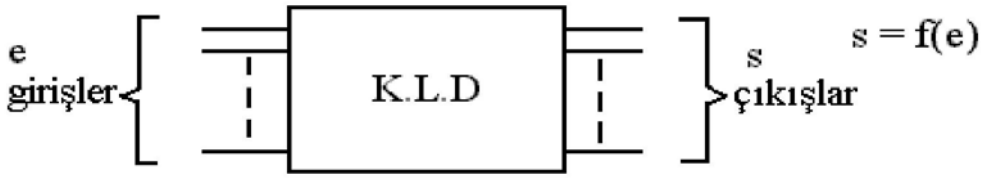
DEVRELENDİRİLMİŞ LOJİK

Lojik (sayısal) Devreleri genel olarak 3 ana grupta inceleyebiliriz ;

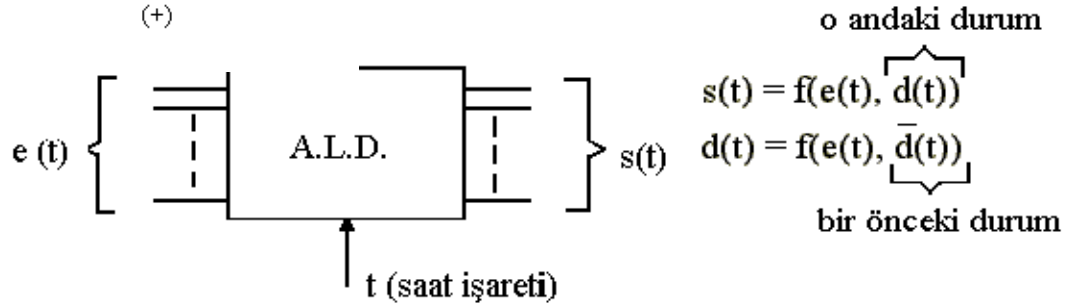
1-) Kombinezonsal Lojik Devreler:



Bu tür lojik devrelerde **herhangi bir andaki çıkışlar**, **sadece girişler tarafından** belirlenir. Bir başka ifadeyle K.L.D. 'lerde çıkışlar zamana bağlı değildir.

2-) Ardışıl Lojik Devreler

$Ç(+25)_{10} = (0.0011001)_2$ (işaret biti (+)) onksiyonu olmayıp aynı zamanda ardışıl lojik devrenin **o andaki durumu** ise **bir önceki durum** ve girişler tarafından belirlenmektedir. O halde bu **elemanı** bulundururlar.



Yukarıdaki **t (saat işareti)** 'ne "**clock(saat) darbeleri**" de denir. Bunlar "Kare dalga" şeklinde periyodik işaretlerdir.

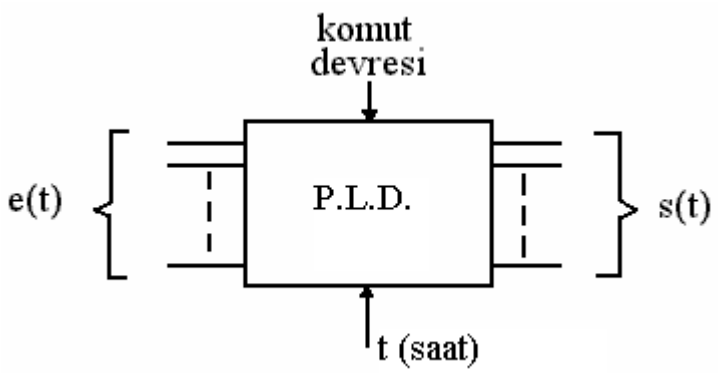
Ardışıl Lojik Devreler, "Senkron" ve "Asenkron" ardışıl lojik devreler diye ikiye ayrılırlar :

a) Senkron ardışıl lojik devrelerde işlemler, clock (saat) işaretleri ile senkronize (eş zamanlı) olarak gerçekleştirilir. Bu tip lojik devrelere "dijital saat devresi" örnek olarak verilebilir.

b) Asenkron ardışıl lojik devreler ise; devrenin girişindeki değişimlere bağlı olarak işlemler artarda (farklı zaman dilimlerinde) gerçekleşir. Bu tür devrelerde saat işareti bulunması da şart değildir. Örnek olarak, para ile çalışan meşrubat makinesi **para atıldığı zaman** harekete geçecek (durum değiştirecek) şekilde tasarlanabilir.

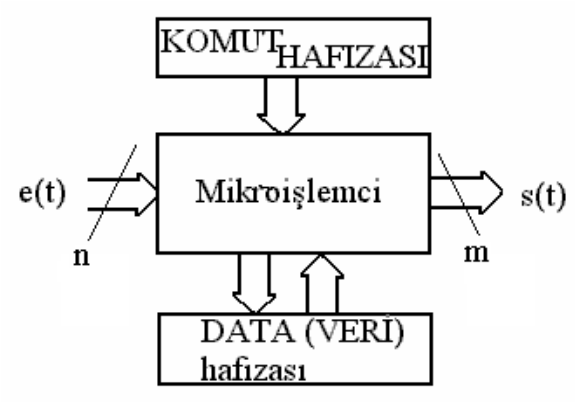
3-) Programlanmış Lojik Devreler:

Devrenin çalışabilmesi veya çıkışında elde edilecek işaretler, **hafızadaki programa** (komut devresine) bağlıdır. Bu dersin konusunu bu tür lojik devreler teşkil etmektedir.



Burada da A.L.D. ye benzer şekilde saat (darbe) girişi vardır ve her "t" anında bir işlem (toplama, çıkarma, lojik VEYA ,OR gibi) gerçekleşir.

Benzer olarak e(t):girişler ; s(t):çıkışlar olmak üzere **"n" tane giriş** ve **"m" tane de çıkışa** sahip bir mikroişlemci düzenini daha detaylı bir şekilde aşağıdaki gibi çizmek mümkündür :



Sistemin genelde birden fazla giriş ve çıkışları olduğu halde programı saklamak için bir **Komut Hafızası** üzerinden işlem yapılacak ve neticede elde edilecek datayı (veriyi, yani sayıyı) saklamak ve okumak için **Data (yada Veri) Hafızası** kullanılacaktır. Mikroişlemcide yapılacak işlemler ise **sırasıyla** komut hafızasından mikroişlemciye gönderilerek icra edilmesi sağlanır.

SAYI SİSTEMLERİ

Sayı sistemleri iyi anlaşılmadan mikroişlemcilerle(ya da mikrodenetleyicilerle) uğraşmak ve onları anlamak imkansız gibidir. Bu nedenle çeşitli örnekler üzerinde sayı sistemleri ve dönüşümlerini anlamaya çalışalım.

$7392 = (7392)_{10} = 7*10^3 + 3*10^2 + 9*10^1 + 2*10^0$
$(23.62)_{10} = 2*10^1 + 3*10^0 + 6*10^{-1} + 2*10^{-2}$
$(4021.2)_5 = 4*5^3 + 0*5^2 + 2*5^1 + 1*5^0 + 2*5^{-1} = (511.4)_{10}$
$(101)_2 = 1*2^2 + 0*2^1 + 1*2^0 = (5)_{10}$

Aşağıda 0 – 15 arasındaki **desimal** sayıların heksadesimal ve binary tabandaki karşılıkları verilmiştir.

Desimal (10`lu)	Heksadesimal (16`lı)	Binary (2`li)
0	0	0000
1	1	0001
2	2	0010

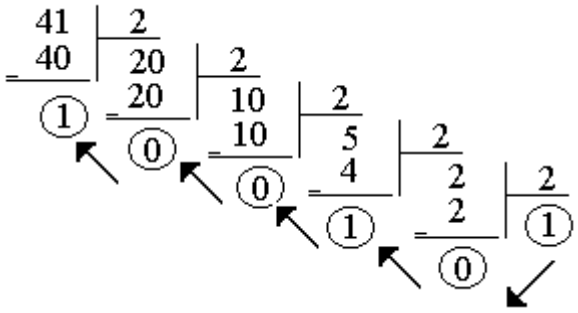
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Örnek Bazı Çevirme İşlemleri:

1) $(0010\ 0111\ 0001\ .\ 1111\ 1100)_2 = (271\ .\ FC)_{16}$ (İkili tabandaki sayı dörtlü gruplar halinde düzenlenerek yukarıdaki tablodan çevrilir.)

$(\ 2\quad 7\quad 1\ .\ F\quad C\ )_{16}$

2) $(41)_{10} = (?)_2$ (Desimal sayıyı binary sayıya çevirmek için sürekli 2 'ye böler ve ters sırayla kalanları alırsak sayının ikili tabandaki karşılığını elde ederiz)

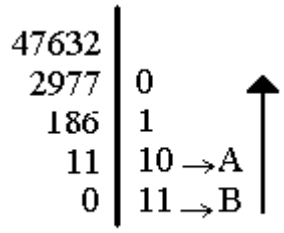


1-0-0-1-0-1 => tersten yazalım => $(101001)_2 = (41)_{10}$

Bu yöntemle on tabanındaki sayıları ikili tabana çeviriyoruz.

3) $(47632)_{10} = (\ ?\ )_{16}$ (**Not**: Desimal sayıyı heksadesimale çeviriyoruz)

Bu çevirme işlemi için de benzer tarzda sayıyı 16 ile artarda böleceğiz. Yine kalanları elde edip gerekenleri 16'lı tabana dönüştüreceğiz.
(Bölme işlemi farklı bir formatta (şekilde) gösterilmiştir. Çizginin solunda artarda 16'ya bölme sırasındaki **bölümler**, sağ tarafta ise **kalanlar** yer almaktadır.)



$(BA10)_{16} = (47632)_{10}$ olarak sonuç elde edilmiş olur.

4) $(0.263)_{10} = (?)_{16}$

Verilen sayı **ondalıklı sayı olduğu için** bu defa **16 'ya bölmeyip 16 ile çarpacağız.**

0.263 * 16 = 4 .208
0.208 * 16 = 3 .328
0.328 * 16 = 5 .248

Burada bölme yerine çarpma yapıldığından her alınan sonucun **<u>tam sayı kısmı</u>** baştan sona doğru (ters yönde değil !) sıralanarak yazılarak sonuç bulunur.

0,------4------3------5 => $(0.435)_{16} = (0.263)_{10}$

5) $(CA\,.\,03)_{16} = (\,?\,)_2$

Burada ise her heksadesimal (16 lı) sayının binary (ikili) karşılığı alınıp yan yana konularak netice elde edilir.

C= 1100, A= 1010, 0= 0000, 3= 0011 olduğu daha önce verilen tablodan bilindiğinden

$(CA.03)_{16} = (1100\ 1010\ .\ 0000\ 0011)_2$ şeklinde ikili tabandaki sonuç elde edilir.

<u>Komplement (Tümleyen) Kavramı:</u>

Bir sayının **iki tip (tür) komplementi** elde edilebilir:

1- Tabana göre komplement (tümleyen)
2- [Taban – 1] e göre komplement (tümleyen)

<u>Örnekler:</u>

Bu kavramı doğrudan örnekler üzerinde inceleyerek anlamaya çalışalım:

- $(52520)_{10}$ → **Tabana göre komplementi** → $10^5 - 52520 = 47480$
  (Sayı 5 hane verilmiş, o halde 10^5 'den verilen sayı çıkartılarak netice bulunuyor.)

- $(0.3267)_{10}$ →**Tabana göre komplementi** →$10^0 - 0.3267 = 0.6733$
  (Sayının tam kısmı yok ! , $10^0 = 1$ 'den verilen sayı çıkarılmış.)

- $(101100)_2$ →**Tabana göre komplementi** →$2^6 - (101100)_2 = (1000000)_2 - (101100)_2 =$
  $= (010100)_2$
  (2^6 sayısı binary olarak 1000000 olarak yazılır)
  ****

- $(52520)_{10}$ → **(Taban-1)'e göre komplementi** →$10^5 - 1 - 52520 = 47479$
  (Yukardaki "tabana göre" olan sonuçtan 1(bir) çıkarılırak netice bulunur)

- $(0.3267)_{10}$→ **(Taban-1)'e göre komplementi** →$10^0 - 10^{-4} - 0.3267 = 0.6732$

- $(101100)_2$ → **(Taban-1)'e göre komplementi** →$(2^6 - 1) - (101100)_2 =$
  $= (111111)_2-(101100)_2 = (010011)_2$ olur.

Binary (İkili) Tabanda Bazı İşlemler:

Biz ikili tabandaki işlemleri genellikle **8 haneli** olarak yapacağız.(Bunun sebebi, ilerde incelenecek PIC mikrodenetleyicinin 8 hane (bit) olmasıdır, böylece buna bir hazırlık yapılmaktadır.)

Aşağıdaki örnekler hem binary hem de heksadesimal (16 'lı) tabanda yapılmış ve sonuçların aynı olduğu gösterilmiştir.

1) Toplama İşlemi (8 haneli)

Binary	**Hex**
1000 1100	8 C
+ 0111 1101	+ 7 F
1 0000 1011	*1* 0 B

Buradaki *1* **elde** olup ihmal edilmez. $(0000)_2 = (0)_{16}$ ve $(1011)_2 = (B)_{16}$ idi.

2) *Çıkarma İşlemi (8 haneli)*

Binary	**Hex**
1000 1100	8 C
- 0111 1111	- 7 F

Bu işlemin sonucunu öğrenebilmek için normal yoldan çıkartma işlemi yapabiliriz.(Bunun için en sağdaki 0 dan 1 çıkarmaya çalışacak çıkmadığı için bir üst haneden 2 borç alıp 2 'den 1 çıkarıp işlemlere devam edecektik) Fakat bu tarz çıkartma, toplamaya göre oldukça zahmetli olacaktır. (Mikroişlemci içindeki Lojik devreler için de bu işlemin gerçekleşmesi toplamaya göre karmaşık yapı gerektirmektedir. Bu yüzden pratikte çıkartma işlemi toplamaya çevrilerek işlemler yapılır.)

Bu amaçla çıkartmada **Komplement (Tümleyen) aritmetiği** kullanılması işi oldukça basitleştirmektedir. Zira, ikinci sayının (çıkanın) tabana göre komplementi elde edilirse artık işlem toplamaya dönüşmüş olmaktadır ki bütün mikroişlemciler bu yolu kullanmaktadır.

Buna göre **birinci sayı (çıkarılan) aynen bırakılır**, üzerinde herhangi bir işlem yapılmaz ! Sonra ikinci sayının **(çıkanın) tabana göre komplementini** bulmamız gerekir. Ancak, daha önce komplement kavramı kısmında verilen örneklerden de anlaşılabileceği gibi komplement bulma işlemi de farklı bir çıkartma gerektirmektedir.

Bunun yerine komplement almada kolay bir yöntem olan aşağıdaki metot uygulanabilir:

Çıkan olan $(0111\ 1111)_2$ sayısının tabana göre komplementi basit olarak iki aşamada bulunabilir:

- **Önce çıkanda** *1 olan bitler(haneler) 0* **;** *0 olan bitler ise 1* **yapılır. Böylece sayımız $(1000\ 0000)_2$ olacaktır. Bu sayı, asıl sayının (taban - 1) e göre komplementidir.**

- Daha sonra bu sayıya **1 eklediğimizde artık tabana göre komplement** elde edilmiş olacaktır. Yani çıkanın tabana göre komplementi olan **$(1000\ 0001)_2$** sayısı elde edildiğinden artık işlem **toplamaya dönüşmüş** olacaktır. O halde çıkartmayı toplamaya dönüştürdüğümüze göre işlemin son halini tekrar yazalım.

Binary:		***Hex :***	
	1000 1100		8 C
	+ 1000 0001		+ 8 1
	1 0000 1101		*1* **0 D**

($1101)_2 = (D)_{16}$ olduğunu hatırlayınız.

Buradaki başta bulunan *1* **Komplement aritmetiği kullanıldığı için** ihmal edilir. Bu *1* aslında çıkarılanın çıkandan büyük olduğunu da göstermektedir. Ve çıkartma işleminin sonucu $(0000\ 1101)_2 = (0D)_{16}$ olarak elde edilmiş olur.

İşaretli Sayılar Kavramı:

Sayıya işaretini gösteren bir hane (bit) ayrılır. Bu bit "0" ise sayı pozitif, "1" ise negatif kabul edilir. **İşaret biti** genelde **en soldaki bit** olarak seçilir. (Mikroişlemci işlem yaparken bunu bilmez. Çünkü bu bir varsayımdır, sayıya **bir bakış açısıdır**. Yoksa işaretli sayılarla yapılan işlemin işaretsiz sayılarla yapılan işlemlerden (yukarıdakinden) farkı yoktur.)

Pozitif Sayılar:

İşaret biti ve **sayının mutlak değeri ile** gösterilir. (Burada nokta işaret bitini ayırt etmek için kullanılmıştır.)

$(+25)_{10} = (0.001\ 1001)_2 = (19)_{16}$ [Burada +25 desimal sayısı mikroişlemcide 19 heksadesimal sayısı ile temsil edilmektedir. Desimal heksadesimal (hex) dönüşümünü hatırlayın]

Negatif Sayılar:

İşaret biti ve **sayının mutlak değerinin tabana göre komplementi** ile temsil edilirler.

$(+25)_{10} = (0.001\ 1001)_2 = (19)_{16}$ olduğunu yukarıda bulmuştuk.

$(+25)_{10}$ sayısının **tabana göre komplementi** (-25) 'tir , ve daha önce verildiği gibi bunu şöyle elde ederiz :

- Önce (0.001 1001) sayısının dualini alalım...(Yani, sayıda 0 yerine 1 ; 1 yerine 0 yazalım.)
- Karşımıza çıkan bu sayı (taban-1) e göre komplement olan (1.110 0110) olacaktır . Bu sayıyı bir arttırdığımızda tabana göre komplementini buluruz : (1.110 0111)

İşte bu $(1.110\ 0111)_2$ sayısı $(-25)_{10}$ sayısına karşılık geliyor (1.) ile başladığı için de negatiftir. Burada **baştaki bitten (1) sonra konan nokta** sayıları işaretli olarak düşündüğümüzü (yorumladığımızı) ifade ediyor. Yoksa mikroişlemcide işaretli işaretsiz sayı farkı YOKTUR.

Mikroişlemcinin hafızasında bu sayı artık $\mathbf{(1.110\ 0111)_2 = (-25)_{10} = (E7)_{16}}$ şeklinde temsil edilecektir.

İşaretli Sayılarla Çeşitli Örnekler:

1) $(+60)_{10} + (+62)_{10} = (+122)_{10}$ olduğunu gösterelim:

0.011 1100
+0.011 1110
0.111 1010 = $(+122)_{10}$ burada en baştaki **(0.) sayının (sonucun) pozitif olduğunu** gösterir.

2) (+60) + (+70) = (+130) olduğunu gösterelim:

0.011 1100
+0.100 0110
1.000 0010 sonucu elde edilir.

Dikkat edilirse sonuç negatif **(!)** gözükmektedir. **Halbuki matematikte iki pozitif sayının toplamı daima pozitiftir.**

Bu durumda **işaretli sayılar gözüyle** yapılan bu toplama işleminin sonucunda işaretin, dolayısıyla **neticenin hatalı** olarak elde edildiği görülmektedir. O halde işaretli işlemlerde dikkatli olunmalıdır.

3) (-60) + (-70) = (-130) olduğunu gösterelim:

Önce (+60) = $(0.011\ 1100)_2$ ve (+70) = $(0.011\ 1100)_2$ olduğunu önceki örnekten biliyoruz. Bu iki sayının negatiflerini bulmak için **Taban`a göre komplementini** alalım:

1.100 0100 = $(C4)_{16} = (-60)_{10}$,
1.011 1010 = $(BA)_{16} = (-70)_{10}$ değerlerini buluruz.

Bu sayıları yukarıdaki gibi toplarsak(- 130) değerini yani gerçek sonucu bulamayacağız, yine sonucun hatalı olduğunu göreceğiz . O HALDE :

ÖNEMLİ SONUÇ : İşaretli sayılarla yapılan işlemlerde **sonuç $(-128)_{10} = (80)_{16}$ 'den küçük ya da $(+127)_{10} = (7F)_{16}$ 'den büyük ise** , bir başka ifade ile sağdaki bitlerden işaret bitine bir etki gelmiş ise sonucun işareti, dolayısıyla kendisi **hatalı** olacaktır. İşaretli işlemlerde netice ASLA **$(80)_{16}$ ve $(7F)_{16}$** aralığının dışına taşmamalıdır.

MİKROİŞLEMCİ (Microprocessor) NEDİR?

CPU (Merkezi İşlem Birimi) olarak da adlandırılır. Bilgisayar programının yapmak istediği işlemleri icra eder.(yerine getirir) CPU hafızasında bulunan komutları **sıra ile** işler. Bu işlem, komutun bulunup getirilmesi (Fetch), kodunun çözülmesi (Encode) ve işlemin yerine getirilmesi (Execute) gibi aşamaları gerektirir.

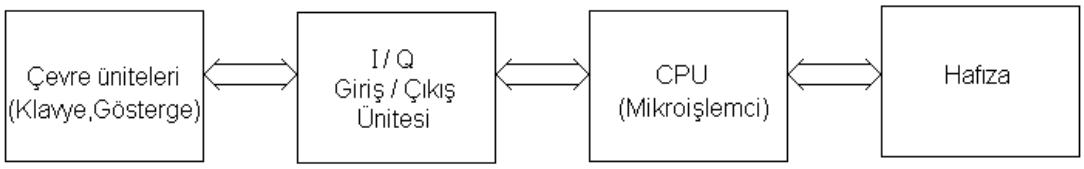
Merkezi İşlem Birimleri (CPU) Kişisel Bilgisayarlarda (PC) kullanıldıkları gibi , sanayi tezgahları, ev aletleri gibi birçok alanda da kullanılırlar. Ancak Mikroişlemciler tek başına çalışamazlar. İhtiyaç duydukları bazı ek elemanlar vardır. Bu elemanlar temel olarak ;

1) Giriş (Input) Ünitesi
2) Çıkış (Output) Ünitesi
3) Hafıza (Memory) Ünitesi 'dir.

Bu elemanlar CPU dışında olduklarından aralarındaki iletişimi VERİ YOLU (Data Bus) ve ADRES YOLU (Address Bus) ile Kontrol hatları (Control Lines) denilen lojik iletim hatları sağlar. Intel, AMD, Cyrix mikroişlemci üreticilerinden bazılarıdır. Şu anda mikroişlemcilerin sadece kişisel bilgisayarlarda PC'lerde kullanıldığını söylersek pek yanlış olmaz. Kontrol işlemlerinde, sanayide

yukarıda sözü edilen ek elemanları da içinde bulunduran **mikrodenetleyiciler (mikrocontroller)** tercih edilir.

CPU ve Çevre Elemanların Basit Yapısı :



MİKRODENETLEYİCİLER (Microcontrollers)

Bir bilgisayar içinde bulunması gereken Hafıza , Giriş/Çıkış ünitesi gibi elemanların CPU ile birlikte **tek bir entegre (chip)** içerisinde üretilmiş haline **Mikrodenetleyici** denir. Böylece hem yer tasarrufu yapılıp maliyet düşürülürken hem de tasarım kolaylaştırılmış ve programlama işlemi basitleştirilmiş olur.

Günümüzde mikrodenetleyiciler otomobillerden kameralara , cep telefonlarından oyuncaklara kadar sayılamayacak alanlarda kullanılır. Mikrodenetleyiciler birçok firma tarafından üretilmektedir. Microchip , Intel , Motorola , SGS Thomson, Hitachi gibi …

Her üreticinin en az birkaç mikrodenetleyicisi vardır. Mesela ; Microchip 12C508 , 16C84 , 16F84 ve 16F877 gibi farklı mikrodenetleyicilere sahiptir ve hemen, hemen aynı komutlarla programlanırlar. Mikrodenetleyici adlarında bulunan harfler aynı aile içinde farklı özelliklere sahip (hafıza yapısı ve miktarı, hız gibi) elemanları ifade eder. Bir uygulama yapmadan önce hangi firmanın , hangi numaralı mikrodenetleyicisinin kullanılacağı tespit edilmelidir. Bunun için KATALOG (Data sheet) adı verilen kaynaklardan ya da internetteki ilgili sitelerden faydalanılır.

Bir Mikrodenetleyici İçinde Bulunabilen Bazı Özellikler Şunlardır :

1) Programlanabilir Dijital Giriş / Çıkış (I / O)
2) Programlanabilir Analog Giriş
3) Seri Giriş / Çıkış (I / O)
4) Darbe(Pals) – PWM (Darbe genişlik modulasyonu) işareti çıkışı
5) Harici hafıza bağlanabilme
6) Dahili hafıza seçenekleri (ROM , PROM, EPROM , EEPROM, Flash gibi)
7) Kesme, Zamanlayıcı ve Sayıcı gibi özellikler.

MİKRODENETLEYİCİ SEÇİMİ

Mikrodenetleyici seçerken öncelikle **uygulama ihtiyacımızın tamamını** karşılamasına sonra da **fiyatına** bakarız. Ayrıca yazılım (program) desteğinin/araçlarının (derleyici, simulatör , emulatör v.s.) bulunup bulunmadığına dikkat ederiz. Piyasada - internette bol miktarda uygulama programlarının bulunabilmesi de örnek olması açısından faydalıdır. Sayılan özellikler göz önüne alınırsa (şu an için) Microchip firması tarafından üretilen kısaca PIC olarak ifade edilen mikrodenetleyicilerin kullanılması oldukça avantajlı gözükmektedir.

PIC İngilizce'de "Çevre Üniteleri Kontrol edici Arabirim" anlamı taşıyan kelimelerin baş harflerinden oluşmuştur.

PIC Mikrodenetleyicilerin Diğer Bazı Avantajları :

1) Destek Yazılımları internetten ücretsiz sağlanır.
2) Çok yaygın ve ucuzdur. Hem profesyonel hem de amatör kullanıma uygundur.
3) İnternette ve kitap/dergilerde çok sayıda örnek programlar vardır.
4) Çok az ve basit birkaç elemanlarla (direnç , kondansatör) donanımları kurulabilir.
5) Komut sayısı az ve basittir.
6)Daha üst seviye diller için (PIC C, PicBasic gibi) **Derleyiciler'e** (compiler) sahiptir.

PIC 'lerde özellikle az sayıda ve basit komutlarla programlanma yapılması,

PIC 16F84 ve PIC 16F877 Mikrodenetleyicileri :

Laboratuarda incelenecek setlerde PIC16F877 bulunmasına rağmen temel ve basit birçok PIC 'in özelliklerini içeren PIC16F84 'ün incelenmesi **başlangıç için** daha yararlı olacaktır. Her iki mikrodenetleyicisinde program hafızaları FLASH (EEPROM) hafızadır. Ve elektrik kesilse bile içlerinde yazılmış **program silinmemektedir**. Yeni başlayanlar ve araştırma yapanlar için kolayca yazılıp silinebilen özelliğe sahip olan Flash Hafızalı PIC 'ler bu açıdan da büyük kolaylık sağlamaktadır. Program hafızası EPROM olanlar da yazılıp silinmekteyse de bu tiplerin içindeki programı silmek için üzerinde bulunan penceresinden 10 -15 dakika UV (Ultra Viole) ışını uygulamak gerekir. PIC16F84 için öğrenilenler önemli oranda PIC 16 / 17 ailesinin tamamı için de geçerlidir.

Bir PIC Programlamak İçin Gerekenler :

1) Bir PC (Kişisel Bilgisayar)
2) Bir metin editörü (Not defteri gibi) kullanmayı bilmek
3) PIC Assembler programına (MPASM, MPLAB gibi) sahip olmak
4) PIC programını entegreye yüklemek(programlamak) için gerekli donanım ve yazılıma sahip olmak
5) Kullanılacak PIC mikrodenetleyicisini edinmek
6) Programlamadan sonra çalıştırmak için güç kaynağı , birkaç elektronik eleman, breadboard (deneme kartı), ölçü aletine sahip olmak gerekir.

ASSEMBLER PROGRAMI

Biz programlarımızı PIC16F84 ve 16F877 'ye ait ve **35 komuttan oluşan** assembly programlama dili ile yazacağız. Basit bir metin editöründe (not defteri gibi) yazılacak olan bu program (asm uzantılı text dosyası) PIC'in çalıştırabileceği hale dönüştürülmelidir (derlenmelidir). Bu işlem PIC için en basit olarak **MPASM** denilen bir "Assembler "programı ile yapılmakta ve sonuçta (. hex) uzantılı ve icra edilebilir (Makine Dili) denilen dosya elde edilmektedir. Bu program artık PIC 'e yüklenmeye(kaydedilmeye) hazırdır. MPASM programının kullanılışı basit olan DOS versiyonları da vardır. Eğer derleme sonrasında MPASM **hata mesajları** vermişse bunların (. asm) dosyası üzerinde düzeltilmesi ve tekrar derlenmesi gerekir. **MPLAB** da benzer işi görmekle birlikte bazı kullanışlı ilave özellikler içermektedir.

PROGRAMLAYICI DONANIMI VE YAZILIMI

Makine diline çevrilmiş icra edilebilir programın (hex uzantılı dosya) PIC 'e yüklenmesi için bir "**Programlayıcıya**" ihtiyaç vardır. Piyasada özel bir mikrodenetleyiciyi yada ailesini programlayan programlayıcılar bulunduğu gibi her tür mikrodenetleyici ve hafızayı programlayan "**Universal Programlayıcılar**" da vardır.

MPASM tarafından derlenerek makine diline çevrilmiş hex uzantılı dosyanın PIC ‘e yazılması için programlayıcı ile birlikte bir de yazılımına da ihtiyaç vardır. Her programlayıcının kendine has yazılımı bulunmaktadır.

PROGRAMLANMIŞ PIC ‘ İN DENENMESİ

İlk deneme için bir breadboard (delikli deneme tahtası) üzerinde fazla zaman harcamadan PIC ’e ve osilatör devresine bağlanması gereken birkaç direnç , kondansatör ve kristal gibi elemanlardan faydalanmak mümkündür. Çıkışları gözlemek için ise LED veya gösterge (display) kullanmak uygun olacaktır. Eğer devre doğru kurulmuş ve beslenmiş ise programın çalışmaması halinde hataların incelenmesi ve programın assembly dili üzerinde düzeltildikten sonra yeniden derlenip PIC ‘e yüklenmesi gerekir.

PIC MİKRODENETLEYİCİLER TÜRLERİ VE ÖZELLİKLERİ

Microchip firması tarafından üretilen farklı PIC grupları (aileler) vardır. Bu aile isimleri verilirken **kelime boyu** (komut kelimesi de denen bu kavram ilerde açıklanacaktır.) dikkate alınmıştır. Aslında tüm PIC mikrodenetleyicilerde dışardan veri alırken yada dışarı veri gönderirken **8 bitlik veri yolu** kullanılır. Ancak program yazılırken aynı komutlar kullanılmasına rağmen bu komutların makine dili (hex) karşılıkları farklı kelime boylarında olabilir. Meselâ ;

PIC 16C5XX ailesi 12 bit kelime boyu
PIC 16CXXX ailesi 14 bit kelime boyu
PIC 17CXXX ailesi 16 bit kelime boyu
PIC 12CXXX ailesi 12 / 14 kelime boyuna sahiptir.

Biz programlayıcı olarak bu kelime boyları ile fazla ilgilenmeyiz . Bizim için seçilen PIC ‘in komutlarını , kullanma kural ve özelliklerini bilmek yeterlidir. Bu özelliklerden kasıt, PIC in hafıza tipi ve miktarı, giriş/çıkış (I/O) portu (ucu) sayısı , analog giriş kabul edip etmemesi v.s. gibi özelliklerdir. Bunları kataloglardan elde etmek mümkündür.

PIC Mikrodenetleyicilerde Hafıza Çeşitleri :

Bir PIC içersinde **iki tür hafıza** bulunur :
Bunlardan biri **“Program hafızası”** dır ve EPROM, EEPROM , vb. gibi hafıza türlerinden birine sahiptir.

Her PIC ‘de bulunan diğer hafıza türü ise “(Data)**Veri Hafızası**” olarak bilinir. Bu hafıza genellikle RAM türü **geçici bilgi depolama** alanıdır. (EEPROM hafızanın da veri saklama amaçlı kullanılan kısmı varsa da bu daha ileri programlama bilgisi gerektirmektedir.) File Registerleri de (W, STATUS, TRISA gibi) RAM hafızada yer alır. Elektrik kesildiğinde burada bulunan veriler silinir.

Bir başka ifadeyle, bir PIC mikrodenetleyicinin hafızası **Program ve Veri hafızası** olmak üzere **2 ayrı bölümden** oluşur. Ayrıca PIC ‘lerdeki BUS yapısı PIC ‘lerin **RISC işlemci** olarak tanıtılmasına, dolayısıyla komut sayısının az ve mikroişlemcilere göre hızlarının fazla olmasına neden olur.

Farklı özelliklerde “**Program hafızalarına**” sahip PIC ‘ler mevcuttur. Bunlar :

EPROM : Hafıza hücrelerine uygun elektrik işaretleri ile (genellikle besleme geriliminin üzerinde bir gerilimde) kayıt yapılır. Elektrik kesilse de program kalır. Silmek ve yenisini yazmak için ultraviole (UV) ışınının belirli bir süre penceresinden uygulanması lazımdır. 2 tip olurlar:

1) Pencereli (UV ışını ile) Silinebilir Tip

2) Penceresiz Silinemeyen (OTP : Bir defa programlanabilir) Tip

Silinmemesi için pencereli olanların pencereleri ışık geçirmeyen bantla kapatılır. OTP 'ler silinemezler, ancak fazla miktarda üretildiklerinde birim maliyetleri daha düşüktür. **Örn:** 16C74 pencereli EPROM tipi bir PIC 'dir.

.

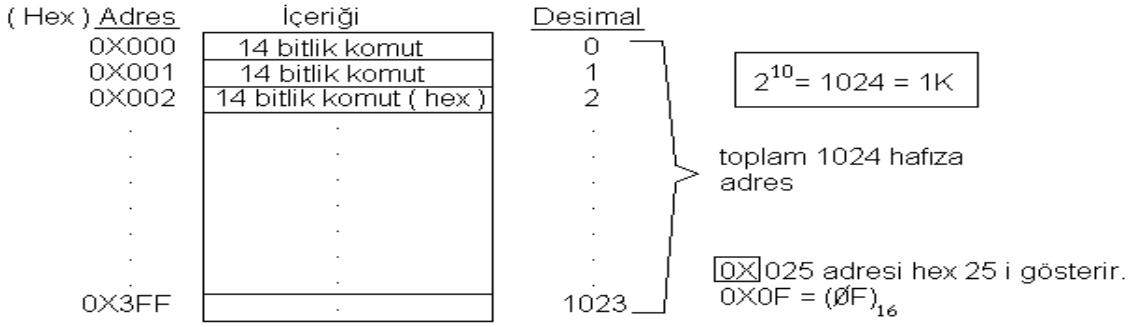
EEPROM (FLASH) : UV ışınına gerek kalmadan elektriksel olarak silinebilir tiptir. Uygulama geliştirme (deneme) maksadına elverişlidir. Dezavantajı hafızaya erişim (okuma / yazma) hızları EPROM 'lara göre düşüktür. **Örn:** 16CB84 , 16F84 bu tip hafızaya sahiptir.

ROM : Sadece okunabilir hafıza olarak bilinir. Bu tip program hafızaları fabrikasyon olarak bir kere yazılırlar. Ucuzdurlar. Ancak çok sayıda programlanmışlarsa ve programda hata olduğu sonradan anlaşılırsa üretilen entegrelerin atılmasını gerektir. **Örn:** PIC 16CR62 , 16CR84 gibi.

PIC16F84 de Program Hafızası :

16F877 den önce kendimize basit bir örnek olarak seçtiğimiz 16F84 ün 1KByte lık program hafızası vardır. Her bir hafıza hücresi içerisine 14 bit uzunluğundaki program komutları saklanır. Bunların her birine **" Komut Kelimesi "** denir. Program hafızası Flash (EEPROM) tipte olup program çalışması esnasında sadece okunabilir.

PIC16F84 için Program Hafıza Haritasını Çizelim :

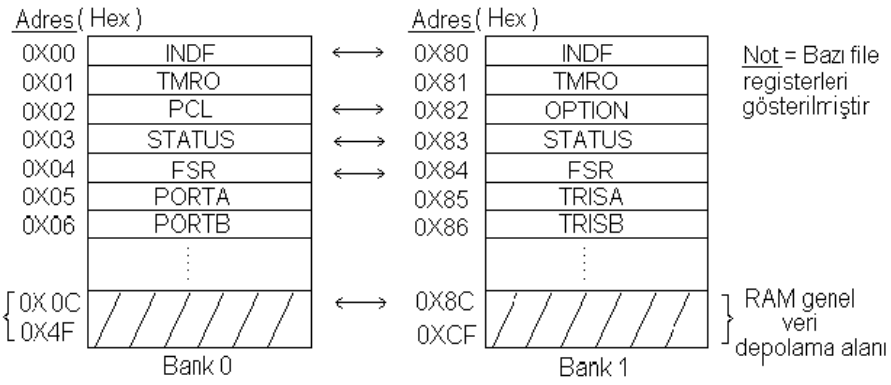


Program hafızasında sadece assembly komutlarının heksadesimal (hex) karşılıkları saklanır (PIC 16F84 için 14 bit). Yukarıdaki hafıza haritasında son adresin (1023)$_{10}$ olma sebebi ilk adresin sıfırdan (0x000) başlamasıdır. İlk adres (0x001)den başlasaydı son adres (1024)$_{10}$ olacaktı.
Ancak toplam adres (2^{10} = 1024) adettir.

PIC16F84 de RAM (VERİ) HAFIZA YAPISI

Başlangıç için örnek aldığımız PIC16F84 'ün (0x00 – 0x4F) adres aralığında toplam (50)$_{16}$ = (80)$_{10}$ adet 8 bit uzunluğunda veri hafıza **(RAM) adresi** vardır.[*NOT*: "0x" kendinden sonra gelen sayının heksadesimal olduğunu belirtir, örn: 0x4F = (4F)$_{16}$ aynı sayılardır] Bunların bir kısmı **Özel File Registerleri** denilen ve yine 8 bit uzunluğunda olan lojik bilgi saklayıcı/kaydedicilere ayrılmıştır. Bu File Registerleri dışında kalan hafıza alanları normal veriler için hafıza olarak kullanılabilir. Mesela 5 + 3 = 8 işlemi için 5 , 3 , 8 sayıları (verileri) buraya saklanabilir. Bu bilgiler 8 bit uzunluğunda olduğu için 05 , 03 , 08 şeklinde hafızada yer alırlar. İlerde bu konular daha ayrıntılı incelenecektir.

16F84 İçin Register Haritası :



Register haritası programın çalışması esnasında kullanılan değişkenleri geçici ya da enerji kesilene kadar saklamak için kullandığımız veri alanlarıdır.

PIC 16F84 ün RAM hafızası 2 sayfadan (Bank, kısım) meydana gelir:
Bank 0 'a ait adresler 0x00 – 0x4F ,
Bank 1 'e ait adresler 0x80 – 0xCF adres aralığındadır.

Register haritasına bakılırsa bazı file registerleri her iki bankta ortak olduğu görülür. Dolayısıyla bu registerlere her iki bankta iken erişilebilir. PCL , STATUS , FSR gibi…

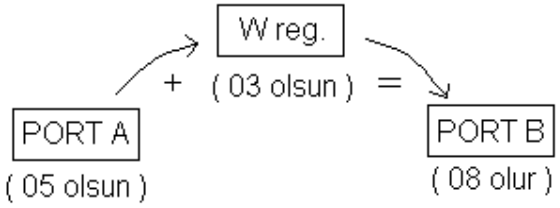
PIC16F84 için **Bank1** de **0x00 ile 0x0B arası özel fonksiyon registerlerine** ayrılmıştır. Benzer şekilde **Bank2** için **0x80 – 0x8B arası da** yine **özel fonksiyon registerlerine** tahsis edilmiştir.

Ayrıca **0x0C - 0x4F adres aralığı** ile **0x8C - 0xCF adres aralığındaki genel veri alanları** birbirlerinin kopyasıdır.Yani, örneğin 0x1B adresi ile 0x9B adresleri gerçekte aynı verileri içermektedir (Ayna Özelliği). Bir başka ifadeyle burada otomatik kopyalama işlemi söz konusudur. Bu sebeple veriyi birine yazıp diğerinden okumak mümkündür.

Bir Bank'ı (Bank 0 veya Bank 1) kullanabilmek için **o Bank'a geçmek gerekir**. Bu değiştirme işlemi programlama sırasında incelenecektir. Bazı sık kullanılan registerler (STATUS gibi) her iki bankta da bulunduğundan bunlar için bank değiştirmek gereksizdir.

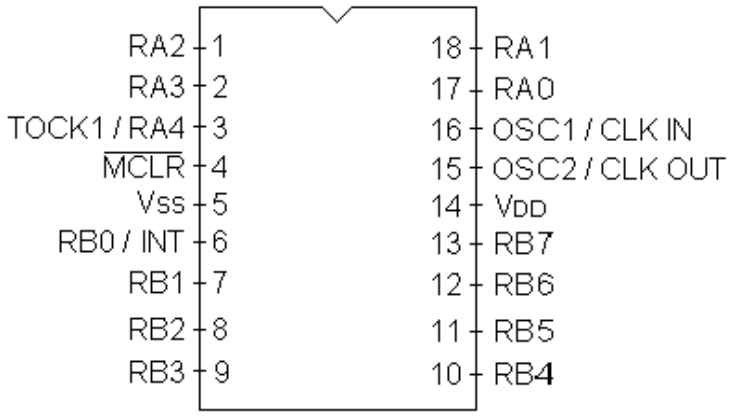
Özel Bir Register (W : Akümülatör)

PIC16F84 de Register haritasında görülmemesine rağmen sık kullanılan ve **"Geçici Depolama Registeri"** de denebilecek bir registerdir. Diğer File Registerlerinden veri okur yada yazarken bu aktarma registerinden faydalanırız. Ayrıca tüm aritmetik işlemler ve bazı atama işlemleri için W registeri kullanmak şarttır. Meselâ ; iki registeri içindeki verileri toplamak istersek bunlardan birinin W de kaydedilmiş olması gerekir. (05) ile (03) ü toplamak için (05)'i örn. PORTA ya, (03)'ü de W ye yazmak gerekir.



Bu işlemlerin hangi komutlarla ve nasıl yapılacağı tablo halinde verilen Assembly komutları, özellikleri ve kullanımı ile açığa kavuşacaktır.

PIC16F84' ün BACAK YAPISI



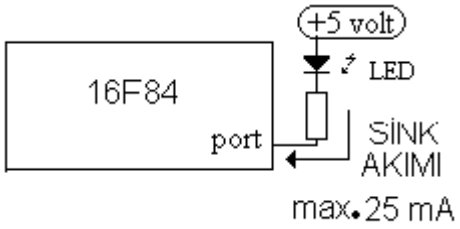
RAX ve RBX şeklinde gösterilenler dijital giriş / çıkış (I / O) (port) uçlarıdır

CMOS olarak üretilen ve Flash tipi hafızaya sahip PIC16F84 de CLK (saat) işareti kesilir ve tekrar uygulanırsa program kaldığı yerden devam eder. V_{DD} (+) , V_{SS} (-) besleme uçlarıdır. En ideal besleme +5V olup gerçekte PIC, (+2V) .... (+6V) arası çalışır. Portlardan dışarı alınan akım yeterli değilse transistor veya röle kullanılır. Bütün CMOS Lojik entegrelerde olduğu gibi burada da kullanılmayan uçlar +5V (Lojik 1) a bağlanmalıdır.

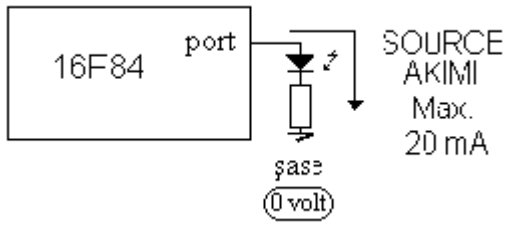
Port Çıkış İken Akımlar :

PIC16F84 de Porttan dışarı alınabilen (source) ya da Porttan içeri çekilebilen (sink) maksimum akımlar bir LED'i rahatlıkla sürebilecek seviyededir.

1) Sink Akımı : (+) 5 volt beslemeden portun içine doğru akan akımdır. Bu durumda porttan içeri **maksimum akacak akım 25 mA** dir. Daha fazla akıtılan akım PIC 'in bozulmasına sebep olabilir.

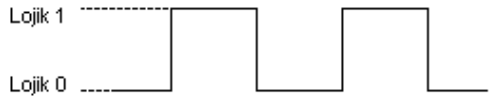


2) Source Akımı : Portun içinden toprağa akan akımdır. Bu durumda **maksimum akacak akım 20 mA** dir.



OSİLATÖR (SAAT / CLOCK) DEVRESİ

PIC 'in program hafızasında bulunan komutların çalışması ;



kare dalga (clock sinyali) ile olur.

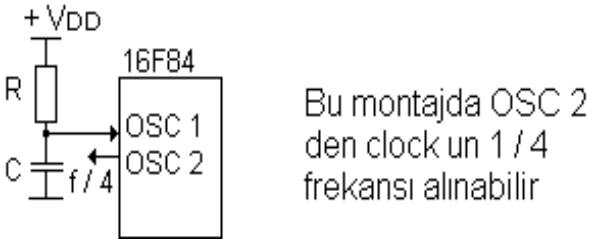
OSC1 denilen 16 nolu PIC16F84 bacağı kare dalganın uygulandığı yerdir. Bacak yapısı görünüşünde **CLK IN** olarak ifade edilmiştir. Dışarıdan buraya uygulanacak kare dalga **OSC2/CLK OUT** 'dan dörde bölünmüş olarak (f / 4)dışarı alınabilir.

PIC 16F84 de en çok kullanılan osilatörler şunlardır :

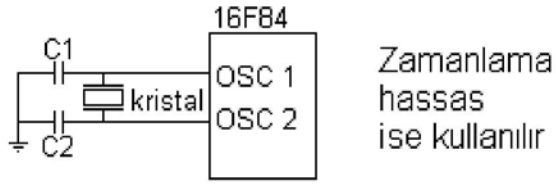
1) RC Tipi (Direnç / Kondansatör)
2) XT Tipi (Kristal veya Seramik Rezonatör)
3) HS Tipi (Yüksek Hızlı Kristal / Seramik Rezonatör)
4) LP Tipi (Düşük Frekanslı Kristal)

Osilatör tipi komut olarak veya programlama esnasında belirtilmelidir. Bu osilatör yapılarından sık kullanılan ikisinin bağlantısı aşağıda gösterilmiştir.

RC Tipi Bağlantı :

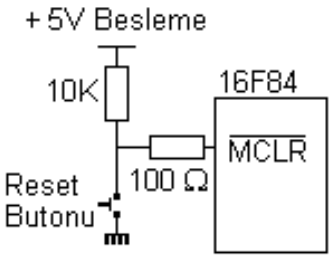


XT Tipi Bağlantı :



RESET($\overline{MCLR}$) UCU VE DEVRESİ

Program herhangi bir nedenle kilitlenirse ya da programı yeniden (baştan) çalıştırmak istersek dışardan PIC 'i reset yapmamız gerekir. Aslında PIC 'in içinde besleme gerilimi belirli bir seviyeye ulaşınca programın çalışmasını sağlayan bir **RESET devresi** (Power-on–reset) vardır. Ayrıca PIC16F84 de yine RESET fonksiyonu gören 4 nolu bacakta $\overline{MCLR}$ ucu vardır. Master (Ana) Clear (Silme - Başlatma) kelimelerinden oluşturulmuş $\overline{MCLR}$ **ucu lojik resetleme gerçekleşir.**. Yani, resetleme işlemi için bu girişe önce kısa bir süre lojik 0 verilmeli daha sonra lojik 1 uygulayarak , programın enerji verildiği andaki duruma (başa) dönüp tekrar başlaması sağlanmalıdır. Bunun için aşağıdaki gibi bir Reset devresi kullanılabilir.



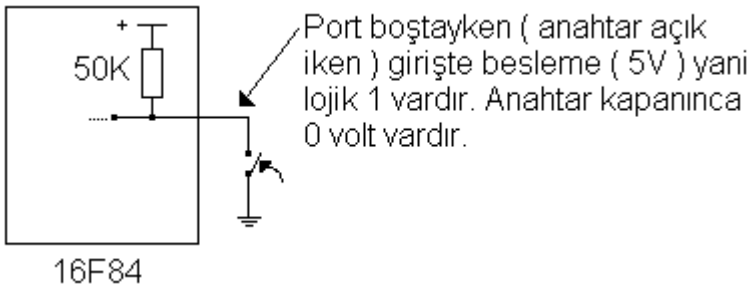
Reset bas / çek şeklinde olup
$\overline{MCLR}$ önce Ø alır , sonra
buton açılınca 10K üzerinde
lojik 1 görür

GİRİŞ / ÇIKIŞ (I / O) PORTLARI:

PIC16F84 de RA ve RB olarak 13 adet dijital(sayısal) port vardır: PIC deki bu portlardan lojik bilgi alınabilir ya da içeri lojik bilgi gönderilebilir. PIC16F84 'de

5 adet (RA0…….RA4) A portu (PORT A)
8 adet (RB0……..RB7) B portu (PORT B) vardır.

Bunlar bizim yazacağımız program ile ister giriş , ister çıkış yapılabilir. RB0….RB7 (B portunun 8 ucu birden) istenirse içerden 50 KΩ dirençle (pull-up direnci) (+) beslemeye bağlanmış gibi etki gösterir (Giriş iken geçerli).



Port boştayken (anahtar açık iken) girişte besleme (5V) yani lojik 1 vardır. Anahtar kapanınca 0 volt vardır.

Bu 8 direnç istenirse option register (opsiyon saklayıcısı) ile geçerli kılınır yada iptal edilebilir. Bunun programlamada nasıl yapılacağı ileri bölümlerin konusudur. İlk enerji verildiği anda bu dirençler devrede yoktur.

A portuna ait 4. bit (RA4) ucuna bir de TOCK1 yazılmıştır. Bu nedenle 16F84 ün harici timer / counter (zamanlayıcı / sayıcı) özelliği de vardır. Yani isterse dışarıdan gelen (uygulanan) darbenin süresi ölçülebilir veya gelen darbeler sayılabilir. Bunlar da bu dersin 2. dönemdeki kısmında incelenebilecektir.

NOT : RA4 ucu normal çıkış portu yapılırken bu uç ile (+ 5 volt) arasına bir 10KΩ luk bir direnç bağlanır. Çünkü bu port açık kollektör (open-collector) biçimindedir.

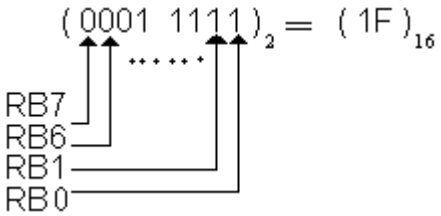
PIC ASSEMBLY DİLİ

Önce bir metin editöründe assembly dili kuralları ile yazılan komutları (hex) kodlara çevirmek (derlemek) için en basit PIC assembler programı olarak (MPASM) kullanılabileceği daha önce belirtilmişti. Assembly dili aslında yapılacak işlerin sırasıyla , komutlar halinde yazılmasından başka bir şey değildir. Komutlar ise İngilizce dilindeki karşılıklarının baş harflerinin birleşmesinden meydana gelir ve tablo halinde verilmiştir.

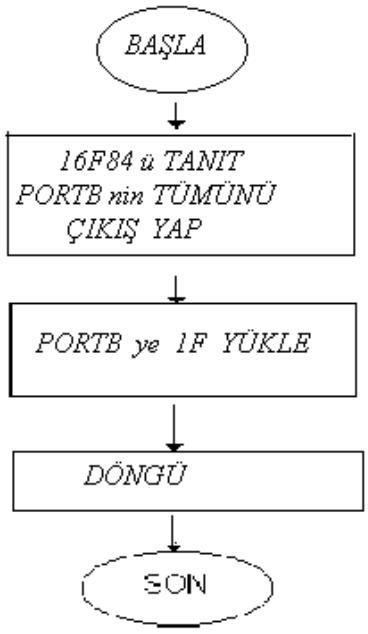
Örn : CLRF PORTB komutu ile B portunun içeriği temizlenmiş (sıfırlanmış) olmaktadır.

Program yazılmadan önce "**Akış Diyagramı**" denilen ve anlaşılırlığı arttıran hatta, gerekirse farklı mikrodenetleyicilere geçişi de kolaylaştıran bir gösterilimden faydalanılabilir.

Örnek olarak ; B portunun (RB0….RB7) uçlarına $(1F)_{16}$ yüklemek isteyelim.



Bu program için gerekli akış diyagramı çizip assembly dilinde programı yazarsak;




```
LIST P=16F84
; Adresler tanımlanıyor...
STATUS  EQU     0x03
PORTB   EQU     0X06
TRISB   EQU     0x08
; B Portu Çıkış Yapılıyor...
START
        CLRF    PORTB
        BSF     STATUS,5
        CLRF    TRISB
        BCF     STATUS,5
; Asıl Program
        MOVLW   0x1F
        MOVWF   PORTB
; Bekleme Kısmı
DONGU
        GOTO DONGU
        END
```

olur.

Yukarıdaki Akış Diyagramı ve Programın yazılışı şu an için tam olarak anlaşılmasa da bu konuda bir fikir verecektir.

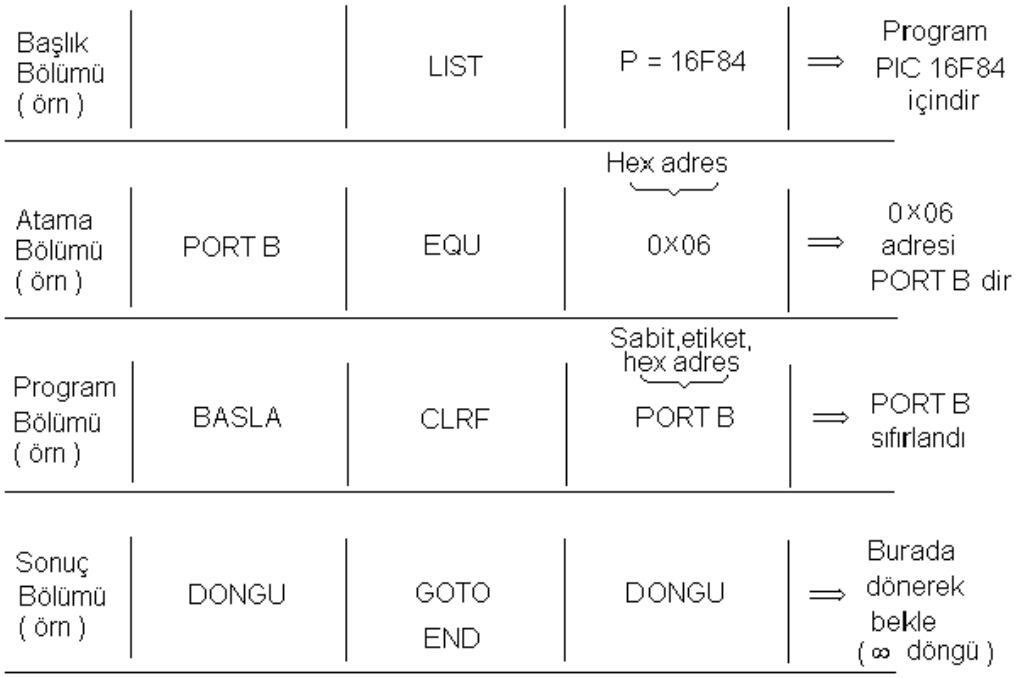
Önemli Not: Yukarıdaki Programda **Program Hafızasının nereden başlayacağını belirten ORG komutu** yer almamıştır. **Bu komut bulunmadığı durumda Program 0x000 adresinden itibaren başlanır**. Ancak Laboratuarda bulunan PIC16F877 setinde programlama yaparken ORG 0x003 yazılarak programın bu adresten başlaması sağlanmalıdır. Zira bu setin PC ile haberleşmesini ve programlanmasını sağlayan PLAB programı 0x000 adresinden başlamış ve yazmaya karşı korunmuştur.

Assembly Dilinde Program Yazarken Dikkat Edilecek hususlar:

1) Noktalı Virgül (;) : Bir satırın başına (;) konursa o satır (hex) koda dönüştürülmez. Bu satır sadece bilgi amaçlıdır. Daha sonra programı değiştirir ya da geliştirirken hatırlamak istediklerimizi (;) den sonra yazabiliriz.

2) Program Bölümleri : Assembly programının da **Başlık , Atama , Program ve Sonuç** bölümleri vardır. Editör önce sanal olarak 3 kolona bölünür. Bunlar Etiket, Komut, Adres(ya da Data)

1)Etiket 2)Komut 3)Adres (yada Data)



Eğer bu üç kolona yazarken sağa / sola kayma yapılırsa derleme sırasında MPASM 'den uyarı (warning) alınır. Oysa düzenli (3 kolona göre) yazılmış , noktalı virgüllerle (;) ile gerekli açıklamalar yapılmış programlar anlaşılır , kalıcı ve gelişmelere açık olacaktır.

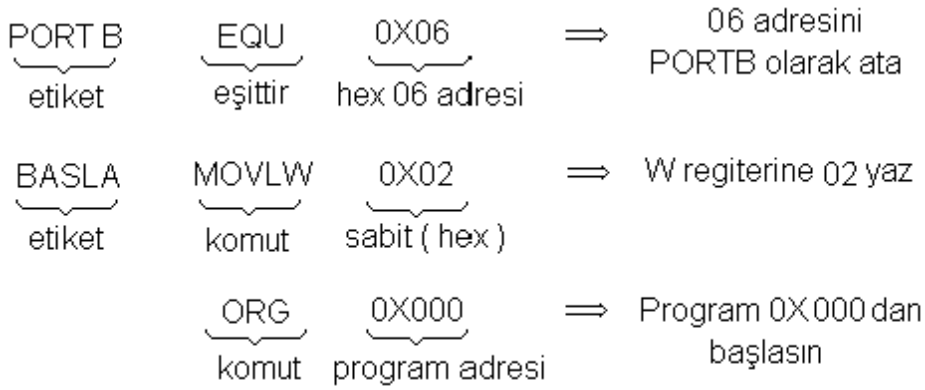
Etiketlerin Özellikleri :

Birinci kolonda yer alan Etiketler PIC hafızasındaki bir adresin atandığı ve hatırlatmayı kolaylaştıran kelimelerdir. Mesela ; 16F84 de PORT B nin 0x06 adresinde olduğu belli olduğuna göre (bknz. File Register tablosu) her sefer 0x06 yı hatırlamak ve yazmaktan kurtulmak için (başlangıçta bir kere) **PORTB EQU 0x06** komutu yazarak 0x06 yerine PORT B kullanmaya hak kazanırız. (EQU eşitleme demektir).Ayrıca tarafımızdan adres atanmayan etiketler de vardır. BASLA , DONGU gibi….

Örnek:

GOTO DONGU (DONGU ye git) komutu etiketi DONGU olan satıra (komuta) gitmeyi sağlar. Etiketleri kullanırken ; 1. kolona yazmaya , bir harfle başlamaya , Türkçe karakter kullanmamaya (DÖNGÜ değil DONGU), küçük / büyük harf duyarlılığına (DONGU yerine başka yerde dongu yazılmaz) dikkat edilmelidir.

Bazı Komut Örnekleri :



PIC ASSEMBLY KOMUTLARI

PIC 16F84 de toplam 35 komut vardır. Bu komutlar farklı şekillerde guruplandırılabilirse de biz aşağıda PIC komutlarını 4 ana grupta toplayarak inceleyeceğiz :

1) Byte Yönlendirmeli Komutlar
2) Bit Yönlendirmeli Komutlar
3) Sabitle Çalışan Komutlar
4) Kontrol Komutları

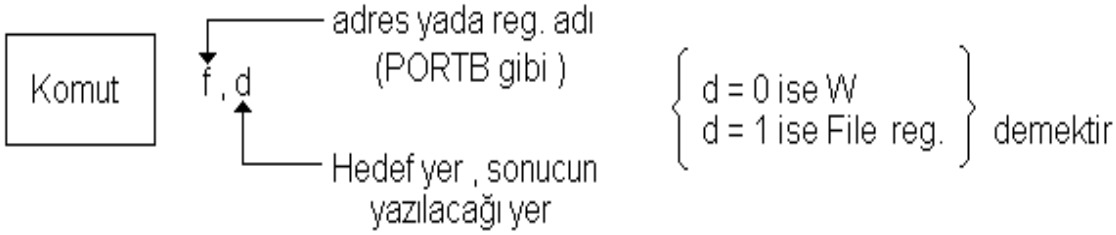
Komutlar yazılırken bazı tarif harfleri kullanılır. Bunlar ;

f : File register (RAM de , veri hafızasında bir adres)
d : Destination (Hedef gönderilen yer) *
d = 0 ise (sonuç W registerine kaydedilir)
d = 1 ise (sonuç File registerine kaydedilir)
k : Sabit veya adres etiketi
b : Bit ifade eder
b : Binary sayı ifade eder (Örn :b'00001111')
d : Desimal sayı ifade eder (Örn :d'18' gibi…) şeklindedir..

* **NOT :** MPASM tarafından **0 yerine W** ; **1 yerine F** kabul edilmektedir.

Şimdi bu komut gruplarını örneklerle açıklayalım:

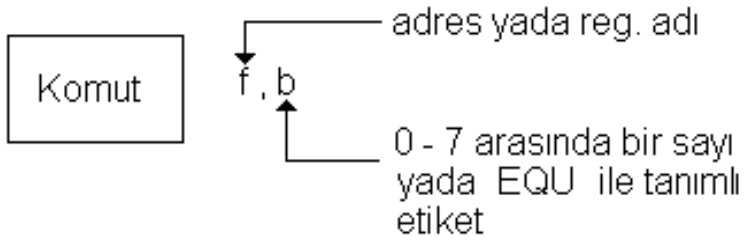
1) Byte Yönlendirmeli Komutlar :



Örnekler :
MOVF 0x03 , 0 ; 0x03 adresinin içeriğini W (Akümülatör) 'nin içine kopyala.

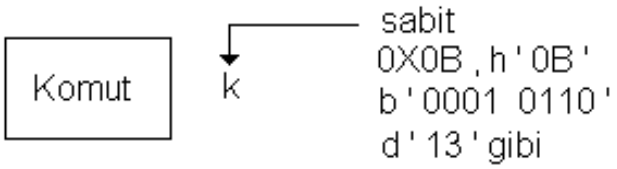
MOVF PORTA , 0 ; PORTA nın içeriğini W nin içine kopyala.
MOVF STATUS , 0 ; STATUS registerini W nin içine kopyala.

2) Bit Yönlendirmeli Komutlar :



Örnekler :
BCF 0x03 , 5 ; 0x03 adresindeki registerin 5. bitini sıfır yap.
BCF PORTB , UCBIT ; PORTB nin 3. bitini 0 yap (UCBIT, 'UCBIT EQU 3' komutu ile programda önceden tanımlı ise).

3) Sabit İşleyen Komutlar :



Örnekler :
MOVLW 0x2F ; W registerine 2F yazar (yükler).
ADDLW b'00010111' ; O anda W registerinde bulunan sayıya (1F)$_{16}$ = (0001 0111) ekler.

4) Kontrol Komutları :

Örnekler :
GOTO DONGU ; Program şartsız olarak DONGU etiketli satıra gider.
CALL GECIKME ; Program GECIKME etiketli alt programa gider. Alt program bitince RETURN veya benzeri bir komutla geri dönülür.

Sayı Ve Karakter Yazılışları

Assembly dilinde programı yazarken sayılar çeşitli formatlarda (biçimlerde) yazılabilir.

Hexadesimal sayılar için:
Atamalarda 0x03 , 3 , 03 , 03h , h ' 03 ' şekillerinden biri kullanılabilir. Ancak diğer komutlarda 0x03 yada h '03' tarzında kullanmak zorunludur.

Örnek : PORTB ye 06 adresini atamak için ;

PORTB EQU 0x06
6
06
06h
h'06' herhangi biri yazılabilir.

Ancak örn. W registerine 03 yüklemek için **MOVLW h'03'** (ya da) **MOVLW 0x03** yazılmalıdır.*

* Notlarımızdaki programlarda biz genellikle h'03' formatını tercih edeceğiz.

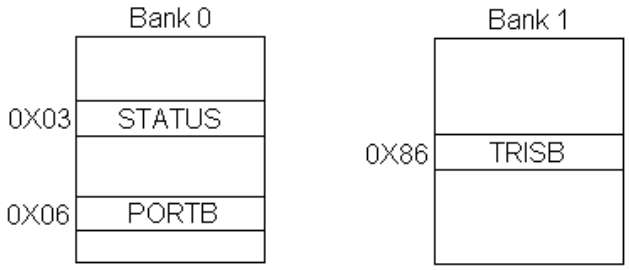
Binary (ikili) sayılar için :
b'0010 0101' şeklinde yazılır.
Örn : $(0010\ 1100)_2$ sayısını W (akümülatör) e yüklemek için ;
MOVLW b'0010 110' yazılabilir.

Desimal sayılar için :
Desimal sayılar ise başına d harfi konup yine tırnak içinde yazılır. d'18' , d'255' gibi.
Örn : $(15)_{10}$ sayısını W ye yüklemek için ;
MOVLW d'15' yazılır.

ASCII karakterler için ;
'A' yada '%' biçimde kullanılmalıdır.
Genellikle RETLW komutu ile beraber kullanılır. Doğrudan tırnak içine alınır.
Örn : RETLW 'B'
RETLW 'X' gibi

ASSEMBLY DİLİNDE İLK PROGRAM:

Önce basit bir program yazarak programın yazılma aşamaları incelenecektir.
Örnek olarak enerji verince PIC16F84 ün **B portunun 0. ve 2. bitlerini (RB0, RB2) lojik (1)**; diğerlerini de Lojik (0) yapalım. Bu durumda $(\mathbf{0000\ 0101})_2 = (\mathbf{05})_{10}$ sayısının PORTB ye yüklenmesi gerekecektir.
Elimizdeki File Register tablosunda **PORTB nin 0x06** adresinde , PORTB 'yi çıkış olarak yönlendirmek için kullanacağımız **TRISB 'nin 0x86** adresinde ve burada bank değiştirmek için kullanacağımız **STATUS registerlerinin 0x03** adresinde olduğunu bulalım. Bu durumda **programda ilgileneceğimiz** bu 3 register Bank0 ve Bank1 de



şeklinde yer alırlar.

Bank Değiştirme İşlemi :

16F84 de STATUS Registerin 5. biti (RP0) bank değiştirme için kullanılır.

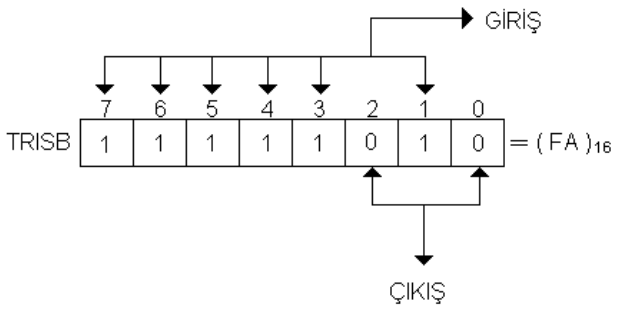
RP0 = 0 (BANK 0 seçilir)

RP1 = 1 (BANK 1 seçilir)

PIC 16F84 e enerji verildiğinde bu bit RP0 = 0 olduğu için BANK 0 seçilmiş olur. Herhangi bir registerin istenen bitini (0) yapmak için BCF , (1) yapmak için BSF komutu kullanılır.
Burada STATUS un 5. bitini 0 yapalım.
(Eğer önceden atama komutu ile STATUS EQU 0x03 tanımlaması yapılmışsa).
BCF STATUS , 5 ile RP0 = 0 yapılır.
(Eğer atama yapılmamışsa). Ancak, BCF h'03' , 5 yazarak RP0 = 0 yapılmış olacaktır.

Port Yönlendirme İşlemi :

PORTB nin **0. ve 2. bitlerini çıkış** yapacağımıza göre , **ilgilenmediğimiz diğer bitler**i (1,3,4,5,6,7) **Giriş yada Çıkış** yapabiliriz. Biz diğer bitleri **Giriş** yapalım.
PORTB , TRISB registeri ile yönlendirilir. TRISB de (1) yapılan bitler GİRİŞ , 0 yapılan bitler de ÇIKIŞ olur. Bu durumda ;

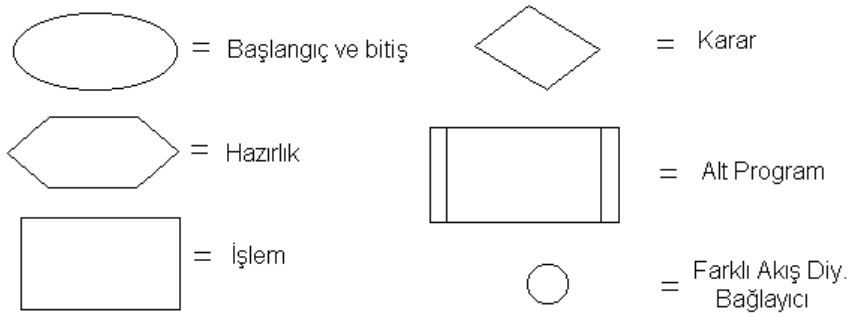


$=(FA)_{16}$ sayısı TRISB ye yüklenecektir.

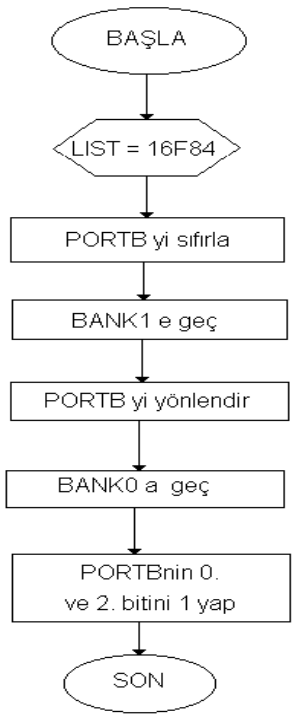
Akış Diyagramı Çizimi :

Genellikle program yazmadan önce akış diyagramı çizmekte büyük yarar vardır. Akış Diyagramları programcının ve daha sonra programı geliştirecek bir başkasının rahatça anlayabileceği tarzda olmalıdır. Asıl olan budur. Aşağıda akış diyagramı için kullanılan bazı standart semboller verilmiştir.

Kullanılan Semboller :



Çok basit programlarda akış diyagramına gerek olmasa da biz burada bir örnek olarak çizelim. Programcı programı yazarken rahatça anlayacağı şekilde akış diyagramını ayrıntılı şekilde vermelidir.
Aşağıda örnek programa ait akış diyagramı ve program birlikte gösterilmiştir.
Her akış diyagramı sembolünün yanında bir ya da daha fazla komut karşılığı yazılmıştır.




```
; ornek.asm - 01 / 11 / 2003
LIST      P = 16F84
CLRF      h'06'
BSF       h'03',5
MOVLW     h'FA'
MOVWF     h'86'
BCF       h'03',5
MOVLW     h'05'
MOVWF     h'06'
END
```

Akış diyagramı her zaman bir tek komuta karşılık gelmeyebilir. İstenirse sembollerin içine gerekli komut doğrudan yazılabilir. Bütün mesele programcının yada ilerde programı geliştirecek kişinin akış diyagramını anlayabilmesidir.

Programda kullanılan bazı komutları sırasıyla açıklayalım:

CLRF h'06' ; 0x06 adresinde bulunan PORTB nin tüm bitlerinin sıfırlanması
BSF h'03',5 ; 0x03de bulunan STATUS un 5.biti (1) yapılarak BANK1 e geçilmesi
MOVLW h'FA' ; W registerine (FA)$_{16}$ yüklenmesi
MOVWF h'86' ; TRISB ye yaz.Ve PORTB nin 0. ve 2. bitlerin Çıkış, diğerlerinin Giriş yap.
BCF h'03',5 ; 0x03de bulunan STATUS un 5.biti (0) yapılarak BANK0 a geçilmesi
MOVLW h'05' ; W ye (05)$_{16}$ yüklenmesi
MOVWF h'06' ; (05)$_{16}$ sayısının PORTB ye yazılarak 0. ve 2. bitlerin (1) yapılması

Atama Komutu (EQU) Kullanmak

EQU komutu ile hatırlanması zor olabilen özel file register adresleri başlangıçta tanımlanabilir. Bu amaçla **atama komutları** kullanılmalıdır. Buna göre yukarıda verdiğimiz **ornek.asm** programını tekrar yazalım.

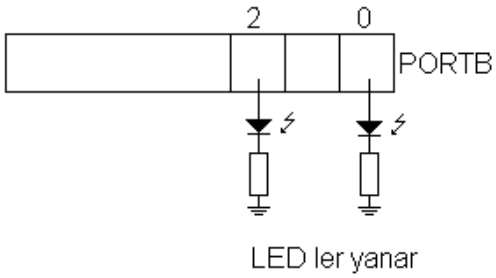
```
;ORNEK.ASM – 01 / 11 / 2003   (programı hatırlamak için ad ve tarihi yazıldı)
LIST      P = 16F84          ; Hangi PIC 'in kullanıldığı makineye bildiriliyor.
PORTB     EQU  h'06'         ; 0x06 adresi PORTB 'ye atandı
STATUS    EQU  h'03'         ; 0x03 adresi PORTB 'ye atandı
TRISB     EQU  h'86'         ; 0x86 adresi PORTB 'ye atandı
CLRF      PORTB              ; B portunu sıfırla.
BSF       STATUS,5           ; Bank1 ' e geç
MOVLW     h'FA'              ; W registerine ( FA )16  sayısını yükle
MOVWF     TRISB              ; B portunu yönlendir.TRISB h'86' olduğu EQU ile bildirildiydi
BCF       STATUS,5           ; Bank0 'a  geç
MOVLW     h'05'              ; W reg. e ( 05 )16 yükle
MOVWF     PORTB              ; B portunun 0. ve 2. bitlerini 1 yap
END                          ; SON
```

Burada komutların İngilizce karşılılarına âşina olunursa komutların ne anlama geldiği daha kolayca anlaşılacaktır. Meselâ ;

BSF STATUS,5 komutu (**B**it **S**et **F**ile (Reg) **STATUS** (bit) **5** [Status'un 5. bitini 1 yap]) demektir. Veya

MOVLW h'05' komutunun da (**M**ove **L**iteral to **W**) " **hex (05)** sayısını W ye aktar" olduğu anlaşılır.

Ve yukarıdaki örnek programın icrasından sonra sonuçta PORTB çıkışında



elde edilecektir.

MPASM İLE DERLEME :

Aslında bilgisayar ekranında **tek bir pencereden ibaret olan MPASM** denilen programda bazı ayarlar çok önemli bir rol oynamaz. Ancak özellikle programda hata varsa veya değişiklikler yapmak istendiğinde bazı dosyaların üretilmesi faydalı olacaktır. Bu nedenle ;

Hata dosyası için (.ERR) = Error File
Programın toplu özeti için (.LIST) = List File üretilmesi tercihleri MPASM de özellikle işaretlenmelidir.

Ayrıca uyarı ve hataları birlikte almak için "**Warning and Errors** " seçeneğini; Processor için kullanacağımız "16F84" tercihini seçiniz. Hex dosyası formatı için "**INHX8M**" i işaretleyiniz. Derlemek için "**Browse**" ikonu ile ornek.asm yi bulunduğu klasörde bulup "**Tamam**"a tıklamak yeterli olacaktır. Sonuçta çıkan pencerede "**Assembly Successful**" mesajı alınırsa derlemenin başarılı olduğu ve artık hex dosyasının programlayıcı kullanarak PIC 16F84 e yüklenebileceği anlaşılır.

MPASM 'nin Ürettiği Önemli Dosyalar :

Aslında MPASM ile derleme sırasında toplam 5 – 6 tane dosya üretmek mümkünse de *** . LST ve *. ERR dosyaları** programcı için önem taşımaktadır. (***** : Dosya adını ifade etmektedir.)

***.LST (List) Dosyası** : Bu dosya assembly programını yazarken kullandığımız metin editöründen (Not Defteri) açılabilir. Bu dosya içinde;

1.Sayfada ; (LOC) sütununda , komutun program hafızası yada RAM adresi
(OBJECT) sütununda , komutların hex karşılıkları
(LINE SOURCE TEXT) sütununda , *.asm (kaynak) programdaki satır numaraları
2.Sayfada ; (SYMBOL TABLE) sütununda , etiketler
(VALUE) sütununda , etiketin adresleri
(xxxx , -----) sütununda , hafızada kullanılan (x) ve boş (-) alanlar
(PROGRAM MEMORY USED) kullanılan , ve
(PROGRAM WORDS FREE) boş bunlar program hafıza alanı miktarlarıdır.

(ERRORS ve WARNING) = Hata ve Uyarı sayısı bulunur.

***.ERR (Hata) Dosyası** : Bu dosyada kullandığımız editör ile açılabilir. Programda komut yada etiket hatası varsa hatalı satır numarası bu dosyada görülür. Mesela ; ornek.asm de 8. satırda bir hata yapmış ve **BSF Status,5** yerine **BFS Status,5** yazmış olalım.

Bu durumda ;

Error [122] C:\MPASM\ORNEK.ASM 8 : Illegal opcode (STATUS)

ifadesi ile karşılaşır. Bu hatanın .LST dosyasında da hatanın yapıldığı satırdan hemen sonra belirtilmesi programcıya kolaylık sağlar.