan oluşur. Aralarındaki iletişimi ise veri yolu (Data bus), adres yolu (Address bus) denilen iletim hatları yapar.



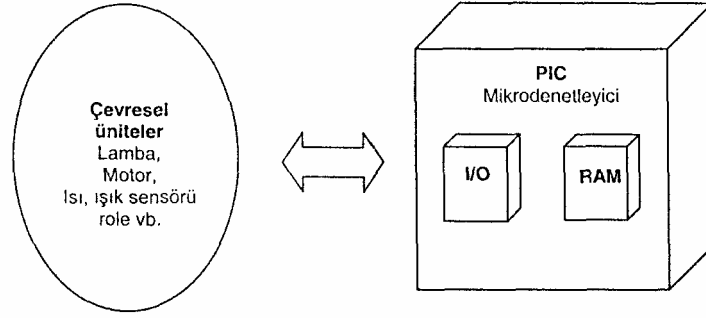
Bir mikroşlemci sisteminin temel bileşenlerinin blok diyagramı

Intel, Cyrix, AMD, Motorola mikroşlemci üreticilerden birkaçıdır, Günümüzde mikroşlemciler genellikle PC adını verdiğimiz kişisel bilgisayarlarda kullanılmaktadır.

MİKRODENETLEYİCİ NEDİR?

Bir bilgisayar içerisinde bulunması gereken temel bileşenlerden RAM, I/O ünitesinin tek bir chip içerisinde üretilmiş biçimine mikrodnetleyici (**Microcontroller**) denir. Bilgisayar teknolojisi gerektiren uygulamalarda kullanılmak üzere tasarlanmış olan mikrodnetleyiciler, mikroşlemcilere göre çok daha basit ve ucuzdur. Günümüz mikrodnetleyicileri otomobillerde, kameralarda, cep telefonlarında, fax-modem cihazlarında, fotokopi, radyo, TV, bazı oyuncaklar gibi sayılamayacak kadar pek çok alanda kullanılmaktadır.

Günümüz mikrodnetleyicileri birçok chip üreticisi tarafından üretilmektedir. Her firma ürettiği chip'e farklı isimler vermektedir. Örneğin Microchip firması ürettiklerine PIC adını verirken, Intel'in ürettiği ve 1980'lerin başında piyasaya sürdüğü 8051, bazen MCS-51 olarak da adlandırılır.



Bir mikrodnetleyici sisteminin temel bileşenlerinin blok diyagramı

Neden Mikroşlemci Değil de Mikrodnetleyici Kullanılıyor?

Mikro işlemci ile kontrol edilecek bir sistemi kurmak için en azından şu üniteler bulunmalıdır; CPU, RAM, I/O ve bu ünitelerin arasındaki veri alış verişini kurmak için DATA BUS (data yolu) gerekmektedir. Elbette bu üniteleri yerleştirmek için baskılı devreyi de unutmamak gerekmektedir. Mikrodnetleyici ile kontrol edilecek sistemde ise yukarıda saydığımız ünitelerin yerine geçecek tek bir chip (Mikrodnetleyici) ve bir de devre kartı kullanmak yeterlidir. Tek chip kullanarak elektronik çözümler üretmenin maliyetinin daha düşük olacağı kesindir. Ayrıca da kullanım ve programlama kolaylığı da ikinci bir avantajdır. İşte yukarıda saydığımız nedenlerden dolayı son zamanlarda bilgisayar kontrolü gerektiren elektronik uygulamalarda mikrodnetleyici kullanmaya eğilimin artmasının haklılığını ortaya koyuyor.

MİKRODENETLEYİCİLER HAKKINDA GENEL BİLGİLER

Neredeyse her mikroşlemci (CPU) üreticisinin ürettiği birkaç mikrodnetleyicisi bulunmaktadır. Bu denetleyicilerin mimarileri arasında çok küçük farklar olmasına rağmen aşağı yukarı aynı işlemleri yapabilmektedirler. Her firma ürettiği chip'e bir isim ve özelliklerini birbirinden ayırmak için de parça numarası vermektedir. Örneğin Microchip ürettiklerine PIC adını, parça numarası olarak da 12C508, 16C84, 16F84, 16C711 gibi kodlamalar verir. Intel ise ürettiği mikrodnetleyicilere MCS-51 ailesi adını vermektedir. Genel olarak bu adla anılan mikrodnetleyici ailesinde farklı özellikleri bulunan ürünleri birbirinden ayırt etmek için parça numarası olarak da 8031AH, 8051AH, 8751AHP, 8052AH, 80C51FA gibi kodlamalar kullanılmaktadır.

Bir uygulamaya başlamadan önce hangi firmanın ürünü kullanılacağına, daha sonra da hangi numaralı denetleyicinin kullanılacağına karar vermek gerekir. Bunun için mikrodnetleyici gerektiren uygulamada hangi özelliklerin olması gerektiği önceden bilinmesi gereklidir. Aşağıda bu özellikler sıralanmıştır:

- Programlanabilir dijital paralel giriş/çıkış.
- Programlanabilir analog giriş/çıkış.
- Seri giriş/çıkış (senkron, asenkron ve cihaz denetimi gibi).
- Motor veya servo kontrol için pals sinyali çıkışı.
- Harici giriş vasıtasıyla kesme.
- Timer vasıtasıyla ile kesme.
- Harici bellek arabirimi.
- Harici bus arabirimi (PC ISA gibi).
- Dahili bellek tipi seçenekleri(ROM, EPROM, PROM ve EEPROM).

- Dahili RAM seçeneği.
- Kayan nokta hesaplaması.

Daha da ayrıntıya girecek olursak bu listede sıralanacak özellikler uzayıp gidecektir. Şimdi de bizim bu kitapta ele aldığımız Microchip'in ürünü olan PIC'i neden seçtiğimize değinelim. Microchip, 8-bit'lik mikrodeneleyici ve EEPROM üreten bir Amerikan şirkettir. Arizona eyaletinde iki, Tayland ve Tayvan'da da birer tane olmak üzere toplam dört fabrika ile kendi alanında dünyada söz sahibi olan bir chip üreticisidir.

Neden PIC?

Bilgisayar denetimi gerektiren bir uygulamayı geliştirirken seçilecek mikrodeneleyicinin ilk olarak tüm isteklerinizi yerine getirip getirmeyeceğine, daha sonra da maliyetinin düşüklüğüne bakmalısınız. Ayrıca, yapacağınız uygulamanın devresini kurmadan önce seçtiğiniz mikrodeneleyicinin desteklediği bir yazılım üzerinde simülasyonunu yapip yapamayacağınızı da dikkate almalısınız.

Yukarda saydığımız özellikleri göz önüne aldığımızda Microchip'in ürettiği PIC'leri kullanmak en akılcı bir yol olduğunu görülmektedir. İşte, bu kitapta PIC'leri ele alınmamızın nedenlerini şöyle sıralayabiliriz.

- Yazılımın Microchip'ten veya internetten parasız olarak elde edilebilmesi.
- Çok geniş bir kullanıcı kitlesinin bulunması.
- PIC'lerin çok kolaylıkla ve ucuz olarak elde edilebilmesi.
- Elektronik hobi olarak uğraşanların bile kullanabildikleri basit elemanları kullanarak yapılan donanımla programlanabilmesi.
- Çok basit reset, clock sinyali ve güç devreleri gerektirmeleri.

PIC, adını İngilizce'deki Peripheral Interface Controller cümlesindeki kelimelerin baş harflerinden almış olan bir mikrodeneleyicidir. Eğer bu cümleyi Türkçe'ye çevirirsek, **çevresel üniteleri denetleyici arabirim** gibi bir anlam çıkacaktır. PIC gerçekten de çevresel üniteler adı verilen lamba, motor, role, ısı ve ışık sensörü gibi 1/0 elemanların denetimini çok hızlı olarak yapabilecek şekilde dizayn edilmiş bir chip'tir. RISC mimarisi adı verilen bir yöntem kullanılarak üretildiklerinden bir PIC'i programlamak için kullanılacak olan komutlar oldukça basit ve sayı olarak da azdır. 1980'lerin başından itibaren uygulanan bir tasarım yöntemi olan RISC (Reduced Instruction Set Computer) mimarisindeki temel düşünce, daha basit ve daha az komut kullanılmasıdır. Örneğin PIC16F84 mikrodeneleyicisi toplam 35 komut kullanılarak programlanabilmektedir.

Neden PIC16F84?

Bu kitapta programlanması ve örnek uygulamaları verilen PIC'in 16F84 serisi olmasının en önemli nedeni: PIC16F84 (veya PIC16F84A) mikrodeneleyicisinin program belleğinin flash teknolojisi ile üretilmiş olmasıdır.

Flash memory teknolojisi ile üretilen bir belleğe yüklenen program, chip'e uygulanan enerji kesilse bile silinmez. Yine bu tip bir belleğe istenirse yeniden yazılabilir. Flash bellekler bu özellikleri ile EEPROM bellekler ile aynı görünmektedirler. Gerçekten de Flash ile EEPROM bellek aynı şeylerdir. Ancak bazı üreticiler tarafından EEPROM belleğe Flash ROM da denilmektedir.

Flash belleğe sahip olan PIC16F84'i programlayıp ve deneylerde kullandıktan sonra, silip yeniden program yazmak PIC ile yeni çalışmaya başlayanlar için büyük kolaylıktır. Böylece işe yeni başlayanlar yaptıkları programlama hataları nedeniyle chip'i atmak zorunda kalmayacaklardır. Gerçi EPROM program memory'si olan chip'lere de yeniden yazmak mümkündür ama, bu durumda bir EPROM silici cihazına ihtiyaç vardır. Bir silici cihaz bulunsa bile programı bellekten silmek için en azından 10-15 dk beklemek zorunda kalınacaktır. İşte PIC16F84'ün bu özelliği mikrodeneleyici kullanmaya yeni başlayanlar için ideal bir seçenektir.

PIC16F84'ü seçmemizin ikinci nedeni de, programlama donanımının çok ucuz ve kullanışlı olması ve hatta çoğu meraklı elektronik kullanıcı tarafından bile üretilebilmesidir. Kitabın Ekler bölümünde adresini verdiğimiz firmanın ürettiği programlayıcı donanımı ve yazılımı ödemeli olarak istenebilmesi Türkiye'deki kullanıcılar için çok büyük bir avantajdır.

PIC16F84'ü programlamak için öğrendiğiniz her şeyi diğer PIC 16/17 mikrodeneleyicilerinin uygulamalarında da kullanabilmeniz, yapılan seçimin doğruluğunu göstermektedir.

PIC PROGRAMLAMAK İÇİN NELERE İHTİYACINIZ VAR?

PIC 16/17 mikrodeneleyicilerin programlamasını ve uygulamalarda nasıl kullanılacağını öğrenmek için neleri bilmek ve nelere sahip olunması gerekenler aşağıda sıralanmıştır:

- IBM uyumlu bir bilgisayara sahip olmak ve temel kullanımları bilmek.

- Bir metin editörünü kullanmasını bilmek.
- Bir assembler programına sahip olmak.
- PIC programlayıcı donanımına sahip olmak.
- PIC programlayıcı yazılımı.
- PIC
- Programlanmış PIC'i denemek için breadboard, güç kaynağı ve elektronik elemanlar.
- Programlanmış bir PIC'i deneme kartı.

IBM Uyumlu Bilgisayar

Assembly program kodlarını kolayca yazabilmek, doğru ve hızlı bir şekilde PIC'in program belleğine gönderebilmek için bilgisayara ihtiyaç vardır. Bir metin editörü kullanarak yazılan program kodları, derlendikten sonra PIC'e gönderilmesi gerekir. Program kodlarının PIC'e yazdırma işlemi paralel veya seri porta bağlanan bir elektronik devre aracılığı ile yapılır. Bu işleri yapabilmek için bilgisayarın temel kullanım fonksiyonlarını bilmeniz gerekir. Aşağıda bilmeniz gereken bazı temel işlemleri ve sahip olmanız gereken minimum konfigürasyonu veriyoruz:

- DOS ya da VVINDOVVS işletim sistemi bildiğinizi, bu işletim sistemi. komutlarıyla klasör oluşturma, dosya kopyalama ve silme, listeleme gibi işlemleri yapabildiğinizi.
- Basit bir editör (EDIT, Notpad gibi) kullanabildiğinizi, bu editörde bir text dosyası oluşturup disket ya da hard diske kaydedebildiğinizi, diskteki bir dosyayı yükleyip üzerinde düzeltmeler yapabileceğinizi,
- Minimum 80486 CPU, 4 MB RAM, 100 MB harddisk ve CD-ROM sürücüsü (Microchip'in CD'lerini kullanabilmek için) bulunan bir PC'ye sahip olduğunuzu kabul ediyoruz.

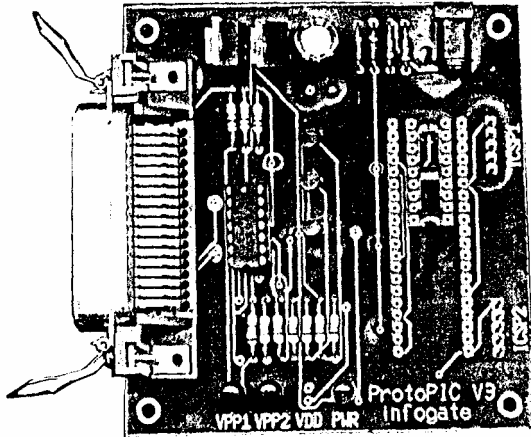
Metin Editörü

Assembly dili komutlarını yazıp bir metin dosyası oluşturmak için EDIT veya NotPad gibi bir editörü kullanabilmeniz gerekir. İsterseniz ASM uzantılı metin dosyalarınızı yazabileceğiniz **PFE** editörünü de kullanabilirsiniz. Bu editörün hem DOS hem de VVINDOVVS altında çalışan versiyonları bulunmaktadır ve PIC konusunda destek veren bir internet sitesinden alınmıştır. Ekler bölümünde adını verdiğimiz firma da bu programı disket içerisinde sunmaktadır.

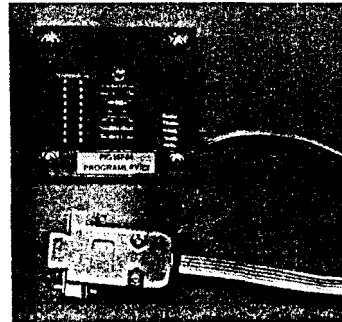
Assembler Programı

PIC Assembly dili adı verilen ve toplam 35 komuttan oluşan programlama dilini bu kitapta öğreneceksiniz. Bu komutları basit bir editörde yazabiliyoruz. Ancak, İngilizce'deki bazı kelimelerin kısaltmasından oluşan bu dilin komutlarını PIC'in anlayabileceği makine diline çeviren bir programa ihtiyacımız vardır. Bu programa assembler adını veriyoruz. Text dosyası biçiminde kaydedilmiş olan assembly dili komutlarını makine diline çeviren MPASM'nin hem DOS altında hem de WINDOWS altında çalışan versiyonu bulunmaktadır. Bu program Microchip firmasının internetteki www.microchip.com adlı sitesinden parasız olarak download edilebileceği gibi kitabın Ekler bölümünde adresi verilen firmadan da elde edebilirsiniz. MPASM'nin kullanımı hakkında detaylı bilgiyi 5. bölümde bulacaksınız.

Microchip bir de içerisinde hem metin editörü hem MPASM assembler programını bulunduran **MPLAB** programını PIC programlayıcılarının kullanımına sunmaktadır. Bu programın bulunduğu CD-ROM yine www.microchip.com adresinden parasız olarak istenebilir. MPLAB'ın kurulumu ve kullanılmasıyla ilgili gerekli detay bilgiyi Ekler bölümünde bulacaksınız.



ProtoPIC programlama kartı



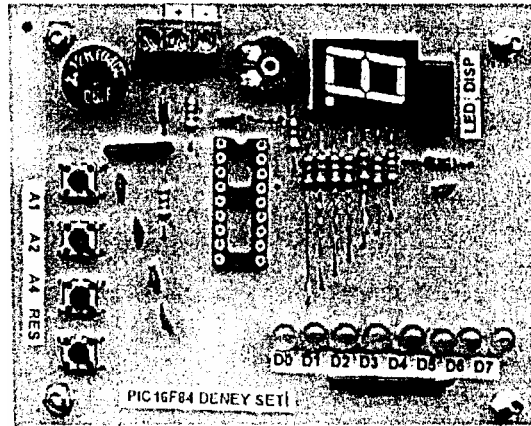
Yukarıda (sağda) PIC16F84 ve PIC16C84 mikrodenetleyicileri programlayabileceğiniz, çok basit bir donanımı bulunan programlama kartı resmi verilmiştir. Bu kartla ilgili detaylı bilgiyi Ekler bölümünde bulacaksınız.

PIC Programlayıcı Yazılımı

MPASM tarafından derlenerek makine diline dönüştürülmüş assembly programı kodlarının PIC'e yazdırılmasında kullanılan bir programa gereksinim vardır. Programlayıcı yazılımları, PIC'i programlamak için kullanılan elektronik karta bağımlıdır. Yani her programlayıcı yazılımı ile elinizde bulunan karta kod gönderemeyebilirsiniz. Genellikle programlama kartı üreticileri, ürettikleri karta uygun yazılımı da birlikte sunarlar. Biz bu kitapta ProtoPIC programlama kartına gönderdiği kodlarla PIC'leri sorunsuz olarak programlayan **P16PRO** adlı yazılımı kullandık ve nasıl kullanılacağını anlattık. P16PRO'yu Windows ortamından da çalıştırabileceğiniz için bu size büyük kolaylık sağlayacaktır.

Programlanmış PIC'i Deneme Kartı

Programladığınız PIC'i breadboard üzerinde kendi kurduğunuz devre de deneyebileceğiniz gibi şekilde görülen özel bir deneme kartı üzerinde de deneyebilirsiniz. Başlangıç ve orta düzey PIC programlayıcılara hitap etmek üzere geliştirilen bu kart üzerinde deneme yapmak, breadbord üzerinde devre kurmaktan çok daha kolaydır. PIC'in port çıkışlarındaki sinyalleri izlemek amacıyla 8 tane LED, bir tane de 7 segmentli LED yerleştirilmiştir. Kart üzerindeki, butonlar, potansiyometre ve LED'ler aracılığıyla farklı şekilde programlanan PIC'lerin kolayca denenmesini sağlar. Bu kartın yapısı ve kullanılması hakkında Ekler bölümünde geniş bilgi bulacaksınız.



2

PIC DONANIM ÖZELLİKLERİ

- ❑ PIC ÇEŞİTLERİ
 - ❑ PIC'LERİN GÖRÜNÜŞÜ
 - ❑ PIC BELLEK ÇEŞİTLERİ
-

PIC ÇEŞİTLERİ

Microchip ürettiği mikrodenetleyicileri 4 farklı gruba (Genellikle aile diye adlandırılır.) ayırarak isimlendirmiştir. PIC ailelerine isim verilirken kelime boyu (VVord lenght) göz önüne alınmıştır. Şimdi kelime boyunun ne anlama geldiğine bakalım. Microprocessor veya mikrodenetleyiciler kendi içlerindeki dahili veri saklama alanları olan registerleri arasındaki veri alış verişini farklı sayıdaki bit'lerle yaparlar. Örneğin 8088 mikro işlemcisi chip içerisindeki veri alış verişini 16-bit ile yaparken, Pentium işlemcileri 32-bit'lik verilerle iletişim kurarlar. Bir CPU veya MCU'nun dahili veri yolu uzunluğuna **kelime boyu** denir.

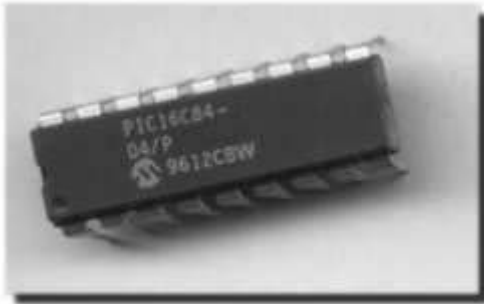
Microchip PIC'leri 12/14/16 bit'lik kelime boylarında üretmektedir ve buna göre aşağıdaki aile isimlerini vermektedir.

PIC16C5XX ailesi 12-bit kelime boyu,
PIC16CXXX ailesi 14-bit kelime boyu,
PIC17CXXX ailesi 16-bit kelime boyu,
PIC12CXXX ailesi 12-bit/14-bit kelime boyuna sahiptir.

Bir CPU veya MCU'nun chip dışındaki harici ünitelerle veri alışverişini kaç bit ile yapıyorsa buna **veri yolu** bit sayısı denir. PIC'ler farklı kelime boylarında üretilmelerine rağmen harici veri yolu tüm PIC ailesinde 8-bit'tir. Yani bir PIC, I/O portu aracılığı ile çevresel ünitelerle veri alışverişi yaparken 8-bit'lik veri yolu kullanır.

PIC programlayıcıları program kodlarını yazarken bir komutun kaç bit'lik bir kelime boyundan oluştuğuyla pek fazla ilgilenmezler. Seçilen bir chip'i programlarken uyulması gereken kuralları ve o chip'le ilgili özelliklerin bilinmesi yeterlidir. Bu özellikler PIC'in bellek miktarı, I/O portu sayısı, A/D dönüştürücüye sahip olup olmadığı, kesme (interrupt) fonksiyonlarının bulunup bulunmadığı, bellek tipinin ne olduğu (Flash, EPROM, EEPROM vb.) gibi bilgilerdir. Bu özelliklerin en son değişikliklerini içeren güncel ve tam bir listesine microchip'in kataloglarından ulaşmak mümkündür. Aşağıdaki tabloda Nisan 1997 tarihine kadar üretilen PIC'lerin listesi verilmiştir. Katalogdan alınan bu tablo orijinali bozulmadan özellikle İngilizce olarak verilmiştir. Çünkü bu gibi bilgilere ulaşmak istenildiğinde Türkçe kataloglar bulmak mümkün değildir. Böylece katalog okurken PIC'lerle ilgili özellikleri veren İngilizce kelimeleri öğrenmeye zorlanmış olacaksınız. Tablo içerisinde anlayamadığınız kelimelerin anlamlarını kitabın sonundaki sözlükten bulabilirsiniz.

PIC'LERİN DIŞ GÖRÜNÜŞÜ



PIC'ler çok farklı ambalajlarla piyasaya sunulmaktadırlar. Bu kitapta program örnekleri verilen PIC16F84, 18-pinli DIP tipinde ambalajlanmış olanıdır. Standart DIP (Dual In-Line Package) olarak ambalajlanmış olan PIC'ler 0.3 inç genişliğindedir. Her iki tarafında 0.1 inç aralıklarla yerleştirilmiş 9 pin bulunmaktadır. PIC16F84 şekil-a da görülmektedir. Şekil-b de ise 0.6 inç genişliğinde, her iki tarafında 0.1 inç aralıklarla yerleştirilmiş 20 pini bulunan PIC16C74 görülmektedir. Bu örneklerin dışında farklı pin sayısına sahip başka DIP ambalajlı PIC'ler de bulunmaktadır. Geniş bilgi için Microchip'in kataloglarına bakmanızı öneririz.

PIC BELLEK ÇEŞİTLERİ

Farklı özellikte program belleği bulunan PIC'ler microchip firması tarafından piyasaya sürülmektedir. Bunlar:

- Silinebilir ve programlanabilir bellek (Erasable PROgrammable Memory-EPROM).
- Elektriksel olarak silinebilir ve programlanabilir bellek (Electrically Erasable PROgrammable Memory-EEPROM). **FLASH** bellek olarak da adlandırılır.
- Sadece okunabilir bellek (Read-Only Memory-ROM).

Her bir bellek tipinin kullanılacağı uygulamaya göre avantajları ve dezavantajları vardır. Bu avantajlar; fiyat, hız, defalarca kullanmaya yatkınlık gibi faktörlerdir.

EPROM bellek hücrelerine elektrik sinyali uygulayarak kayıt yapılır. EPROM üzerindeki enerji kesilse bile bu program bellekte kalır. Ancak silip yeniden başka bir program yazmak için ultra-viole ışını altında belirli bir süre tutmak gerekir. Bu işlemler EPROM silici denilen özel aygıtlarla yapılır. EPROM bellekli PIC'ler iki farklı ambalajlı olarak bulunmaktadır:

- Seramik ambalajlı ve cam pencereci olan tip, silinebilir olan tiptir.
- Plastik ambalajlı ve penceresiz olan tipler ise silinemez (OTP) tiptir.

Seramik ambalajlı ve pencereci olan bellek içerisindeki programın silinmemesi için pencere üzerine ışık geçirmeyen bir bant yapıştırılır. Ultra-viole ışığı ile silinmesi istenildiğinde bu pencere açılır ve silici aygıt içerisinde belirli bir süre bekletilir. Plastik ambalajlı EPROM'lar ise programlandıktan sonra silinmesi mümkün değildir ve fiyatı silinebilen tipe göre oldukça ucuzdur. Silinemeyen tipe OTP (One Time Programmable - Bir defa programlanabilir) olarak adlandırılır.

EEPROM belleği bulunan bir PIC içerisinde program yazmak için PIC programlayıcı vasıtasıyla elektriksel sinyal gönderilir. EEPROM üzerindeki enerji kesilse bile bu program bellekte kalır. Programı silmek veya farklı yeni bir program yazmak istendiğinde PIC programlayıcıdan elektriksel sinyal gönderilir. Bu tip belleğe sahip olan PIC'ler genellikle uygulama geliştirme amacıyla kullanılırlar. Microchip bu tip belleğe çoğu zaman **FLASH** bellek olarak da adlandırmaktadır. Fiyatları silinemeyen tiplere göre biraz pahalıdır. Bellek erişim hızları ise EPROM ve ROM'lara göre daha yavaştır. PIC 16C84 ve PIC 16F84'ler bu tip program belleğine sahiptir.

ROM program belleğine sahip PIC'lerin programları fabrikasyon olarak yazılırlar. EPROM ve EEPROM eşdeğerlerine nazaran fiyatları oldukça düşüktür. Ancak fiyatının düşüklüğünden dolayı gelen avantaj bazen çok pahalıya da mal olabilir. ROM bellekli PIC programlarının fabrikasyon olarak yazılması nedeniyle PIC'in elde edilme süresi uzundur. Programda oluşabilecek bir hatanın PIC'e program yazıldıktan sonra tespit edilmesi, eldeki tüm PIC'lerin atılmasına da neden olabilir. Bu tip PIC'ler çok miktarda üretilecek bir ürünün maliyetini düşürmek amacıyla seçilir. Program hataları giderilemediği için uygulama geliştirmek için uygun değildir. Microchip, ROM program bellekli PIC'lere parça numarası verirken "CR" (PIC16CR62, PIC16CR84 gibi) harfleri kullanılır.

3

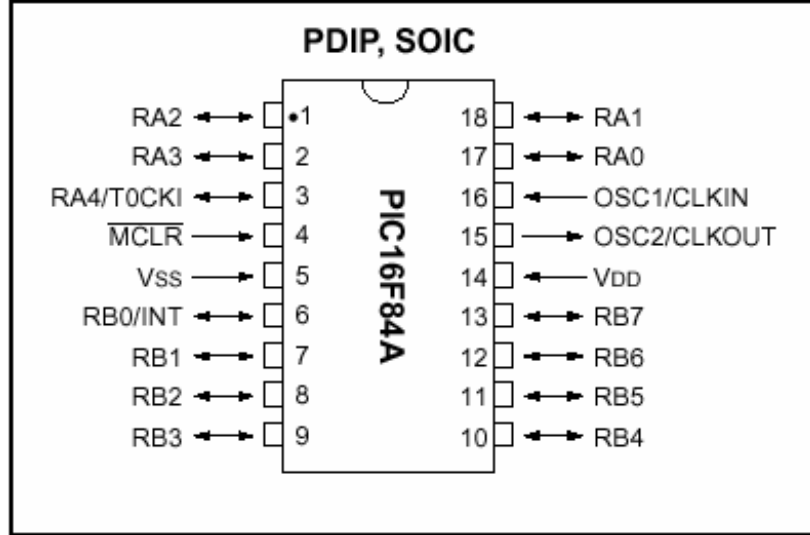
PIC16F84

□ PIC16F84'ÜN PIN GÖRÜNÜŞÜ

□ BESLEME GERİLİMİ

- ❑ CLOCK UÇLARI VE CLOCK OSİLATÖRÜ ÇEŞİTLERİ
- ❑ RESET UCU VE RESET DEVRESİ
- ❑ I/O PORTLARI
- ❑ PIC16F84'ÜN BELLEĞİ

PIC 16F84'ÜN PIN GÖRÜNÜŞÜ

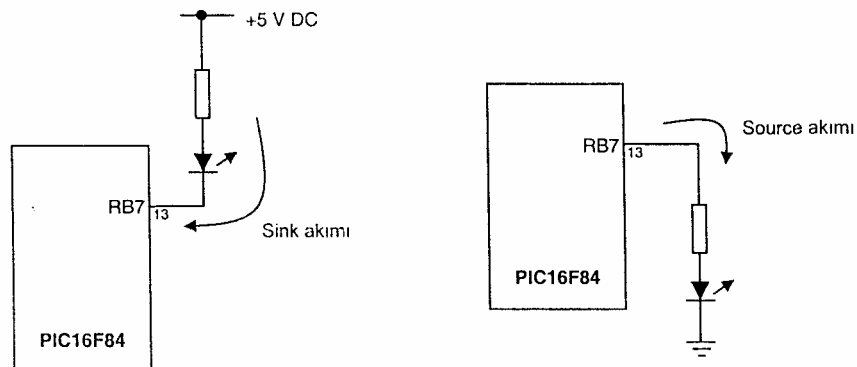


CMOS teknolojisi ile üretilmiş olan PIC16F84 çok az enerji harcar. Flash belleğe sahip olması nedeniyle clock girişine uygulanan sinyal kesildiğinde registerleri içerisindeki veri aynen kalır. Clock sinyali tekrar verildiğinde PIC içerisindeki program kaldığı yerden itibaren çalışmaya başlar. RAO-RA3 pinleri ve RBO-RB7 pinleri I/O portlarıdır. Bu portlardan girilen dijital sinyaller vasıtasıyla PIC içerisinde çalışan programa veri girilmiş olur. Program verileri değerlendirilerek portları kullanmak suretiyle dış ortama dijital sinyaller gönderir. Dış ortama gönderilen bu sinyallerin akımı yeterli olmadığı durumda yükselteç devreleri (röle, transistör v.s) ile yükseltilerek kumanda edilecek cihaza uygulanır. Portların maksimum sink ve source akımları aşağıda verilmiştir. Bu akımlar genellikle bir LED sürmek için yeterli olduğundan, bu kitapta verilen uygulama devrelerinde herhangi bir yükseltme işlemi yapılmamıştır.

I/O pini

Sink akımı	25 mA
Source akımı	20 mA

Hatırlatma amacıyla sink ve source akımlarının ne olduğundan bahsedelim. Sink akımı, gerilim kaynağından çıkış potuna doğru akan akıma, source akımı ise I/O pininden GND ucuna doğru akan akıma denir.

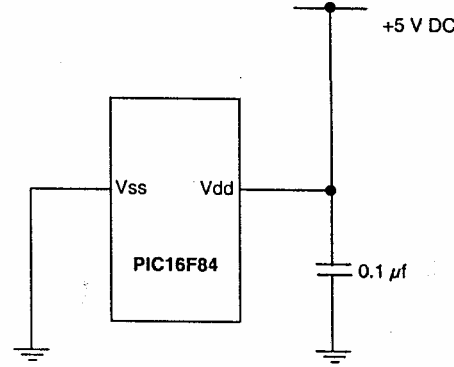


PIC16F84'ün çektiği akım, besleme gerilimine, clock girişine uygulanan sinyalin frekansına ve I/O pinlerindeki yüke bağlı olarak değişir. Tipik olarak 4 MHz'lik clock frekansında çektiği akım 2 mA' kadardır. Bu akım uyuma modunda (Sleep mode) yaklaşık olarak 40 μ A' e düşer. Bilindiği gibi CMOS entegrelerdeki giriş uçları muhakkak bir yere bağlanır. Bu nedenle kullanılmayan tüm girişler besleme geriliminin +5V luk ucuna bağlanmalıdır.

PIC16C84 ve PIC16F84 özellikleri tamamen aynı olan PIC'lerdir. Her ikisi de FLASH belleğe sahip olmalarına rağmen Microchip ilk ürettiği EEPROM bellekli PIC'ler "C" harfi (C harfi CMOS'dan gelmektedir.) ile tanımlarken, son zamanlarda ürettiği EEPROM bellekli PIC'leri "F" harfi (FLASH) ile tanımlamaktadır. Bizim bu kitapta örneklerini verdiğimiz programlar için her iki PIC de kullanılabilir. PIC16F84A ile PIC16F84 arasında da herhangi bir fark yoktur. PIC'i tanımlayan bu harf ve rakamlardan sonra yazılan 10/P, 04/P clock girişine uygulanacak maksimum frekansı belirtir. Örneğin 10 MHz'e kadar frekanslarda PIC16F84-10/P kullanılırken, 4 MHz'e kadar frekanslarda PIC16F84-04/P kullanılabilir.

BESLEME GERİLİMİ

PIC'in besleme gerilimi 5 ve 14 numaralı pinlerden uygulanır. 5 numaralı Vdd ucu +5 V'a, 14 numaralı Vss ucu da toprağa bağlanır. PIC'e ilk defa enerji verildiği anda meydana gelebilecek gerilim dalgalanmaları nedeniyle istenmeyen arızaları önlemek amacıyla Vdd ile Vss arasına 0.1 μ F lık bir dekuplaj kondansatörü bağlamak gerekir. PIC'ler CMOS teknolojisi ile üretildiklerinden çok geniş besleme gerilimi aralığında (2 ~ 6 V) çalışmalarına rağmen 5 V luk gerilim deneyler için ideal bir değerdir.



PIC16F84'e besleme geriliminin bağlanması

CLOCK UÇLARI ve CLOCK OSİLATÖRÜ ÇEŞİTLERİ

PIC belleğinde bulunan program komutlarının çalıştırılması için bir kare dalga sinyale ihtiyaç vardır. Bu sinyale clock sinyali denilir ve Türkçe'de "klok" olarak okunur. PIC16F84'ün clock sinyal girişi için kullanılan iki ucu vardır. Bunlar OSC1(16 pin) ve OSC2 (15.pin) uçlarıdır. Bu uçlara farklı tipte osilatörlerden elde edilen clock sinyalleri uygulanabilir. Clock osilatör tipleri şunlardır:

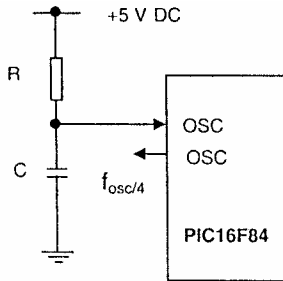
- RC** - Direnç/kondansatör (Resistor/Capacitor).
- XT** - Kristal veya seramik resonatör (Xtal).
- HS** - Yüksek hızlı kristal veya seramik resonatör (High Speed).
- LP** - Düşük frekanslı kristal (Low Power).

Seçilecek olan osilatör tipi PIC'in kontrol ettiği devrenin hız gereksinimine bağlı olarak seçilir. Aşağıdaki tablo hangi osilatör tipinin hangi frekans sınırları içerisinde kullanılabileceğini gösterir.

<u>Osilatör tipi</u>	<u>Frekans sınırı</u>
RC	0 – 4 MHz
LP	5 – 200 KHz
XT	100 KHz – 4 MHz
HS (-04)	4 MHz
HS (-10)	4 – 10 MHz
HS (-20)	4 – 20 MHz

PIC'e bağlanan clock osilatörünün tipi programlama esnasında PIC içerisinde bulunan konfigürasyon bitlerine yazılmalıdır. Osilatör tipini belirten kodları (RC, XT, HS ve LP) kullanarak konfigürasyon bitlerinin nasıl yazılacağı programlama örnekleri verilirken detaylı olarak incelenecektir.

RC clock osilatörü, PIC'in kontrol ettiği elektronik devredeki zamanlamanın çok hassas olması gerekmediği durumda kullanılır. Belirlenen değerden yaklaşık %20 sapma gösterebilirler. Bir direnç ve kondansatörden oluşan bu osilatörün maliyeti oldukça düşüktür. OSC1 ucundan uygulanan clock frekansı R ve C değerlerine bağlıdır. Şekilde RC osilatörün clock girişine bağlantısı ve çeşitli R, C değerlerinde elde edilen osilatör frekansları örnek olarak verilmiştir.

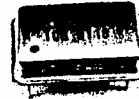
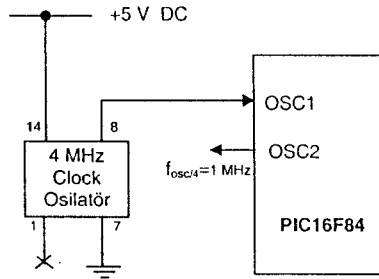


R	C	f _{osc} (yaklaşık)
10 K	20 pF	625 KHz
10 K	220 pF	80 KHz
10 K	0,1 µF	85 KHz

RC osilatörün PIC'e bağlantısı ve örnek R, C değerleri

OSC1 ucundan uygulanan harici clock frekansının 1/4'ü OSC2 ucunda görülür. Bu clock frekansı istenirse devrede kullanılan diğer bir elemanı sürmek için kullanılabilir.

Kristal kontrollü clock osilatörleri zamanlamanın çok hassas olması gerektiğinde kullanılır. Bu tip clock osilatörleri metal bir kutu görünümündedir ve şekilde görülmektedir. Bu tip osilatörlere kondansatör bağlantısı gerekmez. PIC assembly programlama dili ile yazılan zaman geciktirme (Time delay) döngülerinde yapılacak hesaplamaları kolaylaştırmak için genellikle 4 MHz'lik kristal clock osilatörleri kullanılması tavsiye edilir. Bu durumda harici clock frekansı (OSC1) 4'e bölündüğünde, dahili clock frekansı 1 MHz olur (OSC2). Çoğu PIC assembly komutu bir komut saykılı süresinde (dahili clock) çalıştığından, bir komutun işlevini gerçekleştirme süresi 1 mikro saniye olur. Bu süre ise deneysel çalışmalar için oldukça uygundur.



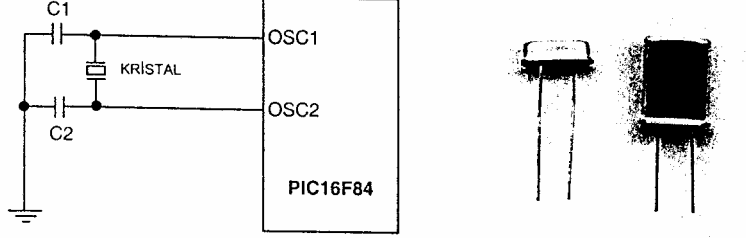
Kristal clock osilatörün PIC'e bağlantısı

Kristal ve kondansatör kullanılarak yapılan osilatörler de zamanlamanın önemli olduğu yerlerde kullanılır. Kristal osilatörlerin kullanıldığı devrelerde kristale bağlanacak kondansatörün seçimine özen göstermek gerekir. Aşağıda hangi frekansta kaç (µF lık kondansatör kullanılacağını gösteren tablo görülmektedir.

OSİLATÖR TİPİ	FREKANS	KONDANSATÖR
LP	32 KHz	33 – 68 pF
	200 KHz	15 – 47 pF
	100 KHz	47 – 100 pF
XT	500 KHz	20 – 68 pF
	1 MHz	15 – 68 pF
	2 MHz	15 – 47 pF
	4 MHz	15 – 33 pF

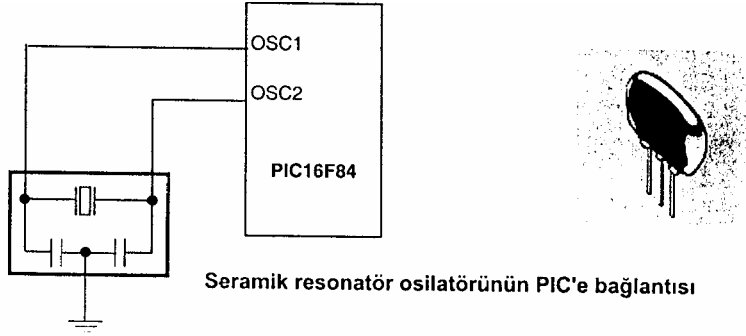
HS	8 MHz	15 – 47 pF
	20 MHz	15 – 47 pF

Seçilen kondansatör değerlerinin yukarıdaki değerlerden yüksek olması, elde edilen kare dalgaların bozuk olmasına ve PIC'in çalışmamasına neden olur. C1 ve C2 kondansatörlerin değerleri birbirine eşit olmalıdır.



Kristalin ve kondansatörlerin PIC'e bağlantısı

Seramik resonatörler, içerisinde kondansatörleri hazır bulunan osilatörlerdir. Fiyatları ucuz ve hassastırlar (+/- %1.3). Küçük bir seramik kondansatöre benzeyen resonatörlerin üç ucu vardır. Bu uçlardan ortadaki toprağa, diğer iki ucu da OSC1 ve OSC2 uçlarına bağlanırlar. Hangi ucun OSC1'e bağlanacağı önemli değildir, her ikisi de bağlanabilir.

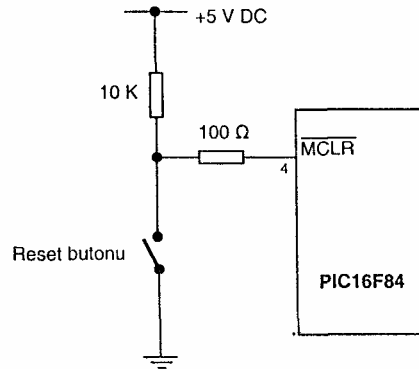


Seramik resonatör osilatörünün PIC'e bağlantısı

RESET UÇLARI VE RESET DEVRESİ

PIC16F84'ün besleme uçlarına gerilim uygulandığı anda bellekteki programın başlangıç adresinden itibaren çalışmasını sağlayan bir reset devresi vardır. Bu reset devresi PIC içerisinde ve "Power-on-reset" denir.

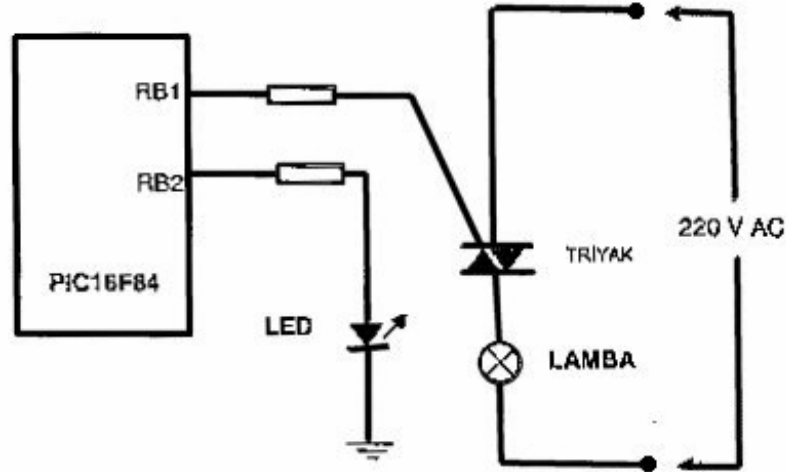
MCLR ucu ise kullanıcının programı kesip, kasti olarak başlangıca döndürebilmesi için kullanılır. PIC'in 4 numaralı MCLR ucuna uygulanan gerilim 0 V olunca programın çalışması başlangıç adresine döner. Programın ilk adresten itibaren tekrar çalışabilmesi için reset ucuna uygulanan gerilimin +5 V olması gerekir. Bir buton aracılığı ile reset işlemini yapan devre şekilde görülmektedir.



PIC16F84'ün reset devresi

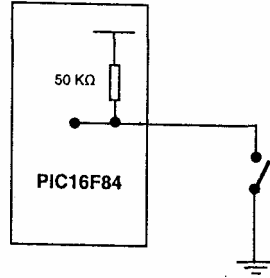
I/O PORTLARI

PIC16F84'ün 13 adet giriş/çıkış portu vardır. Bunlardan 5 tanesine A portu (RA0-RA4), 8 tanesine de B portu (RB0-RB7) denir. 13 portun her biri giriş ya da çıkış olarak kullanılabilir. PIC içerisinde adına TRIS denilen özel bir data yönlendirme registeri vardır. Bu register aracılığı ile portların giriş/çıkış yönlendirmesinin nasıl yapılacağı hakkında detaylı bilgiyi programlama konusunda bulacaksınız. I/O portlarından geçebilecek 25 mA lik bir sink akımı veya 20 mA lik source akımı LED'leri doğrudan sürebilir. Bu akımlar aynı zamanda LCD, lojik entegre ve hatta 220 V luk şehir şebekesine bağlı bir lambayı kontrol eden triyakı bile tetiklemeye yeterlidir. Çıkış akımı yetmediği durumda yükselteç devreleri kullanarak daha yüksek akımlara kumanda etmek mümkün olabilir.



PIC16F84'ün portlarıyla kontrol edilen 220 V luk lamba ve LED

B portunun 8 ucu, PIC içerisinde dahili olarak 50 K Ω luk dirençlerle pull-up yapılmış gibi etki gösterir. Bu durum Şekilde temsili olarak görülmektedir PIC'in içerisinde gerçekte bir pull-up direnci değil, farklı bir mantıksal devre vardır.



PIC16F84'ün B port uçlarının dahili olarak pull-up yapılması

Bu 8 pull-up direncinin tamamı option register içerisindeki yazılım aracılığıyla iptal (Disable) edilebilir veya geçerli (Enable) kılınabilir. Port uçlarından herhangi birisi çıkış olarak yönlendirildiğinde o uçtaki pull-up direnci otomatik olarak iptal olur. PIC'e enerji verildiğinde (Power-on-reset) ise tüm pull-up'lar iptal edilir.

A portunun 4. biti, TOCKI adı verilen harici timer/counter giriş ucu ile ortaklaşa kullanılır. Bu nedenle 16F84'ün pin görünüşü üzerinde 3 numaralı pin ucuna RA4/TOCKI yazılmıştır. RA4 ucu çıkış olarak yönlendirildiğinde açık kollektör özelliğinden dolayı harici olarak muhakkak bir pull-up direncine bağlanmalıdır. RA4 ucundan sadece sink akımı geçebilir, kaynak (source) akımı vermesi mümkün değildir.

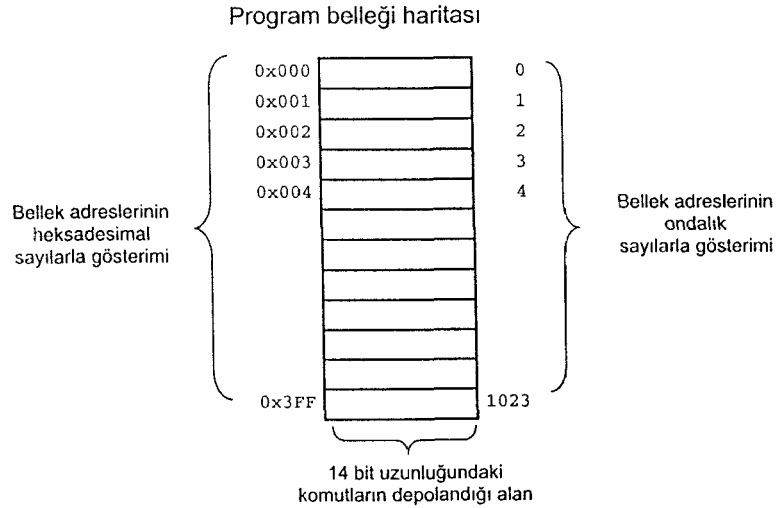
PIC16F84'ÜN BELLEĞİ

PIC16F84 mikrodenetleyicisinin belleği, program ve RAM belleği olmak üzere iki ayrı bellek bloğundan oluşur. Microchip'in kataloglarında PIC denetleyicilerin RISC işlemci olarak tanıtılmasının nedeni de budur. Çünkü Harvard Mimarisi ile üretilen RISC işlemcilerde program belleği ile data belleği birbirinden ayrıdır. Oysa PC'lerde kullanılan çoğu

mikroişlemci mimarisinde böyle bir ayrım yoktur. Bu da demektir ki; burada ayrıntısına girmeye gerek görmediğimiz nedenlerden dolayı mikroişlemciler, mikrodenetleyicilere göre komut işlemede daha yavaşırlar.

Program Belleği

PIC16F84'ün 1 Kbyte'lık program belleği vardır. Her bir bellek hücresi içerisine 14 bit uzunluğundaki program komutları saklanır. Program belleği flash (elektriksel olarak yazılıp silinebilir.) olmasına rağmen, programın çalışması esnasında sadece okunabilir.



PIC16F84'ün program belleği içerisinde sadece assembly komutları saklanır. Bu komutlar dışında RETLW komutu ile birlikte kullanılan sınırlı miktarda data da yüklenilebilir.

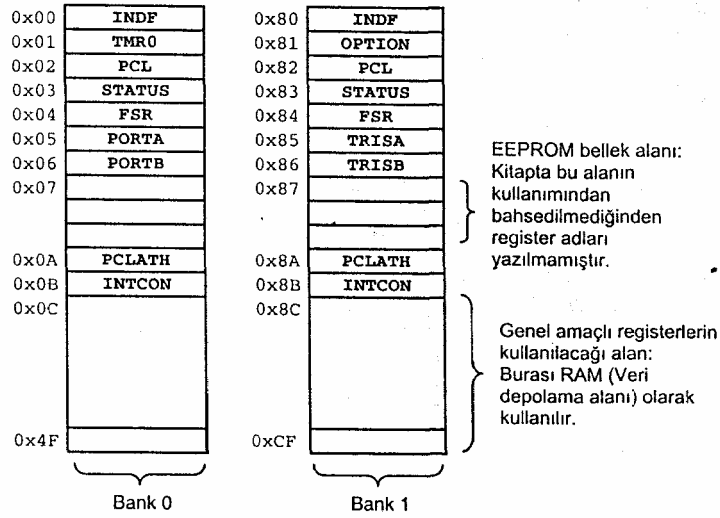
NOT: Yukarıda PIC16F84'ün program bellek haritasındaki adresler gösterilirken sol tarafta heksadesimal notasyon kullanılmıştır. PIC programlama esnasında da bellek adresleri bu şekilde yazılır. 0xXX heksadesimal notasyonunda X'ler 0~F arasındaki herhangi bir sayıyı, "0x" ise bu sayıların heksadesimal

olduğunu belirtir. Örneğin 0x0F, heksadesimal 0F sayısı demektir. 0x3FF ise 3FF heksadesimal sayısını gösterir. PIC16F84'ün program belleğine 14 bit uzunluğunda toplam 1024 tane komut yazılabilir. Bellek haritasında son bellek adresinin 0x3FF=1023 gösterilmesinin nedeni, adresin 0'dan başlamasındandır. Adres 1'den başlasaydı son adres 0x400=1024 olacaktı.

RAM Bellek

PIC16F84'ün 0x00~0x4F adres aralığında ayrılmış olan RAM belleği vardır. Bu bellek içerisindeki **file registerleri** içerisine yerleştirilen veriler PIC CPU'sunun çalışmasını kontrol ederler. File registerlerin bellek uzunluğu 8 bit'tir. Sadece PCLATH registeri 5 bit uzunluğundadır. File register adı verilen özel veri alanlarının dışında kalan diğer bellek alanları, normal RAM bellek olarak kullanılırlar. Yani bu alanlarda programda içerisindeki değişkenler için kullanılır.

File register haritası



PIC16F84'ün RAM belleği iki sayfadan (Bank'tan) meydana gelir. Bank0'daki registerlerin adresleri 0x00~0x4F arasında, bank1'deki registerlerin adresleri de 0x80~0xCF arasındadır. Toplam 80 tane file register olmasına rağmen programlamaya yeni başlayanlar bank1'de 80, bank 2'de 80 olmak üzere 160 register alanı olduğunu sanırlar. Oysa, dikkat edilirse bazı özel amaçlı registerler her iki bank'ta da görülür. Örneğin PCL, STATUS, PCLATH, INTCON gibi. Ayrıca 0x0C adresinden sonra RAM (data belleği) olarak kullanılan bölgedeki veriler 0x8C adresinden itibaren gölgelendirilmiştir(shadowed). Gölgeleme, otomatik kopyalama işlemi olarak algılandığında anlaşılması daha kolay olacaktır. örnekle izah edecek olursak: 0x8C adresine yazılan bir veri 0x00 adresinde de görülür. İşte bu sebeplerden dolayı 160 değil toplam 80 file register alanı vardır.

Bir bank'taki registeri kullanabilmek için o bank'a geçmek gerekir. Bank değiştirme işlemi programlama örnekleri verilirken detaylı olarak işlenecektir. Yalnız burada bir şeyden daha bahsetmek gerekir. Bazı özel registerlerin her iki bank'ta da görülmesinin nedeni, bank değiştirme işlemine gerek duyulmaksızın kullanılabilmesi içindir.

NOT: File register haritası üzerinde adresleri verilmiş olan özel registerlerin işlevini ve nasıl kullanıldığını bu aşamada anlamaya çalışmanız yararsız olacaktır. Programlama örnekleri verilirken çok daha kolay anlayacağınız düşüneyim.

W register

PIC16F84'ün RAM bellek alanında görülmeyen bir de W registeri vardır. W register bir akümülatör veya geçici depolama alanı olarak düşünülebilir. W registerine direkt olarak ulaşmak mümkün değildir. Ancak diğer registerlerin içerisindeki verileri aktarırken erişmek mümkündür. Bir PIC'te gerçekleşen tüm aritmetik işlemler ve atama işlemleri için W register kullanma zorunluluğu vardır. Örneğin iki register içindeki veriler toplanmak istendiğinde, ilk olarak registerlerden birinin içeriği W registre aktarılır. Daha sonra da diğer registerin içerisindeki veri W registeri içerisindekiyle toplanır. Bu registerin kullanım özellikleri yine programlama konusunda detaylı olarak ele alınacaktır.

4 PIC ASSEMBLY

□ ASSEMBLER NEDİR?

- PIC ASSEMBLY DİLİ NEDİR?
- PIC ASSEMBLY DİLİ YAZIM KURALLARI
- PIC ASSEMBLY KONUTLARININ YAZILIŞ BİÇİMİ
- SAYI VE KARAKTERLERİN YAZILIŞ BİÇİMİ
- PIC ASSEMBLY KOMUTLARI

ASSEMBLER NEDİR?

Assembler, bir text editöründe assembly dili kurallarına göre yazılmış olan komutları PIC'in anlayabileceği hexadesimal kodlara çeviren (derleyen) bir programdır. Microchip firmasının hazırladığı **MPASM** bu işi yapan assembler programıdır. Assembler'e çoğu zaman compilerde (derleyici) denilir.

PIC ASSEMBLY DİLİ NEDİR?

Assembly dili, bir PIC'e yaptırılması istenen işlerin belirli kurallara göre yazılmış komutlar dizisidir. Assembly dili komutları İngilizce dilindeki bazı kısaltmalardan meydana gelir. Bu kısaltmalar genellikle bir komutun çalışmasını ifade eden cümlelerin baş harflerinden oluşur. Böylece elde edilen komut, bellekte tutulması kolay (mnemonic) bir hale getirilmiştir. Örneğin:

BTFSC (Bit Test F Skip if Clear) - File registerdeki bit'i test et, eğer sıfırsa bir sonraki komutu atla, anlamında kullanılan İngilizce cümlelerin kısaltmasıdır.

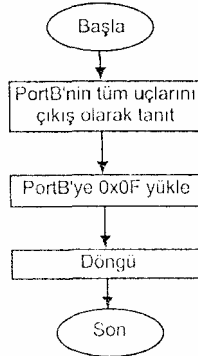
PIC ASSEMBLY DİLİ YAZIM KURALLARI

PIC assembly programlarının yazılması için kullanılan text editörlerinden 1.bölümde bahsetmiştik. Bu editörler Windows altında çalışan NOTPAD veya DOS altında çalışan EDIT en uygunlarıdır. Bunların dışında printer kontrol komutları içermeyen ve ASCII kodunda dosya üretebilen herhangi bir editör de kullanılabilir. MPLAB kullanıldığında ayrıca bir editör kullanmaya gerek yoktur. Çünkü MPLAB'ın içinde hem bir text editörü hem de MPASM bulunmaktadır.

MPASM assembler programının yazılan komutları doğru olarak algılayıp, PIC'in anlayabileceği hexadesimal kodlara dönüştürebilmesi için şu bilgiler program içinde özel formatta yazılması gerekir:

- Komutların hangi PIC16XX için yazıldığı,
- Programın bellekteki hangi adresten başlayacağı,
- Komutların ve etiketlerin neler olduğu,
- Programın bitiş yeri,

Basit bir örnekle bu bilgilerin program içinde nasıl yazıldığını gösterelim. Program ilk olarak PIC16F84'e B portunun 8 ucunu da çıkış olarak tanıttacak. Daha sonra bu porttaki ilk dört bitini lojik 1, sonraki dört bitini de lojik 0 yapacak. Son olarak program sonsuz bir döngüye girecektir. Bu işlemleri yapacak olan programın akış diyagramı ve komutları aşağıdaki gibi olacaktır.



;-----PCTEST1.ASM-----

-----10/3/2000-----

LIST P=16F84

; Adres tanımlama Bloğu

STATUS EQU 0x03

PORTB EQU 0x06

TRISB EQU 0x86

ORG 0x00 ;programı 0x00'dan başlat.

; portların durumunu belirleme bloğu

START

CLRF PORTB ; port B'nin içini sıfırla

BSF STATUS,5 ; BANK1'e geç

CLRF TRISB ; port B'nin uçlarını output yap

BCF STATUS,5 ; tekrar BANK0'a geç

```

;-----
;      program bloğu
;          MOVLW0x0F      ; W registerine 0x0F'i yükle
;          MOVWFPORTB     ; W'yi port B'ye yükle
;-----
;      sonlandırma bloğu
DONGU
        GOTO  DONGU
        END
;-----

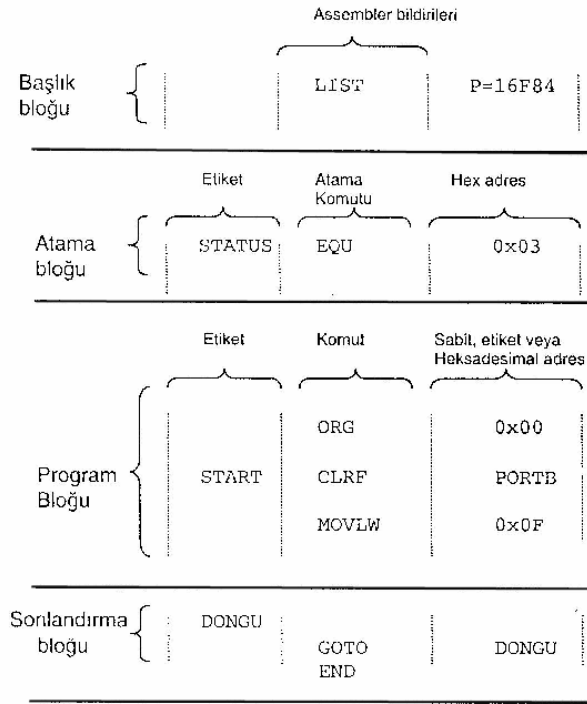
```

Noktalı Virgül (;)

Baş tarafına (;) konulan satır, assembler tarafından hexadesimal kodlara dönüştürülmez. Bu satırlar programın geliştirilmesi esnasında hatırlatıcı açıklamaların yazılmasında kullanılır. Örneğin CLRF ile başlayan satırda "portB'nin içeriği sıfırla" cümlesi, CLRF komutunun ne iş yaptığını açıklar. Programın bölümlerini birbirinden ayırmak için (----- veya =====) çizgileri kullanmak, programı görsel olarak daha okunur hale getirdiği gibi bu çizgiler arasına uyarılar ve açıklamalar da yazılabilir.

Girintiler ve Program Bölümleri

Text editörlerinde birbirinden farklı uzunlukta girintiler veren TAB özelliği vardır. Bu özellikten yararlanarak assembly komutları üç kolona bölünerek yazılır. Bir assembly programı temel olarak dört bölüme ayrılır. Bunlar: Başlık, atama, program ve sonuç bölümleridir.



Assembler, yukarıda komutların üç kolona olduğunu varsayar. Belirtilen kolona yazılmayan bir komut olduğunda ise bunu da kabul eder. Ancak, hexadesimal kodlara dönüştürme (Compile) esnasında hataları bir uyarı (Warning) olarak belirtir. Assembly komutları yazılırken kolonlar arasında verilen TAB'ların uzunluğu önemli değildir. SPACE tuşu ile verilen aralık da assembler tarafından TAB olarak algılanır.

Yukarıda verdiğimiz örnek, programlamaya yeni başlayanlar için karmaşık gelebilir. Ancak, uzun ve zor programlar yazmaya başladıktan sonra açıklamalar yapılmış, bölümlere ayrılmış pic programlarının daha kullanışlı olduğu görülür. Çünkü programlar bu şekilde yazıldığında daha sonraki geliştirmelere açıktır. Aradan zaman geçse bile bir programı geliştirmek için tekrar ele aldığımızda, program içerisine yazılan açıklamalar ihtiyacınız olan hatırlatmaları yapacaktır. Şimdi de verdiğimiz aynı örneği, her iki yazım biçimi arasındaki tercihi kendiniz yapabilmeniz için bölümler arasına boşluk vermeden, açıklamaları silerek sadece üç kolon halinde yeniden yazalım.

```

LIST      P=16F84
STATUS    EQU    0x03

```

belirtildiği gibi bölünerek yazılmış

```

PORTB    EQU    0x06
TRISB    EQU    0x86
          ORG    0x00

START

          CLRF   PORTB
          BSF    STATUS,5
          CLRF   TRISB
          BCF    STATUS,5
          MOVLW 0x0F
          MOVWF  PORTB

DONGU

          GOTO   DONGU
          END

```

Başlık

Programın en başındaki bilgilere başlık bölümü denilir.

```

===== PICTEST1.ASM =====10/03/2000=====
LIST P=16F84

```

Başlık bölümünde program dosyasının adı ve hazırlandığı tarih, istenirse hazırlayanın adı da yazılabilir. İlk satır, bir açıklama satırındır ve assembler tarafından derlenmez.

LIST P=16F84 satırı, programın hangi PIC için yazıldığını belirtir. LIST bir compiler bildirisi. Yani compiler'i yönlendiren bir komuttur ve yegane kullanım amacı yeri burasıdır.

Başlık bölümünde ayrıca verdiğimiz örnekte kullanılmayan INCLUDE komutu da kullanılabilir. INCLUDE komutu adresleri sabit olan STATUS, PORTA, PORTB, TRISA, TRISB gibi özel registerlerin "atamalar" bloğunda adreslerini her defasında belirtme zorunluluğunu ortadan kaldırmak için kullanılan bir compiler bildirisi. Bu bildirinin kullanılışı ilerdeki program örneklerinde daha detaylı olarak verilecektir.

Bu kitaptaki tüm program örnekleri verilirken yukarıdaki örnekte verdiğimiz başlığa benzer bir başlık kullanılacaktır.

Etiketler

PIC belleğindeki bir adresin atandığı, hatırlamayı kolaylaştıran kısaltmalardan meydana gelen sembolik isimlere **etiket** denilir. Örneğin PORTB etiketi, PIC16F84'ün file register belleğindeki B portunun bulunduğu adresi temsil eden etikettir. Etiketler program içerisinde 1. kolona yazılır.

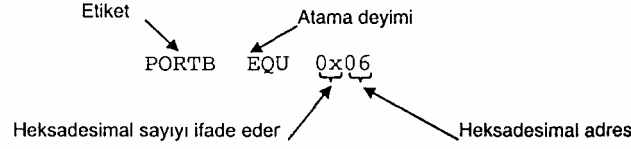
PORTB EQU 0x06 ifadesi program içerisinde yazıldıktan sonra B portunun hangi adreste olduğunu akılda tutmaya gerek yoktur. EQU (eşitleme) ifadesidir. Bu ifade BASIC programlama dilindeki (=), PASCAL programlama dilindeki (:=) ifade ile aynı anlamdadır. Programın herhangi bir yerinde PORTB etiketi kullanıldığında, B portunun adresi olan 0x06 yazılmış gibi işlem görür.

Birinci kolona yazılan ve adres atanmayan etiketler de kullanılabilir. Örneğin START ve DONGU bu tip etiketlerdir. Bu etiketler program akışını istenilen bir yere dallanmasını sağlamak amacıyla kullanılır. Program akışı yukarıdan aşağıya doğru devam ederken GOTO DONGU komutu ile, akış DONGU yazılan etikete dallandırılır. Bu etiketin adresi bir özel register adresi gibi fiziksel bir adres değildir. Bu şekilde tanımlanan bir etikete assembler otomatik olarak bir adres atar. Bu adresi bizim bilmemiz gerekmez.

- Etiket tanımlarken uyulması gereken kurallar şunlardır:
- Etiketler 1. kolona yazılmalıdır.
- Etiketler bir harfle veya alt çizgi(_) ile başlamalıdır.
- Etiketler içerisinde Türkçe karakterler kullanılamaz.
- Etiketler bir assembly komutundan oluşamaz.
- Etiketlerin içerisinde alt çizgi, rakam, soru işareti bulunabilir.
- Etiketler en fazla 31 karakter uzunluğunda olabilir.
- Etiketlerde büyük/küçük harf duyarlılığı vardır. ("Döngü" olarak tanımlanmış bir etiketi program içerisinde "döngü" yazarak kullanılamaz.)

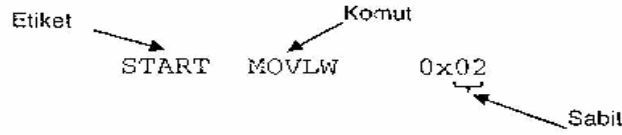
Atama deyimi (EQU)

EQU deyimi PIC16F84'ün belleğindeki bir hexadesimal adresi belirlenen bir etikete atamak için kullanılır. Aşağıda atama deyimine bir örnek gösterilmiştir.



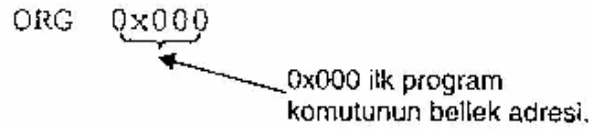
Sabitler

PIC assembly dilinde heksadesimal sayılar birer sabittir, Sabitler MOVLW ve bazı mantıksal ve aritmetik işlem komutlarında kullanılırlar.

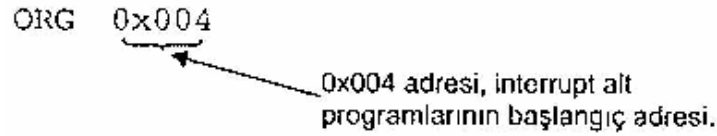


ORG Deyimi

ORG ingilizcedeki "origin" kelimesinden gelmektedir. ORG deyimi iki amaç için kullanılır.



- Program komutlarının hangi adresten itibaren başladığını gösterir.
- PIC16F84'ün Interrupt alt programlarının başlangıç adresini belirlemede kullanılır.



Sonlandırma Bloğu

PIC16F84'ün duraklama (halt) komutu yoktur. Programı belirli bir yerde duraklatmak için bazen sonsuz döngü kullanılır.

```
DONGU
GOTO DONGU
END
```

Yukarıdaki sonsuz döngüde DONGU etiketine assembler otomatik olarak bir adres verir. GOTO DONGU komutu ise program akışını devamlı olarak aynı adrese gönderir. Bu durumda program belirlenen adreste duraklatılmış olur.

END deyimi ise program komutlarının sona erdiğini assembler'a bildirir. Her program sonunda END deyimi muhakkak kullanılmalıdır. Aksi halde program devam derlenirken dosya sonunun belirtilmediğini belirten bir hata mesajı verecektir.

Büyük ve Küçük Harflerin Kullanımı

PIC assembler komutlarının büyük veya küçük harfle yazılması önemli değildir. İstenirse büyük/küçük harf karışımı komutlarda kullanılabilir. Örneğin Movlw, movlw, MOVLW komutları arasında hiçbir fark yoktur. Ancak etiketler büyük/küçük harf duyarlıdır. Start ile START birbirinin aynısı değildir. Herhangi bir karışıklığa neden olmaması için hep büyük veya hep küçük harf kullanmak en iyi seçimdir.

PIC ASSEMBLY KOMUTLARININ YAZILIŞ BIÇİMİ

PIC16F84'ün toplam 35 tane komutu vardır. Bu komutların yazılış biçimini üç grupta toplayabiliriz.

1. Byte-yönlendirmeli komutlar.
2. Bit-yönlendirmeli komutlar.
3. Sabit işleyen komutlar.
4. Kontrol komutları.

Komutların yazılış biçimlerini açıklarken bazı tanımlama harfleri kullanacağız. Önce bu harflerin anlamlarını verelim:

f = File register

d = destination (gönderilen yer)

d=0 → W register

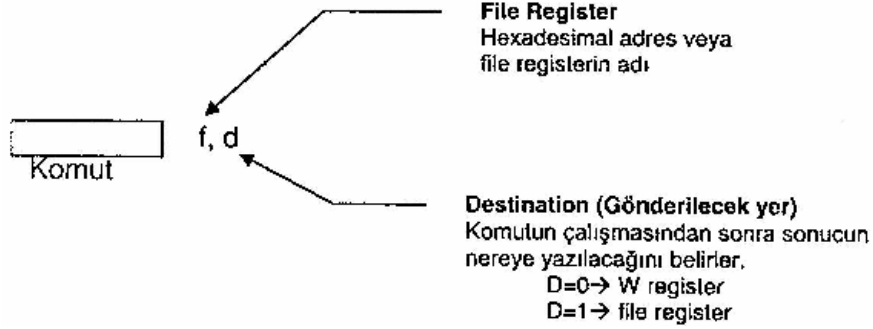
d=1 → file register

k = Sabit veya adres etiketi b = Bit tanımlayıcı

b = Binary sayıları belirleyen harf (örneğin b'00001111' gibi)

d = Desimal sayıları belirleyen harf (Örneğin d'16' gibi)

Byte-Yönlendirmeli Komutlar



Örnek:

MOVF 0x03, 0	;0x03 adresindeki file registerin içeriğini W registeri içerisine kopyalanır.
MOVF STATUS, 0	;STATUS registerin içeriği W registere kopyalanır.
MOVF STATUS, 1	;STATUS registerinin içeriği yine kendi içine yazılır.

NOT: Byte-yönlendirmeli komutlarda destination (gönderilecek" yer) belirleyen d'nin yazıldığı yere 0 veya 1 yazmak hatırlatıcı olmayabilir. MPASM bunu dikkate alarak 0 yerine w, 1 yerine f yazmaya izin verir. MPASM'nin MS-DOS versiyonunda ise w ve f harflerinin otomatik olarak kullanılmasına izin verilmez. Bu durumda her programın tanımlama bölümünde aşağıdaki eşitlikler yazılmalıdır.

W EQU 0

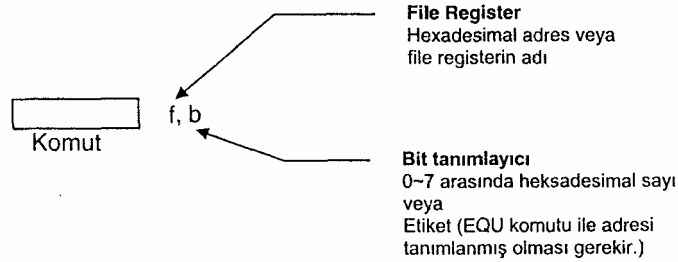
F EQU 1

Bu eşitliklerden sonra komutlarda destination belirlemek için w ve f harfleri kullanılabilir. Örneğin:

INCF, w ; SAY registerinin içeriği 1 arttırıldıktan sonra sonuç W registerine yazılır.(W=SAY+1)

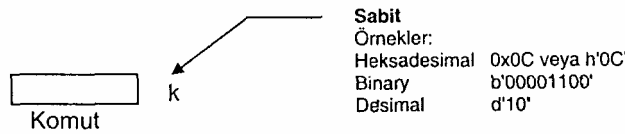
INCF, f ; SAY registerinin içeriği 1 arttırıldıktan sonra sonuç yine SAY registeri içine yazılır.
(SAY=SAY+1)

Bit-Yönlendirmeli Komutlar



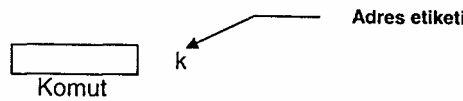
Örnek: BCF 0x03, 5 ; 0x03 adresindeki registerin 5. bitini sıfırla
BSF STATUS, BESBIT ; STATUS registerinin BESBIT etiketiyle tanımlı olan bitini "1" yapar. (Tanımlama bloğunda BESBIT EQU 5 yazılması gerekir.)

Sabit İşleyen Komutlar



Örnek: MOVLW 0x2F ; W registerine 2F heksadesimal sayısını yükler.
ADDLW b'00101111' ; W registeri içerisindeki sayıya 00101111 binary sayısını ekler.

Kontrol Komutları



Örnek: GOTO DONGU ; Program akışı DONGU olarak belirlenen etikete dallanır.
CALL TIMER ; Program akışı TIMER etiketi ile belirlenen adresteki alt programa dallanır.

NOT: Program içerisinde yazılan etiketlere assembler'in otomatik olarak adres verdiğini unutmayın.

SAYI VE KARAKTERLERİN YAZILIŞ BİÇİMİ

PIC assembly komutlarında sayılar heksadesimal, binary veya desimal formda kullanılabilir. Değişik kaynaklarda kullanılan sayı ve karakter gösteriliş biçimleriyle karşılaştığınızda bunları okuyabilmeniz için aşağıda örnekler verilmiştir.

Heksadesimal sayılar

Heksadesimal sayılar "0x", "0" veya "h" harfleriyle başlamalıdır. Örneğin, STATUS registerine 03 adresini atamak için aşağıda gösterilen yazılış biçimleri kullanılabilir.

STATUS	EQU	0X03	
	EQU	3	
	EQU	03	
	EQU	03h	
	EQU	h'03'	(Bu kitapta örneklerde kullanılacak biçim)

MOVLW komutu ile W registeri içerisine yüklenecek olan FF heksadesimal sabitler ise aşağıdaki gibi yazılabilir.

MOVLW	0xFF
	h'FF'

Eğer FF heksadesimal sayısını aşağıdaki gibi kullanmaya kalkarsanız çalışmaz

MOVLW	FP
	FFh

Çünkü kural olarak heksadesimal sayılar muhakkak "0" veya "h" harfi ile başlamalıdır.

Binary Sayılar

Binary sayılar b harfi ile başlamalıdır. Örneğin 00001010 binary sayısını W registeri içerisine yüklemek için aşağıdaki gibi yazılmalıdır.

MOVLW b'00001010'

ANDLM b'00001111'

Desimal sayılar

Desimal sayıların başına d harfi konularak tırnak içerisinde yazılırlar. Örneğin 15 desimal sayısı W registeri içerisine yüklemek için aşağıdaki gibi yazılmalıdır.

MOVLW d'15'

MOVLW d'255'

ASCII Karakterler

Genellikle RETLVV komutu ile birlikte kullanılan ASCII karakterler tırnak içerisine alınarak aşağıdaki gibi yazılırlar.

RETLW 'A'

RETLW 'T'

PIC ASSEMBLY KOMUTLARI

Yer Değiştirme veya Yükleme Komutları			
Komut ve Örnek		İngilizce tanımı	Türkçe açıklaması
MOVLW k		Move Literal to W	K sabit değerini W registerine yükler.
MOVLW h '0F'			W←-OP
MOVF f,d		Move f	f registerinin içeriğini W veya f'e yükler.
MOVF TEST,0			d=0 W←-TEST d=1 TEST←-TEST
MOVWF f		Move W to f	W registerin içeriğini f registerine yükler.
MOVWF PORTA			PORTA←-W
Register İçeriğini Değiştirme Komutları			
CLRF	F	Clear f	f registerinin içeriğini siler (sıfırlar).
CLRF	TRISA		TRISA←-00000000
CLRWF		Clear W	W registerin içeriğini siler (sıfırlar).
CLRWF			W←-00000000
COMF	F,d	Complement f	F registerinin içindeki sayı terslenir. Yani tüm 1'ler 0, 0'lar 1 olur. Sonuç W veya f registerine yüklenir.
COMF	SAY, 0		SAY= 0011011 ise, d=0 W←-11001010 d=1 olsaydı SAY←-11001010
DECF	f,d	Decrement f	F registerinin içerisindeki sayıyı "1" eksiltir. Registerin içeriği h'00' ise, "1" eksiltildiğinde h'FF' olur. Sonuç W veya f registerine yazılır.
DECF	GIT,1		GIT=h' 2C" ise 2C-1=2B d=1 GIT←-2B d=0 olsaydı W←-2B
INCF	f,d	Increment f	F registerinin içerisindeki sayıyı "1" artırır. Registerin içeriği h' FF' ise, "1" artırıldığında h' 00' olur. Sonuç W

			veya f registerine yazılır.
INCF	GIT, 0		GIT=h' 2C' ise 2C+1=2D d=0 W←2D d=1 olsaydı GIT←2D
BCF	f,b	Bit Clear f	f registerinin içerisindeki sayının b.ninci bitini sıfırlar.
BCF	PORTB,5		PORTB = b'11111111' ise, PORTB←b' 11011111'
BSF	f,b	Bit Set f	F registerinin içerisinde sayının b.ninci bitini 1 yapar.
BSF	PORTA,3		PORTA = b' 00000000' ise, PORTA←b' 00001000'
RLF	f,d	Rotate Left f	f registeri içerisindeki sayıyı bir pozisyon sola kaydırır. Registerden taşarak Carry bayrağına yazılan bit, LSB'ye yazılır. Sonuç W veya f registerine yüklenir.
RLF	KAY,0		
RRF	F,d	Rotate Right f	F registeri içerisindeki sayıyı bir pozisyon sağa kaydırır. Registerden taşarak Carry bayrağına yazılan bit, MSB'ye yazılır. Sonuç W veya f registerine yüklenir.
RKF	KAY,1		
SWAPF	f,d	Swap nibbles inf	f registerinin içerisindeki ilk dört bit ile son dört biti yer değiştirir. Sonuç W veya f registerine yüklenir.
SWAPF	DEG,1		DEG = b' 00101111' ise, d=1 olduğundan DEG←11110010 d=0 olsaydı W←11110010'
Program Akışını Kontrol Etme Komutları			
GOTO	k	Go to address	Program akışı k adresine dallanır.
GOTO	DOMGU		Program, DÖNGÜ etiketinin yazıldığı yere dallanır ve buradan itibaren devam eder.
CALL	k	Call subroutine	Program akışı k etiketinin bulunduğu yerdeki alt programa dallanır.
CALL	TIMER		Program TIMER etiketinin yazıldığı alt program satırlarının başlangıcına dallanır ve buradan itibaren devam eder.
KETÜRN		Return from subrouitine	Alt program komutlarının en sonuna yazılan bu komut, program akışını ana programa geri döndürür.
RETLW		Return with Literal in W	Program akışını alt programdan ana programa döndürür ve W registerine k sabitini yükler.
RETLW	H'2F'		Alt programdan ana programa döndürür ve W registerine 2F yüklenir.
RETFIE		Return From Interrupt	Program akışını interrupt alt programından ana programa döndürür.
BTFSC	f,b	Bit Test F, Skip if Clear	F registerinin b.inci bit'ini test eder. Eğer bu bit "0"sa program akışı bir sonraki komuta geçer.
BTFSC	PORTA,2		

BTFS	f,b	Bit Test F, Skip if Set	F registerinin b.inci bit'ini test eder. Eğer bu bit "1"se program akışı bir sonraki komuta geçer.
BTFS	PORTA,0		
DECFSZ	f,d	Decrement f, Skip if Zero	F registerinin içeriğini "1" azaltır. Register içeriği 0'sa bir sonraki komuta atlar. Sonuç W veya f registere yazılır.
DECFSZ	SAYAC,!		SAYAC=h' 2F' ise 2F-1=2E d=0 olsaydı W←h' 2E' d=1 olduğundan SAYAC←h'2E'
INCFSZ	F,d	Increment f, Skip if Zero	F registerinin içeriğini "1" artırır. register içeriği "0"sa bir sonraki komuta atlar. Sonuç W veya f registere yazılır.
INCFSZ	SAYAC, 1		SAYAC=h' 2F' ise 2F+1=30 d=1 SAYAC←h' 30' d=0 W←h'30"

Mikrodenetleyici Kontrol Komutları

CLRWDT		Clear VWatchdog Timer	Watchdog timer'ı sıfırlar. Ayrıca vwatchdog timer'ın prescaler değerini de sıfırlar. Status bitlerinden TO ve PD'yi "1"yapar.
SLEEP		Go into standby mode	Mikrodenetleyiciyi uyuma moduna geçirerek güç harcamasını azaltır. Mikrodenetleyici uyuma modundan reset, watchdog timer ve TOCKI girişi vasıtasıyla çıkar.

Mantıksal Komutlar

ANDLW	K	AND Literal with W	W registerin içeriği ile k sabitine AND işlemini uygular. Sonuç W registerine yazılır.
ANDLW	b' 0110001'		W = b' 10011101' ise, b' 00110001' sabitin değeri b' 00010001' AND işlemi sonucu W←b' 00010001'
ANDWF	f,d	AND W with f	W registeri ile file register içeriğine AND işlemini uygular. Sonuç W veya f registerine yazılır.
ANDWF	TEST,1		W=b' 11111111' ise, TEST=b' 11011110" ise, b' 11011110' AND işlemi sonucu d=0 ise W←b' 11011110' d=1 olduğundan TEST←b' 11011110 '
IORLW	k	Inclusive OR Literal with W	W registerin içeriği ile k sabitine OR işlemini uygular. Sonuç W registerine yazılır.
IORLW	B-00101000-		W =b' 10000100' ise b' 00101000' sabitin değeri b' 10101100' OR sonucu , W ←b' 10101100'
IORWF	f,d		W registeri içeriği ile file registerin içeriğine OR işlemini uygular. Sonuç W veya f registerine yazılır.

IORWF	TEST,1		<p>W = b' 10000100" ise, <u>TEST = b' 00101000'</u> ise, b' 10101100 ' OR sonucu d=0 ise W←b' 10101100" d=1 olursa TEST←b' 0101100 '</p>
XORLW	k	Exclusive OR Literal withW	W registerin içeriği ile k sabitine XOR işlemini uygular. Sonuç W registerine yazılır.
XORLW	b' 00101000'		<p>W = b ' 00000000 ' ise <u>b' 00101000'</u> sabitin değeri b' 11010111" XOR sonucu W←b' 11010111'</p>
XORWF<"	P.S- • .. • . -	Exclusive OR W with f	W register ile file register içeriğine XOR işlemini uygular. Sonuç W veya f registerine yazılır.
XORWF	TEST,1		<p>W = b' 00000000' ise, <u>TEST = b' 00101000'</u> ise, b' 11010111" XOR sonucu d=0 ise W←b' 11010111" d=1 olursa TEST←b' 11010111'</p>

Aritmetik İşlem Komutları

ADDWF	f,d	Add W with f	W registerinin içeriğini f registeri ile toplar. Sonuç W veya f registerine yazılır.
ADDWF	TOPLA,0		<p>W=h' 2A' ise, TOPLA=h ' 31' ise, h' 2A'+h' 31'=b' 5B' d=1 ise TOPLA←h' 5B' d=0 olduğundan W←h' 5B'</p>
ADDLW	k	Add Literal andW	W registerinin içeriğini k sabit değeri ile toplar. Sonuç W registerine yazılır.
ADDLW	H' 2F'		<p>W=h' B0' ise, h' B0'+h' 2F'=h' DF' W←h' DF'</p>
SUBLN	k	Subtract W from Literal	K sabit değerinden W registeri içeriğini çıkarır. Sonuç W registerine yazılır.
SUBLW	H' 90'		<p>W=h' 83' ise h' 90'-h' 83' = h' 07' W←h' 07'</p>
SUBWF	f,d	Subtract W from file register	f registerinin içeriğinden W registerinin içeriğini çıkarır. Sonuç W veya f registerine yazılır.
SUBWF	CIK,1		<p>W=h' 83' ise, CIK=h' 90' ise, h' 90'-h' 83'=h' 07' d=0 ise W←h' 07' d=1 olduğundan CIK←h'07'</p>

İşlem Yapmayan Komut

NOP		No Operation	Bir komut saykılı süresince hiçbir işlem yapmayan bir komuttur. Bir dahili komut süresinde çalışır. Bu nedenle zaman geciktirme işlemlerinde kullanılır.
-----	--	--------------	--

5

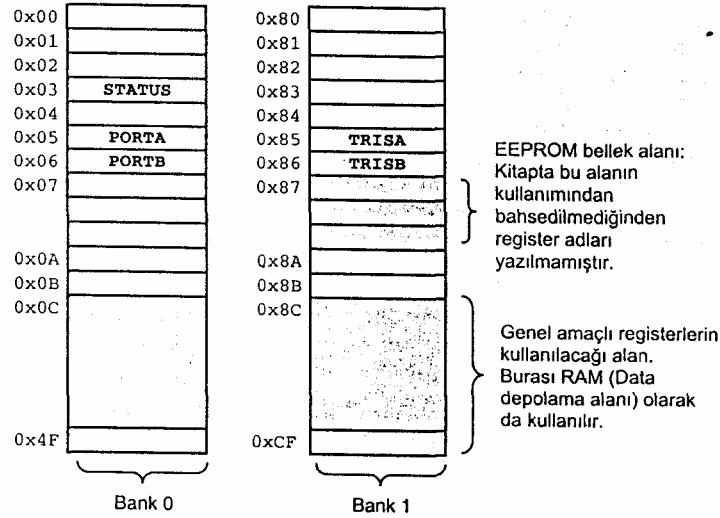
PIC PROGRAMLAMA

- İLK PROGRAMINIZ
- AKIŞ DİYAGRAMI SEMBOLLERİ
- AKIŞ DİYAGRAMININ ÇİZİLMESİ
- ASSEMBLY PROGRAM KOMUTLARININ YAZILMASI
- PROGRAMLARIN DERLENMESİ (MPASM)
- PROGRAMIN PIC'E YAZDIRILMASI
- PROGRAMLANMIŞ PIC'İN DENENMESİ
- MPASM'NİN ÜRETTİĞİ DİĞER DOSYALAR
- INCLUDE DOSYALARI
- KONFIGÜRASYON BİTLERİNİN YAZILMASI

İLK PROGRAMINIZ

Bu ünite, basit bir programı ele alıp, bunun akış diyagramının nasıl çizildiğini, program komutlarının nasıl yazılarak derlendiğini daha sonra da bu programın PIC'e nasıl yazıldığını göstereceğiz.

Program, PIC'e enerji verince portB'nin 0. bit'ine bağlı olan bir LED'i direkt olarak yakacaktır. Programa başlamadan önce kullanacağımız özel file registerlerin adreslerini vereceğiz. Daha sonra da STATUS registerini kullanarak bank değiştirme işleminin nasıl yapıldığını göreceğiz.



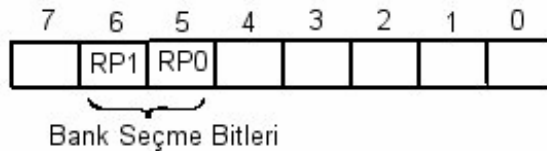
Yukarıda görülen RAM haritasında sadece bu programda kullanılacak olan özel registerler gösterilmiştir. Diğer registerlerin ne işe yaradığı ileriki programlarda sırası geldiğinde verilecektir.

3. Bölümde PIC16F84'ün RAM'inden bahsedilirken RAM'in iki bölümden (Bank0, Bank1) meydana geldiğini söylemiştik. RAM içerisindeki bir özel registerin kullanılabilmesi için, o registerin bulunduğu bank'a geçmek gerektiğinden de bahsetmiştik. Şimdi bank değiştirmenin nasıl yapıldığını görelim.

Bank Değiştirme

Bir bank'tan diğer bank'a geçmek için STATUS register denilen özel registerin 5. ve 6. bit'inin durumunu değiştirmek gerekir.

STATUS REGİSTER

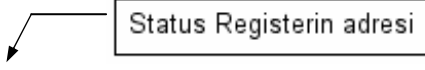


- 00 → Bank 0
- 01 → Bank 1
- 10 → Bank 2
- 11 → Bank 3

PIC16F84'ün sadece 2 bank'ı bulunduğundan, sadece 5. bit'in değerini değiştirmek yeterlidir. 6. bit'in değeri daima "0" olmalıdır. Zaten PIC'e enerji verildiğinde (Power-on reset) 5. ve 6. bit'in değeri "0"dır. Bu bit'ler aynı zamanda diğer reset girişleri (MCLR ucundan yapılan harici reset ve Watchdog timer reset) yapıldığında da "0" olur. Yani PIC'i çalıştırmak için

enerji verildiğinde direkt olarak bank0 seçilmiş olur. Bu durumda bank değiştirme işlemine gerek duyulmaksızın bank0'daki registerler kullanılabilir.

STATUS registerinin 5. bit'ini "1" yapmak için BSF komutu, "0" yapmak için de BCF komutu kullanılır. Aşağıda bank değiştirmek için kullanılan komutlar görülmektedir.


BSF h'03', 5 → Bank1 seçilir.
BCF h'03', 5 → Bank0 seçilir.

Port'ların Giriş veya Çıkış Olarak Yönlendirilmesi

PortA ve PortB'nin uçlarına bağlı bulunan bir I/O elemanını kullanabilmek için portları giriş veya çıkış olarak yönlendirmek gerekir.

PortA'yı →TRISA registeri, PortB'yi →TRISB registeri yönlendirir.

PortA/PortB'nin hangi bit'i giriş yapılmak isteniyorsa, TRISA/TRISB içerisinde o bit'e karşılık gelen bit "1" yapılır. Çıkış olarak yönlendirilmek istenen bit'ler için de TRISA/TRISB içerisine "0" yazılır.

TRISA 1 → PortA → Giriş

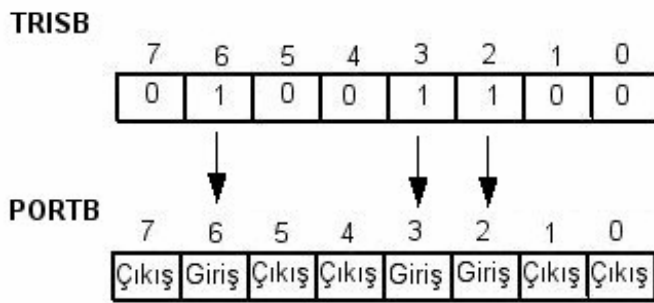
TRISB 0 → PortB → Çıkış

Bu söylediklerimizi bir örnek vererek şematik olarak gösterelim. Örneğin, PortA'nın 1. ve 2. bitleri giriş, diğer bitlerim çıkış olarak yönlendirelim.



PortA'nın sadece 5 ucu bulunduğundan, TRISA registerinin ilk 5 biti içerisine yazılan veriler PORTA'yı yönlendirir. Diğer bit'lerin 0 veya 1 olmasının hiçbir önemi yoktur.

PortB'nin 2, 3, 6. bitlerini giriş diğerlerini çıkış olarak yönlendirmek için de aşağıdaki konfigürasyon yapılır.

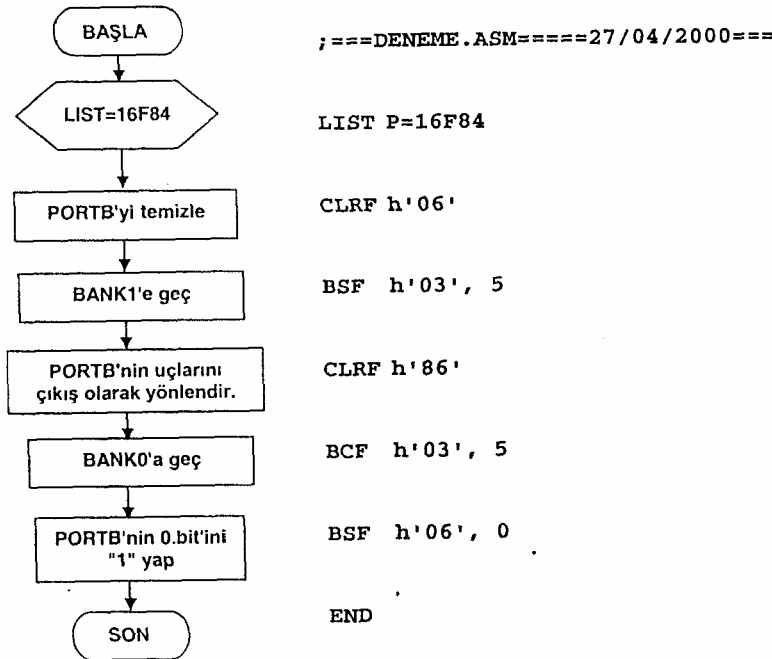
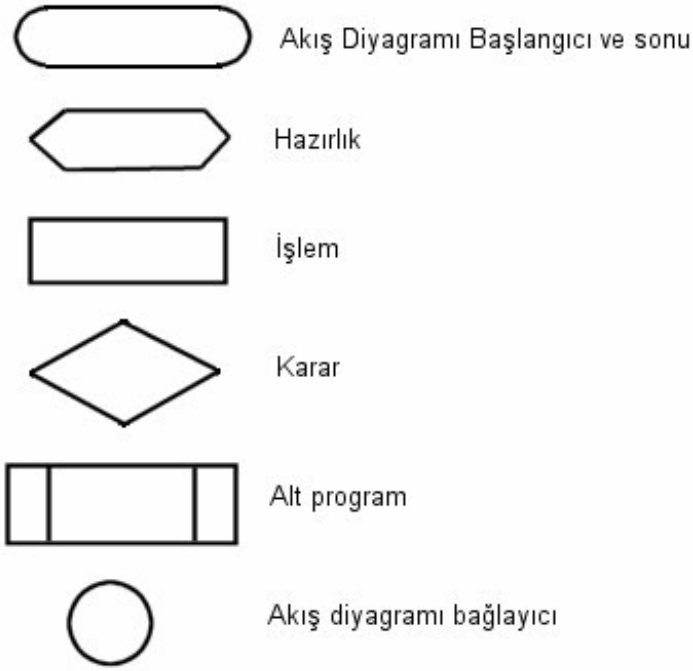


AKIŞ DİYAGRAMI SEMBOLLERİ

Tüm programlama dillerinde olduğu gibi assembly dili programlarını yazmadan önce akış diyagramı çizmek iyi bir alışkanlıktır. Kısa

programlarda, genellikle karar işlemlerinin olmadığı programlarda çoğu zaman akış programı çizmeye gerek duyulmaz. Ancak uzun ve karmaşık mantık işlemlerinin yoğun olduğu programları yazarken direkt olarak komutları yazmaya başlamak kısa bir süre sonra içinden çıkılmaz hale getirebilir. Bu nedenle programları yazmadan önce akış diyagramını çizerek, komutların hangi sıraya göre yazılacağını görmek için görsel bir düşünme ortamı yaratılmış olur.

Programlama dillerinde ortak olarak kullanılan akış sembolleri vardır. Bu sembollerin neler olduğunu ve nerelerde kullanılacağını aşağıda göreceksiniz.



AKIŞ DİYAGRAMININ ÇİZİLMESİ

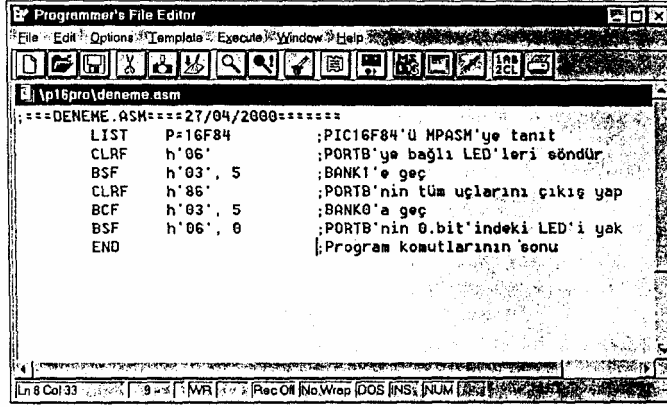
İlk programımız, PIC'e enerji verdiğimizde PORTB'nin 0. bit'ine bağlı LED'i otomatik olarak yakacaktır. Buna göre akış diyagramını çizelim.

Akış diyagramları çizilirken semboller içerisine yazılan açıklamalar çoğu zaman yukarıda olduğu gibi çok fazla ayrıntılı olması gerekmez. Yapılacak işlemler yerine hangi komut kullanılacaksa sembol içerisine o komut direkt olarak da yazılabilir. Akış diyagramı çizmenin amacı, karmaşık mantıksal işlemler gerektiren programları yazarken düşünme kolaylığı sağlamaktır. Bu nedenle akış diyagramları çizirken kesin kurallara

uymaya özen göstermeniz gerekmez. Semboller içerisine yazacağınız hatırlatıcı birkaç kelime ya da işaret çoğu zaman yeterli olabilir.

ASSEMBLY PROGRAM KOMUTLARININ YAZILMASI

Akış diyagramı çizildikten sonra, işlemleri gerçekleştirecek olan assembly komutları yazılır. Assembly komutları ASCII kodunda dosya üreten bir editörde hazırlanması gerekir. Bu editörler, DOS altında çalışan EDIT, VVINDOVVS altında çalışan NOTPAD olabilir. Kullanmaya alışık olduğunuz herhangi bir editör de bu iş için uygundur. Biz bu kitapta PFE adlı bir editörü kullanacağız. Aşağıda assembly komutlarının editörde yazılış biçimi görülmektedir.



```
=====DENEME.ASM=====27/04/2000=====
```

```
LIST P=16F84      ;PIC16F84'ü MPASM'ye tanı
CLRF h'06'        ;PORTB'ye bağlı LED'leri söndür
BSF h'03', 5      ;BANK1'e geç
CLRF h'86'        ;PORTB'nin tüm uçlarını çıkış yap
BCF h'03', 5      ;BANK0'a geç
BSF h'06', 0      ;PORTB'nin 0.bit'indeki LED'i yak
END               ;Program komutlarının sonu
```

ilk programımızın komutlarını yazarken daha sonraki çalışmalarınızda hatırlatıcı olması için yukarıda görüldüğü gibi yanlarına açıklamalar yazabilirsiniz.

Şimdi bu programı MPASM'yi kullanarak derlemeden önce, aynı programı daha anlaşılır ve daha kısa yazılış biçimlerine bir göz atalım.

Dikkat ederseniz komutları yazarken kullandığımız register ve portların RAM bellekteki adreslerini belirttik. Örneğin;

```
CLRF h'06"
BSF h'03', 5
```

PORTB'nin RAM bellekteki adresi

STATUS registerin RAM bellekteki adresi

Registerlerin RAM'daki adreslerini kullanarak komutlar yazıldığında, komutun hangi registeri kullandığını ilk bakışta anlamak zordur. Bu durumda tüm registerlerin adreslerini ezbere bilmek zorunluluğunu getirmektedir. Halbuki atama deyimi (EQU) kullanarak registerleri önceden tanıtmak daha anlaşılır komutlar yazmayı sağlayacaktır.

Atama (EQU) Komutu Kullanarak Program Yazmak

Atama komutları yazılırken register isimlerinin etiket sütununda yazılması gerektiğini 3. bölümdeki "Assembly Dili Yazım Kuralları" başlığı altında bahsetmiştik. Şimdi programı EQU komutunu kullanarak register adlarını etiket sütununa, adreslerini de adres sütununa yazarak programı yeniden düzenleyelim.

- Programı editörünüzde yazdıktan sonra DENEME.ASM adıyla kaydediniz.

NOT: Hazırladığınız DENEME.ASM kaynak dosyasını hangi klasöre kaydedeceğiniz bazen önemli olabilir. örneğin, bizim bu kitaptaki programları PIC'e yazdırmak için kullandığımız P16PRO programı **".HEX"** uzantılı programları sadece belirli bir klasörden yükleyebilir. Bu klasör de P16PRO.EXE programının bulunduğu klasördür. MPASM ise kaynak dosyayı hangi klasörden aldıysa, oluşturacağı (.HEX) uzantılı dosyayı da aynı klasöre kaydeder. Bu durumda editörünüzde oluşturduğunuz ".ASM" uzantılı dosyayı kaydedeceğiniz klasör önemli olmaktadır. Eğer siz başka bir PIC programlayıcı program kullanıyorsanız yukarıda anlattığımız durum geçerli olmayabilir. Bu nedenle kullandığınız programların özelliklerini iyi öğrenmeniz gerekir.

```
=====DENEME.ASM=====27/04/2000=====
```

```
LIST P=16F84      ;PIC16F84'ün MPASM'ye tanı
PORTB EQU h'06'
```

```

STATUSEQU    h'03'
TRISB EQU    h'86"
    CLRF PORTB ;PORTB'ye bağı LED'leri söndür
    BSF STATUS, 5 ;BANK1'e geç
    CLRF TRISB ;PORTB'nin uçlarını. çıkış yap
    BCF STATUS, 5 ;BANK0'a geç
    BSF PORTB, 0 , ;PORTB'nin 0.bitindeki LED'i yak
    END ;Program komutlarının sonu

```

Görüldüğü gibi komutların hangi registeri kullandığı kolayca anlaşılmaktadır örneğin;

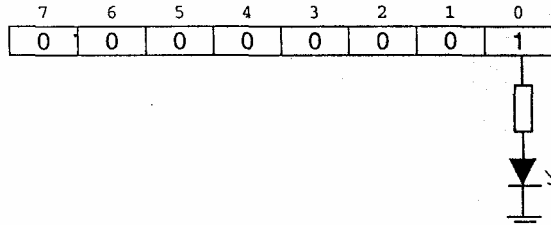
BSF STATUS, 5 BANK1'e geçmek için STATUS registerin 5. bitini "1" yap

STATUS

7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0

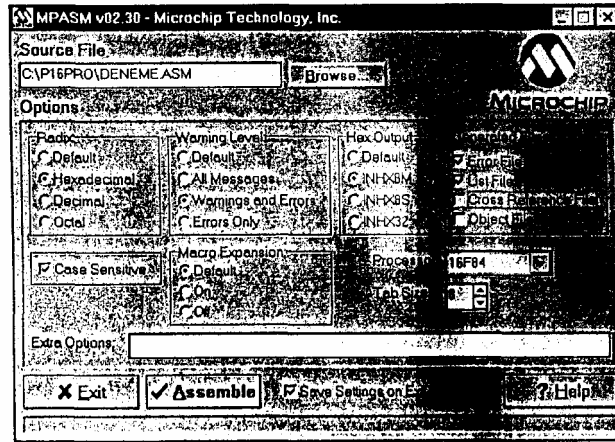
BSF PORTB,0 ;PORTB'nin 0. ucuna bağı LED'i ;yakmak için 0. biti "1" yap.

PORTB



PROGRAMLARIN DERLENMESİ (MPASM)

DENEME.ASM adıyla kaydedilen kaynak programı derlemek için MPAS^ derleyici programını çalıştırmak gerekir. Bu programı çalıştırınca karşınıza aşağıdaki ekran gelecektir.



MPASM derleyici programı kullanmak oldukça kolaydır. Yukarıdaki ekran üzerinde görülen çoğu seçenek: kaynak programın derlenmesinde önemli bir rol oynamaz. Ancak bizim kitapta açıklayacağımız bazı önemli dosyaların (.ERR ve .LST) üretilmesi için aşağıda açıklana¹¹ düzenlemeyi yapmanızda yarar vardır.

- Radix bölümünde "**Hexadecimal**"i seçiniz.
- Warning bölümünde "**Warnings and Errors**" seçiniz.
- Hex output bölümünde "**INHX8M**"yi seçiniz.
- Generated Files bölümünde "Error File", "List File"i seçiniz.
- Case sensitive kutusunu onaylayınız.
- Macro Expansions bölümünde "**Default**"u seçiniz.

- Processor penceresinden "**16F84**" seçiniz.
- Tab size penceresine "**8**" yazınız.

Derlenecek olan programı seçmek için:

- "**Browse**" butonuna tıklayınız. Açılan pencereden **DENEME.ASM** dosyasını işaretleyip "**Tamam**" butonuna basınız. Şimdi "Source File Name" penceresinde derleyeceğiniz programın" adını göreceksiniz.
- Programı derlemek için "**Assembler**" butonuna tıklayınız,

Programın başarıyla derlendiği "**Assembly Successful**" mesajıyla" anlaşılır. Aynı zamanda derleme başarılı olduğunda 100% yazan pencerenin yeşil renge dönüşmesinden anlaşılır.

NOT: Kaynak programda(DE^{AME}.ASM) hatalar varsa, pencere içerisindeki mesaj "Errors Found" biçiminde olacaktır. Pencerenin ortasındaki bandın rengi ise kırmızıya boyanacaktır Eğer böylesi bir durumla karşılaşırsanız hatanın nedenlerini bu aşamada araştırmanıza gerek yoktur. Tek yapacağınız şey editörü yeniden çalıştırıp programı iyice inceleyip kitaptakinden farklı yazdığınız yerleri düzeltmenizdir. Elbette ki programı kaydedip yeniden derlemeniz gerekecektir. Hataların nedenini ve nasıl düzeltileceğini ilgili açıklamalar ileride verilecektir.

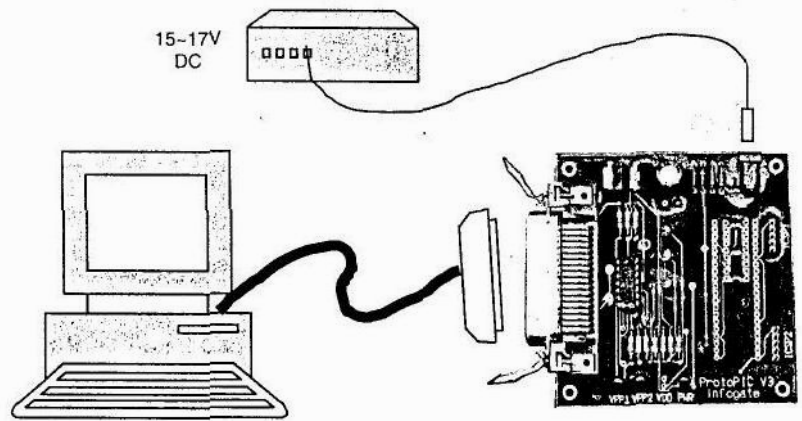
- Assembly programını derlediğinizde **DENEME.HEX** adında hexadesimal kodlara dönüştürülmüş olan bir dosya otomatik olarak kaydedilecektir. Bu dosya, kaynak dosyanın(.ASM uzantılı) olduğu klasörde bulunacaktır.

PROGRAMIN PIC'E YAZDIRILMASI

Hexadesimal kodlara dönüştürülmüş olan assembly programı PIC'e yazdırmak için iki şeye daha ihtiyacımız var. Bunlar:

1. Bilgisayarın paralel veya seri portuna bağlanan **PIC programlama seti**,
2. .HEX uzantılı dosyadaki program kodlarını PIC programlama setine gönderen **PIC programlayıcı yazılımı**.

PIC'leri programlamak için kullanılan çok değişik elektronik devreler kullanmak mümkündür. Bu devrelerin bazıları bilgisayarın seri portuna, bazılarıda paralel portuna bağlanmaktadır. Bizim bu kitaptaki programları PIC'e yazdırmakta kullandığımız set 50'den fazla PIC'i programlayabilen ve adına ProtoPIC denilen elektronik devredir. ProtoPIC, bilgisayarın paralel portuna bağlanmaktadır. Bu devrenin kullanımı ve elde edilmesiyle ilgili detaylı bilgi kitabın ekler bölümünde verilmiştir. Programınızı PIC'e yazdırmaya başlamadan önce lütfen bu bölümü dikkatlice okuyunuz. Aşağıda ProtoPIC setinin bilgisayara bağlantısı görülmektedir.



HEX uzantılı dosyadaki program kodlarını PIC programlama setine paralel veya seri port aracılığıyla PIC'e gönderen çeşitli yazılımlar bulunmaktadır. Ancak her yazılım elinizde bulunan programlama setine uygun olmayabilir. Bu nedenle hangi programlama setini kullanıyorsanız o sete uygun yazılımı kullanmanız gerekir. ProtoPIC programlama seti için hazırlanmış olan ve adına P16PRO denilen yazılım bulunmaktadır. Biz bu kitaptaki programları PIC'e yazdırmak için P16PRO'yu kullandık.

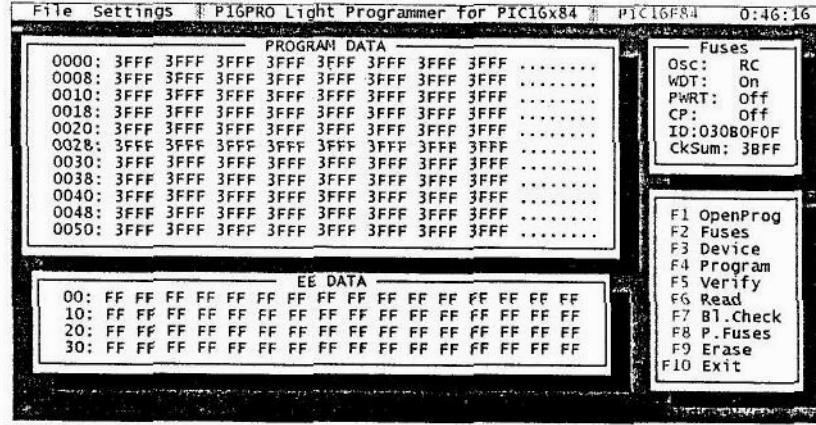
P16PRO'nun Başlatılması

P16PRO bir DOS programıdır. Ancak bu programı Windows altından da çalıştırabilirsiniz. Eğer bilgisayarınız her açılışta Windows 95/98 ile başlıyorsa programı DOS ortamından çalıştırmak için;

- Başlat menüsünden "MS-DOS Komut İstemi"ni seçerek DOS ortamına dönünüz. Daha sonra C:\P16PRO klasörüne geçip, P16PRO yazıp "Enter" tuşuna basınız.

P16PRO'yu çalıştırmak için her defasında DOS ortamına dönmek zor geliyorsa, programınızı en kolay olarak masa üstünden çalıştırmaktır. Bunun için;

- Masa üstündeyken mouse'un sağ tuşuna tıklayınız. Açılan pencereden "Yeni"yi daha sonra da "**Kısayol**"u seçiniz. Kısayol oluştur penceresi açıldığında "**Gözet**" butonuna tıklayarak hard diskinizdeki P16PRO.EXE programını bulup, bu programa uygun bir simge seçerek masaüstüne yerleştiriniz. Bundan sonra simge üzerine tıkladığınızda aşağıdaki program penceresi ekrana gelecektir.



PIC Seçme (F3)

Programlanacak PIC'in seçimini yapmak için:

- F3** tuşuna basın ve "Device" penceresinde programlayacağınız P16F84'e karşılık gelen "1" rakamını yazıp, "**Enter**" tuşuna basın. Yeniden ekrana döndüğünüzde ekranın sağ üst köşesinde seçtiğiniz PIC'i göreceksiniz. Aynı işlemi Alt+S tuşlarına basarak açılan pull-down menüden "Device" komutunu seçerek de yapabilirsiniz.

Program Dosyasını Açma (F1)

P16PRO klasörü içerisine kaydettiğiniz ".HEX" uzantılı dosyayı açmak için;

- F1** tuşuna basın. "Open Program File" penceresinden PIC'e yazdırmak istediğiniz **DENEME.HEX** dosyasını yukarı aşağı tuşlarını kullanarak seçin ve "**Enter**" tuşuna basın.

Ana ekrana döndüğünüzde ekrandaki "PROGRAM DATA" yerinde seçtiğiniz programın dosyasının adını (DENEME.HEX) göreceksiniz. Yine bu pencere içerisinde Heksadesimal kodlara dönüştürülmüş assembly komutlarını göreceksiniz.

Programın yüklenmesi esnasında ekranın ortasındaki küçük bir pencere oluşacaktır.

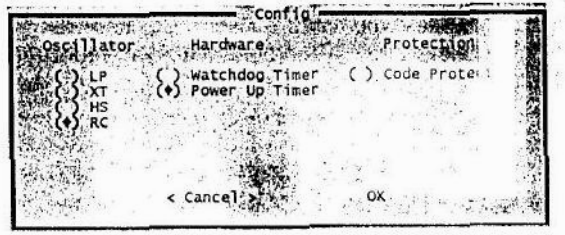
Bu pencerede, (.HEX) uzantılı program içerisinde konfigürasyon bilgilerinin yer almadığı yönünde bir uyarı bulunmaktadır. Konfigürasyon bilgilerinin program içerisinde yer almasının önemi yoktur. Çünkü aynı konfigürasyon bilgilerini ekranın sağ üst köşesinde bulunan "Fuses" penceresinde görülen değerlere sonradan ayarlamak mümkündür.

NOT: Konfigürasyon bilgilerinin nasıl yazıldığını ileride detaylı olarak inceleyeceğiz.

PIC Konfigürasyonunu Ayarlama (F2)

Eğer .HEX dosyanızda PIC konfigürasyonu yer almıyorsa veya yer alan konfigürasyonu değiştirmek istiyorsanız, bunu F2 tuşunu kullanarak yapabilirsiniz.

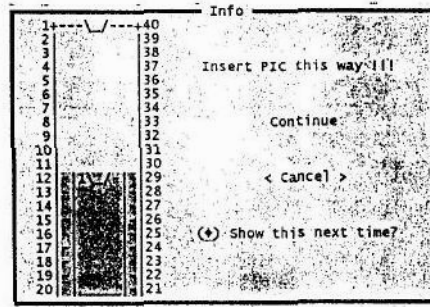
- F2** tuşuna basın ve "Config" penceresindeki sekmeyi TAB ve boşluk bırakma SPACE BAR kullanarak konfigürasyonu seçip "**Enter**" tuşuna basın. Yeniden ekrana döndüğünüzde ekranın sağ üst köşesinde seçtiğiniz konfigürasyonu "Fuses" penceresinde göreceksiniz.
- Şimdi, bu işlemleri yaparak pencerede aşağıdaki konfigürasyonun görülmesini sağlayınız.



Programı PIC'e Yazdırma (F4)

Artık PIC'i programlama aşamasına geldiniz. ProtoPIC programlama setinizin bilgisayara bağlı olduğundan emin olduktan sonra;

- F4 tuşuna basın.
- "Info" penceresinde gösterildiği gibi PIC'i sokete yerleştirin.



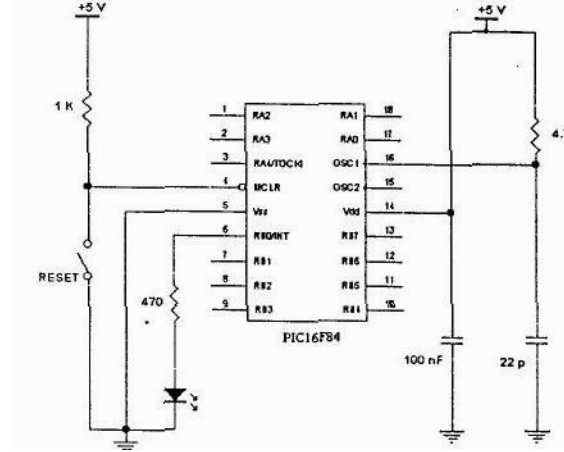
- PIC'i yerleştirdikten sonra programlamaya başlamak için **"Enter"** tuşuna basın. Yeni açılan "Info" penceresinde sırasıyla "Programning", "Verifying", "Blowing fuses", "Verifying fuses" ve "Total time....sec" mesajlarını göreceksiniz. Bu esnada PIC programlama setinin yeşil ışığı yanıp sönerek programlama işleminin yapıldığını gösterecektir.
- Programlama işlemi bittiğinde **"Enter"** tuşuna basarak programlama modundan çıkınız.

PROGRAMLANMIŞ PIC'İN DENENMESİ

Programladığınız PIC'leri deneyip, çalışıp çalışmadığını görmek için iki farklı yöntemi uygulayabilirsiniz. Bunlardan ilki, devreyi Breadboard üzerinde kurmaktır. Bu yöntemle uygulama devresi üzerinde planladığınız her türlü elektronik devreyi kurabilme olanağını bulabilirsiniz. Ancak oldukça zaman alıcı bir yöntemdir. İkinci yöntem ise PIC deneme kartı kullanmaktır. PIC deneme kartları genellikle farklı uygulama programlarının denenmesi için geliştirilmiştir. Port çıkış uçlarına LED'ler, giriş uçlarına da butonlar bağlanmıştır. PCB üzerine kurulduklarından kart üzerinde sonradan değişiklik yapmak mümkün değildir. Bu nedenle programlar kartın dizaynına göre geliştirilmelidir. PIC programlamaya yeni başlayanların farklı programlama tekniklerini uygulayıp öğrenmeleri açısından bakılırsa ideal bir araçtır.

BreadBoard Üzerine Kurulan Devre ile Denemek

- Programladığınız PIC'in çalışmasını izlemek amacıyla aşağıdaki devreyi bir breadbord üzerinde kurunuz.



Bu devrede PIC için gerekli olan clock sinyalini bir RC osilatör sağlamaktadır. MCLR reset girişi terslenmiş olduğundan aktif "0"da işlev görür. Reset girişi, 1K direnç üzerinden 5V'a bağlanmıştır. 100 nF'lık direncin görevi, PIC'e enerji verildiğinde meydana gelebilecek gerilim dalgalanmalarının PIC'e zarar vermesini önlemek içindir.

- Devreye enerji verdiğinizde PIC'in RBO bacağına bağlı olan LED doğrudan yanacaktır. DENEME.ASM assembly programını yazarken yapmak istediğimiz şey de zaten buydu.

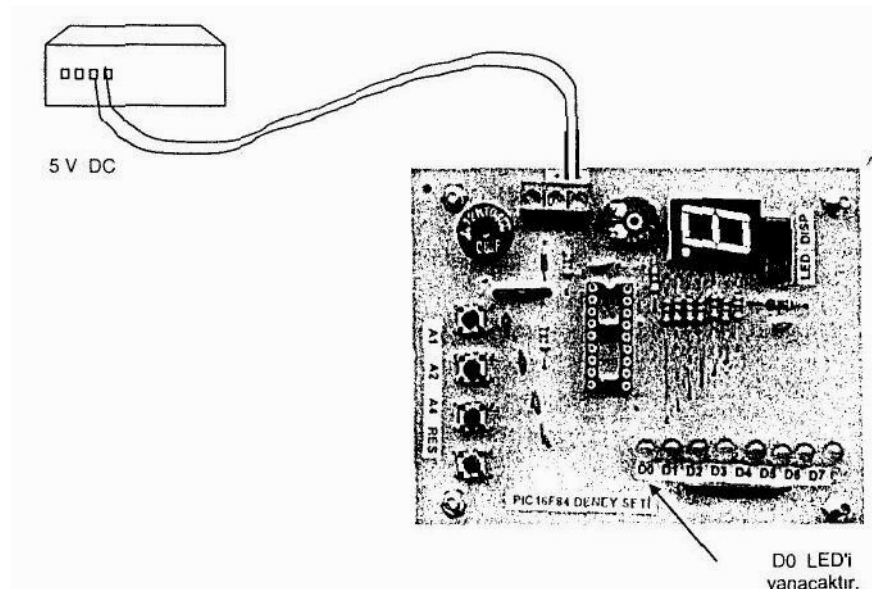
PIC Deneme Kartı ile Denemek

Piyasada eğitim amaçlı olarak hazırlanmış çeşitli kartlar bulunmaktadır. Bunlardan bir tanesini satın alabileceğiniz gibi, aşağıda resmi görülen ve detaylarını ekler bölümünde verdiğimiz kartı kendi olanaklarınızla üretebilirsiniz.

Port adı	Giriş/Çıkış durumu	Bağlı olan eleman
RB0	Çıkış	LED
RB1	Çıkış	LED
RB2	Çıkış	LED
RB3	Çıkış	LED
RB4	Çıkış	LED
RB5	Çıkış	LED
RB6	Çıkış	LED
RB7	Çıkış	LED
RA0	Giriş	Potansiyometre
RA1	Giriş	A1 butonu
RA2	Giriş	A2 butonu
RA3	Çıkış	Buzzer
RA4	Giriş	A4 butonu

NOT: Bu kitapta verilen programlar yukarıda özelliklerini verdiğimiz deneme kartının dizaynına göre verilmiştir. Bu dizaynda PortB'nin tüm çıkışlarına LED bağlanmıştır. Bu durum programlamaya yeni başlayanların port çıkışlarını yönlendirme işlemlerindeki karmaşıklığı ortadan kaldırır. Kitabın arkasındaki adını verdiğimiz firmadan farklı dizaynda olan deneme kartını satın aldıysanız programlarınızı yazarken bu kartın özelliklerini dikkate almanız gerekir.

- Deneme kartınıza programladığınız PIC16F84'ü yerleştirerek 5V DC güç kaynağına bağlayınız. D0 LED'i PIC'e enerji verir vermez yanacaktır.



MPASM'NIN ÜRETTİĞİ DİĞER DOSYALAR

Editör ile hazırladığınız kaynak program (DENEME.ASM) derlendikten sonra .HEX uzantılı bir dosya elde edildiğini gördük. MPASM bu dosyanın dışında (LST, ERR, COD, HXI-I, HXL, XRF) uzantılı dosyaları da oluşturarak yine aynı klasör içerisine kaydeder. Bu dosyaların ismi kaynak program ile aynıdır. Çoğu zaman kullanmak zorunda kalacağımız LST ve ERR dosyaları, assembly komutlarını yazarken yapılabilecek yazım hatalarını bulmakta çok yararlıdır.

.LST Dosyası

.LST dosyasını kullandığınız editörden açabilirsiniz. Bu dosya iki sayfadan oluşmuştur. İlk sayfada (PAGE 1);

- Komutun program belleği veya RAM'daki adresi (LOC),
- Komutların hexadesimal karşılıkları (OBJECT),
- Kaynak program ve satır numaraları (LINE SOURCE TEXT) verilir.

İkinci sayfada ise (PAGE 2);

- Programda kullanılan etiketler (SYMBOL TABLE) ve adresleri (VALUE),
- Bellek kullanım haritası ('X'ler kullanılan alanı, '-' ler kullanılmayan alanı gösterir.)
- Kullanılan ve boş kalan alanın miktarı (DENEME.LST dosyasında kullanılan alan=5, kullanılmayan alan=1019),
- Hata (Error) sayısı, uyarı (Warning) sayısı verilir.

MPASM 02.30 Released DENEME.ASM 5-10-2000 22:57:44 PAGE 1

LOC OBJECT CODE LINE SOURCE TEXT VALUE

00001 ;===DENEME.ASM===27/04/2000=====

	00002	LIST	P=16F84 ;PIC16F84'ün MPASM'ye tanıt
00000006	00003	PORTB EQU	h'06"
00000003	00004	STATUS EQU	h'03'
00000086	00005	TRISB EQU	h'86'
0000 0186	00006	CLRF PORTB	;PORTB'ye bağlı LED'leri söndür
0001 1683	00007	BSF STATUS, 5	;BANKI'e geç
0002 0186	00008	CLRF TRISB	;PORTB'nin uçlarını çıkış yap
0003 1283	00009	BCF STATUS, 5	;BANKO'a geç
0004 1406	00010	BSF PORTB, 0	;PORTB 0.bitindeki LED'i yak
	00011	END	;Program komutlarının sonu

MPASM 02.30 Released DENEME.ASM 5-10-2000 22:57:44 PAGE 2

SYMBOL TABLE

LABEL	VALUE
PORTB	00000006
STATUS	00000003

```
TRISB                      00000086
_16F84                     00000001
MEMORY USAGE MAP ('X' = Used, '-' = Unused)
0000 : XXXXX----- _ _ _ _ _
All other memory blocks unused.
Program Memory Words Used:   5 Program Memory Nords Free: 1019
Errors   :    0
Harnings :    0 reported,    0 suppressed
Messages ;    0 reported,    1 suppressed
```

.ERR Dosyası

Bu dosyayı da kullandığınız editörden açabilirsiniz. Eğer assembly komutlarının yazılışında bir hata yaptıysanız, hatalı satır numarası ve hatanın ne olduğu yazılıdır. Hatasız programlarda bu dosyanın içi boştur. Şimdi DENEME.ASM dosyasında bir hata yaparak bu dosyada ne yazıldığını görelim.

- DENEME.ASM dosyasının 6. satırındaki "**CLRF**" komutunu "**CLR**" olarak değiştirin.
- Daha sonra da programı MPASM ile derleyip, **ERR dosyasını editörle görüntüleyin.**

```
Warning[207] C:\P16PRO\DENEME.ASM 6 : Found label after column 1. (CLR)
Error[122]  C:\P16PRO\DENEME.ASM 6 : Illegal opcode (PORTB)
```

Yazılan kaynak dosyada hata varsa, LST uzantılı dosyada da bu hatalar gösterilir. LST uzantılı dosyadaki hatalar, hatanın yapıldığı satırdan hemen sonra yazılması nedeniyle görülmesi daha kolaydır.

INCLUDE DOSYALARI

Assembly programlarını yazarken kullanılacak olan registerlerin adreslerini tanımlama bölümünde kullanmak, programı daha anlaşılır hale getirmektedir. Aslında PIC16F84 mikrodnetleyicisinin RAM belleğindeki özel registerlerin adresleri sabittir. öyleyse adresleri sabit olan registerleri her programı yazarken yeniden tanımlamak gereksiz görülmektedir. Microchip bu durumu göz önüne alarak **INCLUDE** dosyalarını kullanıma sunmuştur. **Header file** denilen bu dosyaları kullanmak suretiyle programları yazarken her defasında register adreslerini tanımlama zorunluluğu kaldırılmıştır. MPASM ile derlenebilecek her PIC için ayrı bir INCLUDE dosyası bulunmaktadır. Bu dosyalar MPASM programının bulunduğu klasör içerisinde. Dosyaları görmek için editörünüzü kullanabilirsiniz ya da DOS ortamına dönüp D[R komutu ile INC uzantılı dosyaları listeleyebilirsiniz. Şimdi INCLUDE dosyalarının ne işe yaradığını ve nasıl kullanıldığını görelim.

INCLUDE Dosyası Kullanarak Program Yazmak

Bir .INC dosyasını kaynak program içerisinde kullanabilmek için, INCLUDE bildirinden sonra dosyanın ismini yazmak yeterlidir. PIC16F84 mikrodnetleyicisi için kullanacağımız dosyanın adı "P16F84.INC" dir. Şimdi bu dosyayı editörümüzle açarak içerisinde neler yazıldığına bakalım.

- Editörünüzle MPASM klasörü içerisine girerek **P16F84.INC dosyasını açınız.** Aşağıdaki listeyle karşılaşacaksınız.

```
LIST
;P1684.INC Standard Header File, Version 2.00 Microchip Technology, Inc. NOLIST
;This header file defines configurations,registers,and other useful bits of ;information for the PIC16F84 microcontroller. These names are
taken to match the ;data sheets as closely as possible.

;Note that the processor must be selected before this file is included. The ;processor may be selected the following ways:
;1. Command line switch:
;      C:\ MPASM MYFILE.ASM /PIC16F84
;
;      2. LIST directive in the source file
;      LIST P=PIC16F84
;
;      3. Processor Type entry in the MPASM full-screen interface
;-----
;Revision History
;-----
```



```

;Rev:   Date:           Reason:

;2.00  07/24/96        Renamed to reflect the name change to PIC16F84.
;1.01  05/17/96        Corrected BADRAM map
;1.00  10/31/95        Initial Release
;-----
;Verify Processor
;-----
        IFMDEF __16F84
            MESSG "Processor-header file mismatch. Verify selected processor." ENDIF
;-----
;    Register Definitions
;-----
W                EQU    H'0000'
F                EQU    H'0001'
;----- Register Files-----
INDF                EQU    H'0000'
TMRO                EQU    H'0001'
PCL                EQU    H'0002'
STATUS              EQU    H'0003'
FSR                 EQU    H'0004'
PORTA               EQU    H'0005'
PORTB               EQU    H'0006'
EEDATA              EQU    H'0008'
EEADR               EQU    H'0009'
PCLATH              EQU    H'000A'
INTCON              EQU    H'000B'

OPTION_REG           EQU    H'0081'
TRISA               EQU    H'0085'
TRISB               EQU    H'0086'
EECON1              EQU    H'0088'
EECON2              EQU    H'0089'
;----- STATUS      Bits -----
IRP                 EQU    H'0007'
RPI                 EQU    H'0006'
RPO                 EQU    H'0005'
NOT_TO              EQU    H'0004'
NOT_PD              EQU    H'0003'
Z                   EQU    H'0002'
DC                  EQU    H'0001'
C                   EQU    H'0000'
;----- INTCON      Bits -----
GIE                 EQU    H'0007'
EEIE                EQU    H'0006'
TOIE                EQU    H'0005'
INTE                EQU    H'0004'
RBIE                EQU    H'0003'
TOIF                EQU    H'0002'
INTF                EQU    H'0001'
RBIF                EQU    H'0000'
;----- OPTION      Bits -----
NOT_RBPU            EQU    H'0007'
INTEDG              EQU    H'0006'
TOCS                EQU    H'0005'
TOSE                EQU    H'0004'
PSA                 EQU    H'0003'
PS2                 EQU    H'0002'

```

```

PSI EQU H'0001'
PSO EQU H'0000'
;----- EECON1 BitS -----
EEIF EQU H'0004'
WRERR EQU H'0003'
WREN EQU H'0002'
WR EQU H'0001'
RD EQU H'0000'
;-----
; RAM Definition
__MAXRAM H'CF'
__BADRAMH'07', H • 50 •-H • 7F', H'87'
;-----
Configuration Bits
__CP_ON EQU H'000F'
__CP_OPF EQU H'3FFF'
__PWRTE_ON EQU H'3FF7'
__PWRTE_OFF EQU H'3FFF'
__WDT_OM EQU H'3FFF'
__WDT_OFF EQU H'3PFB'
__LP_OSC EQU H'3FFC'
__XT_OSC EQU H'3FFD'
__HS_OSC EOU H'3FFE'
__RC_OSC EQU H'3FFF'

```

Dosyayı inceleyecek olursanız RAM içerisindeki tüm file registerlerin adresleri EQU komutu ile tanımlanmıştır. Bu durumda P16F84.INC dosyasını kaynak program içerisinde İMCUİDE "P16F84.INC" şeklinde yazarsak, aşağıdaki komutları yazmak zorunda kalmayacağız.

```

PORTB EQU h'06'
STATÜS EQU h'03'
TRISB EQU h'86'

```

Status registerin her bir bit'ine bir isim verildiğinden (IRP EQU H'0007', RP1 EQU H'0006', RP0 EQU H'0005'... gibi) hangi bit kullanılmak isteniyorsa o bitin numarası değil de ismi kullanılabilir. örneğin;

```

BSF STATUS, 5 yerine
BSF STATUS,RPO yazılabilir.

```

Yukarıda söylenenler INTCON, OPTION, EECON1 registerleri için de geçerlidir. Ayrıca konfigürasyon bit'lerine de isim verildiğinden, PIC'e yazdırılması gereken konfigürasyon bit'lerinin hexadesimal karşılıkları yerine tanımlanan isimler kullanılır. Şimdi DENEME.ASM kaynak dosyasını INCLUDE dosyası kullanarak yeniden yazalım. Kodların daha da azaldığını göreceksiniz.

```

;===DEMEME.ASM===27/04/2000=====
LIST P=16F84 ;PIC16F84'ün MPASM'ye tanıt.
INCLUDE 'P16F84.IMC'
CLRF PORTB ;PORTB'ye bağlı LED'leri söndür
BSF STATUS, RP0 ;BANK1'e geç
CLRF TRISB ;PORTB'nin uçlarını çıkış yap
BCF STATÜS, RP0 ;BANK0'a geç
BSF PORTB, 0 ;PORTB'nin 0.bitindeki LED'i yak
END ;Program komutlarının sonu

```

KONFIGÜRASYON BİTLERİNİN YAZILMASI

Konfigürasyon bit'leri, PIC'e gerilim uygulandığı anda PIC'in uyması gereken koşulları belirlemede kullanılır. Bu koşullar PIC uygulama devresine bağlıdır. Örneğin uygulama devresinde RC osilatör kullanılıyorsa, bu koşul kaynak program içerisinde bulunmalıdır ve PIC'e yazdırılmalıdır.

Konfigürasyon bit'leri aşağıdaki koşulları belirlemede kullanılır:

- İlk programımızda konfigürasyon bit'lerini yazmadığımızı gördünüz. Aslında program içerisinde yazma zorunluluğu da yoktur. Çünkü çoğu PIC programlayıcı yazılımı, bu bit'lerin programı PIC'e yazdırma esnasında yazılabilmesi veya değiştirilmesi için olanaklar tanımıştır. Örneğin, P16PRO programlayıcısında F2 tuşuna basarak ekrana getirdiğimiz pencerede bu konfigürasyonu yapabiliyoruz.

Koşul	Yazılacak tanım kodları
Kod koruma var	<code>_CP_ON</code>
Kod koruma yok	<code>_CP_OFF</code>
Power-on-reset var	<code>_PWRTE_ON</code>
Power-on-reset yok	<code>_PWRTE_OFF</code>
Watchdog timer devrede	<code>_WDT_ON</code>
Watchdog timer devrede yok	<code>_WDT_OFF</code>
Low Power Osilatör	<code>_LP_OSC</code>
Kristal osilatör	<code>_XT_OSC</code>
High Speed osilatör	<code>_HS_OSC</code>
RC Osilatör	<code>_RC_OSC</code>

[illegible]

6

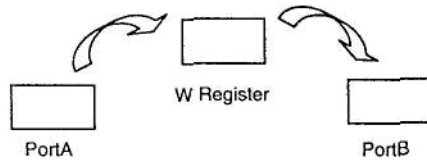
VERİ TRANSFERİ ve KARAR İŞLEMLERİ

- ❑ W REGISTERİNİN KULLANIMI (MOVLW, MOVWF KOMUTLARI)
- ❑ BIT TEST EDEREK KARAR VERME (BTFSC, BTFSS)

W REGISTERİN KULLANIMI (MOVLW, MOVWF KOMUTLARI)

Veri Transferi

W register, RAM içerisindeki file registerlerden bağımsız bulunmaktadır. Registerler arasında veri transferi yapmak için kullanılır. örneğin, PortA içerisindeki veriyi PortB içerisine transfer etmek için aşağıdaki komutları yazmak gerekir.



MOVF PORTA, W ;PortA'nın içeriğini W registre taşı
MOVWF PORTB ;W registerin içeriğini PortB'ye gönder

PortB'ye bağlı olan 8 LED'in ilk dört tanesini yakacak olan veriyi göndermek için de aşağıdaki komutlar yazılmalıdır.

MOVLW H'0F' ;W registerine H 0H yükle

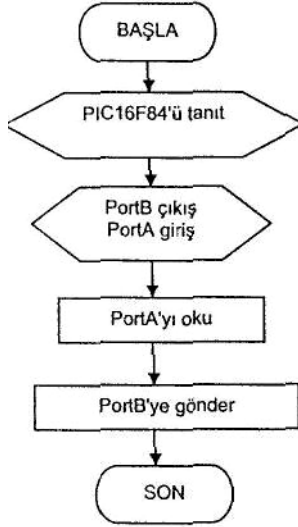
Binary karşılığı b'00001111'

MOVWF PORTB ;W registeri içeriğini PortB'ye gönder

Eğer bir register içerisine H'00' verisi gönderilmek istenirse, o registerin içerisini CLRF komutu ile silmek daha pratik bir yöntemdir.

CLRF PORTB ;PortB'nin içeriğini sıfırla

PROGRAM-1) PIC'e enerji verildiğinde PortA'nın uçlarına bağlı butonlardan hangisi basılı tutulursa, PortB'de o butona karşılık gelen LED'i söndüren program.



==PROG1.ASM==14/05/2000=====

LIST P=16F84

PORTA EQU h'05"

PORTB EQU h'06'

STATUS EQU h'03'

TRISA EQU h'85"

TRISB EQU h'86'

CLRF PORTB

BSF STATUS, 5

CLRF TRISB

MOVLW h'FF'

MOVWF TRISA

BCF STATUS, 5

BASLA MOVF PORTA, W

MOVWF PORTB

DONGU

GOTO DONGU END ;Sonsuz döngü ve program sonu

;PORTB'ye bağlı LED'leri söndür

;BANKI'e geç

;PORTB'nin uçlarını çıkış yap

;W registerine b'FF' yükle

;PortA'nın uçlarını. giriş yap

;BANK='a geç

;PortA'yı oku, sonucu W'ye yaz

;Butonların durumunu PortB'de göster

İşlem Basamakları

1. Programı editörünüzde yazdıktan sonra, PROG1.ASM adıyla kaydediniz. Daha sonra da MPASM ile derleyiniz.
2. PIC programlayıcınızı bilgisayarınıza bağlayıp PIC'i programlayınız. Programlama esnasında aşağıda verilen konfigürasyona dikkat ediniz.

Oscilatör tipi →RC,

Watchdog timer(WDT) →OFF,

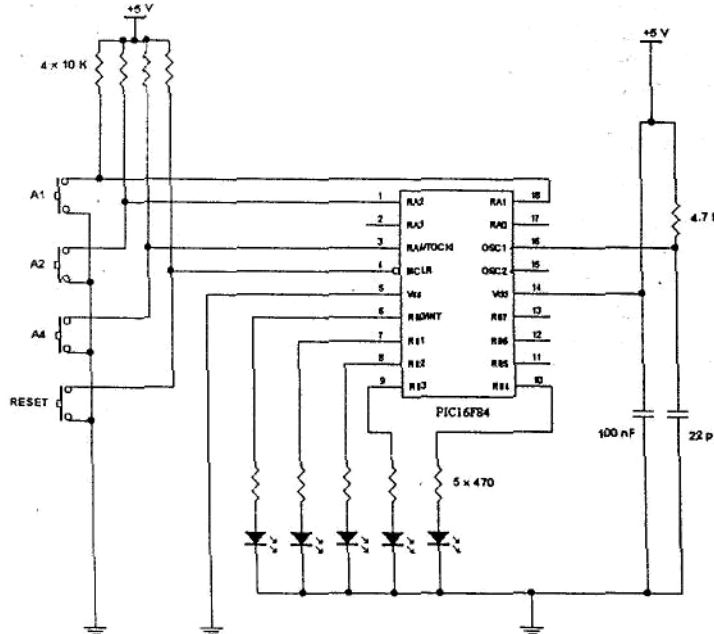
Power Up Timer(PWRT) →ON,

Code Protect (CP) →OFF

P16PRO programının sağ üst tarafındaki Fuses penceresi aşağıdaki gibi olmalıdır.

Fuses	
Osc:	RC
WDT:	Off
PWRT:	On
CP:	Off
ID:	080A0205
CkSum:	8A25

3. Breadboard üzerinde aşağıdaki devreyi kurunuz.



4. Devreye enerji vermeden önce A1, A2, A3 butonlarına birlikte veya tek olarak basınız.
5. Devreye enerji veriniz.
6. Bastığınız butonlarla sönen LED'leri karşılaştırınız.

NOT: Hiçbir butona basmadığınızda RB0-RB4 uçlarına bağlı olan tüm LED'lerin yanık olduğunu göreceksiniz. Sebebi, A portunun tüm uçlarının pull-up oluşundandır. Hangi butona bastıysanız, o porta giren veri 0'dır. A portundaki veri

direkt olarak B portuna transfer edildiğinden, basılan butonun karşılığındaki LED sönecektir.

Programı tekrar çalıştırmak için PIC'in enerjisini kesip yeniden vermeniz gerekmez. RESET tuşuna basmanız yeterlidir.

7. B portundaki LED'leri durumunu değiştirmek için önce butona/butonlara basın, daha sonra da RESET tuşuna basınız.
8. Eğer breadboard üzerine yukarıdaki devreyi kurma olanağınız yoksa, programladığınız PIC'i denemek için hazır deneme kartınızı kullanınız.

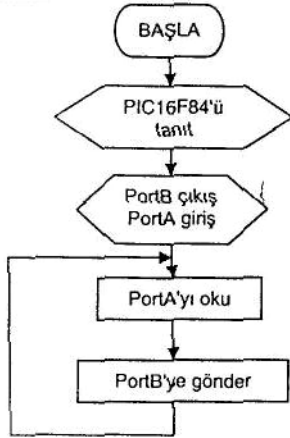
Sonsuz Döngü

PIC16F84 mikrodenetleyicisinde duraklama komutu olmadığı için yukarıdaki programda bu işlemin yerine geçen aldatıcı bir duraklama uyguladık. Programın en sonunda GOTO komutu ile programın akışı DONGU etiketine gönderildi. Görüldüğü gibi DONGU etiketi ile GOTO komutu arasında işlem yapan hiçbir PIC komutu bulunmamaktadır. Bu durumda program sonsuz döngü içerisine girmiştir.

Sonsuz döngü içerisine istenirse komut yazılabilir. Bu durumda RESET tuşuna basılana kadar ya da PIC'in enerjisi kesilene kadar aynı komutlar defalarca tekrarlanabilir. Şimdi PROG1.ASM'de çok küçük değişiklik yaparak bu işlemi görelim.

PROGRAM-2) A portunun uçlarına bağlı butonlara basıldığı sürece, B portunda o butona karşılık gelen LED'i söndüren program.

Bu programda A portundaki RA0-RA4 uçlarının devamlı olarak okunup B portuna gönderilmesi gerekiyor. Bunu devamlı olarak yapabilmek için programı "BASLA" etiketine gönderip, sonsuz döngüyü buraya kurmak gerekiyor. Akış diyagramını yeniden çizelim.



Programın PROG1.ASM ile arasındaki tek fark GOTO komutunun yazıldığı satır olacaktır.

İşlem Basamakları

1. PROG1.ASM' yi editörünüze yükleyerek GOTO ile başlayan satırı aşağıdaki gibi değiştiriniz. Yeni programı PROG2.ASM adıyla kaydediniz.

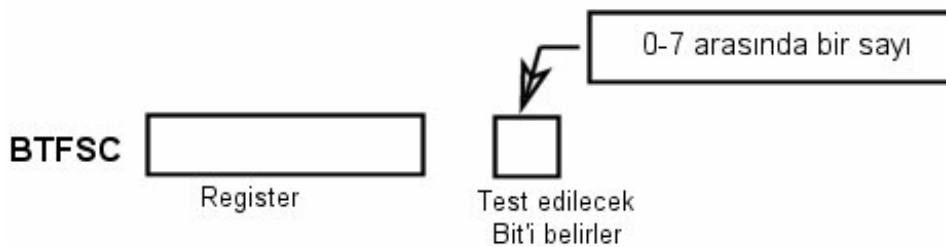
GOTO BASLA ;Okumayı tekrarla

2. Programı derledikten sonra PIC'e yazdırınız.
3. PIC16F84'ü yukarıda kurduğunuz devreye veya deneme kartına yerleştirip programı çalıştırınız.
4. Programın çalışması esnasında buton/butonlara basarak sönen LED'leri izleyiniz.

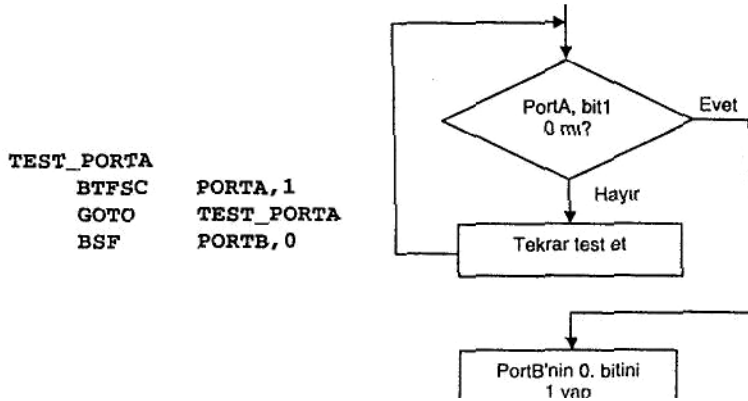
BİT TEST EDEREK KARAR VERMEK (BTFSC, BTFSS)

Bir register içerisindeki herhangi bir bit BTFSC veya BTFSS komutları ile test edilebilir. Bu test sonucuna göre program akışı istenilen komuta dallandırılabilir.

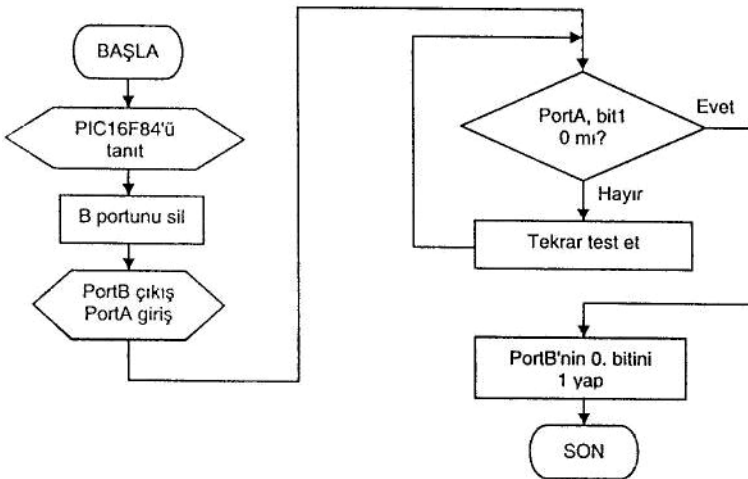
BTFSC komutu İngilizce'deki "Bit Test File register Skip if Clear", komutunun kısaltmasıdır. Türkçe'deki karşılığı "Registerdeki bit'i test et, eğer "0"sa bir sonraki komuta atla" biçimindedir. BTFSS komutu ise "Bit Test file register Skip if Set" cümlesinin kısaltmasıdır. Türkçe karşılığı ise "Registerdeki bit'i test et, eğer "1"se bir sonraki komuta atla"dır.



Komutun işleyişini örnek bir program parçasının akış diyagramını çizerek inceleyelim.



PROGRAM-3) PortB'nin 0. bitine bağlı LED'i, A portunun 1. bit'indeki butona basınca yakan program.



;==PROG3.ASM==22/05/2000=====

```

LIST P=16F84
PORTA    EQU    h'05"
PORTB    EQU    h'06'
STATUS   EQU    h'03'
TRISA    EQU    h'85'
TRISB    EQU    h'86'

CLRF     PORTB      ;PORTB'ye bağlı LED'leri söndür
BSF      STATUS, 5   ;BANK1'e geç
CLRF     TRISB      ;PORTB'nin uçlarına. çıkış yap
MOVLW    h'FF'      ;W registere h'FF' yükle
MOVWF    TRISA      ;PortA'nın uçlarını giriş yap
BCF      STATUS, 5   ;BANK0'a geç

TEST_PORTA
    BTFSC  PORTA, 1   ;A portunun 1. bitini test et
    GOTO   TEST_PORTA ;0 değilse tekrar test et
    BSF    PORTB, 0   ;B portunun 0. Bitini 1(Bet) yap
    DONGU
    GOTO   DONGU
    END
    
```

İşlem Basamakları

1. Programın tanımlamalar bölümünde INCLUDE komutunu kullanınız. Programı PROG3.ASM olarak kaydediniz.
2. Programı derledikten sonra PIC'e yazdırınız.
3. PIC16F84'ü deneme kartına yerleştiriniz.
4. Programı çalıştırdığınızda LED'lerin sönük olduğunu izleyiniz.
5. A1 butonuna basarak RBO ucuna bağlı LED'in yandığını görünüz.

KENDİNİZ UYGULAYINIZ

Farklı butonlara (A2 veya A4'den birisini) basınca, B portundaki (RBO-RB4 uçlarına bağlı herhangi birisi) bir LED'i yakan programı kendiniz yazınız ve deneme kartı üzerinde çalışmasını görünüz.

PROGRAM-4) A portunun 2. bit'indeki butona basınca B portuna bağlı tüm LED'leri yakan program.

Bu programın PROG3.ASM'den tek farkı portB'ye gönderilecek olan verinin farklı (b'00000001' yerine b'11111111') olmasıdır. BCF veya BSF komutlarıyla sadece bir bit'i 0(clear) veya 1(set) yapabiliyoruz. Bu nedenle bu komutun kullanıldığı yerde değişiklik yapmamız

İşlem Basamakları

1. PROGS.ASM'yi editörünüze yükleyip "TEST_PORTA" etiketinden sonra yazılanları aşağıdaki gibi değiştiriniz. Yeni programı PROG4.ASM olarak kaydediniz.

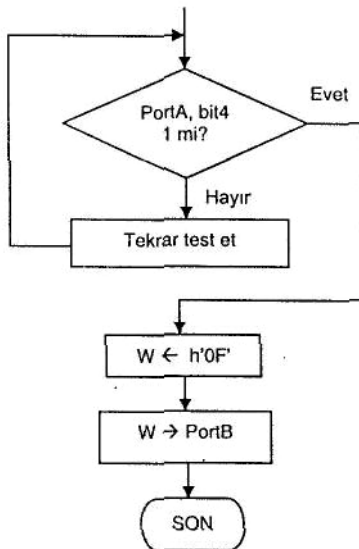
TEST_PORTA

```
BTFSF PORTA, 2      ;PortA'nın 2. Biti 0(clear) nu.?
GOTO TEST_PORTA     ;Tekrar test et
MOVLW H'FF'         ;W registere b'11111111' yükle
MOVWF PORTB         ;W registeri portB'ye gönder
DONGU
GOTO DONGU
END
```

2. Programı derledikten sonra PIC16F84'e yazdırın.
3. PIC'i breadboard üzerine kurduğunuz devre üzerinde ya da deneme kartında çalıştırınız.
4. A2 butonuna basınca B portundaki tüm LED'lerin yandığını görünüz.

PROGRAM-5) A portunun 4. bitine bağlı olan buton basılı tutularak PIC'e enerji verildiğinde, B portundaki LED'leri sönük tutan, butondan el çekildiğinde ilk dört LED'i yakan program.

Bu programın PROG4.ASM'den farkı BTFSF komutu yerine BTFSF komutunu kullanmak zorunda oluşumuzdur. Çünkü A portunun 4. bit'indeki butonu basılı tutduğumuzda buradaki veri 0(clear)dır. Butondan elimizi çektiğimizde veri 1(set)tir. Butona basılı durumdayken LED'lerin sönük kalabilmesini sağlamak için BTFSF komutuyla program akışı "TEST_PORTA" etiketine gönderilmelidir. Buna göre akış diyagramının sadece değişen bölümünü yeniden çizelim.



İşlem Basamakları

1. Editörünüze PROG4.ASM'yi yükleyerek aşağıdaki değişiklikleri yapınız. Yeni programı PROG5.ASM adıyla kaydediniz.

TEST_PORTA

```
BTFSF PORTA, 4      ;PortA'nın 4. bit'i 1(set) mi?
GOTO TEST_PORTA     ;Hayır, tekrar test et
MOVLW H'0F'         ;W registere b'00001111' yükle
MOVWF PORTB         ;W registeri portB'ye gönder
DONGU
GOTO DONGU
END
```

2. Programı PIC16F84'e yazdırdıktan sonra deneme devresine yerleştiriniz.
3. A4 butonunu basılı tutup, PIC'e gerilim uygulayınız.

4. Butona bastığınız sürece LED'lerin sönük kaldığını, elinizi çeker çekmez B portundaki ilk 4 LED'in yandığını izleyiniz.
5. Yeni denemeler yapmak için RESET butonuna basarak programı tekrar çalıştırın.

7 DÖNGÜ DÜZENLEMEK

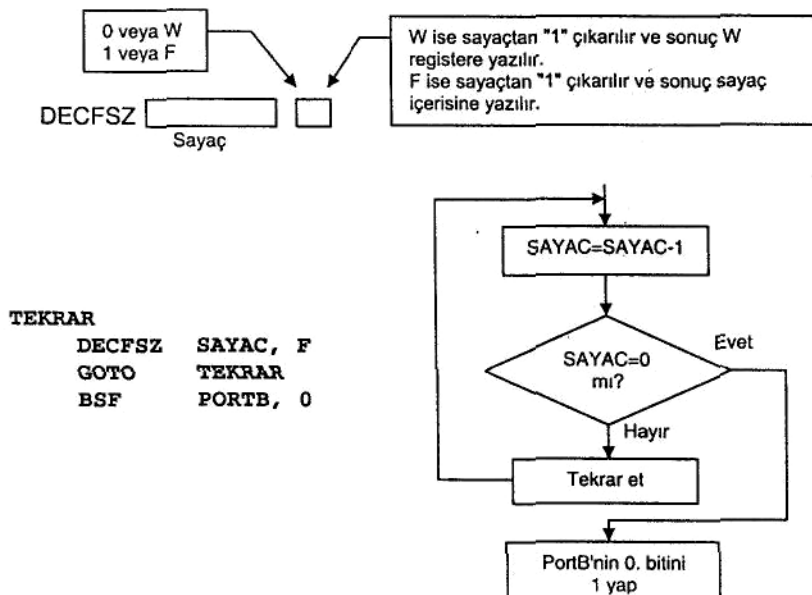
- ❑ SAYAÇ KULLANARAK DÖNGÜ KULLANMAK (DECFSZ KOMUTU)
- ❑ KARŞILAŞTIRMA İLE DÖNGÜ DÜZENLEMEK (SUBLW, SUBWF KOMUTLARI)
- ❑ STATUS REGİSTER

SAYAÇ KULLANARAK DÖNGÜ KULLANMAK (DECFSZ)

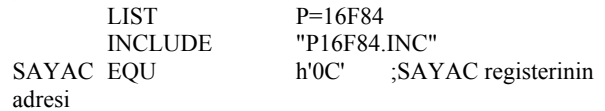
Bazı işlemlerin önceden belirlenen sayıda tekrarlanması gerekebilir. Bu durumda bir register sayaç olarak kullanılır. İlk önce sayaç içerisine işlemlerin tekrar sayısını belirleyen sayı yüklenir. Daha sonra her bir işlem tekrarında sayaç içerisindeki değer bir azaltılır. Azaltma işlemi DECFSZ komutu ile yapılır. Sayaç içerisindeki sayı 0 (sıfır) olunca döngü biter ve program ya bitirilir ya da başka bir komuta dallandırılarak devam eder.

DECFSZ komutu İngilizce'deki "Decrement file register Skip if Zero" dur. Türkçe'ye çevirirsek şu anlam çıkar; Registerden "1" çıkart, eğer sonuç "0"sa bir sonraki komuta atla.

DECFSZ komutunun yazılışı ve örnek akış diyagramı aşağıdaki gibidir.



;==PROG6.ASM====25/05/2000=====

[illegible]

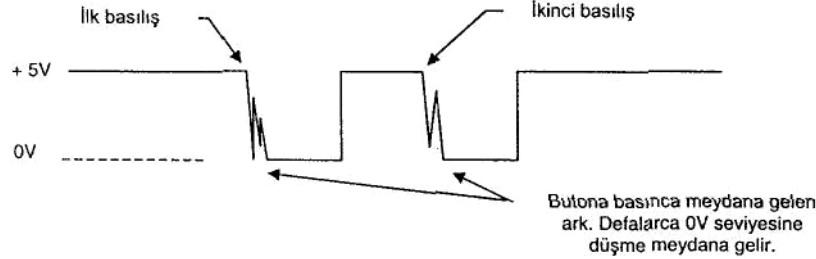
İşlem Basamakları

1. Programı yazdıktan sonra PIC'i programlayınız.

2. PIC'i deneme kartına yerleştiriniz.
3. A1 butonuna kısa sürelerle 10 defa basınız.

NOT: Butona basma sayısı 10'a ulaşmadan PortB'nin 0. bit'indeki LED'in yandığını göreceksiniz. Sebebini şöyle açıklayabiliriz;

A1 butonu pull-up yapılmıştır. Yani butona basılmadığında +5V'ta dır. Basıldığında ise 0V olur. Butona 2 defa basıldığını düşünerek PortA'nın RA1 ucundaki gerilimin eğrisini aşağıdaki gibi çizelim.



Butonlar mekanik kontaklardan yapıldığından butonun bırakılması esnasında ark oluşacaktır. Şekilde de görüldüğü gibi ilk basılışta gerilim dalgalanmaları olmaktadır. Bu dalgalanmalar esnasında gerilim üç defa 0V seviyesine inmektedir. Butona basış biçimi ve kontakların

durumuna göre bu sayı azalabilir veya artabilir. Butona bir defa basıldığı halde 2~3 defa basılmış gibi 0V seviyesine inecektir. PIC komutlarının icra edilme süresi çok kısa olduğundan her 0V seviyesine iniş, butona basılmış gibi işlem görerek SAYAC registeri içerisinde her defasında "1" çıkaracaktır. Bu da bizim beklediğimiz işlemden farklı bir sonuç doğuracaktır.

Butonda meydana gelen arki önlemek amacıyla NOP komutları kullanılmıştır. NOP komutu 1 sayıkl süresince hiçbir şey yapmadan CPU'nun gecikme yapmasını sağlar. Kullanılan beş NOP komutu arki önlemek için yeterli gelmediği deneyde açıkça görülmektedir. Eğer NOP komutu sayısını 100'e çıkarırsak, 100 sayıklık bir gecikme sağlayabiliriz. Bu da ilk 0V seviyesinden sonra oluşan 0V'ların algılanmamasını sağlayabilir.

4. Programdaki NOP komutlarının sayısını 100'e, (Gerekirse daha fazla) çıkararak programın doğru olarak çalışmasını sağlayınız.

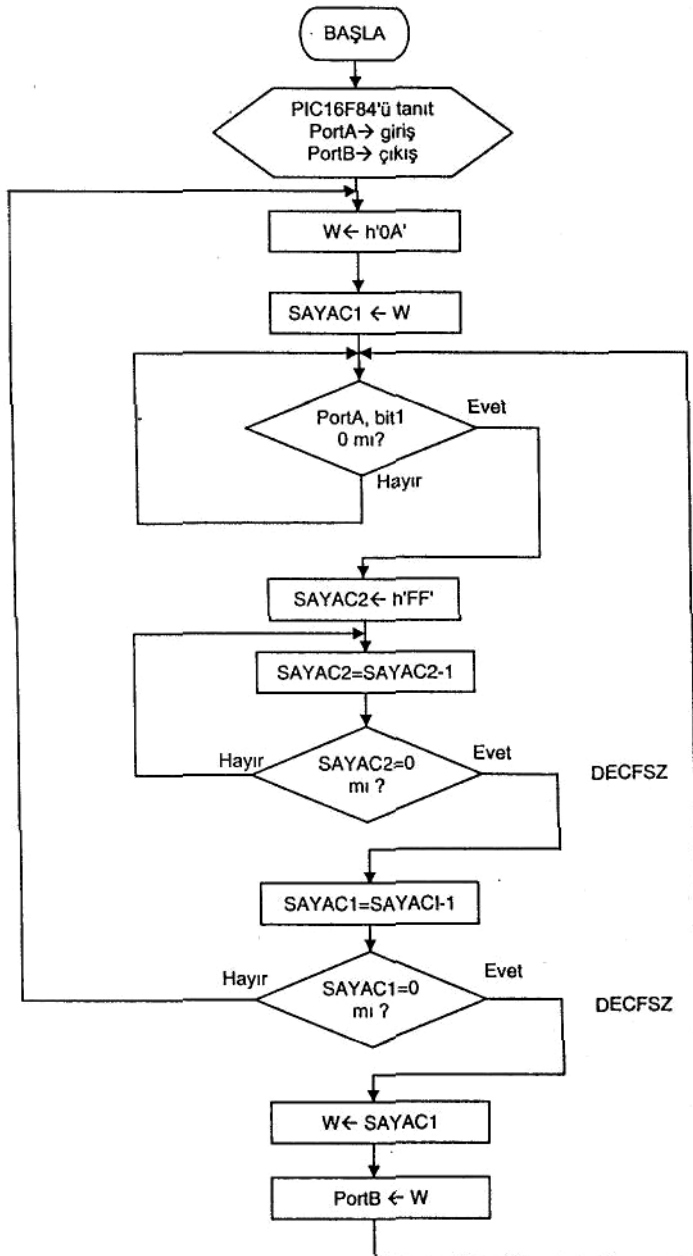
Programlar içerisinde gerekli olan bu tip zaman gecikmelerini sağlamak amacıyla yüzlerce NOP komutu yazmak, belleği gereksiz olarak doldurmak demektir. Aynı zamanda iyi bir programlama tekniği değildir. 8. bölümdeki zaman geciktirme döngüleri bu gibi işlemleri gerçekleştirmek için sıkça kullanılmaktadır.

Basit Bir Zaman Gecikme Döngüsü Yapmak

Şimdi de DECFSZ komutu kullanarak yapılan döngü içerisinde hiçbir işlem yapmayan komutlar kullanarak belirli bir zaman gecikme döngüsü yapılabilir. Örneğin 768 tane NOP komutunu alt alta yazarak gecikme sağlamak yerine, 3 tane NOP komutunu ardı ardına 256 defa çalıştırmak program içerisinde daha az yer kaplar. Şimdi bunu bir örnek programla gösterelim.

PROGRAM-7) A portunun 1. bit'indeki butona bastıkça B portundaki LED'leri binary 9'dan 0'a kadar azaltarak yakan program.

Buton üzerinde meydana gelebilecek arki DECFSZ komutu ile yapılan basit bir zaman gecikme döngüsü ile sağlanmıştır.



```

;====PROG7.ASM====29/05/2000=====
LIST          P=16F84
INCLUDE       "P16F84.INC"
SAYAC1 EQU    H'0C' ;SAYAC1 registerinin adresi
SAYAC2 EQU    H'0D' ;SAYAC2 registerinin adresi
CLRF          PORTB ;PortB'yi sıfırla
BSF           STATUS,5 ;BANK1'e geç
CLRF          TRISB ;PortB'nin tüm uçları çıkış
HOVLW         H'FF' ;W←h'FF •
HOVWF         TRISA ;PortA'nın tüm uçları giriş
BCF           STATUS, 5 ;BANK0'a geç

BASLA
MOVLW         H'0A' ;W←h'0A'
MOVWF         SAYAC1 ;SAYAC1←W

TEST
BTFSC         PORTA,1 ;PortA'nın 1. bit'i 0 mı?
GOTO          TEST ;Hayır, tekrar test. et

GECIKME ;Gecikme sağlama bölümü
MOVLW         H'FF' ;W←h'FF'
MOVWF         SAYAC2 ;SAYAC2←W
GECIKIME ;Gecikme sağlama bölümü
DECFSZ        SAYAC2, F ;SAYAC2←SAYAC2-1, SAYAC2=0 mı?
NOP           ; Arkın sönmesini bekle
NOP           ;
NOP           ;
GOTO          GECIKME ;Hayır, SAYAC2'yi tekrar azalt AZALT
DECFSZ        SAYAC1, F ;SAYACK-SAYAC1-1 ,SAYAC1=0 mı?
GOTO          YAK ;Hayır, YAK etiketine atla
GOTO          BASLA ;Evet, başa dön YAİ;
MOVF          SAYAC1, W ; W←SAYAC1
MOVWF         PORTB ;SAYAC1'i PortB'ye gönder
GOTO          TEST ;PortA'yi tekrar test et.
END           ;Prograa kodlarının sonu

```

İşlem basamakları

1. PIC'i programladıktan sonra breadboard üzerine kurduğunuz devreye y;
2. da deneme kartınıza yerleştiriniz.
3. PIC'e gerilim vererek çalıştırınız.
4. A1 butonuna çok kısa olarak basıp elinizi çekiniz. ilk yanan lamba 9'ı gösterecektir. Lambalar aşağıdaki gibi yanacaktır. 9→
●○○●

Butona tekrar basarak lambaların azalan binary sayıları göstermesini sağlayınız.

8→	○○○●	4→	○○●○	NOT: Meydana gelen arkların yok edilmesi için oluşturduğumuz basit gecikme devresi yeterli olmayabilir. Bu nedenle sayıların azaldığını görmek için butona oldukça kısa basmanız gerekir. Aksi halde sayılar azalırken arada atlamalar olabilir. Gecikme döngüleri konusunda zamanı uzatmak için gerekli programlama tekniğini 8. bölümde
7→	●●●○	3→	●●○○	
6→	○○●○	2→	○●○○	
5→	●○●○	1→	●○○○	

"Zaman geciktirme ve altprogramlar" ünitesinde öğreneceksiniz.

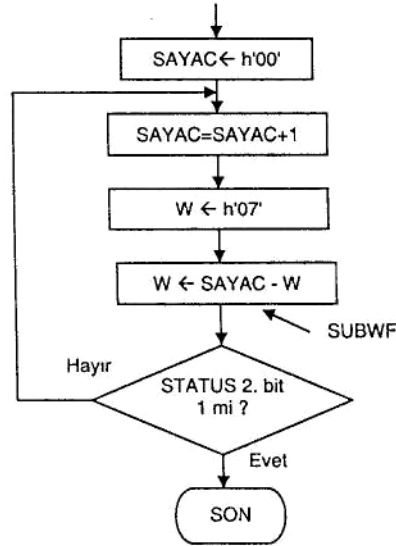
KARŞILAŞTIRMA YAPARAK DÖNGÜ DÜZENLEMEK (SUBLW, SUBWF, INCF, DECF KOMUTLARI)

Bazı işlerin önceden belirlenen sayıda tekrarlanması gerekebilir. Bu durumda bir register sayaç olarak kullanılır. Her bir işlem tekrarında sayaç içerisindeki sayı bir artırılır. Arttırma işlemi INCF komutuyla yapılır. .Sayaç içerisindeki sayı işlemlerin tekrar edilme sayısına ulaşınca, döngü sona erer ve program akışı başka bir komuta geçer. Bu karşılaştırma yöntemiyle yapılan döngünün akış diyagramını aşağıdaki gibi çizebiliriz.

```

CLRF    SAYAC
TEKRAR  INCF    SAYAC, F
        MOVLW   h'07'
        SUBWF   SAYAC, W
        BTFSC   STATUS, 2
        GOTO    TEKRAR
DONGU   GOTO    DONGU
        END

```

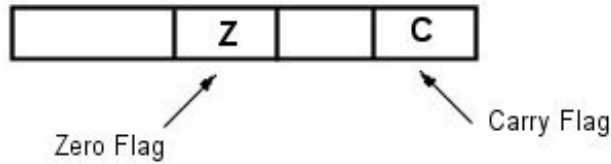


Döngünün tekrar sayısı W registeri içerisine yüklenen sayı ile belirlenir. SAYAC değişkeni içerisindeki sayı döngünün her tekrarında bir artırılır ve SUBWF komutuyla SAYAC'tan W'nin içeriği çıkarılır. Çıkarma işlemi neticesinde STATUS registerin 0. ve 2. bit'indeki değer etkilenir. Bu bit'ler BTFSC komutuyla test edilir. Testler sonucunda istenilen değere ulaşıncaya döngüye son verilir.

STATUS registerin 0. bit'ine CARRY FLAG (Taşma bayrağı), 2. bit'ine ZERO FLAG ((Sıfır

bayrağı) olarak adlandırılır.

STATUS register



SUBWF Komutu

SUBWF komutu çıkarma yapan bir komuttur. Aşağıdaki yazım kuralına göre W registerin içeriğini file registerden çıkarır. Sonucu W register içerisine yazar.

SUBWF File register, W

W ya da F olabilir.
W ise çıkarma sonucu W registere yazılır.
F ise çıkarma sonucu File registere yazılır.

Çıkarma sonucunda etkilenen C ve Z flag içerisindeki değer, W ve file registerin büyüklüklerine bağlıdır.

File register	> W	Z=0	C=1
File register	= W	Z=1	C=1
File register	< W	Z=0	C=0

Örnek program parçaları:

1. F = W durumu

```

        MOVLW   h'50'
        MOVWF   MEM
        SUBWF   MEM, W
W>MEM   BTFSC   STATUS, 2      W=MEM
        GOTO    XXXX          W=MEM
W<MEM   BSF     PORTB, 0

```

2. F<W durumu

```

        MOVLW   h'40'
        MOVWF   MEM
        MOVLW   h'50'
        SUBWF   MEM, W
W=MEM   BTFSC   STATUS, 0      W<MEM
W>MEM   GOTO    XXXX
        BSF     PORTB, 0      W<MEM

```

3. F>W durumu

```

        MOVLW   h'60'

```

	MOVWF	MEM	
	MOVLW	H'50'	
	SUBWF	MEM, W	
W=MEM	BTFSC	STATUS,0	W>MEM
	GOTO	XXXX	W>MEM
W<MEM	BSF	PORTB, 0	

SUBLW Komutu

SUBLW komutu çıkarma yapan bir komuttur. Aşağıdaki yazım kuralına göre sabit (L) içerisinde W registeri içeriğini çıkarır. Sonucu W registre yazar.

SUBLW Sabit ← Sabit değeri (Binary, heksadesimal veya desimal olabilir.)

Çıkarma sonucunda W ve sabitin büyüklükleri C ve Z flagların değerini değiştirir.

File register	> W	Z=0	C=1
File register	= W	Z=1	C=1
File register	< W	Z=0	C=0

Örnek program parçaları:

1. Sabit= W durumu

	MOVLW	h'50'	
	SUBWF	h'50'	
Sabit<W BTFSC	STATUS,2	Sabit=W	
Sabit>W GOTO	XXXX		
	BSF	PORTB, 0	Sabit=W

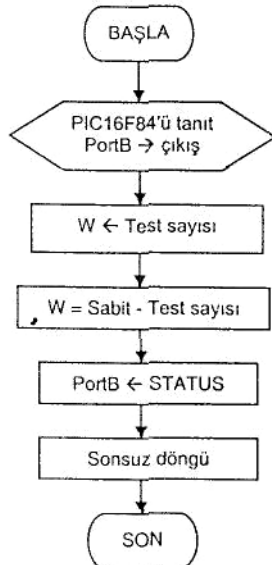
2. Sabit < W durumu

	MOVLW	h'60'	
	SUBLW	h'50'	
Sabit=W BTFSC	STATUS,0	Sabit<W	
Sabit>W GOTO	XXXX		
	BSF	PORTB, 0	Sabit<W

3. Sabit > W durumu

	MOVLW	h'40'	
	SUBLW	h'50'	
Sabit=W BTFSC	STATUS,0	Sabit>W	
	GOTO	XXXX	Sabit>W
Sabit<W	BSF	PORTB, 0	

PROGRAM- 8) SUBLVW komutu ile yapılan bir karşılaştırma sonucunda STATUS registerin içeriğini görüntüleyen program.



```

;===PROG8.ASM===01/06/2 000=====
LIST      P=16F84
INCLUDE "P16F84.INC"
CLRF      PORTB      ;PortB'yi temizle
BSP       STATUS, 5   ;BANK2'ye geç
CLRF      TRISB      ;PortB'nin tüm uçlarını çıkış yap
BCF       STATUS, 5   ;BANK1'e geç

BASLA
MOVLW     h'04'        ; W ← h'04'
SUBLW     h'05"        ; W = h'05' - h'04"
MOVWF     STATUS, W    ; STATUS'ü W registerine yaz
MOVWF     PORTB        ; STATUS'ü LED'lerde göster
DONGU
GOTO      DONGU
  
```

END

İşlem Basamakları

1. PIC'i programlayıp deneme kartı üzerine yerleştiriniz.
2. PIC'i çalıştırınız. PortB'nin LED'lerinin aşağıdaki gibi yandığını görünüz.

76 543210 000»»Cr«

Sabitin değeri h'05', W registerin değerinden h'04" büyük olduğundan C flag'ın (STATUS registerin 0. bit'i) değeri "1", Z flag'ın (STATUS registerin 2. bit değeri de "0". olacaktır. Bu nedenle sadece portB'nin 0. bitine bağlı olan LED yanacaktır. Diğer LED'lerin neden yanık olduğunu anlamak için STATUS register hakkında aşağıda verilen detaylı bilgileri inceleyiniz.

KENDİNİZ UYGULAYINIZ

Sabit değere W registerden büyük veya eşit değerler girerek, PIC'i yeniden programlayınız. Daha sonra da PIC'i çalıştırarak STATUS registerin durumunu LED'lerden izleyiniz.

STATUS REGISTER

STATUS register ALU'nun (Aritmetik Logic Unit) aritmetiksel durumunu (Zero ve Carry flag registerleri) ve RESET durumunu içeren verileri bulundurur. Ayrıca iki bitlik veri de data memory'deki (RAM) bank seçme bitleri için ayrılmıştır.

STATUS registerin içeriği bir çok komutun çalışması neticesinde bit'ler değişir. STATUS registerin Z, DC veya C bitlerini etkileyen bir komut kullanıldığında bu bitlere yazma işlemi yapılamaz. Bu bitlerin 1 veya 0 olm; durumu kullanılan PIC'e bağlıdır. Bank seçmek için kullanılan RPO ve RP1 bitlerini yazmak mümkün olduğu halde, TO ve PD bitlerine yazma işlemi yasaklanmıştır. Bu nedenle bir STATUS registeri etkileyen bir komutun çalıştırılması sonucunda etkilenen bitler beklenenden farklı olabilir.

Örneğin, CLRF STATUS en yüksek üç biti 0(sıfır), Z bit'ini ise 1(bir) yapar. Bu durumda STATUS register "000u u1uu" olarak kalır. (Buradaki "u" nun anlamı unchanged değişmezdir.)

Aşağıdaki tablo STATUS registerin her bir bit'inin hangi durumlarda 1 ve 0 olacağını gösterir.

STATUS REGISTER BIT'LERİ

bit 7	6	5	4	3	2	1	0
IRP	RP1	RP0	TO	PD	Z	DC	C

Bit 7: **IRP**: Register Bank Select bit (Bank seçme bit'i)

0= Bank 0, 1 (h'00'-h'FF')

1=Bank 2,3 (h'100'-h'1FF')

IRP biti PIC16F84 de kullanılmaz. IRP, 0 (sıfır) olarak kalmalıdır.

Bit 6-5:**RP1:RP0**: Register Bank Select bit (Bank seçme bitleri)

00= Bank 0 (h'00'-h'FF')

01=Bank 1 (h'80'-h'FF')

10=Bank2(h'100'-h'17F')

11=Bank3(h'180'-h'1FF')

PIC16F84'de sadece RP0 bit'i kullanılır. RP1, 0 (sıfır) olarak kalmalıdır.

Bit 4: **TO**: Time-out bit (Zaman dolma bit'i)

1= PIC'e enerji verildiğinde, CLRWDT ve SLEEP komutu çalışınca.

0= WDT zamanlayıcısında, zaman dolduğunda.

Bit 3: **PD**: Power-down bit (Enerji kesime bit'i)

1= PIC'e enerji verildiğinde ve CLRVDVT komutu çalışınca.

0= SLEEP komutu çalışınca.

Bit 2: **Z**: Zero bit (Sıfır bifi)

1 = Bir aritmetik veya mantıksal komutun sonucu 0 (sıfır) olduğunda.

0= Bir aritmetik veya mantıksal komutun sonucu 0 (sıfır) olmadığında.

Bit 1: **DC**: Digit carry/borrow bit (Taşma/ödünç biti) ADDLW ve AODVVF komutları kullanıldığında oluşan taşma ve ödünç alma olduğunda)

1= All dört bitin 4. bit'inde taşma meydana geldiğinde.

0= Alt dört bitin 4. bit'inde taşma meydana gelmediğinde.

Bit 0: **C**: Carry/borrow bit (Taşma/ödünç biti) ADDLW ve ADDVVF komutları kullanıldığında. Oluşan taşma ve ödünç alma olduğunda)

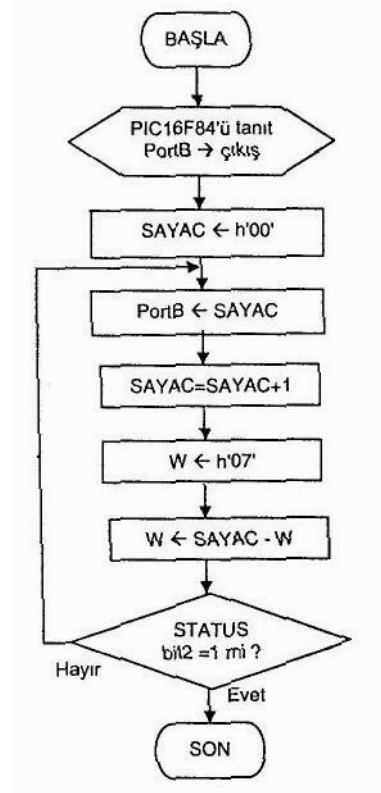
1= En soldaki 7. bit'te taşma olduğunda.

0= En soldaki 7. bit'le taşma olmadığında,

Not: RLF ve RRF komutları çalıştığında en sol bit veya sağ bitin değeri carry bit'ine yüklenir.

PROGRAM-9) SUBWF komutu kullanılarak oluşturulan döngüye örnek program.

Bu programda SAYAC adlı bir registerin içerisine döngünün tekrar sayısı konulmuştur. Döngünün her tekrarında INCF komutuyla SAYAC içerisindeki değer bir arttırılır. SUBWF komutuyla W register içerisindeki sabit sayı karşılaştırılır. Karşılaştırma sonunda etkilenen zero flag kontrol edilerek döngü sona erdirilir.



```

;===PROG9.ASM===05/06/2000=====
LIST    P=16F84

```

```

INCLUDE "P16F84.INC"

```

```

SAYAC    EQU    h'0C'          ;SAYAC registerini tanımla

```

```

CLRF     PORTB    ;PortB'yi temizle

```

```

BSF      STATUS, 5 ;BANK2'ye geç

```

```

CLRF     TRISB    ;PortB'nin tüm uçlarını çıkış yap

```

```

BCF      STATUS, 5 ;BANK1'e geç

```

```

BASLA

```

```

CLRF     PORTB    ;portB'yi sil

```

```

CLRF     SAYAC    ;SAYAC'ı sil

```

```

TEKRAR

```

```

MOVF     SAYAC, W  ;SAYAC'ı W registere yükle      MOVWF
PORTB    ;W registere PortB'ye gönder            IMCF   SAYAC, F

```

```

;SAYAC ← SAYAC + 1

```

```

MOVLW    h'07'    ;W registere h'07" yükle

```

```

SUBWF    SAYAC, W  ;W ← SAYAC - W

```

```

BTFSS    STATUS, 2 ;STATUS bit2=1 mi?

```

```

GOTO     TEKRAR    ;Hayır, TEKRAR'a git

```

```

DONGU

```

```

GOTO     DONGU

```

```

END

```

İşlem Basamakları

1. Programı PIC'e yazdırıp, deneme katınıza veya breadboard üzerine kurduğunuz devreye yerleştiriniz.
2. PIC'e gerilim uygulayınız.
3. PortB'deki LED'ler birkaç mikrosaniye sonra h'06' sayısını binary olarak gösterecektir.

76543210

00000●●●

Sonuç: PIC'e gerilim uyguladıktan sonra, PortB'de yukarıdaki sayının gösterilmesine kadar geçen süre, döngünün çalışma süresidir. Bu süre döngünün tekrar sayısı ile doğru orantılıdır. MOVLW h'07' satırında W registeri içerisine atanan h'07' sayısı yerine başka bir sayı yazmak suretiyle döngünün tekrar etme sayısı değiştirilebilir.

Yukarıdaki programda, döngü içerisine tekrar edilecek herhangi bir komut yazılmamıştır (Programda verilmek istenen konunun özünü daha karmaşık hale getirmemek için). Eğer bir komutu önceden tespit ettiğiniz sayıda çalışmasını istiyorsanız, bu döngü içerisine yazabilirsiniz.

KENDİNİZ UYGULAYINIZ

SAYAC içerisine ve W register içerisine konulacak sayıları kendiniz tespit ederek LED'lerin binary 9 sayısını

7 6 5 4 3 2 1 0 an programı kendiniz yapınız. LED'ler aşağıdaki gibi yanması gerekir.

○○○○●○○●

ZAMAN GECİKTİRME DÖNGÜLERİ

Bazı işlemlerin yapılması esnasında belirli bir zaman hiçbir şey yapılmadan beklenmesi gerekebilir. Böyle bir beklemenin gerekliliği 7. bölümde görülmüştü. Bir butonun basıldığında/bırakıldığında kontakların birbirine teması esnasında ark meydana geliyordu. Bu arkı önlemek için zaman geciktirme döngüsü gerekiyordu.

Zaman geciktirme işlemlerini yazılım döngüleri kullanarak yapabildiğimiz gibi donanımın (PIC16F84) bize sunduğu özel geciktirme olanaklarını kullanarak da yapabiliriz. Donanım geciktirmelerinin (TMRO ve WDT) nasıl kullanıldığını daha sonraki bölümlerde ele alınacaktır.

Bir zaman geciktirme döngüsünde, gecikme zamanını tespit etmek için komutların icra süresi göz önüne alınır. RC osilatör kullanılan PIC devrelerinde bir komutun icra süresini hassas olarak saptamak mümkün değildir. Ancak kristal veya seramik rezonatör kullanılan devrelerde çok hassas gecikme döngüleri elde etmek mümkündür.

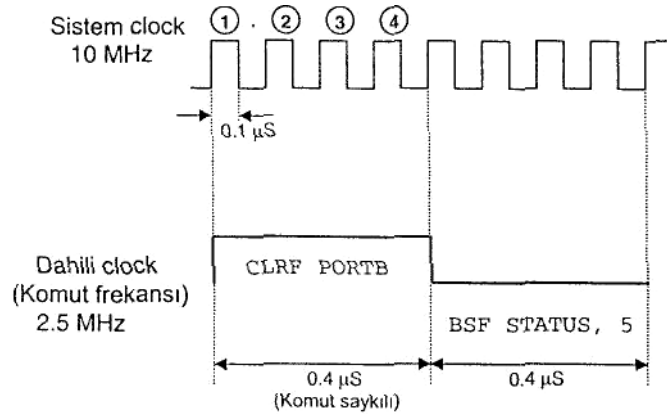
Dahili Komut Saykılı

PIC16F84 mikrodeneleyicisinin girişine uygulanan osilatör frekansı 4'e bölünür. Bir komutun icra süresi bu frekansın bir saykılı süresindedir.

PIC16F84'un osilatör girişine 4 MHz'lik bir kristal osilatör frekans uygulanırsa, PIC16F84 bu frekansı kendi içerisinde 4'e böldüğünde, 1 MHz'lik bir dahili frekans elde edilir. Bu frekansın bir saykılı ($T=1/f=1/1\text{MHz}=1 \mu\text{s}$) bir komutun icra süresidir.

Örneğin, PIC16F84" uygulanan XC osilatörün frekansı 10 MHz ise, CLRF PORTB ve BSF STATUS, 5 komutlarının çalışma sürelerini şöyle gösterebiliriz:

10 MHz'lik frekansın bir saykılı $T=1/f=1/10^6 \text{ Sn}=0.1 \mu\text{s}$ eder. Dahili komut saykılı ise $4 \times 0.1 \mu\text{s}$ eder.



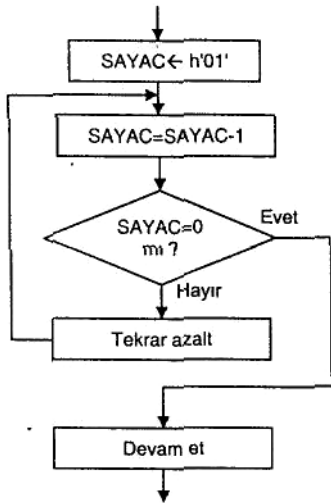
PIC16F84 mikrodeneleyicisinin komutlarından 10 tanesi hariç diğerleri 1 komut saykılı süresinde çalışır. Bu komutlar aşağıdakilerdir.

Komut	Komut saykılı
GOTO	2
CALL	2
RETURN	2
Program counter'a (PC) veri yazan komutlar	2

DECFSZ	1 (Register içersindeki sayı 1'se) 2 (Register içersindeki sayı 0'sa)
RETLVV	2
RETFIE	2
İNCFSZ	1 (Register içersindeki sayı 1'se) 2 (Register içersindeki sayı 0'sa)
BTFSC	1 (Test edilen bit 1'se) 2 (Test edilen bit 0'sa)
BTFSS	1 (Test edilen bit 0'sa) 2 (Test edilen bit 1'se)

Tek Döngü ile Minimum Zaman Geciktirme

Bir gecikme döngüsü hazırlanırken sayaç olarak kullanılacak bir file register tanımlanır. Bu register içersine döngünün tekrar sayısı yüklendikten sonra DECFSZ komutu ile tekrar sayısından her defasında "1" çıkartılır. Çıkarma sonucu "0" olunca döngü sona erdirilir. Şimdi bu şekilde düzenlenen bir döngüde kaç saykıl oluştuğunu akış diyagramını çizerek görelim. SAYAC registeri içersine yüklenen sayı $N=h'01'=d'1$ 'dir.



Komut	Saykıl	
MOVLW h'01'	1	N-1
MOVWF SAYAC	1	
DECFSZ SAYAC, F	1x0+2	N-1
GOTO DONGU	0	
Toplam	4 saykıl	

Tek Döngü ile Maksimum Zaman Geciktirme

SAYAC registeri içersine yüklediğimiz sayıyı h'FF' yaparsak yukarıda akış diyagramını çizdiğimiz gecikme döngüsünden maksimum gecikmeyi sağlarız. SAYAC registeri içersine yüklediğimiz sayı $N=h'FF'=d'255$ ' dir.

Komut	Saykıl	
MOVLW h'FF'	1	N-1
MOVWF SAYAC	1	
DECFSZ SAYAC, F	1x254+2	N-1
GOTO DONGU	2x254	
Toplam	766 saykıl	

Komut Saykıl Sayısının Bulunması

Döngüdeki saykıl sayısı programın başında SAYAC adlı registre atanan (N) sayısına bağlı

olduğu görülmektedir. Öyleyse bir döngünün kaç saykıl gecikme oluşturacağını bulan bir formül oluşturacak olursak;

$$\text{Komut Saykıl Sayısı} = 1+1+1x(N-1)+2+2x(N-1)$$

$$=1+1+N-1+2+2N-2$$

$$\text{KSS} = 3N+1$$

N sayısı büyük bir sayı olduğunda 1 saykıl gecikme döngüsünde çok önemli bir rol oynamayacağı düşünülürse, yukarıdaki formüldeki 1 sayısını iptal edebiliriz. Bu durumda formülü aşağıdaki gibi yazabiliriz.

$$\text{KSS}=3N$$

Örnek: 10 MHz'lik bir osilatörden uygulanan clock frekansı uygulanan PIC16F84'de 100 µS'lik gecikme sağlayan bir döngüde kaç saykıl gecikme olduğunu bulalım.

10 MHZ'lik frekansın bir saykılı $T=1/f=1/10^6$ Sn=0.1 µS eder. Dahili komut saykılı ise $4x0.1\mu S=0.4\mu S$ dir.

	Komut	Saykıl
	MOVLWH'52"	1
	MOVNF SAYAC	1
	NOP	1
	NOP	1
	NOP	1
DONGU	DECFSZ SAYAC, F	1x81+2
	GOTO DONGU	2x81
	Toplam	250 saykıl

Bu döngüde SAYAC içerisine atanan N sayısı h'52' nin desimal sayı karşılığı 82'dir. Buna göre döngünün kaç komut saykılı oluşturduğunu formülle bulacak olursak;

$$KSS=3xN+1= 3x82+1=247 \text{ saykıl}$$

Saykıl sayısını 250'ye getirmek için eklenen üç NOP komut saykılını eklersek,

$$KSS=247+3=250 \text{ eder.}$$

Bu döngünün ne kadar sürelik bir gecikme sağlayacağını bulalım:

$$\text{Gecikme Süresi(GS)}=250*0.4 \mu\text{S}= 100\mu\text{S}$$

N sayısının bulunması

PIC'e 4 MHz'lik frekans uygulandığında bir komut saykılının 1 μS olacağını biliyoruz. Bu durumda N sayısını belirlemek kolaydır. 100 μS 'lik bir gecikme için 100 saykılılık bir gecikme döngüsü oluşturulmalıdır. N sayısını hesaplarsak $N=KSS/3= 100/3=33.3$ olmalıdır. Bu sayı yaklaşık 33 desimal sayıdır. Heksadesimale dönüştürsek, h'21'eder.

Özetlersek, 4 MHz'lik harici clock frekansında 100 μS 'lik gecikme oluşturan döngüde SAYAC registeri içerisine atanan sayı h'21' olmalıdır.

PIC'e uygulanan frekans 4 MHz dan farklı olursa bu defa KSS sayısını bulmak için bir formül üretelim;

PIC'e uygulanan harici frekansa f, dahili komut frekansına f_k dersek;

Dahil komut frekansı(f_k)= f/4 olur. Komut saykılı $T_k=1/f_k$ olur. (Bir komutun çalışma süresi)

f_k yı yerine koyarsak:

$$T_k = \frac{1}{f} = \frac{4}{f} \mu\text{S} \quad KSS = \frac{GS}{\frac{4}{f}} = \frac{GSf}{4} \quad KSS=\text{Gecikme süresi(GS)}/T_k \text{ dır.}$$

$KSS=3xN$ olduğunu daha önce bulmuştuk. Şimdi N'i yalnız bırakarak KSS' nin yukarıda bulduğumuz eşitliğini yerine koyarsak;

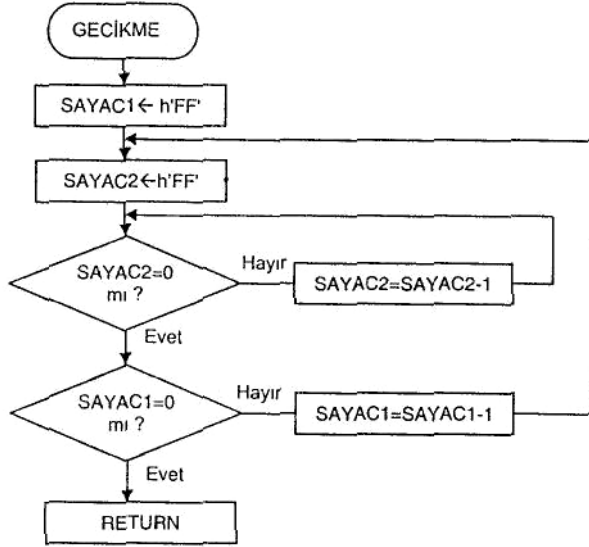
$N=KSS/3$ burada KSS'yi yerine koyalım.

$$N = \frac{\frac{GSf}{4}}{3} = \frac{GSf}{12} \quad \text{neticede } N=GSxf/12 \quad \text{formülüyle N sayısı bulunur.}$$

Örnek: 300 μS 'lik gecikme yapan zaman geciktirme döngüsünde, N sayısı kaç olmalıdır? (PIC'e bir kristal osilatörden 10 MHz'lik frekans uygulanmaktadır.) $GS=300\mu\text{S}$, $f=10\text{MHz}$ $N=GSxf/12$ $N=300x10/12=250$ 250 sayısını heksadesimale dönüştürsek, N=h'FA' olarak buluruz.

Çift Döngülü Zaman Geciktirme

Maksimum gecikme yapan tek döngüde oluşturulan 766 saykılılık süre az gelebilir. Bu durumda iç içe iki veya daha fazla döngü kullanılabilir. Şimdi bu kitaptaki çoğu zaman geciktirme işlemlerinde kullanacağımız çift döngünün akış diyagramını ve programı yapalım.



N Sayısının Hesaplanması

SAYAC1 registerine konulan sayıya M, SAYAC2 registerine yüklenen sayıya da N diyerek programı yazalım.

	Komut		Saykıl
GECIKME	MOVLW	h'FE'	1
	MOVWF	SAYAC1	1
DONGU1	MOVLW	h'FF'	1xM
	MOVWF	SAYAC2	1xM
DONGU2	DECFSZ	SAYAC2,F	1xNxM
	GOTO	DONGU2	2xNxM
	DECFSZ	SAYAC1,F	1Xm
	GOTO	DONGU1	2Xm
	RETURN		2
	Toplam		196608 saykıl

Toplam komut saykıl sayısı M ve N sayılarının 3 katı kadardır. Öyleyse bunu formül biçiminde yazarsak; Yaklaşık olarak **KSS=3xMxN** olur.

Bu geciktirme programında M ve N sayılar h'FF' olduğuna göre elde edilen komut saykıl sayısı maksimum elde edilebilecek olan sayıdır. Bu sayı da,

$KSS=3 \times 256 \times 256 = 196608$ saykıl eder. 1 MHz'lik dahili clock saykılında bu süre $196608 \mu S = 197 \mu S$ (yaklaşık)

Örnek: Çift döngülü bir zaman geciktirme programından 12 μS lik bir gecikmesi isteniyorsa M ve N sayıları ne olmalıdır?

Not: PIC'e harici olarak uygulanan clock frekansı= 4 MHz dir. Bu frekans 4'e bölündüğünde 1 MHz'lik dahili komut frekansı elde edilir. Çözüm:

$KSS=3 \times M \times N$ dir.

M ve N sayılarına aynı sayıları vererek bu programı yapacağımızı düşünelim.

$M=N$ olduğundan yukarıdaki formülü,

$KSS=3 \times M^2$ olarak yazabiliriz.

12 mS= 12000 μS eder. Dahili clock frekansı 1 MHz olduğundan 12000 komut saykılı gerekir.

$12000=3 \times M^2$ $M^2= 2000/3$ $M=63.2$ desimal = h'3F'

M ve N birbirine eşit olduğundan $M=h'3F'$ $N=h'3F'$ olarak bulunur.

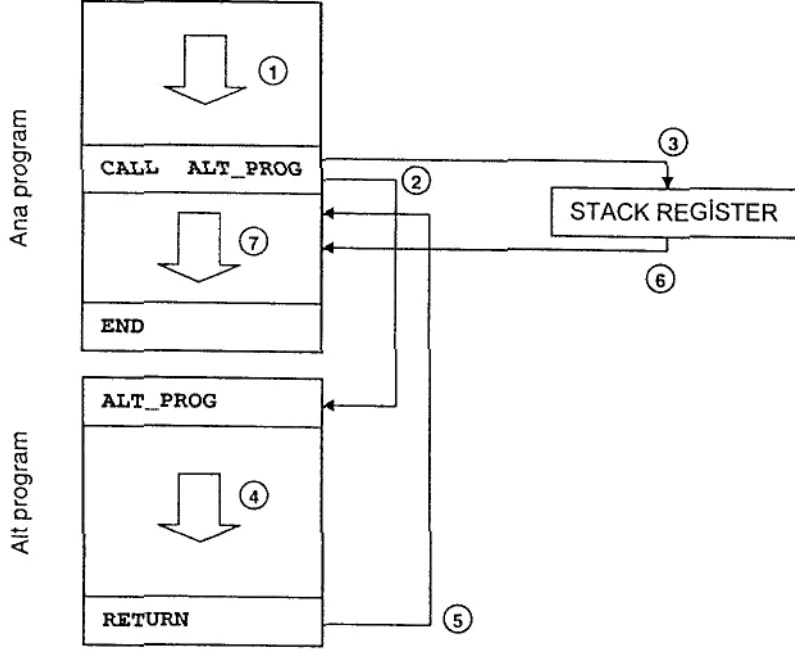
Böylece çift döngülü zaman geciktirme devresinde 12 mS lik zaman gecikmesi için, SAYAC1 ve SAYAC2 registerleri içerisine h'3F' sayısı yüklenmelidir.

ALT PROGRAMLAR

Program içerisinde defalarca tekrar edilmesi gereken program parçaları olabilir. Bu durumda aynı program komutlarını her defasında yazmak hem zor olacak, hem de bellekte çok fazla yer kaplayacaktır. Bu sorunu gidermek amacıyla:

alt programlar bir defa yazılır, kullanılması gerektiğinde alt programın adı CALL komutundan sonra yazılarak çağrılır. Bu andan itibaren programın akışı alt programa geçer. Alt program komutları bittiğinde en sonuna yazılan RETURN komutu vasıtasıyla ana programa dönülür ve kaldığı yerden devam eder.

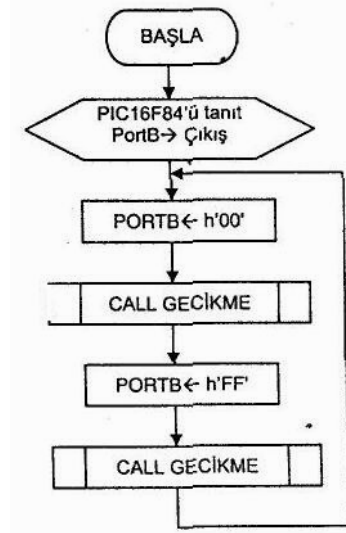
Bir alt programın çalışmasını şematik olarak aşağıdaki gibi gösterebiliriz.



CALL ALT_PROG komutuyla alt program çağrılarak çalıştırılmaya başlandığı anda, programın tekrar geri döneceği adres bilinmesi gerekir. İşte STACK REGİSTER bu adresin yazıldığı özel bir registerdir. Programcının bu registre adresin nasıl yazıldığı ile ilgilenmesi gerekmez. Çünkü assembler programı bu işlemi otomatik olarak yapar. Eğer çağrılan alt program içerisinde başka bir alt program daha çağrıldıysa bu adresler stack register içerisinde ardı ardına yazılır. Bu nedenle registre stack (yığın) adı verilmiştir. Alt programlardan RETURN komutuyla geri dönülürken en son yazılan adres ilk önce işlem görür.

Şimdi yukarıda verdiğimiz örnekteki işlem sırasını açıklayalım:

1. Ana programın ilk komutundan itibaren program çalışmaya başlar.
2. CALL_ALT_PROG (Programın akışı ALT_PROG adlı alt programa geçer)
3. Ana programdan çıkıldığı andaki adres STACK REGİSTER'e yazılır.
4. Alt program komutları çalışır.
5. RETURN, Alt program komutlarının bittiğini gösterir. Program akışını ana programda kaldığı yerdeki adrese gönderir.
6. STACK registerde yazılı bulunan aynı adres alınır.
7. Ana program kaldığı yerden itibaren çalışmaya başlar ve END komutu ile sona erer.



PROGRAM-10) Zaman gecikme döngüsü kullanarak PortB'ye bağlı olan tüm LED'leri belirli zaman aralıklarıyla yakıp-söndüren program.

```

;==PROG10.ASM==07/06/2000==
LIST P=16F84
INCLUDE "P16F84.IHC"
SAYAC1 EQU h'0C'
SAYAC2 EQU h'0D'
CLRF PORTB
BSF STATUS, 5
CLRF TRISB
BCF STATUS, 5 TEKRAR
MOVLW h'00'
MOVWF PORTB
CALL GECİKME
MOVLW h'FF'
MOVWF PORTB
CALL GECİKME
GOTO TEKRAR

```

```

GECIKME           ;Alt program başlangıcı
    MOVLW          h'FF'
    MOVWF          SAYAC1
DONGU1
    MOVLW          h'FF'
    MOVWF          SAYAC2
DONGU2
    DECFSZ         SAYAC2, F
    GOTO           DONGU2
    DECFSZ         SAYAC1, F
    GOTO           DONGU1
    RETURN
END

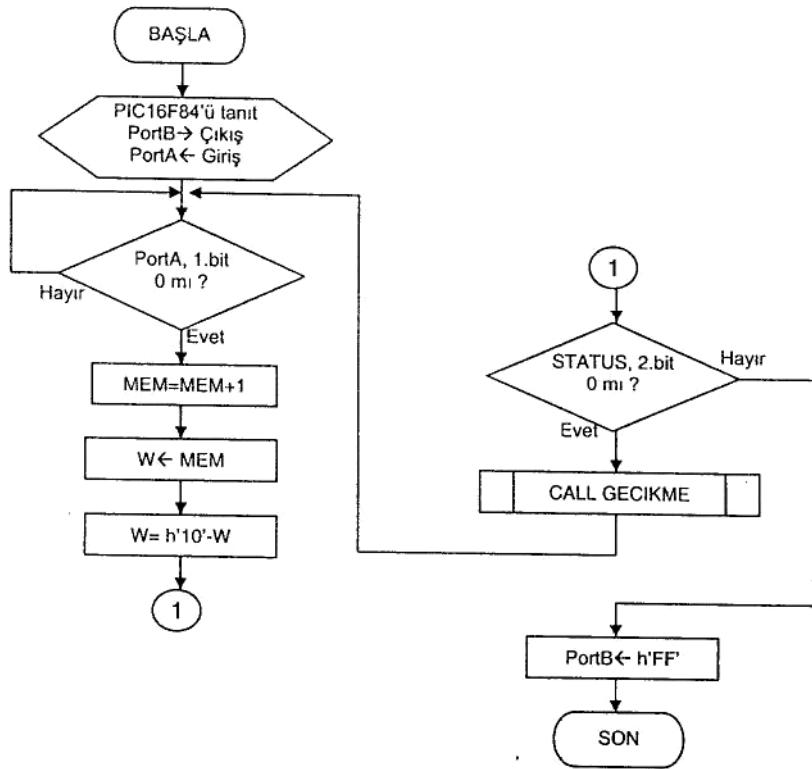
```

İşlem Basamakları

1. PIC'i programlayarak deneme kartınıza veya breadboard üzerine kurduğunuz devre üzerine yerleştiriniz.
2. Deneme setine gerilim uygulayarak PortB'deki tüm LED'lerin belirli aralıklarla yanıp söndüğünü görünüz.
3. SAYAC1 ve SAYAC2 değişkenleri içerisine atanan sayılan değiştirerek LED'lerin 0.5 sn aralıklarla yanıp-sönmesini sağlayınız.

PROGRAM-11) PortA'nın 1. Numaralı ucuna bağlı butona 10 defa bastıktan sonra PortB'deki tüm LED'leri yakan program.

Bu programı 7. bölümde döngüleri anlatırken yaptığımız örnekte uygulamıştık. Ancak A portuna bağlı butonda meydana gelen arki söndürmek için düzenlediğimiz döngü bunu engelleyemediğini gördünüz (Gecikme zamanı çok kısa olduğu için.) Şimdi gecikme zamanını istediğimiz kadar ayarlayabildiğimiz program döngülerini öğrendiğimize göre, bu işlemi tam olarak gerçekleştiren PIC'i programlayalım.



```

;==PROG11.ASM==08/06/2000=====
LIST    P=16F84
INCLUDE  "P16F84.IMC"
SAYAC1 EQU  h'0C'
SAYAC2 EQU  h'0D'
MEM EQU     h'0E'
CLRF     PORTB    ;PortB'yi sıfırla
BSF      STATUS, 5    ;BANK2'ye geç
CLRF     TRISB     ;PortA'nın 1. Bit'i giriş

```

```

BSF    TRISA, 1      ;portB'nin uçları çıkış
BCF    STATUS, 5     ;BANK1'e geç
CLRF   MEM           ;MEM registerini sıfırla
TEKRAR
BTFSC  PORTA, 1      ;PortA'nın 1.bit'i 0 mı?
GOTO   TEKRAR        ;Hayır, tekrar test et
INCF   MEM           ;Evet, MEM=MEM+1
MOVF   MEM, W        ;W←MEM
SUBLW  d'10'         ;W = d'10' - W
BTFSC  STATUS, 2     ;STATUS'un 2.bit'i 0 mı ?
GOTO   YAK           ;Hayır, Z=1
CALL   GECIKME        ;Evet, buton arkının sönmesini bekle
GOTO   TEKRAR        ;Butonu test için başa git YAK
MOVLW  h'FF'         ;W←h'FF'
MOVWF  PORTB         ;PortB'deki tüm LED'leri yak DONGU
GOTO   DONGU
;=====; Gecikme alt programı.=====
GECIKME
MOVLW  h'FF'
MOVWF  SAYAC1
DONGU1
MOVLW  h'FF'
MOVWF  SAYAC2
DONGU2
DECFSZ SAYAC2, F
GOTO   DONGU2
DECFSZ SAYAC1, F
GOTO   DONGU1
RETURN
END

```

İşlem Basamakları

- 1-) PIC'i programlayarak deneme kartınıza yerleştiriniz.
- 2-) Deneme kartına gerilim vererek PIC'i çalıştırınız.
- 3-) A1 butonuna belirli aralıklarla 10 defa basınız. Butona 10. defa bastığınızda portB'deki tüm LED'lerin yandığını görünüz.

9

BİT KAYDIRMA ve MANTIKSAL İŞLEM KOMUTLARI

- ☐ SOLA KAYDIRMA (RLF)
- ☐ SAĞA KAYDIRMA (RRF)
- ☐ COMF ve SWAPF KOMUTLARI

SOLA KAYDIRMA (RLF)

RLF komutu, belirlenen bir file register içerisindeki bit'lerin pozisyonunu her defasında bir sola kaydırmak için kullanılır. Register içerisindeki bit'ler sola kaydılda MSB bit'i, STATUS registerde bulunan carry flag içerisine yazılır. Carry flag içeriği ise registerin LSB bit'ine yazılır.

NOT: MSB (Most Significant Bit- Değerliği en yüksek bit, yani en soldaki bit.)

LSB (Least Significant Bit- Değerliği en düşük bit, yani en sağdaki bit.)

RLF komutunun yazılışı aşağıdaki gibidir.

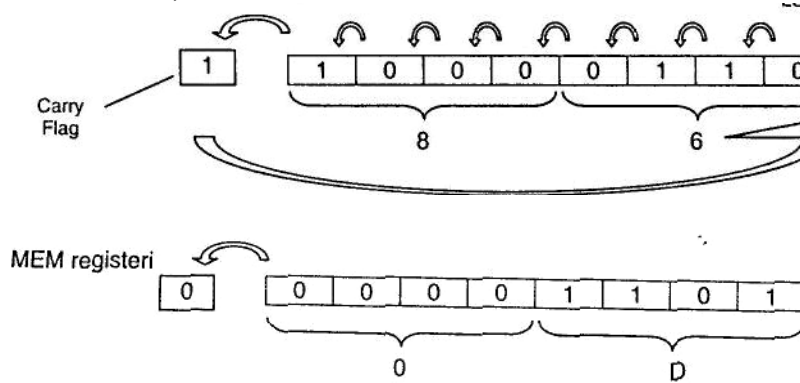
RLF File Register d Destination (Gideceği yer) W veya F

Destination W ise kaydırma sonucunda elde edilen bit paterni W registre, F ise file registre yazılır. örneğin MEM adındaki bir file register içeriği h'86" ise RLF komutu çalıştırıldığında MEM registerinin içeriğinin ne olduğunu şematik olarak gösterelim. H'86'=b'10000110'

MOVLW h'86'

MOVWF MEM

RLF MEM, F MSB



Bazen önceden belirlenen bir bit paternini devamlı olarak kaydırmak gerekebilir. Bu işlem PIC'in portlarına bağlı olan bir cihazı çalıştırmak için ya da portlara bağlı lambalarda ışık gösterisi yapmak için kullanılabilir. Şimdi bu işleme bir örnek yapalım.

PROGRAM-12) PortB'ye bağlı 8 LED üzerindeki bir LED'in yanışını belirli aralıklarla kaydıran (LED0'dan LED7'ye doğru) program. Yanarak kayan LED en sona geldiğinde tüm LED'ler sönmük kalır.

;=====PROG12.ASM=====09/06/2000=====

LIST P=16F84

INCLUDE "P16F84.IHC"

SAYAC1 EQU h'0C'

SAYAC2 EQU h'0D'

CLRF PORTB

;PortB'yi sıfırla

BCF STATUS, 0

;Carry flag'ı sıfırla

BSF STATUS, 5

;BANK1'e geç.

CLRF TRISB

;PortB'nin tüm uçları çıkı.ç

BCF STATUS, 5

;BANK0'a geç.

MOVLW h'01'

;b'00000001' sayısını W'ye yükle

MOVWF PORTB

;W registerini PortB'ye yükle

TEKRAR

CALL GECIKME

;Gecikme yap

RLF PORTB, F

;PortB'deki veriyi sola kaydır.

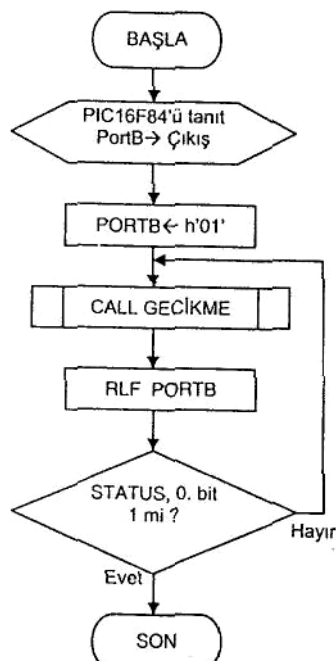
BTFSS STATUS, 0

;Carry flag 1 mi?

GOTO TEKRAR

;Hayır,

DONGU




```

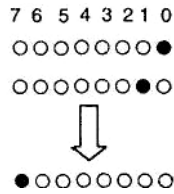
GOTO DONGU ;Evet, Sonsuz döngüye gir. GECIKME ;Gecikme alt programı
MOVLW h'FF'
MOVWF SAYAC1
DONGU1
MOVLW h'FF'

MOVWF SAYAC2
DONGU2
DECFSZ SAYAC2, F
GOTO DONGU2
DECFSZ SAYAC1, F
GOTO DONGU1
RETURN
END

```

İşlem Basamakları

1. PIC'i programlayınız. Deneme kartınıza yerleştirerek gerilim uygulayıp devreyi çalıştırınız.
2. En soldaki bit'ten başlayarak sağa doğru yanan LED'i izleyiniz. Program bittiğinde tüm LED'ler sönmük kalacaktır. Programı tekrar çalıştırmak için RES tuşuna basınız.



SAĞA KAYDIRMA

RRF komutu, belirlenen bir file register içerisindeki bit'lerin pozisyonunu her defasında bir sağa kaydırmak için kullanılır. Register içerisindeki bit'ler sağa kaydırdığında LSB bit'i, STATUS registerde bulunan carry flag içerisine yazılır. Carry flag içeriği ise registerin MSB bit'ine yazılır.

RRF komutunun yazılışı aşağıdaki gibidir.

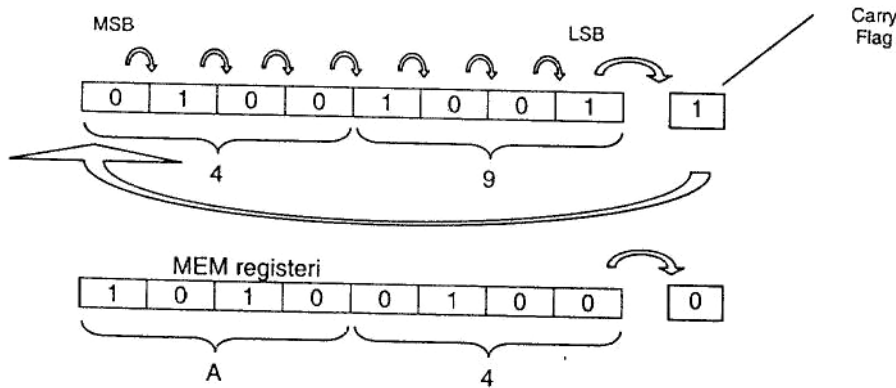
RRF Destination (Gideceği yer) W veya F

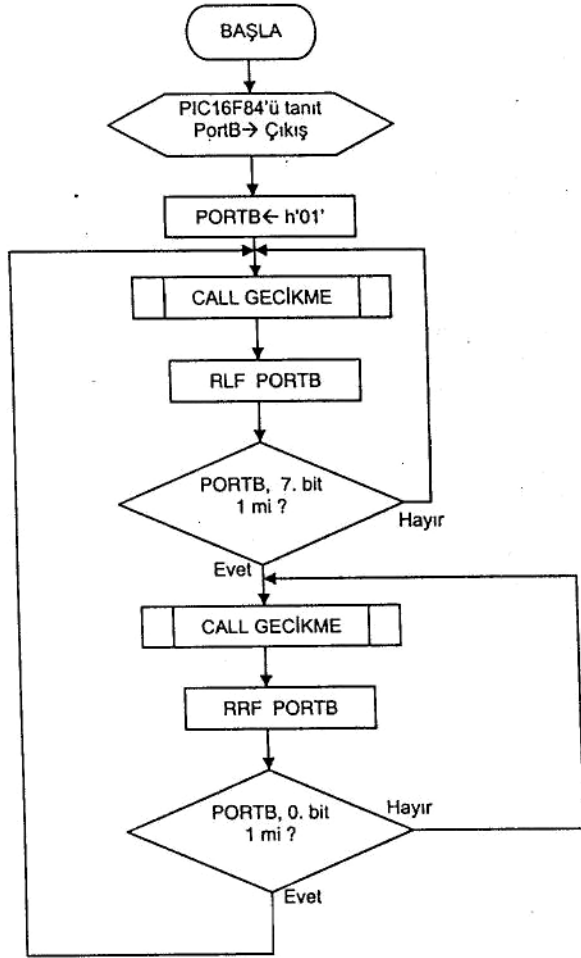
Destination W ise kaydırma sonucunda elde edilen bit paterni W registre, F ise file registre yazılır. Örneğin MEM adındaki bir file register içeriği H'49' ise RLF komutu çalıştırıldığında MEM registerinin içeriğinin ne olduğunu şematik olarak gösterelim. H'49'= b'01001001' dir.

```

MOVLW h'49'
MOVWF MEM
RKF MEM, F

```





PROGRAM-13) PortB'ye bağlı olan 8 LED üzerinde bir LED'in yanışını sağa-sola kaydıran ve bu işlemi devamlı olarak yaptıran program. (Bu programın adını çok eskiden ünlü bir TV dizisindeki otomobilden alarak "Karasimsek" diyoruz. Bu otomobilin de önündeki LED'lerin yanışı sağa-sola kayarak yanıyordu,); Bu nedenle programa kısaca karasimsek programı diyebilirsiniz.

=====PROG13.ASM=====12/06/2000=====

```

LIST    P=16F84
INCLUDE "P16F84.INC"
SAYAC1 EQU    h'0C'
SAYAC2 EQU    h'0D'
CLRF    PORTB
BCF     STATUS, 0 ;Carry flag'ı sıfırla
BSF     STATUS, 5
CLRF    TRISB
BCF     STATUS, 5
MOVLW   h'01'      ;b'00000001' sayısını W'ye yükle
MOVWF   PORTB      ;W registerini PortB'ye yükle
SOL     CALL    GECIKME ;Gecikme yap
        RLF     PORTB, F ;PortB'deki veriyi sola kaydır.
        BTFSS   PORTB, 7 ;PortB 7. Bit 1 mi?
        GOTO    SOL    ;Hayır, sola kaydır.
SAG     CALL    GECIKME ;Gecikme yap
        RRF     PORTB, F ;PortB'deki veriyi sağa kaydır.
        BTFSS   PORTB, 0 ;PortB 0. Bit 1 mi?
        GOTO    SAG    ;Hayır, sağa kaydır
        GOTO    SOL    ;Evet, sola kaydır.
        GECIKME      ;Gecikme alt programı
        MOVLW    h'FF'
        MOVWF    SAYAC1
DONGU1  MOVLW    h'FF'
        MOVWF    SAYAC2
DONGU2  DECFSZ   SAYAC2, F

```

```

GOTO DONGU2
DECFSZ SAYACI, F
GOTO DONGU1
RETURN
END

```

İşlem Basamakları

1. PIC'i programlayıp, deneme kartı üzerine yerleştirerek çalıştırınız.
2. Yanan LED'in önce en sola ulaştıktan sonra bu defa sağa doğru kaydığını ve bu işlemin devamlı olarak tekrar ettiğini görünüz.

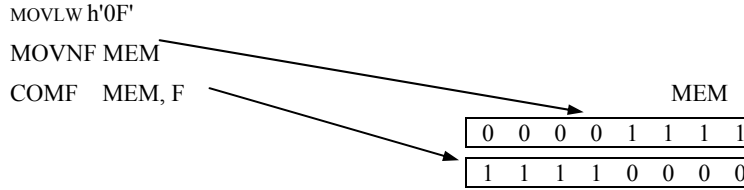
Yanan LED'in en sola ulaştığında PORTB'nin 7. Bitinin "1" olup olmadığı kontrol edildiğine dikkat ediniz. Yanan LED en sağa ulaştığında da 0. bit kontrol edilmektedir.

COMF VE SWAPF KOMUTLARI

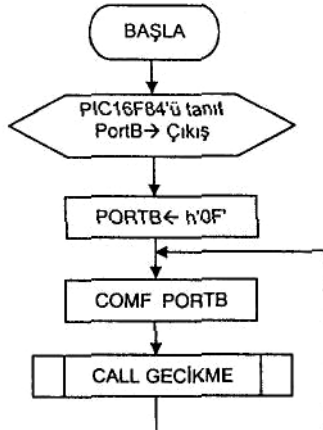
COMF komutu, seçilen bir file register içerisindeki bit'leri tersine (complement) çevirir. Yani 1'ler 0, 0'lar 1 olur. Yazılışı aşağıdaki gibidir.

COMF File register d Destination (Gideceği yer) W veya F

Destination olarak W ise terslenen veri W registerine, F ise file register yazılır. Örneğin MEM 'adını verdiğimiz bir register içerisinde H'oF' verisi yerleştirilmişse, COMF komutu çalıştırıldığında oluşan durumu şematik olarak aşağıdaki gibi gösterebiliriz.



PROGRAM-14) PortB'deki LED'leri dönüşümlü olarak ilk önce ilk dört bit'indeki, sonra da son dört bit'indeki LED'leri yakan program.



```

=====PROG14.ASM=====14/06/2000=====
LIST P=16F84
INCLUDE "P16F84.INC"
SAYAC1 EQU h'0C'
SAYAC2 EQU h'0D'
CLRF PORTB
BSF STATUS, 5
CLRF TRISB
BCF STATUS, 5
MOVLW h'0F' ;b'00001111' sayısını W'ye yükle
MOVWF PORTB ;w registerini PortB'ye yükle

TERSLE
COMF PORTB, F ;PortB'deki veriyi tersle
CALL GECIKME ;Gecikme yap
GOTO TERSLE

```

```

GECIKME          ;Gecikme alt programı
    MOVLW h'FF'
    MOVWF SAYAC1
DONGU1
    MOVLW h'FF'
    MOVWF SAYAC2
DONGU2
    DECFSZ SAYAC2, F
    GOTO DONGU2
    DECFSZ SAYAC1, F
    GOTO DONGU1
    RETURN
    END

```

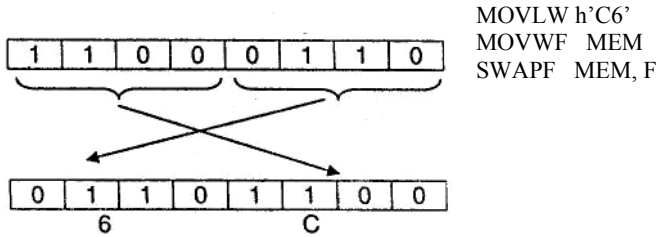
İşlem Basamakları

1. Programı PIC'e yazdırdıktan sonra deneme kartına takıp, gerilim uygulayınız.
2. PortB'deki LED'lerin dönüşümlü olarak ilk önce ilk dört tanesi daha sonra son dört tanesinin aralıklarla yandığını görünüz.

SWAPF komutu, seçilen bir file register içerisindeki verinin ilk dört bit'iyle son dört bit'ini yer değiştirir. Yazılışı aşağıdaki gibidir.

SWAPF File Register d Destination (Gideceği yer) W veya F

MEM adlı bir file register içerisine h'C6' verisi yerleştirilirse SWAPF komutu çalıştırıldığında oluşan durumu şematik olarak aşağıdaki gibi gösterebiliriz.



MANTIKSAL İŞLEM KOMUTLARI

Mantıksal işlem komutları W veya bir file register içerisinde istenilen bir veya birkaç bit'in değerini bir komut saykılında değiştirmek ya da test etmek amacıyla kullanılırlar. Bu komutlar ANDLW, ANDWF, IORLW, IORWF, XORLW, XORWFdir.

ANDLW Komutu (İstenilen bir ya da birkaç bit'i "0" yapmak)

W register içeriğini istenen bir sabit veri ile AND'ler., elde edilen sonucu W registre yazar. Bu komut W register içerisinde istenilen bir veya birkaç bit'in değerini "0" yapmak için kullanılır. Komutun yazılışı aşağıdaki gibidir.

ANDLW Sabit

NOT: Bir veriye AND işlemi uygulamaya "Maskeleme" denilir. Yani bit "0" ile AND'lenirse o bit'in değeri "0" olur. Diğerleri aynen kalır.

"1" ile AND'leme bit'lerin değerini değiştirmez.

"0" ile AND'lemede bit'in değeri "0" olur (Maskeleme).

Örneğin, W register içerisindeki veri b'11010111' ise 2. ve 7. bit'in değerini "0" yapmak için b'01111011' sabiti ile AND'lemek gerekir. Şimdi bunun nasıl yapıldığını şematik olarak gösterelim.

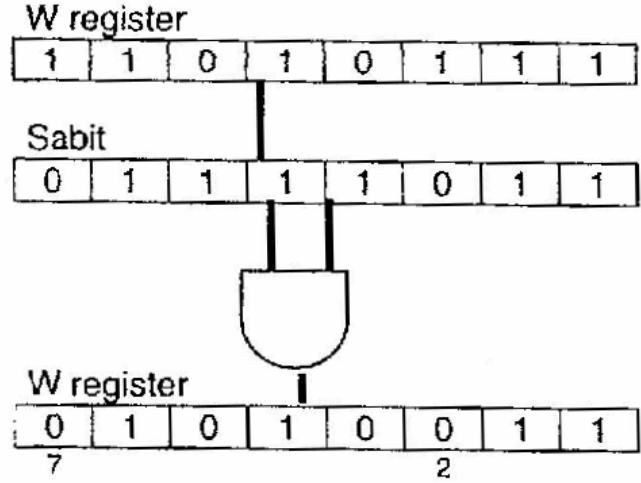
```

    MOVLW b'11010111
    ANDLW b'01111011
    MOVWF PORTB
DONGU
    GOTO DONGU

```

AND mantığı

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



Yukarıdaki işlemi BCF komutu ile yapmak isteseydik, birden fazla BCF komutu kullanmak gerekirdi. Halbuki, ANDLW komutuyla aynı anda (Bir komut saykılında) 2 bit'in değeri değiştirebildik.

ANDWF Komutu

Seçilen bir fi'le register ile W register içeriğini AND'ler, sonuç W veya F register içerisine yazılır. Komutun yazılışı aşağıdaki gibidir.

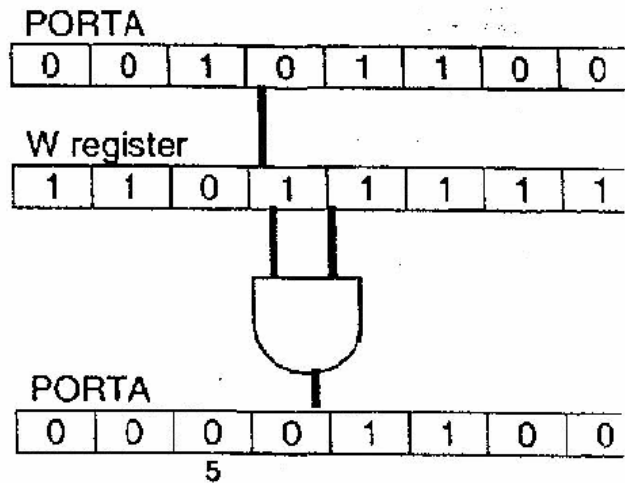
ANDWF File Register d Destination (Gideceği yer) W veya F

Örneğin, PORTA registerinin içeriği b'00101100' olsun. W registerin içerisine b'l 1011111' verisi yerleştirerek 5. bit'in içeriğini "0" yapan ve PORTB'ye gönderen komutları yazalım.

```
MOVLW  b'1011111
ANDWF  PORTA, W
MOVWF  PORTB
DONGU
GOTO   DONGU
```

AND mantığı

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



IORLWV Komutu (İstenilen bit'in değerini "1" yapmak)

W registerin içeriğini istenen bir sabit veri ile OR'lar, elde edilen sonucu V registre yazar. Komutun yazılışı aşağıdaki gibidir.

IORLW Sabit

NOT: Bu komut W register içerisindeki 8 bit'lik bir verinin istenen bir veya birkaç bit'inin değerini "1" yapmak için kullanılır.

"0" ile OR'lamada bit'lerin değeri değişmez.

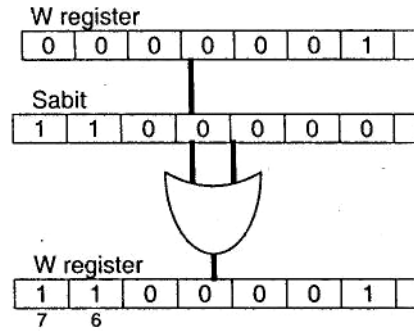
"1" ile OR'lamada bit'in değeri "1" olur.

Örneğin, W register içerisindeki veri b'00000011' ise, bu verinin 6. ve 7. bit'indeki verileri de "1" yapmak için b'11000000' sabiti ile OR'lamak gerekir. B işlemi şematik olarak gösterelim.

MOVLW b'00000011'
IORLW b'11000000'
MOVWF PORTB

OR mantığı

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



IORWF Komutu

Seçilen bir file register ile W registerin içeriğini OR'lar, sonucu W registre ya da file registre yazar. Komutun yazılışı aşağıdaki gibidir.

IORWF Gideceği yer W veya F

XORLW Komutu (İstenilen bir bit'i terslemek)

W register içeriğini istenen bir sabit veri ile XOR'lar, elde edilen sonucu W register içerisine yazar. Komutun yazılışı aşağıdaki gibidir.

XORLW

NOT: Bu komut W registeri içerisinde istenilen bir veya birkaç bit'in değerini tersler, yani "1" ise "0", "0" ise "1" yapar.

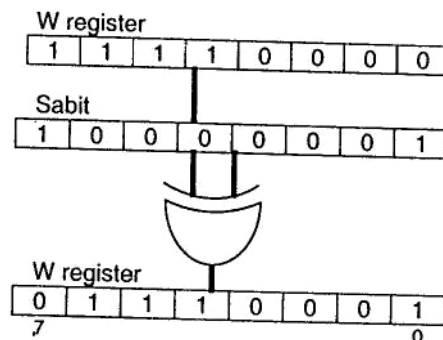
"0" ile XOR'lamada bit'lerin değeri değişmez. "1" ile XOR'lamada bit'in değerinin tersi elde edilir.

Örneğin W registeri içerisindeki veri b'11110000' ise, bu verinin 0. ve 7. bit'lerini terslemek için b'10000001' sabiti ile XOR'lamak gerekir. Bu işlemi şematik olarak gösterelim.

MOVLW b'11110000'
XORLW b'10000001'
MOVWF PORTB

XOR mantığı

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



XORWFKomutu

Seçilen bir file register ile W registerinin içeriği XOR'lanır. Sonuç W veya F register içerisine yazılır. Komutun yazılışı aşağıdaki gibidir.

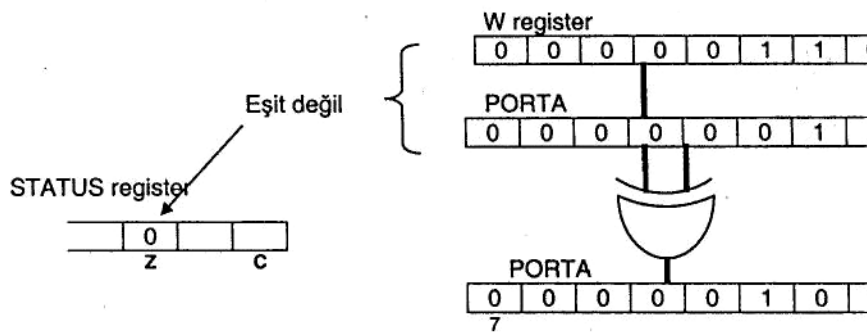
XORWF File Register d Destination (Gideceği yer) W veya F

Bir Byte'lık İki Veriyi Karşılaştırmak (XORLW, XORWF)

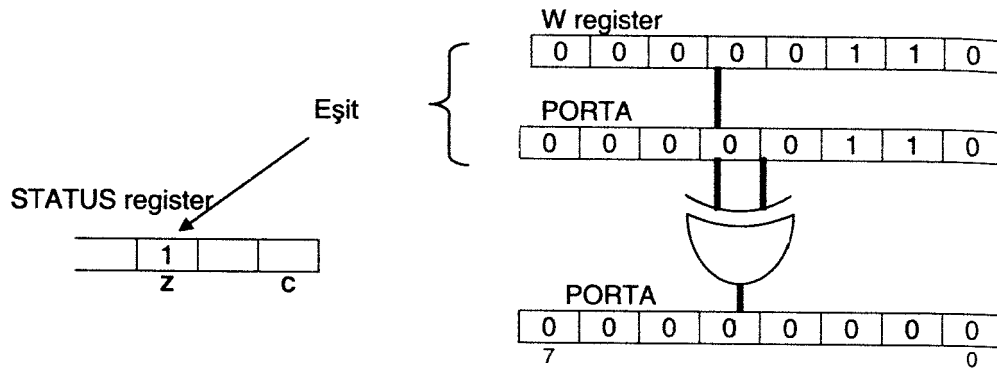
W register içindeki bir byte'lık veriyi istenilen bir byte'lık sabit veri ile aynı olup olmadığını test etmek amacıyla XORLW komutu kullanılır. Bir file register içerisindeki veriyi W register içindeki veriyle karşılaştırmak için de XORWF komutu kullanılır. Veriler aynı ise XOR'lama sonucunda Z flag "1" olur. Veriler aynı değilse Z flag "0" olur.

Örneğin, PORTA registerinin içerisindeki verinin b'00000110' verisi ile aynı olup olmadığını kontrol etmek için aşağıdaki komutları yazmak gerekir.

```
MOVLW b'00000110'  
TEST_PORTA  
XORWF PORTA, F  
BTFSS STATUS, 2  
GOTO TEST_PORTA DEVAM  
MOVF PORTA  
MOVWF PORTB
```



PORTA'nın içerisindeki veri W registerinin içerisine yerleştirdiğimiz veri aynıysa bu defa Z flag "1" olur.



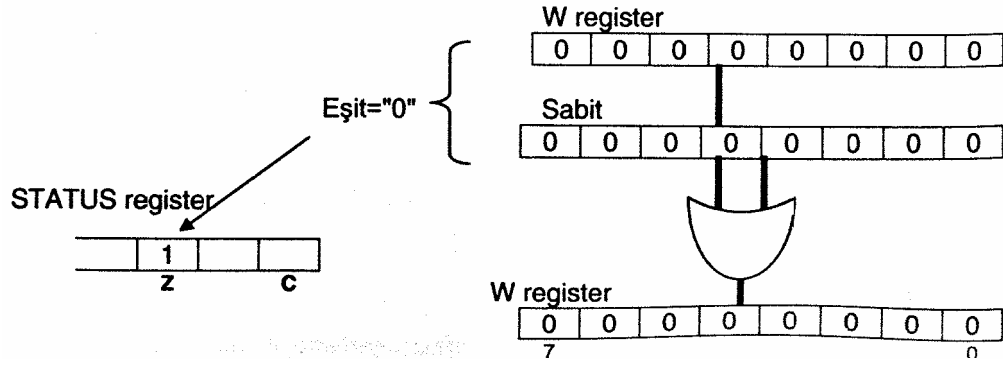
Bir Byte'lık Veriyi "0" ile Karşılaştırmak (IORLW, IORWF)

W register içindeki bir byte'lık verinin "0" olup olmadığını test etmek için "0" sabit verisiyle OR'lanır. Bu işlem için IORLW komutu kullanılır.

Bir file register içerisindeki verinin "0" olup olmadığını kontrol etmek için, W register içerisine "0" verisi atandıktan sonra bu ikisi OR'lanır. Bu işlem için IORWF komutu kullanılır. OR'lama sonucunda karşılaştırılan byte'lar aynıysa, yani içerikleri "0" sa, Z flag "1" olur.

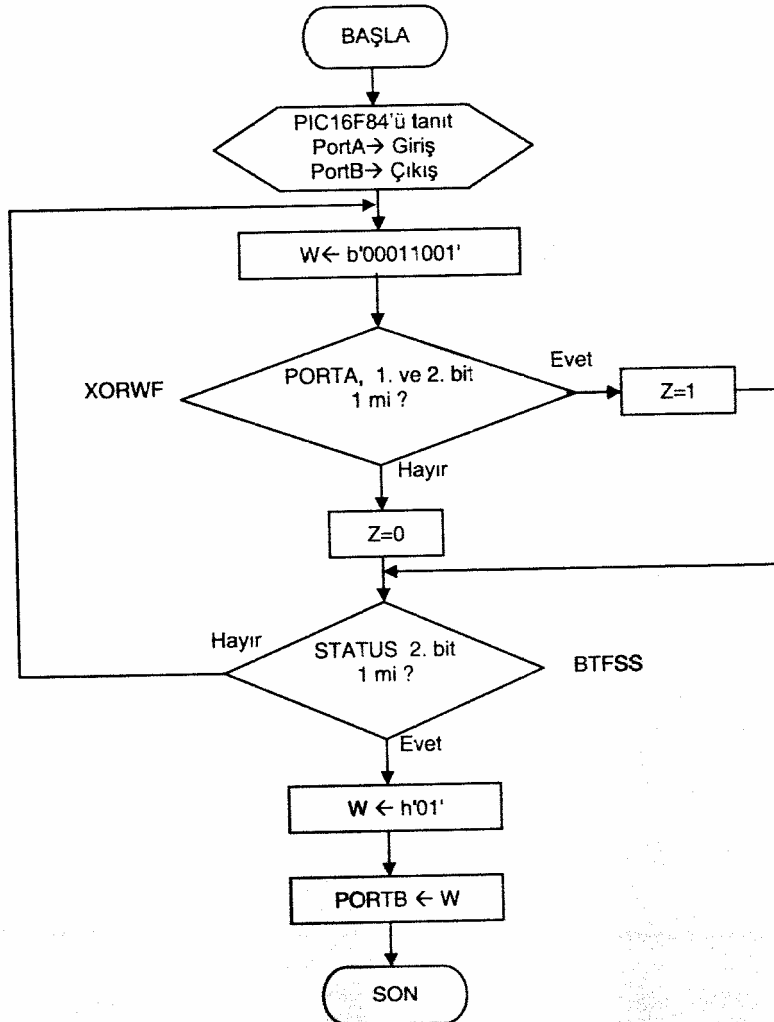
Örneğin, W register içeriğinin "0" olup olmadığını kontrol etmek için aşağıdaki komutları yazmak gerekir.

```
MOVLW b'00000000'  
TEST_W  
IORLW b'00000000'  
BTFSS STATUS, 2  
GOTO TBST_W DEVAM
```



PROGRAM-15) PortA'nın 1. ve 2. bit'lerine bağlı olan A1 ve A2 butonlarının her ikisi birden basılı olduğunda PortB'nin 0. bit'ine bağlı LED'i yakan program.

NOT: PortA'nın girişleri pull-up yapıldığından butonlara basılmadığında girişler "1" dir. PIC16F84'ün portA girişlerinden sadece 5 tanesi kullanıldığından PortA'daki 8 bitlik verinin değerliği en yüksek son üç bitinin değeri daima "0" dır. Bu nedenle A1 ve A2 butonlarının her ikisi birden basılı olduğunda portA'daki veri '00011001' olur.




```

;==PROG15 -ASM====17/06/2000=====
LIST P=16F84
INCLUDE "P16P84.INC"
CLRF PORTB ;PortB'yi sil
BSF STATUS, 5 ;BANK1'e geç
MOVLW h'FF' ;W←h'FF'
MOVWF TRISA ;PortA'nın tüm uçları giriş
CLRF TRISB ;PortB'nin tüm uçları çıkış
BCF STATUS, 5 ;BANK0'a geç
TEST_PORTA
MOVLW b'00011001' ;←b'00011001'
XORWF PORTA, W ;W←PORTA XOR W
BTFSS STATUS, 2 ;Z flag 1 mi?
GOTO TEST_PORTA ;Hayır, PortA'yı tekrar test et
YAK
MOVLW h'01' ;Evet, W←h'01'
MOVWF PORTB ;PortB'nin 0.LED'ini yak
DONGU
GOTO DONGU

```

10

ARİTMETİK İŞLEMLER

- ❑ ARİTMETİK İŞLEM KOMUTLARI
- ❑ 8 BİT TOPLAMA
- ❑ 16 BİT TOPLAMA
- ❑ 8 BİT ÇIKARMA
- ❑ 16 BİT ÇIKARMA

ARİTMETİK İŞLEM KOMUTLARI

Bazı PIC16 uygulamalarında aritmetik işlemler gerekebilir. 8 bit'lik aritmetik işlemler yapmak oldukça kolaydır. Aritmetik işlemlerin mantığını basitçe anlatabilmek için negatif sayıların toplama veya çıkarmasını bu kitapta ele almayacağız. Aritmetik işlemler için gerekli komutlar aşağıda verilmiştir.

	KOMUT	AÇIKLAMA
ADDLW	Sabiti W registerden çıkarır.	
ADDWF	W registerle F(file) registerini toplar.	
SUBLW	Sabitten W registerini çıkarır.	
SUBWF	F registerden W registerini çıkarır.	
RLF	F register içerisindeki bit'leri bir sola kaydırır. (Bit'lerin dönüşü Carryflag aracılığı ile olur.)	
RRF	F register içerisindeki bit'leri birsağa kaydırır. (Bit'lerin dönüşü Carryflag aracılığı ile olur.)	

PIC16, 0-255 arasındaki 8 bit'lik sayıları pozitif sayılar olarak ele alır. Negatif bir sayı 2'nin complement'i (tersi) alınması suretiyle ifade edilerek toplama veya çıkarması yapılabilir. Ancak bu işlem, kitabımızın hedeflediği teknik bilginin üzerindedir. Bu nedenle sadece 8 ve 16 bit'lik pozitif sayıları toplama çıkarma ve çarpma işlemlerini öğrenerek mikro denetleyicilerle yapabileceğiniz oldukça kullanışlı programlar yapabileceksiniz.

0-255 sayıları arasındaki 8 bit'lik toplamada, carry flag elde edilen sonucun 8 bit'in dışına taşıp taşmadığını gösterir. Eğer, Carry Flag=0 ise, sonuç 8 bit'ten fazla değildir.

Carry Flag=1 ise, sonuç 8 bit'ten fazladır. Yani taşma vardır.

0-255 sayıları arasındaki 8 bit'lik çıkarmada, carry flag, elde edilen sonucun negatif veya pozitif olduğunu gösterir. Eğer, Carry Flag=1 ise, sonuç pozitifdir. Carry Flag=0 ise, sonuç negatiftir.

Çıkarma işleminde Carry Flag (Taşma bayrağı) gerçekten bir taşma işlemini göstermediğine dikkat ediniz. PIC16'nın STATUS registerinde ödünç alma (Borrow Flag) olmadığından, Carry Flag'ın "0" olma durumunu ödünç alma olarak değerlendirilir. Bu konuyu 16 bit çıkarma işlemini anlatırken daha iyi anlayacaksınız.

8 - BIT TOPLAMA

Bir byte'lık iki sayı iki şekilde toplanabilir. ilki, W register sabit bir sayı ile toplanır(ADDLW), sonuç W registre yazılır. ikincisi, W registerle bir file register içeriği toplanır(ADDWF), sonuç W veya F registre yazılır.

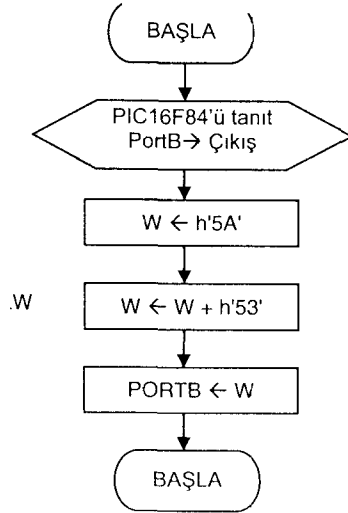
ADDLW h'02' ; h'02' saya-sı W registre eklenir.

ADDWF MEM, F ; W ile MEM toplanıp, MEM'e yazılır.

Heksadesimal sayı toplama örnekleri:

	h'06'	h'02'	h'03''	h'04'	h'FF'
	+ h'01'	+ h'FD'	+ h'FD'	+ h'FD'	+ h'01'
Sonuç	h'07'	h'FF'	h'00'	h'01'	h'00'
CarryFlag	0	0	1	1	1

İki sayının toplamından elde edilen sonuç h'FF' den büyükse, taşma meydana gelecek ve Carry Flag "1" olduğunda ne yapmak gerektiğini 16 bit toplamada anlatacağız.



PROGRAM-16) W register içerisindeki h'5A' sayısıyla, h'53' sayısını toplayıp, sonucu B portundaki LED'lerde gösteren program. /-----s.

```

;==PROG16.ASM==17/06/2000=====
LIST P=16F84
INCLUDE "P16F84.IMC"
CLRF PORTB
BSF STATUS, 5
CLRF TRISB
BCF STATUS, 5
MOVLW h'5A'
ADDLW h'53'
MOVWF PORTB
DONGU
GOTO DONGÜ
END

```

H'5A'+h'53'=h'AD'

01011010	→	h'5A'
+ 01010011	→	h'53'

10101101	→	h'AD'

0

Carry Flag

İşlem Basamakları

1. PIC16F84'u programlayarak deneme devrenizin üzerine yerleştirip, gerilim uygulayınız.
2. PortB'deki LED'lerin "••••••••" biçiminde yandığını görünüz

NOT: Deneme kartı üzerindeki LED'lerin sıralanmasına dikkat ediniz. Bizim verdiğimiz bit paterninin sırası, en soldaki MSB, en sağdaki LSB'dir.

3. Toplamları h'FF' i geçmeyen iki sayıyı yukarıdaki programda kendiniz girerek sonucu devreniz üzerinde LED'lerden izleyiniz.

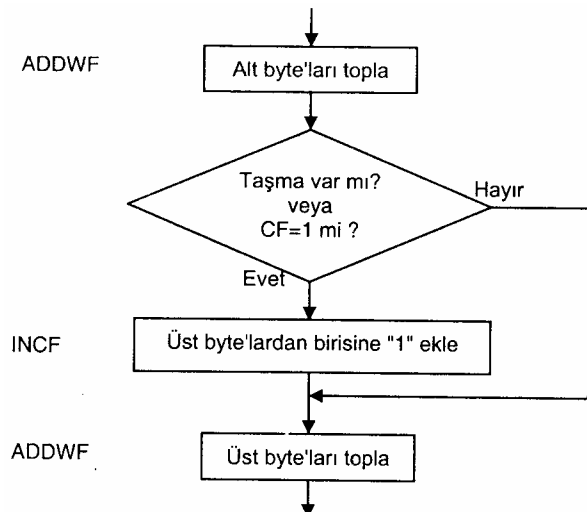
16- BİT TOPLAMA

Toplama işleminde kullandığımız sayılar 1 byte=h'FF'=d'256' dan daha büyük olduğunda ne yapmamız gerekir? Kolay, iki veya daha fazla byte ile gösteririz. Örneğin, h'01FD' sayısını iki parçaya bölerek h'01' için bir A registeri, h'FD' için bir B registeri kullanarak işlemleri yaparız. Aşağıda 16-bit toplama işleminin nasıl yapıldığını anlatırken B registerinin içeriğine alt byte, A registerinin içeriğine de üst byte diyeceğiz.

16-bit toplama işlemlerinde vereceğimiz örneklerde kullanacağımız 2 byte, 0-65535 arasındaki desimal sayıları ifade etmeye yeterli olacaktır. Bilgisayar aritmetiğinde 2 byte'lık veriler kullanarak yapılan işlemlere "Double precision - çift duyarlılık" işlemler adı verilir. Şimdi double precision toplama işleminin nasıl yapılacağını görelim:

1. Toplanacak iki sayının önce alt byte'ları toplanır.
2. Eğer alt byte'ların toplamında taşma varsa, üst byte'lardan birisine "1" eklenir.
3. Üst byte'lar toplanır.

Double precision toplama işlemini bir akış diyagramı çizerek ifade edelim:



Örneğin h'1613' sayısı ile h'01F0' sayısını toplarsak alt byte'ta meydana gelen taşmayı heksadesimal ve binary olarak toplama işleminde nasıl meydana geldiğini aşağıdaki gibi gösterebiliriz.

$$\begin{array}{r}
 \text{h}'1613' = \text{d}'5651' = \text{b}'0001011000010011' \\
 + \text{h}'01F0' = \text{d}'0496' = \text{b}'0000000111110000 \\
 \hline
 \text{h}'1803' = \text{d}'6147' = \text{b}'0001100000000011
 \end{array}$$

PROGRAM-17) h'61A3' ve h'2EE0' sayılarını toplayan program. Program çalıştığında toplamın alt byte'ı PortB'deki LED'lerde görülür. Üst byte'ın toplamını görmek için A1 butonuna basılır.

2 byte'lık h'61A3' sayısına A, h'2EE0' sayısına da B dersek, bu sayıları 1 byte'lık veri depolayabilen 2 tane file register kullanmamız gerekir. Şimdi programda kullanacağımız file registerlerin adını belirleyelim.



```

;====PROG17.ASM====22/06/2000=====s=
LIST    P=16F84
INCLUDE "P16F84.INC"
CLRF    PORTB           ;PORTB" yi sil.
BSF     STATUS, 5       ;BANK1'e geç.
CLRF    TRISB           ;B portu tüm uçları çıkış.
MOVLW   h'FF'           ;W←h'FF'
MOVWF   TRISA           ;A potu tüm uçları giriş.
BCF     STATUS, 5       ;BANK0'a geç.
AL      EQU    h'0C'     ;AL registerinin adresi
AH      EQU    h'0D'     ;AH registeri adresi
BL      EQU    h'0E'     ;BL registerinin adresi
BH      EQU    h'0F'     ;BH registerinin adresi
BASLA
    MOVLW h'A3'         ;W←h'A3'
    MOVWF AL            ;AL←h'A3'
    MOVLW h'61'         ;W←h'61'
    MOVWF AH            ;AH←h'61'
    MOVLW h'E0'         ;W←h'E0'
    MOVWF BL            ;BL←h'E0'
    MOVLW h'2E'         ;W←h'2E'
    MOVWF BH            ;BH←h'2E'

TOPLA
    MOVF  AL, W          ;W←AL
    ADDWF BL, P          ;BL=BL+W(AL), alt byte toplamı
    BTFSC STATUS, 0      ;CARRY FLAG=1 mi ?
    INCF  BH, P          ;Evet, BH=BH+1
    MOVF  AH, W          ;W←AH
    ADDWF BH, F          ;BH=BH+W(AH) , üst byte toplamı

ALT_BYTE_GOSTER
    MOVF  BL, W          ;W←BL
    MOVWF PORTB          ;Alt byte toplamını göster

TEST_A1
    BTFSC PORTA, 1       ;A1 butonuna basıldı mı?
    GOTO  TEST_A1        ;Hayır, tekrar test et

UST_BYTE_GOSTER
    MOVF  BH, W          ;Evet, W←0BH
    MOVWF PORTB          ;Üst byte toplamını göster.

DONGU
    GOTO  DONGU
    END

```

İşlem Basamakları

1. PIC16F84'ü programlayarak deneme kartınıza yerleştiriniz.
2. Deneme kartınıza gerilim uygulandığında LED'lerin "•oooo••" biçiminde yanarak alt byte'ların toplamını gösterdiğini izleyiniz. (h'83')

3. A1 butonuna basınız, bu defa üst byte'ların toplamını LED'lerde "●○○○○○" biçiminde görünüz.(h'90')
4. Alt byte'ı tekrar görmek için RESET butonuna basınız.
5. PROG17.ASM'de değişkenler içerisine girilen verileri değiştirerek yeni oluşan toplamı önce kendiniz hesaplayınız. Daha sonra PIC'İ programlayarak sonucu deneme kartınız üzerinde gözleyiniz.

8 - BIT ÇIKARMA

Bir byte'lık iki sayının iki şekilde çıkarması yapılabilir. İlki, sabit bir sayıdan W registerin içeriği çıkarılır(SUBLW). ikincisi, seçilen bir file register içeriğinden W register içeriği çıkarılır(SUBWF), sonuç W veya F registre yazılır.

SUBLW h'02' ;h'02'den W registeri çıkarılır.

SUBWF MEM, F ;MEM'den W registeri çıkarılır.

Küçük sayıdan büyük sayı çıkarılınca Carry Flag=0, büyük sayıdan küçük sayı çıkarılınca veya sayılar birbirine eşit olunca Carry Flag=1 olur.

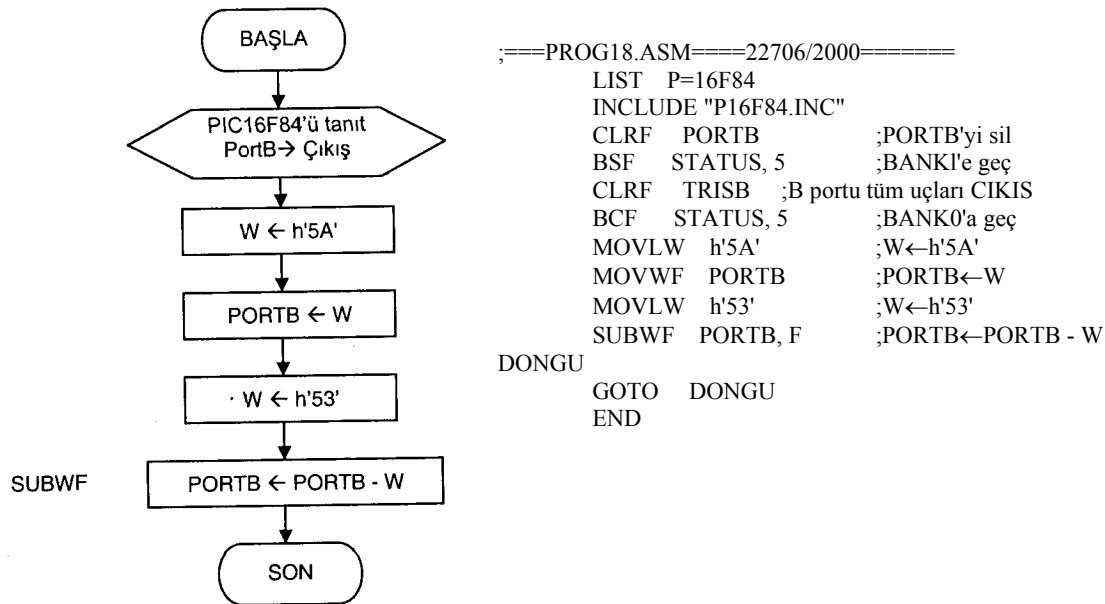
Örnekler:

h'05'	h'05'	h'05'
- h'06'	- h'05'	- h'04'
h'FF'	h'00'	h'01'

CarryFlag 0 1 1

CarryFlag 0 ise sonuç negatif, 1 ise pozitifdir.

PROGRAM-18) PORTB registeri içerisindeki h'5A' sayısından W registeri içerisindeki h'53' sayısını çıkaran, sonucu PORTB'ye bağlı LED'lerde gösteren program.(Büyük sayıdan küçük sayı çıkarma örneği.)



İşlem Basamakları

1. PIC16F84'ü programlayıp deneme kartına yerleştiriniz.
2. Deneme kartına gerilim uygulayıp programı çalıştırdığınızda portB'deki LED'ler "○○○○●●" biçiminde yanacaktır.

PROGRAM-19) PORTB registeri içerisindeki h'52' sayısından W registeri içerisindeki h'53' sayısını çıkaran, sonucu PORTB'ye bağlı LED'lerde gösteren program. (Küçük sayıdan büyük sayıyı çıkarma örneği.)

```

;==PROG19.ASM==22/06/2000====
LIST P=16F84
INCLUDE "P16F84.INC"
CLRF PORTB ;PORTB'yi SİL
BSF STATUS, 5 ;BANK1'e geç
CLRF TRISB ;B portu tüm uçları çıkış
BCF STATUS, 5 ;BANK0'a geç
MOVLW h'52' ;W←h'52'
MOVWF PORTB ;PORTB←W
MOVLW h'53' ;W←h'53'

```

```

SUBWF PORTB, F      ;PORTB←PORTB - W
COMPF PORTB         ;PortB'nin tersini al
INCF PORTB          ;PortB'ye "1" ekle
DONGU
GOTO DONGU
END

```

İşlem Basamakları

1. PIC16F84'ü programlayıp deneme kartına yerleştiriniz.
2. Deneme kartına gerilim uygulayıp programı çalıştırdığınızda portB'deki LED'lerin "oooooooo•" biçiminde yandığını görünüz.

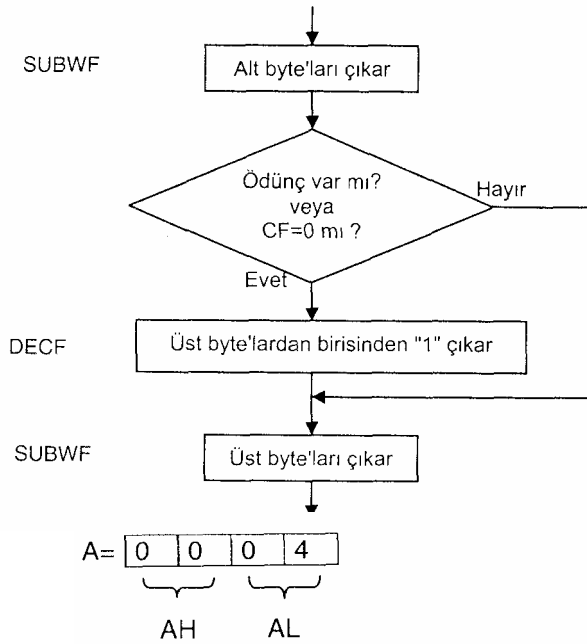
16- BITÇIKARMA

Double precision çıkarmada iki byte'lık veriler kullanıldığından sayıları ifade etmek için iki file register kullanmak gerekir.

Örneğin, 2 byte'lık h'61A8' sayısına A, h'2EE0' sayısına da B dersek, bu sayılardan her birini 1 byte'lık veri depolayabilen 2 tane file register kullanmamız gerekir.



1. Şimdi double precision çıkarma işleminin nasıl yapıldığını görelim:
2. Alt byte'lar birbirinden çıkarılır.
3. Alt byte'ların çıkarılması sonucunda ödünç alma varsa (Carry Flag=0'sa), üst byte'lardan birisinden "1" çıkarılır.
4. Eğer ödünç alma yoksa (Carry Flag=1'se) üst byte'ların birbirinden çıkarılması yapılır.



PROGRAM-20) h'0004' sayısından h'0001' sayısını çıkaran program. Program çalıştığında çıkarma sonucunun alt byte'ı PortB'deki LED'lerde görülür. Üst byte'ı görmek için A1 butonuna basılır.

2 byte'lık h'0004' sayısına A, h'0001' sayısına da B dersek, bu sayıları 1 byte'lık veri depolayabilen 2 tane file register kullanmamız gerekir. Şimdi programda kullanacağımız file registerlerin adını belirleyelim.

```

=====PROG20.ASM=====23706/2000=====
LIST P=16F84
INCLUDE "P16F84.INC"
CLRF PORTB      ;PORTB'yi sil.
BSF STATUS, 5   ;BANK1'e geç.
CLRF TRISB      ;B portu tüm uçları çıkış.
MOVLW h'FF'     ;W←h'FF'
MOVWF TRISA     ;A potu tüm uçları giriş.
BCF STATUS, 5   ;BANK0'a geç.
AL EQU h'0C'    ;AL registerinin adresi
AH EQU h'0D'    ;AH registeri adresi
BL EQU h'0F'    ;BL registerinin adresi

```

```

BH      EQU    h'0D'    ;BH registerinin adresi
BASLA
        MOVLW   h'04"    ;W←h'04'
        MOVWF   AL        ;AL←h'04'
        MOVLW   h'00'    ;W←h'00'
        MOVWF   AH        ;AH←h'00'
        MOVLW   h'01'    ;W←h'01'
        MOVWF   BL        ;BL←h'01'
        MOVLW   h'00'    ;W←h'00'
        MOVWF   BH        ;BH←h'00'

CIKAR
        MOVF    BL, W      ;W←BL
        SUBWF   AL, F      ;AL=AL-W(BL), alt byte sonucu
        BTFSS   STATUS, 0  ;CARRY FLAG=0 mı ?
        DECF    AH, F      ;Evet, AH=AH-1
        MOVF    BH, W      ;Hayır, W←BH
        SUBWF   AH, F      ;AH=AH-W(BH), üst byte sonucu
ALT_BYTE_GOSTER
        MOVF    AL, W      ;W←AL
        MOVWF   PORTB      ;Alt byte sonucunu göster TEST_A1
        BTFSC   PORTA, 1   ;A1 butonuna basıldı mı?
        GOTO    TEST_A1    ;Hayır, tekrar test et
        MOVF    AH, W      ;Evet, W←AH
        MOVWF   PORTB      ;Üst byte sonucunu göster.
DONGU
        GOTO    DONGU
END

```

İşlem Basamakları

1. PIC16F84'ü programlayarak deneme kartınıza yerleştiriniz.
2. Deneme kartınıza gerilim uygulandığında LED'lerin "oooooooo●●" biçiminde yanarak alt byte'ların çıkarma sonucunu gösterdiğini izleyiniz. (h'03')
3. A1 butonuna basınız, bu defa üst byte'ların çıkarma sonucunu LED'lerde
4. "oooooooo" biçiminde görünüz.(h'00')
5. PROG20.ASM'de değişkenler içerisine girilen verileri değiştirerek yeni oluşan çıkarma sonucunu önce kendiniz hesaplayınız. Daha sonra PIC'İ programlayarak sonucu deneme kartınız üzerinde gözleyiniz.

11

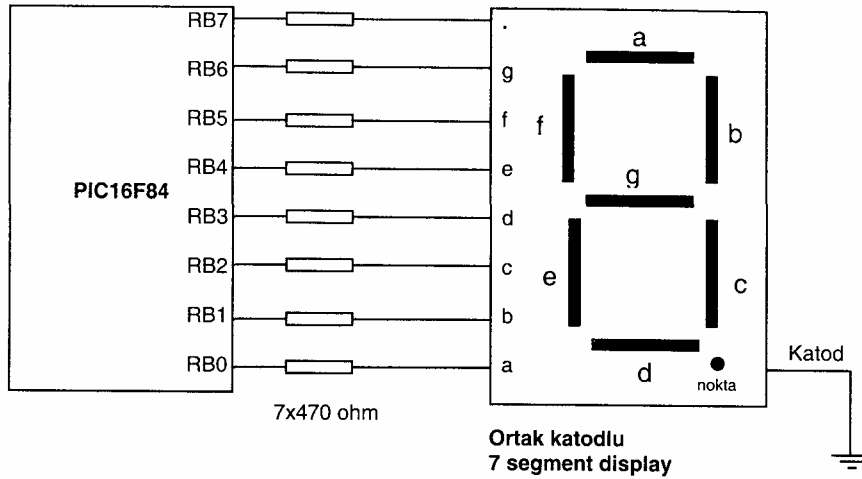
ÇEVİRİM TABLOLARI

- ☐ ÇEVİRİM TABLOSU NEDİR?
- ☐ PROGRAM COUNTER
- ☐ 7 SEGMENT DISPLAY'İ KULLANMAK
- ☐ STEP MOTOR KONTROLÜ

ÇEVİRİM TABLOSU (LOOKUP TABLE) NEDİR?

Çevrim tabloları bir kodu başka bir koda çevirmek için kullanılırlar. Örneğin PORTB'ye bağladığımız 7 segment display'in üzerinde heksadesimal karakterleri görmek istediğimizi düşünelim. Çevrim tablosuna yerleştirdiğimiz heksadesimal koda karşılık gelen uygun kodu seçip, PORTB'ye göndermemiz gerekir. Aşağıda ^ segment sürücünün 0~F arasındaki sayıları göstermesi için gereken kodlar verilmiştir. Bu kodları örnek olarak vereceğimiz programlarda kullanacağız.

(Çevrilecek kod Hex sayısı)	Çevrilen 7 segment kodu (PORTB'ye)	7 segment uçlarındaki veri	7 segment'te görülecek sayı
h'00'	h'3F'	00111111	0
h'01'	h'06'	00000110	1
h'02'	h'5B'	01011011	2
h'03'	h'4F'	01001111	3
h'04'	h'66'	01100110	4
h'05'	h'6D'	01101101	5
h'06'	h'7D'	01111101	6
h'07'	h'07'	00000111	7
h'08'	h'7F'	01111111	8
h'09'	h'BF'	01101111	9
h'0A'	h'77'	01110111	A
h'0B'	h'7C'	01111100	b
h'0C'	h'39'	00111001	C
h'0D'	h'5E'	01011110	d
h'0E'	h'79'	01111001	E
h'0F'	h'71'	01110001	F
Nokta	h'80'	10000000	

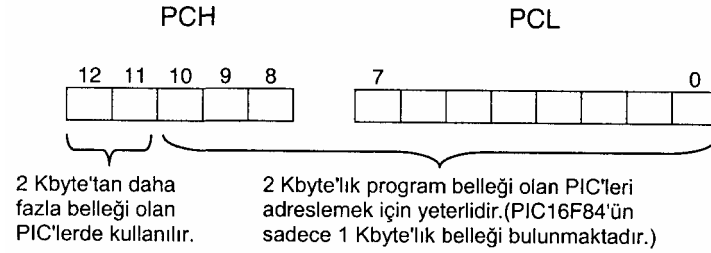


Örnek program vermeye başlamadan önce çevrim tablosuna yerleştirilen uygun kodun seçiminin nasıl yapılacağından bahsedelim. İlk olarak bu seçme işini gerçekleştiren program sayıcı'yı (PROGRAM COUNTER), daha sonra da kodu ana programa gönderen RETLW komutunu incelememiz gerekiyor.

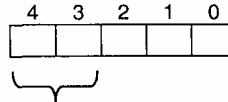
PROGRAM COUNTER (SAYICI)

PIC16F84'ün 13 bit'lik bir program sayıcısı vardır. GOTO ve CALL komutlarıyla kullanılan 11 bit'lik adresler 2 Kbyte'lık program belleği bulunan PIC'leri adreslemek için yeterlidir. PIC16F84'ün 1 Kbyte'lık program belleği olduğundan daha önce bahsetmiştik.

PIC programlarında, program sayıcı PC kodu ile kullanılır. Program sayıcının alt 8 bit'ine PCL, üst 5 bit'ine de PCH adı verilir.



Program sayıcısının üst 5 bit'ini doğrudan okumak ve yazmak mümkün değildir. Ancak PIC16F84'ün RAM belleğinde h'0A' adresinde bulunan ve adına PCLATH verilen özel registerden veri yüklenebilir. PCLATH registeri de 5 bit'tir.



2 Kbyte'tan daha fazla belleği olan PIC'lerde gereklidir. Bu tip PIC'lerde 1'den fazla BANK bulunmaktadır. Diğer BANK'lara geçebilmek için bu iki bit kullanılır.

14 bit program belleği olan PIC'lerde (PIC16CXXX ve PIC16FXX) herhangi bir adresi tanımlamak için 11 bit yeterlidir ($2^{11}=2048=2$ Kbyte). 1 Kbyte'lık belleği olanlarda ise 10 bit yeterlidir ($2^{10}=1024=1$ Kbyte).

Bu kitapta örnekleri verilecek olan programlar genellikle 256 Kbyte'ı ($2^8=256$ byte) geçmeyeceğinden, program belleğinin ilk 8 bit'ini (PCL) kullanmak yetecektir. Eğer 256 byte'tan daha fazla yer kaplayan programlar yazmak gerekirse PCLATH registerinin 0, 1 ve 2. bit'ini de kullanmak gerekecektir. PCLATH kullanımı bizim hedef aldığımız teknik bilginin üstünde aşırı detay olduğundan bu konuya girmeyeceğiz.

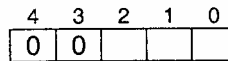
RETLW Komutu

Bir alt programdan ana programa dönüş için kullanılan RETURN komutuna çok benzer. RETLVV de ana programa dönüş için kullanılır. Tek bir farkla; ana programa dönüş esnasında W registere bir sabit sayı yüklenir. Örneğin, ana programa döndüğünde W registerinin içerisinde h'3F' sayısı bulunması isteniyorsa, komut aşağıdaki gibi yazılır.

RETLW h'3F' ;W'ye h'3F' yükle ve ana programa dön.

PCLATH'ı kullanma ihtiyacınız olmamasına rağmen bu register hakkında bilmeniz gereken önemli bir noktadan söz edelim. PCLATH'ın içeriği PIC'e gerilim uygulandığında (Power-On-Reset) b'00000' dir. Program içerisinde PCH'a veri yazma komutu kullanmadığınız sürece bu registerle herhangi bir işiniz yoktur. Buna rağmen PIC16F84'ün çalışmasında bir aksaklığa neden olmaması için 4. ve 3. bit'lerin içeriğini sıfırlanmalıdır. Bunun yerine program içerisinde PC değil de PCL kullanmak da aynı çözümü getirir. Bu durumda program sayıcısının (PC) sadece alt 8 bit'i (PCL) kullanılmış olur.

PCLATH



PROGRAM-21) 7 segment display üzerinde "5" sayısını gösteren program. **NOT:** Program 5 sayısına karşılık gelen h'6D' sayısını seçmek için kullanılan PC ve çevrim tablolarının kullanılmasını açıklamaları bakımından önemlidir.

=====PROG21.ASM=====26/06/2000=====

```

LIST P=16F84
INCLUDE "P16F84.INC"
CLRF PORTB ;PortB'yi sil
BSF STATUS, 5 ;BANK1'e geç
CLRF TRISB ;PortB çıkış
BCF STATUS, 5 ;BANK0'a geç

BASLA
    MOVLW h'05' ;W←h'05'(test sayısı)
    CALL CEV_TAB
    MOVWF PORTB

DONGU
    GOTO DONGU

CEV_TAB
    ADDWF PCL, F ;PCL←W(h'05')
    RETLW h'3F'
    RETLW h'06"
    RETLW h'5B'
```

```

RETLW h'4F'
RETLW h'66'
RETLW h'6D' ;W←h'6D'
RETLW h'7D'
RETLW h'07'
RETLW h'7F'
RETLW h'6F'
RETLW h'77'
RETLW h'7C'
RETLW h'39'
RETLW h'5E'
RETLW h'79'
RETLW h'71'
RETLW h'80'
END

```

Programın çalışmasını BASLA etiketinden itibaren takip edelim;

1. MOVLW h'05' komutuyla W registere h'05' sayısı yüklenir.
2. CALL CEV_TAB komutuyla program akışı alt programa geçer.
3. ADDWF PCL, F komutuyla program sayıcının alt 8 bit'i (PCL) içerisine W registerinde bulunan h'05' sayısı yüklenir. Program akışı 5. satırdaki RETLWV komutuna dallanır.
4. RETLW h'6D' komutu W register içerisine h'6D' sayısını yükler. Buradaki h'6D' sayısı, PORTB'ye bağlı olan 7 segment display'de "5" sayısının görüntülenmesi için gerekli olan koddur.
5. Ana programa dönen program akışı MOVWF PORTB komutuyla devam eder. Böylece PORTB'ye gönderilen h'6D' sayısı 7 segment display'de "S" sayısının görüntülenmesini sağlar.

İşlem Basamakları

1. Breadboard üzerine yukarıda verilen devreyi kurunuz. PROG21,ASIV programını deneme kartı üzerinde kolaylıkla çalıştırabilirsiniz. Kart üzerinde yapılacak şey, LED-SEGMENT seçme anahtarını SEGMENT konumuna almaktır.
2. PIC16F84'ü programlayarak devreniz üzerine yerleştiriniz. PIC'e gerilim uyguladığınızda 7 segment display üzerinde "5" sayısını görünüz.

PROGRAM-22) PORTB'nin uçlarına bağlı 7 segment display'de 0~F arasında saydıran program. Sayılar arasındaki duruş GECIKME alt programı tarafından sağlanmıştır.

```

;====PROG22.ASM====27/06/2000=====
LIST P=16F84

```

INCLUDE "P16F84.INC"

```

SAYAC1 EQU h'0C'
SAYAC2 EQU h'0D'
SAYAC EQU h'0E'
CLRF PORTB
BSF STATUS, 5
CLRF TRISB
BCF STATUS, 5

BASLA
MOVLW h'00' ;b'00001111' sayısını W'ye yükle
MOVWF SAYAC ;SAYAC←W DONGÜ
MOVF SAYAC, W ;W←SAYAC
ANDLW B'00001111' ;W'nin üst dört bitini sıfırla
CALL CEV7SEG ;Kod çevinne alt programını çağır
MOVWF PORTB ;Kodu 7 segmentte göster
INCF SAYAC, F ;SAYAC←SAYAC+1
CALL GECIKME ;Gecikme yap
GOTO DONGU ;Yeni bir sayı için başa dön.

CEV7SEG
ADDWF PCL, F ;PCL←W(SAYAC)
RETLW h'3F' ;1
RETLW h'06' ;2
RETLW h'5B' ;3
RETLW h'4F' ;3
RETLW h'66' ;4
RETLW h'6D' ;5
RETLW h'7D' ;6
RETLW h'07' ;7
RETLW h'7P' ;8
RETLW h'6P' ;9

```

```

RETLW h'77' ;A
RETLW h'7C' ;b
RETLW h'39' ;C
RETLW h'5E' ;d
RETLW h'79' ;E
RETLW h'71' ;F
GECIKME ;Gecikme alt programı
MOVLW h'FF'
MOVWF SAYAC1
DONGU1
MQVLW h'FF'
MOVWF SAYAC2
DONGU2
DECFSZ SAYAC2, F
GOTO DONGU2
DECPSZ SAYAC1, F
GOTO DONGU1
RETURN
END

```

İşlem Basamakları

1. PIC16F84'ü programlayarak, breadboard üzerine kurduğunuz devre veya deneme kartına yerleştiriniz.
2. PIC'e gerilim uygulayarak programı çalıştırınız. 7 segment üzerinde 0~F arasındaki sayıların belirli zaman aralıklarıyla görülecektir. Sayı "F" e gelince tekrar "0" a dönecektir.

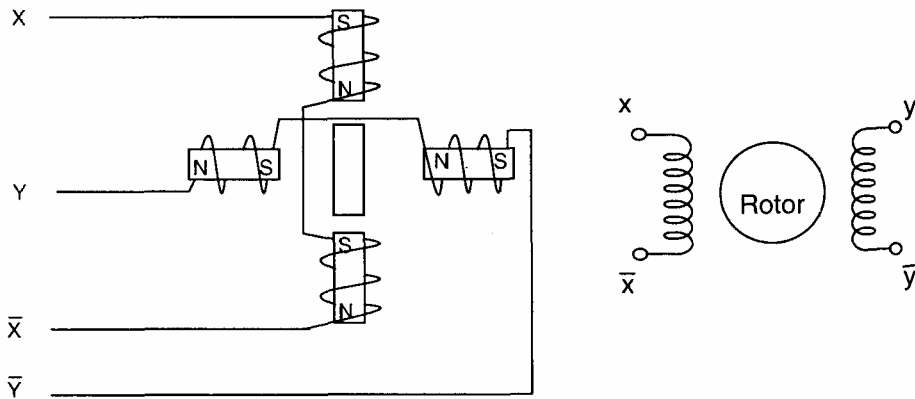
STEP MOTOR KONTROLÜ

PIC16F84 ile bir step motorun kontrolü yapılabilir. Yani motor bir yöne döndürülebilir, devir yönü değiştirilebilir. Kontrol edilecek motorun kutup sayısı, adım açısı uygulama programlarında değişiklikler gerektirir. Bu nedenle önce step motorların çalışma prensibini hatırlatalım.

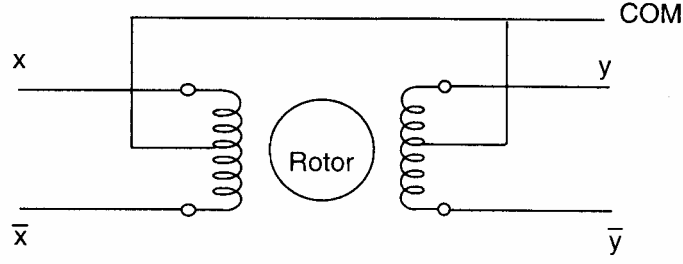
Step motorların dönen kısmı(rotor) sabit mıknatıstan yapılmıştır. Duran kısmında(stator) ise belirli aralıklarla yerleştirilmiş elektromıknatıslar bulunmaktadır. Elektromıknatısın içerisinde geçen akımın yönüne göre N-S kutuplarının yönü de değiştirilebilmektedir. Bir step motorun döndürülmesi için belli bir sırayla bu elektromıknatısların enerjilenmesini sağlayan gerilimler motor uçlarından uygulanır. Böylece rotordaki sabit mıknatıs, statorun enerjilenen kutupları tarafından yönlendirilir(N-S kutupları birbirini çeker, N-N veya S-S kutupları birbirini iter.)

Step motor kontrolü uygulamalarında eski bir disket sürücünden sökeceğiniz bir step motoru kullanmak en uygun bir yoldur. Bu tür motorlarda genellikle dört kablo çıkışı vardır ve bipolar step motor adı verilir. PIC'in çıkış uçlarını bu uçlara direkt olarak bağlanması mümkün değildir. Çünkü PIC16F84'ün verebileceği akım 2 mA civarındadır. Step motor çok daha fazla akım çektiğinden, PIC çıkışı muhakkak bir transistörlü sürücü devresi kullanılarak motora bağlanmalıdır. Bu kitapta hedef alınan konu PIC assembly olduğundan detaylı elektronik bilgisine girilmeyecektir.

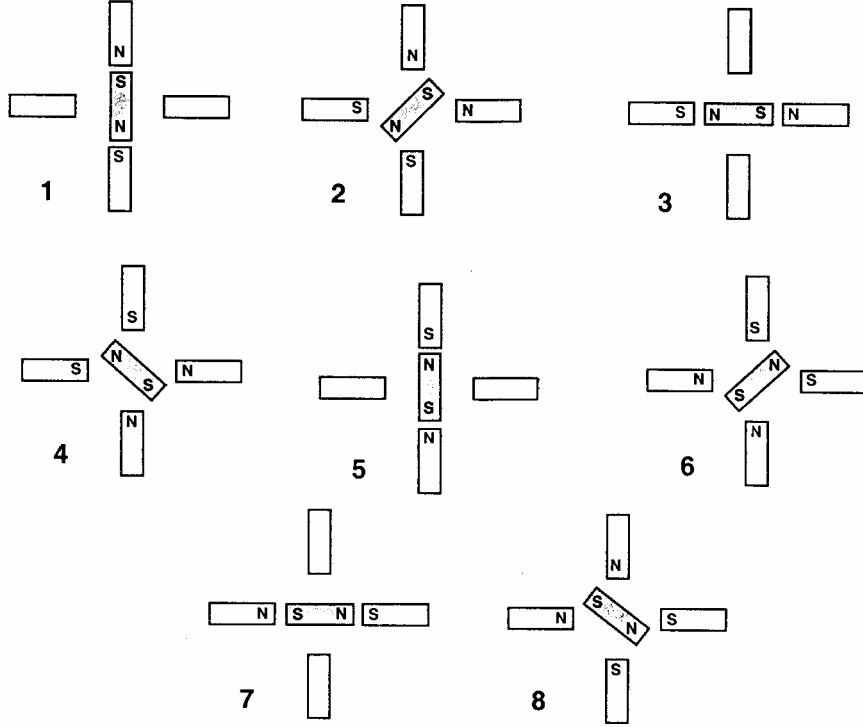
Aşağıda uygulama programını vereceğimiz step motorun basitleştirilmiş elektromıknatıs(stator) bağlantı şekli verilmiştir. Gerçek bir step motorda bu stator sargılarında çok miktarda bulunmaktadır. Sargı sayısı, step motorun adım sayısı ile direkt olarak bağlantılıdır.



Kullandığımız step motor 2 phase (faz), 20 adımlıdır. Bu da 360° lik tam bir dönüş içerisinde 18° lik adımlarla ilerleyen motor demektir. X-X ve Y-Y uçlarına uygulanacak gerilimler elektromıknatıslarda oluşacak N-S kutuplarının yönünü belirleyecektir. Step motorların diğer bir çeşidi de unipolar step motorlardır. Bu motorlarda beş kablo çıkışı vardır. Bunlarda da yine iki sarım olmakla birlikte, her sarımın ortasından bir uç daha çıkarılmıştır. Bu uçlar birbirine bağlanarak(COM) uç elde edilmiştir. Aşağıda bu tip motorun şematik gösterilişi verilmiştir.



Şimdi de dört uçlu bir step motorun sağa doğru dönmesi için stator sargılarında oluşması gereken kutuplaşmayı adım adım gösterelim.



Akımın geçiş yönü mıknatıslanmanın da yönünü belirleyeceğinden yukarıdaki şekillerde verilen adımlardaki kutuplaşmayı sağlayacak gerilimler X, X-bar, Y, Y-bar uçlarından verilir. PIC uçlarından verilecek olan bu gerilimler için aşağıdaki tabloyu çıkarabiliriz.

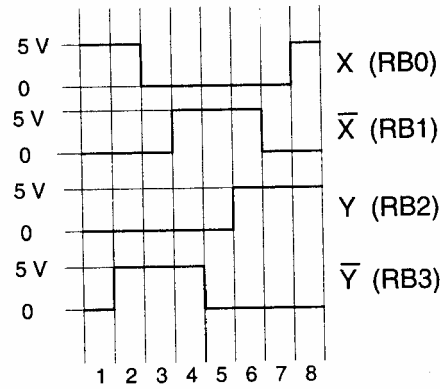
Adım	X-bar	X	Y	Y-bar
1	1	0	0	0
2	1	0	0	1
3	0	0	0	1
4	0	1	0	1
5	0	1	0	0
6	0	1	1	0
7	0	0	1	0
8	1	0	1	0
9(1)				

1. adım ile aynı

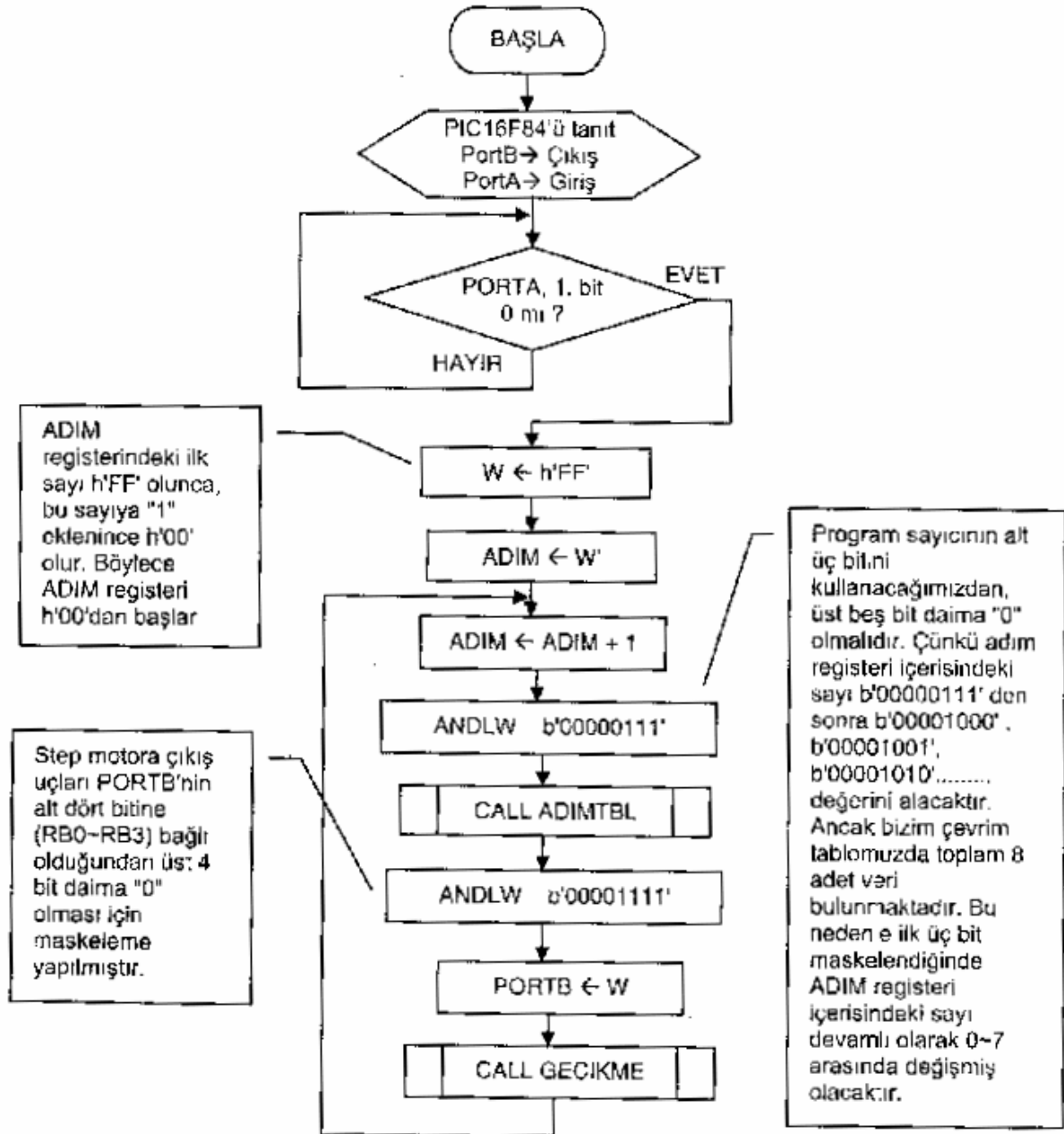
Yukarıdaki gerilimler tablosundaki "1" ler, motorun uçlarına uygulanacak olan pozitif gerilimleri ifade etmektedir. Bu gerilim 5 V olabileceği gibi, motor sürücü devresi (Transistörlü bir yükselteç devresi) tarafından elde edilecek daha büyük bir gerilim de olabilir.

Biz uygulamalarımızı denemek için PORTB'ye bağladığımız LED'leri kullandığımız için bu tablodaki değerleri PIC assembly programında kullandığımız çevrim tablosuna (lookup table) yazmanız yeterlidir. Ancak bir motor sürücü devresi yapıldıysa transistörleri sürmek için daha farklı bir tablo gerekebilir. Step motor kullanılan gerçek uygulama devresinde bu durum göz önüne alınmalıdır.

PIC çıkışından step motora uygulanan sinyalin frekansı çok yüksek olduğundan step motor bu frekansta uygulanan gerilimlerle yeterli kutuplaşma sağlamayabilir. Bu nedenle çevrim tablosundaki adımlar arasında gecikme sağlayan alt program muhakkak kullanılmalıdır. Aşağıda PIC uçlarından çıkması gereken gerilimlerin zaman tablosu verilmiştir, inceleyiniz.



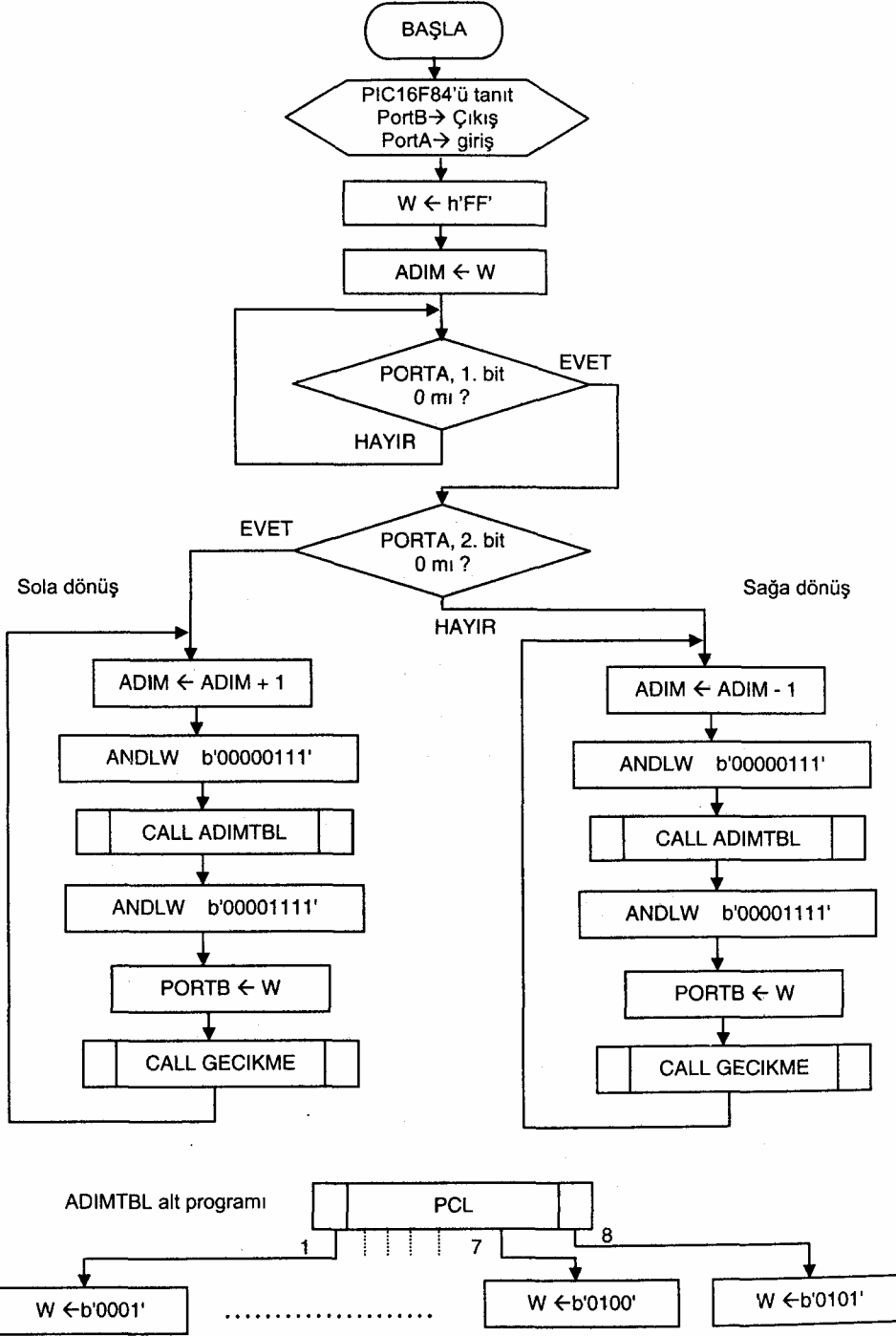
PROGRAM-23) Bipolar (4 uçlu) bir step motoru bir yöne doğru A1 butonuna bastıkça adım adım döndüren program. Step motor Liçlarına gönderilecek gerilimler PORB'deki RBO-RB3 uçlarına bağlanan LED'lerde görülür.



```

;=====PROG23.ASM=====30/06/2000=====
LIST P=16F84
INCLUDE "P16F84.INC"
SAYACIEQU h'0C'
  
```

PROGRAM-24) Bir step motoru PORTA'nın 1. bitine bağlı A1 butonuna basınca sağa, A1 ve A2 butonlarına birlikte basınca da sola döndüren program. Step motor uçlarına gönderilecek olan gerilimler PORTB'deki LED'lerde görülür.



```

=====PROG24.ASM=====02/07/2000=====
LIST    P=16P84
INCLUDE "P16F84.INC"

SAYAC1 EQU    h'0C'
SAYAC2 EQU    h'0D'
ADIM EQU      h'0E'

CLRF    PORTB        ;PortB'yi sil
BSF     STATUS, 5    ;BANK1'e geç
MOVLW   h'FF'        ;W←h'FF'
MOVWF   TRISA        ;PortA'nın tüm uçları giriş
CLRF    TRISB        ;PortB'nin tüm uçları çıkış
BCF     STATUS, 5    ;BANK0'a geç
MOVLW   h'FF'        ;W←h'FF'
MOVWF   ADIM         ;ADIM←W (h'FF')

BASLA

```

```

        BTFSC PORTA, 1      ;PORTA'nın 1.biti 0 mı?
        GOTO BASLA         ;Hayır, tekrar test et.
        BTFSC PORTA, 2      ;PORT'nın 2. biti 0 mı?
        GOTO SAG           ;Hayır, sağa döndür.

SOL
        INCF ADIM, F        ;ADIM←ADIM+1 (h'FF'+h'01'=h'00')
        MOVF ADIM, W        ;W←ADIM
        ANDLW b'00000111'   ;W'nin üst 5 bitini maskele.
        CALL ADIMTBL        ;Bit paternini seç.
        AMDLW b'00001111'   ;w'nin üst 4 bitini maskele.
        MOVWF PORTB         ;Bit paternini PORTB'de göster.
        CALL GECIKME        ;LED'leri bir süre yanık tut.
        GOTO                ;Yeni bir bit paterni için git.

BASLA
SAG
        DECF ADIM, F        ;ADIM←ADIM+1 (h'FF'+h'01'=h" 00 • )
        MOVF ADIM, W        ;W←ADIM
        ANDLW b'00000111'   ;W'nin üst 5 bitini maskele.
        CALL ADIMTBL        ;Bit paternini seç.
        AMDLW b'00001111'   ;W'nin üst 4 bitini maskele.
        MOVWF PORTB         ;Bit paternini PORTB'de göster.
        CALL GECIKME        ;LED'leri bir süre yanık tut.
        GOTO BASLA         ;Yeni bir bit paterni için git.

ADIMTBL
        ADDWFPCL, F
        RETLW b'0001'
        RETLW b'1001'
        RETLW b'1000'
        RETLW b'1010'
        RETLW b'0010'
        RETLW b'0110'
        RETLW b'0100'
        RETLW b'0101'

GECIKME
        MOVLW h'FF'
        MOVWF SAYAC1
DONGU1
        MOVLW h'FF'
        MOVWF SAYAC2
DONGU2
        DECFSZSAYAC2, F
        GOTO DONGU2
        DECFSZSAYAC1, F
        GOTO DONGU1
        RETURN
        END

```

İşlem Basamakları

1. PIC16F84'ü programlayarak deneme kartınıza yerleştiriniz. PIC'e gerilim uygulayarak programı çalıştırınız.
2. A1 butonuna basarak motorun sağa dönmesi için PortB'den çıkan gerilimleri RB0-RB3 arasındaki LED'lerden izleyiniz.
3. A1 ve A2 butonlarına birlikte basarak motoru ters yönde çevirecek gerilimleri LED'lerden izleyiniz.

- ❑ KESME (INTERRUPT) NEDİR?
- ❑ INTCON REGISTER
- ❑ KESME KAYNAKLARI
- ❑ KESME ALT PROGRAMLARININ DÜZENLENMESİ

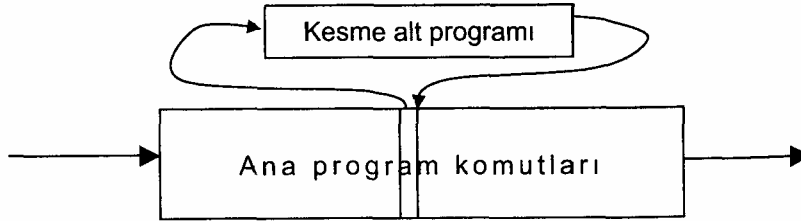
KESME (INTERRUPT) NEDİR?

Mikro işlemcilerle yeni çalışmaya başlayan çoğu kimseler, interrupt kelimesini duymalarına rağmen, kullanımlarının zor olduğu düşüncesiyle programları içerisinde kullanmaktan çekinirler. Oysa, öğrenilmesi ve uygulaması pek de zor olmayan interrupt alt programları kullanılarak, program içerisinde kullanılacak komut sayısı azaltılır ve bir sürü mantıksal karışıklıklar önlenir.

Peki, nedir interrupt? Konu başlığından da anlaşıldığı gibi Türkçe karşılığı "Kesme" dir. Kesme işlemini günlük hayattan bir örnek vererek açıklayalım: Diyelim ki televizyonda sevdiğiniz bir artistin filmini izliyorsunuz ve bu anda telefon çaldı. Ne yaparsınız? İlk olarak konsantrasyonunuz bozulur ve ne yapacağımıza karar verirsiniz. Eğer önemli bir telefon bekliyorsanız, televizyonun sesini keser, video player'ın record butonuna basarak filmi kaydedersiniz. Telefon konuşması bittikten sonra da kaydettiğiniz filmi izlemeye başlarsınız. Eğer önemli bir telefon beklemiyorsanız kararınız bu defa telefona cevap vermeyerek arayanın telesekretere mesaj bırakması yönünde olur ve filmi izlemeye devam edersiniz.

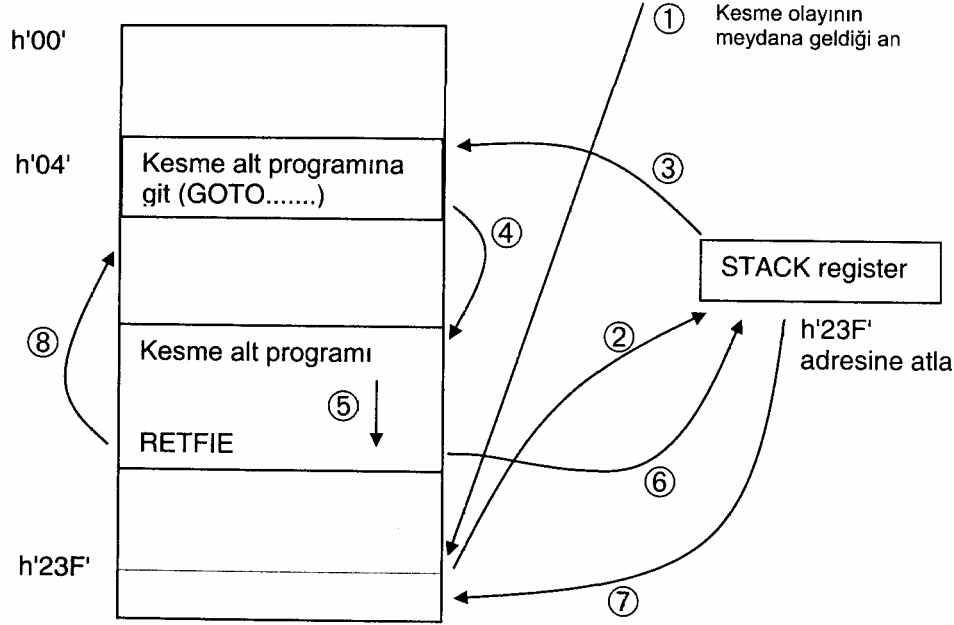
İşte, günlük hayatta uğradığınız bir kesme. Ama ne yaptınız? Bir yolunu bulup filmi izlemekten vazgeçmediniz. PIC kesmeleri de yukarda anlattığımız olaya çok benzer. Öyleyse PIC'te oluşan kesmeyi şöyle izah edebiliriz:

PIC'in port girişlerinden veya donanım içerisindeki bir sayıcıdan gelen sinyal nedeniyle belleğinde çalışmakta olan programın kesilmesi olayıdır. Programın kesildiği andan itibaren önceden hazırlanan bir alt program çalışır. Alt program işlevini bitirdikten sonra ana program kaldığı yerden itibaren tekrar çalışmasına devam eder. Netice olarak bir kesme, ana programın çalışmasını sadece duraklatır, ama hiçbir zaman işlevini devam ettirmesini engellemez.



İlk bakışta bir kesme meydana gelmesinin, alt program çağırma işleminden farkı olmadığı izlenimi verebilir. Ancak çok önemli bir fark vardır. Normal alt program çağırma, program içerisine yazılan komutlar vasıtasıyla yapılır. Kesme alt programlarının çağırılmasını ise donanımda oluşan değişiklikler yapar.

Bir kesme meydana geldiğinde, o anda çalışmakta olan komutun çalışması tamamlanır. Daha sonra program akışı PIC program belleğinin h'0004' adresine (PIC16F84'de) atlar ve bu adresteki komutu çalıştırır. Bu komut, "kesme servis programı" veya kısaca "kesme alt programı" diyeceğimiz alt programını çağıran bir GOTO komutudur. PIC, kesme alt programı çalıştıktan sonra ana programın hangi adresine geri döneceğini unutmamalıdır. Bu nedenle mikro denetleyici kesme oluştuğu anda çalışan komutun adresini STACK(Yığın) registerine kaydeder. Alt program işlevini tamamlayıp, program akışı ana programa geçince bu adresten itibaren devam eder. Kesme alt programından ana programa dönüş komutu RETFIE (Return From Interrupt)'dir.



- Kesme olayı esnasında meydana gelen olayları sıralayacak olursak;
- Kesme olayı meydana geldiğinde STACK registerin olduau adrese (h'23F') atla.
- Ana programın kaldığı adresi STACK registere yaz.
- h'04' adresindeki komutu çalıştır(Bu komut kesme alt programını çağırır.)
- Kesme alt programının olduğu adrese atla.
- Kesme alt programını çalıştır. Alt programın en son komutu RETFIE'dir.
- STACK (Yığın) registerin bulunduğu adrese git.
- Ana programa dönüş adresini al.
- Ana programın kesildiği yerdeki adresten bir sonraki adrese git ve devam et.

INTCON REGISTERİ

INTCON (Interrupt Control) registeri RAM bellekte h'0B' adresinde bulunan özel registerlerden bir tanesidir. Bu register içerisinde her bir kesme kaynağı için bir flag ve bir de global kesme bayrağı vardır. Tüm kesme işlemlerinin kontrolü bu register aracılığı ile yapılır.

bit7		6	5	4	3	2	1		0		
IE		C	EIE		E	TOI E	INT E	RBI E	TOI F	INT F	RBI F

Bit 7: **GIE**: Tüm kesme işlemlerini iptal etme bayrağı

0= Tüm kesmeler geçersiz (Disable)

1 = Aktif yapılmış olan tüm kesmeler geçerli (Enable)

Bit 6: **EEIE**: EEPROM belleğe yazma işlemi tamamlama kesmesi

0= Geçersiz (Disable)

1=Geçerli (Enable)

Bit 5: **TOIE**: TMRO sayıcı kesmesini aktif yapma bayrağı

0= Geçersiz(Disable)

1=Geçerli(Enable)

Bit 4: **INTE**: Harici kesmeyi aktif yapma bayrağı

0= Geçersiz(Disable). Harici kesmeler kabul edilmez.

1= Geçerli (Enable). Harici kesmeler kabul edilir.

Bit 3: **RBIE**: PORTB(4, 5, 6, 7. bit'leri) değişiklik kesmesini aktif yapma bayrağı

0= Geçersiz(Disable). PORTB'deki değişiklikler kesme oluşturur.

1= Geçerli(Enable). PORTB'deki değişiklikler kesme oluşturmaz.

Bit 2: **TOIF**: TMRO sayıcısı zaman aşımı bayrağı

0= Zaman aşımı yok.

1= Zaman aşımı var. (h'FF' den h'00' a geçiş.)

Bit 1: **INTF**: Harici kesme bayrağı

0= Harici kesme oluşmadığında.

1= Harici kesme oluştuğunda.

Bit 0: **RBIF**: PortB değişiklik bayrağı

0= RB4-RB7 uçlarında değişiklik yok.

1= RB4-RB7 uçlarından en azından birisinde değişiklik var.

KESME KAYNAKLARI

PIC16F84 kesmeleri aşağıdaki 4 kaynaktan gelebilir:

1. Harici(External) kesme (PIC16F84'ün RBO/INT ucundan giren sinyal.)
2. TMRO sayıcısında oluşan zaman aşımı kesmesi (TMRO sayıcısının h'FF' den H'00'a gelmesi.)
3. PORTB (4, 5, 6 ve 7 bit'ler) lojik seviye değişikliğinden.
4. EEPROM belleğe yazma işleminin tamamlanmasında meydana gelen kesme

Harici Kesmeler

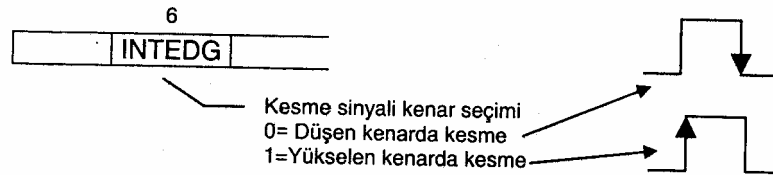
Harici kesmelerin kullanılabilmesi için iki şey gereklidir; yazılım ve donanım. Yazılım aracılığı ile PORTB'nin RBO/INT ucu dışardan gelebilecek kesmeyi alabilecek biçimde hazırlanmalıdır. Bu hazırlama işlemi için iki işlem yapılır;

1. RB0/INT ucu giriş olarak yönlendirmeli.
2. INTCON registeri içerisindeki ilgili bayrak (INTE bayrağı) kullanılarak harici kesme işlemi aktif (geçerli) yapılmalıdır.

Harici kesmenin kullanılabilmesi için bir de donanım gereksinmesi vardır. Bu da RBO ucundan sinyal girmek için gerekli elektronik devredir.

RBO/INT girişinden uygulanacak sinyalin kenar tetiklemesi önemlidir. OPTION registerin 6. bit'i bu uçtan girilen sinyalin yükselen kenarda mı yoksa alçalan kenarda mı kesme oluşturulacağına karar vermekte kullanılır.

OPTION register



Kesme alt programının çalışması esnasında gelebilecek yeni kesmeleri engellemek için, kesme oluştuğu zaman INTCON register'deki 4. bit (INTE bayrağı) "0" yapılmalıdır.

Eğer bir harici kesme(RBO/INT ucundan sinyal girişi) meydana gelirse, INTCON register içerisindeki INTF bayrağı "1" olur. INTF bayrağı interrupt alt programı içerisinde tekrar "0" yapılmalıdır. Aksi halde tekrarlanan kesmelerle karşılaşılır. Yukarıda söylediklerimizi kısaca özetlersek, bir harici kesme oluştuğunda yapılacak işlemlerin sırası şöyledir:

- Sonraki kesmeleri pasif (geçersiz) yapmak için (INTE bayrağını "0" yap.)
- Kesme alt programını çalıştır.
- INTF kesme bayrağını "0" yap.
- Yeni kelimeleri aktif(geçerli) yapmak için (INTE bayrağını "1" yap.)

TMRO Sayıcı Kesmesi

TMRO sayıcı kesmesi, bu özel sayıcı içerisindeki sayının h'FF'den h'00'a gelince oluşur. Bu sayıcıdan ve kesme işleminin oluşması 13. bölümde açıklanacaktır.

PORTB Lojik Seviye (RB4-RB7) Değişiklik Kesmesi

PORTB'nin 4, 5, 6, 7. bit'lerinde meydana gelen bir değişiklikte INTCON egisteri'nin 0. bit'i (RBIF) "1" olur. Bu kesme INTCON registeri'nin 3. bit'i (RBIE)

aracılığıyla aktif veya pasif yapılabilir. PORTB'nin 0, 1, 2, 3. bit'lerindeki değişiklikler kesme oluşturmaz.

PORTB'deki değişikliği algılamak için, bu porttaki en son değer, RB4-RB7 uçlarından okunan veri ile karşılaştırılır. Eski ve yeni okunan veriler OR'lanır. Farklılık varsa RBIF bayrağı (INTCON registerin 0. bit'i) "1" olur.

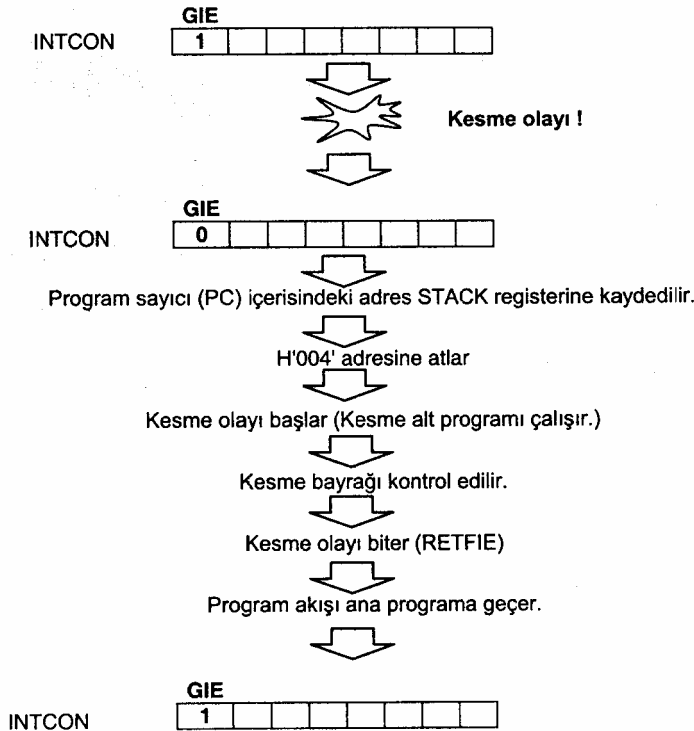
PORTB kesmesi aşağıdaki şekillerde silinebilir;

- RBIE bit'i (INTCON registerin 3. bit'i) silinmek suretiyle,
- PORTB'yi okuduktan sonra RBIF bit'ini silmek suretiyle.

KESME ALT PROGRAMLARININ DÜZENLENMESİ

Tüm Kesme İşlemlerini Aktif Yapma Bayrağı (GIE)

Bir kesme olayının meydana gelmesi esnasında INTCON registerinin 7. biti "0" olur. Bu işlem kesme alt programının çalışması esnasında yeni bir kesmenin program akışını bozmaması için PIC tarafından otomatik olarak yapılır. Kesme alt programı çalışmasını RETFIE komutu ile sona erdirip, ana programa döndüğü anda ise sonraki kesmelerin geçerli (enable) olabilmesi için tekrar otomatik olarak "1" yapılır. Şimdi olabildiği için tekrar otomatik olarak "1" yapılır. Şimdi kesme olayında oluşan olaylara genel bir bakış yaparsak aşağıdaki şemayı çizebiliriz.



Kesme Esnasında W ve Status Registeri Saklamak

Bir kesme olayı meydana geldiğinde program sayısı (PC) içeriği STACK registere otomatik olarak kaydedilir. Mikrodenetleyici bu sayede kesme işleminden dönüşte hangi adrese döneceğini bilir. Diğer registerlerin içeriğinin değişmemesi isteniyorsa, bu registerlerin korunma işlemi programcının sorumluluğuna bırakılmıştır. Eğer W ve STATUS registerin içeriği korunmak isteniyorsa, gerekli komutlar kesme alt programının içerisinde yer almalıdır. Register içeriğini saklama işlemi alt programın başında, geri yükleme işlemi ise sonunda bulunmalıdır.

Şimdi bir kesme oluştuğunda yapılması gereken işlemleri sıralayalım:

- W registeri bir değişkene yükle.
- STATUS registeri bir değişkene yükle.
- Kesme işlemini gerçekleştir.
- STATUS registeri geri yükle.
- W registeri geri yükle.
- Kesme alt programından dön (RETFIE).

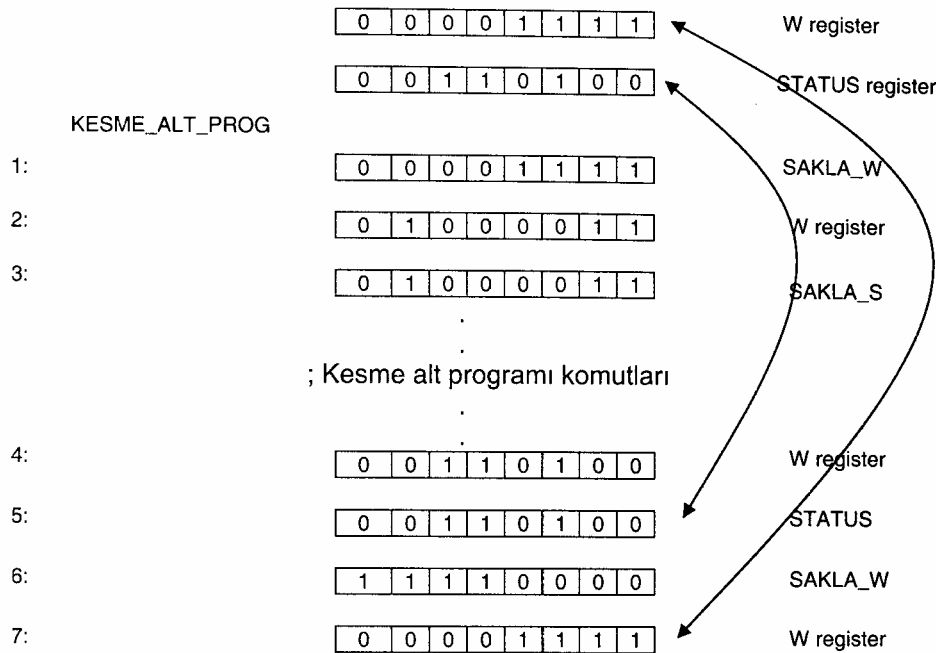
Bu işlemleri yapacak komutları yazalım:

```
ORG h'004"
GOTO KESME_ALT_PROG
•
•
KESME_ALT_PROG
1:    MOVWF SAKLA_W      ;W registerini yükle
2:    SWAPF STATUS, W    ;STATUS içeriğini SWAP yap.
3:    MOVWF SAKLA_S      ;STATUS içeriğini sakla
•
•
;Kesme alt programı komutları
•
4:    SWAPF SAKLA_S, W   ;STATUS içeriğini yeniden SWAP yap
5:    MOVWF STATUS       ;STATUS registre yeniden yükle
6:    SWAPF SAKLA_W, F   ;W içeriğini SWAP yap.
7:    SWAPF SAKLA_W, W   ;W'yi yeniden SWAP yap ve W'ye yaz
    RETFIE
```

SWAPF komutu bir register içeriğini başka bir registre STATUS registerini değiştirmeden yüklemek için kullanılmıştır. SVVAPF komutuyla alt dört bit ile üst dört bit'in yeri değişir. Bu nedenle alt programdan geri dönülmeden önce tekrar SVVAPF komutu eski haline getirilmesi gerekir.

Belki, bu işlem için neden MOVF komutu kullanmadığımız aklınıza gelebilir. Kullanamayız, çünkü MOVF komutu STATUS registerdeki Z(zero-sıfır) bayrağının içeriğini değiştirir. Bu da zero flag'ın durumunu kontrol ederek işlem yapan bir programda hatalara neden olabilir.

Şimdi yukarda yapılan işlemlerin daha pekişmesi için bir örnek verelim. Kesme alt programına girmeden önce W registerin içeriği b'00001111', STATUS registerin içeriği de b'00110100' olsun. Yukarıdaki program komutlarının her birinde W ve STATUS'un içeriklerini inceleyelim.

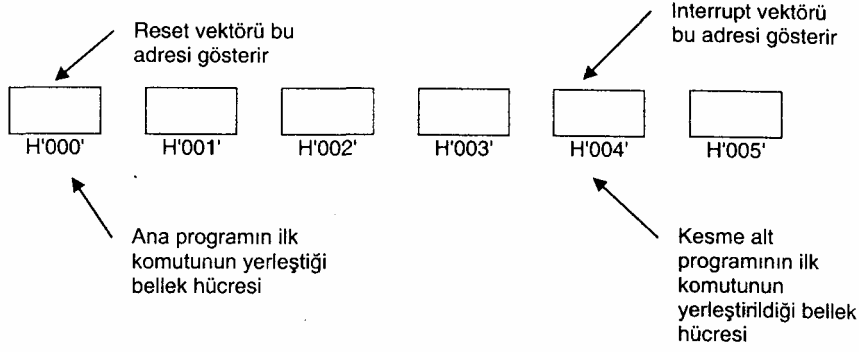


Görüldüğü gibi interrupt alt programı içerisinde çalışan komutlar W ve STATUS'un değerini değiştirmesi ihtimaline karşılık SAKLA_W ve SAKLA_S registerleri geçici depolama registeri olarak kullanılmıştır. Ana programa geri dönüşte bu saklama registerlerinin içerikleri aynı kalması sağlanmıştır.

Kesme Alt Programları Nereye Yazılmalı?

Bir kesme meydana geldiğinde "Interrupt vektör" adı verilen adres gösterici program belleğinin h'004' adresini gösterir. İşte kesme alt programının ilk komut buraya yazılmalıdır.

Kesme kullanılmadığı zaman ana program, program belleğinin h'0000 adresinden itibaren h'0004' adresine doğru herhangi bir karışıklığa sebep olmada çalışır. Eğer kesme kullanılıyorsa programcı tarafından başka bir çalışma sırası düzenlenmelidir.



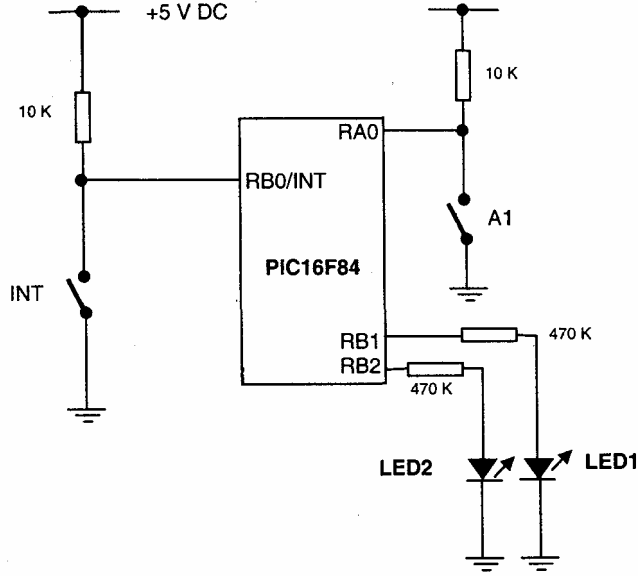
Interrupt kullanılan programların düzenlenmesi aşağıdaki gibi olmalıdır:

```
ORG h'000'
GOTO BASLA           ;Ana program başlangıcı
ORG h'004'
GOTO INT_ALT_PROG    ;Kesme alt programı başlangıcı.
;
BASLA
    Ana program komutlara.
    .
INT_ALT_PROG
    Kesme alt programı komutları
    .
RETFIE
```

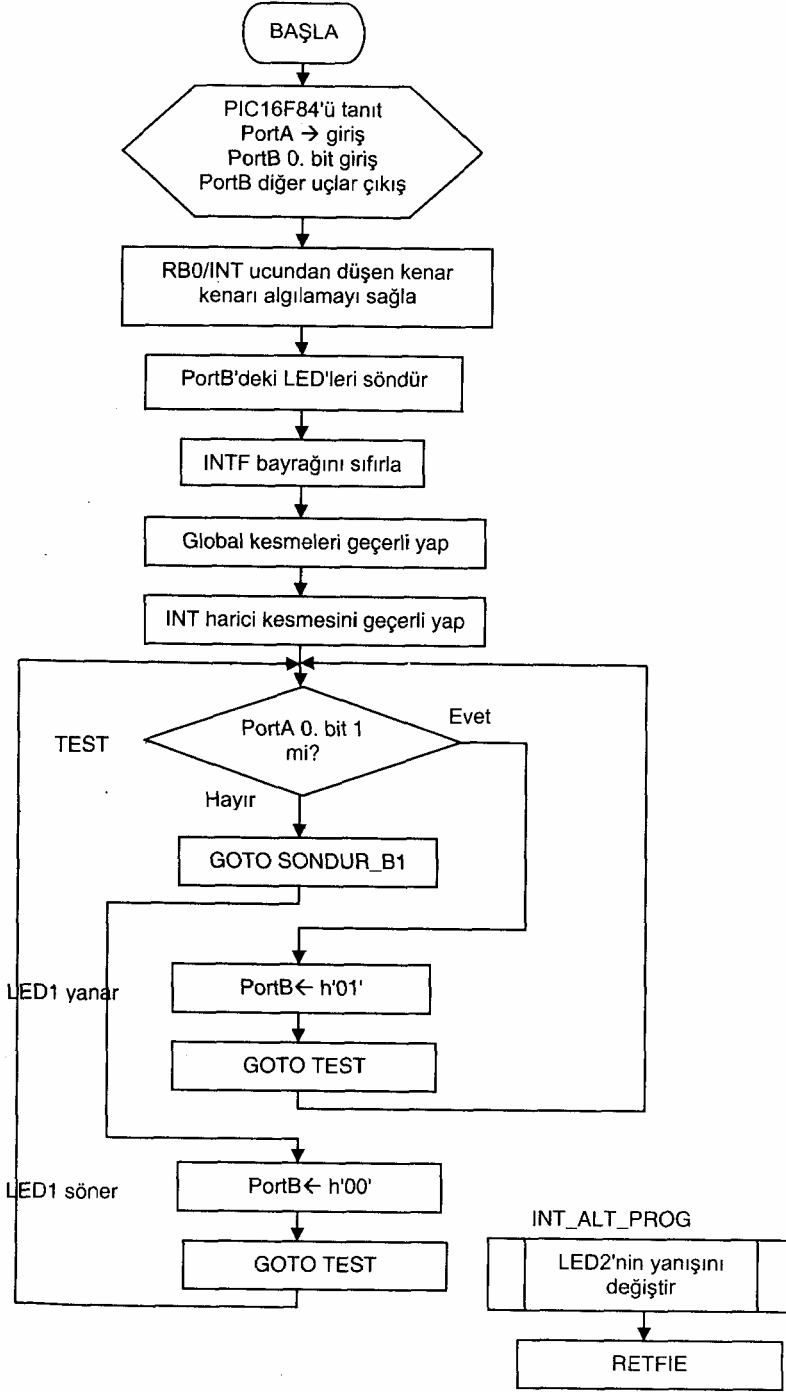
Kesme Gecikmesi

Bir kesme oluştuğunda, kesme alt programı çalışmadan önce gecikme meydana gelir. Bu gecikmeye "kesme gecikmesi" denir. Aşağı-yukarı 3 veya 4 komut saykılı süresidir. Zamanlamanın çok önemli olduğu uygulamalarda bu gecikme göz önüne alınmalıdır.

PROGRAM-25) Bu program RBO/INT ucundan girilen bir sinyal ile kesme oluşturulmasına örnektir. Aşağıdaki şekilde de görüldüğü gibi portA'nın 1. bit'ine bağlı butonun basılı olup olmadığını gösteren basit bir programdır. A1 butonuna basıldığı anda RB1 ucuna bağlı LED1'in yanışı bu programın devamlı olarak çalıştığını gösterir. RB0/INT ucundan bir sinyal girerek kesme oluşturmak için RB0 ucuna bir buton bağlanmıştır. INT butonuna basınca kesme oluşur ve kesme alt programı çalışarak RB2 ucuna bağlı olan LED2'yi yakar. Bu da ana program normal olarak çalışırken harici bir kesme meydana geldiğinde programın buna tepki verdiğini ve LED2'yi yaktığını gösterir.



Örnek olarak verdiğimiz deneme kartındaki portB'nin uçlarının tamamına LED bağlandığından program-25'i bu programı denemek için kullanamayız. Yukarıdaki devreyi breadboard üzerinde kurunuz. Besleme ve osilatör devreleri için kullanılacak elemanlar şema üzerinde gösterilmemiştir. Elinizdeki olanaklara göre kristal veya RC osilatör devresini önceki bilgilerinizden yararlanarak kurunuz.



=====PROG25.ASM=====05/07/2000=====

LIST P=16F84

INCLUDE "P16F84.INC"

ORG h'000'

GOTO BASLA

ORG h'004'

GOTO INT_ALT_PROG

BASLA

```

MOVLW h'FF'           ;W←h'FF'
MOVWF TRISA           ;PortA tüm uçlar giriş
MOVLW h'00000001'     ;W←h'01'
MOVWF TRISB           ;PortB 0. bit giriş
MOVLW h'10111111'     ;W←b'10111111', düşen kenar
MOVWF OPTION_REG      ;W'yi OPTION registre yükle
  
```



```

        CLRF  PORTB          ;PortB'yi sil
        BCF  INTCON, 1      ;INTF bayrağını sil,
                             ;kesme sinyaline hazırla
        BSF  INTCON, 7      ;Global kesmeyi geçerli yap
        BSF  INTCON, 4      ;RB0/INT kesmesini geçerli yap
TEST
        BTFSS PORTA, 0      ;PortA'nın 1. bit'ini test et
        GOTO SOMDUR_LED1
YAK_LED1
        BSF  PORTB, 1        ; PortB'nin 1. bit'ini "1" yap
        GOTO TEST
SONDUR_LED1
        BSF  PORTB, 1        ; PortB'nin 1. bit'ini "0" yap
        GOTO TEST
INT_ALT_PROG
        BCF  INTCON,1        ; INTF bayrağını sil
        MOVLW b'00000100'    ; Terslenecek bit'i W'ye yükle
        XORWF PORTB,F        ; RB2'yi tersle
        RETFIE               ; Kesme alt programından dön.
END

```

Programdaki

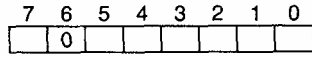
```

        MOVLW b'10111111'
        MOVWF OPTION_REG

```

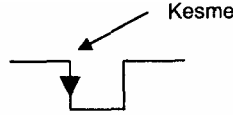
Komutlarıyla RBO ucundan girilen sinyalin düşen kenarında kesme oluşturmaları sağlanmıştır.

OPTION REGISTER



INTEDG

Kesme sinyali kenar seçimi
0= Düşen kenarda kesme



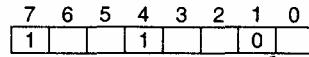
INTCON registerindeki bayrakların kurulması için de aşağıdaki komutlar kullanılmıştır.

```

        BCF  INTCON,1
        BSF  INTCON,7
        BSF  INTCON,4

```

INTCON REGISTER



GIE

Tüm kesme işlemlerini
aktif yapma bayrağı

INTE

Harici kesmeyi aktif
yapma bayrağı

INTF

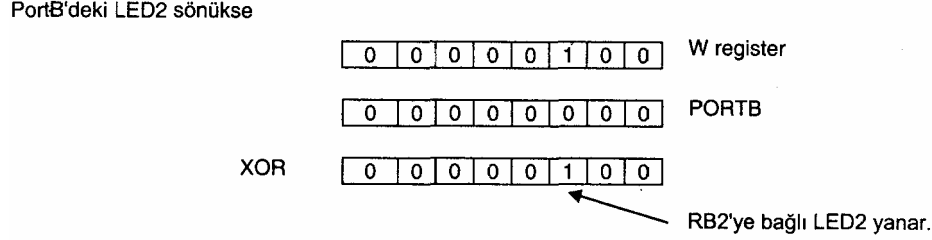
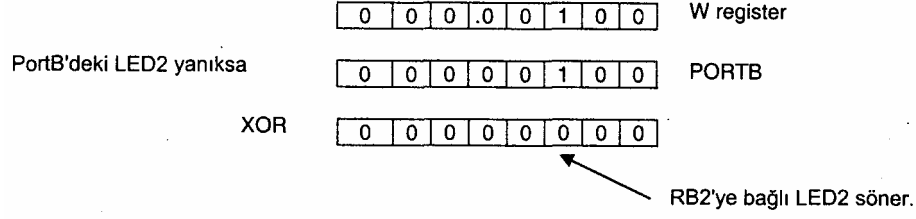
Harici kesme bayrağı
0= Kesme yok
1= Kesme var

Kesme olayının meydana geldiğini gösteren ve PortB'nin 2. bit'ine bağlı LED2'nin durumunu değiştirmek için aşağıdaki komutlar kullanılmıştır.

```

        MOVLW b'00000100'
        XORWF PORTB, F

```



İşlem Basamakları

1. PIC16F84'ü programlayınız.
2. Bu programı denemek için breadboard üzerinde kurduğunuz devre üzerinde değişiklik yapmak zorundasınız. Dikkat ederseniz PortB çıkışının tümüne LED bağlamıştık. Şimdi RB1 ucuna bağladığımız. LED'i çıkarıp, buton bağlamamız gerekiyor. Diğer bağlantılar aynen kalabilir.
3. PIC'i kurduğunuz devre üzerine yerleştirip gerilim uygulayınız.
4. A1 butonuna basınız. Butona bastığınız sürece LED1'in yanık kaldığını görünüz.
5. Programın çalışması esnasında bir kesme meydana getirmek için INT butonuna kısa süreyle basınız. LED2 yanıkça, sönecektir. Eğer sönük durumdaysa yanacaktır.

Sonuç: Görülüyor ki, PIC programlarında kesme kullanıldığında mikro denetleyicinin işlem yapma gücünü genişletebiliyoruz. Ana program devamlı olarak işlevini sürdürüyor(LED1' istediğimiz zaman yakıp, söndürebiliyoruz.) Ancak dışarıdan gelen bir sinyal bu işlemi kısa bir süreyle kesip başka bir işi (LED2'nin durumunu değiştiriyor.) yapıyor ve akış yine ana programa geçiyor.

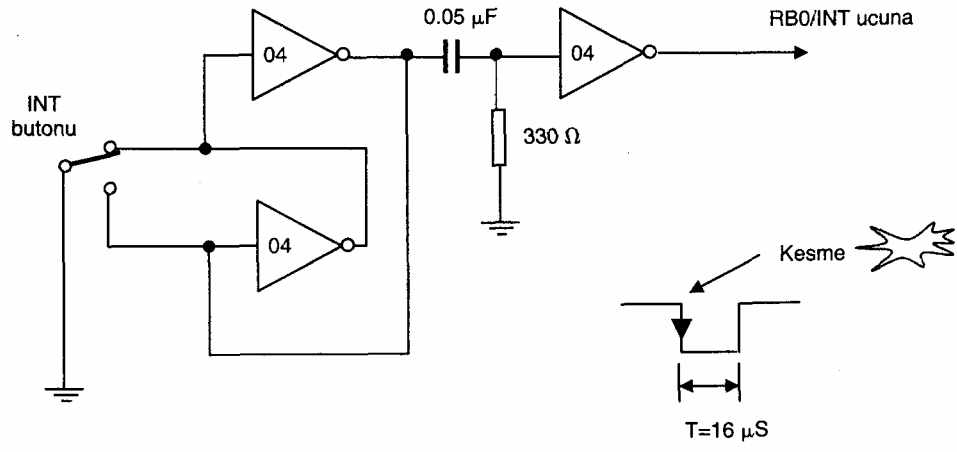
PIC'e uygulanan kesmeler istenirse periyodik olarak yapılabilir. Örneğin:

kesme girişi olarak bir clock sinyali kullanılabilir. Bir sayıcının belirli bir değere ulaştığında ürettiği sinyal de kesme meydana getirilebilir. Periyodik kesme uygulama program örneklerini 13. bölümde TMRO ve WDT sayıcılarını anlatırken vereceğiz.

Tek Pals Üreticinin Kullanımı

Program-25'de RBO girişi bir direnç vasıtasıyla pull-up yapılmıştır. INT butonuna basıp bırakılınca "1_r" biçiminde bir sinyal oluşur. Bu sinyalin ilk düşen kenarında kesme oluşur. Ancak butona basılıp bırakılma esnasında ark da oluşabilir. Oluşan bu arklar, arka arkaya birden fazla kesmenin meydana gelmesine sebep olabilir. Gerçi bu kesmeleri önlemek etmek için mikroişlemci otomatik olarak GEI bayrağını "0" yapar, ama kesme alt programının çok kısa sürede bitmesi nedeniyle arktan oluşan sinyaller yeni bir kesme olarak algılanabilir. Bu durumda LED2'nin yanma durumunu değiştirmesinde karışıklıklar oluşabilir.

Eğer programı çalıştırdığınızda bir aksaklık olursa, bu aksaklığın INT butonundan meydana gelebileceğini biliniz. INT ucuna uygulanan kesme sinyalindeki arkı önlemek ve pals süresini kısaltmak için aşağıdaki "One-shut-palse (tek pals üretici)" devresini kullanmanız gerekir. INT butonunun çift konumlu ve yaylı olmamasına dikkat ediniz. Devrede kullanılan entegre 74HC04 entegresidir.



13

DONANIM SAYICILARI

- ❑ DONANIM SAYICISI / ZAMANLAYICISI NEDİR?
- ❑ TMR0 SAYICI /ZAMANLAYICISI (TMR0 Counter/Timer)
- ❑ OPTION REGISTER
 - ❑ TMR0 SAYICISININ ÖZELLİKLERİ
 - ❑ WDT ZAMANLAYICISI (Watchdog Timer)

DONANIM SAYICISI/ZAMANLAYICISI NEDİR?

Önceki bölümlerde port çıkışlarına gönderdiğimiz sinyaller arasında bir gecikme olmasını istediğimiz zaman "gecikme" adında bir alt program oluşturduk. İşte programcının yaptığı bu tip zaman geciktirme alt programlarına "**software timer**-yazılım zamanlayıcısı" denir.

PIC mikrodenetleyicilerinin önemli özelliklerinde biri de, PIC donanımının içerisinde yazılım zamanlayıcısı ile aynı görevini yapan "**hardware timer** donanım zamanlayıcısı" bulunmasıdır. Bir zamanlayıcı aynı zamanda sayıcıdır da. Peki donanım sayıcısı ne demektir? Belirli bir değerden (Genellikle h'00'dan) başlayıp, içerisindeki sayıları birer birer artan bir file registerdir. Sayı artışı PIC içerisindeki dahili komut saykılı vasıtasıyla otomatik olarak yapılır. Sayma aralıkları nedeniyle bir gecikme olmaktadır. Bu gecikme kullanılarak program içerisinde zaman geciktirme döngüleri düzenlenebilir.

Türkçe'ye tam olarak yerleşmeyen zamanlayıcı ve sayıcı kavramları yerine "timer" ve "counter" kelimelerini kullandığımızda şöyle okuyunuz;

Timer (Taymır) →zamanlayıcı

Counter (Kauntır) →sayıcı

PIC'ler iki tip timer'a sahiptir. İlki, TMR0 olarak adlandırılan 8-bit bir sayıcıdır. RAM belleğin h'01' adresinde TMRO ile ifade edilen özel bir registerdir. İkincisi de, "VWatchdog timer-WDT" adı verilen zamanlayıcıdır. Farklı PIC versiyonlarında işlevleri değişen ve TMRO, TMR1 gibi adlar verilen timer'lar bulunmaktadır.

TMR0 SAYICI/ZAMANLAYICISI(TIMER/COUNTER)

RAM belleğin h'01' adresinde TMRO adı verilen özel bir register TMRO sayıcısıdır. TMRO programlanabilen bir sayıcıdır. Yani saymaya istenilen bir sayıdan veya h'00'dan başlatılabilir, herhangi bir anda içeriği sıfırlanabilir. Bu sayıcıyı detaylı olarak incelemeden önce özellikleri şöyle sıralayabiliriz:

- 8-bit bir sayıcıdır.
- Yazılabilir/okunabilir.
- Programlanabilen frekans bölme değeri (Prescaler value)
- Harici veya dahili clock ile sayı artışı
- Düşen veya yükselen kenar tetiklemesi (harici olarak)
- Sayıcı değeri artan yöndedir (H'00',h'01',h'02'.....h'FF')
- TMR0'ın değeri h'FF'den h'00'a gelince ilgili flag'i "1" yaparak kesme oluşturur.

TMRO sayıcısının önemli özelliklerinden birisi de ana program veya kesme alt programları çalışırken sayma işlemini durdurulmasıdır. Sayma işlemi devam ederken h'FF' den h'00'a geçişte meydana gelen taşma (overflow), INTCON registerin 2. bit'inde (T0IF bayrağı) görülür. Yani, bu esnada T0IF bayrağı "1" olur. Gerekliyse bu bayrak kontrol edilerek kesme oluşturulabilir.

Şimdi TRMO ve WDT sayıcılarının kontrolünde kullanılan OPTION registerden bahsettikten sonra ayrıntılara geçelim.

OPTION REGISTER

Option register, RAM belleğin 1. bank'ındaki h'81' adresinde bulunan özel bir registerdir. B portunun çıkışlarının pull-up yapılma durumunu kontrol eden, kesme sinyalinin tetikleme kenarını seçen, TMRO ve WDT için frekans bölme sayısı için gerekli bit'leri bulunduran, TMRO veya VVDT'yi seçme bayrağı bulunan ve tüm bu işleri kontrol eden 8-bit'lik bir registerdir. Aşağıda bu registerin her bir bit'inin ne görevleri yaptığı açıklanmıştır.

7	6	5	4	3	2	1	0
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

Frekans bölme sayısı
(Prescaler value)

PSA: Frekans bölücü seçme bit'i

0: Frekans bölme sayısı TMR0 için geçerli

1: Frekans bölme sayısı WDT için geçerli

T0SE: TMR0 sinyal kaynağı kenar seçme bit'i

0: RA4/T0CKI ucundan düşen kenar tetiklemesi

1: RA4/T0CKI ucundan yükselen kenar tetiklemesi

T0CS: TMR0 sinyal kaynağı seçme bit'i

0: Dahili komut saykılı seçilir.

1: Harici dijital sinyal(RA4/T0CKI ucu)

INTEDG: Harici kesme sinyali kenar seçme bit'i

0: RB0/INT ucundan düşen kenarda tetikleme

1: RB0/INT ucundan yükselen kenarda tetikleme

RBPU: PortB pull-up geçerli yapma bit'i

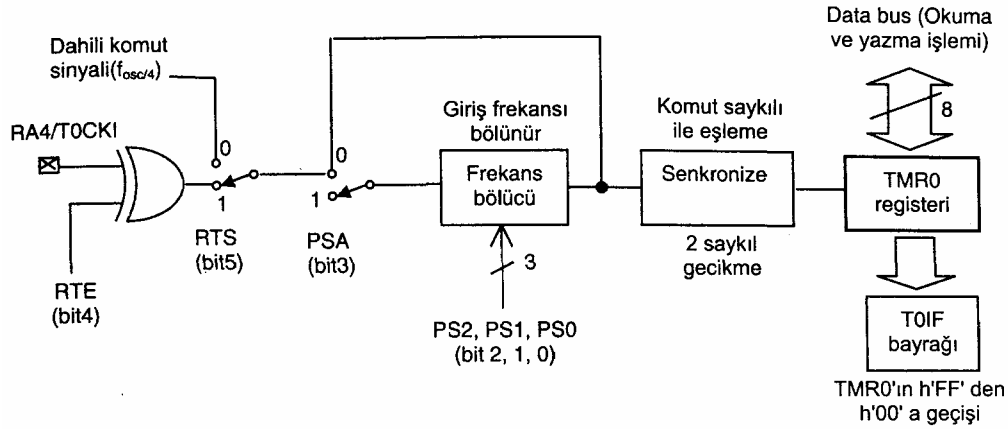
0: PortB uçlarındaki pull-up'lar iptal edilir.

1: PortB uçlarındaki pull-up'lar geçerli yapılır.

Frekans bölme sayısı	TMR0 oranı	WDT oranı
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

TMRO SAYICININ ÖZELLİKLERİ

TMRO Sayıcısının özelliklerini ve çalışmasını anlamak aşağıdaki blok şema üzerinde daha kolay olacaktır.



Sayıcı içindeki sayının artırılması için gerekli olan clock sinyali iki farklı kaynaktan sağlanabilir:

1. Dahili komut sinyali,
2. Harici dijital sinyal (RA4/T0CKI ucu).

PIC'i çalıştıran osilatör olarak kristal kullanıldığında dahili komut sinyali de çok hassastır. Bu nedenle dahili komut sinyali seçildiğinde sayma ve zamanlama işlemleri çok hassas olarak yapılabilir.

• Harici dijital sinyal portA'nın 3. bit'inden (RA4/T0CKI) uygulanır.

Bazı işlemler için dahili komut sinyalinin frekansı çok yüksek olabilir. Bu durumda daha düşük frekanslı harici bir sinyal kaynağı kullanılabilir. Harici dijital sinyal, PIC tarafından sayılması gereken bir sinyal de olabilir.

• TMR0'ın sinyal kaynağını seçmek için OPTION registerin 5. bit'i (T0CS) kullanılır.

• Sinyal kaynağından gelen sinyal, direkt olarak TMR0'ı besleyebileceği gibi frekans bölücü aracılığı ile de beslenebilir.

• Frekans bölme sayısı TMR0 veya WDT'ye OPTION registerin 3. bit'i (PSA) ile yönlendirilir. Her iki sayıcıdaki bölme oranları birbirinden farklı olduğuna dikkat ediniz.

• OPTION registerin 0, 1, 2. bit'leri kullanılarak 8 farklı frekans bölme değeri seçilebilir. TMR0 sayıcısını tetikleyecek olan sinyal direkt olarak sinyal kaynağından sağlanmak isteniyorsa, yani frekans bölücü bypas yapılmak isteniyorsa, frekans bölme değeri WDT'ye atanır. Bu işlemin PSA bit'i ile yapıldığını söylemiştik.

- Frekans bölme değeri TMR0'a atandığında, TMR0'a yazmak için kullanılan tüm komutlar frekans bölme değerini siler. Bu komutlar şunlar olabilir.

CLRF, MOVWF, BSF gibi....

- Frekans bölme değeri kullanılmadan, direkt olarak harici sinyal kullanılırsa, dahili komut sinyali ile eşlemeyi(senkronizasyonu) sağlamak için 2 sayıklık bir gecikme sağlanır.(Blok şemaya bakınız.)
- Harici dijital sinyal uygulandığında TMRO sayıcısının içerisindeki sayının, sinyalin hangi kenarında (Yükselen/düşen) artacağını belirlemek için OPTION registerin 4. bit'i (T0SE) aracılığı ile belirlenir.
- TMR0 sayıcısından çıkış sinyalleri aşağıdaki durumlarda oluşur:
 1. RAM belleğin h'01" adresindeki TMRO registeri okununca.
 2. Sayıcının h'FF'den h'00'a geçişinde meydana gelen taşma sinyalinin INTCON registerinin 2. bit'ine (TOIF bayrağı) "1" yazılması anında.
- TMR0'daki sayıları artıran, sayıcıya uygulanan sinyallerdir.

Sayılar, harici dijital sinyal aracılığı ile artırıldığında istenirse bu sinyaller sayılabilir. Bu durumda TMR0 sinyal sayıcı olarak kullanılmış olur.

Frekans Bölme Sayısının (Prescaler) Kullanılması

OPTION registerin 0, 1,2. bit'leri (PS0-PS2) içerisine yerleştirilen sayılar, TMRO veya WDT'ye uygulanan sinyali böler. Böylece sayma hızları değiştirilebilir. 3 bit'lik bu sayı TMRO veya WDT'de birbirinden farklı 8 farklı oran seçme olanağı yaratır. PSA bit'i, prescaler sayısının hangi sayıcıya uygulanacağını belirler. Aşağıda hangi prescaler değerinde giriş frekansının kaç bölüneceğini gösteren tablo verilmiştir.

Frekans bölme sayısı (Prescaler)	TMR0 oranı	WDT oranı
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

Örneğin,

Prescaler değeri b'000' seçilirse, TMR0 oranı 1/2, WDT oranı 1/1 seçilir. Prescaler değeri b'010' seçilirse, TMR0 oranı 1/8, WDT oranı 1/4 seçilir. Prescaler değeri b'111' seçilirse, TMR0 oranı 1/256, WDT oranı 1/128 seçilir.

TMR0 ve WDT Oranı

TMR0 veya WDT sayıcılarının kaç dahili komut sayıklında bir defa bir üst sayıya geçişini belirleyen orandır. Örneğin,

TMR0 oranı 1/2 ise, 2 komut sayıklında bir defa üst sayıya geçiş olur. TMR0 oranı 1/8 ise, 8 komut sayıklında bir defa üst sayıya geçiş olur. TMR0 oranı 1/256 ise, 256 komut sayıklında bir defa üst sayıya geçiş olur.

Bilindiği gibi program belleğine yerleştirilen komutların çalışması için PIC'e harici bir osilatörden clock sinyali (fosc) uygulanması gerekir. Bu frekans PIC tarafından 4'e bölünerek OSC2 ucundan dışarıya verilir. İşte 4'e bölünen bu frekansın bir sayıklı bir komutun çalışması için geçen süredir. Örneğin, 4 MHz lik bir kristal osilatör kullanılan PIC'te, dahili komut sayıklı (Periyodu) süresi şöyle bulunur

$$f_{\text{komut}} = f_{\text{osc}}/4 = 4 \text{ MHz}/4 = 1 \text{ MHz}$$

$$T_{\text{komut}} = 1/f_{\text{komut}} = 1/1 \text{ MHz} = 1\mu\text{S} \text{ (Dahili komut sayıklı süresi)}$$

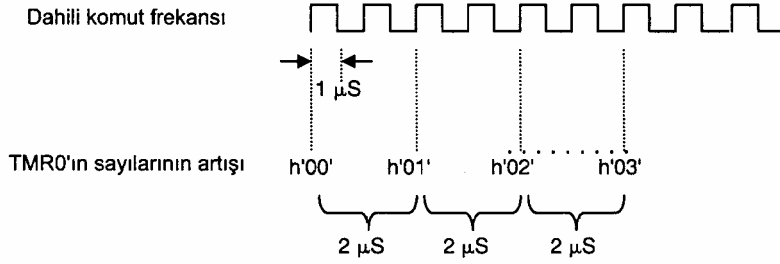
Şimdi de clock frekansı bilinen bir PIC'te TMR0 sayıcısının kaç μS aralıklarla saydığını veya kaç μS aralıklarla kesme sinyali verdiğinin nasıl hesaplanacağı ile ilgili örnekler verelim.

Örnekler:

1. Prescaler değeri b'000' seçilirse TMR0'ın sayma aralığı süresi ne olur?

Prescaler= b'000' olduğunda →TMR0 oranı 1/2 olur.

$f_{\text{OSC}} = 4 \text{ MHz}$ olarak alınırsa → $T_{\text{komut}} = 1 \mu\text{S}$ olur.



Görülüyor ki, TMR0 sayıcısı dahili komut saykılıının 2 saykılında 1 defa artıyor. Böylece TMR0'ın sayma aralığı süresi kolayca hesaplanmaktadır.

$$\begin{aligned} \text{TMR0 sayma aralığı süresi} &= T_{\text{komut}} \times \text{TMR0 oranı} \\ &= 1 \mu\text{S} \times 2 = 2 \mu\text{S} \end{aligned}$$

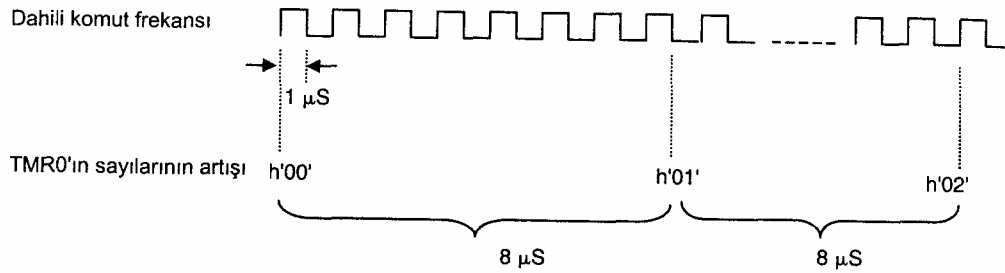
TMR0 saymaya başladığındaki ilk sayı h'00' ise bu sayı h'FF'e gelince kesme sinyali oluşturacaktır. Bu sayı aralığı ondalık olarak 256 eder. öyleyse TMR0'ın kaç saniye aralıklarla kesme sinyali vereceğini bulalım:

$$\begin{aligned} \text{Kesme gecikmesi} &= \text{TMR0 sayma aralığı} \times 256 \\ &= 2 \mu\text{S} \times 256 = 512 \mu\text{S} \end{aligned}$$

2. Prescaler değeri b'010' seçilirse TMR0'ın sayma aralığı süresi ve kesme gecikmesi kaç μS olur?

Prescaler=b'010' olduğunda → TMR0 oranı 1/8 olur.

$f_{\text{osc}} = 4 \text{ MHz}$ olarak alınırsa → $T_{\text{komut}} = 1 \mu\text{S}$ olur.



TMR0 sayıcısı dahili komut saykılıının 8 saykılında 1 defa artmaktadır.

$$\text{TMR0 sayma aralığı süresi} = T_{\text{komut}} \times \text{TMR0 oranı} = 1 \mu\text{S} \times 8 = 8 \mu\text{S}$$

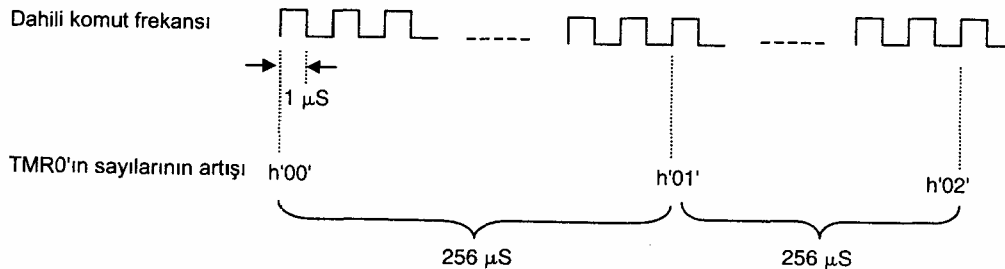
TMR0 saymaya başladığındaki ilk sayı h'00' ise TMR0'ın kaç saniye aralıklarla kesme sinyali vereceğini bulalım:

$$\begin{aligned} \text{Kesme gecikmesi} &= \text{TMR0 sayma aralığı} \times 256 \\ &= 8 \mu\text{S} \times 256 = 2048 \mu\text{S} \rightarrow 2 \text{ ms} \end{aligned}$$

- 3-) Prescaler değeri b'111' seçilirse TMR0'ın sayma aralığı süresi ve kesme gecikmesi kaç μS olur?

Prescaler=b'111' olduğunda → TMR0 oranı 1/256 olur.

$f_{\text{osc}} = 4 \text{ MHz}$ olarak alınırsa → $T_{\text{komut}} = 1 \mu\text{S}$ olur.



TMR0 sayıcısı dahili komut saykılıının 256 saykılında 1 defa artmaktadır.

$$\text{TMR0 sayma aralığı süresi} = T_{\text{komut}} \times \text{TMR0 oranı} = 1 \mu\text{S} \times 256 = 256 \mu\text{S}$$

TMR0 saymaya başladığındaki ilk sayı h'00' ise TMR0'ın kaç saniye aralıklarla kesme sinyali vereceğini bulalım:

$$\begin{aligned} \text{Kesme gecikmesi} &= \text{TMR0 sayma aralığı} \times 256 \\ &= 256 \mu\text{S} \times 256 = 65536 \mu\text{S} \rightarrow 65.5 \text{ ms} \end{aligned}$$

Frekans Bölme Sayısının Atanması

TMR0 sayısını direkt olarak sinyal kaynağından beslemek için yapılacak işlem, frekans bölme sayısını WDT'ye atamaktır. Prescaler değeri WDT'ye atandığında, TMR0'a yazmak için kullanılan CLRF, MOVWF, BSF vs. konutlar prescaler değerini sıfırlayarak yeni bir prescaler değeri girmeye hazırlar.

Prescaler değerini TMR0'dan WDT'ye veya WDT'den TMR0'a atama işlemi yapılırken prescaler'ın sıfırlanması nedeniyle PIC'in çalışması esnasında istenmeyen reset'ler oluşabilir. Bu reset'lemeleri önlemek için aşağıdaki program parçaları muhakkak kullanılmalıdır.

TMR0'dan WDT'ye Prescaler Değeri Atamak:

```
BCF STATUS, 5 ; BANK0'a geç.
CLRF TMR0 ; TMR0 sayıcısı ve prescaler silinir.
BSF STATUS ; BANK1'e geç.
CLRWDI ; WDT'yi sil.
MOVLW b'xxxxlxxx' ; Yeni prescaler değerini seç.
MOVWF OPTION_REG ; OPTION registere yaz.
BCF STATUS, 5 ; BANK0'a geç.
```

WDT'den TMR0'a Prescaler Değeri Atamak:

```
CLRWDI ; WDT'yi ve prescaler'ı sil.
BSF STATUS, 5 ; BANK1'e geç.
MOVLW b'xxxx0xxx' ; TMR0'ı, yeni prescaler değerini ve sinyal kaynağını seç.
MOVWF OPTION_REG ; OPTION registere yaz.
BDF STATUS, RP0 ; BANK0'a geç.
```

TMRO Sayıcısının Kullanılması

TMRO sayıcısının kurulması:

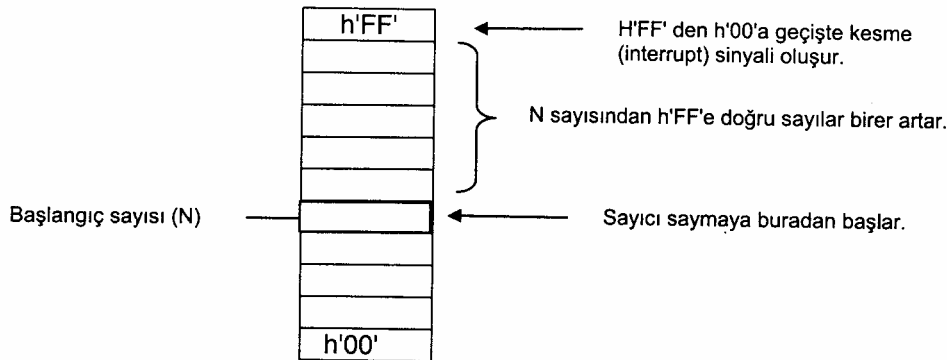
- Yukarıda verilen komutlar aracılığı ile prescaler değerinin atanmasıyla.
- OPTION registerdeki bit'lere uygun veri göndererek.

TMRO sayıcısının başlatılması:

- TMR0 sayıcısına bir sayı yazmak, sayıcıyı başlatır.

Saymaya başlama sayısının değiştirilmesi:

- TMR0 sayıcısı h'00'dan değil, istenirse başka bir sayıdan başlatmak mümkündür. Bunun için TMR0 registerine istenilen sayının atanması yeterlidir.



TMRO'ın çalıştığını nasıl anlarız?

- TMR0 registerini herhangi bir anda okuyarak.
- TMR0 registerinin herhangi bit'ini test ederek.
- H'FF' den h'00'a geçişte oluşan kesme sinyalini test ederek. Kesme anında INTCON registerinin 2. bit'i (T0IF bayrağı) "1" olur.
- TMR0'ın çalışması esnasında PIC16F84 yüklendiği diğer işleri yapmaya devam eder.

Neler yapılmadığı sürece TMRO çalışmasına devam eder?

- Silinmediği sürece (CLRF TMR0)
- içerisine sayı yazılmadığı sürece (Örneğin MOVLW h'05', MOVWF TMR0)
- Mikrodenetleyici RESET yapılmadığı sürece.

Her kesme oluşunda TMR0'ın başlangıç sayısı yeniden yüklenmeli mi?

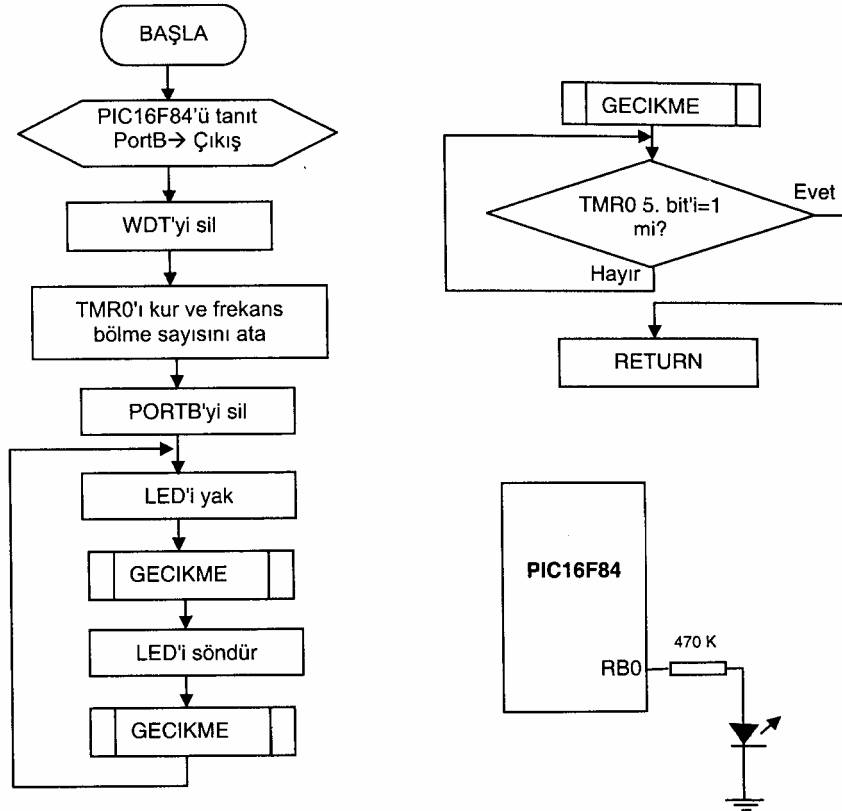
- Evet, çünkü h'FF'e ulaşan sayıcı otomatik olarak h'00'dan başlar.
- Eğer başlangıç sayısı devamlı olarak belirli bir (N) sayısı olması isteniyorsa, her kesme oluştuğunda bu sayı TMR0'a atanmalıdır.

TMR0'ı durdurmak mümkün müdür?

- Hayır, PIC çalıştığı sürece saymaya devam eder. Fonksiyonlarından yararlanılmak istenmiyorsa, program içerisinde kullanılmaz. Ancak kesme bayrağını (TOIF) "1" yapmaya devam eder.

PROGRAM-26) PortB'nin 0. bit'ine bağlı LED'i flash yaptıran program. LED'in yanıp sönmeye aralıklarındaki gecikmeyi TMR0 sayıcısı yapmaktadır. Bu program dijital çıkış sinyali üretmek için kullanılmıştır. TMR0'ın sinyal kaynağı olarak **dahili komut sayıklı** kullanılmış ve TMR0 oranı 1/256 seçilmiştir.

Kristal osilatör kullanılan uygulamalarda kesme gecikmesi çok kısa olduğundan LED'in flash yapması görülmeyebilir. Bu durumda RB0 ucuna osiloskop bağlayarak çıkış izlenebilir. Eğer RC osilatör kullanılıyorsa seçilen R ve C değerlerine göre frekans çok düşürülürse LED'in yanıp söndüğü görülür.



```
=====PROG26.ASM=====s=09/07/2000=====
LIST      P=16F84
INCLUDE "P16F84.INC"
CLRF  PORTB
BSF   STATUS, 5      ;Bank1
CLRF  TRISB          ;PORTB tüm uçları çıkış
BCF   STATUS, 5      ;Bank0

BASLA
CLRWDT          ;Prescaler atama işlemine hazırla
BSF   STATUS, 5      ;Bank1
MOVLW  b'11010111'   ;TMR0'ı, yeni prescaler değerini
                        ;ve sinyal kaynağını seç
MOVWF  OPTION_REG    ;OPTION registere yaz.
CLRF  PORTB YAK
BSF   PORTB, 0      ;LED'i yak.
CALL  GECIKME        ;GECIKME alt programını çağır. SOMDÜR
BCF   PORTB, 0      ;LED'i söndür.
CALL  GECIKME        ;GECIKME alt programını çağır.
GOTO  YAK            ;Yakıp-söndürmeye devam et.

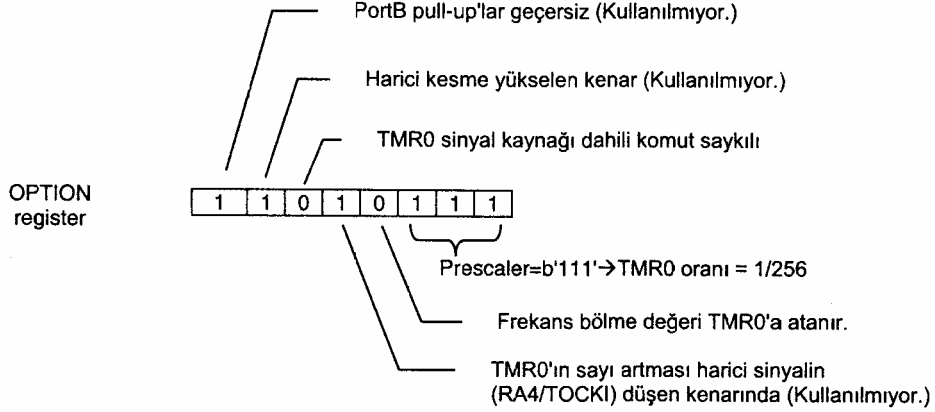
GECIKME
CLRF  TMR0          ;TMR0'ı h'00'a kur, saymaya başla
```

```

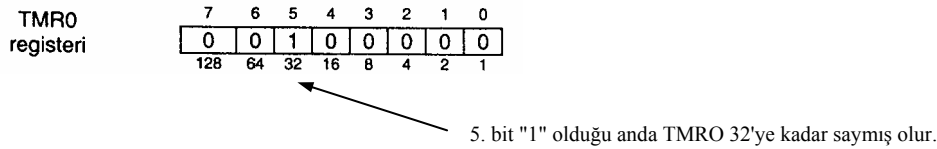
TEST_BIT
  BTFSS TMR0, 5      ;TMR0'ın 5. bit'i "1" mi?
  GOTO  TEST_BIT      ;Hayır, 5. bit'i tekrar test et.
  RETURN              ;Ana programa dön.
END

```

Programda OPTION registre atanan verilerin anlamı aşağıdaki gibidir.

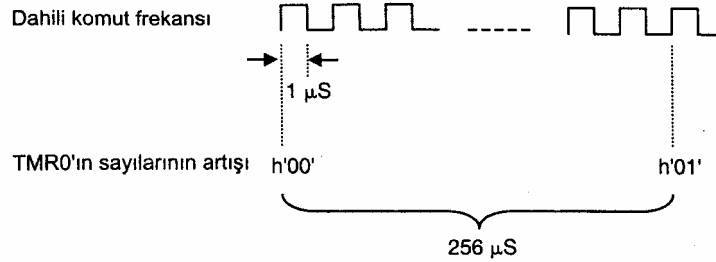


TMR0 registerinin tamamı okunabilir veya bu örnekte olduğu gibi sadece bir bit'i test edilir. TMR0'ın 5. bit'i "1" olduğunda ulaşılan sayı 32'dir (desimal). Yani sayıcı 0'dan 32'ye kadar saydırılmaktadır.



Deneme devresinde 4 MHz lik kristal osilatör kullanıldığı düşünülürse; Bir komut saykılı 1 μ S'dir.

TMR0 içerisindeki sayılar 256 komut saykılında bir defa artacağından (TMR0 oranı 1/256 olduğundan.)



TMR0, 32'ye kadar sayacağından 0'dan 32'ye kadar sayma süresi:

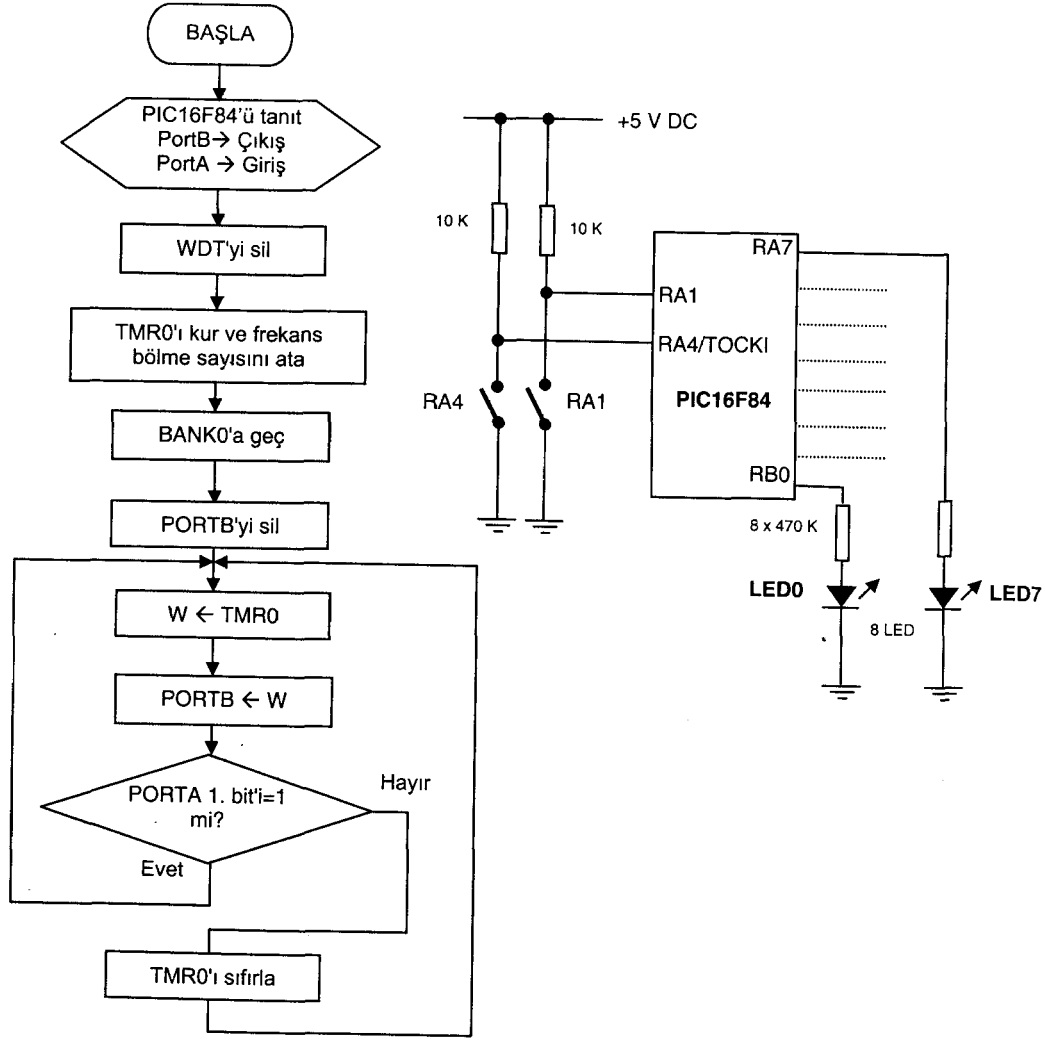
$256 \mu\text{S} \times 32 = 8192 \mu\text{S} \rightarrow 8.2 \text{ mS}$ eder.

Biz deneme devrelerinde RC osilatör kullandığımızdan bu süre çok daha uzun olabilir.

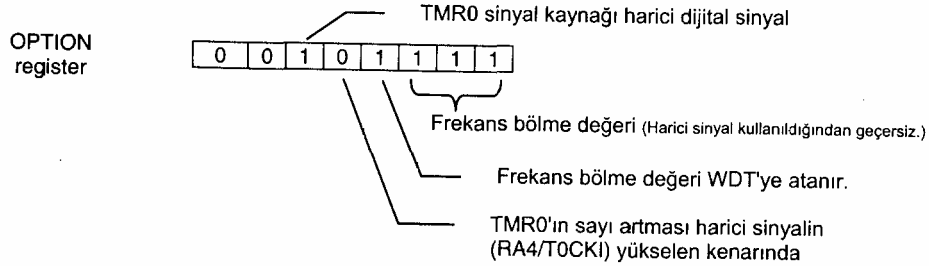
İşlem Basamakları

1. PROG26.ASM programını derledikten sonra PIC16F84'e yazdırınız.
2. Breadboard üzerinde kurduğunuz devre üzerindeki RC osilatörün frekansını oldukça düşürmek için $R=10 \text{ K}$, $C=0.1 \mu\text{F}$ seçiniz. Yaklaşık olarak 85 Hz'lik bir frekans elde edeceksiniz.
3. PIC'i devrenize yerleştirerek programı çalıştırınız. PortB'nin 0. bit'indeki LED'in flash yaptığını görünüz.
4. Deneme devrenizde kristal osilatör kullanıyorsanız LED'in flash yaptığını göremeyeceksiniz. Bu durumda bir osilaskop bağlayarak elde edilen kare dalgayı görün ve bir saykılının kaç μS de oluştuğunu ölçünüz.

PROGRAM-27) TMR0 sayıcısının sinyal kaynağı olarak **harici dijital sinyal** (RA4/T0CKI ucu) kullanılmasına örnek programdır. Ayrıca TMR0'ın okunabilme ve sinyal sayıcı olarak kullanılabilme özelliğini de gösterir. Program, PortA'nın 3. bit'ine bağlı olan butona (RA4) basıldığında portB'deki LED'lerde binary olarak artan sayıları gösterir. RA1 butonuna basınca TMR0 registerini sıfırlar ve saymaya 0'dan itibaren tekrar başlatır.



Programda OPTION registre atanan verilerin anlamı aşağıdaki gibidir.



TMR0 sinyal kaynağı olarak RA4 girişi seçildiğinden, bu butona her basışta TMR0'ın içerisindeki sayı 0'dan itibaren yükselir. RA4 butonuna basılmadığı sürece (Yeni bir sinyal gelinceye kadar) TMR0 içerisindeki sayı sabit kalır. TMR0'ın sıfırlanması yine harici bir girişle (RA1 butonu) yapılıyor.

```

=====PROG27.ASM=====12/07/2000=====
LIST P=16F84
INCLUDE "P16F84.IMC"
CLRF PORTB
BSF STATUS, 5 ;Bank1
CLRF TRISB ;PORTB tüm uçları çıkış
MOVLW h'FF' ;W←h'FF'
MOVWF TRISA ;PortA tüm uçları giriş
BCF STATUS, 5 ;Bank0

BASLA
CLRF PORTB ;PortB'yi sil
CLRF TMR0 ;TMR0 sayıcısı ve prescaler'ı sil

```

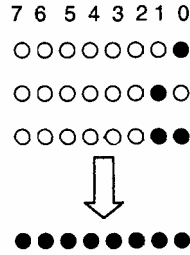
```

CLRWDW      ;WDT'yi sil
BSF STATUS, 5 ;BANK1
MOVLW b'00101111' ;Prescaler deęerini ata
MOVWF OPTION_REG ;OPTION registere yaz.
BCF STATUS, 5 ;BANK0
CLRF PORTB ;PortB'yi sil
DONGU
MOVF TMR0, W ;W←TMRO
MOVWF PORTB ;PORTB←W
BTFS PORTA, 1 ;PortA 1. bit "0" mı?
CLRF TMRO ;Evet, TMR0'ı sıfırla
GOTO DONGU ;Hayır, tekrar test et
END

```

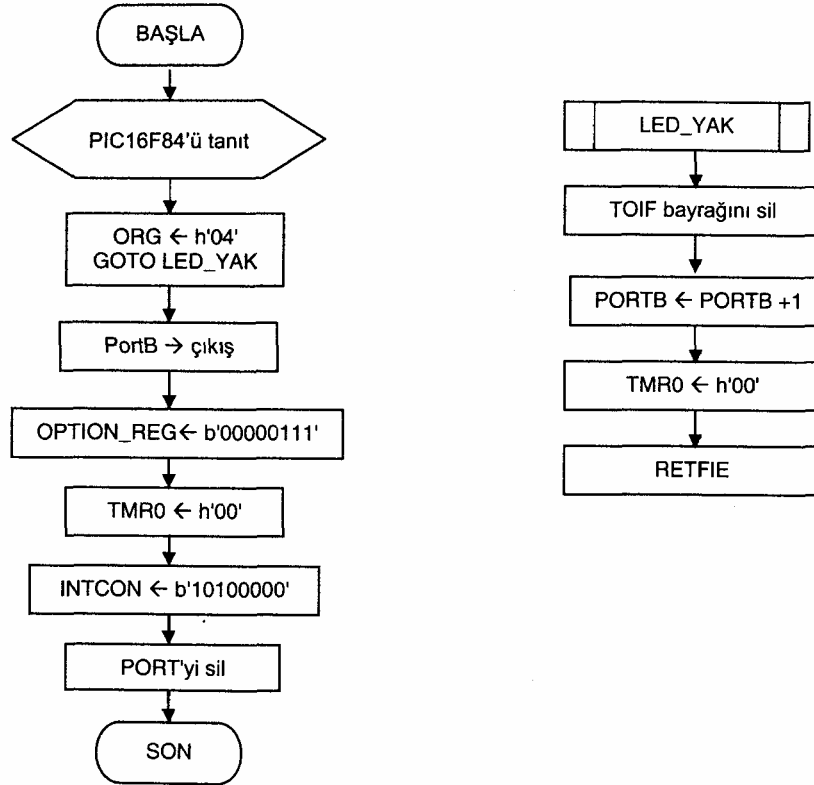
İşlem Basamakları

- 1 PROG27.ASM'yi PIC16F84'e yazdırıp, deneme kartına yerleştiriniz.
- 2 Deneme kartına gerilim uygulayarak programı çalıştırınız.
- 3 A4 (PortB'nin 3. bit'ine baęlı) butona bastıkça PORTB'deki 8 LED'in binary artan sayıları göstermesini saęlayınız.



TMR0 Sayıcı Kesmesine Ait Örnekler

PROGRAM-28) TMR0 sayıcı kesmesine örnek programdır. PortB'deki LED'lerin binary olarak artan sayıları göstermesini saęlar. Bu programdaki LED'lerin sayma aralıklarındaki duruşu için TMRO zamanlayıcısı kullanılmıştır.

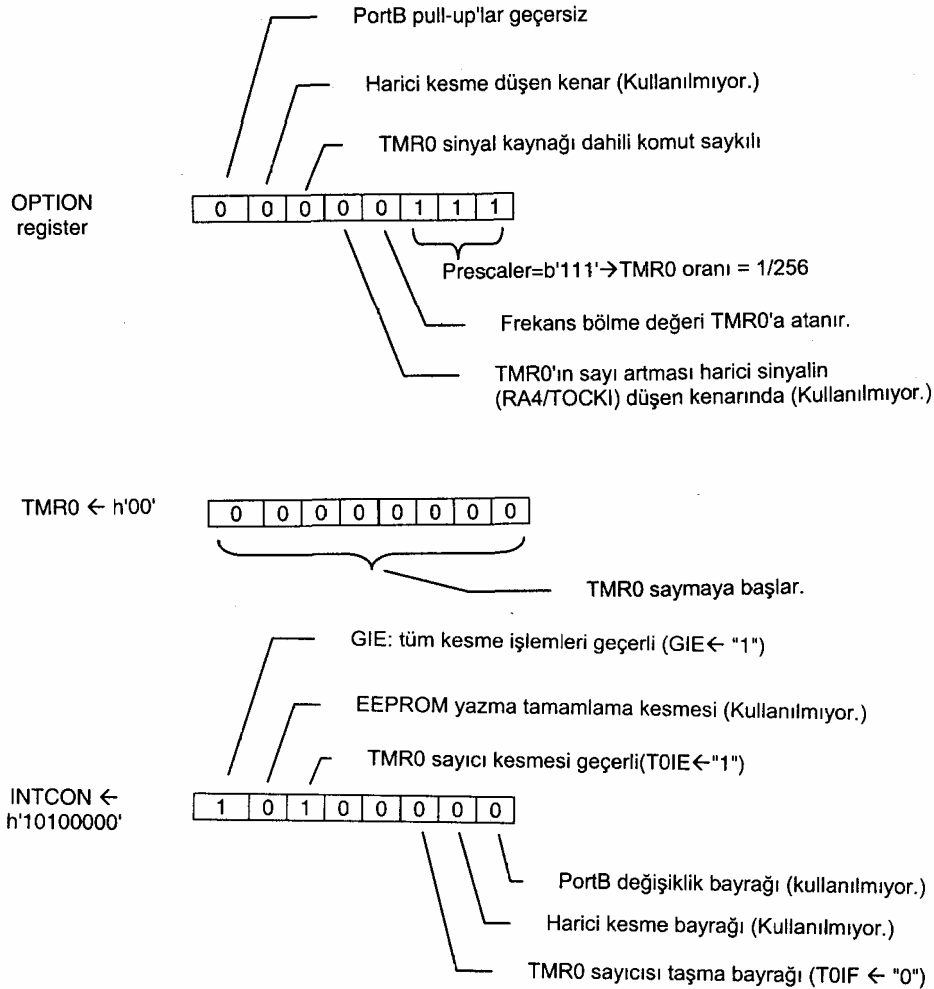


```

;==PROG28.ASM====17/07/2000=====
LIST P=16F84
INCLUDE "P16F84.INC"
ORG h'00'
GOTO BASLA
ORG h'04'
GOTO LED_YAK ;Kesme alt programına git.
BASLA
BSF STATUS, 5 ;Bank1
CLRF TRISB ;PORTB tüm uçları çıkış
MOVLW b'00000111' ;W←b'00000111' 1/256
MOVWF OPTION_REG ;OPTION_REG←W
BCF STATUS, 5 ;BANK0
MOVLW h'00' ;W←h'00'
MOVWF TMR0 TMR0←W
MOVLW b'10100000' ;GIE←"1", TOIE←"1", TOIF←"0"
MOVWF INTCON ;INTCON←W
CLRF PORTB ;PortB'yi sil
DONGU
GOTO DONGU
LED_YAK
BCF INTCON, TOIF ; TOIF←"0"
INCF PORTB, F ; PORTB←PORTB+1
MOVLW h'00' ; W←h'00'
MOVWF TMR0 ; TMR0←W
RETFIE ; Alt programdan dön
END

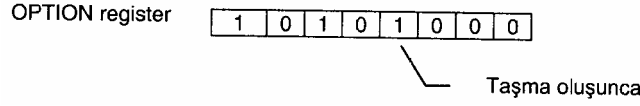
```

Programdaki OPTION ve INTCON registerlerin kurulmasını ve programın açıklamasını şekillerle açıklayalım.

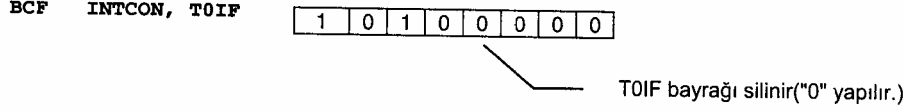


LED_YAK kesme alt programında yapılan işlemler

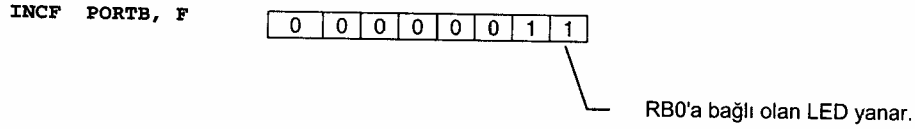
TMR0'ın içerisindeki sayı h'FF'e ulaşınca kesme oluşur ve OPTION registerinin 3. bit'i (TOIF) "1" olur.



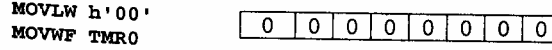
TOIF bayrağını sonraki kesmelere hazırlama için tekrar "0" yapılmalıdır.



Program içerisinde PORTB'ye bağlı olan LED'lerin binary artan sayıları göstermesi için PORTB'nin içeriği her defasında "1" artırılır.



TMR0 sayıcısının tekrar h'00'dan itibaren sayması için TMR0'a h'00' sayısı atanmalıdır. Bu sayı atanmaması halinde yine h'00'dan başlayacaktır. Ancak, saymaya başlama sayısı farklı olması isteniyorsa, o sayı mutlaka TMR0'a atanmalıdır.



TMR0 Sayıcısını İstenilen Bir Sayıdan Başlatmak

TMR0'ın içerisine h'00' atandığında **maksimum kesme süresi**, h'FF' atandığında ise **minimum kesme süresi** elde edilir. Şimdi 4 Mhz clock osilatörü ile çalışan bir PIC'te prescaler 1/256 seçildiğinde elde edilen maksimum kesme süresini bulalım.

$$f_{osc} = 4 \text{ Mhz ise } \rightarrow T_{komut} = 1 \mu S$$

TMR0=0 olduğundan h'00' dan h'FF'e kadar 256 defa sayar.

Prescaler= 1/256 olduğundan, TMR0 sayıcısı 256 sayıkılda 1 defa sayılarını artırır. Öyleyse maksimum kesme süresi;

Kesme gecikmesi = $T_{komut} \times \text{Prescaler} \times (\text{h'FF} - \text{TMR0 ilk sayısı})$ formülüyle bulunur.

Kesme gecikmesi = $1 \mu S \times 256 \times (256 - 0) = 65536 \mu S \rightarrow 65.5 \text{ mS}$ eder.

Gecikme zamanını hesaplamak için f_{osc} frekansını kullanarak bir formül oluşturursak aşağıdaki formülü kullanabiliriz.

$$\text{Kesme gecikmesi} = \frac{(256 - \text{TMR0 ilk sayı}) \times \text{prescaler}}{f_{osc} / 4}$$

Kesme gecikmesi : TMR0'ın oluşturacağı kesme sinyali aralığı (μS)

TMRO ilk sayısı : TMR0'a atanan sayı (Ondalık sayı)

Prescaler : Frekans bölme sayısı (2, 4, 8, 16, 32, 64, 128, 256). Prescaler kullanılmadığı zaman bu sayıyı 2 alınız(Senkronizasyon nedeniyle.)

f_{osc} : PIC'e uygulanan osilatörün frekansı (MHz)

Eğer istenilen kesme gecikme süresini sağlamak için TMRO ilk sayısının ne alması gerektiğini bulmak için bu defa aşağıdaki formülü kullanabiliriz.

$$\text{TMR0 ilk sayı} = 256 - \left[\frac{\text{Kesme gecikme süresi} \times f_{osc}}{4 \times \text{Prescaler}} \right]$$

Örnek: 1.28 mS kesme gecikmesi sağlamak için TMRO içerisine hangi sayının atanması gerekir? ($f_{osc}=4 \text{ Mhz}$, Prescaler=128)

1.28mS-> 1280 μS eder.

$$TMR0 \text{ ilk sayıyı} = 256 - \left[\frac{1280 \mu S \times 4 \text{ MHz}}{4 \times 128} \right] = 256 - \frac{5120}{512} = 246$$

Heksadesimal karşılığı h'F6' dır.

TMR0'a h'FF' atandığında **minimum kesme süresi** oluşur. Ancak LED'in yanıp-sönme süresi 0 μS olmaz. Çünkü kesme alt program komutlarının çalışma süresi minimum kesme süresini oluşturur. Bu da kesme oluştuğunda GOTO LED_YAK komutuyla alt programa geçen akıştan sonra çalışan komutlardır. Bu süre RETFIE komutuyla biter. Şimdi bu süreyi hesaplayalım. $f_{OSC} = 4 \text{ MHz}$ olduğunu düşünelim.

Dahili komut saykılı = 1 μS

Ana programda TMR0'ın çalışmaya başlatılırken kullanılan komutlar:

MOVLVV h'FF'	1 saykıl
MOVWF TMR0	1 saykıl
Toplam	= 2 saykıl

Kesme oluşuktan sonra alt programda çalışan komutlar:

Kesme gecikmesi	2 saykıl
GOTO LED_YAK	2 saykıl
BCF TCON, T0IF	1 saykıl
INCF PORTB, F	1 saykıl
MOVLW h'00'	1 saykıl
MOVWF TMR0	1 saykıl
RETFIE	2 saykıl
Toplam =	10 saykıl

Yukarda yaptığımız hesaplama göre kesme oluştuğunda çalışan komutlar 10+2=12 komut saykılı süresi geçecektir. Bir komut saykıl 1 μS olduğuna göre toplam süreyi hesaplayalım;

10 saykıl x 1 $\mu S = 12 \mu S$ eder.

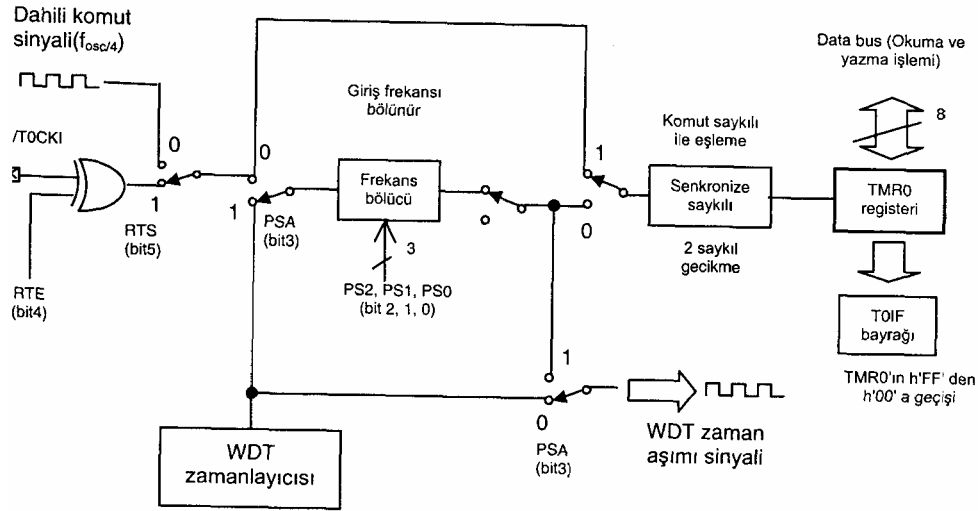
Görüldüğü ki, formülle hesaplama sonucunda 0 μS lik kesme süresi hesaplamaımıza rağmen, alt program komutlarının çalışması hesaba katılınca **minimum kesme süresi 12 μS olmaktadır** ($f_{osc} = 4 \text{ MHz}$ olduğunu düşündüğümüzde.)

WDT ZAMANLAYICISI (WATCHDOG TIMER)

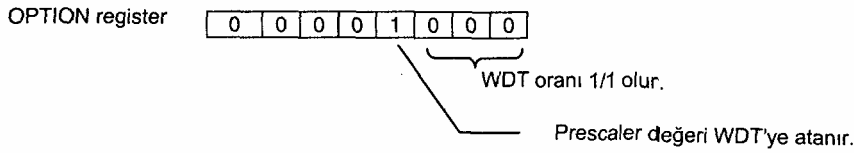
PIC donanımı içerisinde bulunan ikinci bir zamanlayıcıdır. WDT İngilizce'deki "WatchDog Timer" kelimesinin kısaltmasından gelir. Watchdog'un Türkçe'deki karşılığı "Bekçi köpeği"dir. Elbette mecazi anlamda kullanılan bir kelime olmasına rağmen, mikrodenetleyici içerisindeki programın bir yerde bekçiliğini yapmaktadır Nasıl mı? PIC'in önceden belirlenen sürede yapması planlanan bir işi zamanında yapmadıysa, yani döngü kontrolden çıkıp da kilitlendiyse, WDT devreye girer ve PIC'i reset eder. Genellikle program kontrolü elden kaçtığı durumda PIC'in kontrolünü tekrar ele alarak yazılım veya donanım problemlerinin bekçiliğini yapar WDT ayrıca SLEEP(Uyuma) moduna girmiş PIC'i uyandırarak yeniden çalışmaya devam etmesini sağlamak amacıyla kullanılır.

WDT zamanlayıcısının çalışmasına bir örnek olarak da şunu verebiliriz. Diyelim ki seri üretim yapan bir fabrikadaki bir bant üzerinden geçen ürünlerin montajını yapan robot, mikrodenetleyici vasıtasıyla yapılmaktadır Herhangi bir sorun olmadığında ürünlerin montajı PIC tarafından yapılmaktadır Bant üzerinde 3'ür aksaklık oluştuğunda ürün geçişi duracağından bir gecikme olacaktır Bu gecikme WDT zamanlayıcısının zaman aşımı süresini geçerse WDT bu anda devreye girerek ya bir alarm sistemi çalıştırılarak teknisyenleri uyaracak ya da 31-programı reset yaparak programın baştan çalıştırılmasını sağlayacaktır

Şimdi WDT zamanlayıcısının kullanılmasının çalışmasını blok şema çizerek açıklayalım. Blok şemada TMR0 zamanlayıcısının da birlikte çizilmesinin nedeni çoğu kurma işlemlerinin (OPTION ve INTCON registerleri) birlikte yapılmasındandır. Böylece bu registerlerin kurulması daha kolay anlaşılacaktır



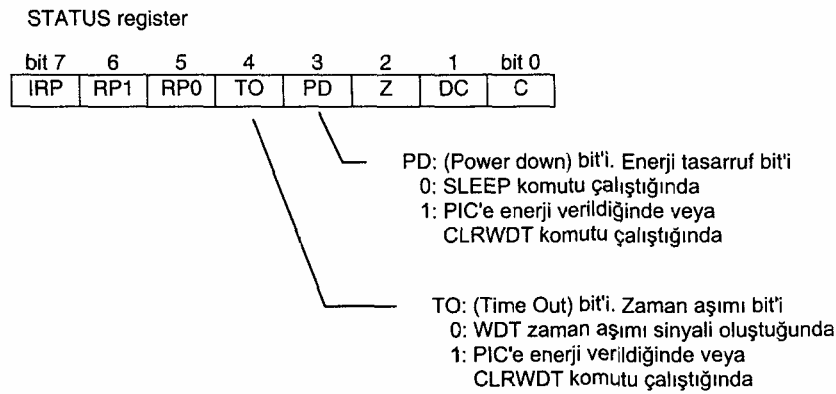
OPTION registerin WDT ile ilgili 4 bit'i vardır. WDT zamanlayıcısını seçmek için OPTION registerin 3. bit'i "1" yapılmalıdır. WDT oranının ise 0, 1, 2. bit aracılığı ile seçilen frekans bölme sayısı (prescaler) ile belirlendiğini söylemiştik. Seçilen prescaler değerine göre TMR0'ın ve VVDT'nin farklı oranlar aldığına dikkat ediniz.



WDT zamanlayıcısının sinyal kaynağı, PIC içerisine yerleştirilmiş bir osilatördür. Bu osilatörün WDT'ye sağladığı nominal zaman aşımı süresi 18 mS dir. Prescaler değeri artırılarak bu süre 2.3 saniyeye kadar çıkarılabilir.

Zaman Aşımı Süresi

WDT'nin CLRWDt komutuyla reset yapıldıktan sonra saymasını tamamlayıp, baştan tekrar saymaya başladığı anda kadar geçen süredir. WDT'nin saymasını tamamladığı anda zaman aşımı sinyali verir. Bu sinyal STATUS registerin 3. ve 4. bit'indeki flagların durumunu değiştirir.



WDT zamanlayıcısıyla kontrolden çıkmış bir programı tekrar resetleyerek kontrolü ele almayı planladıysanız ana program başında bu bit'leri kontrol etmelisiniz. Reset gerçekleştikten hemen sonra da bu bit'leri kontrol etmek gerekir. Böylece WDT'nin zaman aşımı süresi dolduğunda program akışı başka bir yere dalandırabilirsiniz. Örnek program parçası:

```
WDT_KONT
    BTFSS STATUS, 4 ; T0 bit'i 1 mi?
    CALL AAA        ; Evet, AAA alt programını çağır.
    MOVLW h'FF'     ; Hayır, program akışına devam et.
```



```

MOVWF TRISA    ; PORTA'nın tüm uçları giriş
.
.
.
.

```

PIC içerisinde bulunan WDT zamanlayıcısının geçerli (enable) veya geçersiz (disable) yapılması yazılım aracılığı ile yapılmaz. PIC programlamaya yeni başladığınızda PIC'e yazma programı anlatılırken konfigürasyon bit'lerinden bahsetmiştik. Bu konfigürasyon bit'leri program içerisinde yazıldığı gibi, PIC programlayıcı programındaki "Fuses" penceresi içerisinde de enable yapılacağını söylemiştik. Bu nedenle WDT zamanlayıcıyı kullanabilmek için PIC'i programlama esnasında "Fuses" penceresinden Watchdog Timer="ON" yapılmalıdır.

5. bölümde konfigürasyon bit'lerini program içerisinde belirlenebileceğini söylemiştik. Hatırlatmak amacıyla bir örnek verecek olursak, VVDT'nin aktif yapıldığı konfigürasyon satırı aşağıdaki gibi olmalıdır.

```
_CONFIG _CP_OFF & _WDT_OFF & PWRT_ON & _RC_OSC
```

WDT zamanlayıcısını programın çalışması esnasında geçersiz (disable) /apmak mümkün değildir. Sadece CLRWDWT komutu kullanılarak RESET yapılabilir. Bu durumda WDT, h'00'dan itibaren tekrar saymaya başlar. Sayma süresi bitince STATUS registerindeki T0 bayrağını "0" yapar.

SLEEP Komutunun Kullanılması

SLEEP komutu genellikle PIC'in içerisindeki programın devamlı olarak çalışması gerekmeyen uygulamalarda kullanılır. Bu komut çalışınca program olduğu yerde durur. Tüm özel registerler (OPTION, STATUS, INTCON vs.) ve RAM içerisindeki bilgiler saklı tutulur. PIC uyuma (Sleep) moduna girmiş olur. Bu esnada PIC16F84'ün çektiği akım 40 µA e kadar düşer.

SLEEP komutu kullanılarak, program içerisinde kullanılan ve herhangi bir çevresel olayı test eden software döngülerini kullanıma gereksinimini ortadan kaldırır. Böylece daha sade program yazma olanağını yaratır. Örneğin bir hırsız alarm sistemini kontrol eden programda, programın devamlı olarak çalışması gerekmez. Belirli aralıklarla çevredeki olayları denetlemesi gereken bu sistemdeki PIC'i uyuma modundan WDT uyandırır. Program çalışarak giriş portlarından alarmin çalışmasını gerektirmeyen bir giriş yoksa, tekrar SLEEP komutunu çalıştırarak PIC'i uyuma moduna sokar.

SLEEP modundan çıkış sadece WDT'nin zaman aşımı sinyali ile değil, harici kesme sinyalleri vasıtasıyla da yapılabilir.

PROGRAM-29) PortB'ye bağlı 8 LED üzerinde binary artan sayıları gösteren program. WDT'ye atanan prescaler değerine göre, LED'ler saymaya devam ederken portB içerisindeki veri h'FF'e ulaşmadan WDT zaman aşımı sinyali nedeniyle LED'lerin yanışı 0'dan itibaren tekrar başlatılır.

```

;====PROG29.ASM====10/08/2000=====
LIST P=16F84
INCLÜDE "P16F84.INC"
SAYAC1 EQU  h'0C'
SAYAC2 EQU  h'0D'
BASLA
    BSF  STATUS, 5          ; BANK1
    MOVLW b'00001110'      ; WDT seçilir.
    MOVWF OPTION_REG       ; OPTION_REQ←W
    CLRF  TRISB             ; PortB'nin uçları çıkış
    BCF  STATUS, 5          ; BANK0
SONDUR
    CLRF  PORTB             ; PortB'yi sil
    YAK
    CALL  GECIKME           ; GECIKME alt programı
    INCF  PORTB, 1          ; PORTB←PORTB + 1
    GOTO  YAK
GECIKME
    MOVLW h'4F'             ; W←h'4F'
    MOVWF SAYAC1            ; SAYAC1←W
DONGU1
    CLRF  SAYAC2            ; SAYAC2←0
DONGU2
    DECFSZ SAYAC2, 1
    GOTO  DONGU2
    DECFSZ SAYAC1, 1
    GOTO  DONGU1
    RETURN

```

END

İşlem Basamakları

- 1 PIC16F84'e PROG29.ASM programını yazınız.
- 2 Programı yazdırırken kullandığımız PIC programlayıcıdaki "Fuses" penceresinde WDT=ON yaptıktan sonra yazdırma işlemini başlatınız.
- 3 PIC'i deneme kartına yerleştirerek programınızı çalıştırınız. PortB'deki LED'lerde binary artan sayıları görünüz. Sayma işlemi devam ederken WDT zaman aşımı sinyali programı reset ederek tekrar O'dan itibaren saymaya başlatacaktır.
- 4 LED'lerdeki sayı kaçta ulaştığında reset gerçekleşeceğini şu anda söylememiz mümkün değildir. Çünkü kullandığınız RC osilatörün frekansına ve WDT oranına bağlıdır. WDT oranını değiştirerek programın reset olma süresini değiştiriniz. PIC'i yeniden programlayarak deneme kartınızda deneyiniz.

14

D/A ve A/D ÇEVİRME İŞLEMLERİ

- ❑ DİJİTAL/ANALOG ÇEVİRİCİ
- ❑ ANALOG/DİJİTAL ÇEVİRİCİ

DİJİTAL/ANALOG ÇEVİRİCİ

PIC16F84 kullandığınız bir uygulama devresinde analog voltaj çıkış gerekebilir. Bu durumda portlardan çıkan dijital sinyalleri analog voltaja çevirme işlemi söz konusu olmaktadır. Bu işlemi yapmak için bir çok yöntem vardır Bunlardan ilki, PIC çıkışına ladder (merdiven) direnç devresi ve bir OP-AMF bağlayarak yapılan D/A çeviricidir. ikincisi ise, çok daha basit bir yöntem olan PIC çıkışına D/A çevirici entegre bağlamaktır.

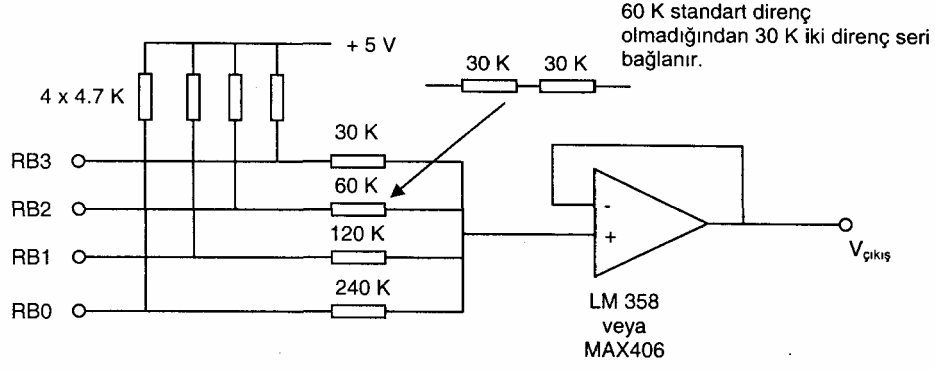
Microchip firması PIC kullanıcılarının bu ihtiyacını karşılamak amacıyla piyasaya sürdüğü bir çok mikrodnetleyisi bulunmaktadır.

Örneğin PIC14C000 mikrodnetleyicisi girişine uygulanan dijital sinyali doğrudan analog voltaj çıkış olarak verebilmektedir. D/A çevirici entegre bu PIC'in içerisine yerleştirilmiş olduğundan çıkışında başka bir devre kullanmaya gerek kalmamaktadır. Bu tip PIC'lerin incelenmesi ayrı bir kitap konusu olduğunda şimdilik ayrıntılara girmeyeceğiz.

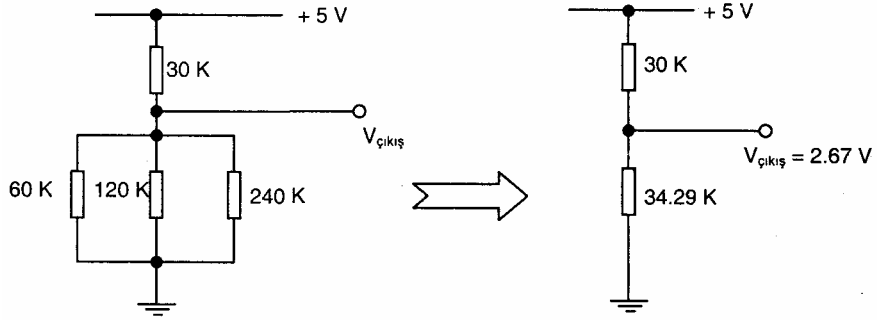
Bu kitapta bizim ele aldığımız PIC16F84'ün sadece dijital girişi ve çıkışı bulunmaktadır. D/A çevrimi yapabilmek için yapabileceğimiz tek şey, PIC çıkışından aldığımız dijital sinyali analog gerilime çevirme işlemidir. Şimdi D/A çevrimi işlemi için kullanmak zorunda olduğumuz yöntemleri açıklayalım.

Ladder(Merdiven) Direnç Devresi Kullanmak

Örnek D/A çevirici olarak vereceğimiz devrede PIC'ten 4 bit'lik dijital ve çıktığını düşünürsek, aşağıdaki ladder direnç devresi ile bu sinyali Analog gerilim çeviren devreyi aşağıdaki gibi çizebiliriz.



PortB'den çıkan sinyalin b'00001000' (RB3=1) olduğunu düşünersek, bu devrede $V_{\text{çıkış}}$ ucundan alınacak gerilimin değerini bulalım:

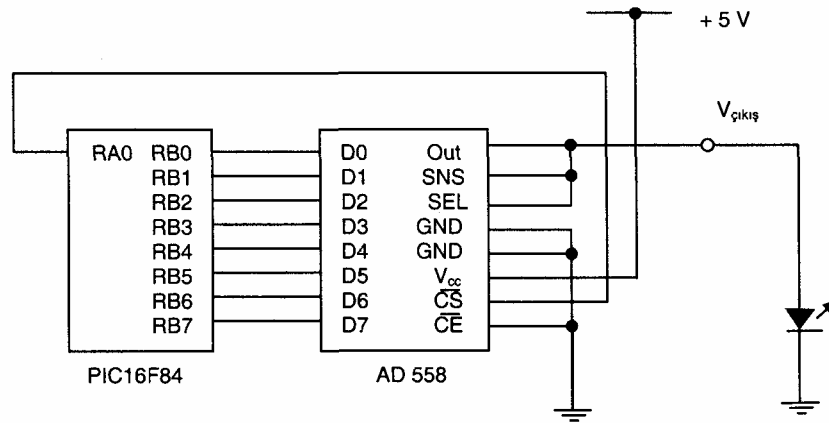


0-5 V arasında analog gerilim elde etmek için gerekli olan dijital çıkışlar aşağıdaki gibidir.

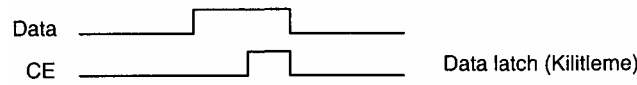
RB3	RB2	RB1	RB0	$V_{\text{çıkış}}$
0	0	0	0	0.00
0	0	0	1	0.33
0	0	1	0	0.67
0	0	1	1	1.00
0	1	0	0	1.33
0	1	0	1	1.67
0	1	1	0	2.00
0	1	1	1	2.33
1	0	0	0	2.67
1	0	0	1	3.00
1	0	1	0	3.33
1	0	1	1	3.67
1	1	0	0	4.00
1	1	0	1	4.33
1	1	1	0	4.67
1	1	1	1	5.00

8 Bit D/A konvertör Entegresi Kullanmak

PIC'ten çıkan dijital veriyi analog gerilime dönüştürebilmek için uygulanabilecek ikinci yöntem bir D/A entegresi kullanmaktır. Biz P/N AD558 entegresini örnek olarak vereceğiz. Bu entegre içerisinde yukarıdaki örnekte verdiğimiz ladder dirençler ve toplayıcı amplifikatör (OP-AMP) hazır olarak bulunmaktadır. Entegre 5 V DC ile çalışmakta ve girişine uygulanan dijital sinyali 0-2.55 V arasında gerilim vermektedir.



AD 558 entegresindeki CE ve CS uçlarının fonksiyonları aynıdır. Birisini (CE) toprağa bağlayın. CS giriş verisinin yükselen kenarında girilen datayı kilitlemek için kullanılan uçtur. Bu nedenle portA'nın 0. bit'inden uygulanacak sinyalin yükselen kenarında kilitleme yapılması gerekir.



PROGRAM-30) PortB'nin RBO-RB7 uçlarından çıkan dijital veriyi analog veriye çeviren program. Bu program yukarıda AD 558 entegresiyle kurulan devreye göre yapılmıştır. PortB'nin ucundan çıkması istenen veri b'10000000'dir. Bu veri AD 558 entegresiyle 1.27 V'luk gerilime çevrilir.

;==PROG30.ASM==12/08/2000=====

LIST P=16P84

INCLUDE "P16F84.INC"

BASLA

```
BSF STATUS, 5 ;Bank1
CLRF TRISA ;PORTA tüm uçları çıkış
CLRF TRISB ;PORTB tüm uçları çıkış
BCF STATUS, 5 ;Bank0
CLRF PORTA ;CS ucunu "0" yap
MOVLW b'10000000';W←b'10000000'
MOVWF PORTB ;AD 558 entegresine giden veri
BSF PORTA, 0 ;AD 558'in CS ucu yükselen kenar
BCF PORTA, 0 ;AD 558'in CS ucu düşen kenar
DONGU
GOTO DONGU
```

AD 558 entegresine girilen verilerin kaç V'luk gerilime dönüştürüleceğine ait birkaç örnek veren tablo aşağıdadır.

Haksadesimal sayı	V _{ÇIKIŞ}
h'00'	0.00
h'0F'	0.15
h'B0'	1,27
h'FF'	2.55

İşlem Basamakları

1. PROGSO.ASM'yi PIC16F84'e yazdırınız.
2. AD 558 entegresiyle yapılan A/D çevirici PIC devresini breadboard üzerine kurunuz.
3. PIC'i deneme devrenize yerleştirerek gerilim uygulayınız. V_{çıkış} ucuna bir voltmetre bağlayarak 1.27 V'luk gerilimi ölçünüz.

PROGRAM-31) AD 558 konvertörü kullanarak kurulan devrede testere dişi dalga üreten program.

;==PROG31.ASM==14/08/2000=====

LIST P=16P84

INCLUDE "P16F84.INC"

VOLT EQU h'0C'

SAYAC EQU h'0D'

BASLA

```
BSF STATUS, 5 ; Bank1
CLRF TRISA ; PORTA tüm uçları çıkış
CLRF TRISB ; PORTB tüm uçları çıkış
```

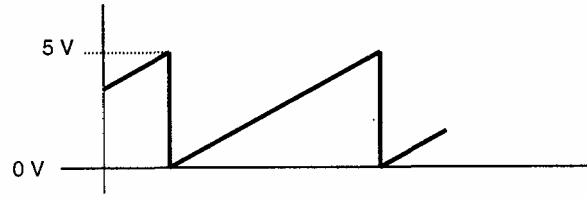
```

BCF STATUS, 5 ; Bank0
CLRF PORTA ; CS ucunu "0" yap
CLRF VOLT ; VOLT deęişkenini sıfırla
DONGU
MOVF VOLT, W ; W←VOLT (h'00')
MOVWF PORTB ; PORTB←W
BSF PORTA, 0 ; AD558'deki datayı kilitle
BCF PORTA, 0 ;
INCF VOLT, F ; VOLT←VOLT +1
GOTO DONGU

```

İşlem Basamakları

1. PIC16F84'ü programlayarak AD558 ile yaptığınız devre üzerine yerleştiriniz.
2. $V_{\text{çıkış}}$ ucuna bir osilaskop bağlayınız ve ayarlarını yapınız.
3. $V_{\text{çıkış}}$ aşağıdaki testere dişi dalgayı görünüz.



PROGRAM-32) AD558 entegresiyle kurulan devrede sinüs dalgası üreten program. Bu programda portB'ye gönderilen binary sayıları çevrim tablosu içerisine yerleştirilmiştir. 19 voltaj seviyesi daha da arttırılarak sinüs dalgasında birbirine daha yakın gerilimler elde edilebilir. Bu durumda çıkış eğrisi titreşim olmayacaktır. Sinüs dalgasının frekansı, çevrim tablosuna yerleştirilen sayıların çokluğu ile düşürüleceği gibi, çıkış verileri arasında gecikme alt programı kullanılarak da ayarlanabilir.

```

;====PROG32.ASM====16/08/2000=====
LIST P=16F84
INCLUDE "P16F84.INC"
SAYAC EQU h'0D'
OFFSET_SAY EQU h'0C'
BASLA
    BSF STATUS, 5 ;Bank1
    CLRF TRISA ;PORTA tüm uçları çıkış
    CLRF TRISB ;PORTB tüm uçları çıkış
    BCF STATUS, 5 ;Bank0
    CLRF PORTA ;CS ucunu "0" yap
SAYKIL
    CLRF OFFSET_SAY ;Tablo ofset sayacını sıfırla
ADIM
    MOVF OFFSET_SAY, W ;W←Ofset sayacını yaz
    CALL TABLO
    MOVWF PORTB ;Veriyi portB'ye çık
    BSF PORTA, 0 ;AD558'e kilitleme sinyali
    BCF PORTB, 0
    CALL GECIKME
    MOVF OFFSET_SAY, W ;Ofset sayacını W'ye yaz
    SUBLW h'23' ; Saykıl bitti mi?
    BTFSC STATUS, 2 ; Zero flag'ı kontrol et.
    GOTO SAYKIL
    INCF OFFSET_SAY ; OFFSET_SAY←OFFSET + 1
    GOTO ADIM
GECIKME
    MOVLW h'01' ; 3N+3 saykıl(desimal), N=1
    MOVWF SAYAC ; W←SAYAC
TEKRAR
    DECFSZ SAYAC, F ; SAYAC←SAYAC - 1
    GOTO TEKRAR ; SAYAC 0 değil
    RETURN
TABLO
    ADDWF PCL, F ; OFFSET_SAY'ı PLC'ye yükle

```

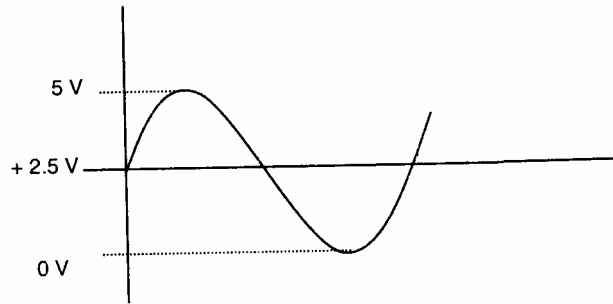
```

RETLW h'80'      ; 1.28 Volt
RETLW h'95'      ; 1.50
RETLW h'AD'      ; 1.72
RETLW h'Cl'      ; 1.92
RETLW h'D3'      ; 2.10
RETLW h'E2'      ; 2.26
RETLW h'EF'      ; 2.39
RETLW h'F8'      ; 2.48
RETLW h'FD'      ; 2.54
RETLW h'FF'      ; 2.55
RETLW h'FD'      ; 2.54
RETLW h'F8'      ; 2.48
RETLW h'EF'      ; 2.39
RETLW h'E2'      ; 2.26
RETLW h'D3'      ; 2.10
RETLW h'Cl'      ; 1.92
RETLW h'AD'      ; 1.72
RETLW h'95'      ; 1.50
RETLW h'80'      ; 1.28
RETLW h'6A'      ; 1.06
RETLW h'54'      ; 0.84
RETLW h'40'      ; 0.64
RETLW h'2E'      ; 0.46
RETLW h'1E'      ; 0.30
RETLW h'11'      ; 0.17
RETLW h'08"      ; 0.08
RETLW h'02'      ; 0.02
RETLW h'00'      ; 0.00
RETLW h'02'      ; 0.02
RETLW h'08'      ; 0.08
RETLW h'11'      ; 0.17
RETLW h'1E'      ; 0.30
RETLW h'2E'      ; 0.46
RETLW h'40'      ; 0.64
RETLW h'54"      ; 0.84
RETLW h'6A'      ; 1.06
END

```

İşlem Basamakları

1. PIC16F84'ü programlayarak AD558 ile yaptığınız devre üzerine yerleştiriniz.
2. $V_{\text{çıkış}}$ ucuna bir osilaskop bağlayınız ve ayarlarını yapınız.
3. $V_{\text{çıkış}}$ ucunda aşağıdaki sinüs dalgasını görünüz.



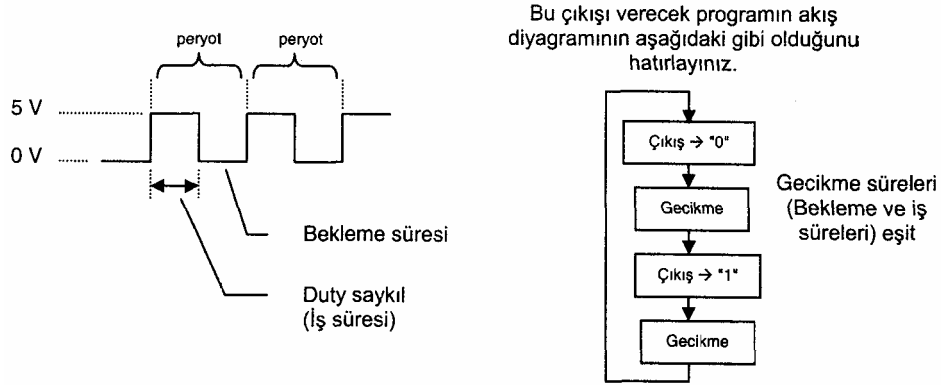
PWM (Pulse Width Modulation) Metodu Kullanmak

PWM (Pulse-Width-Modulation → "Puls genişliğini ayarlama") metodu PIC16F84'ün çıkış uçlarından birisinden kare dalga sinyali üretme işlemidir. Kare dalganın puls genişliğini ayarlama suretiyle elde edilen gerilimin DC voltmetre ile ölçülen değerini (Ortalama değeri) değiştirilmiş olur.

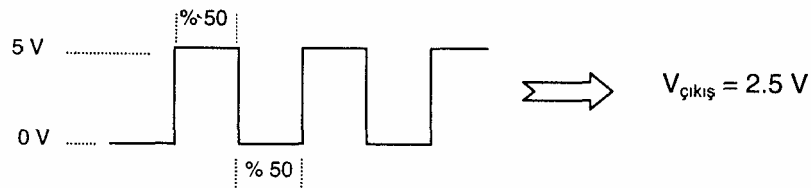
Bundan önceki bölümlerde yaptığımız çoğu programlarda portB'nin sadece bir bit'ine bağlı LED'i flash (Yakıp-söndürme) yapmıştık. Eğer LED'i çıkarıp yerine bir voltmetre bağlayacak olsaydık, 2.5 V civarında bir gerilim okuyacaktık. Şimdi PIC programı aracılığı ile bu gerilimin nasıl değiştirileceği hakkında temel elektronik bilgisini verdiğimizde yapılacak programın hiç de zor olmadığını göreceksiniz.

PWM Metodu

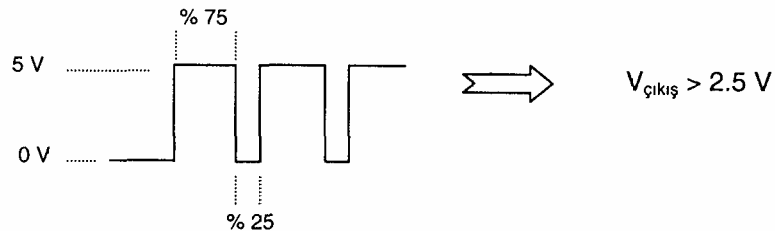
PWM metodunu anlayabilmek için eşit aralıklarla LED yakıp söndürme programlarında elde ettiğimiz çıkış eğrisini inceleyelim.



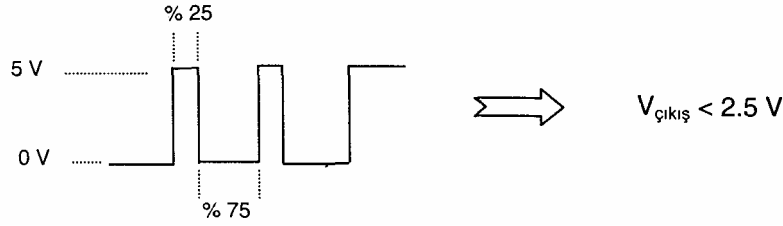
Yukarıdaki çıkış eğrisinde ış süresi, LED'in yanış süresi, bekleme süresi de sönmüş kalma süresidir. Eğer bir periyot içerisindeki bu süreler birbirine eşitse çıkış gerilimin ortalama değeri 2.5 V olur.



Bir periyot içerisindeki ış süresi ve bekleme süresi oranları değiştirilirse çıkışta ölçülecek gerilimin değeri de değiştirilmiş olur.



Görüldüğü gibi ış süresi uzatıldığında çıkış geriliminin ortalama değeri 2.5 V tan büyük olmaktadır, Eğer ış süresi % 100 olursa çıkış gerilimi 5 V, ış süresi % 0 olursa çıkış gerilimi 0 V olur. Bu defa ış süresinin bekleme süresinden az olduğu duruma örnek verelim.



Çıkış geriliminin ortalama değeri bir periyot içerisindeki iş süresi ve bekleme süresi oranlarıyla doğru orantılıdır.

Örneğin;

İş süresi % 50 ise çıkış gerilimi (5 V) → 2.5 V

İş süresi % 25 ise çıkış gerilimi (5 V) → 1.25 V

İş süresi % 75 ise çıkış gerilimi (5 V) → 3.75 V olur.

İş ve Bekleme Süresinin Tespit Etmek

RC osilatörle beslenen PIC16F84'ün RB0 ucundan 0 -5 V arasında değişen analog gerilim elde etmek istediğimizi düşünelim. Önce çıkış periyodunun iş süresini ve bekleme süresini belirleyen GECIKME alt programının nasıl olması gerektiğine karar verelim.

GECIKME

MOVWF SAYAC

DONGU

DECFSZ SAYAC, F

GOTO DONGU

RETURN

SAYAC registeri içerisindeki ilk sayı h'FF' olursa, her defasında SAYAC'tan "1" çıkarılır. SAYAC'ın "0" a gelmesi için 256 döngü gerekir. Bu GECIKME alt programıyla çıkış periyodunun % 100'ünü elde ettiğimizi düşünersek, iş süresi ve bekleme süresini tespit etmek için h'FF' /Desimal 256) sayısının yüzdesini almak gerekir. Örneğin;

2.5 V'luk çıkış için % 50 iş süresi % 50 bekleme süresi gerekir.

Bu durumda SAYAC registeri içerisine atayacağımız sayılar ne olmalıdır ?

$$256 \times (\% 50) = 128 \rightarrow \text{h}'80' \text{ (İş süresi için)}$$

$$256 \times (\% 50) = 128 \rightarrow \text{h}'80' \text{ (bekleme süresi için)}$$

örneğin;

1.25 V'luk çıkış için %25 iş süresi % 75 bekleme süresi

$$256 \times (\% 25) = 64 \rightarrow \text{h}'40' \text{ (İş süresi için)}$$

$$256 \times (\% 75) = 192 \rightarrow \text{h}'C0' \text{ (Bekleme süresi için)}$$

örneğin;

0.5 V'luk çıkış için % 10 iş süresi % 90 bekleme süresi

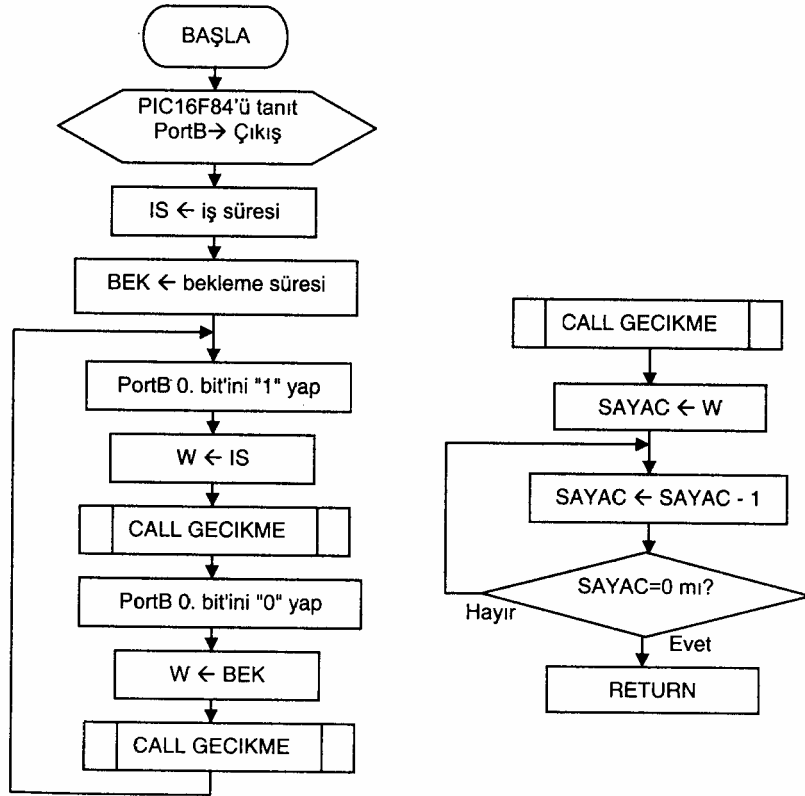
$$256 \times (\% 10) = 25.6 \rightarrow 26 \rightarrow \text{h}'1A' \text{ (iş süresi için)}$$

$$256 \times (\% 90) = 230.4 \rightarrow 230 \rightarrow \text{h}'E6' \text{ (bekleme süresi)}$$

PROBLEM-33) PortA'nın 0. bit'ine bağlı bir voltmetrede 2.5 V'luk gerilim üreten PIC programı. Çıkış geriliminin analog değeri (2.5 V) portB'deki dijital sinyalin (5 V) % 50 si olacağından,

IS değişkenine → h'80'

BEK değişkenine → h'80 sayıları atanmıştır.



;;=PROG33.ASM=====21/08/2000=====

```

LIST P=16F84
INCLUDE "P16F84.INC"
IS EQU h'0D'
BEK EQU h'0C'
SAYAC EQU h'0E'
ORG h'00'

BASLA
    BSF STATUS, 5 ; Bank1
    CLRF TRISB ; PORTB tüm uçları çıkış
    BCF STATUS, 5 ; Bank0
    CLRF PORTB ; CS ucunu "0" yap
    MOVLW h'80' ; W←h'80'
    MOVWF IS ; IS←W, İş süresini ata
    MOVLW h'80' ; W←h'80'
    MOVWF BEK ; BEK ←W, bekleme süresini ata

TEKRAR
    BSF PORTB, 0 ; PORTB'nin 0. bit'i "1"= 5V
    MOVF IS, W ; W←IS, iş süresini al
    CALL GECIKME
    BCF PORTB, 0 ; PORTB'nin 0. bit'i "0"= 0 V
    MOVF BEK, W ; W ←BEK, bekleme süresini al
    CALL GECIKME
    GOTO TEKRAR

GECIKME
    MOVWF SAYAC ; IS ya da BEK içerisindekini SAYAC'a yaz.

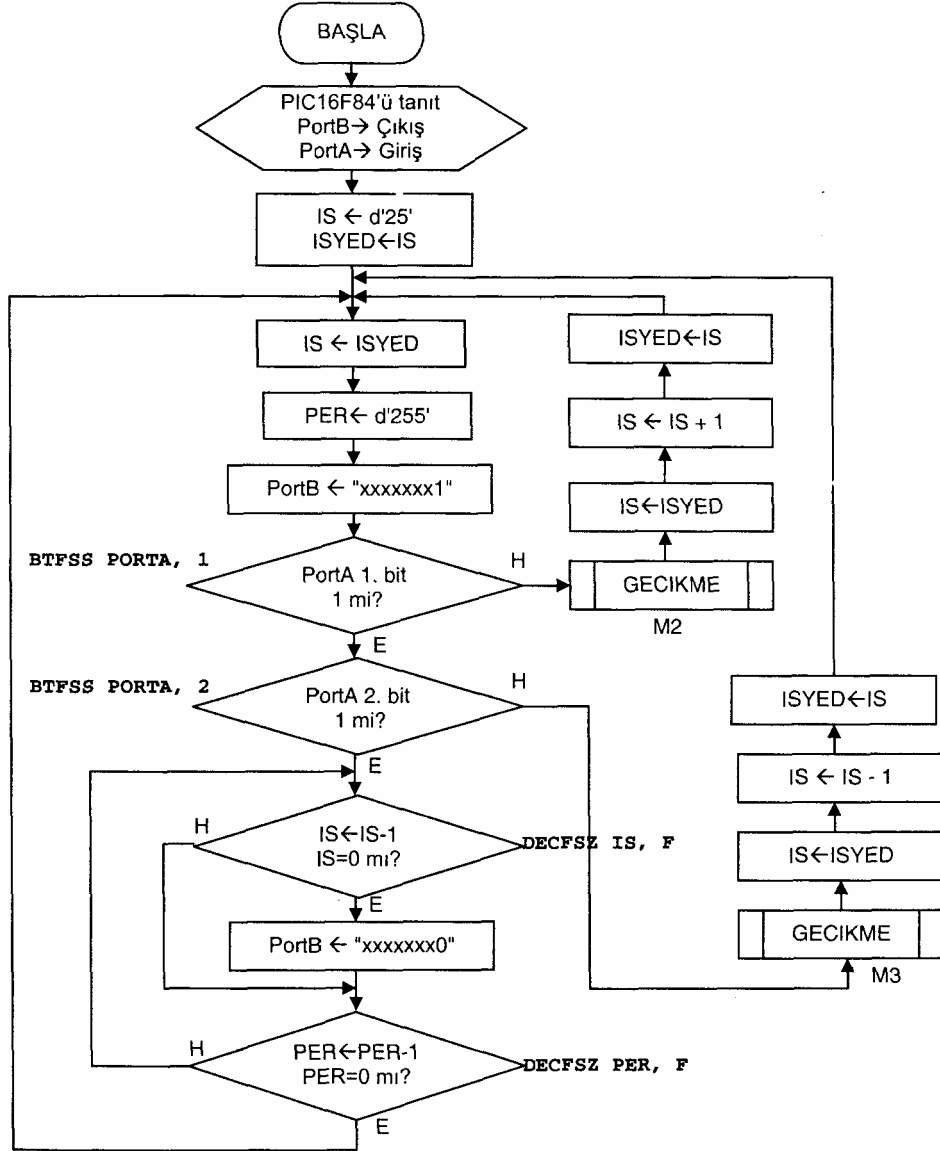
DONGU
    DECFSZ SAYAC, F
    GOTO DONGU
    RETURN
    END
  
```

İşlem Basamakları

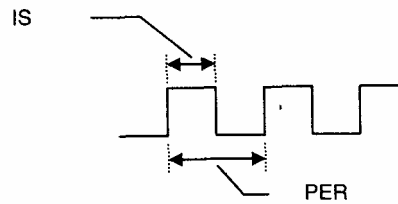
1. PIC16F84'ü, PROG33.ASM ile programlayınız.
2. PIC'i deneme kartına yerleştiriniz.

- Devreye gerilim uygulayarak programı çalıştınız. PortB'nin 0. bit'ine bağlı LED'in yarı parlaklıkta yandığını görünüz. (Çıkış 2.5 V olduğundan)
- Breadboard üzerinde kurduğunuz devre üzerinde deneme yapıyorsanız, PortB'nin RBO ucuna bir voltmetre bağlayarak 2.5 V luk gerilimi ölçünüz.

PROGRAM-34) PortA'nın 1. bit'indeki A1 butonuna basınca portB'nin 0.bit'indeki LED'in parlaklığını arttıran, A2 butonuna basınca parlaklığını azaltan program. (PWM metodu ile D/A çeviriciye ikinci bir örnek programdır.)



Bu programda 33. problemdekinden farklı bir yöntem kullanılarak iş ve bekleme süreleri tespit edilmiştir. A1 butonuna basılınca IS değişkeninin içerisindeki sayı arttırılır. IS süresince portB'nin 0. bit'i "1" olarak kalır. IS süresi bitince geriye kalan PER süresince de portB'deki 0. bit "0" olarak kalır. A2 butonu IS içerisindeki sayıyı azaltır. IS süresi büyüdükçe LED parlaklaşır, küçüldükçe donuklaşır. PER süresi sabit olarak "250" alınmıştır. İstenirse bu süre değiştirilebilir. PER süresi uzadıkça portB'den çıkan sinyalin frekansı düşer, kısaldığında ise artar. IS süresi PER süresinden büyük olmamalıdır. Aksi halde LED devamlı olarak yanık kalır.



```

;====PROG34.ASM====25/08/2000=====
LIST      P=16F84
INCLUDE "P16F84.INC"
IS        EQU    h'0D"    ;İş süresi değişkeni
PER        EQU h'0C'      ;Peryod süresi değişkeni
ISYED EQU    h'0E'      ;IS'i yedekleme değişkeni
SAYAC1    EQU h'0F"
SAYAC2 EQU h'10'
          CLRFB PORTB      ;PortB'yi sil
          BSF STATUS, 5 ;BANK1'e geç
          CLRFB TRISB      ;PortB tüm uçları çıkış
          MOVLW h'FF'
          MOVWF TRISA      ;PortA'nın tüm uçları giriş
          BCF STATUS, 5 ;BANK0'a geç

BASLA
          MOVLW d'25'      ;W←d'10'
          MOVWF IS          ;IS 8 W
          MOVWF ISYED;IS' i yedekle

DONGU
          MOVF ISYED, W ;W←ISYED
          MOVWF IS          ;IS'i yedekten al
          MOVLW d'250'      ;W←d'100'
          MOVWF PER          ;PER←W
          BSF PORTB, 0 ;PORTB ←'xxxxxxx1'
          BTFSS PORTA, 1 ;PORTA, 1. bit 1 mi?
          GOTO M2          ;Hayır, M2'ye git
          BTFSS PORTA, 2 ;Evet, PORTA. 2. bit 1 mi?
          GOTO M3          ;Hayır, M3'e git.

PWM0
          DECFSZ IS, F ;IS=IS-1, IS=0 mı?
          GOTO PWM1      ;Hayır, PWM1'e git
          BCF PORTB, 0 ;Evet, PORTB←'xxxxxxx0'

PWM1
          DECFSZ PER, F ;PER=PER-1, PER=0 mı?
          GOTO PWM0      ;Hayır, PWM0'a git
          GOTO DONGU      ;Evet, DONGU'ye git.

M2
          CALL GECIKME ;Buton arkı sönmesi için bekle
          MOVF ISYED, W
          MOVWF IS          ;IS'yi yedekten al
          IMCF IS, F ;IS=IS+1
          MOVF IS, W
          MOVWF ISYED ;IS'yi yedekle
          GOTO DONGU

M3
          CALL GECIKME ;Buton arkı sönmesi için bekle
          MOVF ISYED, W
          MOVWF IS          ;IS'yi yedekten al
          DECF IS, F ;IS=IS-1
          MOVF IS, W
          MOVWF ISYED ;IS'yi yedekle
          GOTO DONGU

GECIKME ;Gecikme alt programı
          MOVLW h'0F'
          MOVWF SAYAC1

D1
          MOVLW h'FF'
          MOVWF SAYAC2

D2
          DECFSZ SAYAC2, F
          GOTO D2
          DECFSZ SAYAC1, F
          GOTO D1
          RETURN
          END

```

İşlem Basamakları

1. PIC'i programlayarak deneme kartınız üzerine yerleştiriniz.
2. Deneme kartına enerji verince portB'nin 0. bit'indeki LED donuk bir şekilde yanacaktır. Çünkü $IS=d'25'$, $PER=d'250'$ sayıları atanmıştır. Bu durumda LED, bir periyot süresinin 1/10 inde "1", 9/10 unda "0" olarak kalacaktır.
3. A1 butonuna birkaç defa basarak IS içerisindeki sayıyı arttırınız. LED'in parlaklığı gittikçe artacaktır.
4. A1 butonuna basmaya devam ettikçe LED en parlak duruma geldikten sonra tekrar sönecektir, Bu, IS ve PER değişkenlerinin içerisindeki sayıların eşit duruma geldiğini gösterir.
5. A2 butonuna basarak bu defa LED'in parlaklığını azaltınız.
6. PortB'nin 0. bit'ine bir dijital voltmetre bağlayarak bu uçtaki gerilimin 0-5V arasında değiştiğini görünüz.

ANALOG / DİJİTAL ÇEVİRİCİ

PIC16F84'ün sadece dijital verileri işlediğini biliyoruz. Bu nedenle analog bir verinin dijital veriye çevrilmesi için yapılacak programlar çok önemlidir.

PIC'e analog veri girmek, giriş uçlarından birisinden gerilim uygulamak demektir. PIC, içerisinde bulunan program vasıtasıyla bunu çıkış uçlarında binary veriye çevirirse bu da dijital çıkış demektir. Bu işlemi yapan piyasada microchip'in ürettiği bir çok mikrodenetleyici bulunmaktadır. Örneğin, PIC16C7X serisi PIC'lerin analog giriş için özel uçları ve içlerinde A/D çeviriciler bulunmaktadır. Son zamanlarda çok popüler olan ve "Flash" belleğe sahip olan PIC16F877 mikrodenetleyicisi de yine bu özelliklere sahiptir.

Analog sinyal genellikle bir sensör (Isı sensörü, ışık sensörü, manyetik sensör vb.) vasıtasıyla sağlanabilir. Biz bu kitapta vereceğimiz örneklerde kullanacağımız PIC devrelerinde bu iş için voltaj bölücü potansiyometre kullanacağız.

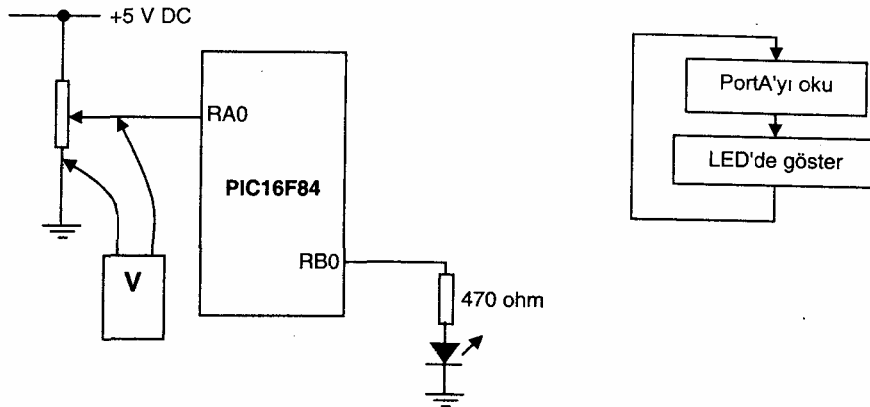
Bir PIC içerisine analog veri girmek için yapılacak en kolay yöntem bir A/D konvertör kullanmaktır. Böylece analog gerilim konvertör entegresinde dijital veriye dönüştürülür ve PIC girişlerine uygulanır. Bu kolay metodu elektronik bilgisine sahip olanlar çok kolaylıkla uygulayabileceklerdir. Biz bu yöntemi kitapta vermeyeceğiz.

Şimdi ilk olarak PIC16F84'ün girişine uygulanan bir gerilim seviyesini nasıl algıladığını inceleyelim.

PIC16F84'ün Giriş Seviyesinin Ölçümü

PIC16 mikrodenetleyicilerinin I/O ucuna uygulanan threshold gerilimi vardır Bu gerilimin altındaki gerilimler "0"(low), üstündeki gerilimler ise "1"(High) olarak algılanır, Bu gerilim bir potansiyometre ve dijital voltmetre kullanılarak ölçülebilir Girişin nasıl ("1" veya "0") algılandığını görmek için de LED kullanılır. Yapılacak işlemi bir örnek verelim.

PROGRAM-35) PIC16F84'ün giriş seviyesinin ölçülmesine örnek program. Bu programı deneyebilmek için aşağıdaki devreyi breadboard üzerine kurmanız gerekmektedir.



```

;===PROG35.ASM===27/08/2000=====
LIST P=16F84
INCLUDE "P16F84.INC"
BASLA
    BSF STATUS, 5
    CLRF TRISB
    MOVLW h'0'
    MOVWF TRISA
    BCF STATUS, 5
    CLRF PORTB
DONGU
    BTFSS PORTA, 0 ;PortA, 0. bit 1 mi?
    GOTO SONDUR ;Hayır, LED'i söndürmeye git.
    BSF PORTB, 0 ;Evet, LED'i yak.
    GOTO DONGU
SONDUR
    BCF PORTB, 0 ;LED'i söndür.
    GOTO DONGU
END

```

İşlem Basamakları

1. PIC16F84'ü programlayınız.
2. Bu programı denemek için yukarıdaki devreyi breadboard üzerine kurup gerilim uygulayınız.
3. İlk olarak, potansiyometreyi ayarlayarak RAO ucuna uygulanan gerilimi 1V un altına getiriniz.
4. Potansiyometreyi ayarlayarak yavaş yavaş gerilimi yükseltiniz. Bir an LED yanmaya başlayacaktır. Bu gerilime V_{lash} gerimi diyelim. V_{flash} gerilimi PIC'ten PIC e değişebilir. Yaklaşık olarak ölçülmesi gereken gerilim 15 V civarında olacaktır.

A/D çevrim Metodu Kullanarak Direnç Ölçmek

Yukarıdaki deneyden edindiğimiz bilgileri kullanarak PIC16F84'ün ucuna bağlı bir direncin değerini ölçebiliriz. Ölçülen değer portB'ye bağlı LED'lerde binary olarak okunabilir.

Bu yöntem, değeri bilinmeyen bir direncin, bir kondansatörü V_{flash} değerine kadar ne kadar sürede şarj ettiğini ölçme esasına dayanır. Ölçme neticesinin doğru olup olmadığını anlayabilmek için, değeri bilinen bir direnç veya potansiyometre kullanmakta yarar vardır. TMR0 sayıcısı şarj süresini ölçmek için kullanılır.

Ölçme işlemi şöyle yapılır:

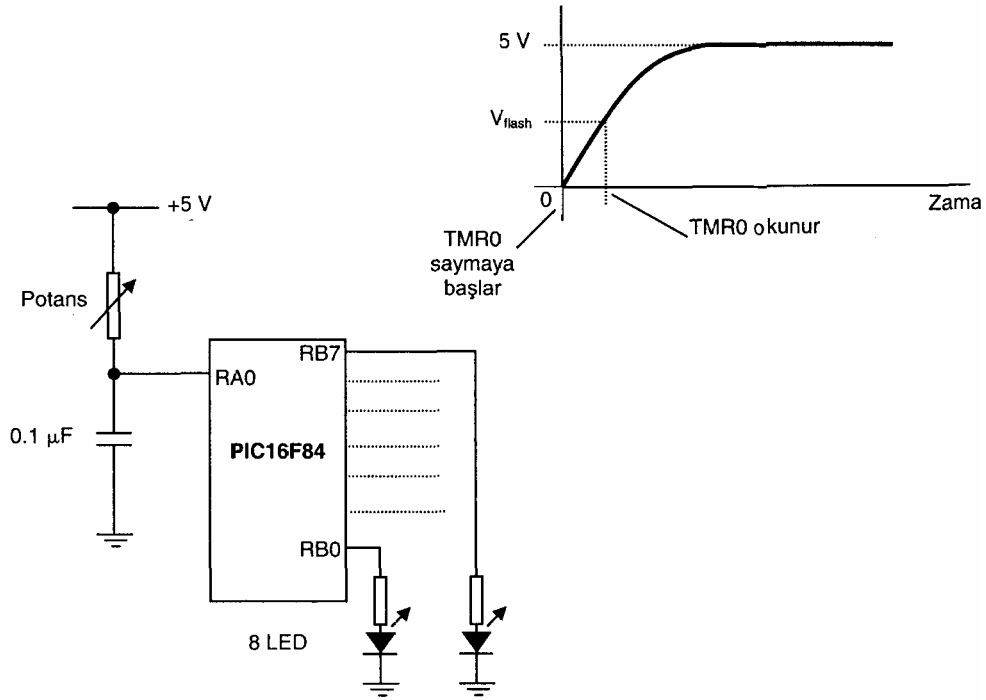
Kondansatör deşarj edilir. Bu işlem portA'yı çıkış olarak yönlendirdikten sonra RA0 ucuna "0" gönderilerek yapılır.

Kondansatörün deşarjı belirli bir süre alır. Bu süreden sonra RA0 UCL giriş olarak yönlendirilir ve bu anda TMR0 çalıştırılır.

Program RA0 portunu izler, bu uçtaki gerilim V_{flash} gerilimine ulaştığında TMR0 sayıcı registeri okunur.

Şarj süresi direncin değeriyle ters orantılıdır. Direnç büyüdükçe şarj süresi de büyüyeceğinden TMR0 registeri içerisinde daha büyük bir sayı okunur.

Yukarıdaki yöntemle direncin gerçek değerini ölçmek için ilk olarak değeri bilinen bir direncin ne kadar sürede TMR0 sayıcısını belirli bir sayıya ulaştırdığı bulunmalıdır. Daha sonra bu direnç değerinin TMR0'da göstermesi gereken değer prescaler değeriyle ayarlanarak belirli bir sayı göstermesi sağlanır. Ölçülecek daha sonraki dirençler referans alınan ilk dirençle kıyaslanarak gerçek değeri bulunur.



PROGRAM-36) RAO ucuna bağlı bir potansiyometrenin direnci değiştirildikçe portB'deki yanan LED'leri binary olarak saydıran program.

```

=====PROG36.ASM=====03/09/2000=====
LIST P=16F84
INCLUDE "P16F84.INC"
ORG h'00'
GOTO BASLA
ORG h'04"
GOTO KESME

BASLA
BSF STATUS, 5
CLRF TRISB
CLRF TRISA
BCF STATUS, 5
CLRF PORTB
MOVLW b'10100000' ;TMR0 registerini kur.
MOVWF INTCON ;INTCON registerine yaz.
CLRWDI ;WDI'yi sil.
BSF STATUS, 5 ;BANK1
MOVLW b'11010001';TMR0'ı seç, dahili komut saykılı
MOVWF OPTION_REG
BCF STATUS, 5 ;BANK0
BCF PORTA, 0 ;Kondansatörü deşarj yap.
CLRF TMR0 ;TMR0'ı başlat, deşarjı başlat

DESARJ
BTFS TMR0, 7 ;Deşaj süresince bekle
GOTO DESARJ
BSF STATUS, 5 ;BANK1
BSF TRISA, 0 ;PORTA, 0. bit giriş
BCF STATUS, 5 ;BANK0
CLRF TMR0 ;TMR0'ı (Şarj süresini) başlat

V_FLASH
BTFS PORTA, 0 ;RA0 girişine bak. 0. bit "1" mi?
GOTO V_FLASH ;Hayır, tekrar başla
MOVF TMR0, W ;Evet, giriş "1" oldu. TMR0'ı oku
MOVWF PORTB ;Kondansatör dolma süresini göster
BCF INTCON, 5 ;TMR0 kesmelerini geçersiz yap

DONGU
GOTO DONGU ;İşlem bitti

KESME

```

```

        BCF   INTCON, 5      ;Sonraki TMR0 kesmelerini iptal et
        MOVLW h'AA'
        MOVWF PORTB         ;TMR0 süre dolma durumunu göster
DONGU   GOTO  DONGU
        END

```

İşlem Basamakları

1. PIC16F84'ü programlayınız.
2. Örnek olarak verilen devreyi breadboard üzerine kurup, PIC'i yerleştiriniz
3. PIC'e enerji verdikten çok kısa bir süre sonra portB'deki LED'lerde binar sayıyı görünüz.
4. Potansiyometreyi tekrar ayarlayıp, RESET tuşuna basarak tekrar programı çalıştırın. Yeni direncin LED'lerde bir öncekinden farklı değer gösterdiğini izleyiniz.

Buzzer'in Çalıştırılması

Örnek olarak verdiğimiz deneme kartının A portunun 3. bit'ine bir buzzer bağlıdır. Bu buzzer'e gönderilen palslerin aralıkları değiştirilmek suretiyle buzzer'den çıkan sesin yüksekliği değiştirilebilir. Aşağıdaki program bu işleme örnek bir programdır.

PROGRAM-37) A1 butonuna basılı tutulunca, portA'nın 3. bit'ine bağlı ole buzzer'de kesik ve kalın bir ses çıkarır. A2 butonu basılı tutulunca ise kesik ve t ses çıkarır. Kesik seslerin buzzerden çıktığı anda portB'deki LED'ler de yan söner.

```

;====PROG37.ASM====06/09/2000=====
        LIST  P=16F84
        INCLUDE "P16F84.INC"
DLYH EQU  h'08'      ;Gecikme zamanı sayacı 1
DLYL EQU  h'09'      ;Gecikme zamanı sayacı 2
BZCNT EQU  h'0C'      ;Buzzer çalışması
BASLA
        MOVLW h'00'
        MOVWF PORTA   ;PortA'yı sil
        MOVWF PORTB   ;PortB'yi sil
        BSF  STATUS, 5 ;BANK1
        MOVLW b'00010111';RA0-RA2 giriş, RA3 çıkış
        MOVWF TRISA    ;PORTA'yı yönlendir
        MOVLW b'00000000';RB0~RB7 çıkış
        MOVWF TRISB    ;PORTB'yi yönlendir
        BCF  STATUS, 5 ;BANK0
        BSF  PORTA, 3  ;RA3=1
MLOOP
        CLRWD
        BTFSC PORTA, 1      ; RA1=0 nu.?
        GOTO MAIN1
        MOVLW h'FF'
        MOVWF PORTB        ;Tüm LED'leri yak
        CALL BZCALLL      ;BZ Pi
        MOVLW h'00'
        MOVWF PORTB        ;Tüm LED'leri söndür.
MAIN1
        BTFSC PORTA, 2      ; RA2=0 mı?
        GOTO MAIN2
        MOVLW h'FF'
        MOVWF PORTB        ;TÜM LED'leri yak
        CALL BZCALLH      ; BZ Pi
        MOVLW h'00'
        MOVWF PORTB        ;Tüm LED'leri söndür
MAIN2
        CALL DLY50MS
        CALL DLY50MS
        CALL DLY50MS
        CALL DLY50MS
        CALL DLY50MS
        CALL DLY50MS
        GOTO MLOOP
;Zaman geciktirme alt programı
DLY500U

```

```

        MOVLW d'02'
        GOTO DLYP3
DLY50MS
        MOVLW d'255'
DLYP3
        MOVWF DLYH
DLYLP2
        MOVLW d'33'
        MOVWF DLYL      ;CLKx3xTime
DLYLP1
        DECFSZ DLYL, 1      ;DLY-1
        GOTO DLYLP1
        DECFSZ DLYH, 1
        GOTO DLYLP2
        RETLW 0      ;Return
;Buzzer call subroutine
BZCALL
BZCALLH
        MOVLW d'44'      ;44 defa
        MOVWF BZCNT
BC1
        BCF PORTA, 3      ;RA3=0
        CALL DLY500U
        BSF PORTA, 3      ;RA3=1
        CALL DLY500U
        DECF BZCMT, P
        BNZ BC1
        RETLW 0
BZCALLL
        MOVLW 44
        MOVWF BZCNT
BC2
        BCF PORTA, 3      ;RA3=0
        CALL DLY500U
        CALL DLY500U
        BSF PORTA, 3      ;RA3=1
        CALL DLY500U
        CALL DLY500U
        DECF BZCMT, F
        BNZ BC2
        RETLW 0
END

```


Kitabın içerisindeki örnek programları denebileceğiniz PIC deneme kartının şeması ve aşağıda verilmiştir. Bu kart bizim tarafımızda üretilmiş ve elektronik malzemelerin kolaylıkla elde edilebilir olması en büyük avantajıdır. isterseniz, internet sitelerinden parasız olarak elde edebileceğiniz benzer kart şemaları ve baskı devre şemalarını download edip kendi olanaklarınızla üretebilirsiniz.

