1- Mikroişlemci Nedir?

Yapısında **CPU**, **Bellek** ve **Giriş/Çıkış birimleri** bulunan devrelere **mikroişlemci** denir.

a- CPU (Central Process Unit)

CPU'lar mikroişlemcinin hafızasındaki programları bulma, çağırma ve onları çalıştırma görevi görürler.

CPU 3 birimden oluşur bunlar **ALU**, **REGISTER** ve **KONTROL BİRİMİ** (CLOCK).

2- Mikroişlemcileri Birbirinden Ayıran Özellikler

a- Kelime Uzunluğu

Mikroişlemcinin her saat darbesinde işlem yapabileceği bit sayısına kelime uzunluğu denir, kısaca **bir kaydedicinin bit sayısıdır** diyebiliriz. İşlemciler bu süre zarfında komutları yorumlar veya bellekteki veriler üzerinde işlem yapar.

Kelime uzunluğu veri yolu uzunluğuna eşittir.

Komutlar veya veriler küçük gruplar hâlinde işlenirse hızda azalma meydana gelir.

b- Komut İşleme Hızı

İşlemci (CPU) her saat sinyalinde bir sonraki işlem basamağına geçer. İşlemcinin hızını incelerken saat frekansına ve komut çevrim sürelerine bakmak gerekir.

Saat frekansı işlemcinin içinde bulunan osilatörün frekansıdır.

Komut çevrim süresi ise herhangi bir komutun görevini tamamlayabilmesi için geçen süredir.

Bir mikroişlemcinin hızını arttıran temel unsurlar şöyle sıralanabilir:

Kesme altyordamlarının çeşitleri	Kelime uzunluğu	Bilgisayar belleğine ve giriş/çıkış birimlerine erişim hızı
İşlemci komut kümesi çeşidi	CPU tasarım teknolojisi	Zamanlama ve kontrol düzeni

c- Adresleme Kapasitesi

Bir işlemcinin adresleme kapasitesi, adresleyebileceği veya doğrudan erişebileceği **bellek alanının büyüklüğü**dür. Bu büyüklük işlemcinin adres hattı sayısına bağlıdır.

d- Farklı Adresleme Modları

Mikroişlemcinin işleyeceği bilgiyi farklı erişim şekilleri, **adresleme yöntemleri** olarak ifade edilir. Bazı adresleme türleri aşağıda sıralanmıştır.

Doğrudan adresleme	Dolaylı adresleme	İndisli adresleme	Akümülatör ve imalı adresleme
Kaydedici adresleme	Mutlak adresleme	Göreceli adresleme	Veri tanımlı adresleme

e- İlave Edilecek Devrelerle Uyumluluk

Mikroişlemcili sisteme eklenecek devrelerin ve giriş çıkış birimlerinin en azından işlemci hızında çalışması gerekir. Sisteme takılan birimlerin hızları mikroişlemciye göre düşükse mikroişlemcinin hızı diğer elemanlardaki yavaşlıktan dolayı düşer.

3- Mikroişlemciyi Oluşturan Birimler ve Görevleri

a- Kaydediciler (Register)

Hafızadaki veriler ALU tarafından işlenirken kullanılan kalıcı ve geçici alanlardır. Registerler işlemcinin çekirdeğinde olduklarından verilere ulaşmak daha hızlı gerçekleşir.

a.1- Akümülatör (ACC ya da A)

Akümülatörler, bilgisayarın aritmetik ve mantık işlemleri sırasında depo görevi yapan önemli bir **kaydedicidir**. Ayrıca sisteme gelen verinin ilk alındığı yerdir.

a.2- İndis Kaydedicileri

Hesaplamalarda ara değerlerin **geçici** tutulmasında kullanılmaktadır. Program döngülerinde ve zamanlama uygulamalarında bir **sayıcı** olarak kullanılmaktadır. Bellekte depolanmış bir dizi verinin üzerinde bir **indisçi** olarak kullanılmaktadır.

a.3- Program Sayıcı (Program Counter)

Mikroişlemcinin yürütmekte olduğu program komutlarının **adres bilgisini** tuttuğu bir **kaydedicidir**. Hangi komutun hangi sırada kullanılacağının program sayıcı (PC) bilir.

Program Counter Bit Genişliği = Adres Yolu Genişliği

a.4- Yığın İşaretçisi (SP)

Yığın, mikroişlemcinin kullandığı **geçici bellek bölgesi** olarak tanımlanır.

b- Aritmetik Mantık Birimi (Aritmetic Logic Unit) (ALU)

Kaydediciler üzerinde aritmetiksel ve mantıksal işlemin yapıldığı alandır. Yapılan işlemin sonucu **kaydediciler üzerinde saklanır**. Ayrıca **komut sayıları** ALU'nun içinde tutulmaktadır.

8 bitlik mimariye sahip bir mikroişlemcideki ALU en fazla 8 bitlik sayılar üzerinde işlem yapar. ALU iki gruba ayrılır.

b.1- Aritmetiksel İşlemler

Toplama, çıkarma, çarpma, bölme işlemleri ve ondalıklı sayılarla işlemler yapılabilmektedir.

b.2- Mantıksal İşlemler

ve (and - çarpma), veya (or - toplama), özel veya (xor), değil (not) gibi işlemler bu ünitede yapılır.

Bütün bu işlemler **kapı** ve **flip-flop**'lardan oluşan bir sistem tarafından yürütülmektedir.

c- Kontrol Birimi (Clock)

Kontrol birimi, sistemin tüm işleyişinden ve işlemin zamanında yapılmasından sorumludur. Kontrol birimi, CPU'da işlenen verilerin bellekte saklanması, bellekte program bölümünde bulunan komut kodunun alınıp getirilmesi, kodunun çözülmesi, ALU tarafından işlenmesi ve sonucun geri belleğe konulması için gerekli olan kontrol sinyalleri üretir. Özetle işlemcinin çalışmasını yönlendiren birimdir.

4- Merkezi İşlemci Biriminde İletişim Yolları

a- Veri Yolu

Merkezi işlem biriminden bellek ve giriş/çıkış birimlerine veri göndermede ya da bu birimlerden işlemciye veri aktarmada kullanılan hatlar, veri yolu olarak isimlendirilir.

8-bitlik mikroişlemcilerde veri yolu 8 hat içerirken 16- bitlik işlemcilerde 16 hattı içerir.

Veri yolunda **çift yönlü** iletişim mümkün olmaktadır.

b- Adres Yolu

Verinin alınacağı (okunacağı) veya verinin gönderileceği (yazılacağı) adres bölgesini temsil eden bilgilerin taşınmasında kullanılan hatlar, adres yolu olarak isimlendirilir.

Adres yolu, **tek yönlüdür** ve paralel iletişim sağlayacak yapıdadır.

c- Kontrol Yolu

Birimler arasındaki ilişkiyi düzenleyen sinyallerin iletilmesi amacıyla kullanılan hatlar kontrol yolu olarak adlandırılır.

Kontrol yolu **çift yönlüdür**.

Kontrol yolunda bulunan sinyaller üç farklı işlemi gerçekleştirmek için kullanılır:

Sinyal Seçimi: Sistemde kullanılacak sinyallerin ve sinyallerin uygulanacağı yerin belirlenmesi işlemini gerçekleştiren sinyallerdir.

Yön Tayini: Sistemdeki verinin ne yöne gideceğini belirleyen sinyallerdir (okuma veya yazma).

Zamanlama: Yapılacak işlemlerin sırasını ve zamanlamasını belirleyen sinyallerdir.

5- Bellek (Memory)

a- RAM Bellekler (Random Access Memory)

Mikroişlemcinin elektrik alması durumunda geçici hafıza olarak kullandığı birimdir. Elektrik kesildiği zaman bu veriler silinir ve bir daha kullanılmaz. Özel bir sıra takip etmeden herhangi bir adrese erişildiği için **rastgele erişimli bellek** olarak isimlendirilir.

b- ROM Bellekler (Read Only Memory)

Yalnız okunabilen birimlere ROM bellekler denir. Bu bellek elemanlarının en büyük özelliği enerjisi kesildiğinde içindeki bilgilerin silinmemesidir. ROM belleklere bilgiler, üretim aşamasında yüklenir, **örneğin çamaşır makinesinin içinde ROM bulunur**.

c- Programlanabilir ROM Bellek (PROM)

PROM'lar bir kez programlanabilir. Bu bellek elemanı entegre şeklindedir. Kaydedilen bilgiler enerji kesildiğinde silinmez.

d- Silinebilir Programlanabilir ROM Bellek (EPROM)

Kaydedilen bilgiler enerji kesildiğinde silinmez. EPROM belleğe yeniden yazma işlemi yapmak için "EPROM" üzerindeki bant kaldırılıp ultraviyole altında belirli bir süre tutmak gerekir.

e- Elektriksel Yolla Değiştiriliebilir ROM Bellek (EEPROM)

Üzerindeki bilgiler, elektriksel olarak yazılabilen ve silinebilen bellek elemanlarıdır. “EEPROM”u besleyen enerji kesildiğinde üzerindeki bilgiler kaybolmaz. Önemli değişkenlerin değerlerini buraya kaydederiz. **Flash belleklerde bir EEPROM türüdür.**

BOYUTA GÖRE		**HIZA GÖRE**
1- Sabit Disk, EEPROM	**KALICI**	1- Kaydedici
2- RAM	**GEÇİCİ**	2- Ön Bellek
3- Ön Bellek	**GEÇİCİ**	3- RAM
4- Kaydedici	**GEÇİCİ**	4- Sabit Disk, EEPROM

MİKROİŞLEMCİ	**MİKRODENETLEYİCİ (MİKROBİLGİSAYAR)**
Bilgisayar sisteminin kalbidir.	**Gömülü sitemlerin kalbidir.**
Güç tüketimi fazladır.	**Güç tüketimi daha azdır.**
Bellek ve giriş/çıkış portları harici olarak eklenir bu yüzden devre büyüktür.	**Dahili bellek ve giriş/çıkış portları vardır. Bu yüzden devre küçüktür.**
Aynı anda birçok işlem yapabilir.	**Aynı anda tek bir iş yapabilirler.**
	Von Neuman Tasarım Mimarisi’ne sahiptir. RISC Komut Mimarisi’ne sahiptir.

SORU-1: Kaydedici, hafıza birimleri arasında neden en hızlısıdır?

Lojik kapıları olduğu için.

1- Mikrodenetleyici Mimarileri

a- Mikrodenetleyici / Mikrobilgisayar Tasarım Yapıları

Dışarıdan gelen bir veriyi hafızasına alan, derleyen ve sonucunda çıktı elde eden bir bilgisayardır.

a.1- Von Neuman Tasarım Mimarisi

İlk bilgisayarlar Von Neuman yapısıyla geliştirilmiştir. Bu bilgisayar beş birimden oluşmaktaydı:

ALU	Kontrol Birimi	Bellek	Giriş/Çıkış Birimi	Bu birimler arasında iletişimi sağlayan yolardan oluşur.

Program komutları ve veriler **aynı bellekte** bulunduğundan gerekli olduğunda **tek bir yol** üzerinden işlemciye gönderilir, **önce komut daha sonra veri işlenir.**

Von Neuman mimarisine sahip bir bilgisayar aşağıdaki sıralı adımları gerçekleştirir:

1. Program counter'ın gösterdiği adresten (bellekten) komutu getir. **FETCH**

2. Program counter'ın içeriğini bir artır. (Program Counter'ın arttırma özelliği vardır.)

3. Getirilen komutun kodunu kontrol birimini kullanarak çöz. **DECODE**

4. İlk adıma geri dönülür.

a.2- Harvard Tasarım Mimarisi

Harvard Tasarım Mimarisinin, Von Neuman Mimarisinden farkı; veri ve komutların **ayrı ayrı belleklerde tutulmasıdır** dolayısıyla, verilere ve komutlara **farklı yollar**dan ulaşılır, bu sayede çalışması daha hızlıdır. Harvard Mimarisi, Von Neuman Mimarisi'ne göre daha **hızlıdır.**

Veri yolu ile komut yolu ayrıdır. **Veri Yolu = Kaydedicinin Bit Sayısı**

b- Pipeline (Boru Hattı) Mimarisi

Pipeline, birden fazla işin paralel olarak aynı anda yürütülmesidir. Pipeline biri işin süresini değiştirmiyor yapılan iş miktarını arttırıyor. Ayrıca komut sayısının Pipeline ile alakası yoktur, ister çok komut olsun ister az komut.

Pipeline, Harvard Mimarisi'nde daha verimlidir.

İşlemci her clock'ta tek komut gerçekleştirir. Von Neuman Mimarisi olsaydı aynı anda 2 değişken (int a ve int b) fetch yapamazdık yani getiremezdik çünkü tek bir yol var.

Avantajları	Dezavantajları
İşlemcinin işlem hacmi artar.	Pipeline olmayan bir sistemin tasarımı daha kolay ve maliyeti az olur.
ALU tasarımı hızlı fakat karmaşıktır.	Pipeline bir sistemin işlemci performansını ölçmek daha zordur.
Veri yolu ve komut yolu aynı olduğu için daha hızlıdır.	Pipeline sistemin yönerge gecikmesi, pipeline olmayana göre daha fazladır. Bu da sisteme ekstra flip-floplar eklenmesinden kaynaklanır.

Ekleme komutu ile çıkarma komutunun süreleri farklıdır. **Clock en uzun süreye göre ayarlıdır**.

Fetch Süresi + Decode Süresi + Execute Süresi = Bir Tane Clock Ayarlanmış Oldu

Komut, İşlemci Tarafından Nasıl İşlenir? (Bir Komut Nasıl Çalışır?)

1- Komutu/veriyi getirir. **FETCH (Değişken Sayısı)**
2- Kod çözülerek anlamlı hale gelmiş olur. **DECODE (Komut Sayısı)**
3- Komut neyse onu çalıştırır. **EXECUTE (İşlem Sayısı)**

Fetch, işlemci ile hafıza arasında veyahut 2 tane memory arasında gerçekleşir.

Decode, kontrol birimi tarafından gerçekleştirilir.

Execute, ALU veya çıkış portlarında gerçekleştirilir.

Örneğin tek çamaşır makinesi, tek kurutma makinesi ve tek ütü makinesi var ve yıkama işlemi 30 dk, kurutma 40 dk, ütü de 20 dk sürüyor. Sırada 4 kişi bekliyor ve birinin işi tamamen bitmeden diğeri işlerini yapmaya başlamıyor. 16:00'da başladığımızı farz edersek ancak 00:00'da 4 müşterinin de işi bitmiş olacaktı. Ancak birinci müşterinin çamaşır makinesiyle işi bittip kurutma makinesini kullanmaya başladığında, 2. müşterinin beklemesine ihtiyaç yok ve çamaşır makinesini kullanmaya başlayabilir çünkü yıkama, kurulama ve ütü işlemleri de birbirinden bağımsız işlemler.

Özetlemek gerekirse, birinci komut fetch yapıldı yani getirildi. Fetch'den sonra birinci komut decode yapıldı yani çözüldü. Birinci komut decode yapılırken ikinci komut fetch yapıldı. Decode yapıldıktan sonra komut çalıştırılır. Bu döngü devam eder.

c- Mikroişlemci Mimarileri

c.1- CISC Komut Mimarisi

CISC mimarisinin amacı, bir görevi olabildiğince az sayıda kod satırıyla tamamlamaktır.

Yavaş ve karmaşık bir donanıma sahiptir. Bu mimarinin en önemli iki özelliği;

- Değişken ve karmaşık uzunluktaki komutlar bellek tasarrufu sağlar.
- Karmaşık komutlar birden fazla komutu tek bir hale getirirler.

Bir problemi ele aldığımız zaman CISC'de RISC'e göre daha az komut yazarız çünkü komut seti fazla.

c.2- RISC Komut Mimarisi

Hızlıdır, basit donanıma sahiptir, kısa zamanda tasarlanabilir.

SORU-1: RISC, CISC'den neden daha hızlıdır?	**Komutların devreleri basit olduğu için.**
Mikrodenetleyiciler RISC'i, bilgisayarlarımız CISC'i kullanır.	

CISC Mimarisinin Avantajları	CISC Mimarisinin Dezavantajları
Sistemdeki kontrol biriminden daha ucuzdur.	Komut kodu daha karmaşık bir hale gelmiştir.
Derleyici karmaşık olmak zorunda değildir.	Farklı komutlar farklı çevrim sürelerinde çalıştıkları için makinenın performansını düşürecektir.
	Komutlar işlenirken bayrak (durum) bitlerinin dikkat edilmesi gerekir. Bu da ek zaman süresi demektir.

2- İnterfacing (Arayüzü)

Mikroişlemci arabirimi, iki veya daha fazla cihazı bağlama işlemidir. Örnek; mikroişlemci ile tuştakımı.

1- Mikroişlemciler ve Mikrodenetleyiciler

Mikroişlemci, aritmetik işlemleri yapabilen, bağlandığı aletlerle iletişim kurma kabiliyetine sahip, mikro boyutlardaki bilgisayarların kontrol ünitesidir, ayrıca **önbelleğe** sahiptirler.

Mikroişlemci ile kontrol edilebilecek bir sistemi kurmak için en azından şu birimler bulunmalıdır; **CPU, RAM, Giriş/Çıkış birimi** ve bu birimler arasında veri/adres alışverişini sağlamak için **bilgi iletim yolları (DATA BUS)** gerekmektedir.

Mikrodenetleyici (sayısal bilgisayar) üç temel kısımdan **(CPU, Giriş/Çıkış birimi** ve **Hafıza)** oluşur. Mikrodenetleyicide önbellek yoktur. Mikrodenetleyiciyi, kendi işlemcisi, ana kartı, belleği, portları bulunan bir bilgisayar gibi düşünebiliriz dolayısıyla neredeyse yalnız başına çalışabilme kabiliyetine sahiptir. Mikrodenetleyiciler kontrol için kullanılır.

PIC mikrodenetleyicileri en yaygın mikrodenetleyici olarak karşımıza çıkmaktadır. Bu mikrodenetleyicilerin en fazla kullanılmasının nedeni **Flash Hafızaya** sahip olmasıdır. Flash Hafıza teknolojisi ile üretilen belleğe yüklenen program, **enerji kesilse bile silinemez**. Yine bu tip bir belleğe istenilirse **yeniden yazılabilir**.

Osilatör: Mikroişlemcinin düzgün çalışabilmesi için gereklidir. Görevi, saat darbeleri üretmektir.

2- Basitten Karmaşığa Mikroişlemci Yapısı

a- 8-Bitlik Mikroişlemciler

Kaydediciler: Flip-Floplardan oluşan birimlerdir. İşlemci içerisinde olduklarından belleklere göre daha hızlı çalışırlar.

Zamanlama ve Denetim Birimi: Bu birim fetch, decode ve execute için gerekli olan denetim sinyalleri üretir. Özetle sitemin tüm işleyişinden ve işlemin zamanında yapılmasından sorumlu olan birimdir.

Adres Yolu: Komut veya verinin bellekte bulunduğu adresten alınıp getirilmesi veya adres bilgisinin saklandığı yoldur.

Veri Yolu: İşlemciden belleğe veya Giriş/Çıkış birimlerine veri yollamada ya da tersi işlemlerde kullanılır.

Kontrol Yolu: Sisteme bağlı birimlerin denetlenmesini sağlayan özel sinyallerin oluşturduğu bir yapıya sahiptir.

3- Mikrodenetleyici

a- Mikrodenetleyici İşletimi

Bir PIC mikrodenetleyicisinde bir komutun işlenme süreci 4 aşamada gerçekleştirilir.

1. FETCH: Komutu/veriyi getirir.

2. Decode: Kod çözülerek anlamlı hale gelmiş olur.

3. Execute: Komut neyse onu çalıştırır.

4. İşlemi tamamlama (Comlete Process): Bazı komutlarda işlem sonucunu W ya da file register'a yazma süreci olarak düşünülmüştür, bazı komutlarda ise bu süreç içerisinde işlem yapılmaz.

b- Mikrodenetleyici Yapısı

Mikrodenetleyicilerin yapısı **Von Neumann Mimarisi** olarak adlandırılır.

PIC, **Harvard Mimarisine** sahiptir ve veri belleği ile program belleği vardır. Data belleği bilgi yoluna (BUS), program belleği ise program yoluna (BUS) sahiptir. İç yapısı basittir ve işletimi hızlıdır.

c- PIC Mikrodenetleyici Çeşitleri

Bir CPU ya da MCU'nun dahili veri yolu uzunluğuna **kelime boyu** denir.

d- Mikrodenetleyici Seçimi

Burada mikrodenetleyicinin belleğinin yazılım için yeterli büyüklükte olması, kullanılması düşünülen **ADC** (Analog Dijital Dönüştürücü) kanalı, **port sayısı**, **zamanlayıcı sayısı** ve **PWM** (Pulse Widht Modulation- Darbe Genişlik Modülasyonu) kanalı sayısı önemlidir. Seçimi etkileyen bu noktaları kısaca belirtedecek olursak;

- **Mikrodenetleyicinin İşlem Gücü**
- **Belleğin Kapasitesi ve Tipi**
- **Giriş/Çıkış Uçları**
- **Özel Donanımlar**
- **Kod Koruması** (mikrodenetleyicinin sahip olduğu kod koruması özellikle ticari uygulamalarda program kodunun korunmasına olanak sağlamaktadır).

NOT: Okuma yazma işlemi, işlemci ile memory arasındadır.

********** NOT **********

int a = 10;
int b = 20, c, d;
c = a + b;
 	
İlk ikisi komut değil. Sonuncusu komut.
	.exe dosyası oluştu.
	Bu program RAM'e yüklenmeli yani **yönlendirme** **(Kodların RAM'de yerlerinin ayrılıp RAM'e yüklenmesine denir.)** yapılmalı.
	İşlemci komutları analiz eder, kontrol birimi komutu aldı ve çözdü.
	İşlemci analize ilk komutlardan başlar, değişkenlerden değil.
	ALU işlemleri önce işlemciye kaydeder sonra adrese yazılır.
	D, memory'de yani yönlendirmede var ama programda çağırmadık yani kullanmadık.

c = a+b; işleminde a+b işlemi yapıldıktan sonra bu işlem register'a kaydedilir daha sonra adrese gönderilir. İşlemci c = a+b; işlemini yaparken a'nın ve b'nin adresini RAM'e gönderir. a'nın daha sonra b'nin değeri RAM'deki adres bölgesinden alındıktan sonra işlem yapılır.	

Bir Komutun ya da Bir Değişkenin İşlemci Tarafından Alınması İçin;

1- Program Counter, komutun adresini adres yoluna yükler, ardından bir sonraki komutun adresini tutar. **(Program Counter sadece komutların adreslerini tutar, değişkenlerin değil.)**
2- Kontrol hattıyla memory'ye sinyal gönderilir, adres doğru geldi mi diye, memory de evet geldi diye sinyal gönderir.
3- İşlemci okuma mı yazma mı olduğuna karar verir. (**R (1)** ya da **W (0)** gönderir bunun için tek hat yeterlidir)
4- Memory R komutunu görür görmez o adreste ne varsa veri yoluna gönderir. Veri yoluna yükledikten sonra memory işlemciye ulaştı mı diye (sinyal) gönderir.

Yazma İşleminde; işlemci tarafından adres (adres yolu üzerindedir), adres yoluna aktarılır, işlemci veri yoluna işlemin sonucunu yükler ve gönderir.

- Atama işlemi (=) varsa adres ataması var demektir ve atama işlemini, int main (){ } içinde yazabiliriz.

- İlk komutu almak için memory'ye adres göndermeliyiz, memory'ye göndermesi içinde adres yoluna.

- Komutu, işlemci içindeki kontrol birimi çözer. Komutları çözdükten sonra ALU'ya göndeririz. Değişkenlerin adresi **data adress register**'da tutulur.

Program Counter, komutun adresini tutar, değişkenin adresine karışmaz sebebi ise; 1- Sırası karışmasın diye. 2- Değişken adresinin arttırılmasına gerek yok.

- İşlemcinin içinde değişkenler vardır bunlar ALU'ya gönderilip sonuç alınabilir, ALU sonucu kaydediciye kaydeder.

- İşlemci, işlemin sonucunu veri yoluna gönderir bu adrese veri yazılır.

- Okuma veya yazma işleminde kesinlikle adres gereklidir.

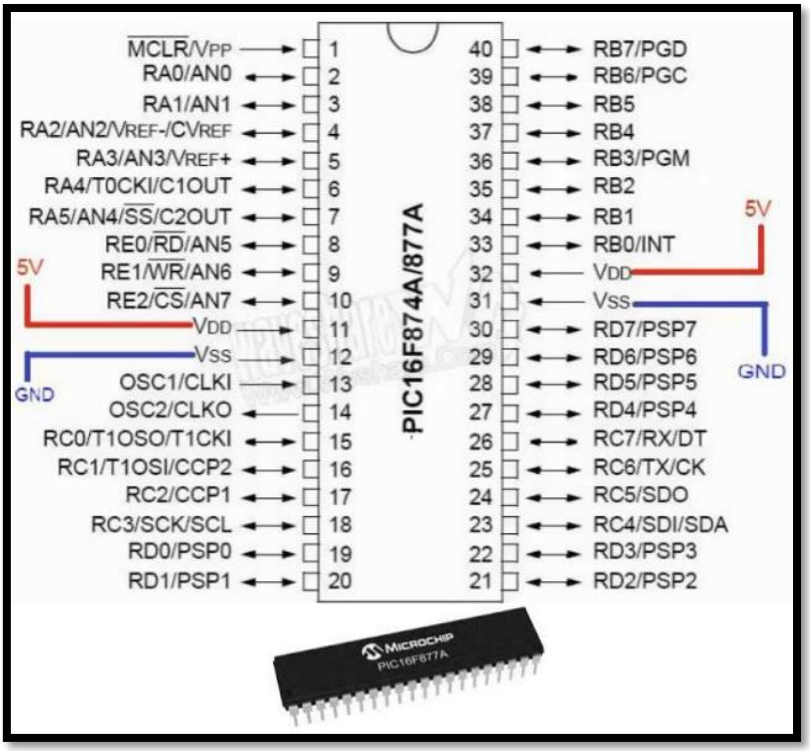
- Mikrodenetleyicide 4 çeşit hafıza birimi vardır;

EEPROM	Ana Bellek (RAM)	Kaydediciler (İşlemcinin içinde yer alırlar.)	Program Memory

- Mikroişlemcilerin içinde neler vardır?

Hafıza Birimi	Clock (Hız)	Kaydedici	Decoder	Kontrol Birimi	ALU	Yollar

1- PIC Mikrodenetleyici



PIC Mikrodenetleyici yapılarına göre Giriş / Çıkış Port sayıları bulunmaktadır. Bu portlara baktığımızda her portun yanında birden fazla görev yazılmaktadır. Örneğin 24 ve 25. portlar hem giriş dijital, çıkış dijital hem de haberleşme için kullanılabilir.

2- MikroC Yazılımı

a- En Temel MicroC Fonksiyonu

Ana Fonksiyon	**Sonsuz Döngü**	**Gecikmeler**
void main() { }	while(1){ } for(;;) { }	Komutlar arası gecikmeler: delay_ms(DEĞER);

b- MicroC Komutları

TRIS Komutu:

Portları giriş ya da çıkış olarak ayarlamak için kullanılır. Tris komutu hangi port için kullanılacaksa o portun harfi TRIS kelimesinden sonra getirilir. Örnek A portunu giriş ya da çıkış olarak kullanmak için TRISA, B portu için TRISB komutu kullanılır. Burada TRIS komutunun alabileceği iki değer vardır. Bunlar 0 ve 1'dir. TRIS komutu 0 ise port çıkış, 1 ise port giriş olarak kullanılır.

TRIS**X** = **Değer**; **X** = Mikrodenetleyicideki portun adı (A, B, C, D, E) X en fazla kelime uzunluğunun değeri kadar olabilir. **Değer** = 0 (Çıkış-Okuma) ya da 1 (Giriş-Yazma) **Değer;** Binary, Hex, onluk sistemde olabilir. Kaç port varsa o kadar TRIS vardır.	**Portun tüm pinleri için;** TRISB = 0b00000000; Port B'nin **bütün pinlerini** binary olarak çıkış ayarla. TRISB = 0x00; (hexadecimal olarak) TRISB = 0; (decimal olarak) **Portun belli bir pini için;** TRISB.f3 = 1; Port B'nin 3. pinini giriş olarak ayarla.

Port Komutu: (0 = 0 VOLT 1 = 5 VOLT)

PORT'ları yazmadan önce TRIS'leri yazmalıyız. PORT'larla voltaj değerini göndeririz. Port komutu değer atamak ya da değer okumak için kullanılır. Port komutu da TRIS komutu gibi hangi port için kullanılıyorsa o portun harfini sonuna alır. PortA komutu A portu bacaklarındaki dijital bilgileri ifade eder. Port komutu kullanıldığında port komutundan sonra kullanılan = işaretinin sağındaki değer istenilen porta gönderilir.

PortX = Değer; X = Port adı (A, B, C, D, E) Değer = 0 (Çıkış) ya da 1 (Giriş)	**Portun bütün pinleri için;** PORTB = 0b00000000; Port B'nin bütün pinlerine 0 volt gönder. PORTB = 0b11111111; Port B'nin bütün pinlerine 5 volt gönder. **Portun belli bir pini için;** PORT.fN = Değer 0 veya 1; N = Porta pin numarası PORTB.f3 = 1; Port B'nin 3. Pinine 5 volt gönder

Gecikme Komutu:

Fonksiyon ismi ← delay_ms(1000); → Parametre

c- **Proteus Programının Kullanımı**

c.1- PIC Mikrodenetleyicinin Çalışma Hızı (Clock)

PIC mikrodenetleyiciye verilen fonksiyonların doğru bir şekilde çalışması için PIC mikrodenetleyicinin çalışma hızının ayarlanması gerekmektedir. Burada iki hızın da aynı olmasına dikkat edilmelidir aksi takdirde mikrodenetleyiciye verilen fonksiyonlar doğru bir şekilde çalışamaz.

1- MikroC Hakkında

a- Açıklama Satırları (Comments)

// MikroC ile motor kontrol	/* Bilgisayar Bölümü MikroC ile motor kontrol */

b- Karakter ve String İfadeler

Karakterler tek tırnak arasına yazılmaktadır, örneğin; 'b'. Stringleri ise çift tırnak arasına yazılmaktadır, örneğin; "Bilgisayar Bölümü" .

c- Do-While Döngüsü

While döngüsü ile aynı şekilde kullanıma sahiptir ancak bir farkı vardır o da şart sağlansa da sağlanmasada döngü 1 kez (program ilk çalıştığında) çalışır.

d- return Komutu

return deyimi fonksiyondan geriye değer döndürmek amacıyla kullanılmaktadır.

e- goto Komutu

goto komutu, program akışının herhangi bir şarta bağlı olmaksızın yönlendirilmesi amacıyla kullanılmaktadır. Program akışının goto deyiminin bulunduğu noktadan daha önceki bir konuma yönlendirilmesi **Geri Sapma**, daha sonraki bir konuma yönlendirilmesi de **İleri Sapma** olarak adlandırılmaktadır.

********** NOT **********

- RB7/PGD ---> Ana Özellik Yan Özellik

- Portlar aynı zamanda kaydedicidir.

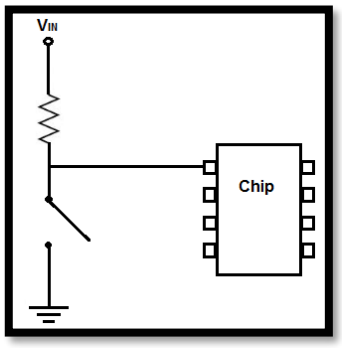
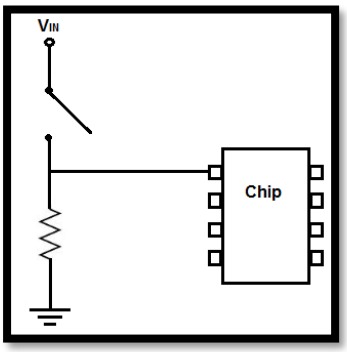
- Herhangi bir portta bit sayısı mikroişlemcinin kelime uzunluğunu aşmaması lazım.

define	**const**
Adres yok. (Hafızada yer kaplamaz.) main içinde yazılmaz.	Adres var. (Hafızada yer kaplar.) main içinde yazılabilir.

- Değişkenler (int, char gibi) **port ayarlarından** (TRIS, PORT) **önce initializes** edilmelidir bunun sebebi; memory organizasyonunu derleyici yapamadığı için int, char gibi değerler en başta tanımlanır.

- Sabit bir şeyler olacaksa direk void main'in içine yazabiliriz. Yani sürekli çalışacaksa direk void main fonksiyonunun içine yazabiliriz bunun için sonsuz döngü kurmamıza gerek yok.

int a = 10;
int b = 20;
int c = a+b;
3 adet fetch var.
1 adet execute var.
	int c = a+b+10;
	2 adet execute var.

Pull Up:	**Pull Down:**
Butona basıldıktan sonra +5 Volt, daha az direnç bulunan yolu tercih edeceği için toprağa gidecektir ve entegre üzerine 0 volt seviyesinde bir gerilim düşecektir. 	Düğmeye basıldığında gerilim kaynağıyla PIC girişi kısa devre olur. Buton ile +5 Volt verdiğimizi varsayarsak lojik sistemin diğer ucunda yüksek değerde bir pull down direnci bulunduğu için dışarıdan verilen +5 Volt lojik sistemimize uygulanacaktır. 

Çıkış = kontrol ederiz yani PIC'den dışarı bilgi akıyorsa TRIS 0'ı kullanırız. Örneğin; PIC'den bir bilgi gidiyor böylelikle led yanıyor yine aynı şekilde PIC'den bir bilgi geliyor bu sayede motor hareket ediyor.	
B kaydedicinin (portunun) tüm pınlerini çıkış yap.	(TRIS B = 0;)
B portunun ilk 4 bitini çıkış yap.	TRISB.F0 = 0; TRISB. F1 = 0; TRISB. F2 = 0; TRISB. F3 = 0; PORTB. F0 = 0; PORTB. F1 = 0; PORTB. F2 = 0; PORTB. F3= 0;

Giriş = değer (veri) okumadır yani dışardaki elemandan PIC'e bilgi gönderiliyorsa TRIS 1'dir. Örneğin sensör, buton gibi eleman bağlanabilir.	
B kaydedicinin (portunun) tüm pınlerini giriş yap.	(TRIS B = 1;)
B portunun ilk 4 bitini giriş yap.	TRISB.F0 = 1; TRISB. F1 = 1; TRISB. F2 = 1; TRISB. F3 = 1; PORTB. F0 = 1; PORTB. F1 = 1; PORTB. F2 = 1; PORTB. F3= 1;

1- LCD

a- LCD Pinleri ve Bağlantı Şeması

sbit LCD_RS at RB4_bit; **sbit LCD_EN at RB5_bit;** **sbit LCD_D7 at RB3_bit;** **sbit LCD_D6 at RB2_bit;** **sbit LCD_D5 at RB1_bit;** **sbit LCD_D4 at RB0_bit;** **sbit LCD_RS_Direction at TRISB0_bit;** **sbit LCD_EN_Direction at TRISB1_bit;** **sbit LCD_D7_Direction at TRISB5_bit;** **sbit LCD_D6_Direction at TRISB4_bit;** **sbit LCD_D5_Direction at TRISB3_bit;** **sbit LCD_D4_Direction at TRISB2_bit;**	**1- Bu ayarlar void main() 'in üstünde yapılır.** **2- Bu pin ayarları her zaman aynıdır sadece portlar ve pinler değişebilir bunlarda kullanıcının isteğine bağlıdır.** **3- Soldaki ilk 6 satıra bakacak olursak RB4, RB0 gibi ifadelerden anlayacağımız üzere LCD'nin bacakları B portuna bağlanılmış.** **4- Altındaki son 6 satıra bakacak olursak bunlarda B portunun hangi pinlerine bağlı olduğunu gösterir bize.**

b- LCD Fonksiyonları

Lcd_Init();	LCD kütüphanesini tanımlamış olduk yani çağırdık.
LCD_OUT():	Bu komut LCD ekrana, göndermek istediğimiz bilgiyi yollar. Lcd8_Out(1, 3, "HOSGELDIN"); Dersek 1. satır ve 3. haneden itibaren HOSGELDIN yazısını göstermiş oluruz.
Lcd_Chr():	İstediğimiz satır ve sütuna tek bir karakter yazdırmak içindir. Lcd8_Chr(2, 3, 'X'); Komutu 2.satır 3.haneye bir X harfi LCD data hafızasına gönderilecektir.
LCD_CURSOR_OFF():	İşaretçiyi (imlecin yanıp sönmesini) kapatır.
LCD_SHIFT_LEFT:	LCD ekranındaki yazıları sola doğru kaydırır.
LCD_SHIFT_RIGHT:	LCD ekranındaki yazıları sağa doğru kaydırır.
LCD_CLEAR:	LCD ekranını temizler.
FloatToStr(sayi, gelen) **IntToStr(…)**	Bilindiği üzere LCD ekrana sadece string veri tipinde veri girişi sağlanabilir. Float veri tipini string veri tipine dönüştürür. !! void main() fonksiyonunun içinde sayi adında float bir verimizin olduğunu ve gelen adında da 15 karakterlik bir char veri tipimizin olduğunu düşünelim, çalışma mantığı bu şekildedir.

Kullanıcı Tanımlı Fonksiyonlar

Kullanıcı tanımlı fonksiyonlar, programı yazan kişi tarafından oluşturulmaktadır. Kullanıcı tanımlı fonksiyonlar 3'e ayrılır:

1- Geriye değer döndürmeyen ve parametre almayan fonksiyonlar.
2- Geriye değer döndürmeyen ve parametre alan fonksiyonlar.
 	
3- Geriye değer döndüren ve parametre alan fonksiyonlar.
	1- **void** ledYak () {
	KOMUTLAR}
	2- **void** ledYak (int X) {
	KOMUTLAR}
	ledYak(5)
	3- **int** ledYak (int X) {
	KOMUTLAR
	return ledYak;}

2- Analog Dijital Çevirici (Analog Dijital Converter) (ADC)

Analog sinyaller (sıcaklık, basınç, nem, ses, ışık gibi fiziksel değerler) dijital sistemler tarafından direkt ölçülememektedir. Analog sinyal dediğimiz bu sinyaller aslında elektrik akımlarıdır. İşte **ADC modülünün yaptığı iş** bu sinyalin voltaj büyüklüğünü ölçmek ve mikrodenetleyicide işlenebilecek dijital sinyale dönüştürmektir.

ADCON0 KAYDEDİCİSİ

7	6	5	4	3	2	1	0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO-DONE	-	ADON

ADON: Analog/Dijital çeviriciyi yetkilendirme bitidir. ADON=1 ise Analog/Dijital çevirici açıktır ve işlem yapılabilir. ADON=0 ise Analog/Dijital çevirici kapalıdır.
GO/DONE: Eğer ADON biti de 1 ise A/D çevirici statü biti görevini üstlenir. GO/DONE=1 ise A/D çevirici işlem yapıyor demektir. GO/DONE=0 ise A/D çevirici üzerinde herhangi bir işlem yapılmıyordur.
CHS0:CHS2 Analog Kanal Seçme Bitleri: A/D çevirici için kanal seçim bitlerini oluşturur. Bu bitlere verilecek değerlerle A/D çevirme için hangi kanalın seçileceği belirlenir. PIC16F877'de A/D çevirici için 8 kanal mevcuttur.
000 = Kanal 0, (RA0/AN0)
001 = Kanal 1, (RA1/AN1)
010 = Kanal 2, (RA2/AN2)
011 = Kanal 3, (RA3/AN3)
100 = Kanal 4, (RA5/AN4)
101 = Kanal 5, (RE0/AN5)
110 = Kanal 6, (RE1/AN6)
111 = Kanal 7, (RE2/AN7)
ADCS0:ADCS1: A/D çevirici için clock frekansı seçim bitleridir. Bu bitlere verilecek değerler ile A/D çevirme işlemi esnasında kullanılacak frekans değeri bize sunulan değerler içerisinden seçilir.
00 = FOSC/2
01 = FOSC/8
10 = FOSC/32
11 = FRC (Harici bir RC osilasyon kaynağından gelen clock darbeleri kullanılır.)

ADCON1 Kaydedicisi

7	6	5	4	3	2	1	0
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
0	0	0	0	1	0	0	1

PCFG0 : PCFG3: A/D çevirici portunun biçimini düzenlemeyi sağlayan bitlerdir. Yani A/D çevirme işleminde kullanılacak pinlerin nasıl davranacağını belirlememize olanak sağlarlar.
ADFM: A/D çevirme işlemi esnasında meydana gelen verinin biçimini belirlemeye yarayan bittir. ADFM=1 ise ADRESH kaydedicisinin MSB kısmındaki altı biti 0 kabul edilir ve A/D çevirme sonucunda elde edilen veri ADRESH'ın 2 bitlik LSB kısmına ve ADRESL'ye yazılır. ADFM=0 ise ADRESL'nin LSB kısmındaki 6 biti 0 kabul edilir ve A/D çevirme sonucu elde edilen veri ADRESL'nin son iki bitine ve ADRESH'a yazılır.
ADRESH ve ADRESL kaydedicileri A/D dönüşümün 10 bit sonucunu kapsar. A/D dönüşümü bittiği zaman, sonuç A/D sonuç kaydedicisinin içine yüklenir.

Sonuçta bir A/D çevrimi için yapmanız gerekenleri şu şekilde sıralayabiliriz.

1- Hangi kanalları A/D çevrimine açacağınızı ve çevrim sonucunu nasıl yazdırmak istediğinizi göz önünde bulundurarak ADCON1 yazmacının (kaydedicisinin) değerini ayarlayın.
2- Çevrimi açmak ve bu çevrimi hangi kanaldan (A/D çevrimini birkaç kanaldan aynı anda yapamazsınız) yapacağınızı ve çevrim periyodunu ayarlamak için ADCON0 yazmacına gerekli değerleri ayarlayın.
3- ADCON0'ın Go/Done bitini 1 yaparak çevirme işlemini başlatın.
4- Go/Done bitinin 0 değerini almasını bekleyin. Bu değerin 0 olması işlemin bittiğinin, sonucunda ADRESH ve ADRESL'ye yazıldığını belirtir.
5- Sonuç yazılma ayarlarınızı göz önünde bulundurarak sonucu değerlendirebilirsiniz.

FİNAL ÖNCESİ QUİZ

ADCON1 = 0b1000xxxx
0000
0010
0100
1110

ADCON0 = 0b0000xxxx
0000
0010
0100
1110

ADFM 8 bitten büyük olursa ADCON1, 8 bitten küçük olursa ADCON0 kullanırız. (Biz genelde ADCON1 kullanırız.)

PCFG3: PCFG0	AN7(1) RE2	AN6(1) RE1	AN5(1) RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VREF +	VREF-	KANAL/REFS
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0

NOT: Okunan değer mutlaka bir değişkene aktarılıyor. Değişken boyutu 16 bitten az olmayacaktır.

Analog Sinyal

1. Kesintisiz ve süreklidir.
2. Ses dalgaları 20 Hz ve 20 kHz arasında değişir.
3. Yönü ve şiddeti zamanla değişen sinyallerdir.

ADC birimlerinin en önemli özellikleri **çözünürlükleri yani bit sayılarıdır.** 10 bitlik bir ADC, 0 ile 1023 arasında değerler üretebilirken, 8 bitlik bir mikrodenetleyici 0 ile 255 arasında değerler alabilmektedir. ADC biriminin bit sayısı arttıkça çözünürlüğü de artmaktadır. Diyelim ki elimizde çıkış genliği 0V ile 5V arasında değişen bir basınç sensörümüz var ve mikrodenetleyicimiz içerisindeki ADC birimi 10 bitlik. O halde 5V/1023 = 0.00488 gibi bir değer elde ederiz. Bu değer bizim ADC birimimizin hassasiyetini temsil etmektedir. Yani ADC birimimiz 4.8mV değerindeki voltaj değişimlerini hissedebilmektedir. ADC'den okunan değer sadece o anda okunan değerin dijital karşılığıdır.

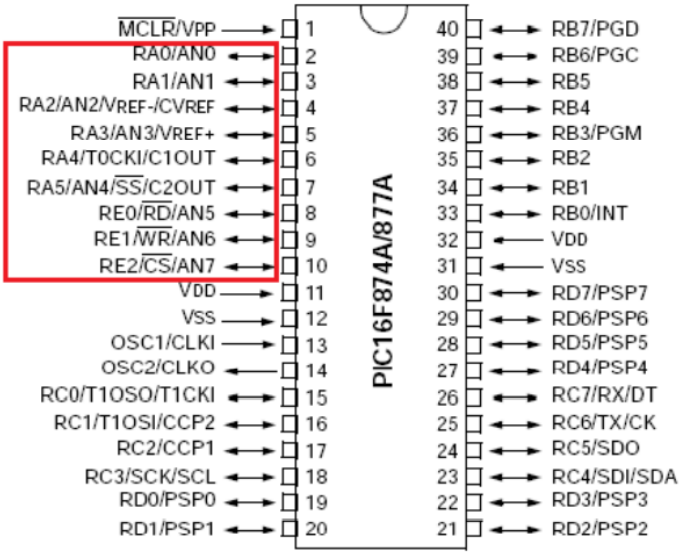
PIC16F877'de **8 kanallı** ADC modülü bulunmaktadır, bu modüllerden **10 bitlik giriş** alınabilir. **Bu kanallar PORTA 'da ve PORTE 'de bulunur**. Bu kanallardan herhangi birine gelen bir analog sinyal 2'lik sayı sistemine dönüştürülür. Dijital bilgiye dönüştürülen analog sinyalin (AN portlarından gelen bilgi) karşılığı **ADRESH** ve **ADRESL kaydedicilerine (yazmaç) yazılır.**

Analog portundan (AN portları) gelen değer 255 'ten büyük olduğunda değer nasıl kaydedilir?
Gelen değer **255'ten küçük ise ADRESL kaydedicisinde** değer kaydedilir ve **ADFM bitini 0** yaparız böylelikle sonuçlar sol tarafa yaslı olarak ayarlanır. (**ADFM** biti en baştaki bittir.)
Gelen değer **255'ten büyük ise** 255'in üstündeki değer **ADRESH kaydedicisinde** kaydedilir ve **ADFM bitini 1** yaparız yani ADRESL ilk 8 biti, ADRESH diğer bitleri kaydeder.

ADC modülünün işlemlerinde 4 önemli kaydedici (yazmaç) vardır. Bunlar **ADCON0**, **ADCON1**, **ADRESH** ve **ADRESL**'dir**.**

- ADCON'lar Analog/Dijital çevriminin kontrolünü yönetirler.
- ADRES'ler ise sonuçların yazıldığı yazmaçlardır. Burada bir noktayı belirtmemiz yararlı olacaktır, ADCON0 ve ADRESH yazmaçları Bank0'da bulunurken, ADCON1 ve ADRESL yazmaçları Bank1'de bulunur.

ADC_init() Dışarıdan değer alacaksak kullanırız.	**ADC_read()** Dışarıdan gelen değeri okur. Örneğin; volt = ADC_read(0) yazdığımızda Analog 0 girişini yani 0. kanaldaki bilgiyi oku volt değişkenine 10 bitlik dijital bilgi olarak yaz.



********** NOT **********

Yüksek Seviyede Yazılan Programın İşlemci Tarafından İşlenmesi İçin Hangi Adımlar Gerçekleşmelidir? (Program Nasıl Çalışır?)

Öncelikle mikroişlemcinin hangi mimariye sahip olduğunu bilmemiz gerekir.

1- ASM diline dönüştürülür. (Birinci sebep; tek komutta tek işlem yapıldığı için ikinci sebep; her komutta bir iş.)
2- Lincar (Bağlama) işlemi yapılır. (Eğer main'in üzerinde kütüphane varsa Lincar işlemi gerçekleşir aksi takdirde gerçekleşmez.)
3- .obj dosyası oluşur. (Memory organizasyonu ve işlemci uyumlu komutları barındırır.)

int a = 10; RAM'de gerçekleşir.
int b = 20; RAM'de gerçekleşir.
int c = a+b;
int a = c+b;
a, b ve c değerleri bizim anlamamız için vardır işlemci gözünden bakacak olursa bunlar adres anlamındadır.
Mikroişlemciye ilk önce c = a+b; komutu gitti.
Mikroişlemci ile memory veyahut RAM (emin değilim) arasında 4 adet yol vardır:
veri yolu, veri adresi, komut yolu ve komut adresi.

4- Komut getirme adımları. (Fetch – Decode – Execute)

NOT: Veri yolu ve komut yolu sayesinde aynı anda fetch yapılır.

NOT: Pipeline execute'da dezavantajlı durumdadır bunun sebebi ise komutun karmaşık olması yani dolayısıyla uzun sürmesidir.

3- Zamanlayıcılar (Timers) (Kaydedici)

Zamanlayıcılar, gerekli zaman veya sayma işlemlerini yaparlar.

Her darbede (clock) zamanlayıcının kaydedici biti 1 artar. Her zamanlayıcının (kaydedicinin) bir hassasiyeti bulunmaktadır. 8-bit veya 16-bit.

Neden zamanlayıcı kullanılır? İşlemcinin gerçek bir zamanda çalışması için.

1- Led yak	2- Bekleme süresi 3000ms
3- Led söndür	4- Bekleme süresi 3000ms

Bu durumda mikrodenetleyici 3 saniye aralıklarla ledi yakar ve söndürür, fakat 3 saniye bekleme süresinde mikrodenetleyici başka bir işlem yapamaz.

PIC 16F877A'da bulunan zamanlayıcılar:
Timer 0: 8-bit (overflow ve karşılaştırma) (Görevi, kendisine gelen sinyal darbelerini saymaktır.)
Timer 1: 16-bit (overflow)
Timer 2: 8-bit (karşılaştırma)

Timer 0 nasıl kullanılır:

1- Timer 0 ve tüm genel kesmeler aktif yapılmalı.
2- Mikroişlemci hızı belirlemek (RC osilatörü, clock palsi)
3- Frekans bölücü (Prescaler) belirlemek

PRESCALER: Sayımın veya sürenin daha uzun sürmesini sağlamak için zaman kaynağının frekansını belirli bir sayıya böler. İşlemci hızını (frekansını) belli parçalara bölerek zamanlayıcı sayım için daha fazla zaman alır. Normalde Timer0 zamanlayıcısı 255'e kadar bir sayım gerçekleştirebilir**. Prescaler ile bu sayımın süresini artırabiliriz.** Örneğin Prescaler 4 olduğunda yani her 4 clokta sayım bir artar.

Programda Timer 0 programcı tarafından belirlenen zamana vardığında Timer 0 kesmesi aktif (yani 1) olur ve geçici bir kesme bayrağı kaldırılır, dolayısıyla ISR fonksiyonu içinde bu bayrak biti kontrol edilmelidir.

Zamanlayıcı ile süre veya zaman üretimi: **config**Timer0 **= 255 – (T * Frekans) / (4 * Prescaler)**
Örnek: Zamanlayıcı (Timer 0) ile 10ms süre üret.
config**Timer0** = 255 – (10*10^-3 ***** 4*10^6) / (4 ***** 64) =98,75 ≅ 99
Yani zamanlayıcı ile 10 ms elde etmek için zamanlayıcının kaydedicisine 99 değeri vermemiz gerekmektedir. Kaydedici 99'dan 255'e kadar sayar yani 99, 100, 101 …255. Kaydedicinin değeri her 255 değerine geldiğinde tekrar zaman kaydedicisini 99 değeri ile başlatmamız gerekmektedir. 99'dan 255'e geldiğinde yazmış olduğumuz fonksiyonlar gerçekleşir.

1- **255'den çıkarmamızın sebebi Timer 0'ın 8 bit olmasıdır. (8 bit'in max değeri 255'dir. Yani $2^8 - 1 = 255$)**
2- **T bizden istenilen zamandır. (Burada 10 milisaniye istenilmiş.)**
3- **Frekans mikrodenetleyiciyi hangi frekansla çalıştırdığımızdır. (MicroC'de 8mHz olarak seçiyorduk ya bu işte o ayar.)**
4- **Prescalar değerini seçerken sonucumuzun 0-255 arasında olmasına dikkat etmeliyiz neden 0-255 çünkü Timer 0 8 bittir. 2, 4, 8, 16, 32 değerlerini yazdığımızda sonucumuz 0-255 arasında gelmedi ama 64 yazdığımızda 0-255 arasında bir değer geldi o yüzden 64'ü seçtik. Aşağıdaki, TMR0 Rate tablosundaki değerlrei yerleştirdik. (2,4,8,16,32,64,128,256)**
5- **Son olarak kaydedici değerimiz pozitif ve tam sayı olmalıdır. (Ondalıklı değer çıkarsa o sayıyı yuvarlarız.)**

255'den sonraki değere geçmek istediğinde yani overflow olduğunda, overflow biti yani bayrak biti (TMR0IF=1;) 1 olur.
Zamanlayıcıların çalışması için GIE ve PEIE 1 olmalıdır.

000 1:2 ---> 1 tane clock 2 tane clock'a dönüşmüş oldu.
001 1:4 ---> 1 tane clock 4 tane clock'a dönüşmüş oldu.

PSA biti -> 1'se WDT'a
-> 0'sa TMR0'ın tablosuna bakmalıyız.

Bit Value	TMR0 Rate	WDT Rate
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

********** NOT **********

- Zamanlayıcılar (mikroişlemci içinde bir parçadır) = sayaç demektir.
- Sayaç belli bir değere ulaştığında fonksiyonu tetikler (yazılımsal olarak değil, donanımsal olarak). Donanımsal olduğu için kaydediciler (flip-floplar aracılığıyla) kullanılır.

Neden farklı zamanlayıcılar kullanılır?
1- Bit sayıları farklıdır.
2- Donanımsal olarak zamanlayıcılar ya karşılaştırma ya da overflow ile tetiklenir.

PORTD = 255; D portunun bütün pinleri 1'dir (1111 1111)
PORTD = 1; D portunun 1. pini 1'dir (0000 0001)

Prescaler: Görevi clockları birleştirmektir. Clocklar birleşirse zaman aralığı artar. Prescaler 1 olursa WDT'a, 0 olursa zamanlayıcı ya da sayaç için kullanmış oluruz.

Zamanlayıcılarda karşılaştırma ve taşıma olayları vardır.
Karşılaştırma: 2 adet kaydedicimiz var. İlk kaydedicimiz 0'dan max değere kadar sayar. Diğer kaydedici ise istenen süreye göre bir değer ile beslenir.
A Kaydedicisi -> Sayaç 0 1 2 3 4 … 100 IF == 1 B Kaydedicisi -> Sabit bir değer 100
Taşıma: Bir adet kaydedici var bu kaydedici bir değer ile beslenir ve 0'dan saymaya başlar, beslenen değere vardığında IF == 1 olur.

INTF -> Dışardan tetikleme vardır.	TMR0IF -> RB0	RBIF -> 4, 5, 6 ve 7. pinler

```
ınterreup(){
        if (TMR0IF == 1){
                gelen_deger = 99;
                TMR0IF = 0;     // TMR0: Timer 0'ın saydığı değerleri hafızasında tutan registerdir.
        }
}
```
Overflow olduğunda başlangıç değeriyle tekrar beslemeliyiz (burdaki başlangıç değerimiz 99) ve TMR0IF'ı 0 yapmalıyız.

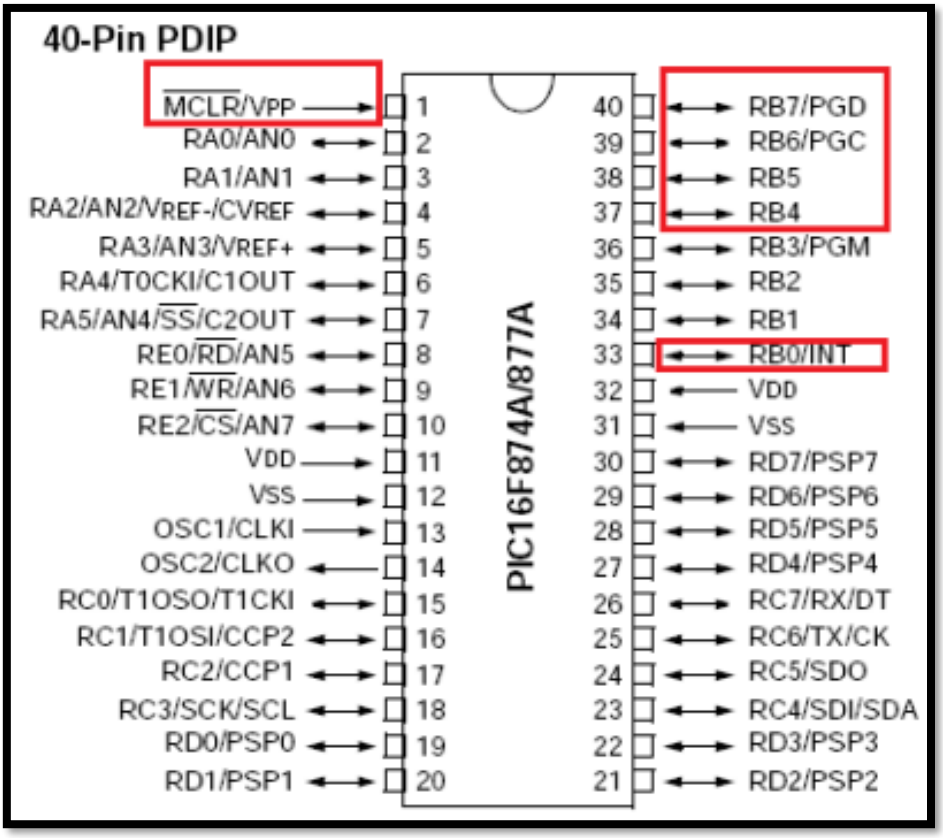
4- Kesme (Interrupt)

İşlemci başka bir görevi yerine getirmek için mevcut görevi bıraktığı bir durumdur.

Nasıl Çalışır?

İşlemci herhangi bir görevi gerçekleştirdiğinde, Interrupt Service Routine (ISR) kesme işlemi geldiğinde işlemci yürüttüğü görevi durdurur ve gelen kesme programına hizmet etmeye gider ve sonra durdurduğu görevi tamamlamak için geri döner.

Kesme Türleri 1- Harici Kesmeler (Dışarıdan gelen uyartılardır.) a. MCLR pini b. INT, RB pinleri 2- Dahili Kesmeler a. Timer 0 ve Timer 1 Zamanlayıcılar	Birden fazla kesme kullandığımız zaman mikroişlemci nasıl ayırt eder? Her kesmenin bayrak biti vardır, bayrak bitleri sayesinde kontrol edilir.

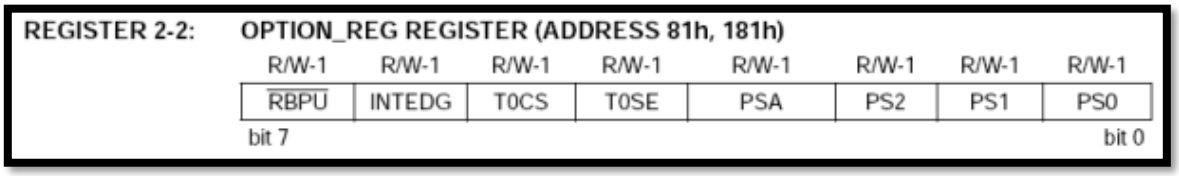


B Portunun 0. Pininde (RB0) Kesme Olayı (**Kendi başına bayrak biti vardır**)

MicroC'de B portunun 0. pininde (RB0) kesme ayarları:

Prescaler değerinin 2'lik tabandaki karşılığını PS2, PS1 ve PS0 bitlerine yazdık. Yukarıdaki örnek için konuşacak olursak prescaler değerimiz 64'dü 2'lik tabandaki karşılığı ise 101 (10 ms üret örneği); OPTION_REG.PS2 = 1; OPTION_REG.PS1 = 0; OPTION_REG.PS0 = 1;

1- option_reg kaydedicinin intedg bitinin 0 (düşen kenar) veya 1 (yükselen kenar) olarak ayarlanması.



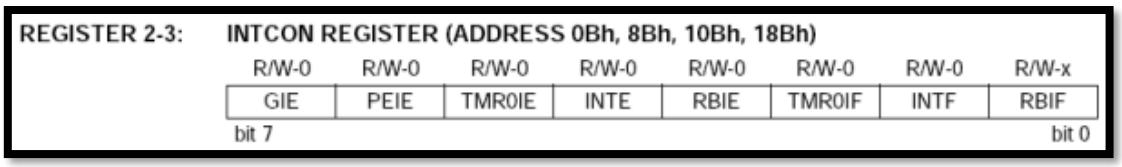
REGISTER 2-2: OPTION_REG REGISTER (ADDRESS 81h, 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Bit 6 INTEDG: Interrupt (Kesme) Kenar Seçim Biti
1 = RB0/INT pininin yükselen kenarında kesmesi **(yani butona basıldığı anda)**
0 = RB0/INT pininin düşen kenarında kesmesi **(yani butondan elimizi çektiğimiz anda)**

Kesme işleminin gerçekleştiği kenarı (pini) seçtikten sonra. Bir kesme olduğunda, kesme bayrağı 1'dir. Bu nedenle, kesme görevi tamamlandıktan sonra kesme bayrağının kontrol edilmesi gerekmektedir, yani kesme bayrağını sıfırlamak gerekmektedir. (**option_reg.intedg = 0; //1**)

1- intcon **(bütün kesmeleri açıp kapatmaya ve birtakım ayarlamalar yapmaya yarayan registerdır)** kaydedicinin 4. bitini, 6. bitini ve 7. bitini ayarlamak gerekmektedir.



REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

Bit 4 INTE: RB0/INT Harici Kesme Etkinleştirme Biti; INT0 harici kesme pini etkinleştir.
1 = RB0/INT harici kesmesini etkinleştirir. (intcon.inte=1;)
0 = RB0/INT harici kesmesini devre dışı bırakır. (intcon.inte=0;)

Bit 6 PEIE: Maskesiz Çevresel Kesme Etkinleştirme Biti
1 = Maskelenmemiş tüm çevresel kesmeleri etkinleştirir. (intcon.PEIE=1;)
0 = Tüm çevresel kesmeleri devre dışı bırakır.

Bit 7 GIE: Genel kesmeleri etkinleştirir.
1 = Maskelenmemiş tüm kesintileri etkinleştirir. (intcon.gie=1;)
0 = Tüm kesmeleri devre dışı bırakır.

2- Kesmelere hizmet eden program veya fonksiyon yazılmalı bu fonksiyonun void interrupt() adında ve türünde olmalı ve bu fonksiyon işlemci tarafından sürekli olarak intcon kaydedicisinin 1. bitini (kesme bayrak biti) takip etmektedir.

Bit 1 INTF: RB0/INT Harici Kesme İşaret Biti
1 = RB0/INT harici kesmesi oluştu (yazılımda temizlenmelidir)
0 = RB0/INT harici kesmesi gerçekleşmedi

```
void interrupt(){
   if(intcon.intf==1){   //if INT0 interrupt occured
      portd.b0 = ~portd.b0; //toggle pin D0
      //Komutlar
      intcon.intf=0;     //reset INT0 flag mutlka sıfırlamalıyız } }
```

B Portunun 4-5-6-7. Pinlerinde Kesme Olayı (Aralarından birinin bayrak biti tarafından kontrol edilebilir)

Bu kesme PORTB'nin 4. 5. 6. ve 7. bitlerinde herhangi bir değişiklik olup olmadığını kontrol eden ve bu pinlerde herhangi bir durum değişikliği var ise bayrağını kaldıran bir kesmedir.

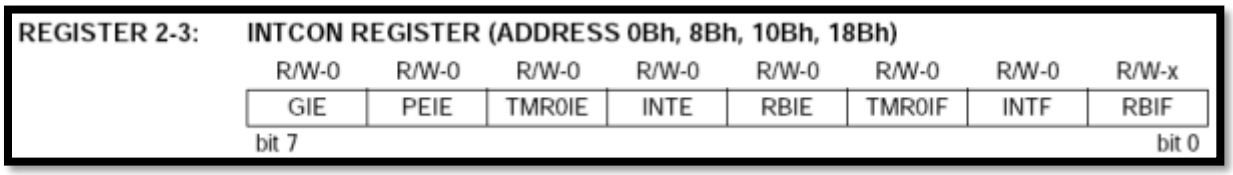
Bu kesme PORTB'nin RB4. RB5. RB6. RB7. bacaklarının tamamının giriş (yani 1) yapılması halinde geçerlidir. Bacaklardan birisi çıkış yapılır ise kesme iptal olur.

RB0 kesmesinden farkı, RB0 kesmesinde bizim istediğimiz durum gerçekleşirse (0'dan 1'e veya 1'den 0'a değişme) kesme oluşuyordu. Burada ise ilgili pinlerden herhangi birisi bir önceki durumunu değiştirdiği an kesme meydana gelmektedir. Yani diyelim ki PORTB.RB7 ilk durumda 0 değerinde, PIC çalışmaya başladıktan sonra eğer PORTB.RB7 ilk durumda 0'dan 1'e geçerse kesme oluşur. Daha sonra PORTB.RB7 tekrar 1'den 0'a geçerse bir kesme daha oluşur. Bu RB4. RB5. RB6. RB7 pinleri için geçerlidir.

1- B portunun 4-5-6-7. pinlerinin giriş olarak ayarlanması gerekir.

```
TRISB.B4 = 1;
TRISB.B5 = 1;
TRISB.B6 = 1;
TRISB.B7 = 1;
```

2- **intcon** kaydedicinin 3. bitini, 6. bitini ve 7. bitini ayarlamak gerekmektedir.



REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

Bit 3 RBIE: RB Port Değişikliği Kesme Etkinleştirme Biti
1 = RB portu değişikliği kesmesini etkinleştirir.
(**intcon.rbie=1;** // B4-B7 portlarının pin değiştirme kesmelerini etkinleştirir.)
0 = RB portu değişikliği kesmesini devre dışı bırakır.

Bit 6 PEIE: Maskesiz Çevresel Kesinti Etkinleştirme Biti
1 = Maskelenmemiş tüm çevresel kesmeleri etkinleştirir. (**intcon.PEIE=1;**)
0 = Tüm çevresel kesmeleri devre dışı bırakır.

Bit 7 GIE: Genel kesmeleri etkinleştirir. (Global Intertupt Enable)
1 = Maskelenmemiş tüm kesintileri etkinleştirir. (**intcon.gie=1;**)
0 = Tüm kesmeleri devre dışı bırakır.

3- Kesmelere hizmet eden program veya fonksiyon yazılmalı bu fonksiyonun void interrupt() adında ve türünde olmalı ve bu fonksiyon işlemci tarafından sürekli olarak intcon kaydedicisinin 0. bitini (Bit 0 RBIF: RB Port Change Interrupt Flag bit) takip etmektedir.

1 = RB7:RB4 pinlerinden en az birinin durumu değişti; bir uyumsuzluk koşulu biti ayarlamaya devam edecektir. PORTB'yi okumak, uyumsuzluk durumunu sona erdirecek ve bitin temizlenmesine izin verecektir (yazılımda temizlenmelidir).

0 = RB7:RB4 pinlerinin hiçbiri durum değiştirmedi.

NOT: B portunun 4. 5. 6. ve 7. bitlerindeki kesmelerin bayrak bitleri ortaktır, yani bu pinler üzerinde değişim sadece Change Interrupt Flag biti tarafından kontrol edilmektedir.
NOT: B portunun 4. 5. 6. ve 7. bitlerinde 0 (düşen kenar) veya 1 (yükselen kenar) ayarları bulunmaktadır.
NOT: B portunun 4. 5. 6. ve 7. pinlerindeki kesme bayrak biti ortak olduğundan dolayı bu pinlerin herhangi biri konum değiştirdiğinde kesme fonksiyonu gerçekleşir. Dolayısıyla bu pinleri ayrı ayrı kontrol etmemiz gerekmektedir.
NOT: INTCON.TMR0IF == 0; koda öncelikle bunu yazmalıyız.

5- PWM (Pulse Width Modulation – Sinyal&Darbe Genişlik Modülasyonu)

PWM = Analog Çıkış

Üretilecek olan kare dalganın genişliğini kontrol ederek, çıkışta üretilmek istenen analog elektriksel değerin elde edilmesi tekniğidir.

PWM tekniğinin mantığı mikrodenetleyicinin pininden dışarıya çıkan 5V değerini sürekli olarak değilde kesik kesik çıkışa aktarmaktır.

Örneğin; mikrodenetleyicimizin bir pininden 1 saniye boyunca 5V, 1 saniye boyunca 0V alınmakta ve bu böyle devam etmektedir. Bu sinyalin ortalama voltajı 2,5V olur. Frekansı ise $\mathbf{f} = \frac{\mathbf{1}}{\mathbf{T}}$ formülünden (1/2 saniye) 0,5Hz olarak bulunur. Bu pinden 0,5 saniye boyunca 5V, 1,5 saniye boyunca 0V geliyorsa frekans yine 0,5Hz olmakla birlikte çıkış geriliminin ortalaması 1,25V olur. Yani frekans sabit tutularak sinyalin 5V değerinde kalma süresinin değiştirilmesi, böylelikle çıkıştaki ortalama voltajın ayarlanmasına **darbe genişlik modülasyonu** denilmektedir. Sinyalin 5V seviyesinde kaldığı cycle'a **duty cycle** denir. Her zaman %0 - %100 arası bir değer alır. Örneğin; 2 saniye boyunca 5V'da kalması.

CCP1 ve CCP2 pinlerinde uygulanır, 16F877A'da PORTC1 ile PORTC2 pinleridir.

PWM'in kullanım alanları: Ana kart fanının hız kontrolünde, farklı tipte çalışan motorların hızını ve açısını ayarlamak, LED'lerin parlaklık şiddetini ayarlamak için kullanılır.

********** NOT **********

CCP1 ve CCP2 pinlerinde uygulanır.

Mikrodenetleyicinin hangi özelliği kullanılarak PWM yapılır?

- Counter ya da karşılaştırma özelliğini kullanarak.

Bu olay için T2CON, T2CKPS1 ve CCP1CON registerleri kullanılır.

Duty cycle hiçbir zaman %100 olmaz.

Frekans = $\frac{\mathbf{1}}{\mathbf{t\ (zaman)}}$ Duty Cycle = $\frac{\mathbf{t_{on}}}{\mathbf{t_{on}+ t_{off}}}$. %100

Periyot (süre)

CCP1M3 gibi ayarları yapmaya gerek yok.

Periyot Register = ((1/Frekans) / (1/Fosc.4.Prescaler)) – 1

Fosc = Mikrodenetleyicinin Frekansı Prescaler = Mikrodenetleyicilerde prescaler ayarı yoktur bu yüzden prescaler yerine 1 yazabiliriz.

Periyot Register 8 bittir elde edilen değer 0-255 arasında olması lazım.

Frekans üretiminde PWM kullanımı

Period Register = ((1 / Frequired) / (1 / Fosc * 4 * prescaler)) - 1

- Sonuç 255'ten büyük olmamalı çünkü PR2 kaydedici 8 bitlik
- Sonuç virgülü sayı olmamalı
- Sonuç pozitif bir sayı olmalı
- İstenen frekansın üretilmesi için PR2 kaydedicisinin değeri kaç olmalı.

Frequired = istenen frekans, Fosc = frekans osilatör, Prescaler = Timer2

Fosc değeri PIC'in hızı yani 8 MHz mi 4MHz mi ona göre yazıyoruz.

[*] Final_Oncesi_Quiz.c

```
sbit LCD_RS at RB0_bit;
sbit LCD_EN at RB1_bit;
sbit LCD_D7 at RB5_bit;
sbit LCD_D6 at RB4_bit;
sbit LCD_D5 at RB3_bit;
sbit LCD_D4 at RB2_bit;

sbit LCD_RS_Direction at TRISB0_bit;
sbit LCD_EN_Direction at TRISB1_bit;
sbit LCD_D7_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB4_bit;
sbit LCD_D5_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB2_bit;

char txt[8];
int okunan;

void main() {
     Lcd_Init();
     ADC_Init();        //Initilaze AD module with default settings
     PWM1_Init(5000);   // 5000 milisanyiye = 5 saniye
     PWM1_Start();
     TRISA = 0;
     TRISB = 0;
     TRISC = 0;
     ADCON1 = 0b00001110;
     PR2 = 124;         // Periot Register
                        // PWM formülündeki Prescaler yazan yere 1,4 veya 16 yazarak işlemi denemeliyiz.
                        // Ne zaman pozitif ve 255'den küçük çıkarsa sonucumuz o sayıyı Prescaler yerine kullanmalıyız.
     T2CKPS1_bit = 0;   // Bunları her zaman yaz. (Prescaler ayarı)
     T2CKPS0_bit = 1;   // Bunları her zaman yaz. (Prescaler ayarı)
     while(1){
          okunan = ADC_READ(0);               // hangi kanalı kullandıysak onu yazarız ADC pdf sinde yazıyor.
                                              // read analog value from channel 2
          IntToStr(okunan, txt);
          Lcd_Out(1, 1, txt);

          if(okunan <= 0){
               PWM1_Set_Duty(0);
          }
          else if (okunan > 0 && okunan <= 75){
               PWM1_Set_Duty(63);
          }
          else if (okunan >= 75 && okunan <= 150){
               PWM1_Set_Duty(128);
          }
          else if (okunan >= 150 && okunan <= 225){
               PWM1_Set_Duty(192);
          }
          else if (okunan >= 255 && okunan <= 308){
               PWM1_Set_Duty(255);
          }
     }
}
```

Duty Cycle 8 bittir. Eğer hoca 4'e bölün derse yukardaki örnekte olduğu gibi 255'i 4'e böleriz.

PWM1_Set_Duty(x); ---> Buradaki x değerleri 4'e böldüğümüzde elde ettiğimiz değerlerdir. %0, %25, %50, %75, %100

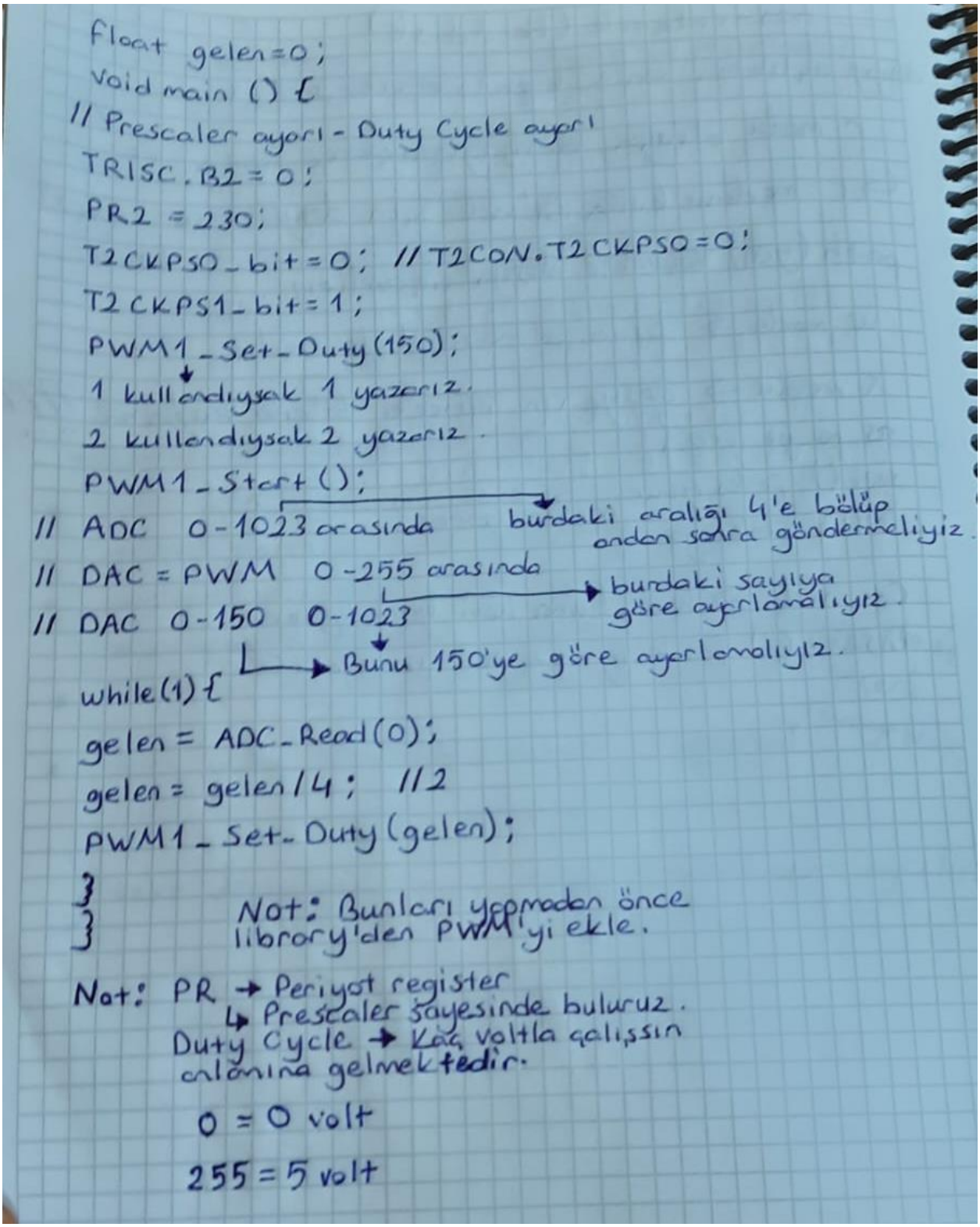
ADC_Read(0); ---> PIC'de 0. Porta bağlanndığı için 0 yazdık.

ADC 10 bit, PWM 8 bittir.

PWM1_Init(X); --> X değeri, frekans değeridir.

PWM_Set_Duty(X); ---> X değeri 0-255 arasında bir değer olabilir. 0 iken voltajın %0'ını alırız, 127 iken voltajın %50'sini, 255 iken voltajın %100'ünü alırız.

PWM1_Start(); --> PWM çıkışını veriyor.



6- WDT (Watch Dog Timer – Bekçi Köpeği Zamanlayıcısı)

Mikrodenetleyi içinde ayrı bir donanım zamanlayıcıdır. Bu zamanlayıcının kullanım amacı diğer timerlardan (Timer0, Timer1 ve Timer3) biraz farklıdır. Adı üstünde programın bekçiliğini yapmaktadır. Yani program içinde yapılacak işlemlerde aksama durumu, programın kilitlenmesi veya program kontrolden çıktığı durumlarda WDT zamanlayıcısı devreye girmektedir. WDT programı resetler (yeniden başlatır) ama status registerinde bu durumu belirtir. Komutlar yeniden başlatılır.

// WDT aktif ise WDT sayacı saymaya başlar ve içerisinde tanımlanan değere ulaştığı anda işlemciye reset atar.

PIC'i sleep (uyku) modundan çıkartmak için de kullanılabilir, Fakat burada komutlar kaldığı yerden devam eder.

WDT biriminin kendine ait RC osilatörü vardır. Çünkü sistemin clock palsi kesilse bile WDT çalışması gerekir.

WDT nasıl çalışır ?

1- WDT özeliği aktif olmalı
2- WDT için Prescaler aktif olmalı
3- WDT için uygun Prescaler seçilmeli

WDT zaman aşım normal süresi 18ms dir. Prescalar değer ile 128 *18ms =2304ms yani yaklaşık 2.3 saniyeye kadar uzatılabilir.

TimeOut = 18ms * Prescaler

PS2, PS1, PS0	Prescaler Rate	Timout
0,0,0	1:1	18mS
0,0,1	1:2	36mS
0,1,0	1:4	72mS
0,1,1	1:8	144mS
1,0,0	1:16	288mS
1,0,1	1:32	576mS
1,1,0	1:64	1.1Seconds
1,1,1	1:128	2.3Seconds

REGISTER 5-1: OPTION_REG REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**

bit 6 **INTEDG**

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
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 : 120	1 : 64
111	1 : 256	1 : 128

********** NOT **********

WDT, enerji tasarrufu sağlar.

WDR -> PS2, PS1 ve PS0 kaydedicileriyle ayarlanır.

PSA biti -> 1'se WDT'a

-> 0'sa TMR0'ın tablosuna bakmalıyız.

WDT'ın çalışması için ayarlar (edit) kısmından enable yapmalıyız.

asm clrwdt; -> WDT'ı temizler, sıfırlar.

asm sleep; -> Mikrodenetleyici uyku moduna geçer.

Prescaler ayarına göre sleep modunun süresini ayarlayabiliriz.

Kesmeler, delay fonksiyonundan etkilenmez sebebi ise clockları ve hafıza yerleri ayrıdır.