

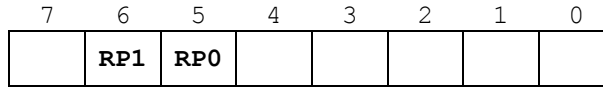
Yukarıda görülen RAM haritasında sadece bu programda kullanılacak olan özel registerler gösterilmiştir. Diğer registerlerin ne işe yaradığı ileriki programlarda sırası geldiğinde verilecektir.

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 REGISTER



Bank seçme bit'leri

00 → Bank0	}	PIC16F84'de kullanılmaz.
01 → Bank1		
10 → Bank2		
11 → Bank3		

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.

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

Port'ların Giriş veya Çıkış Olarak Yönlendirilmesi

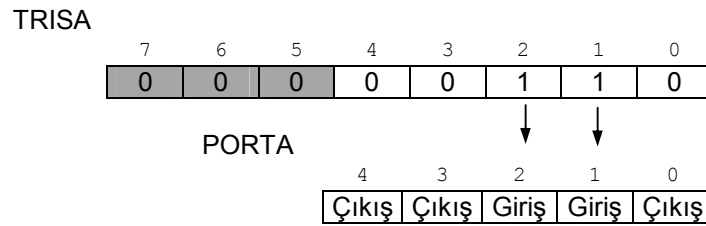
PortA ve PortB'nin uçlarına bağlı bulunan bir I/O elemanını kullanabilmek için portları giriş veya çıkış olarak yönlendirmek gerekir.

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 içerisinde "0" yazılır.

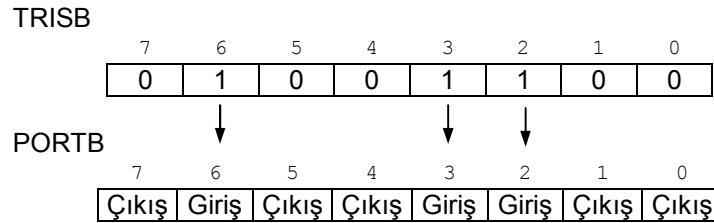
TRISA } 1 → PortA → Giriş
TRISB } 0 → PortB → Çıkış

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



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

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

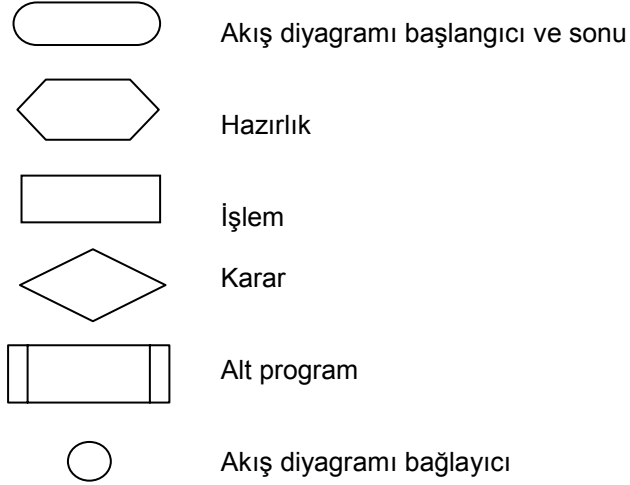


AKIŞ DİYAGRAMI SEMBOLLERİ

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 olunur.

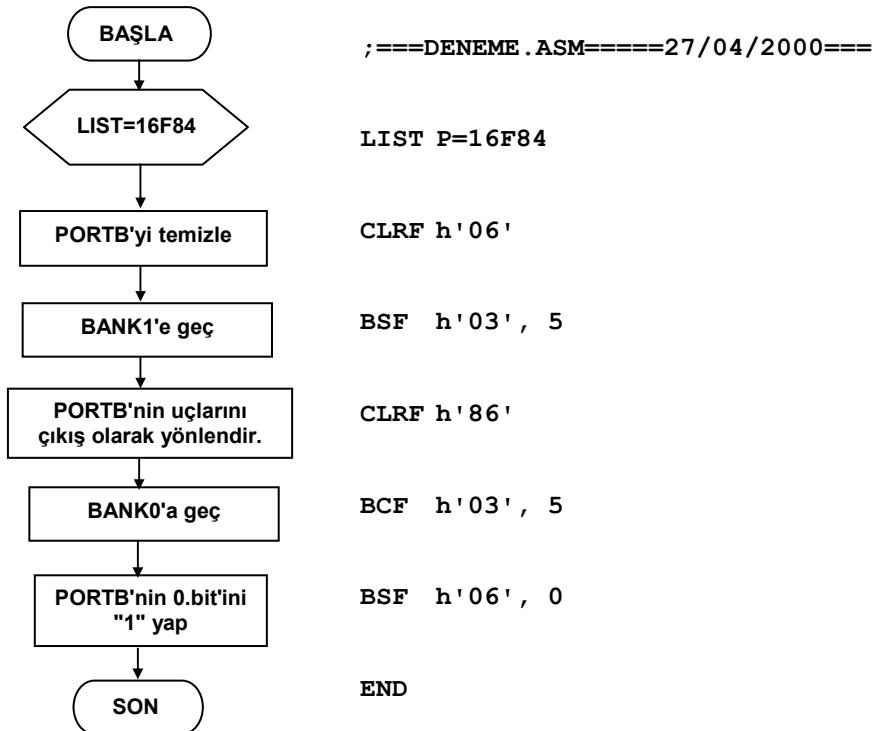
Programlama dillerinde ortak olarak kullanılan akış sembolleri vardır. Bu sembollerin neler olduğunu ve nerelerde kullanılacağını aşağıda göreceksiniz.

54 Mikrodenetleyiciler ve PIC Programlama



AKIŞ DİYAGRAMININ ÇİZİLMESİ

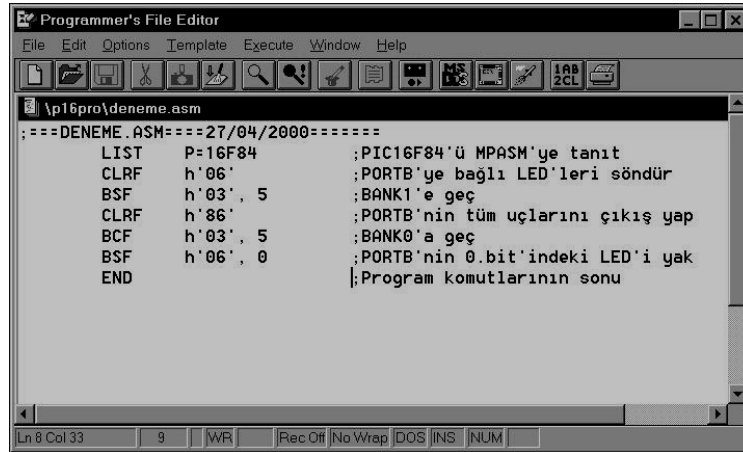
İlk programımız, PIC'e enerji verdiğimizde PORTB'nin 0. bit'ine bağlı LED'i otomatik olarak yakacaktır. Buna göre akış diyagramını çizelim.



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, WINDOWS 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 tanıtt
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
```

İlk 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'
        PORTB'nin RAM bellekteki adresi
BSF     h'03', 5
        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 kaydedeceğinizi 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 tanıt
PORTB   EQU    h'06'
STATUS  EQU    h'03'
TRISB   EQU    h'86'
CLRF    PORTB      ;PORTB'ye bağlı 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.bit'indeki LED'i yak
END      ;Program komutlarının sonu
```

Görüldüğü gibi komutların hangi registeri kullandığı kolayca anlaşılmalıdır. Örneğin;

```
BSF STATUS, 5 ;BANK1'e geçmek için STATUS
               ;registerin 5. bit'ini "1" yap
```

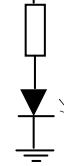
STATUS

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

```
BSF PORTB, 0 ;PORTB'nin 0. ucuna bağlı LED'i
              ;yakmak için 0. bit'i "1" yap
```

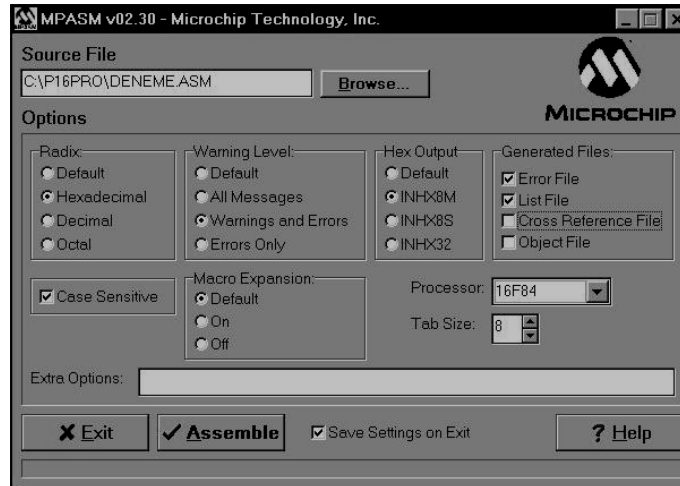
PORTB

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



PROGRAMLARIN DERLENMESİ (MPASM)

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



58 Mikrodenetleyiciler ve PIC Programlama

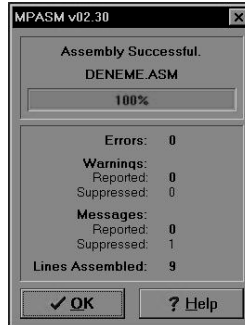
MPASM derleyici programı kullanmak oldukça kolaydır. Yukarıdaki ekran üzerinde görülen çoğu seçenek kaynak programın derlenmesinde önemli bir rol oynamaz. Ancak bizim kitapta açıklayacağımız bazı önemli dosyaların (.ERR ve .LST) üretilmesi için aşağıda açıklanan düzenlemeyi yapmanızda yarar vardır.

- Radix bölümünde "**Hexadecimal**"i seçiniz.
- Warning bölümünde "**Warnings and Errors**"ü 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**"ü 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 derleyeceğimiz programın adını göreceksiniz.
- Programı derlemek için "**Assemble**" butonuna tıklayınız.

Programın başarıyla derlendiği aşağıdaki pencerede görüldüğü gibi "**Assembly Successful**" mesajından anlaşılır. Aynı zamanda derleme başarılı olduğunda 100% yazan pencerenin yeşil renge dönüşmesinden anlaşılır.



NOT: Kaynak programda(DENEME.ASM) hatalar varsa, pencere içerisindeki mesaj "Errors Found" biçiminde olacaktır. Pencerenin ortasındaki bandın rengi ise kırmızıya boyanacaktır. Eğer böyle 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 düzeltileceği ile 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.